Lorem Ipsum Documentation

Release

Lorem Ipsum Team

May 02, 2017

Assessment 4

Hello we are team Lorem Ipsum and we are developing a murder mystery game.

This Page will be developed further as time goes on, but for now you can access the first assessment pieces on this site, or visit our homepage.

For Assessment 3, we are working with the project produced by Team Farce. For Assessment 4, we are working with the project produced by Team JAAPAN.

CHAPTER 1

Team Members

- Brooke Hatton @brookke
- Ben Jarvis @beno11224
- Benjamin Grahamslaw @Penguin-Benjamin
- Jason Mashinchi @jasonmash
- Joseph Shufflebotham @joeShuff
- Vishal Soomaney @TheProgrammingDuck

CHAPTER 2

Contents

Requirements

Introduction

At our first meeting we read the brief, and with that in mind, we played cluedo to get a feel for a general detective game. This sparked discussion as to what works in a game, and it gave us ideas for how our game should work. We decided the most effective way of thinking about the game was like "guess who". Using the brief and our new ideas, we produced a list of features we'd like to include in the game, and then prioritised those.

After the initial meeting, we produced a number of user scenarios [1], and used our list of features to help produce a draft of the requirements. From these we identified ambiguous points and produced questions we needed to get answers for. We met multiple times with the customer throughout the design process, to present our requirements and ask questions we had. Using their feedback, we made any necessary changes.

Following the response we got after the Assessment 1 feedback was released, we decided it was best to rewrite our requirements. This lead to substantial improvements in the clarity and categorisation of our requirements, and this has benefitted us as it made the requirements easier to test against when it came to implementing the game.

When designing the requirements, we took these points into account:

The requirements should be categorised as:

- Functional requirements - these define what the system should do
- Nonfunctional requirements - these define the behaviour of the system
- The requirements should be achievable within the time allocated of this project
- The requirements should consider the hardware in which the game should run
- The requirements should meet all points included in the brief
- We produced a survey asking about input methods and accessibility [2]. We got a sample of our target market to respond. From this we found:
- The preferred way of interaction with the game was keyboard and mouse.

- There were no results from colourblind people, however we still felt it was important that our game was accessible, so have included accessibility features as a "could" requirement. Colour blindness is a condition apparent in 1 in 12 men and 1 in 200 women [3] so we feel that it is a requirement that some people would benefit from.

The requirements are laid out in tables based on the IEEE standard for system requirements, as we felt this was a good standard to adhere to. The tables are split according to type (functional or non-functional), and category. Some requirements have associated risks, these are referenced in the table below, and are defined in the risks document.

Each requirement is given a unique identifier to make it easy to locate, and for traceability. The identifiers are made up of three numbers, using the following system:

- The first number is category, this represents the functional area of the requirement
- The second number represents how important the requirement is:
- 1 = Must implement
- 2 = Should implement
- 3 = Could implement
- The third number is the position in the list, used to ensure the identifier is unique.

Functional Requirements

Game

ID	Requirement	Success Criteria	Alternative	Risk ID
1.1.1	To start the game there must be a main menu	The game has a working clear menu	The game starts immediately when its opened	21-24
1.1.2	The game must be fully controlled by a mouse and keyboard	The game can be interacted with using only a mouse and keyboard	The game is controlled by an Xbox controller.	21-24
1.1.3	Game must have different 'plotlines'	Each gameplay is different in some way	Game only has 2 plot lines.	21-24
1.2.1	There should be a way of suspending or pausing the game	There is the option to pause the game which opens a pause menu	The game cannot be paused	21-24
1.3.1	Could be controlled by a gamepad	The game can be interacted with using a gamepad	Game is only controllable by keyboard and mouse.	21-24
1.3.2	Could have a sound track	Music will be played when the game is running	There is no sound track.	21-24
1.3.3	If game has a soundtrack, it must have an option to turn the sound track off	There is a player accessible way to turn the sound track off within the game	The soundtrack would always be on.	21-24

Player

ID	Requirement	Success Criteria	Alternative	Risk ID
2.1.1	The player must have a personality that is customisable	The players personality changes in the game	The player personality does not change.	21-24
2.1.2	Must not be able to accuse a NPC unless enough evidence has been found	The player cannot see the option to accuse an NPC unless they have interacted with enough clues	Can accuse an NPC without evidence.	21-24
2.1.3	The player must start in a central room at the start of every game	When the game starts, the player should be in the centre of the "Ron Cooke Hub"	Player can start in any room.	21-24
2.1.4	The player must be able to navigate between rooms on the map	The player can move throughout the map and transition to other rooms when desired	The player progresses through rooms automatically.	21-24
2.1.5	The game must be turn-based where two players swap between who is playing.	The game switches who is playing every few interactions with the game.	Necessary requirement.	21-24
2.2.1	The player should be able to see their current personality level	The game should present the player with an GUI element showing their personality level	The Player will not be able to see their current personality level	21-24

NPC

ID	Requirement	Success Criteria	Alternative	Risk ID
3.1.1	Each killer must have a motive	The killers all have motives	All killer have the same motive.	21-24
3.1.2	There must be 10 NPCs(non playable characters).	The player should be able to locate 9 NPCs as well as there being 1 victim.	There are less than 10 NPC's.	21-24
3.1.3	The NPCs must all exhibit differing personalities.	The NPCs interact with the player in differing ways.	The NPCs will all act in the same way to the player.	21-24

Map

ID	Requirement	Success Criteria	Alternative	Risk ID
4.1.1	The game must have a map containing 10 separate rooms	The player should be able to visit 10 different rooms in the game	The game has less than 10 rooms.	21-24
4.1.2	All rooms must be of varying sizes.	The player should be able to notice all the rooms being of different sizes and shapes.	There are several rooms of equal size.	21-24
4.1.4	The map must have a 'secret' room	The map contains a room that is not accessible at the start of the game.	Necessary Requirement.	21-24
4.1.5	The secret room is not accessible until a puzzle is solved	After solving a puzzle, the secret room becomes accessible.	Necessary Requirement	21-24

Clue

ID	Requirement	Success Criteria	Alternative	Risk ID
5.1.1	There must be at least one clue in each room of the map	The player can navigate to every room and be able to locate a clue	Some rooms may have no clues, some may have multiple	21-24
5.1.2	The player must be able to interact with a clue	The player should be able to interact with a clue once it has been located	The player gets the clue without interaction.	21-24
5.2.1	There should be a journal wher e clues are placed by a player for future reference	The player can see a journal in the GUI that allows visibility of collected clues	Clues are stored internally but the player will not be able to see them	21-24

Score

ID	Requirement	Success Criteria	Alternative	Risk ID
6.1.1	There must be a score shown to players in the game	The player must see a score displayed in the GUI	There will be no scoring.	21-24
6.2.1	There could be an online scoreboard to keep high scores	There could be a scoreboard in the GUI that presents the all time high scores	There will be a local list of high scores, or no scoring	21-24

Dialogue

ID	Requirement	Success Criteria	Alternative	Risk ID
7.1.1	The player must be able to interact with an NPC	A player can go up to an NPC and talk to them	The player cannot interact with NPC's.	21-24
7.1.2	The player must have the option of questioning an NPC	When a player talks to an NPC, they should have the option to question them	The player cannot question an NPC.	21-24
7.1.3	The player must have the option of ignoring an NPC	When a player talks to an NPC, they should have the option to ignore them	The player cannot ignore an NPC.	21-24
7.1.4	The player must have the option of accusing an NPC	When a player talks to an NPC, they should only have the option to accuse them if they have found enough clues to accuse the NPC	The player cannot accuse an NPC.	21-24
7.1.5	The player must choose from a set of questions when interacting with an NPC that reflects different personalities	When a player talks to an NPC, and chooses to question them, they can choose from multiple speeches with different personality levels. Eg. Aggressive	The player only has one	21-24
7.1.6	Each NPC must respond differently to questions from a Player depending on both NPC's and Player's personality and characteristic s	When an NPC responds to a player after being questioned, their response must be determined by their characteristic s and the player's personality	All NPC's respond in the same way.	21-24
7.1.7	The player must be shown introductory and closing dialogue.	Before the player can play they are shown an introduction and once they have completed the game the player is given a 'goodbye speech'.	The player will not be given any context dialogue.	21-24

Win/Lose Conditions

ID	Requirement	Success Criteria	Alternative	Risk ID
8.1.1	The game must be 'won' when the player successfully accuses the murderer	If the player accuses the murderer then the game is won.	This is a necessary requirement.	21-24
8.1.2	The game must be 'lost' when the player accuses too many NPCs	If the player accuses too many NPCs then the game is lost.	The game will not be able to be 'lost'	21-24

Nonfunctional Requirements

Game

ID	Requirement	Success Criteria	Alternative	Risk ID
1.1.4	Must run on the university computers	An executable is provided the runs on the computers	The game will not run on university computers.	7
1.1.5	Must run on Windows 10	An executable is provided that runs on windows 10	The game will not run on windows 10.	7
1.2.2	Should run on MacOS	An executable is provided that runs on MacOS	There will not be an executable that runs on MacOS	7

NPC

ID	Requirement	Success Criteria	Alternative	Risk ID
3.1.4	Each NPC must have a personality that affects and is affected by game play.	The NPC will respond best to different types of question. For example, an aggressive NPC will respond best when questioned nicely.	All NPC's have the same personality.	21-24
3.1.5	The killer and victim must be randomly selected each time the game begins from two sub-lists of killers and victims.	When the game starts, the victim and the killer has been selected at random.	The killer and victim is the same every time.	21-24
3.1.6	Each NPC must be randomly assigned to a room at the start of the game	All NPCs should be situated within a different room at the start of the game.	Each NPC is always in the same room.	21-24

Map

ID	Requirement	Success Criteria	Alternative	Risk ID
4.1.3	The room where the murder occurred must be randomly selected at the start of every game	One random room should be the selected murder location at the start of every game	The murder room is always the same.	21-24

Clues

ID	Requirement	Success Criteria	Alternative	Risk ID
5.1.4	The murder weapon clue must be found before the player can accuse any NPCs	The player cannot accuse an NPC until they've located the murder weapon clue	Can accuse without the murder weapon.	21-24
5.1.5	Most clues must help with identifying the killer	A clue will narrow down the number of suspects left to be the killer	All clues help identify the killer	21-24
5.1.6	At the start of the game, clues must be randomly assigned to each room in the map	There must be at least one clue in every room of the map at the start of the game	Clues always in same location.	21-24
5.1.7	The motive clue must be found before the player can accuse any NPCs	The player cannot accuse an NPC until they've located the motive clue	Can accuse without the motive clue	21-24
5.2.2	Clues could be picked up by a player and placed in a journal	The player can interact with a clue and place it in their journal for future reference	Clues will be stored internally, but my not be seen by the player	21-24

Score

ID	Requirement	Success Criteria	Alternative	Risk ID
6.1.2	The player's score must take into account the time taken	The score must change depending on how long the game has lasted	There will be no scoring.	21-24
6.1.3	The player's score must take into account the number of wrong accusations	The score must change depending on how many accusations the player has made	There will be no scoring.	21-24
6.1.4	The player's score must take into account the number of questions asked	The score must change depending on how many questions the player has asked	There will be no scoring.	21-24
6.1.5	The player's score must take into account the number of clues found	The score must change depending on how many clues have been found by the player	There will be no scoring.	21-24

Dialogue

ID	Requirement	Success Criteria	Alternative	Risk ID
7.1.8	The type of question asked to an NPC by a player must determine the player's personality	When a player chooses a speech to say to an NPC, their personality level is affected by their choice	The type of question asked affects nothing.	21-24
7.1.9	If an NPC is accused and isn't the killer then the NPC must refuse to interact for the rest of the game	When a player interacts with a previously accused NPC they shouldn't get a response	The NPC does not mind being falsely accused.	21-24
7.1.10	If an NPC is ignored, the Player cannot interact with the NPC again until a change in the situation occurs.	The Player cannot question, accuse or ignore an ignored NPC again until another clue is found, the Player moves to a different room or the Player talks with a different character.	The Player can question, accuse or ignore an ignored NPC without any changes to the situation.	21-24

Bibliography

[1] Appendix A [online] http://docs4.lihq.me/en/latest/Assessment4/appendixA.html [Created 21/11/16]

[2] Appendix C [online] http://docs4.lihq.me/en/latest/Assessment4/appendixA.html [Created 21/11/16]

[3] Colour Blind awareness [online] http://www.colourblindawareness.org/colour-blindness/, [Accessed 3/11/16]

Risk Assessment and Mitigation

Introduction

Risk management is an important part of any project, we must prepare for what could happen during the course of the project in order to be able to quickly recover and stay on track. The risks which are shown below take into account the scale of the project and aim to cover only risks which are realistic within this context.

To determine risks we brainstormed various scenarios - such as a teammate being ill for more than a few weeks. From these scenarios, we collected possible risks, and worked out the likelihood of them occurring. To determine the risk we discussed how it would impact the project, focussing on how many knock-on effects the issue would cause.

The risks are presented in a tabular format, with the following columns:

- **Risk ID** - this allows for traceability across the project
- **Description** - describes what the risk is for
- **Likelihood** - each risk has an estimated likelihood on a scale
  - **High** =good chance this risk will occur, about 75% chance
  - **Medium** =equal chance of risk occurring or not, about 50% chance
  - **Low** =not likely to occur, about 25% chance, however some risks may be lower
- **Impact** - this describes the impact the risk would have to progress in the project
- **Severity** - shows the severity of the impact on the project on a scale
  - **High** =a major setback which could affect the whole project

  - **Medium** =could add up to a week of extra work and may threaten a deadline
  - **Low** =may mean a few extra hours of work, but nothing major
- **Mitigation** - describes how how we will aim to avoid such a risk and deal with it
- **Owner** - describes the owner of the problem, where the owner is the person most likely to be responsible for the issue.

The overall table is split into sections which group together similar risk such as software risks. Each section is then ordered by severity, highest first. Equal severity is ordered by likelihood.

This table will be regularly consulted in an attempt to monitor the risks and try to ensure they do not occur and catch them early if they are occurring.

Due to the size of the team we feel that these risks are appropriate and accurate.

Table of risks

Software risks

ID	Description	Like-li-hood	Impact	Sever-ity	Mitigation	Owner
1	Our game may be slow or unresponsi ve.	Medium	No one will want to play the game.	High	Improve efficiency of the game wherever possible and regularly check performance	Coding Team
2	Software library doesn't work or lacks a feature e.g. has a bug that stops the game from working, or is missing a feature required for the game to work.	Medium	We would struggle to implement the feature we want to add. We would also use up lots of time trying to solve the issue.	Medium	Test the elements of the library you plan to use beforehand . Also, make sure the library has an active community surrounding it and that bugs are fixed quickly. If it was early stages we could get a new library but this would require us to rewrite our code to work with the new library.	Soft-ware Library Owner
3	Code is hard to understand and seems too complex.	Low	Could cause bugs and makes bug fixing harder. Slows down the productivi ty of the group.	Medium	Use meaningful variable names and plenty of comments, both in code and in commit messages. Make sure code is reviewed by the majority of members before it gets merged into the repository .	Coding Team
4	Conflicts in git. Different members changing the same code.	High	May need to move code around and even rewrite.	Low	Make sure people work on separate elements by assigning them to different tasks and if not then make use of Gits tools.	Coding Team
5	Our own software doesn't work as intended.	High	Will need to bug fix. Loss of time and potentiall y productivi ty if that function or feature is the bottleneck of the game.	Low	This is a normal part of software developmen t. We all make mistakes. However, before code is approved by the group we will use unit testing that will test key functions of the game as we develop them meaning that should a function break we will know about it before it's merged.	Coding Team/Design Team/Project Man-ager

Hardware risks

ID	Description	Like-li-hood	Impact	Sever-ity	Mitigation	Owner
6	Personal computer breaks long term or is lost.	Low	Could lose work and be unable to work.	Low	Ensure work is saved online to google drive cloud service and that code is stored on github. Department PC's should be accessible most days and have all the tools we need.	Fi-nal User
7	Personal computer crashes while working.	Medium	Potentiall y will have lose work, meaning you lose time doing it again.	Low	Save regularly, google docs[2] will do this for us. Regularly commit code to personal branches so that it stored elsewhere other than your PC .	Fi-nal User

Risks with people

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
8	A team member leaves the module or even the course.	Low	They may have only access to their work, also the rest of the team will have more to do.	High	As above store online but also try to keep each other motivated to avoid this.	Project Team
9	A team member is ill/away for a week or two.	High	They might have been skilled in a certain area that no other member can do well.If they have the only access to work may get behind from it.	Medium	Hard to avoid, but we should store work online where everyone can access. If we work in pairs to complete tasks then there will be less of a chance of having one person who knows the most about one area.	Project Team
10	Arguments within the team.	Medium	Disrupts the work of the team and prevents us moving forwards. Also, unpleasant for the team as a whole.	Medium	Try to avoid conflict but if necessary have proper debates perhaps using a mediator, do not keep issues hidden.	Project Manager
11	Lack of communicat ion.	Medium	Tasks may be done twice or not done at all.	Medium	Keep strong communicat ion using the tools we plan to use.	Project Manager
12	A team member does not do their work.	Medium	Could disrupt other members work and could make the other team members annoyed.	Low	Don’t give members too much work or work they cannot do, ensure that the team communicat es well and regularly meets up to discuss how the work is going.	Project Team/Manag er

Risks with tools

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
13	Google drive servers stop working.	Low	Could lose/lose access to work that is stored there.	Medium	Store work locally , and on other services.	Google
14	Central git repository [1] is lost in some way.	Low	Temporaril y lose access to it.	Low	Keep up to date local copies so can be easily restored. We could host our own local copy should github go down.	Git/Coding Team
15	Website hosting fails.	Low	Users lose access to the website.	Medium	The website files are stored on github and every team member has a local copy of the repository on their computer so we could bring the site back up on a different server. The site is also protected by cloud-flar e[3] who will provide a cached version of the site if our host were to go down.	Website Hosting Owner

Requirements risks

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
16	Not including a requiremen t which is required by the customer.	Low	We let the customer down and have failed them.	High	Make sure key requiremen ts are elicited from the customer so they get what they want.	Requiremen ts Team
17	A requiremen t could change/ be added.	High	May need to rewrite code or add extra code to account for it. Extra time will be needed.	Medium	Our software architectu re must be flexible and able to be changed easily.	Requiremen ts Team
18	Stating a requiremen t that we cannot actually achieve.	High	Let down the customer and also waste time.	Medium	Be sensible when deciding requiremen ts, be sure you can achieve them.	Requiremen ts Team/Codin g Team
19	Ambiguity in requireme nts.	Medium	May end up making something which is not what was originally intended.	Medium	Ensure requiremen ts are clear and check any ambiguitie s with the customer.	Requiremen ts Team
20	Choosing requiremen ts that the customer doesn't really want.	Medium	Waste time which could be spent on requiremen ts they did want.	Low	Ensure you know which requiremen ts the customer really wants and which can be ignored.	Requiremen ts Team

Estimation risks

ID	Description	Like-li-hood	Impact	Sever-ity	Mitigation	Owner
21	Expect the team or a team member can do more than they actually can.	Medium	Work is not done or is done to an insufficie nt standard.	Medium	Give tasks that people can do and if they can't then help them. When working on difficult tasks work in pairs to complete the task meaning individual team members don't feel as overwhelme d by the task	Project Man-ager
22	We may underestim ate how long it will take to do some work.	Medium	Work ends up taking longer than expected or not done to the standard it could be done. This could cause other areas of the project to suffer	Medium	Set realistic timings to do work and be realistic on how long a task will take. Account for unforeseen delays in our plan adding time where we can catch up.	Project Man-ager
23	Be too pessimisti c about what we can achieve.	Medium	We end up with a product which is not as good as it could have possibly been.	Low	Push our limits but also stay realistic and within the requiremen ts. If we have extra time then we can use it to enhance the product.	Project Man-ager
24	Distribute tasks incorrectl y.	Low	Team over/under worked.	Low	Distribute tasks appropriat ely and tell others if feel over/under worked.	Project Man-ager

Bibliography

[1] GitHub [online] Available https://github.com [Accessed 01/11/2016]

[2] Google Drive [online] Available https://www.google.com/drive/ [Accessed 01/11/2016]

[3] Cloud Flare [online] Available https://www.cloudflare.com/ [Accessed 01/11/2016]

Methods and Planning

Software Engineering Methods & Tools

Development Process

We are using an agile-scrum [0] approach for our software engineering project, as it allows our team to carry out work in a flexible manner, while providing the flexibility to evaluate our progress and plan for the next phase of work as we go along. Details of how we use this methodology to run our meetings and plan tasks are found in the team organisation section.

One of the key tools needed for any software engineering project is a version control system. This is a tool that will allow us to keep track of code changes, and collaborate on the same piece of code. We're using GitHub[1] for our code repository, as it's a popular tool that our team members are familiar with.

While developing, we plan to use git's branch and pull request functionality to help us manage multiple developers on the same project. To ensure our code quality is high, we ensure that a different developer performs a code review for every pull request, and that appropriate unit tests have been added and successfully pass before approving the branch to be merged.

We are using test driven development for our acceptance tests, as these are being written before development starts to ensure our code meets the requirements. While development is occurring, unit tests will be added to our code using JUnit[2]. This is so that we maintain a high level of test coverage, and have confidence that our code is working as intended. It will help with ensuring regressions do not occur. We are using CircleCI[3], a continuous integration server, to run all the unit tests every time a commit is made to GitHub, this will ensure that nobody accidentally breaks the codebase. For more details, please see the testing report.

To manage our development tasks, we plan to use the GitHub Projects feature of our code repository. This lets us manage our tasks and code issues with a Kanban board[4], and allows for items to be assigned to team members so everybody knows who's doing what. GitHub also allows us to link issues to the project, and these are useful because team members can collaborate on design decisions and keep track of what's happening in the code. More about our team organisation is discussed later.

Development Tooling

To develop the project, we decided, after much deliberation, to use Java[5] and IntelliJ IDEA[6] as our programming language and IDE. This is because Java is a language accessible to all members of the team, as well as the entirety of our cohort. We did give serious consideration to other programming languages such as C#[7] with Unity[8] however few in our team have any experience in the language which means it may take longer and cause problems later on in the process.

We needed a library that provides useful game APIs to help speed up our development process. We did some research and found 4 options of libraries we could use: Swing[9], Mini2Dx[10], LibGDX[11], LWJGL[12]. We quickly realised LWJGL and Swing weren't right for our project. In the end we decided to use the LibGDX Java game development framework. We chose LibGDX over Mini2DX because LibGDX has support for the IDEs that we intend to use, and it provides features that simplify several game development steps.

Design Processes

We will need to do some designing whilst the development stages are progressing. The main tool we will be using to design the game assets is Piskel[13], an online tool that allows us to easily create game assets. We expect designs to start as paper prototypes that will be developed further as the project goes on.

As we'd have to produce a map for our game (see requirement 4.1.1), we were pleased to discover that LibGDX has support for tiled maps. There is useful software called Tiled [14] that allows us to create a high quality 2D map and export it for use with the framework. This will save us massive amounts of time compared to alternative solutions, and this can be used to improve gameplay and ensure that there are few bugs.

Collaboration Tools

In addition to using GitHub projects as mentioned above, we also need some communication and collaboration tools to help us keep the project on track.

For the first few weeks we were using Facebook Messenger[15] for team communication, but it became apparent that it wasn't up to the job as we kept losing important pieces of information. We decided to switch to Slack [16] as it allows us to have separate chat channels for different aspects of the project, letting us keep all necessary information separated. Since separate discussions are split into separate channels, we are able to find important information more easily for future reference. We use this to consistently report our progress, ask questions on ambiguous issues and organise future in-person meetings.

We are using integration between our development tools and Slack to help us keep track of progress in one clear feed, in particular the GitHub integration and the CircleCI integration. This helps with our development method, as it ensures all team members are kept up to date with the status of the repository and test results.

For online meetings we are using Join.me [18] Join.me is a free tool with voice, text and video chat as well as a screen sharing feature which allows a presenter to share their screens with other team members. Join.me allows efficient and effective team meetings to take place even whilst all members are in separate locations. We made extensive use of Join.me over Christmas holiday period, where we used it to hold a full team meeting, using our scrum method as described later.

For document storage and collaboration, we are using Google Drive[19] with a shared team folder. This contains all of the documents and other files we've been working on. We are making use of Google Docs, Sheets & Forms[19] as they allow us to collaboratively work on documents at the same time, as well as providing mechanisms that allow us to review and comments on our documents.

Assessment 2 Review

At the beginning of Assessment 3, we reviewed our performance on the previous assessment, and in general we found we were happy with our performance. We were generally happy with our choices of development tooling and our software engineering methodology, we especially found our code reviews really helped with code quality and for catching bugs.

However, we identified that we struggled to keep our project management tools up to date, which meant that it was unclear who was working on what, and how much progress had been made. We then had a debate as to whether to start using Asana[17] instead of Slack and GitHub Projects, as Asana is able to perform the same tasks as the other tools do. This would allow us to minimise the number of tools we need to keep up to date, however we decided that the cost of switching outweighed the benefits, as the effort of transitioning the team to a new tool was considerable. As such, we have continued to use Slack and GitHub Projects for daily communication, and we have all being making an effort to keep the systems up to date so we can avoid the problems we faced last time. We have made sure that all tasks can be seen on GitHub Projects and issues exist for all tasks being worked on at any given time, whenever progress is made on a task or queries exist about it, comments are made on the issue relating to the task.

As in previous assessments, we continued using the SCRUM methodology and had 2 in-person meetings every week. We held some debates as to whether virtual meetings would be more appropriate due to most of us working on our own bits of implementation, keeping Github updated meant we were all kept up to date with the progress of each task so meetings weren't as necessary in some cases. Despite this we determined that weekly meetings were indeed required to have team reviews of all work done, this way any issues can be caught early. Meetings were also essential for the sake of deciding the future direction of the project and so that all members had the same vision of what we wanted the game to look like at the end.

Assessment 3 Methodology

Once we determined what project to take on board for this assessment, we held a meeting to discuss what we wanted the project to look like at the end of the assessment. This involved analysing the inherited code and determining our first steps to further develop the project.

As we analysed the code, we quickly realised that although it was well commented and the game worked well, the code itself wasn't very well organised. Classes and methods were often named incorrectly or in a confusing manner, files were not placed in a coherent and thought through file structure, and in many cases there were several classes worth of code jammed into a single class. We determined that our first priority was to refactor the code to make it more intelligible and whilst doing this, also to understand how each method works so that we know how to use it when developing future features.

In our first meeting, we analysed the brief and identified the requirements that had yet to implemented. We then split these requirements into work that needed to be done, and created individual tasks and issues to cover this work on GitHub. We then distributed these evenly amongst ourselves, we also had several unassigned tasks that could be picked up by any team member when they had completed their own work. We made a conscious effort in this assessment to keep GitHub as up to date as possible, and to carry out discussion in the comments of the appropriate issue - we found being stricter about our process helped with communication, and helped prevent problems we had run into previously.

We ensured all documents were updated as we went along to ensure everything was written about whilst fresh in our minds, this allowed the documents to be as accurate as possible. It also allowed us to have a better idea of how much work we had left to do since we knew exactly how much time documentation was taking instead of having to estimate how long it would take once we got to it.

Assessment 4 Methodology

The fourth assessment also began with choosing a project. As our original game had been chosen by a number of other teams we had the nice opportunity to take back an updated version of the original game. We even had a few different options to choose from. After our experience working in Assessment 3 with new code we knew better what we were looking for when choosing for Assessment 4. We had discovered some issues with the code we chose for Assessment 3 when it was too late to change. For this reason we opted to be more thorough with our analysis and decision making. We looked at all the options as a team. This led to much debate within the team as our recent experience of software engineering allowed us to see benefits to choosing multiple projects.

In the end we opted for a simple project which was closely based on our original which we knew well. Although this lacked some features of another project such as a nice speech UI and CircleCI it was eventually chosen largely due to its familiarity and our awareness that Assessment 4 was likely to be about extension and not so much extras that were already there. Once this choice was made we started to discuss how we would go about the project.

Team Organisation

Roles

We decided early on to have a team leader role. The team leader is responsible for ensuring everybody has a task to be working on, and for making sure progress is being made. They are also responsible for producing meeting plans, and answering queries of any team member.

In Assessment 1, we voted to decide a group leader - Brooke Hatton won as we felt he was a natural leader. For Assessment 2, we decided that Jason Mashinchi would take over as group leader, as he had experience in a software development team. We plan to alternate group leaders throughout the assessments, to give the other member a break.

Every member of the team is assigned tasks to do at the end of every meeting for the sprint ahead. We decide who does what based on who wants to take on the task, or based on previous experience if it's relevant to a task.

Meetings

The team aims to have a meeting at least once a week, during which our team leader acts as our scrum master. We may not strictly follow the agile-scrum rules, but we've found that our approach works well for us.

Our team leader prepares for every meeting by producing a brief plan of what needs to happen during the meeting to ensure that we are making progress. Having a plan means that we can stay on track within the meeting and don't go off track and don't make progress. We tend to keep meetings high-level so that we don't waste time on implementation details that can be decided without the entire group present.

Meetings are scheduled to happen multiple times a week depending on availability of team members. Our meetings always signify the start of a new sprint. We typically start with each member going through what they've been working on in the previous sprint, and they raise any problems or questions they may have. This allows us to catch issues, concerns or blockers that have arisen early on in the meeting, so we can take them into account when planning the next sprint.

We then proceed to discuss what needs to be done during the next sprint, and assign tasks to each individual in the team. This is great because it means everybody has something to be working on for the next week, and ensures that we are making sufficient progress in the project.

During Sprints

Everybody is assigned a task to be working on during sprints, and they are responsible for ensuring their bit of work gets done. Every team member uses the kanban board containing issues on GitHub to keep the rest of the team up to date on where their task is at, and they make sure that their appropriate issues/tasks are assigned to themselves.

If any problems arise during the week, team members use their assigned GitHub issues to discuss anything related to their task, or we communicate through Slack when we need to discuss something more general. Splitting up the communication methods in this manner allows us to find discussion when necessary, as often we make design decisions within GitHub that can be referenced later.

Systematic Plan

Plan for Assessment 2

The focus of the team will fully switch to the second assessment on Wednesday 9th November, once the first assessment has been submitted. We plan on completing the second assessment by Tuesday Spring Week 2 giving us a week to account for any unexpected developments or fixing issues that happen to arise. It also gives us time to analyse, criticise and improve our own work thus improving the quality of our code and enhancing the functionality and efficiency. This will also ensure that we have very high quality documentation, once all group members are happy with the readability of our code we plan on requesting that students from other teams look at sections of our code to test readability. There will also be a 2 week rest from SEPR to account for time spent studying for exams and also the exam week itself.

Plan for Assessment 3

After the completion of Assessment 2 on Tuesday 24th January we will work as a team to decide the project to take on for Assessment 3. This will be done within one week, after which we shall decide upon areas that need to be worked upon over the next assessment, and delegate tasks accordingly to team members. Documentation is started in the first week with everyone focusing on making sure reports and code documentation are up to date. within the next week all documents will be near to completion and the code will have started to be updated. Again, we plan on completing the work a week before the deadline to give us time to fix any issues found.

Plan for Assessment 4

Once Assessment 3 is completed, we will decide on a new project to pick up as a team. Ideally, we will be able to choose one of the three updated iterations of the original game we submitted to assessment 2.

Once we begin assessment 4, we will meet to analyse the new requirements that have been assigned for this assessment and determine how this will affect the architecture of the game. Upon deciding what features need to be updated, the work will be divided up into tasks and these tasks will be assigned equally to all members.

After we have made a good start on the code, half of the team will start working on documentation so that we have all work done on time. We aim to keep our methods the same, since as a group we feel that our current methods work effectively. This means weekly scrum meetings will continue to be held, as well as continued use of GitHub projects. As can be seen in the Gantt chart (Appendix B), we aim to finish the assessment with a week to spare, so that we are prepared for any unexpected issues that may arise.

Bibliography

[0] Waterfall to Agile: Flipping the Switch - Bhushan Gupta [Online] Available: http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf [Accessed 25/10/2016]

[1] Github [Online] www.github.com [Accessed 8/11/2016]

[2] JUnit [Online] http://junit.org [Accessed 22/01/2017]

[3] CircleCI[Online] https://circleci.com/ [Accessed 22/01/2017]

[4] Kanban board [Online] www.github.com [Accessed 8/11/2016]

[5] Java [Online] www.oracle.com [Accessed 8/11/2016]

[6] IntelliJ [Online] www.jetbrains.com [Accessed 8/11/2016]

[7] C# [Online] www.msdn.microsoft.com [Accessed 8/11/2016]

[8] Unity [Online] www.unity.com [Accessed 8/11/2016]

[9] Swing [Online] www.oracle.com [Accessed 8/11/2016]

[10] Mini2DX [Online] www.mini2dx.org [Accessed 8/11/2016]

[11] libGDX [Online] www.libgdx.badlogicgames.com [Accessed 8/11/2016]

[12] LWJGL [Online] www.lwjgl.org [Accessed 8/11/2016]

[13] Piskel [Online] www.piskelapp.com [Acessed 8/11/2016]

[14] Tiled [Online] www.mapeditor.org [Accessed 8/11/2016]

[15] Facebook Messenger [Online] https://en-gb.messenger.com/ [Accessed 23/01/2017]

[16] Slack [Online] www.slack.com [Accessed 8/11/2016]

[17] Asana [Online] https://asana.com/ [Accessed 01/02/2017]

[18] Join.me [Online] www.join.me [Accessed 8/11/2016]

[19] Google Drive [Online] https://www.google.com/drive/ [Accessed 23/01/2017]

[20] Gantt Chart [Online] http://lihq.me/Downloads/Assessment3/AppendixB.pdf

Appendix: Task Assignment Summary

Assessment 1:

- Ben Jarvis was assigned the requirements document
- Benjamin Grahamslaw was assigned the risks and mitigations report
- Brooke Hatton and Jason Mashinchi were assigned the Architecture report
- Joseph Shufflebotham and Vishal Soomaney were assigned the Methods report

Assessment 2:

- Benjamin Grahamslaw was assigned the GUI report & user manual
- Brooke Hatton, Jason Mashinchi & Joe Shufflebotham were responsible for the implementation of the game, testing and the design choices made.
- Other team members contributed towards implementing smaller features in the game
- Jason Mashinchi was assigned the testing report
- Brooke Hatton & Vishal Soomaney were assigned the architecture report
- Ben Jarvis & Benjamin Grahamslaw were assigned the document updates

- Joe Shufflebotham & Vishal Soomaney were assigned the implementation report

Assessment 3:

For Assessment 3, we didn't assign documents to each individual in the same way we did in previous assessments. As well as contributing towards the code, we found it was more effective if we all contributed to documents when we found something that needed to be changed. The list below was created using the issues we had completed and assigned to ourselves on GitHub.

Benjamin Grahamslaw

- Updated user manual
- Updated GUI report
- Wrote introduction and some sections of change report
- Implemented scoring with Ben J
- Created character dialogue with Ben J
- Contributed towards implementation report

Ben Jarvis

- Reviewed and updated risks
- Reviewed and updated requirements
- Updated acceptance and unit test documents
- Implemented scoring with Ben G
- Created character dialogue with Ben G

Brooke Hatton

- Performed an extensive review of the architecture and planned refactoring
- Refactored maps & rooms
- Refactored dialogue system
- Improved database structure
- Added test coverage
- Contributed towards implementation report

Jason Mashinchi

- Updated testing report
- Added StatusBar
- Added personality meter
- Refactored JournalScreen
- Refactored InterviewScreen
- Updated methods report, updated Gantt chart
- Contributed towards implementation report
- Updated website and Read the docs

Joe Shufflebotham

- Implemented player movement in game
- Added NarratorScreen
- Implemented clues, motive clues and means clues
- Added Suspects to the map
- Contributed towards implementation report

Vishal Soomaney

- Implemented MainMenuScreen
- Updated acceptance tests
- Updated requirements
- Updated methods report
- Contributed towards implementation report

Assessment 4:

Benjamin Grahamslaw - Helped implement the puzzle screen

Jason Mashinchi - Created evaluation and testing report - Worked on the implementation of the multiplayer system - Was meeting organiser - Contributed towards project review and implementation report

Brooke Hatton - Created the architecture report - Implemented the puzzle system - Helped create the secret room - Contributed towards implementation report

Joseph Shufflebotham - Worked on implementation of the multiplayer system - Contributed towards implementation report - Implemented the new sprites

Benjamin Jarvis - Updated the requirements document - Worked on implementation of the multiplayer system - Contributed towards implementation report

Vishal Soomaney - Created the project review document - Implemented the secret room and extra point system - Updated the acceptance test and methods documents - Contributed towards implementation report

Implementation Report

Change list & justification

Add new class 'PlayerSwitchScreen'

- Our new requirement [2.1.5] states that the game must be turn based. This means that the players will have to switch between who is playing the game, as the game will only be running locally. During this time, and to prompt the players to switch over, this new screen is displayed.
- It includes a title saying: "Time to switch players!" and a button to switch to the next player.
- It includes a leaderboard that shows the top scoring players, so players can see who is winning.
- This class extends 'AbstractScreen'

Add new class 'NumberOfPlayersSelectionScreen'

- The new requirement [2.1.5] states that the game must be turn based. To complete this, there needs to be two or more players.

- This screen allows the user to set how many people will be playing. It includes a slider from 2 to the maximum amount of players, which can be set by changing a constant. It also contains 2 buttons, one to start the game and one to return to the main menu.
- The initial requirement was for 2 players to be able to play the game, however the way we had implemented the code was dynamic enough to allow us to have as many players as we'd like.
- We set an arbitrary limit of 10 players and allowed the user to choose the number of players as this was minimal effort. We set the maximum to 10 as we felt that to be a suitable number based on the type of game. We felt this method was much better than just 2 player turn-based as the game would be much more competitive.
- A new button 'multi-player' was added to the main menu to navigate the player to this screen.
- It extends 'AbstractScreen'

Add new class 'ScenarioBuilder'

- The new requirement [2.1.5] states we need to add turn based multiplayer. This means the game scenario will need to be the same across all players, however one player's actions shouldn't affect another player's game.
- This class generates the game state, including the clues, NPCs, and murderer/motives.
- This class is used to build the games for each player that is playing. It generates instances of the 'GameSnapshot', explained below.

Add new class 'GameSnapshot'

- GameSnapshot is used to hold the instance information for each game. It was created to allow us to complete requirement [2.1.5].
- The snapshot stores information that will need to differ for each player. This includes Clues, NPCs, location etc, as these have player dependent properties.
- As each game needs to be the same (same killer/clues), the GameSnapshot instances start as duplicates - with the same values, but in different memory locations.

GUI changes

- Unlike the project we picked up in assessment 3, this one had the required dependencies for the use of TrueType Fonts. This allowed us to have far more interesting fonts in the game, especially for our titles.
- We also implemented new skins for our buttons, labels, sliders and checkboxes to give the game a more refined look.
- No architectural changes were required for this as we simply edited the Assets and UIHelpers classes that already existed.

Music changes

- We disabled the music [1.3.3] by default, this means that the player has to enter the "Settings" menu and manually untick the "Muted" checkbox to enable the music.

Secret Room Implementation

- The new requirements [4.1.4] require the game to include a 'secret room'
- The new room was made with Tiled, using the same tileset as existing rooms.
- The transition to the room takes place when the puzzle is solved.
- Logic was added to check for the rendering of the secret room. When this was detected, an image overlay was added on top of the room to create a 'torch in a dark room' effect.
- This image was created by us and was fully black aside from a transparent circle in the center. As the game is centered around the player, this allowed the player to only see the few tiles around their character thus giving the room a more mysterious feel, we determined that this was in line with the theme of the game.

- Logic was added to the random NPC distribution process to ensure that no NPC spawned in the secret room, as we wanted the room to be an extra feature and not something that was essential to the completion of the game.
- No significant architectural changes were required, as no methods or classes were required for the implementation of the secret room.

Puzzle implementation

- One of the new requirements [4.1.5] we were given in this assessment was the implementation of a puzzle that had to be solved before being able to enter a secret room. We decided on a puzzle in which the player must click three books in a book case in the correct order to unlock the room. The idea took inspiration from the classic false books often seen in films.
- Each player gets to solve the puzzle, however the puzzle differs per player.
- For this we implemented a "Puzzle" class which handles the logic for the puzzle. The bookshelf tiles were given a boolean "secretRoom" property, and one of the bookshelves in the game are randomly chosen from the possible values when the game is generated.
- When the player clicks on the correct bookshelf in the game, a transition screen is generated. Following which, a puzzle screen is generated containing 9 buttons. 3 of the buttons are randomly assigned as the correct ones and if all 3 are clicked in a row then the secret room is revealed, if any of the erroneous buttons are clicked then the puzzle resets itself.

Extra Score

- To make the secret room [4.1.4] more interesting, we placed an item in it that provided extra points to the player that found it. When the player interacts with the extra score item, they obtain a random score between 300 and 700 points.
- Although the extra score functionality wasn't requested by the client, we felt that its addition was necessary, as it provided a purpose for players to visit the secret room.
- The implementation of this involved adding a property called "Extra Score" to the item tile to provides extra points. In this case the item was a tile that looked like a cash pile.
- Methods were added to the 'Room' class to search through the tileset of the room each time a room was initialised. These searched for the "Extra Score" property and added the locations of any tiles with said property to a List.
- Logic was then added to check whether an item had this property every time an interaction was made, if it did then the extra score would be added to the player's score and some dialogue would appear.
- Further logic was then added to disable the item after this interaction took place so that no further score could be obtained by any player. This disabling logic meant that in a multiplayer scenario, only the first player to find the cash pile would get the extra points.

Change Report

Change management

The first thing we did as a team was to have a good look at the code and deliverables we had inherited from Team Farce. We wanted to ensure we were familiar with the documentation and code, so we could make informed decisions on what changes we needed to make.

We were aware that we would need to mainly do two types of change:

- Corrective changes: We discovered some faults and errors within the codebase which we tried to fix.

- Additive changes: To complete Assessment 3 we needed to add a number of features in order to complete the game to the requirements specification.

We also needed to be aware of two other forms of change we may need:

- Deletive changes: We discovered some features of the other teams reports and code that did not work or were buggy. This especially caused problems when trying to use their methods and classes to implement new features.
- Perfective changes: We found that the games architecture was, in some places, badly organised and corrected this with extensive refactoring in order to make working with the code easier for ourselves and potentially others. This work is described in the implementation report, along with other code changes.

As a team we evaluated the advantages and disadvantages of the code and documents, then we used what we had learned to help decide which documents to work with and what changes needed to be made. We decided to keep all of our documents, and adapt them to suit the new project as we felt this was achievable as we were already very familiar with our documentation. This choice of documentation also made our lives easier when it came to traceability.

We also looked at the code and decided which of the change types mentioned above were required for each part. We analysed our requirements and looked for any changes that needed to be made, including adding in requirements related to the features that were already implemented in the game that we inherited. We moved on to implementation, where some parts of the code were kept the same, some parts were changed slightly, and some parts of the code were completely changed.

After updating our requirements, we began development on the changes and additions that needed to be made. We tried to ensure traceability of requirements both forwards and backwards. It is important to ensure it is clear where new requirements came from, but also to be clear where these new requirements are used in the other documents, where appropriate.

Our overall focus for change management was to ensure we used the best of what we had to use. If what we were given was good we kept it, otherwise we changed or replaced it. We wanted to make the very best game that we could that would fit the requirements, and we were not afraid of making big changes in order to achieve this goal. In our documents new changes were shown using a blue font to make it obvious.

Report updates

GUI report

This document was modified with the following changes:

- Added descriptions for new GUI elements implemented in the game or inherited from Team Farce. We looked at which elements of the GUI we had brought over from our original GUI and also look at the new elements that have been added, such as the journal.
- The report now describes new screens which were added this assessment, the clue found screen and the narrator screen, these are important elements of the finished game so needed to be included in the GUI report.
- Updated descriptions of GUI elements that we brought over from our game
- Expanded content about how user interaction works with the GUI. We felt that our current GUI report did not effectively communicate this, so it was something that we tried to incorporate within the new GUI report. User interaction is an important factor in GUI design that we should not have missed first time round.
- Added a description of our thought process and the factors we considered when designing the GUI

URL:

- http://docs3.lihq.me/en/latest/Assessment3/gui.html or
- http://lihq.me/Downloads/Assessment3/GUI3.pdf

Testing report

We continued to use the testing document our team created for Assessment 2. This document was modified with the following changes:

- Update references from Assessment 2 to 3
- Update test statistics (number of tests, success rate, number of failures)
- Updated links for Assessment 3
- Added paragraph (2nd from top) explaining how tests work in our project (originally from Team Farce), and how we've adapted and improved their testing methodologies to better fit our workflow. This includes setting up CircleCI for continuous integration. This paragraph also mentions that we kept our own acceptance tests as we also kept our requirements, as well as mentioning that we used and extended upon the unit tests provided with the project we took on.
- Unit test report (see Appendix E) has been updated to reflect the new project and any changes made to the tests. The test results have been added. All unit tests (both added and inherited from Team Farce) have been linked to our requirements, for traceability.
- Acceptance test report (see Appendix D) has been updated to reflect the new project choice, and the results have been updated to reflect upon the latest version of the game. We felt that the acceptance tests weren't specific enough so we updated them to make them more specific, this ensures that the tests can be perfectly replicated. We also tried to remove ambiguous terms such as the word "interact" when referring to picking up clues or speaking to suspects.
- Added reference sources for testing summary, these are placed in the new bibliography
- Added section about automated test coverage, as we started using JaCoCo

URL:

- http://docs3.lihq.me/en/latest/Assessment3/testing.html or
- http://lihq.me/Downloads/Assessment3/Test3.pdf

Methods and plans

This document was modified with the following changes:

- Added a description of a discussion we had about team management tools. We considered switching to Asana which is a tool similar to GitHub projects to show tasks that need to be done and have discussions, however we felt that it was best to carry on with GitHub projects as it is where our code is and that should ensure we all look at it and keep it up to date.
- Added a description of a debate held about in-person meeting frequency and necessity.
- Described the approach we intend to have when selecting a project for Assessment 4, we will hopefully select one of the three teams that chose our game in Assessment 3
- Added our current thoughts on what we will need to do when we are given new requirements and how we will tackle them, regardless of what they are
- Describes potential task distribution and continuing methods
- Added a detailed description of our methods used in this assessment, including dealing with the new code and the assignment of tasks. We felt it was important that our methods document described the methods used in this assessment, as it gives a good insight into how we worked as a team to tackle this assessment.
- Added a Gantt chart for assessment four to the document to show how we will split up our time for the next assessment

- Updated the appendix which includes the task assignment summary. Each team member was given tasks for Assessment 3

URL:

- http://docs3.lihq.me/en/latest/Assessment3/methods.html or
- http://lihq.me/Downloads/Assessment3/Plan3.pdf

URL for updated plan: http://lihq.me/Downloads/Assessment3/AppendixB.pdf

Risk assessment and mitigation

The risk management document did not change very much, as we were happy with the report from last assessment. Previously our approach and presentation was heavily reworked as described in the previous assessment to include risk ownership.

We looked at the risk management document given to us by the other team but did not really find anything we wanted to add to our own document.

As a team we were very happy with the risk management document that we had and it's format so we decided to keep it. However, we modified this document with the following changes:

- Added database risks to the risk table. When we took on this new project we inherited a new kind of software that we had not previously been using. The other team heavily used a large database, and the use of this new software presented new risks that needed to be looked at. Therefore we discussed these risks as a team and added them into our own document to make the risks specification complete with our new game.
- One database risk is to do with incorrect data being saved into the database, this could lead to unexpected scenarios when running the game.
- Another risk is that data is not loaded into our game at all, which could cause the game to not run and would be a major problem with how the game is implemented in this assessment.
- Details and mitigation for both of these risks can be found in the updated document.

URL:

- http://docs3.lihq.me/en/latest/Assessment3/riskAssessmentAndMitigation.html or
- http://lihq.me/Downloads/Assessment3/Risk3.pdf

GUI Report

Design processes

Our design process for the game was based on some simple principles, we wanted to make sure that all GUI elements in the game were clear, intuitive and easy to use. As a team we were clear that we wanted the GUI design to be centered around the user and making sure that a user would find it enjoyable to use, this in turn would hopefully lead to an enjoyable experience playing the game which is another high priority for us as a team. This is a game so it should be fun. We also aimed to have a game that has a high level of usability. One example would be the click to move option which uses an A* search algorithm and makes movement very usable.

Player interactions

We designed the user interaction in the game using the principles designed above. We were focused on usability, so we took the decision to use mouse control for all user interactions. This was because we felt the mouse is the most common and familiar method of interaction with a computer and will be obvious to the user. It also removes the need for a mental transition between keyboard control and mouse control, as jumping between the two can be jarring, as we found in the last assessment.

At first we thought it was best to interact with clues and NPC's by walking up and pressing the Enter key, however we realised that this was not intuitive, not to mention walking across rooms was boring and also could have an unfair effect on the player's score. For this reason we switched to just clicking on the clue or suspect instead, as this option addresses these issues.

Main menu

The main menu will be the first screen that a player will see upon loading up the game. It will provide the user with the following options:

- **New game button** - this starts the game
- **Exit button** - this closes the game
- **Settings button** - go to settings screen
- **Multiplayer** - go to number of players select screen

The buttons are all designed in the same way throughout the game. They are designed to be easy to distinguish from the background and for all text displayed on them to be easy to read.

Related requirements: 1.1.1 **Realisation**: Appendix F:1

Main navigation screen

The main navigation screen contains two elements - a map and a status bar.

While the player is playing the game they will be able to see their character on the map and move around from room to room. The character is displayed in the middle of the map for good playability. This screen also has a status bar overlay at the bottom, which allows the player to switch between the map and Inventory screen, and also shows the player's current score and personality. You can also go to the pause menu. Also shows which player turn it is in multiplayer.

Related requirements: 2.1.4,2.2.1,3,4(Map,clues and BPC sections) **Realisation**: Appendix F:2

Dialogue screen

The dialogue screen allows communication with suspects/NPCs. The GUI is an overlay over the main navigation screen and displays the text in a box which is being spoken by the NPC and player.

Related requirements: See dialogue sections of requirements. **Realisation**: Appendix F:3

Inventory

The Inventory is a collection of information that the player has obtained so far in the game. It contains a clues list which shows the player the clues that they have collected so far

Related Requirements:5.2.1, 5.2.2 **Realisation**: Appendix F:4

Settings screen

The settings screen contains a tick box to mute music, it also will have volume sliders.

Related Requirements:1.3.2 **Realisation**: Appendix F:5

Pause screen

The pause screen will let the player quit, go to the settings menu or resume the game.

Related Requirements:1.2.1 **Realisation**: Appendix F:6

Number of Players Select Screen

The number of players select screen allows the player/players to decide how many players you want in your multiplayer game. The screen features a very simple and intuitive slider which sets the number of players.

Related requirements: 2.1.5 **Realisation**: Appendix F:7

Player Switch Screen

The player switch screen is shown every time the game switches from one player to another in multiplayer mode. It says which player goes next and also shows a leaderboard table detailing who is currently winning at this stage in the game.

Related requirements: 2.1.5 **Realisation**: Appendix F:8

Puzzle Screen

The puzzle screen is the screen which lets the player attempt to complete the puzzle and enter the secret room. The puzzle screen looks like a bookshelf with book style buttons. These buttons can be clicked causing them to move. If the player gets the sequence wrong these books all reset. If the player gets the sequence right this screen is replaced by the secret room and the main navigation screen as usual.

Related requirements: 4.1.5 **Realisation**: Appendix F:9

Testing

Methods & Approaches

We decided to take a requirements focused view of testing, so that our tests are built around making sure the requirements were being met for the project, but also making sure that we tested key parts of our code that may not directly correspond to any requirement. We decided to use two types of testing, these can be seen below.

As we chose a Java project for Assessment 4, no changes to the testing approaches or methodology were made. The project we chose uses JUnit unit tests, like our original project, the only difference is that we enabled CircleCI automated testing. We decided to keep our original testing report, and adapt it where necessary. We have also kept most of our acceptance tests, as we felt they were still applicable and are appropriately linked to our requirements. The unit tests were taken from the project we inherited, and when needed we continuously added to, modified or removed these tests as we developed the project.

Acceptance Testing: this allows us to ensure the end product meets the requirements, and that a user can perform common scenarios successfully.

- Each test takes the form of a list of steps to carry out in the game.
- If a user can perform the list of steps successfully, the test passes, else it fails.
- These tests have to be run manually, because it would take too long to automate them and we have time constraints on the project.
- They are good for testing both functional and nonfunctional requirements.
- Can test functional requirements as the tests can confirm whether the game has the appropriate functionality to meet the requirement.
- Can test non-functional requirements as tests can confirm the project behaves as expected.
- Allows us to detect regressions during refactoring or other code changes.
- We use a **test-driven development** approach here, where the acceptance tests were written before the code, and more tests should pass as more of the requirements are implemented [1].

Unit Testing: this allows us to check that our code does what it is meant to.

- Takes into account both normal and edge cases to ensure normal behaviour and boundaries are handled correctly.
- The unit tests are written as the code is being implemented, due to their reliance on the code structure and design decisions.
- We have used both black box testing (doesn't take into account how code works) and white box testing (uses inner workings of code) when designing our unit tests.
- Allows us to detect regressions during refactoring or other code changes
- Written as JUnit unit tests in Java, because it is compatible with our project, easy to use, and has high quality documentation [2].
- These tests are automated, so whenever a commit is pushed to GitHub, the test script is run on CircleCI (our continuous integration server).
- We chose CircleCI for it's reporting abilities, as it provides a html website, and breakdowns of failing tests with debug logs where applicable, making it easy to diagnose problems.
- After the CI runs the unit tests, the result is pushed back to GitHub and updates our team Slack. GitHub is setup to block merging pull requests if a unit test fails.
- Continuous integration helps remove much of the effort required to run tests, and ensures that unit tests are taken into account when developing the project [3].
- These tests also alert us if the project fails to build with each commit, as the project is built by the CI server in order to run unit tests. This can let us know about possible code problems, like syntax errors.

Our software development process involves creating new branches for new features, and when a branch is ready, a pull request is created to merge the branch into the master branch. We've setup our pull request flow to block merging if the unit tests have failed, which helps us catch code issues sooner rather than later. Even within a couple of days of starting programming, the testing checks had saved us from merging code that looked good, but was in fact broken in a non-obvious way.

For Assessment 3, we added automated test coverage, using JaCoCo, which determines how well unit tested our project is. JaCoCo is an open source toolkit for measuring Java code coverage, which indicates how much of the code is tested [4]. This has been helpful for identifying which parts of the game are lacking in tests, and we've used the associated data to improve our test coverage. We run test coverage alongside our JUnit tests on CircleCI so that we always have a current statistic that we can use. We continued to use both of these in Assessment 4.

Bibliography

[1] AgileData.org - Test Driven Development [Online] http://agiledata.org/essays/tdd.html [Accessed 16/02/2017]

[2] JUnit [Online] http://junit.org [Accessed 16/02/2017]

[3] Thoughtworks.com - Continuous integration [Online] https://www.thoughtworks.com/continuous-integration [Accessed 16/02/2017]

[4] JaCoCo [Online] http://www.jacoco.org/jacoco/ [Accessed 16/02/2017] Test Report ———-

Overall Statistics

- **52 tests**
- **0 failures**
- **100% successful**

Unit Tests

Below are the statistics generated by our unit test runner on CircleCI for our Assessment 4 code. These tests are functions within the code (although they don't get compiled into the final executable) that tests a small unit of code at a time. They are fully automated, and were automatically run when the appropriate commit was made to GitHub.

- **23 tests**
- **0 failures**
- **100% successful**

All of the unit tests in the project have passed, which indicates the absence of bugs in the tested code sections. More comments on what this result means are on the next page. The unit tests have test IDs indicated by 1, followed by their index in the list.

See Appendix E for a full listing of the unit tests.

Acceptance Tests

We have manually run through our acceptance tests to check the game works as intended, and to verify that the game fulfils the requirements. The results are shown below.

- **29 tests**
- **0 failures**
- **100% successful**

More comments on what this result means are on the next page. The acceptance tests have test IDs indicated by 2, followed by their index in the list.

See Appendix D for a full listing of the acceptance tests.

Test Information

The tests are associated with an appropriate requirement to allow for traceability, and to check that the code meets any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly tested, and some tests do not link directly to a requirement but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly work as intended.

Results & Evaluation

The acceptance and unit tests for this project all passed.

The test completeness is not perfect, nor is it possible to be. The unit tests only check their section of the code works as intended, and doesn't cover the integration of that code, or it's use within the entire project. What the unit tests do indicate, is that the specific code that they test works as intended, and they indicate the absence of bugs in that specific code. When combined with the acceptance tests, the test completeness is improved (but still not perfect) as not only are key functions of the code tested, but the overall product is tested to ensure it meets the requirements.

This project doesn't have perfect test correctness either. The unit tests could have bugs in them meaning bugs could be missed in the code, or the unit tests may not cover every edge case or normal use case that exists, which means bugs and issues could slip by undetected. The same sort of thing happens with acceptance testing, as typically a limited range of scenarios are tested, which may not account for the all of the very many possible ways of using the game. Also, the acceptance tests included in this project need to be run manually, which means that human error could occur and affect the overall correctness of the tests.

To improve our testing **completeness** and **correctness**, additional types of testing would be useful additions, such as fully automated end-to-end testing, or integration testing, along with more tests of any type. Introducing different types of tests would alert developers about the presence of a different kind of bug, which would allow identification of bugs that have previously been uncaught. Adding more tests would also decrease the chances of code regressions being missed, or other issues or bugs being missed during the testing process.

Executable Test Plan

Unit Tests

This project is unit tested with JUnit. Tests are located in the `/game/tests` directory in the GitHub repository. For documentation on writing these tests, please see https://github.com/junit-team/junit4/wiki

Coverage

The project includes Jacoco coverage, which checks to see how well tested our classes are, the higher the coverage the better. This is ran automatically by CircleCI and availible under the "Artifacts" tab.

Running Unit Tests

For every commit CircleCI runs all the included tests as well as the coverage of these tests, however, we recommend that you run tests locally too before committing.

We have included a handy test configuration inside the repository that can be run from IntelliJ IDEA.

Adding Tests

- Create new class for tests under `/game/tests/src` When naming the class end the name with `UnitTests` for consistency e.g. `PlayerUnitTests`
- This class should extend `GameTester` this initialises the backend of the game so that tests run correctly.

- Import `org.junit.Test`
- Write a test function using assertions, and use `@Test` decorator above it
- See this page for examples of assertions: https://github.com/junit-team/junit4/wiki/assertions
- Run your tests locally and see if they pass!

CircleCI

Viewing Results

After tests have run the results are displayed in the "Test Summary" tab on CircleCI. This tab contains a summary of the testing result, along with any tests that have failed.

If the tests have failed and no test summary is provided, this normally means that the code doesn't compile, or there is a problem with the test code. To gather more information, scroll down to read the console output from when the tests were run.

Also, CircleCI collects test "artifacts", which are located in the "Artifacts" tab. This contains useful output files such as:

- HTML output: A website that provides a visual testing report with more details
- JUnit XML output: XML files for each class that provide raw data about each run test
- Jacoco Coverage Results: These contain a breakdown of the coverage for each of our classes.

Configuration

We have included a configuration file for setting up CircleCI tests in the root directory of the project. See `circle.yml`.

To setup CircleCI, you will need to link your GitHub account. Once this is done, you can add a project within CircleCI, which will automatically setup tests using the configuration file (`circle.yml`) mentioned earlier.

Whenever a commit is pushed to GitHub, CircleCI will run the tests and inform GitHub of the result, which will be displayed against each commit, and in any pull requests.

Acceptance Tests

We've also included a bunch of acceptance tests that can be run manually to ensure the game performs as expected, and meets the requirements. These have not been automated, so will need to be run by hand. These can be found in the Appendix for the Assessment 2 documents.

Evaluation & Testing

Within the product development process, it is important for to evaluate how well the final product fits the brief to ensure the product is useful for its intended purpose. It is also important to test the final product, which means that the code must be of an appropriate quality.

Evaluation

Product evaluation involves determining if the final product meets its brief, to do this we used the requirements that had been produced and adapted throughout the assessments.

We produced a set of specific, detailed requirements for the initial brief, which were used during product development to ensure the game did what it was supposed to. These requirements are known to meet the brief, making them useful for evaluating the final product.

Due to the changes introduced to the brief in Assessment 4, the requirements produced for earlier assessments were modified and adapted to fulfil the brief. The changes include requirements [2.1.5, 4.1.1, 4.1.4, 4.1.5]. These can be found in the requirements document on our website. The code was adapted accordingly.

Using the requirements document, we evaluated our final product. To achieve this we met as a team, where we went through each requirement, and ensured it was properly implemented in the final product. This also involved using our acceptance tests to ensure that an end user can perform appropriate actions that prove the requirements have been met. See Appendix D for the acceptance tests.

As our code was written using the principles of test driven development, we wrote acceptance and unit tests before we started coding. We wrote the acceptance tests beforehand using the requirements as a guide - this proved helpful for our evaluation, as it meant we had a set of tests to run that ensure our final product meets its brief.

The acceptance tests allowed us to evaluate our final product effectively as they allowed us to ensure the solution implemented in our final product performed as expected, and that the associated requirements were met.

Although acceptance tests were useful for evaluating the final product, not all the requirements had associated acceptance tests, which meant we had to determine as a team whether the untested requirements were met. For example, we had to manually determine that this requirement was met: [1.1.4] Must run on university computers.

Testing

Our team focused on testing, which involved ensuring the quality of our final product was high throughout the assessments. We focused on setting high standards for the code we were writing, as well as ensuring the game was fully tested and maintainable – all qualities important for a software engineering product.

The approach we took to testing was consistent across every assessment completed by our team – the only addition in Assessments 3 & 4 was code coverage. Other than that, we stuck to the same tooling and methodology as we felt it had been successful.

We wrote the game code using the principles of test driven development, using a combination of unit tests and end-to-end acceptance tests. The purpose of this testing was to ensure our code was of high quality – and using two different types of testing mechanisms helped with this. We had regular code reviews and continuous integration to aid with code quality, this is detailed later.

As a team, we determined appropriate quality code for our product to meet the following points:

- Code should function as expected
- Code should be covered by acceptance tests, to ensure it meets the requirements
- Code should be unit tested where appropriate, to ensure regressions do not occur
- Code should pass all automated unit tests
- Code should be reviewed by a developer not involved with writing it
- Code should be fully documented, including JavaDoc comments

Unit tests

Firstly, we wrote unit tests where possible before we started implementing features of the game. We found this highly useful for ensuring the code quality was high, because it forced us to think about the system design, as well as the function of the code – this includes thinking about inputs, outputs and API design before writing a single line of code. This approach involved ensuring our code functions as expected, and they tended to be written using a white-box approach. See Appendix E for the unit test listings.

In general, we enforced a development rule that meant all new code had to have unit tests, if achievable. Writing unit tests wasn't practical for every part of the codebase – especially UI orientated parts of the game. This is because often there are many ways of implementing the same thing, and we were unable to find an approach to testing the user interface with a reasonable amount of effort. We were limited in time, so there was a trade-off between amount of unit testing, and amount of progress on the product we had to consider. However, we used test-driven development with unit tests wherever it was reasonable to do so, and where it wouldn't take up too much time we could better spend on other aspects of the project.

Alongside writing unit tests, we set up a continuous integration server to run our tests every time a commit was made to our code repository. This was an added check to ensure we did not break anything in the codebase while implementing new features – this has helped ensure code quality is high, because it has allowed us to detect regressions and possible bugs in the code, with minimal effort after the initial implementation of the unit tests. Before committing any code to our "master" branch, we ensured all unit tests had passed on our continuous integration server.

Our continuous integration server also ran a script that determined code coverage – this is a key indicator that allows us to identify how much of the codebase is unit tested. We aimed to maximise our test coverage throughout the development process – we inherited a codebase with 20% coverage, and ended up with 33% coverage in our final product. The test report is available on our website. Ideally, we would have liked to increase that test coverage much more, however we were time limited and felt resources would be better spent elsewhere.

Code reviews

Our development process involved code reviews at every stage. This was a way of allowing our team to test and ensure the quality of code being written is high.

Each new feature or bug fix that was implemented in our product was coded normally by one or two individuals in a separate branch, a copy of the "master" branch of the code. Our "master" branch of code was the latest, working version of the game, and code could not be committed from branches into the "master" branch without having gone through a code review.

Our code reviews consisted of other team members commenting and discussing the contents of code awaiting a merge into master. We did this to ensure that new code is readable, maintainable and sensible – this helps ensure code quality is high because often when somebody who isn't involved in the implementation of a feature sees the code, they spot mistakes or issues not noticed by the developer. This is beneficial, as it has caught issues and problems within the codebase that otherwise would have gone unnoticed, and has helped with the code structure as often reviewers would suggest better ways of doing things.

We would also block merging into the master branch if the unit tests had failed on continuous integration – this would normally be identified by the reviewer at the code review stage.

Acceptance tests

As mentioned earlier, we also wrote end-to-end acceptance tests to ensure that our final product met the requirements we had determined from the brief. These tests allowed us to ensure the user interface worked as expected, and to ensure that all requirements were being met by the implemented code.

Pair programming

Parts of the implementation was done using pair programming – where we had two individuals working together during the coding. The use of this method ensured that these key parts of code were reliable and efficient since there were two people solving problems rather than one.

We decided in group meetings whether to pair program a section of code or not. Pair programming also allowed team members to teach and learn from each which in turn led to better code with less bugs.

Code documentation

Throughout the development process, we focused on adding code comments and documentation of code base to ensure that other developers can pick up our code and be able to maintain it. We used JavaDocs for comment formatting - this proved useful because it allowed us to generate a HTML website containing structured and organised formatting of our code documentation, which we found helpful to reference while developing.

Overall we felt like the code quality and testing approaches we took in this project were successful towards ensuring a high standard of work, but as ever there are more things we could have done. Ideally, we would have liked to have achieved a greater test coverage overall, however we were time limited, and it wasn't feasible to write extensive testing for the project we inherited in the time we were provided with.

Meeting the requirements

The product produced by our team meets almost all of the requirements determined from the brief given to us by the client. However, there are a number of minor requirements that weren't completed in time for the hand in, these are discussed later.

In general, we feel the product produced by our team has fulfilled the requirements set for the product - it is a usable, playable and enjoyable murder mystery game set in the Ron Cooke Hub. As requested, key functionality like dynamic storytelling have been implemented, along with a comprehensive selection of features including scoring, clues, characters, and additional items like the journal/inventory that aid the user experience.

The game we have built includes turn-based multiplayer support as requested - this was implemented in a flexible manner, initially to support two players. Due to the code structure of the implementation, this flexibility has given us the ability to have as many players as necessary with no negative effects on the gameplay, all by changing a single integer variable. We felt this was a good thing to show off, as it demonstrated that the codebase is quality and not prone to breaking with additional modifications and enhancements.

However, not all of the requirements our team set for ourselves were met. These are as follows:

- 1.3.1 Could be controlled by a gamepad
- This wasn't implemented in our final product due to lack of time, the team felt this feature didn't provide enough of a benefit with respect to the amount of work involved in implementing it. This requirement was always a nice-to-have thing and wasn't initially specified by the client, so we decided to leave it out of the final product for this reason.
- 6.2.1 There could be an online scoreboard to keep high scores
- As with the previous requirement, an online scoreboard would have been a bonus feature, and wasn't specified in the brief from our client. This requirement would have required a web service to be developed in order to store the scores online - our team felt this wasn't achievable in the time we had to develop the product.
- 7.1.7 The player must be shown introductory and closing dialogue
- This was a requirement we had implemented in our Assessment 3 project, which provided users with an introduction to the game and wrapped it up at the end. We intended to implement this requirement for Assessment 4, however we had to prioritise features, and came to the conclusion our time could be better spent elsewhere.

The full requirements document containing all of the requirements for the final product, including the un-implemented ones, can be found here.

Overall, our team is satisfied that the final product we have produced meets all of the core requirements of our client, despite missing a couple of nice-to-have features. Given more time, our team would have liked to add additional features and functionality to the game, to give it more polish and an improved user experience, however we are completely happy with the outcome of our final product, and are satisfied it meets the client's needs and requirements.

Architecture & Traceability

Overview

The project we inherited [1] used an updated version of our game for Assessment 2. The same basic architecture that we specified for the original project [2] still applies, with any changes that Team JAAPAN applied. We made necessary changes to the architecture to adapt the project for the new Assessment 4 requirements, these are detailed below.

The overall architecture of our final software project is presented as a UML 2.X model [1]. This is because UML is a clear, industry-standard format for presenting the structure of a software project, as it indicates classes, as well as their attributes, methods and relations. To generate the attached diagram, we used the built-in functionality included in the IntelliJ IDEA IDE.

Attached final UML diagram: http://lihq.me/Downloads/Assessment4/UML/Final.png (Ignore non-standard UML notation)

Throughout this document and others in this assessment we reference our requirements using requirement ID's ([X.X.X]), these ID's correspond to a row the requirements table [3]. This makes for great traceability throughout the project.

Architecture

Packages

Where appropriate, the architecture is further separated into packages - this improves the structure of the code because it allows for a structured organisation of the classes. The packages are reusable, aiding with maintainability, as well as providing the ability for reuse in another project if necessary. The UML model featuring the un-expanded packages which can be found here http://lihq.me/Downloads/Assessment4/UML/Packages.png.

The packages in the architecture are outlined below:

- "screen" - Expanding upon LibGDX's approach to screens, we created our own screen manager. Our game screens represent different states of the game, for example the navigation screen, this state controls the player moving around the map.
- "screen/elements" - Contained within the screen package is the elements package. Elements contains all the universal UI elements that are layered onto screens. These allow us to repeatedly use the same UI elements across the game, this makes extending the game adding new screens or sections very simple.
- "people" - This contains all people related classes, including AbstractPerson, NPC's or the Players.
- "people/controller" - Within the people package is the controller package. This is contains all of the input related classes for the game, as users are represented by the Player class.
- "models" - This package contains all the main objects of the game, and these store the associated data. Some of these classes hold instances of one another. For example, the 'Map' has a list of 'Room's, which in turn, has a list of 'Clue's in that room.

Inheritance & Abstraction

We have used inheritance and abstraction of classes throughout the architecture, either by extending existing LibGDX classes or by creating our own abstract classes. The purpose of this is to help with structure and maintainability, as code is reused where possible, and classes are sensibly named and designed.

- AbstractScreen [1.1.1, 1.2.1, 2.2.1, 5.2.1, 6.1.1] - this is a base class for user interface screens. It contains key methods and attributes that need to be implemented by individual screens.
- AbstractPerson [2.1.1, 2.1.5, 3.1.2, 7.1.6, 3.1.6] - this class contains methods that relate to any character in the game. Both the ‘Player‘ and ‘NPC‘ classes inherit from ‘AbstractPerson‘ as they share functionality.
- We have extended LibGDX’s Sprite class, used for objects that are displayed in the game like Clues [5.1.1, 5.2.2] and AbstractPerson (NPC and Players). This makes it easy for us to display the objects correctly and provides consistent methods to draw these on the screen making development easier.

The use of abstraction has further meant that the structure of our final architecture is easy to maintain and expand, something which we experienced first hand when adding the puzzle to the game.

Utility Classes

The architecture includes some static classes that contain methods used by many classes:

- UIHelpers [1.1.1, 1.2.1, 1.3.3, 2.2.1, 5.2.1, 6.1.1, 7.1.7] - this provides methods to generate buttons and other basic UI elements, which has lead to a consistent UI that is easy to change in the future.
- Assets [1.1.1, 1.3.2, 4.1.1, 5.1.1, 5.2.1] - this class contains methods for loading the various assets used by the game such as fonts, music, sound effects, UI skins and other UI elements.
- Settings [1.2.1, 1.3.3] - this class stores all of the game-wide options. This includes options such as: volume, map zoom and other variables/constants that need to be accessed in many classes.

Architecture changes for Assessment 4

Other than the changes outlined below the final architecture has remained unchanged from the previous [4]. We found that the previous architecture was well structured, and easy to work with, and fulfilled all the necessary requirements of the game.

The brief for Assessment 4 contained a number of changes that required the modification of the game’s architecture. The changes we made are listed below.

Multiplayer [2.1.5]

- To implement turn-based multiplayer within the existing game, we had to restructure the architecture. We introduced a GameSnapshot class to keep track of player specific game objects. This is because many objects within the previous architecture were stored in different places, making them difficult to keep track of, and this would have proved challenging when it came to switching players.
- The GameSnapshot class contains all the data that is unique to one player’s game. This is so the actions of one player don’t affect the game of another player. Introducing this class simplifies the processes of:
- Creating new games for each of the players
- Changing to another snapshot when it’s the next player’s turn without losing the data of the previous player.
- To simplify the creation of GameSnapshot instances, we added another class called ScenarioBuilder that allowed us to easily generate identical games for each of the players. Moving this logic to a separate class made it easier

to create new snapshots, and improved the clarity of our final architecture by separating out the generation and storing of associated game data.

- To expand the game in the future, new variables and objects that are specific to one player should be defined in the GameSnapshot class and non specific global variables and objects in one of the screen classes or GameMain.

Puzzle [4.1.5]

- The addition of the puzzle was simple and required a couple of additional classes and no major restructuring.
- The puzzle is separated into two classes, PuzzleScreen and PuzzleElement. These classes are an extension to the previous architecture.
- The PuzzleScreen extends from the AbstractScreen and is used to render the user interface elements of the puzzle.
- The Puzzle class is contained within the element package, and is responsible for the puzzle UI and logic.
- Each player has their own puzzle assigned to them which is stored in the GameSnapshot. This is so that each player's puzzle is unique to them.

Secret Room [4.1.4]

- Due to the structure of the previous architecture adding the secret room meant no additional classes were needed.
- A new map file was created in Tiled for the room graphics.
- We initialized the secret room as a new instance of the Room class, which was referenced in the Map class. Some additional methods were added to existing classes to accommodate the unique way players access this room, otherwise the architecture remained the same.

Screen Manager

- Although this wasn't needed for the requirements, we decided to add a screen manager class to the architecture. This as it abstracted the switching of screens away from the already large main game class which leads to easier maintainability of the code and simplifies the method of switching between screens.

Bibliography

[1] "Team JAAPAN Documentation", Team JAAPAN - University of York, 2017. [Online]. Available:http://jaapan.alexcummins.uk . [Accessed: 06- Apr- 2017].

[2] "Architecture report - Lorem Ipsum", Lorem Ipsum - University of York, 2017. [Online]. Available: http://docs4.lihq.me/en/latest/Assessment2/architecture.html. [Accessed: 06- Apr- 2017].

[3] "Requirements table - Lorem Ipsum" Lorem Ipsum - University of York, 2017. [Online] http://docs4.lihq.me/en/latest/Assessment4/requirements.html [Accessed: 06- Apr- 2017].

Project Review

Approach to team management

We decided to use agile-scrum methodology [1] with bi-weekly meetings and weekly sprints to give us a dynamic and flexible approach to software development. During our meetings, we discussed how the project was progressing and assigned tasks to each other. These tasks made up the sprint, and lasted up to a week. Our meetings allowed us to adapt our workload based on our current situation, or when necessary to help deal with new risks and requirements that arose.

Our first few meetings consisted of general discussion about what needed to be done - often this meant our meetings went off-track and we would spend time discussing ideas and tasks that weren't a priority. We noticed communication issues when key decisions were being made (Risk ID:11), as several team members were ill-informed of changes to parts of the project and were unable to adapt their own sections quickly enough. To solve these problems we decided a team leader was needed to help manage the project – this role involved planning meetings and ensuring enough progress was being made. After our first assessment, the team leader found the workload was too high, so this lead to us having two team leaders who alternated between each assessment.

Even with a team leader we found "a lack of communication" (Risk ID:11) and "arguments within the team" (Risk ID: 10) to still be an issue. In accordance with the mitigations in the risk assessment document, we decide to use GitHub issues [2] to assign members to tasks and have them feedback their progress in the comments, this made key decisions easy to find and debate.

We began to learn each other's strengths and weaknesses, and assigned tasks to each other accordingly. Initially, this meant team members with more technical experience took charge of the code, whereas other members took on more documentation – this meant some members didn't have a good understanding of key design decisions and lead to productivity loss. To solve this we engaged in pair programming to get every member up to speed with the code, and spread documentation and programming across the team.

As the project progressed, we faced dissatisfaction within the team about how the project was being managed by the team leader. The leader was assigning tasks to individuals and checking up on progress to ensure things got done, however this approach meant that some members weren't happy as they would have preferred more flexibility and team consensus on key issues.

We had a discussion and realised we hadn't clearly agreed the scope of the team leader role, and individuals were getting frustrated as everyone wanted the best for the team. We realised we weren't updating GitHub [2] Projects frequently enough, meaning it was hard to keep track of how well the project was progressing. This led the team leader to assigning tasks to individuals so progress would get made.

For the last two assessments we agreed to update GitHub more often, so that we could all keep track of project progress. We found that approach to be more effective, however some members still thought the team leader had too much control. Therefore we decided to replace the team leader with a meeting leader who would set an agenda for meetings, and we left project management to the entire team.

Despite our best efforts, issues were found with our new team structure. Although some members proactively picked up and completed whatever bits of work had to be done, some others did less work than in previous assessments due to the lack of structure and control. Over the holidays we aimed to meet at least once to ensure progress was being made. This worked for us during the Christmas holiday for Assessment 2, however over Easter this did not happen (Risk ID:11) due to a lack of leadership. If we had to do another assessment, we'd probably reinstate a leader with a different job. The leader would ensure progress is being made and set up meetings and their agendas, but not assign tasks to individuals unless there were serious problems, in which case the entire team would be involved.

Software engineering tools and methods

From the start of the project, we chose several tools to use for software development, some of which were more effective for us than others. This document discusses how these tools evolved over the course of the project, but for overall details on tools and the justification of their use, please see the methods document.

Programming Tools:

We chose Java [3] and the LibGDX [4] library for development purposes, and continued to use it for all our assessments because we found them easy to work with and well documented. They featured everything we needed to create the game. Throughout the assessments we had the option of selecting a C# with Unity project, however we decided to stick with Java because we felt those projects had a more maintainable structure, and we were familiar with the technologies used.

The team decided to standardise on IntelliJ [5] throughout all of our assessments. An advantage of IntelliJ is its ability to produce an accurate UML (Unified Modeling Language) diagram based on our code for us, saving us the time of having to manually do this as we did in first assessment by using draw.io [6] as described in the SEPR [7] lecture on UML.

Testing:

We used test driven development (where appropriate) writing unit tests using the JUnit [8] library with CircleCI [9] to run our unit tests whenever a commit was made to GitHub. The usage of unit tests and CircleCI helped limit the occurrence of bugs in our code and mitigated "Our software doesn't work as intended" (Risk ID: 5). This saved a lot of time that would have otherwise been spent finding and fixing bugs in the project and found them to work well together. This was a factor in choosing a Java project, as we could take our existing knowledge and apply advanced testing tooling to other teams' projects. We later added JaCoCO to run test coverage on our code to motivate us to improve our code.

Collaboration Tools:

Initially we used Facebook Messenger [10] as our main communication tool as all members used it frequently, however we soon found that things were getting lost in one big thread, so we decided to switch to Slack [11]. During the development process, our team found Slack useful as it was synced to our GitHub repository to give us all updates, however throughout the process we often found ourselves reverting to Messenger for convenience, as notifications weren't always reliable.

For remote scrum meetings we used a tool called Join.me [12], we found this very effective for meetings during the Christmas holiday period, and for remote demonstrations and pair programming. This helped prevent (Risk ID:11) from occurring over the holidays.

We used GitHub [2] as our code repository, it allowed us to intuitively keep track of code changes, and collaborate on the same piece of code. One risk that is common with code collaboration tools is "Conflicts in Git. Different members changing the same code" (Risk ID: 4) - to mitigate this we avoided working on the same file at any one time by separating tasks up, we were not aware of any tool that would remove any of these issues fully. Despite this, we found using GitHub very effective and used the surrounding functionality like projects and issues extensively. The ability to enforce a code review before merging branches proved essential for catching issues, and the integration with CircleCI meant that we were always aware of testing.

Task Management

To help manage the project, the Projects and issue features of GitHub were used to track of project progress and assigned to tasks to individuals. We found it ineffective to start with, because we often forgot to keep it up to date, however for the final assessments, we strived to update GitHub which proved useful as we felt the benefit of being able to see who was doing what and the project status in one place.

We conclude that if we were to do the project again, we would use the same tools again as they helped us effectively achieve the project goals. Throughout the project, we found it useful to evaluate how well our tools were working for us, and what we could do differently to improve.

Bibliography

[1] B.Gupta, 2012. [Online]. Available: http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf. [Accessed: 27-Apr-2017].

[2]"Build software better, together", GitHub, 2017. [Online]. Available: https://github.com/ . [Accessed: 27- Apr-2017].

[3]"java.com: Java + You", Java.com, 2017. [Online]. Available: https://www.java.com/en /. [Accessed: 06- Apr-2017].

[4] “libgdx”, Libgdx.badlogicgames.com, 2017. [Online]. Available: https://libgdx.badlogicgames.com/ . [Accessed: 06- Apr- 2017].

[5]”IntelliJ IDEA the Java IDE”, JetBrains, 2017. [Online]. Available: https://www.jetbrains.com/idea/ . [Accessed: 06- Apr- 2017].

[6] “Draw.io”, jgraph, 2017. [Online]. Available: https://draw.io/[Accessed: 03- Jan- 2017]

[7] “SEPR”, Vle.york.ac.uk, 2017. [Online]. Available: https://vle.york.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_8
[Accessed: 06- Apr- 2017].

[8] “JUnit - About”, Junit.org, 2017. [Online]. Available: http://junit.org/junit4/ . [Accessed: 06- Apr- 2017].

[9] “Continuous Integration and Delivery”, CircleCI, 2017. [Online]. Available: https://circleci.com/. [Accessed: 06- Apr- 2017].

[10]”Messenger”, Facebook, 2017. [Online]. Available: https://en-gb.messenger.com/. [Accessed: 06- Apr- 2017].

[11]”Slack: Where work happens”, Slack, 2017. [Online]. Available: https://slack.com/. [Accessed: 06- Apr- 2017].

[12]”Free Screen Sharing, Online Meetings & Web Conferencing | join.me”, Join.me, 2017. [Online]. Available: https://www.join.me/. [Accessed: 06- Apr- 2017].

A: User Scenarios

Persona 1

David, 18 year old male at a Computer Science Open Day. He is not an avid gamer and his interests lie more in electronics.

Scenario

David is initially unsure as to how the game is played, however he is provided a tutorial by the narrator which explains the puzzle, how he is expected to solve the puzzle, how to move the main character, interact with clues and non player characters. After becoming accustomed to the keys and finding a clue, he is unsure as to what the next step is to solve the puzzle. He interacts with the narrator who subsequently provides hints as to the next course of action that will get the user closer to solving the puzzle. Near the end of the game he becomes stuck and spends a long time attempting to find out which of his 3 final suspects is the murderer, the narrator pops up on the side of the screen to ask whether the player requires assistance, David says “yes” and is told to speak to a character which holds the final piece of information required to solve the puzzle.

Persona 2

Louise, 17 year old female student at an open day. Enjoys playing puzzle games and is colour-blind.

Scenario

Although she is initially worried about missing a clue or finding it hard to read some of the text, she is able to go through the introductory tutorial without issue due to the text colour and background colour contrast in a way to avoid issue with most forms of colourblindness. She is able to see the clues quite clearly due to map tile colours and clue item colours being chosen in a way so as to not cause problems to most people will colourblindness. She is then able to solve the crime quicker than most and accuse the correct murderer of the crime on her first try thanks to her experience played puzzle games.

Persona 3

Ben, 19 year old second year Computer Science student. He is an avid gamer who takes games quite seriously and always tries to get the high score.

Scenario

He progresses through the tutorial quickly and is already accustomed to using keyboard keys to interact with the game. As he attempts to concentrate and solve the puzzle as quickly as possible so as to get the high score, the background noise begins to bother him. Fortunately he notices that there is a “turn music off” button in the sidebar of the game and this allows him to continue playing comfortably. Although he finishes the game with a good score, he is unable to beat the high score and wishes to play again, fortunately due to the game having a set of different characters from which murderers and victims are chosen randomly, he can play several times whilst going through a new, different and exciting storyline each time.

C: Survey Results

Figure 1

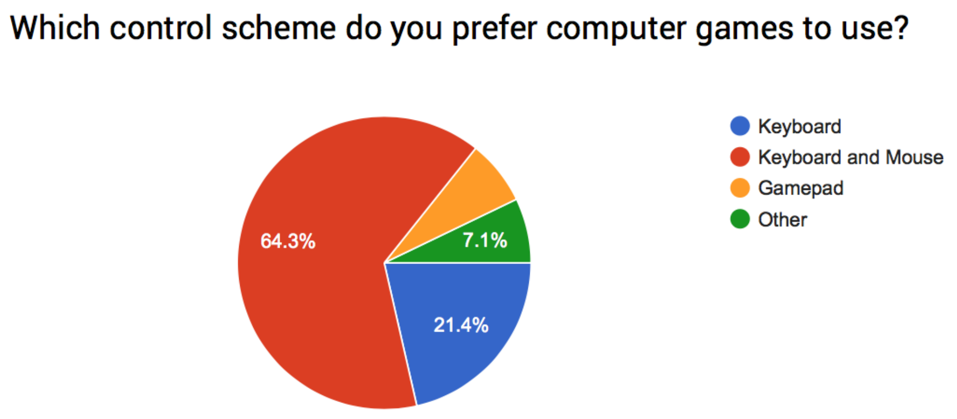
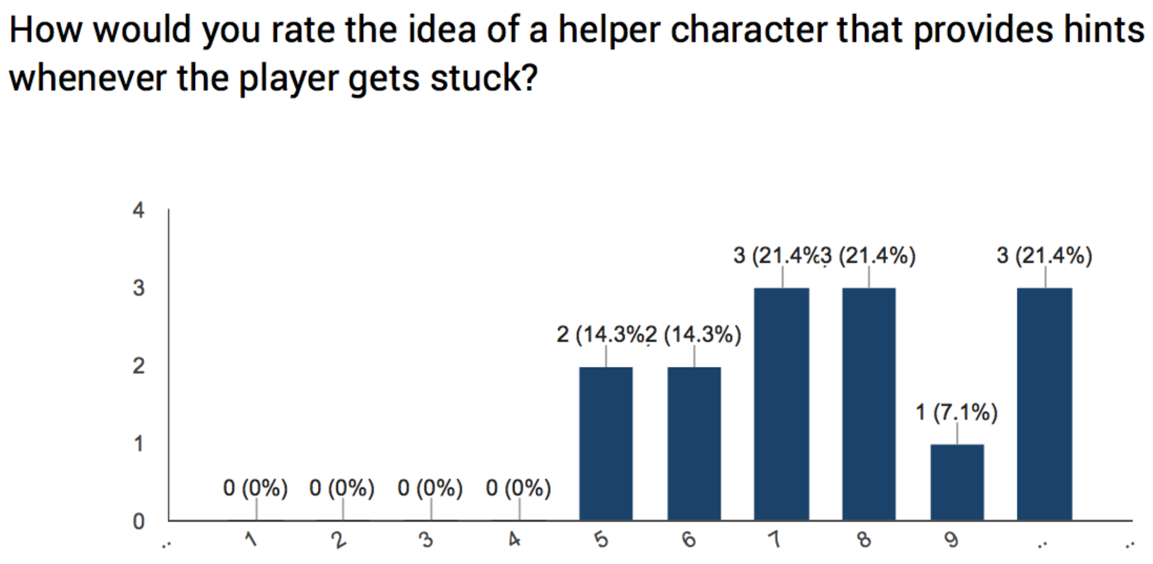
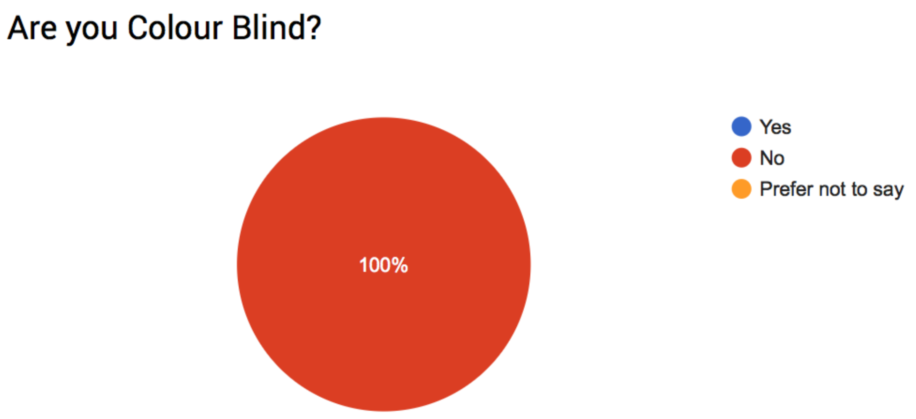


Figure 2



Bar Chart for Helper Character 1 = not helpful 10 = extremely helpful

Figure 3



Pie Chart Colour blind.

D: Acceptance Tests

Below is a table of the acceptance tests included within this project.

These tests need to be manually run using a copy of the game executable. They involve following a series of steps, verifying each of the assertions in the steps is true, and that the relevant GUI exists to allow the steps to be carried out. If all of the steps can be carried out, and their assertions are true, the test passes. If not, the test fails.

The acceptance tests are associated with an appropriate requirement to allow for traceability, and the tests aim to check that the code works for any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly unit tested, and some tests do not link up directly to a requirement, but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly function as intended.

Test Listing

ID	Test Steps	Req ID	Criticality	Result
2.01	To test the Main Menu: • Run game executable. • Check that the Main menu is shown. • Click on the New Game button. • Ensure the screen changes to the Narrator Screen. • Restart the game and click on the Quit button • Ensure that the game closes.	1.1.1	Low	Passed
2.02	To test the player movement using key presses: • Run game executable • On the Main Menu click on "New Game". • On the Narrator Screen click on "Start Game". • Once the game has loaded, press "W" on the keyboard. • Ensure that the player has moved upwards. • Press "S" on the keyboards • Ensure that the player has moved downwards. • Press "A" on the keyboard. • Ensure that the player has moved to the	1.1.2	High	Passed

E: Unit Tests

Below is a table of the unit tests included within this project.

The unit tests are associated with an appropriate requirement to allow for traceability, and the tests aim to check that the code works for any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly unit tested, and some tests do not link up directly to a requirement, but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly function as intended.

ID	Test Name	Purpose	Crit- i- calit y	Class	Req ID	Re- sult
1.01	testName	Verifies the name of a clue has been set correctly	Low	ClueU- nitTe st	5	Passed
1.02	testDescri ption	Verifies the descriptio n of the clue has been set correctly	Low	ClueU- nitTe st	5	Passed
1.03	testTileCo ordinates	Verifies the location of the clue has been set as expected	High	ClueU- nitTe st	5.1.1	Passed
1.04	testEquali ty	Verifies that identical clues are considered equal	Medium	ClueU- nitTe st	5	Passed
1.05	testMurder Weapon	Verifies that a murder weapon has been chosen correctly	High	ClueU- nitTe st	5.1.4	Passed
1.06	testRedHer ring	Verifies that Red Herrings have been chosen correctly	Low	ClueU- nitTe st	5	Passed
1.07	testFinish Interactio n	Verifies that the game snapshot class keeps track of how many interactio ns the associated player has left in their current turn correctly	High	GameS- napsh otTests	2.1.5	Passed
1.08	testGet- NPC s	Verifies that data stored on NPCs is stored and retrieved correctly	High	GameS- napsh otTests	3	Passed
1.09	testGet- Nam e	Verifies that the name stored and retrieved for the NPC is correct	Low	NPCU- nitTes ts	3	Passed
1.10	testPerson ality	Verifies that the personalit y stored and retrieved for the NPC is correct	Medium	NPCU- nitTes ts	3	Passed
1.11	testIntera ctFindingC lues	Verifies that the correct clue is correctly collected when interacted , also correctly altering the score	High	PlayerUnit Tests	5.1.2	Passed
1.12	testPlayer Name	Verifies that the playerName is stored and returned correctly	Low	PlayerUnit Tests	2.1.5	Passed
1.13	testPlayer Personalit y	Verifies that the players personalit y can be manipulate d and stored correctly	Medium	PlayerUnit Tests	2.1.1	Passed
1.14	doesPlayer Move	Verifies that the player is able to move correctly in all four cardinal directions	High	PlayerUnit Tests	2.1.4	Passed
1.15	test- CanAcc use	Verifies that the player is not able to accuse without evidence	Low	PlayerUnit Tests	7.1.4	Passed
1.16	testScore	Verifies that the players score can be modified correctly	Medium	PlayerUnit Tests	6.1.1	Passed
1.17	testPlayTi me	Verifies that how long a player has played for is stored correctly	Low	PlayerUnit Tests	6.1.2	Passed
1.18	testGetTra nsition	Verifies that the player transition s between rooms correctly	High	RoomU- nitTe sts	2.1.4	Passed
1.19	testAddTra nsition	Verifies that new transition s are added correctly	High	RoomU- nitTe sts	2.1.4	Passed
1.20	testWalkab le	Verifies certain tiles are and aren’t walkable	Medium	RoomU- nitTe sts	2.1.4	Passed

F: GUI Examples

Download (AppendixF.pdf)

Requirements

Introduction

At our first meeting we read the brief, and with that in mind, we played cluedo to get a feel for a general detective game. This sparked discussion as to what works in a game, and it gave us ideas for how our game should work. We decided the most effective way of thinking about the game was like "guess who". Using the brief and our new ideas, we produced a list of features we'd like to include in the game, and then prioritised those.

After the initial meeting, we produced a number of user scenarios [1], and used our list of features to help produce a draft of the requirements. From these we identified ambiguous points and produced questions we needed to get answers for. We met multiple times with the customer throughout the design process, to present our requirements and ask questions we had. Using their feedback, we made any necessary changes.

Following the response we got after the Assessment 1 feedback was released, we decided it was best to rewrite our requirements. This lead to substantial improvements in the clarity and categorisation of our requirements, and this has benefitted us as it made the requirements easier to test against when it came to implementing the game.

When designing the requirements, we took these points into account:

The requirements should be categorised as:

- Functional requirements - these define what the system should do
- Nonfunctional requirements - these define the behaviour of the system
- The requirements should be achievable within the time allocated of this project
- The requirements should consider the hardware in which the game should run
- The requirements should meet all points included in the brief
- We produced a survey asking about input methods and accessibility [2]. We got a sample of our target market to respond. From this we found:
- The preferred way of interaction with the game was keyboard and mouse.
- There were no results from colourblind people, however we still felt it was important that our game was accessible, so have included accessibility features as a "could" requirement. Colour blindness is a condition apparent in 1 in 12 men and 1 in 200 women [3] so we feel that it is a requirement that some people would benefit from.

The requirements are laid out in tables based on the IEEE standard for system requirements, as we felt this was a good standard to adhere to. The tables are split according to type (functional or non-functional), and category. Some requirements have associated risks, these are referenced in the table below, and are defined in the risks document.

Each requirement is given a unique identifier to make it easy to locate, and for traceability. The identifiers are made up of three numbers, using the following system:

- The first number is category, this represents the functional area of the requirement
- The second number represents how important the requirement is:
- 1=Must implement
- 2=Should implement
- 3=Could implement

- The third number is the position in the list, used to ensure the identifier is unique.

Functional Requirements

Game

ID	Requirement	Success Criteria	Alternative	Risk ID
1.1.1	To start the game there must be a main menu	The game has a working clear menu	The game starts immediately when its opened	21-24
1.1.2	The game must be fully controlled by a mouse and keyboard	The game can be interacted with using only a mouse and keyboard	The game is controlled by an Xbox controller.	21-24
1.1.3	Game must have different 'plotlines'	Each gameplay is different in some way	Game only has 2 plot lines.	21-24
1.2.1	There should be a way of suspending or pausing the game	There is the option to pause the game which opens a pause menu	The game cannot be paused	21-24
1.3.1	Could be controlled by a gamepad	The game can be interacted with using a gamepad	Game is only controllable by keyboard and mouse.	21-24
1.3.2	Could have a sound track	Music will be played when the game is running	There is no sound track.	21-24
1.3.3	If game has a soundtrack, it must have an option to turn the sound track off	There is a player accessible way to turn the sound track off within the game	The soundtrack would always be on.	21-24

Player

ID	Requirement	Success Criteria	Alternative	Risk ID
2.1.1	The player must have a personality that is customisable	The players personality changes in the game	The player personality does not change.	21-24
2.1.2	Must not be able to accuse a NPC unless enough evidence has been found	The player cannot see the option to accuse an NPC unless they have interacted with enough clues	Can accuse an NPC without evidence.	21-24
2.1.3	The player must start in a central room at the start of every game	When the game starts, the player should be in the centre of the "Ron Cooke Hub"	Player can start in any room.	21-24
2.1.4	The player must be able to navigate between rooms on the map	The player can move throughout the map and transition to other rooms when desired	The player progresses through rooms automatically.	21-24
2.2.1	The player should be able to see their current personality level	The game should present the player with an GUI element showing their personality level	The Player will not be able to see their current personality level	21-24

NPC

ID	Requirement	Success Criteria	Alternative	Risk ID
3.1.1	Each killer must have a motive	The killers all have motives	All killer have the same motive.	21-24
3.1.2	There must be 10 NPCs(non playable characters).	The player should be able to locate 9 NPCs as well as there being 1 victim.	There are less than 10 NPC's.	21-24
3.1.3	The NPCs must all exhibit differing personalities.	The NPCs interact with the player in differing ways.	The NPCs will all act in the same way to the player.	21-24

Map

ID	Requirement	Success Criteria	Alternative	Risk ID
4.1.1	The game must have a map containing 10 separate rooms.	The player should be able to visit 10 different rooms in the game	The game has less than 10 rooms.	21-24
4.1.2	All rooms must be of varying sizes.	The player should be able to notice all the rooms being of different sizes and shapes.	There are several rooms of equal size.	21-24

Clue

ID	Requirement	Success Criteria	Alternative	Risk ID
5.1.1	There must be at least one clue in each room of the map	The player can navigate to every room and be able to locate a clue	Some rooms may have no clues, some may have multiple	21-24
5.1.2	The player must be able to interact with a clue	The player should be able to interact with a clue once it has been located	The player gets the clue without interaction.	21-24
5.1.3	The player must find the 3 separate parts of the motive clue before the full motive clue can be obtained.	When 3 motive clue parts are found, the player obtains the whole motive clue.	The motive clue appears after interacting with 5 NPCs.	21-24
5.2.1	There should be a journal wher e clues are placed by a player for future reference	The player can see a journal in the GUI that allows visibility of collected clues	Clues are stored internally but the player will not be able to see them	21-24

Score

ID	Requirement	Success Criteria	Alternative	Risk ID
6.1.1	There must be a score shown to players in the game	The player must see a score displayed in the GUI	There will be no scoring.	21-24
6.2.1	There could be an online scoreboard to keep high scores	There could be a scoreboard in the GUI that presents the all time high scores	There will be a local list of high scores, or no scoring	21-24

Dialogue

ID	Requirement	Success Criteria	Alternative	Risk ID
7.1.1	The player must be able to interact with an NPC	A player can go up to an NPC and talk to them	The player cannot interact with NPC's.	21-24
7.1.2	The player must have the option of questioning an NPC	When a player talks to an NPC, they should have the option to question them	The player cannot question an NPC.	21-24
7.1.3	The player must have the option of ignoring an NPC	When a player talks to an NPC, they should have the option to ignore them	The player cannot ignore an NPC.	21-24
7.1.4	The player must have the option of accusing an NPC	When a player talks to an NPC, they should only have the option to accuse them if they have found enough clues to accuse the NPC	The player cannot accuse an NPC.	21-24
7.1.5	The player must choose from a set of questions when interacting with an NPC that reflects different personalities	When a player talks to an NPC, and chooses to question them, they can choose from multiple speeches with different personality levels. Eg. Aggressive	The player only has one	21-24
7.1.6	Each NPC must respond differently to questions from a Player depending on both NPC's and Player's personality and characteristic s	When an NPC responds to a player after being questioned, their response must be determined by their characteristic s and the player's personality	All NPC's respond in the same way.	21-24
7.1.7	The player must be shown introductory and closing dialogue.	Before the player can play they are shown an introduction and once they have completed the game the player is given a 'goodbye speech'.	The player will not be given any context dialogue.	21-24

Win/Lose Conditions

ID	Requirement	Success Criteria	Alternative	Risk ID
8.1.1	The game must be 'won' when the player successfully accuses the murderer	If the player accuses the murderer then the game is won.	This is a necessary requirement.	21-24
8.1.2	The game must be 'lost' when the player accuses too many NPCs	If the player accuses too many NPCs then the game is lost.	The game will not be able to be 'lost'	21-24

Nonfunctional Requirements

Game

ID	Requirement	Success Criteria	Alternative	Risk ID
1.1.4	Must run on the university computers	An executable is provided the runs on the computers	The game will not run on university computers.	7
1.1.5	Must run on Windows 10	An executable is provided that runs on windows 10	The game will not run on windows 10.	7
1.2.2	Should run on MacOS	An executable is provided that runs on MacOS	There will not be an executable that runs on MacOS	7

NPC

3.1.4	Each NPC must have a personality that affects and is affected by game play.	The NPC will respond best to different types of question. For example, an aggressive NPC will respond best when questioned nicely.	All NPC's have the same personality.	21-24
3.1.5	The killer and victim must be randomly selected each time the game begins from two sub-lists of killers and victims.	When the game starts, the victim and the killer has been selected at random.	The killer and victim is the same every time.	21-24
3.1.6	Each NPC must be randomly assigned to a room at the start of the game	All NPCs should be situated within a different room at the start of the game.	Each NPC is always in the same room.	21-24

Map

ID	Requirement	Success Criteria	Alternative	Risk ID
4.1.3	The room where the murder occurred must be randomly selected at the start of every game	One random room should be the selected murder location at the start of every game	The murder room is always the same.	21-24

Clues

ID	Requirement	Success Criteria	Alternative	Risk ID
5.1.4	The murder weapon clue must be found before the player can accuse any NPCs	The player cannot accuse an NPC until they've located the murder weapon clue	Can accuse without the murder weapon.	21-24
5.1.5	Most clues must help with identifying the killer	A clue will narrow down the number of suspects left to be the killer	All clues help identify the killer	21-24
5.1.6	At the start of the game, clues must be randomly assigned to each room in the map	There must be at least one clue in every room of the map at the start of the game	Clues always in same location.	21-24
5.1.7	The motive clue must be found before the player can accuse any NPCs	The player cannot accuse an NPC until they've located the motive clue	Can accuse without the motive clue	21-24
5.2.2	Clues could be picked up by a player and placed in a journal	The player can interact with a clue and place it in their journal for future reference	Clues will be stored internally, but my not be seen by the player	21-24

Score

ID	Requirement	Success Criteria	Alternative	Risk ID
6.1.2	The player's score must take into account the time taken	The score must change depending on how long the game has lasted	There will be no scoring.	21-24
6.1.3	The player's score must take into account the number of wrong accusations	The score must change depending on how many accusations the player has made	There will be no scoring.	21-24
6.1.4	The player's score must take into account the number of questions asked	The score must change depending on how many questions the player has asked	There will be no scoring.	21-24
6.1.5	The player's score must take into account the number of clues found	The score must change depending on how many clues have been found by the player	There will be no scoring.	21-24

Dialogue

ID	Requirement	Success Criteria	Alternative	Risk ID
7.1.8	The type of question asked to an NPC by a player must determine the player's personality	When a player chooses a speech to say to an NPC, their personality level is affected by their choice	The type of question asked affects nothing.	21-24
7.1.9	If an NPC is accused and isn't the killer then the NPC must refuse to interact for the rest of the game	When a player interacts with a previously accused NPC they shouldn't get a response	The NPC does not mind being falsely accused.	21-24
7.1.10	If an NPC is ignored, the Player cannot interact with the NPC again until a change in the situation occurs.	The Player cannot question, accuse or ignore an ignored NPC again until another clue is found, the Player moves to a different room or the Player talks with a different character.	The Player can question, accuse or ignore an ignored NPC without any changes to the situation.	21-24

Bibliography

[1] Appendix A [online] http://docs3.lihq.me/en/latest/Assessment3/appendixA.html [Created 21/11/16]

[2] Appendix C [online] http://docs3.lihq.me/en/latest/Assessment3/appendixA.html [Created 21/11/16]

[3] Colour Blind awareness [online] http://www.colourblindawareness.org/colour-blindness/, [Accessed 3/11/16]

Risk Assessment and Mitigation

Introduction

Risk management is an important part of any project, we must prepare for what could happen during the course of the project in order to be able to quickly recover and stay on track. The risks which are shown below take into account the scale of the project and aim to cover only risks which are realistic within this context.

To determine risks we brainstormed various scenarios - such as a teammate being ill for more than a few weeks. From these scenarios, we collected possible risks, and worked out the likelihood of them occurring. To determine the risk we discussed how it would impact the project, focussing on how many knock-on effects the issue would cause.

The risks are presented in a tabular format, with the following columns:

- **Risk ID** - this allows for traceability across the project
- **Description** - describes what the risk is for
- **Likelihood** - each risk has an estimated likelihood on a scale
  - **High** =good chance this risk will occur, about 75% chance
  - **Medium** =equal chance of risk occurring or not, about 50% chance
  - **Low** =not likely to occur, about 25% chance, however some risks may be lower
- **Impact** - this describes the impact the risk would have to progress in the project
- **Severity** - shows the severity of the impact on the project on a scale
  - **High** =a major setback which could affect the whole project

   - **Medium** =could add up to a week of extra work and may threaten a deadline
   - **Low** =may mean a few extra hours of work, but nothing major
- **Mitigation** - describes how how we will aim to avoid such a risk and deal with it
- **Owner** - describes the owner of the problem, where the owner is the person most likely to be responsible for the issue.

The overall table is split into sections which group together similar risk such as software risks. Each section is then ordered by severity, highest first. Equal severity is ordered by likelihood.

This table will be regularly consulted in an attempt to monitor the risks and try to ensure they do not occur and catch them early if they are occurring.

Due to the size of the team we feel that these risks are appropriate and accurate.

Table of risks

Software risks

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
1	Our game may be slow or unresponsi ve.	Medium	No one will want to play the game.	High	Improve efficiency of the game wherever possible and regularly check performance	Coding Team
2	Software library doesn't work or lacks a feature e.g. has a bug that stops the game from working, or is missing a feature required for the game to work.	Medium	We would struggle to implement the feature we want to add. We would also use up lots of time trying to solve the issue.	Medium	Test the elements of the library you plan to use beforehand . Also, make sure the library has an active community surrounding it and that bugs are fixed quickly. If it was early stages we could get a new library but this would require us to rewrite our code to work with the new library.	Software Library Owner
3	Code is hard to understand and seems too complex.	Low	Could cause bugs and makes bug fixing harder. Slows down the productivi ty of the group.	Medium	Use meaningful variable names and plenty of comments, both in code and in commit messages. Make sure code is reviewed by the majority of members before it gets merged into the repository .	Coding Team
4	Conflicts in git. Different members changing the same code.	High	May need to move code around and even rewrite.	Low	Make sure people work on separate elements by assigning them to different tasks and if not then make use of Gits tools.	Coding Team
5	Our own software doesn't work as intended.	High	Will need to bug fix. Loss of time and potentiall y productivi ty if that function or feature is the bottleneck of the game.	Low	This is a normal part of software developmen t. We all make mistakes. However, before code is approved by the group we will use unit testing that will test key functions of the game as we develop them meaning that should a function break we will know about it before it's merged.	Coding Team/Design Team/Project Manager

Hardware risks

ID	Description	Like-li-hood	Impact	Sever-ity	Mitigation	Owner
6	Personal computer breaks long term or is lost.	Low	Could lose work and be unable to work.	Low	Ensure work is saved online to google drive cloud service and that code is stored on github. Department PC's should be accessible most days and have all the tools we need.	Fi-nal User
7	Personal computer crashes while working.	Medium	Potentiall y will have lose work, meaning you lose time doing it again.	Low	Save regularly, google docs[2] will do this for us. Regularly commit code to personal branches so that it stored elsewhere other than your PC .	Fi-nal User

Risks with people

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
8	A team member leaves the module or even the course.	Low	They may have only access to their work, also the rest of the team will have more to do.	High	As above store online but also try to keep each other motivated to avoid this.	Project Team
9	A team member is ill/away for a week or two.	High	They might have been skilled in a certain area that no other member can do well.If they have the only access to work may get behind from it.	Medium	Hard to avoid, but we should store work online where everyone can access. If we work in pairs to complete tasks then there will be less of a chance of having one person who knows the most about one area.	Project Team
10	Arguments within the team.	Medium	Disrupts the work of the team and prevents us moving forwards. Also, unpleasant for the team as a whole.	Medium	Try to avoid conflict but if necessary have proper debates perhaps using a mediator, do not keep issues hidden.	Project Manager
11	Lack of communicat ion.	Medium	Tasks may be done twice or not done at all.	Medium	Keep strong communicat ion using the tools we plan to use.	Project Manager
12	A team member does not do their work.	Medium	Could disrupt other members work and could make the other team members annoyed.	Low	Don't give members too much work or work they cannot do, ensure that the team communicat es well and regularly meets up to discuss how the work is going.	Project Team/Manag er

Risks with tools

ID	Description	Like-li-hood	Impact	Sever-ity	Mitigation	Owner
13	Google drive servers stop working.	Low	Could lose/lose access to work that is stored there.	Medium	Store work locally , and on other services.	Google
14	Central git repository [1] is lost in some way.	Low	Temporaril y lose access to it.	Low	Keep up to date local copies so can be easily restored. We could host our own local copy should github go down.	Git/Coding Team
15	Website hosting fails.	Low	Users lose access to the website.	Medium	The website files are stored on github and every team member has a local copy of the repository on their computer so we could bring the site back up on a different server. The site is also protected by cloud-flar e[3] who will provide a cached version of the site if our host were to go down.	Web-site Host-ing Owner

Requirements risks

ID	Description	Like-li-hood	Impact	Sever-ity	Mitigation	Owner
16	Not including a requiremen t which is required by the customer.	Low	We let the customer down and have failed them.	High	Make sure key requiremen ts are elicited from the customer so they get what they want.	Require-men ts Team
17	A requiremen t could change/ be added.	High	May need to rewrite code or add extra code to account for it. Extra time will be needed.	Medium	Our software architectu re must be flexible and able to be changed easily.	Require-men ts Team
18	Stating a requiremen t that we cannot actually achieve.	High	Let down the customer and also waste time.	Medium	Be sensible when deciding requiremen ts, be sure you can achieve them.	Require-men ts Team/Codin g Team
19	Ambiguity in requireme nts.	Medium	May end up making something which is not what was originally intended.	Medium	Ensure requiremen ts are clear and check any ambiguitie s with the customer.	Require-men ts Team
20	Choosing requiremen ts that the customer doesn't really want.	Medium	Waste time which could be spent on requiremen ts they did want.	Low	Ensure you know which requiremen ts the customer really wants and which can be ignored.	Require-men ts Team

Estimation risks

ID	Description	Like-li-hood	Impact	Sever-ity	Mitigation	Owner
21	Expect the team or a team member can do more than they actually can.	Medium	Work is not done or is done to an insufficie nt standard.	Medium	Give tasks that people can do and if they can't then help them. When working on difficult tasks work in pairs to complete the task meaning individual team members don't feel as overwhelme d by the task	Project Man-ager
22	We may underestim ate how long it will take to do some work.	Medium	Work ends up taking longer than expected or not done to the standard it could be done. This could cause other areas of the project to suffer	Medium	Set realistic timings to do work and be realistic on how long a task will take. Account for unforeseen delays in our plan adding time where we can catch up.	Project Man-ager
23	Be too pessimisti c about what we can achieve.	Medium	We end up with a product which is not as good as it could have possibly been.	Low	Push our limits but also stay realistic and within the requiremen ts. If we have extra time then we can use it to enhance the product.	Project Man-ager
24	Distribute tasks incorrectl y.	Low	Team over/under worked.	Low	Distribute tasks appropriat ely and tell others if feel over/under worked.	Project Man-ager

Database risks

25	Database has glitches in it or text from it is assigned to the wrong character.	Medium	The wrong number of NPCs could be gener-ated or the questionin g system may not work causing us to not meet the requiremen ts for the assessment .	High	Thoroughly test the database and immediatel y fix any bugs we find. The database could be removed entirely and replaced with a simpler system with a lower chance of failure however this would not be possible in the time given.	Coding team
26	Game fails to load in data from the database.	Medium	The game will have issues with dialogue,c lues etc and may well not run at all.	High	Test to ensure data is being loaded in cor-rectly.	Coding team

Bibliography

[1] GitHub [online] Available https://github.com [Accessed 01/11/2016]

[2] Google Drive [online] Available https://www.google.com/drive/ [Accessed 01/11/2016]

[3] Cloud Flare [online] Available https://www.cloudflare.com/ [Accessed 01/11/2016]

Methods and Planning

Software Engineering Methods & Tools

Development Process

We are using an agile-scrum [0] approach for our software engineering project, as it allows our team to carry out work in a flexible manner, while providing the flexibility to evaluate our progress and plan for the next phase of work as we go along. Details of how we use this methodology to run our meetings and plan tasks are found in the team organisation section.

One of the key tools needed for any software engineering project is a version control system. This is a tool that will allow us to keep track of code changes, and collaborate on the same piece of code. We're using GitHub[1] for our code repository, as it's a popular tool that our team members are familiar with.

While developing, we plan to use git's branch and pull request functionality to help us manage multiple developers on the same project. To ensure our code quality is high, we ensure that a different developer performs a code review for every pull request, and that appropriate unit tests have been added and successfully pass before approving the branch to be merged.

We are using test driven development for our acceptance tests, as these are being written before development starts to ensure our code meets the requirements. While development is occurring, unit tests will be added to our code using JUnit[2]. This is so that we maintain a high level of test coverage, and have confidence that our code is working as intended. It will help with ensuring regressions do not occur. We are using CircleCI[3], a continuous integration server, to run all the unit tests every time a commit is made to GitHub, this will ensure that nobody accidentally breaks the codebase. For more details, please see the testing report.

To manage our development tasks, we plan to use the GitHub Projects feature of our code repository. This lets us manage our tasks and code issues with a Kanban board[4], and allows for items to be assigned to team members so everybody knows who's doing what. GitHub also allows us to link issues to the project, and these are useful because team members can collaborate on design decisions and keep track of what's happening in the code. More about our team organisation is discussed later.

Development Tooling

To develop the project, we decided, after much deliberation, to use Java[5] and IntelliJ IDEA[6] as our programming language and IDE. This is because Java is a language accessible to all members of the team, as well as the entirety of our cohort. We did give serious consideration to other programming languages such as C#[7] with Unity[8] however few in our team have any experience in the language which means it may take longer and cause problems later on in the process.

We needed a library that provides useful game APIs to help speed up our development process. We did some research and found 4 options of libraries we could use: Swing[9], Mini2Dx[10], LibGDX[11], LWJGL[12]. We quickly realised LWJGL and Swing weren't right for our project. In the end we decided to use the LibGDX Java game development framework. We chose LibGDX over Mini2DX because LibGDX has support for the IDEs that we intend to use, and it provides features that simplify several game development steps.

Design Processes

We will need to do some designing whilst the development stages are progressing. The main tool we will be using to design the game assets is Piskel[13], an online tool that allows us to easily create game assets. We expect designs to start as paper prototypes that will be developed further as the project goes on.

As we'd have to produce a map for our game (see requirement 4.1.1), we were pleased to discover that LibGDX has support for tiled maps. There is useful software called Tiled [14] that allows us to create a high quality 2D map and export it for use with the framework. This will save us massive amounts of time compared to alternative solutions, and this can be used to improve gameplay and ensure that there are few bugs.

Collaboration Tools

In addition to using GitHub projects as mentioned above, we also need some communication and collaboration tools to help us keep the project on track.

For the first few weeks we were using Facebook Messenger[15] for team communication, but it became apparent that it wasn't up to the job as we kept losing important pieces of information. We decided to switch to Slack [16] as it allows us to have separate chat channels for different aspects of the project, letting us keep all necessary information separated. Since separate discussions are split into separate channels, we are able to find important information more easily for future reference. We use this to consistently report our progress, ask questions on ambiguous issues and organise future in-person meetings.

We are using integration between our development tools and Slack to help us keep track of progress in one clear feed, in particular the GitHub integration and the CircleCI integration. This helps with our development method, as it ensures all team members are kept up to date with the status of the repository and test results.

For online meetings we are using Join.me [18] Join.me is a free tool with voice, text and video chat as well as a screen sharing feature which allows a presenter to share their screens with other team members. Join.me allows efficient and effective team meetings to take place even whilst all members are in separate locations. We made extensive use of Join.me over Christmas holiday period, where we used it to hold a full team meeting, using our scrum method as described later.

For document storage and collaboration, we are using Google Drive[19] with a shared team folder. This contains all of the documents and other files we've been working on. We are making use of Google Docs, Sheets & Forms[19] as they allow us to collaboratively work on documents at the same time, as well as providing mechanisms that allow us to review and comments on our documents.

Assessment 2 Review

At the beginning of Assessment 3, we reviewed our performance on the previous assessment, and in general we found we were happy with our performance. We were generally happy with our choices of development tooling and our software engineering methodology, we especially found our code reviews really helped with code quality and for catching bugs.

However, we identified that we struggled to keep our project management tools up to date, which meant that it was unclear who was working on what, and how much progress had been made. We then had a debate as to whether to start using Asana[17] instead of Slack and GitHub Projects, as Asana is able to perform the same tasks as the other tools do. This would allow us to minimise the number of tools we need to keep up to date, however we decided that the cost of switching outweighed the benefits, as the effort of transitioning the team to a new tool was considerable. As such, we have continued to use Slack and GitHub Projects for daily communication, and we have all being making an effort to keep the systems up to date so we can avoid the problems we faced last time. We have made sure that all tasks can be seen on GitHub Projects and issues exist for all tasks being worked on at any given time, whenever progress is made on a task or queries exist about it, comments are made on the issue relating to the task.

As in previous assessments, we continued using the SCRUM methodology and had 2 in-person meetings every week. We held some debates as to whether virtual meetings would be more appropriate due to most of us working on our own bits of implementation, keeping Github updated meant we were all kept up to date with the progress of each task so meetings weren't as necessary in some cases. Despite this we determined that weekly meetings were indeed required to have team reviews of all work done, this way any issues can be caught early. Meetings were also essential for the sake of deciding the future direction of the project and so that all members had the same vision of what we wanted the game to look like at the end.

Assessment 3 Methodology

Once we determined what project to take on board for this assessment, we held a meeting to discuss what we wanted the project to look like at the end of the assessment. This involved analysing the inherited code and determining our first steps to further develop the project.

As we analysed the code, we quickly realised that although it was well commented and the game worked well, the code itself wasn't very well organised. Classes and methods were often named incorrectly or in a confusing manner, files were not placed in a coherent and thought through file structure, and in many cases there were several classes worth of code jammed into a single class. We determined that our first priority was to refactor the code to make it more intelligible and whilst doing this, also to understand how each method works so that we know how to use it when developing future features.

In our first meeting, we analysed the brief and identified the requirements that had yet to implemented. We then split these requirements into work that needed to be done, and created individual tasks and issues to cover this work on GitHub. We then distributed these evenly amongst ourselves, we also had several unassigned tasks that could be picked up by any team member when they had completed their own work. We made a conscious effort in this assessment to keep GitHub as up to date as possible, and to carry out discussion in the comments of the appropriate issue - we found being stricter about our process helped with communication, and helped prevent problems we had run into previously.

We ensured all documents were updated as we went along to ensure everything was written about whilst fresh in our minds, this allowed the documents to be as accurate as possible. It also allowed us to have a better idea of how much work we had left to do since we knew exactly how much time documentation was taking instead of having to estimate how long it would take once we got to it.

Team Organisation

Roles

We decided early on to have a team leader role. The team leader is responsible for ensuring everybody has a task to be working on, and for making sure progress is being made. They are also responsible for producing meeting plans, and answering queries of any team member.

In Assessment 1, we voted to decide a group leader - Brooke Hatton won as we felt he was a natural leader. For Assessment 2, we decided that Jason Mashinchi would take over as group leader, as he had experience in a software development team. We plan to alternate group leaders throughout the assessments, to give the other member a break.

Every member of the team is assigned tasks to do at the end of every meeting for the sprint ahead. We decide who does what based on who wants to take on the task, or based on previous experience if it's relevant to a task.

Meetings

The team aims to have a meeting at least once a week, during which our team leader acts as our scrum master. We may not strictly follow the agile-scrum rules, but we've found that our approach works well for us.

Our team leader prepares for every meeting by producing a brief plan of what needs to happen during the meeting to ensure that we are making progress. Having a plan means that we can stay on track within the meeting and don't go off track and don't make progress. We tend to keep meetings high-level so that we don't waste time on implementation details that can be decided without the entire group present.

Meetings are scheduled to happen multiple times a week depending on availability of team members. Our meetings always signify the start of a new sprint. We typically start with each member going through what they've been working on in the previous sprint, and they raise any problems or questions they may have. This allows us to catch issues, concerns or blockers that have arisen early on in the meeting, so we can take them into account when planning the next sprint.

We then proceed to discuss what needs to be done during the next sprint, and assign tasks to each individual in the team. This is great because it means everybody has something to be working on for the next week, and ensures that we are making sufficient progress in the project.

During Sprints

Everybody is assigned a task to be working on during sprints, and they are responsible for ensuring their bit of work gets done. Every team member uses the kanban board containing issues on GitHub to keep the rest of the team up to date on where their task is at, and they make sure that their appropriate issues/tasks are assigned to themselves.

If any problems arise during the week, team members use their assigned GitHub issues to discuss anything related to their task, or we communicate through Slack when we need to discuss something more general. Splitting up the communication methods in this manner allows us to find discussion when necessary, as often we make design decisions within GitHub that can be referenced later.

Systematic Plan

Plan for Assessment 2

The focus of the team will fully switch to the second assessment on Wednesday 9th November, once the first assessment has been submitted. We plan on completing the second assessment by Tuesday Spring Week 2 giving us a week to account for any unexpected developments or fixing issues that happen to arise. It also gives us time to analyse, criticise and improve our own work thus improving the quality of our code and enhancing the functionality and efficiency. This will also ensure that we have very high quality documentation, once all group members are happy with the readability of our code we plan on requesting that students from other teams look at sections of our code to test readability. There will also be a 2 week rest from SEPR to account for time spent studying for exams and also the exam week itself.

Plan for Assessment 3

After the completion of Assessment 2 on Tuesday 24th January we will work as a team to decide the project to take on for Assessment 3. This will be done within one week, after which we shall decide upon areas that need to be worked upon over the next assessment, and delegate tasks accordingly to team members. Documentation is started in the first week with everyone focusing on making sure reports and code documentation are up to date. within the next week all documents will be near to completion and the code will have started to be updated. Again, we plan on completing the work a week before the deadline to give us time to fix any issues found.

Plan for Assessment 4

Once Assessment 3 is completed, we will decide on a new project to pick up as a team. Ideally, we will be able to choose one of the three updated iterations of the original game we submitted to assessment 2.

Once we begin assessment 4, we will meet to analyse the new requirements that have been assigned for this assessment and determine how this will affect the architecture of the game. Upon deciding what features need to be updated, the work will be divided up into tasks and these tasks will be assigned equally to all members.

After we have made a good start on the code, half of the team will start working on documentation so that we have all work done on time. We aim to keep our methods the same, since as a group we feel that our current methods work effectively. This means weekly scrum meetings will continue to be held, as well as continued use of GitHub projects. As can be seen in the Gantt chart (Appendix B), we aim to finish the assessment with a week to spare, so that we are prepared for any unexpected issues that may arise.

Gantt Chart

A Gantt chart [20] containing the schedule for key tasks for Assessment 4 can be found in the appendix. This chart includes priorities, task dependencies and a critical path.

Bibliography

[0] Waterfall to Agile: Flipping the Switch - Bhushan Gupta [Online] Available: http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf [Accessed 25/10/2016]

[1] Github [Online] www.github.com [Accessed 8/11/2016]

[2] JUnit [Online] http://junit.org [Accessed 22/01/2017]

[3] CircleCI[Online] https://circleci.com/ [Accessed 22/01/2017]

[4] Kanban board [Online] www.github.com [Accessed 8/11/2016]

[5] Java [Online] www.oracle.com [Accessed 8/11/2016]

[6] IntelliJ [Online] www.jetbrains.com [Accessed 8/11/2016]

[7] C# [Online] www.msdn.microsoft.com [Accessed 8/11/2016]

[8] Unity [Online] www.unity.com [Accessed 8/11/2016]

[9] Swing [Online] www.oracle.com [Accessed 8/11/2016]

[10] Mini2DX [Online] www.mini2dx.org [Accessed 8/11/2016]

[11] libGDX [Online] www.libgdx.badlogicgames.com [Accessed 8/11/2016]

[12] LWJGL [Online] www.lwjgl.org [Accessed 8/11/2016]

[13] Piskel [Online] www.piskelapp.com [Acessed 8/11/2016]

[14] Tiled [Online] www.mapeditor.org [Accessed 8/11/2016]

[15] Facebook Messenger [Online] https://en-gb.messenger.com/ [Accessed 23/01/2017]

[16] Slack [Online] www.slack.com [Accessed 8/11/2016]

[17] Asana [Online] https://asana.com/ [Accessed 01/02/2017]

[18] Join.me [Online] www.join.me [Accessed 8/11/2016]

[19] Google Drive [Online] https://www.google.com/drive/ [Accessed 23/01/2017]

[20] Gantt Chart [Online] http://lihq.me/Downloads/Assessment3/AppendixB.pdf

Appendix: Task Assignment Summary

Assessment 1:

- Ben Jarvis was assigned the requirements document
- Benjamin Grahamslaw was assigned the risks and mitigations report
- Brooke Hatton and Jason Mashinchi were assigned the Architecture report
- Joseph Shufflebotham and Vishal Soomaney were assigned the Methods report

Assessment 2:

- Benjamin Grahamslaw was assigned the GUI report & user manual
- Brooke Hatton, Jason Mashinchi & Joe Shufflebotham were responsible for the implementation of the game, testing and the design choices made.
- Other team members contributed towards implementing smaller features in the game
- Jason Mashinchi was assigned the testing report
- Brooke Hatton & Vishal Soomaney were assigned the architecture report
- Ben Jarvis & Benjamin Grahamslaw were assigned the document updates
- Joe Shufflebotham & Vishal Soomaney were assigned the implementation report

Assessment 3:

For Assessment 3, we didn't assign documents to each individual in the same way we did in previous assessments. As well as contributing towards the code, we found it was more effective if we all contributed to documents when we found something that needed to be changed. The list below was created using the issues we had completed and assigned to ourselves on GitHub.

Benjamin Grahamslaw

- Updated user manual
- Updated GUI report
- Wrote introduction and some sections of change report
- Implemented scoring with Ben J
- Created character dialogue with Ben J
- Contributed towards implementation report

Ben Jarvis

- Reviewed and updated risks
- Reviewed and updated requirements
- Updated acceptance and unit test documents
- Implemented scoring with Ben G
- Created character dialogue with Ben G

Brooke Hatton

- Performed an extensive review of the architecture and planned refactoring
- Refactored maps & rooms
- Refactored dialogue system
- Improved database structure
- Added test coverage
- Contributed towards implementation report

Jason Mashinchi

- Updated testing report
- Added StatusBar
- Added personality meter
- Refactored JournalScreen
- Refactored InterviewScreen
- Updated methods report, updated Gantt chart
- Contributed towards implementation report
- Updated website and Read the docs

Joe Shufflebotham

- Implemented player movement in game
- Added NarratorScreen
- Implemented clues, motive clues and means clues
- Added Suspects to the map
- Contributed towards implementation report

Vishal Soomaney

- Implemented MainMenuScreen
- Updated acceptance tests
- Updated requirements
- Updated methods report
- Contributed towards implementation report

Implementation Report

Introduction

The code we inherited from Team Farce fully implemented all the requirements for Assessment 2, however, there were several additional requirements for Assessment 3 that we had to implement, which required some architecture changes.

Once we had reviewed the code of the inherited game, we determined we needed to perform some extensive refactoring to simplify the code and game logic, as well as extending the game to meet the requirements. This document contains a detailed list and justification of the major changes below. Some changes were made to the project that aren't listed

- these include improved commenting and minor additions or changes with minimal impact, these can be viewed by visiting our repository [1]. The refactoring work is most evident by comparing the UML diagrams before and after the work completed on this assessment. To see the architecture changes described below in terms of a UML class diagram, please refer to our current UML diagram [2] and the original diagram that we inherited in this assessment [3]. We once again used IntelliJ to produce the UML class diagram, this allowed for the diagram to be completely accurate since there was no room for external software or manual errors. This means that there was no chance of having any traceability issues and saved us a lot of time that would have otherwise been spent creating and checking the diagram.

All of the required features for Assessment 3 from the requirements have been fully implemented.

New features & changes to previous software

Throughout our documents we reference to the requirements[4] using [x.x.x] with 'x.x.x' matching a requirement in the requirements table. We have outlined significant changes in bold, and added more details underneath.

In this document we refer to non-playable characters as NPC's and Suspects using these terms interchangeably.

Change list & justification

Refactor GUI into screens

- We decided it would be useful to refactor out all the GUI elements within the game into smaller modules using LibGDX's Screen class. Doing this allows us to improve the maintainability of the game, and speed up development as each GUI screen are now, smaller independent classes.
- To do this we first split up a large render method in the root MIRCH class of the game, which we found tough to work with as it was long and full of duplicate code, into separate LibGdx's screens keeping the relevant logic together.
- We introduced an AbstractScreen class to our architecture, which is extended by all game screens containing methods and properties that will be used by the other screens further improving the abstraction of our architecture.
- We removed the DisplayController class and heavily restructured the GUIController class to use LibGDX's screens, a simpler approach to handling the various points of the game.
- The GUIController now changes screens depending on GameState, and contains an update method that is called whenever the screen is refreshed.
- We added or fixed appropriate unit tests for all new or modified classes

Removed RenderItem class

- The original purpose of a RenderItem was to group together objects and sprites for rendering on screen, however we found this was inefficient. To improve this we have made all objects that should be rendered on the screen extend the LibGDX sprite class. The main benefit is now we only have to update one object rather than two improving the game logic. This changed the architecture significantly, it lost most of the previous groups split of 'back end' and 'front end' architecture which we didn't find to be a good fit for the game as it was all running in the same application. We felt that although we lost the clear difference, it was a large improvement in the code both for maintainability as we only have to worry about one object rather than two for each displayed item.

Restructured the Player and Suspect class [3.1.2]

- The Player code has been moved from the MIRCH class to its own class Player. This helped simplify the large MIRCH class and helped separate concerns to improve maintainability and improve our architecture.
- The Player and the Suspect class now extend a new AbstractPerson class - this was done as the Player class shares a considerable amount of code with the Suspect class, thus adding abstraction.

- The AbstractPerson class inherits from the existing MapEntity class, to further increase abstraction.
- There are 10 non-player characters, 9 of these are alive in the game at any one time as one of them is randomly chosen to be the victim. This meets the requirement on the number of non-player characters the game must have. [3.1.2]

Replaced map system

- Part of our requirements [4.1.1] were to have 10 rooms that were accessible in the game. The system we inherited randomly generated the map from a series of room templates.
- We felt a substantial amount of refactoring was necessary as the implemented system had a number of problems which mean it often failed to meet the requirements:
- Some rooms weren't always accessible due to random map generation.
- Collision system was unreliable, due to this NPCs often escaped or spawned outside of the map.
- After evaluating our options, we decided the quickest and easiest solution was to replace the map system with the one we had already built for our previous project, as this was already fully working and very easy to work with.
- This refactoring involved bringing over our Map and Room classes, along with their dependencies, including the Vector2Int class.

Improve handling of movement and input [1.1.2] [2.1.4]

- We brought over our PlayerController class. This uses events to trigger the movement of the player, which let us remove the input and movement out of the large Mirch class.
- This helped with improving the efficiency of the code because we are now using event handlers rather than polling the keyboard and mouse for changes on every render. This helps ensure that the game runs well on any computer [1.1.4].
- We defined an abstract move method in the AbstractPerson class and implemented it in the Player class, as well as a method in the Suspect class to randomly move the NPC. Previously to move the player three different methods were being called: one to check the input, another to scale the input, and one to move the player by the scaled input, whereas now this is all contained within the Player's move method, so only one method needs to be called to move the player.
- Added the ability to move the Player [2.1.4] by clicking the mouse, using an implementation of A* search to ensure the Player avoids all obstacles. This was implemented because we felt it fit better with the user interaction model used elsewhere, with the mouse as the primary input device.

Added StatusBar [2.2.1]

- Added a StatusBar class with the GUI refactoring. This was a simple addition to the architecture and it helped increase abstraction within our code. Previously each screen handled its own navigation, however this is no longer necessary as each screen in the game uses the same status bar.
- The status bar contains the navigation buttons that were previously placed on screen near the top, as well as the player's score [6.1.1] and player personality [2.2.1].

JournalScreen refactoring

- The JournalGUIController class was refactored into the JournalScreen with the GUI refactoring work
- The journal was extensively refactored to improve the readability of the code, and abstract away appropriate duplicate code.
- The journal GameStates were simplified. The unnecessary "journalHome" GameState was removed, as it originally linked to the journal navigation, which required an extra step to see useful content. We felt this was an unnecessary step as it slowed game play, so we replaced it with "journalClues" which links directly to a useful (clues) page in the journal.

- Added two public methods to the Journal class so they can be accessed by the JournalScreen.
- The journal screen provides lists of found clues, previous conversations and a notepad.

Added scoring [6.1.1] [6.1.2] [6.1.3] [6.1.4] [6.1.5]

- We added a score property to the GameSnapshot, with two getters and setters. We put it in the GameSnapshot because it can be accessed throughout the game with minimal code changes.
- The score changes throughout the game (via the modifyScore() method)
- The score increases when the player finds clues [6.1.5], or correctly accuses a NPC.
- The score decreases when the player asks questions to the NPCs [6.1.4], and a large score is lost for a wrong accusation [6.1.3]. The player's score also decreases by 1 every 5 seconds [6.1.2] to simulate the importance of speed during an investigation.

Added NarratorScreen

- This screen was added to inform the player about game progress - such as the introduction to the game, and the response for winning or losing the game.
- We added this to provide useful feedback and help the player along the game.
- It features our team's mascot, Sir Heslington the duck, who will say a speech to the player. The speech can be set using methods included in the screen.

Database changes

- We felt the need to simplify the database we inherited with the codebase as we felt the database design was too complicated for the problem it was trying to solve, and was hard to expand upon. When we tried to change things like increasing the number of NPC's to meet the requirement [3.1.2], this cause the game to fail.
- 18 tables were removed due to our refactoring to make the code simpler:

Character_costume_links, Character_means_links, Character_motive_links, Clue_means_requirements, Clue_motive_requirements, Clue_murder_requirements, Clue_victim_requirements, Costumes, Dialogue_text_screens, Follow_up_questions, Potential_prop_instances, Prop_clue_implication, Protoprops, Question_and_responses, Question_intentions, Response_clue_implication, Room_templates, Room_types

- 1 table was added: Character_clues, used for many-to-many relationship between characters and clues for the scenario generation process in the game.
- Modifying the database in this way has helped simplify the game logic, maintainability and understanding of how the game works. This helped us expand the game and meet the requirements of the project faster.
- The dialogue was moved from the database to static json files, more details about this can be found in the dialogue refactoring section of this document.

Dialogue refactoring [7.1.2], [7.1.5], [7.1.6] [2.1.1]

- Removed the dialogue related tables in the database, these were replaced with json files. This decision was taken because the inherited implementation was broken and required extensive refactoring, and we found it quicker to use our prior code for handling dialogue with json files
- We removed these classes: AggregateDialogueTreeAdder, DialogueTree, IDialogueTreeAdder, NullDialogueTreeAdder, QuestionAndResponse, QuestionIntent, QuestionResult, DialogueOption and SingleDialogueTreeAdder. Doing this simplified the architecture further.
- Two new classes were added, Dialogue and InterviewScreen. These are explained further below.
- The Dialogue class parses and verifies the JSON files containing the dialogue content
- Each Person (Suspect or Player) has a Dialogue object which is used in the InterviewScreen to get the relevant dialogue. It provides different styles of questioning for the user to select from [7.1.5].

- For the Suspect the dialogue file also controls how the suspect should respond dependant on the style of questioning [7.1.6].
- Player Personality was added with this work, this is stored as in integer in the GameSnapshot class
- The personality is a value between -10 and 10, providing a scale between very aggressive, and very polite. The player can be anywhere in between. If the player being is too aggressive, or too polite they cannot use dialogue for the other extreme until they have brought their personality back to a neutral level. This means that the personality is dynamic and customisable by the player [2.1.1]

InterviewScreen refactoring [7.1.1], [7.1.2], [7.1.3], [7.1.4]

- The InterviewGUIController class was refactored into the InterviewScreen class due to the GUI refactoring into screens. This improved the structure of the code and allowed us to expand the game further to meet the various interview based requirements.
- Some of the GameStates were changed to reflect the changes to the dialogue system, and these states are implemented by the InterviewScreen class
- InterviewResponseBox and InterviewResponseButton GUI elements were added to the project, these are used for adding response [7.1.3], accuse [7.1.4] and question [7.1.2] buttons for the player to select from during an Interview. These elements are self contained, and do not contain any game logic in order to separate concerns. Event handlers are handled with the initialising code, which makes it easy to maintain.

Adding Clues [5.1.1] [5.1.3]

- With the restructuring of the map, clues are hidden in the hiding locations defined in each of the room files. This simplifies the architecture as it allows us to remove the Prop class, lots of additional code and database tables that were previously used to define possible hiding locations and clues.
- Each possible killer has 5 clues that point to them, the killer is selected at random when the game starts and their clues are added to the map.
- There is one ‘easter egg’ clue that doesn’t provide any help.
- A means clue (a weapon) is selected from a subset of the clues.
- There is a set of motive clues, one [5.1.3] of these is selected randomly and split into 3 separate clue objects. This adds 3 to the total amount of clues that have to be found in the game, allowing us to meet the requirement of having at least 10 clues in the game. Since we have 10 rooms as well, this allows us to have at least one clue per room. [5.1.1]
- All 10 clues are distributed randomly into one of the many hiding locations in each of the rooms.
- Added FindCluesScreen. This screen is displayed when the player finds a clue in the map. It displays the found clue, along with any relevant information. When the user presses continue, the sprite of the clue spins and flies toward the “Journal” tab on the StatusBar. This is to provide a hint towards clicking on the “Journal” tab to view found clues.

Main Menu [1.1.1]

- The MainMenuScreen has been added, it allows the user to start the game as well as quit the game.
- Aside from the addition of the MainMenuScreen class, no other architecture change was necessary for the implementation of the main menu.

Bibliography

[1] Our code repository [Online] Available: https://github.com/Brookke/li-mirch [Accessed: 20/02/17]

[2] Current team Lorem Ipsum UML class diagram [Online] Available: http://lihq.me/Downloads/Assessment3/CurrentUML.png [Accessed: 20/02/17]

[3] Original team Farce UML class diagram [Online] Available: http://lihq.me/Downloads/Assessment3/OriginalUML.png [Accessed: 20/02/17]

[4] Link to updated Requirements document [Online] Available: http://lihq.me/Downloads/Assessment3/Req3.pdf [Accessed: 20/02/17]

Change Report

Change management

The first thing we did as a team was to have a good look at the code and deliverables we had inherited from Team Farce. We wanted to ensure we were familiar with the documentation and code, so we could make informed decisions on what changes we needed to make.

We were aware that we would need to mainly do two types of change:

- Corrective changes: We discovered some faults and errors within the codebase which we tried to fix.
- Additive changes: To complete Assessment 3 we needed to add a number of features in order to complete the game to the requirements specification.

We also needed to be aware of two other forms of change we may need:

- Deletive changes: We discovered some features of the other teams reports and code that did not work or were buggy. This especially caused problems when trying to use their methods and classes to implement new features.
- Perfective changes: We found that the games architecture was, in some places, badly organised and corrected this with extensive refactoring in order to make working with the code easier for ourselves and potentially others. This work is described in the implementation report, along with other code changes.

As a team we evaluated the advantages and disadvantages of the code and documents, then we used what we had learned to help decide which documents to work with and what changes needed to be made. We decided to keep all of our documents, and adapt them to suit the new project as we felt this was achievable as we were already very familiar with our documentation. This choice of documentation also made our lives easier when it came to traceability.

We also looked at the code and decided which of the change types mentioned above were required for each part. We analysed our requirements and looked for any changes that needed to be made, including adding in requirements related to the features that were already implemented in the game that we inherited. We moved on to implementation, where some parts of the code were kept the same, some parts were changed slightly, and some parts of the code were completely changed.

After updating our requirements, we began development on the changes and additions that needed to be made. We tried to ensure traceability of requirements both forwards and backwards. It is important to ensure it is clear where new requirements came from, but also to be clear where these new requirements are used in the other documents, where appropriate.

Our overall focus for change management was to ensure we used the best of what we had to use. If what we were given was good we kept it, otherwise we changed or replaced it. We wanted to make the very best game that we could that would fit the requirements, and we were not afraid of making big changes in order to achieve this goal. In our documents new changes were shown using a blue font to make it obvious.

Report updates

GUI report

This document was modified with the following changes:

- Added descriptions for new GUI elements implemented in the game or inherited from Team Farce. We looked at which elements of the GUI we had brought over from our original GUI and also look at the new elements that have been added, such as the journal.
- The report now describes new screens which were added this assessment, the clue found screen and the narrator screen, these are important elements of the finished game so needed to be included in the GUI report.
- Updated descriptions of GUI elements that we brought over from our game
- Expanded content about how user interaction works with the GUI. We felt that our current GUI report did not effectively communicate this, so it was something that we tried to incorporate within the new GUI report. User interaction is an important factor in GUI design that we should not have missed first time round.
- Added a description of our thought process and the factors we considered when designing the GUI

URL:

- http://docs3.lihq.me/en/latest/Assessment3/gui.html or
- http://lihq.me/Downloads/Assessment3/GUI3.pdf

Testing report

We continued to use the testing document our team created for Assessment 2. This document was modified with the following changes:

- Update references from Assessment 2 to 3
- Update test statistics (number of tests, success rate, number of failures)
- Updated links for Assessment 3
- Added paragraph (2nd from top) explaining how tests work in our project (originally from Team Farce), and how we've adapted and improved their testing methodologies to better fit our workflow. This includes setting up CircleCI for continuous integration. This paragraph also mentions that we kept our own acceptance tests as we also kept our requirements, as well as mentioning that we used and extended upon the unit tests provided with the project we took on.
- Unit test report (see Appendix E) has been updated to reflect the new project and any changes made to the tests. The test results have been added. All unit tests (both added and inherited from Team Farce) have been linked to our requirements, for traceability.
- Acceptance test report (see Appendix D) has been updated to reflect the new project choice, and the results have been updated to reflect upon the latest version of the game. We felt that the acceptance tests weren't specific enough so we updated them to make them more specific, this ensures that the tests can be perfectly replicated. We also tried to remove ambiguous terms such as the word "interact" when referring to picking up clues or speaking to suspects.
- Added reference sources for testing summary, these are placed in the new bibliography
- Added section about automated test coverage, as we started using JaCoCo

URL:

- http://docs3.lihq.me/en/latest/Assessment3/testing.html or
- http://lihq.me/Downloads/Assessment3/Test3.pdf

Methods and plans

This document was modified with the following changes:

- Added a description of a discussion we had about team management tools. We considered switching to Asana which is a tool similar to GitHub projects to show tasks that need to be done and have discussions, however we felt that it was best to carry on with GitHub projects as it is where our code is and that should ensure we all look at it and keep it up to date.
- Added a description of a debate held about in-person meeting frequency and necessity.
- Described the approach we intend to have when selecting a project for Assessment 4, we will hopefully select one of the three teams that chose our game in Assessment 3
- Added our current thoughts on what we will need to do when we are given new requirements and how we will tackle them, regardless of what they are
- Describes potential task distribution and continuing methods
- Added a detailed description of our methods used in this assessment, including dealing with the new code and the assignment of tasks. We felt it was important that our methods document described the methods used in this assessment, as it gives a good insight into how we worked as a team to tackle this assessment.
- Added a Gantt chart for assessment four to the document to show how we will split up our time for the next assessment
- Updated the appendix which includes the task assignment summary. Each team member was given tasks for Assessment 3

URL:

- http://docs3.lihq.me/en/latest/Assessment3/methods.html or
- http://lihq.me/Downloads/Assessment3/Plan3.pdf

URL for updated plan: http://lihq.me/Downloads/Assessment3/AppendixB.pdf

Risk assessment and mitigation

The risk management document did not change very much, as we were happy with the report from last assessment. Previously our approach and presentation was heavily reworked as described in the previous assessment to include risk ownership.

We looked at the risk management document given to us by the other team but did not really find anything we wanted to add to our own document.

As a team we were very happy with the risk management document that we had and it’s format so we decided to keep it. However, we modified this document with the following changes:

- Added database risks to the risk table. When we took on this new project we inherited a new kind of software that we had not previously been using. The other team heavily used a large database, and the use of this new software presented new risks that needed to be looked at. Therefore we discussed these risks as a team and added them into our own document to make the risks specification complete with our new game.
- One database risk is to do with incorrect data being saved into the database, this could lead to unexpected scenarios when running the game.
- Another risk is that data is not loaded into our game at all, which could cause the game to not run and would be a major problem with how the game is implemented in this assessment.
- Details and mitigation for both of these risks can be found in the updated document.

URL:

- http://docs3.lihq.me/en/latest/Assessment3/riskAssessmentAndMitigation.html or
- http://lihq.me/Downloads/Assessment3/Risk3.pdf

GUI Report

Design processes

Our design process for the game was based on some simple principles, we wanted to make sure that all GUI elements in the game were clear, intuitive and easy to use. As a team we were clear that we wanted the GUI design to be centered around the user and making sure that a user would find it enjoyable to use, this in turn would hopefully lead to an enjoyable experience playing the game which is another high priority for us as a team. This is a game so it should be fun. We also aimed to have a game that has a high level of usability. One example would be the click to move option which uses an A* search algorithm and makes movement very usable.

Player interactions

We designed the user interaction in the game using the principles designed above. We were focused on usability, so we took the decision to use mouse control for all user interactions. This was because we felt the mouse is the most common and familiar method of interaction with a computer and will be obvious to the user. It also removes the need for a mental transition between keyboard control and mouse control, as jumping between the two can be jarring, as we found in the last assessment.

At first we thought it was best to interact with clues and NPC's by walking up and pressing the Enter key, however we realised that this was not intuitive, not to mention walking across rooms was boring and also could have an unfair effect on the player's score. For this reason we switched to just clicking on the clue or suspect instead, as this option addresses these issues.

Main menu

The main menu will be the first screen that a player will see upon loading up the game. It will provide the user with the following options:

- **New game button** - this starts the game
- **Exit button** - this closes the game

The buttons are all designed in the same way throughout the game. They are designed to be easy to distinguish from the background and for all text displayed on them to be easy to read.

Related requirements: 1.1.1 **Realisation**: Appendix F:1

Main navigation screen

The main navigation screen contains two elements - a map and a status bar.

While the player is playing the game they will be able to see their character on the map and move around from room to room. The character is displayed in the middle of the map for good playability. This screen also has a status bar overlay at the top which allows the player to switch between the map and journal screen, and also shows the player's current score and personality.

When you click on a clue, the clue is shown and described. When you click on a detective the screen changes to the dialogue screen.

Related requirements: 2.1.4,2.2.1,3,4(Map,clues and BPC sections) **Realisation**: Appendix F:2

Dialogue screen

The dialogue screen allows communication with suspects/NPCs. The GUI contains an image of the suspect and contains the interview flow. This involves the player first selecting whether to question, accuse or ignore the suspect, followed buttons to select a clue to question the suspect about, and dialogue style choices.

Related requirements: See dialogue sections of requirements. **Realisation**: Appendix F:3

Journal

The Journal is a collection of information that the player has obtained so far in the game. It is layed out like a notebook and contains buttons that allow the player to view clues, see the interview log of conversations and write in a notepad. It contains the following sections:

- Clues list - shows the player the clues that they have collected so far
- Conversation history - provides a list of the interview dialogue that have occurred with suspects so far in the game
- Notepad - allows the player to enter any notes they feel appropriate

Related Requirements:5.2.1, 5.2.2 **Realisation**: Appendix F:4

Find Clue Screen

When the player clicks on a clue within the map, the find clue screen is shown. It has a large graphic of the clue that has been found, and on the right hand side is a text box describing the clue. The name of the clue is at the top and there is a button at the bottom to continue and go back to the map, the map can be seen in the background.

This screen is used to inform the player about the clue they have found and ensures they are aware of the implications of it. We use an animation to "fly out" the clue to the journal, this is intended to increase user awareness of the journal so that they can use it to guide their thought processes when deducting the killer in the game.

Related requirements:5.1.2 **Realisation**: Appendix F:5

Narrator Screen

The narrator screen has a narrator character, Sir Heslington (the duck), with a speech box containing the text which he is saying. This screen is used when you first load up the game, and the speech box text is used to explain the premise. It is also used when you have found all 3 parts of the motive clue. Finally it is used for when the player wins or loses.

Related requirements:7.1.7,8.1.1,8.1.2 **Realisation**: Appendix F:6

Testing

Methods & Approaches

We decided to take a requirements focused view of testing, so that our tests are built around making sure the requirements were being met for the project, but also making sure that we tested key parts of our code that may not directly correspond to any requirement. We decided to use two types of testing, these can be seen below:

As we chose a Java project for Assessment 3, no changes to the testing approaches or methodology were made. The project we chose uses JUnit unit tests, like our original project, the only difference is that we enabled CircleCI automated testing. We decided to keep our original testing report, and adapt it where necessary. We have also kept most of our acceptance tests, as we felt they were still applicable and are appropriately linked to our requirements (which we decided to keep as the brief hasn't changed). The unit tests were taken from the project we inherited, and when needed we continuously added to, modified or removed these tests as we developed the project.

Acceptance Testing: this allows us to ensure the end product meets the requirements, and that a user can perform common scenarios successfully.

- Each test takes the form of a list of steps to carry out in the game.
- If a user can perform the list of steps successfully, the test passes, else it fails.
- These tests have to be run manually, because it would take too long to automate them and we have time constraints on the project.
- They are good for testing both functional and nonfunctional requirements.
- Can test functional requirements as the tests can confirm whether the game has the appropriate functionality to meet the requirement.
- Can test non-functional requirements as tests can confirm the project behaves as expected.
- Allows us to detect regressions during refactoring or other code changes.
- We use a **test-driven development** approach here, where the acceptance tests were written before the code, and more tests should pass as more of the requirements are implemented [1].

Unit Testing: this allows us to check that our code does what it is meant to.

- Takes into account both normal and edge cases to ensure normal behaviour and boundaries are handled correctly.
- The unit tests are written as the code is being implemented, due to their reliance on the code structure and design decisions.
- We have used both black box testing (doesn't take into account how code works) and white box testing (uses inner workings of code) when designing our unit tests.
- Allows us to detect regressions during refactoring or other code changes
- Written as JUnit unit tests in Java, because it is compatible with our project, easy to use, and has high quality documentation [2].
- These tests are automated, so whenever a commit is pushed to GitHub, the test script is run on CircleCI (our continuous integration server).
- We chose CircleCI for it's reporting abilities, as it provides a html website, and breakdowns of failing tests with debug logs where applicable, making it easy to diagnose problems.
- After the CI runs the unit tests, the result is pushed back to GitHub and updates our team Slack. GitHub is setup to block merging pull requests if a unit test fails.
- Continuous integration helps remove much of the effort required to run tests, and ensures that unit tests are taken into account when developing the project [3].
- These tests also alert us if the project fails to build with each commit, as the project is built by the CI server in order to run unit tests. This can let us know about possible code problems, like syntax errors.

Our software development process involves creating new branches for new features, and when a branch is ready, a pull request is created to merge the branch into the master branch. We've setup our pull request flow to block merging if the unit tests have failed, which helps us catch code issues sooner rather than later. Even within a couple of days of starting programming, the testing checks had saved us from merging code that looked good, but was in fact broken in a non-obvious way.

For Assessment 3, we added automated test coverage, using JaCoCo, which determines how well unit tested our project is. JaCoCo is an open source toolkit for measuring Java code coverage, which indicates how much of the code is tested [4]. This has been helpful for identifying which parts of the game are lacking in tests, and we've used the associated data to improve our test coverage. We run test coverage alongside our JUnit tests on CircleCI so that we always have a current statistic that we can use.

Bibliography

[1] AgileData.org - Test Driven Development [Online] http://agiledata.org/essays/tdd.html [Accessed 16/02/2017]

[2] JUnit [Online] http://junit.org [Accessed 16/02/2017]

[3] Thoughtworks.com - Continuous integration [Online] https://www.thoughtworks.com/continuous-integration [Accessed 16/02/2017]

[4] JaCoCo [Online] http://www.jacoco.org/jacoco/ [Accessed 16/02/2017]

Test Report

Overall Statistics

- **67 tests**
- **0 failures**
- **100% successful**

Unit Tests

Below are the statistics generated by our unit test runner on CircleCI for our Assessment 3 code. These tests are functions within the code (although they don't get compiled into the final executable) that tests a small unit of code at a time. They are fully automated, and were automatically run when the appropriate commit was made to GitHub.

- **42 tests**
- **0 failures**
- **100% successful**

All of the unit tests in the project have passed, which indicates the absence of bugs in the tested code sections. More comments on what this result means are on the next page. The unit tests have test IDs indicated by 1, followed by their index in the list.

See Appendix E for a full listing of the unit tests.

Acceptance Tests

We have manually run through our acceptance tests to check the game works as intended, and to verify that the game fulfils the requirements. The results are shown below.

- **25 tests**
- **0 failures**
- **100% successful**

All of the acceptance tests passed as all of the requirements have been implemented. More comments on what this result means are on the next page. The acceptance tests have test IDs indicated by 2, followed by their index in the list.

See Appendix D for a full listing of the acceptance tests.

Test Information

The tests are associated with an appropriate requirement to allow for traceability, and to check that the code meets any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly tested, and some tests do not link directly to a requirement but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly work as intended.

Results & Evaluation

The acceptance and unit tests for this project all passed.

The test completeness is not perfect, nor is it possible to be. The unit tests only check their section of the code works as intended, and doesn't cover the integration of that code, or it's use within the entire project. What the unit tests do indicate, is that the specific code that they test works as intended, and they indicate the absence of bugs in that specific code. When combined with the acceptance tests, the test completeness is improved (but still not perfect) as not only are key functions of the code tested, but the overall product is tested to ensure it meets the requirements.

This project doesn't have perfect test correctness either. The unit tests could have bugs in them meaning bugs could be missed in the code, or the unit tests may not cover every edge case or normal use case that exists, which means bugs and issues could slip by undetected. The same sort of thing happens with acceptance testing, as typically a limited range of scenarios are tested, which may not account for the all of the very many possible ways of using the game. Also, the acceptance tests included in this project need to be run manually, which means that human error could occur and affect the overall correctness of the tests.

To improve our testing **completeness** and **correctness**, additional types of testing would be useful additions, such as fully automated end-to-end testing, or integration testing, along with more tests of any type. Introducing different types of tests would alert developers about the presence of a different kind of bug, which would allow identification of bugs that have previously been uncaught. Adding more tests would also decrease the chances of code regressions being missed, or other issues or bugs being missed during the testing process.

Executable Test Plan

Unit Tests

This project is unit tested with JUnit. Tests are located in the `/game/tests` directory in the GitHub repository. For documentation on writing these tests, please see https://github.com/junit-team/junit4/wiki

Coverage

The project includes Jacoco coverage, which checks to see how well tested our classes are, the higher the coverage the better. This is ran automatically by CircleCI and availible under the "Artifacts" tab.

Running Unit Tests

For every commit CircleCI runs all the included tests as well as the coverage of these tests, however, we recommend that you run tests locally too before committing.

We have included a handy test configuration inside the repository that can be run from IntelliJ IDEA.

Adding Tests

- Create new class for tests under `/game/tests/src` When naming the class end the name with `UnitTests` for consistency e.g. `PlayerUnitTests`
- This class should extend `GameTester` this initialises the backend of the game so that tests run correctly.
- Import `org.junit.Test`
- Write a test function using assertions, and use `@Test` decorator above it
- See this page for examples of assertions: https://github.com/junit-team/junit4/wiki/assertions
- Run your tests locally and see if they pass!

CircleCI

Viewing Results

After tests have run the results are displayed in the "Test Summary" tab on CircleCI. This tab contains a summary of the testing result, along with any tests that have failed.

If the tests have failed and no test summary is provided, this normally means that the code doesn't compile, or there is a problem with the test code. To gather more information, scroll down to read the console output from when the tests were run.

Also, CircleCI collects test "artifacts", which are located in the "Artifacts" tab. This contains useful output files such as:

- HTML output: A website that provides a visual testing report with more details
- JUnit XML output: XML files for each class that provide raw data about each run test
- Jacoco Coverage Results: These contain a breakdown of the coverage for each of our classes.

Configuration

We have included a configuration file for setting up CircleCI tests in the root directory of the project. See `circle.yml`.

To setup CircleCI, you will need to link your GitHub account. Once this is done, you can add a project within CircleCI, which will automatically setup tests using the configuration file (`circle.yml`) mentioned earlier.

Whenever a commit is pushed to GitHub, CircleCI will run the tests and inform GitHub of the result, which will be displayed against each commit, and in any pull requests.

Acceptance Tests

We've also included a bunch of acceptance tests that can be run manually to ensure the game performs as expected, and meets the requirements. These have not been automated, so will need to be run by hand. These can be found in the Appendix for the Assessment 2 documents.

A: User Scenarios

Persona 1

David, 18 year old male at a Computer Science Open Day. He is not an avid gamer and his interests lie more in electronics.

Scenario

David is initially unsure as to how the game is played, however he is provided a tutorial by the narrator which explains the puzzle, how he is expected to solve the puzzle, how to move the main character, interact with clues and non player characters. After becoming accustomed to the keys and finding a clue, he is unsure as to what the next step is to solve the puzzle. He interacts with the narrator who subsequently provides hints as to the next course of action that will get the user closer to solving the puzzle. Near the end of the game he becomes stuck and spends a long time attempting to find out which of his 3 final suspects is the murderer, the narrator pops up on the side of the screen to ask whether the player requires assistance, David says "yes" and is told to speak to a character which holds the final piece of information required to solve the puzzle.

Persona 2

Louise, 17 year old female student at an open day. Enjoys playing puzzle games and is colour-blind.

Scenario

Although she is initially worried about missing a clue or finding it hard to read some of the text, she is able to go through the introductory tutorial without issue due to the text colour and background colour contrast in a way to avoid issue with most forms of colourblindness. She is able to see the clues quite clearly due to map tile colours and clue item colours being chosen in a way so as to not cause problems to most people will colourblindness. She is then able to solve the crime quicker than most and accuse the correct murderer of the crime on her first try thanks to her experience played puzzle games.

Persona 3

Ben, 19 year old second year Computer Science student. He is an avid gamer who takes games quite seriously and always tries to get the high score.

Scenario

He progresses through the tutorial quickly and is already accustomed to using keyboard keys to interact with the game. As he attempts to concentrate and solve the puzzle as quickly as possible so as to get the high score, the background noise begins to bother him. Fortunately he notices that there is a "turn music off" button in the sidebar of the game and this allows him to continue playing comfortably. Although he finishes the game with a good score, he is unable to beat the high score and wishes to play again, fortunately due to the game having a set of different characters from which murderers and victims are chosen randomly, he can play several times whilst going through a new, different and exciting storyline each time.

B: Project Plan (Gantt Chart)

We've produced a systematic plan for Assessment 4. This is in the format of a Gantt chart, created in Excel.

Download (AppendixB.pdf)

C: Survey Results

Figure 1

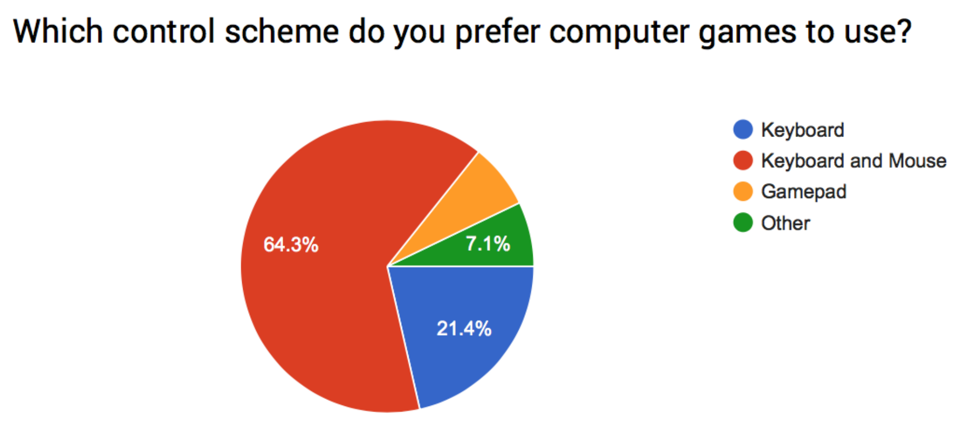
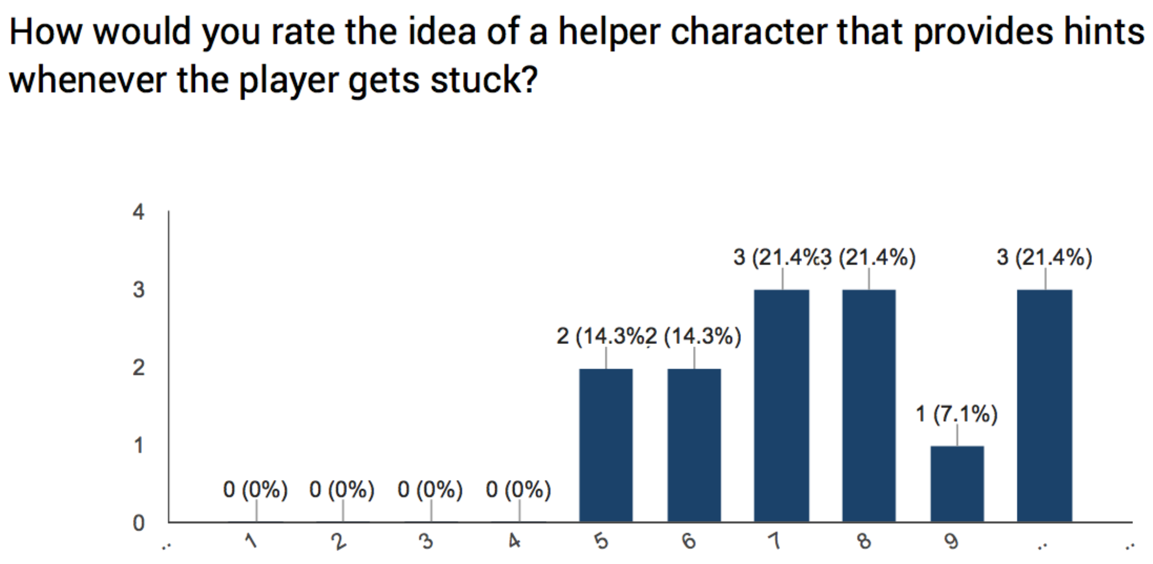
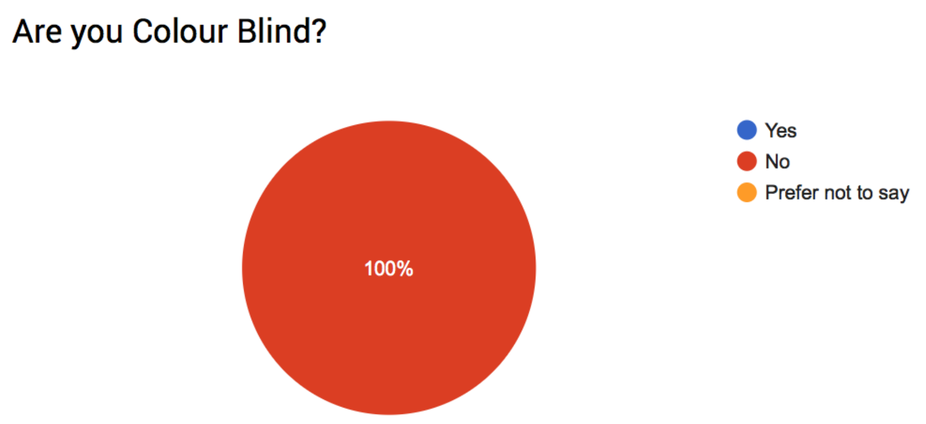


Figure 2



Bar Chart for Helper Character 1 = not helpful 10 = extremely helpful

Figure 3



Pie Chart Colour blind.

D: Acceptance Tests

Below is a table of the acceptance tests included within this project.

These tests need to be manually run using a copy of the game executable. They involve following a series of steps, verifying each of the assertions in the steps is true, and that the relevant GUI exists to allow the steps to be carried out. If all of the steps can be carried out, and their assertions are true, the test passes. If not, the test fails.

The acceptance tests are associated with an appropriate requirement to allow for traceability, and the tests aim to check that the code works for any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly unit tested, and some tests do not link up directly to a requirement, but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly function as intended.

Test Listing

ID	Test Steps	Req ID	Criticality	Result
2.01	To test the Main Menu: • Run game executable. • Check that the Main menu is shown. • Click on the New Game button. • Ensure the screen changes to the Narrator Screen. • Restart the game and click on the Quit button • Ensure that the game closes.	1.1.1	Low	Passed
2.02	To test the player movement using key presses: • Run game executable • On the Main Menu click on "New Game". • On the Narrator Screen click on "Start Game". • Once the game has loaded, press "W" on the keyboard. • Ensure that the player has moved upwards. • Press "S" on the keyboards • Ensure that the player has moved downwards. • Press "A" on the keyboard. • Ensure that the player has moved to the	1.1.2	High	Passed

E: Unit Tests

Below is a table of the unit tests included within this project.

The unit tests are associated with an appropriate requirement to allow for traceability, and the tests aim to check that the code works for any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly unit tested, and some tests do not link up directly to a requirement, but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly function as intended.

ID	Test Name	Purpose	Criticalit y	Class	Req ID	Result
1.1	constructor	Verifies GUI is constructe d correctly	High	GUIControl ler_Test	1.1.4	Passed
1.2	screenCanB eSet	Verifies that a screen from the game can be set as the current screen	Medium	GUIControl ler_Test	1.1.4	Passed
1.3	screenCanB eChanged	Verifies screen can be changed to a different screen	Medium	GUIControl ler_Test	1.1.4	Passed
1.4	getTime	Verifies the game returns the correct time value	Medium	GameSnapsh ot_Test	6.1.2	Passed
1.5	getRooms	Verifies the game returns the correct list of rooms	High	GameSnapsh ot_Test	4.1.1	Passed
1.6	getClues	Verifies the game returns the correct list of clues	High	GameSnapsh ot_Test	5	Passed
1.7	getClue	Verifies that the list held by the journal contains clues in the right format	Low	Journal_T est	5.2.1	Passed
1.8	addClue	Verifies that clues are added to the journal properly	Medium	Journal_T est	5.2.1	Passed

Continued on next page

Table 2.1 – continued from previous page

1.9	addConvers ation	Verifies that conversati on snippets are added in the correct format	Low	Journal_T est	7	Passed
1.10	getConvers ation	Verifies that getConvers ation returns all the con- versati on snippets that it is currently holding	High	Journal_T est	7	Passed
1.11	SetRoomGet Room	Verifies that a room, once set, is returned in the correct format.	Medium	MapEntity_Test	4	Passed
1.12	getName	Verifies that a map en- tity returns its name correctly	Low	MapEntity_Test	•	Passed
1.13	genDescrip tion	Verifies that a map entity returns its descriptio n correctly	Low	MapEntity_Test	•	Passed
1.14	getTexture	Verifies that a map entity sets its texture correctly	Medium	MapEntity_Test	•	Passed
1.15	setSpeech	Verifies that the narrator screen text can be set correctly	Low	NarratorSc reen_Test	7.1.7	Passed
1.16	updateSpee ch	Verifies that the narra- tor screen text can be updated correctly	Low	NarratorSc reen_Test	7.1.7	Passed
1.17	getTransit ion	Verifies that the player transition s between rooms	High	Room_Test	2.1.4	Passed

Continued on next page

Table 2.1 – continued from previous page

1.18	addTransit ion	Verifies that adding new transition s between rooms is correctly implemente d	High	Room_Test	2.1.4	Passed
1.19	walkable	Verifies that walkable tiles are walkable and tiles that aren't walk-able aren't walkable	High	Room_Test	•	Passed
1.20	trigger	Verifies that a trigger tile is triggerabl e, and one that isn't trig-gerabl e isn't triggerabl e	High	Room_Test	•	Passed
1.21	matRotatio n	Verifies that doormats are orientated the correct way	Low	Room_Test	2.1.4	Passed
1.22	distribute RoomsGiveC lues	Verifies that the distribute Clues() method distribute s clues so that there is at minimum 1 clue per room	Medium	ScenarioBu ilder_Tes t	5.1.1	Passed
1.23	distribute CluesDiffR ooms	Verifies that the clues aren't being given to the same room	Medium	ScenarioBu ilder_Tes t	5.1.1	Passed
1.24	generateMo tives	Verifies that the motive clue is split into 3 equal parts	Low	ScenarioBu ilder_Tes t	5.1.3	Passed
1.25	modifyScor eAddition	Verifies that the correct score is added to the current score	Low	Scoring_T est	6	Passed

Continued on next page

Table 2.1 – continued from previous page

1.26	modifyScor eSubtracti on	Verifies that the correct score is sub-tracted from the current score	Low	Scoring_T est	6	Passed
1.27	updateScor eNoDecrease	Verifies that the score doesn't change if the time passing is less than 5 seconds	Low	Scoring_T est	6.1.2	Passed
1.28	updateScor eHasDecrea se	Verifies that the the score decreases by 1 after 5 seconds	Low	Scoring_T est	6.1.2	Passed
1.29	getInfo	Verifies that all all parts of the clue are stored corretly	High	Clue_Test	5	Passed
1.30	isMotive	Verifies that motive clues return true as a motive clue and false if they are not motive clues	High	Clue_Test	5.1.3	Passed
1.31	isMeans	Verifies that means clues return true as a motive clue and false if they are not means clues.	High	Clue_Test	5	Passed
1.32	constructo rValidatio nFail	Verifies that given a false .JSON file, the game encoun-ters "JSON not being verified" error.	Low	Dialogue_ Test	7	Passed
1.33	constructo r2Validati onPass	Verifies that given a valid .JSON file no error is thrown.	High	Dialogue_ Test	7	Passed

Continued on next page

Table 2.1 – continued from previous page

1.34	getUsingCl ue	Verifies that when given a clue, the dialogue getter returns a valid response.	High	Dialogue_ Test	7	Passed
1.35	getUsingSt ring	Verifies that when given a string, the dialogue getter returns a valid response.	High	Dialogue_ Test	7	Passed
1.36	aStar	Verifies that the A-Star algorithm produces the correct result.	Low	Player_Te st	2	Passed
1.37	testPlayer Name	Verifies the player name is returned correctly	Low	Player_Te st	•	Passed
1.38	init	Verifies that the NPCs holds data about themselves correctly	High	Suspect_T est	3	Passed
1.39	hasBeenAcc used	Verifies that once a NPC who isn't the murderer has been accused, the NPC records that they have been accused.	Low	Suspect_T est	7.1.9	Passed
1.40	setPositio n	Verifies that the position of a NPC can be successful ly set.	High	Suspect_T est	3	Passed
1.41	setKiller	Verifies that the setKiller method successful ly sets the NPC as the killer.	High	Suspect_T est	3.1.5	Passed

Continued on next page

Table 2.1 – continued from previous page

1.42	setVictim	Verifies that the setKiller method successful ly sets the NPC as the victim.	High	Suspect_Test	3.1.5	Passed

F: GUI Examples

Download (AppendixF.pdf)

Requirements

Highlighted changes:

- All requirements have been reviewed and rewritten where appropriate
- IDs have changed due to recategorisation of requirements
- Improved introduction and document formatting
- More details are in the updates report

Introduction

At our first meeting we read the brief, and with that in mind, we played cluedo to get a feel for a general detective game. This sparked discussion as to what works in a game, and it gave us ideas for how our game should work. We decided the most effective way of thinking about the game was like “guess who”. Using the brief and our new ideas, we produced a list of features we’d like to include in the game, and then prioritised those.

After the initial meeting, we produced a number of user scenarios [1], and used our list of features to help produce a draft of the requirements. From these we identified ambiguous points and produced questions we needed to get answers for. We met multiple times with the customer throughout the design process, to present our requirements and ask questions we had. Using their feedback, we made any necessary changes.

Following the response we got after the Assessment 1 feedback was released, we decided it was best to rewrite our requirements. This lead to substantial improvements in the clarity and categorisation of our requirements, and this has benefitted us as it made the requirements easier to test against when it came to implementing the game.

When designing the requirements, we took these points into account:

- The requirements should be categorised as:
  - Functional requirements - these define what the system should do
  - Nonfunctional requirements - the define the behaviour of the system
- The requirements should be achievable within the time allocated of this project
- The requirements should consider the hardware in which the game should run
- The requirements should meet all points included in the brief
- We produced a survey asking about input methods and accessibility [2]. We got a sample of our target market to respond. From this we found:

- The preferred way of interaction with the game was keyboard and mouse.
- There were no results from colourblind people, however we still felt it was important that our game was accessible, so have included accessibility features as a “could” requirement. Colour blindness is a condition apparent in 1 in 12 men and 1 in 200 women [3] so we feel that it is a requirement that some people would benefit from.

The requirements are laid out in tables based on the IEEE standard for system requirements, as we felt this was a good standard to adhere to. The tables are split according to type (functional or non-functional), and category. Some requirements have associated risks, these are referenced in the table below, and are defined in the risks document .

Each requirement is given a unique identifier to make it easy to locate, and for traceability. The identifiers are made up of three numbers, using the following system:

- The first number is category, this represents the functional area of the requirement
- The second number represents how important the requirement is:
  - 1=Must implement
  - 2=Should implement
  - 3=Could implement
- The third number is the position in the list, used to ensure the identifier is unique.

Functional Requirements

Game

ID	Requirement	Success Criteria	Alternative	Risk ID
1.1.1	To start the game there must be a main menu	The game has a working clear menu	The game starts immediately when its opened	21-24
1.1.2	The game must be fully controlled by a mouse and keyboard	The game can be interacted with using only a mouse and keyboard	The game is controlled by an Xbox controller.	
1.1.3	Game must have different ‘plotlines’	Each gameplay is different in some way	Game only has 2 plot lines.	21-24
1.2.1	There should be a way of suspending or pausing the game	There is the option to pause the game which opens a pause menu	The game cannot be paused	21-24
1.3.1	Could be controlled by a gamepad	The game can be interacted with using a gamepad	Game is only controllable by keyboard and mouse.	
1.3.2	Could have a sound track	Music will be played when the game is running	There is no sound track.	21-24
1.3.3	If game has a soundtrack, it must have an option to turn the sound track off	There is a player accessible way to turn the sound track off within the game	The soundtrack would always be on.	

Player

ID	Requirement	Success Criteria	Alternative	Risk ID
2.1.1	The player must have a personality that is customisable	The players personality changes in the game	The player personality does not change.	21-24
2.1.2	Must not be able to accuse a NPC unless enough evidence has been found	The player cannot see the option to accuse an NPC unless they have interacted with enough clues	Can accuse an NPC without evidence.	21-24
2.1.3	The player must start in a central room at the start of every game	When the game starts, the player should be in the centre of the "Ron Cooke Hub"	Player can start in any room.	
2.1.4	The player must be able to navigate between rooms on the map	The player can move throughout the map and transition to other rooms when desired	The player progresses through rooms automatically.	21-24
2.2.1	The player should be able to see their current personality level	The game should present the player with an GUI element showing their personality level	The Player will not be able to see their current personality level	21-24

NPC

ID	Requirement	Success Criteria	Alternative	Risk ID
3.1.1	Each killer must have a motive	The killers all have motives	All killer have the same motive.	21-24
3.1.2	There must be 10 non playable characters.	The player should be able to locate 10 NPCs in rooms during the game	There are less than 10 NPC's.	21-24

Map

ID	Requirement	Success Criteria	Alternative	Risk ID
4.1.1	The game must have a map containing 10 separate rooms	The player should be able to visit 10 different rooms in the game	The game has less than 10 rooms.	21-24

Clue

ID	Requirement	Success Criteria	Alternative	Risk ID
5.1.1	There must be at least one clue in each room of the map	The player can navigate to every room and be able to locate a clue	Some rooms may have no clues, some may have multiple	21-24
5.1.2	The player must be able to interact with a clue	The player should be able to interact with a clue once it has been located	The player gets the clue without interaction.	21-24
5.2.1	There should be an inventory where clues can be placed by a player for future reference	The player can see an inventory in the GUI that allows visibility of collected clues	Clues are stored internally but the player will not be able to see them	21-24

Score

ID	Requirement	Success Criteria	Alternative	Risk ID
6.1.1	There must be a score shown to players in the game	The player must see a score displayed in the GUI	There will be no scoring.	21-24
6.2.1	There could be an online scoreboard to keep high scores	There could be a scoreboard in the GUI that presents the all time high scores	There will be a local list of high scores, or no scoring	21-24

Dialogue

ID	Requirement	Success Criteria	Alternative	Risk ID
7.1.1	The player must be able to interact with an NPC	A player can go up to an NPC and talk to them	The player cannot interact with NPC's.	21-24
7.1.2	The player must have the option of questioning an NPC	When a player talks to an NPC, they should have the option to question them	The player cannot question an NPC.	21-24
7.1.3	The player must have the option of ignoring an NPC	When a player talks to an NPC, they should have the option to ignore them	The player cannot ignore an NPC.	21-24
7.1.4	The player must have the option of accusing an NPC	When a player talks to an NPC, they should have the option to accuse them if they have found enough clues to accuse the NPC	The player cannot accuse an NPC.	21-24
7.1.5	The player must choose from a set of questions when interacting with an NPC that reflects different personalities	When a player talks to an NPC, and chooses to question them, they can choose from multiple speeches with different personality levels. Eg. Aggressive	The player only has one	21-24
7.1.6	Each NPC must respond differently to questions from a Player depending on both NPC's and Player's personality and characteristics	When an NPC responds to a player after being questioned, their response must be determined by their characteristics and the player's personality	All NPC's respond in the same way.	21-24

Nonfunctional Requirements

Game

ID	Requirement	Success Criteria	Alternative	Risk ID
1.1.4	Must run on the university computers	An executable is provided the runs on the computers	The game will not run on university computers.	7
1.1.5	Must run on Windows 10	An executable is provided that runs on windows 10	The game will not run on windows 10.	7
1.2.2	Should run on MacOS	An executable is provided that runs on MacOS	There will not be an executable that runs on MacOS	7
1.3.4	Could run on a mobile platform	An executable is provided to run on mobile platforms	The game will only run on desktop operating systems	
1.3.5	Could have a colour blind mode	The colours in the game can be configured by the user to be more accessible	There will not be a colour blind mode	21-24

NPC

ID	Requirement	Success Criteria	Alternative	Risk ID
3.1.5	Each NPC must have a personality that affects and is affected by game play.	The NPC will respond best to different types of question. For example, an aggressive NPC will respond best when questioned nicely.	All NPC's have the same personality.	21-24
3.1.6	The killer and victim must be randomly selected each time the game begins from two sub-lists of killers and victims.	When the game starts, the victim and the killer has been selected at random.	The killer and victim is the same every time.	21-24
3.1.7	Each NPC must be randomly assigned to a room at the start of the game	All NPCs should be situated within a different room at the start of the game.	Each NPC is always in the same room.	21-24

Map

ID	Requirement	Success Criteria	Alternative	Risk ID
4.1.3	The room where the murder occurred must be randomly selected at the start of every game	One random room should be the selected murder location at the start of every game	The murder room is always the same.	21-24

Clues

ID	Requirement	Success Criteria	Alternative	Risk ID
5.1.3	The murder weapon clue must be found before the player can accuse any NPCs	The player cannot accuse an NPC until they've located the murder weapon clue	Can accuse without the murder weapon.	21-24
5.1.4	Most clues should help with identifying the killer	A clue will narrow down the number of suspects left to be the killer	All clues help identify the killer	21-24
5.1.5	At the start of the game, clues must be randomly assigned to each room in the map	There must be at least one clue in every room of the map at the start of the game	Clues always in same location.	21-24
5.2.2	Clues could be picked up by a player and placed in an inventory	The player can interact with a clue and place it in their inventory for future reference	Clues will be stored internally, but my not be seen by the player	21-24

Score

ID	Requirement	Success Criteria	Alternative	Risk ID
6.1.2	The player's score must take into account the time taken	The score must change depending on how long the game has lasted	There will be no scoring.	21-24
6.1.3	The player's score must take into account the number of wrong accusations	The score must change depending on how many accusations the player has made	There will be no scoring.	21-24
6.1.4	The player's score must take into account the number of questions asked	The score must change depending on how many questions the player has asked	There will be no scoring.	21-24
6.1.5	The player's score must take into account the number of clues found	The score must change depending on how many clues have been found by the player	There will be no scoring.	21-24

Dialogue

ID	Requirement	Success Criteria	Alternative	Risk ID
7.1.7	The type of question asked to an NPC by a player must determine the player's personality	When a player chooses a speech to say to an NPC, their personality level is affected by their choice	The type of question asked affects nothing.	21-24
7.1.8	If an NPC is accused and isn't the killer then the NPC must refuse to interact for the rest of the game	When a player interacts with a previously accused NPC they shouldn't get a response	The NPC does not mind being falsely accused.	21-24

Bibliography

[1] Appendix A [online] docs.lihq.me/en/2.0.0/Assessment2/appendixA.html [Created 21/11/16]

[2] Appendix C [online] docs.lihq.me/en/2.0.0/Assessment2/appendixC.html [Created 21/11/16]

[3] Colour Blind awareness [online] http://www.colourblindawareness.org/colour-blindness/, [Accessed 3/11/16]

Risk Assessment and Mitigation

Highlighted changes:

- Each risk now has a responsible risk owner
- Improved description as to how we came up with the risks and improved formatting
- More details are in the updates report

Introduction

Risk management is an important part of any project, we must prepare for what could happen during the course of the project in order to be able to quickly recover and stay on track. The risks which are shown below take into account the scale of the project and aim to cover only risks which are realistic within this context.

To determine risks we brainstormed various scenarios - such as a teammate being ill for more than a few weeks. From these scenarios, we collected possible risks, and worked out the likelihood of them occurring. To determine the risk we discussed how it would impact the project, focussing on how many knock-on effects the issue would cause.

The risks are presented in a tabular format, with the following columns:

- **Risk ID** - this allows for traceability across the project
- **Description** - describes what the risk is for
- **Likelihood** - each risk has an estimated likelihood on a scale
  - **High** =good chance this risk will occur, about 75% chance
  - **Medium** =equal chance of risk occurring or not, about 50% chance
  - **Low** =not likely to occur, about 25% chance, however some risks may be lower
- **Impact** - this describes the impact the risk would have to progress in the project
- **Severity** - shows the severity of the impact on the project on a scale
  - **High** =a major setback which could affect the whole project
  - **Medium** =could add up to a week of extra work and may threaten a deadline
  - **Low** =may mean a few extra hours of work, but nothing major
- **Mitigation** - describes how how we will aim to avoid such a risk and deal with it
- **Owner** - describes the owner of the problem, where the owner is the person most likely to be responsible for the issue.

The overall table is split into sections which group together similar risk such as software risks. Each section is then ordered by severity, highest first. Equal severity is ordered by likelihood.

This table will be regularly consulted in an attempt to monitor the risks and try to ensure they do not occur and catch them early if they are occurring.

Due to the size of the team we feel that these risks are appropriate and accurate.

Table of risks

Software risks

ID	Description	Like- li- hood	Impact	Sever- ity	Mitigation	Owner
1	Our game may be slow or unresponsi ve.	Medium	No one will want to play the game.	High	Improve efficiency of the game wherever possible and regularly check performance	Coding Team
2	Software library doesn't work or lacks a feature e.g. has a bug that stops the game from working, or is missing a feature required for the game to work.	Medium	We would struggle to implement the feature we want to add. We would also use up lots of time trying to solve the issue.	Medium	Test the elements of the library you plan to use beforehand . Also, make sure the library has an active community surrounding it and that bugs are fixed quickly. If it was early stages we could get a new library but this would require us to rewrite our code to work with the new library.	Soft- ware Library Owner
3	Code is hard to understand and seems too complex.	Low	Could cause bugs and makes bug fixing harder. Slows down the productivi ty of the group.	Medium	Use meaningful variable names and plenty of comments, both in code and in commit messages. Make sure code is reviewed by the majority of members before it gets merged into the repository .	Coding Team
4	Conflicts in git. Different members changing the same code.	High	May need to move code around and even rewrite.	Low	Make sure people work on separate elements by assigning them to different tasks and if not then make use of Gits tools.	Coding Team
5	Our own software doesn't work as intended.	High	Will need to bug fix. Loss of time and potentiall y productivi ty if that function or feature is the bottleneck of the game.	Low	This is a normal part of software developmen t. We all make mistakes. However, before code is approved by the group we will use unit testing that will test key functions of the game as we develop them meaning that should a function break we will know about it before it's merged.	Coding Team/Desig n Team/Proje ct Man- ager

Hardware risks

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
6	Personal computer breaks long term or is lost.	Low	Could lose work and be unable to work.	Low	Ensure work is saved online to google drive cloud service and that code is stored on github. Department PC's should be accessible most days and have all the tools we need.	Final User
7	Personal computer crashes while working.	Medium	Potentiall y will have lose work, meaning you lose time doing it again.	Low	Save regularly, google docs[2] will do this for us. Regularly commit code to personal branches so that it stored elsewhere other than your PC .	Final User

Risks with people

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
8	A team member leaves the module or even the course.	Low	They may have only access to their work, also the rest of the team will have more to do.	High	As above store online but also try to keep each other motivated to avoid this.	Project Team
9	A team member is ill/away for a week or two.	High	They might have been skilled in a certain area that no other member can do well.If they have the only access to work may get behind from it.	Medium	Hard to avoid, but we should store work online where everyone can access. If we work in pairs to complete tasks then there will be less of a chance of having one person who knows the most about one area.	Project Team
10	Arguments within the team.	Medium	Disrupts the work of the team and prevents us moving forwards. Also, unpleasant for the team as a whole.	Medium	Try to avoid conflict but if necessary have proper debates perhaps using a mediator, do not keep issues hidden.	Project Manager
11	Lack of communicat ion.	Medium	Tasks may be done twice or not done at all.	Medium	Keep strong communicat ion using the tools we plan to use.	Project Manager
12	A team member does not do their work.	Medium	Could disrupt other members work and could make the other team members annoyed.	Low	Don't give members too much work or work they cannot do, ensure that the team communicat es well and regularly meets up to discuss how the work is going.	Project Team/Manag er

Risks with tools

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
13	Google drive servers stop working.	Low	Could lose/lose access to work that is stored there.	Medium	Store work locally , and on other services.	Google
14	Central git repository [1] is lost in some way.	Low	Temporaril y lose access to it.	Low	Keep up to date local copies so can be easily restored. We could host our own local copy should github go down.	Git/Coding Team
15	Website hosting fails.	Low	Users lose access to the website.	Medium	The website files are stored on github and every team member has a local copy of the repository on their computer so we could bring the site back up on a different server. The site is also protected by cloud-flar e[3] who will provide a cached version of the site if our host were to go down.	Website Hosting Owner

Requirements risks

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
16	Not including a requiremen t which is required by the customer.	Low	We let the customer down and have failed them.	High	Make sure key requiremen ts are elicited from the customer so they get what they want.	Requiremen ts Team
17	A requiremen t could change/ be added.	High	May need to rewrite code or add extra code to account for it. Extra time will be needed.	Medium	Our software architectu re must be flexible and able to be changed easily.	Requiremen ts Team
18	Stating a requiremen t that we cannot actually achieve.	High	Let down the customer and also waste time.	Medium	Be sensible when deciding requiremen ts, be sure you can achieve them.	Requiremen ts Team/Codin g Team
19	Ambiguity in requireme nts.	Medium	May end up making something which is not what was originally intended.	Medium	Ensure requiremen ts are clear and check any ambiguitie s with the customer.	Requiremen ts Team
20	Choosing requiremen ts that the customer doesn't really want.	Medium	Waste time which could be spent on requiremen ts they did want.	Low	Ensure you know which requiremen ts the customer really wants and which can be ignored.	Requiremen ts Team

Estimation risks

ID	Description	Likelihood	Impact	Severity	Mitigation	Owner
21	Expect the team or a team member can do more than they actually can.	Medium	Work is not done or is done to an insufficie nt standard.	Medium	Give tasks that people can do and if they can't then help them. When working on difficult tasks work in pairs to complete the task meaning individual team members don't feel as overwhelme d by the task	Project Manager
22	We may underestim ate how long it will take to do some work.	Medium	Work ends up taking longer than expected or not done to the standard it could be done. This could cause other areas of the project to suffer	Medium	Set realistic timings to do work and be realistic on how long a task will take. Account for unforeseen delays in our plan adding time where we can catch up.	Project Manager
23	Be too pessimisti c about what we can achieve.	Medium	We end up with a product which is not as good as it could have possibly been.	Low	Push our limits but also stay realistic and within the requiremen ts. If we have extra time then we can use it to enhance the product.	Project Manager
24	Distribute tasks incorrectl y.	Low	Team over/under worked.	Low	Distribute tasks appropriat ely and tell others if feel over/under worked.	Project Manager

Bibliography

[1] GitHub [online] Available https://github.com [Accessed 01/11/2016]

[2] Google Drive [online] Available https://www.google.com/drive/ [Accessed 01/11/2016]

[3] Cloud Flare [online] Available https://www.cloudflare.com/ [Accessed 01/11/2016]

Methods and Planning

Highlighted changes:

- Group leader has changed
- Added more documentation about our software development process and methods
- Improved section discussing the way we work and methodologies
- Updated Assessment 3 plan
- In terms of how we work, the group now runs as more singular units than in groups.
- More details are in the updates report

Software Engineering Methods & Tools

Development Process

We are using an agile-scrum [0] approach for our software engineering project, as it allows our team to carry out work in a flexible manner, while providing the flexibility to evaluate our progress and plan for the next phase of work as we go along. Details of how we use this methodology to run our meetings and plan tasks are found in the team organisation section.

One of the key tools needed for any software engineering project is a version control system. This is a tool that will allow us to keep track of code changes, and collaborate on the same piece of code. We're using GitHub[1] for our code repository, as it's a popular tool that our team members are familiar with.

While developing, we plan to use git's branch and pull request functionality to help us manage multiple developers on the same project. To ensure our code quality is high, we ensure that a different developer performs a code review for every pull request, and that appropriate unit tests have been added and successfully pass before approving the branch to be merged.

We are using test driven development for our acceptance tests, as these are being written before development starts to ensure our code meets the requirements. While development is occurring, unit tests will be added to our code using JUnit[2]. This is so that we maintain a high level of test coverage, and have confidence that our code is working as intended. It will help with ensuring regressions do not occur. We are using CircleCI[3], a continuous integration server, to run all the unit tests every time a commit is made to GitHub, this will ensure that nobody accidentally breaks the codebase. For more details, please see the testing report.

To manage our development tasks, we plan to use the GitHub Projects feature of our code repository. This lets us manage our tasks and code issues with a Kanban board[4], and allows for items to be assigned to team members so everybody knows who's doing what. GitHub also allows us to link issues to the project, and these are useful because team members can collaborate on design decisions and keep track of what's happening in the code. More about our team organisation is discussed later.

Development Tooling

To develop the project, we decided, after much deliberation, to use Java[5] and IntelliJ IDEA[6] as our programming language and IDE. This is because Java is a language accessible to all members of the team, as well as the entirety of our cohort. We did give serious consideration to other programming languages such as C#[7] with Unity[8] however few in our team have any experience in the language which means it may take longer and cause problems later on in the process.

We needed a library that provides useful game APIs to help speed up our development process. We did some research and found 4 options of libraries we could use: Swing[9], Mini2Dx[10], LibGDX[11], LWJGL[12]. We quickly realised LWJGL and Swing weren't right for our project. In the end we decided to use the LibGDX Java game development framework. We chose LibGDX over Mini2DX because LibGDX has support for the IDEs that we intend to use, and it provides features that simplify several game development steps.

Design Processes

We will need to do some designing whilst the development stages are progressing. The main tool we will be using to design the game assets is Piskel[13], an online tool that allows us to easily create game assets. We expect designs to start as paper prototypes that will be developed further as the project goes on.

As we'd have to produce a map for our game (see requirement 4.1.1), we were pleased to discover that LibGDX has support for tiled maps. There is useful software called Tiled [14] that allows us to create a high quality 2D map and export it for use with the framework. This will save us massive amounts of time compared to alternative solutions, and this can be used to improve gameplay and ensure that there are few bugs.

Collaboration Tools

In addition to using GitHub projects as mentioned above, we also need some communication and collaboration tools to help us keep the project on track.

For the first few weeks we were using Facebook Messenger[15] for team communication, but it became apparent that it wasn't up to the job as we kept losing important pieces of information. We decided to switch to Slack [16] as it allows us to have separate chat channels for different aspects of the project, letting us keep all necessary information separated. Since separate discussions are split into separate channels, we are able to find important information more easily for future reference. The team engages in daily communication on Slack, we use this to consistently report our progress, ask questions on ambiguous issues and organise future in-person meetings.

We are using integration between our development tools and Slack to help us keep track of progress in one clear feed, in particular the GitHub integration and the CircleCI integration. This helps with our development method, as it ensures all team members are kept up to date with the status of the repository and test results.

For online meetings we are using Join.me [17] Join.me is a free tool with voice, text and video chat as well as a screen sharing feature which allows a presenter to share their screens with other team members. Join.me allows efficient and effective team meetings to take place even whilst all members are in separate locations. We made extensive use of Join.me over Christmas holiday period, where we used it to hold a full team meeting, using our scrum method as described later.

For document storage and collaboration, we are using Google Drive[18] with a shared team folder. This contains all of the documents and other files we've been working on. We are making use of Google Docs, Sheets & Forms[18] as they allow us to collaboratively work on documents at the same time, as well as providing mechanisms that allow us to review and comments on our documents.

Team Organisation Roles

We decided early on to have a team leader role. The team leader is responsible for ensuring everybody has a task to be working on, and for making sure progress is being made. They are also responsible for producing meeting plans, and answering queries of any team member.

In Assessment 1, we voted to decide a group leader - Brooke Hatton won as we felt he was a natural leader. For Assessment 2, we decided that Jason Mashinchi would take over as group leader, as he had experience in a software development team. We plan to alternate group leaders throughout the assessments, to give the other member a break.

Every member of the team is assigned tasks to do at the end of every meeting for the sprint ahead. We decide who does what based on who wants to take on the task, or based on previous experience if it's relevant to a task.

Meetings

The team aims to have a meeting at least once a week, during which our team leader acts as our scrum master. We may not strictly follow the agile-scrum rules, but we've found that our approach works well for us.

Our team leader prepares for every meeting by producing a brief plan of what needs to happen during the meeting to ensure that we are making progress. Having a plan means that we can stay on track within the meeting and don't go off track and don't make progress. We tend to keep meetings high-level so that we don't waste time on implementation details that can be decided without the entire group present.

Meetings are scheduled to happen multiple times a week depending on availability of team members. Our meetings always signify the start of a new sprint. We typically start with each member going through what they've been working on in the previous sprint, and they raise any problems or questions they may have. This allows us to catch issues, concerns or blockers that have arisen early on in the meeting, so we can take them into account when planning the next sprint.

We then proceed to discuss what needs to be done during the next sprint, and assign tasks to each individual in the team. This is great because it means everybody has something to be working on for the next week, and ensures that we are making sufficient progress in the project.

During Sprints

Everybody is assigned a task to be working on during sprints, and they are responsible for ensuring their bit of work gets done. Every team member uses the kanban board containing issues on GitHub to keep the rest of the team up to date on where their task is at, and they make sure that their appropriate issues/tasks are assigned to themselves.

If any problems arise during the week, team members use their assigned GitHub issues to discuss anything related to their task, or we communicate through Slack when we need to discuss something more general. Splitting up the communication methods in this manner allows us to find discussion when necessary, as often we make design decisions within GitHub that can be referenced later.

Systematic Plan

Assessment 2

The focus of the team will fully switch to the second assessment on Wednesday 9th November, once the first assessment has been submitted. We plan on completing the second assessment by Tuesday Spring Week 2 giving us a week to account for any unexpected developments or fixing issues that happen to arise. It also gives us time to analyse, criticise and improve our own work thus improving the quality of our code and enhancing the functionality and efficiency. This will also ensure that we have very high quality documentation, once all group members are happy with the readability of our code we plan on requesting that students from other teams look at sections of our code to test readability. There will also be a 2 week rest from SEPR to account for time spent studying for exams and also the exam week itself.

Assessment 3

After the completion of Assessment 2 on Tuesday 24th January we will work as a team to decide the project to take on for Assessment 3. This will be done within one week, after which we shall decide upon areas that need to be worked upon over the next assessment, and delegate tasks accordingly to team members. Documentation is started in the first week with everyone focusing on making sure reports and code documentation are up to date. within the next week all documents will be near to completion and the code will have started to be updated. Again, we plan on completing the work a week before the deadline to give us time to fix any issues found.

A Gantt chart [19] containing the schedule for key tasks from all the assessments can be found in the appendix. This chart includes priorities, task dependencies and a critical path.

Gantt Chart

Below is an extract of our Gantt chart. The full version can be seen on our website under Appendix B (http://lihq.me/Downloads/Assessment2/AppendixB.pdf).

Bibliography

[0] Waterfall to Agile: Flipping the Switch - Bhushan Gupta [Online] Available: http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf [Accessed 25/10/2016]

[1] Github [Online] www.github.com [Accessed 8/11/2016]

[2] JUnit [Online] http://junit.org [Accessed 22/01/2017]

[3] CircleCI[Online] https://circleci.com/ [Accessed 22/01/2017]

[4] Kanban board [Online] www.github.com [Accessed 8/11/2016]

[5] Java [Online] www.oracle.com [Accessed 8/11/2016]

[6] IntelliJ [Online] www.jetbrains.com [Accessed 8/11/2016]

[7] C# [Online] www.msdn.microsoft.com [Accessed 8/11/2016]

[8] Unity [Online] www.unity.com [Accessed 8/11/2016]

[9] Swing [Online] www.oracle.com [Accessed 8/11/2016]

[10] Mini2DX [Online] www.mini2dx.org [Accessed 8/11/2016]

[11] libGDX [Online] www.libgdx.badlogicgames.com [Accessed 8/11/2016]

[12] LWJGL [Online] www.lwjgl.org [Accessed 8/11/2016]

[13] Piskel [Online] www.piskelapp.com [Acessed 8/11/2016]

[14] Tiled [Online] www.mapeditor.org [Accessed 8/11/2016]

[15] Facebook Messenger [Online] https://en-gb.messenger.com/ [Accessed 23/01/2017]

[16] Slack [Online] www.slack.com [Accessed 8/11/2016]

[17] Join.me [Online] www.join.me [Accessed 8/11/2016]

[18] Google Drive [Online] https://www.google.com/drive/ [Accessed 23/01/2017]

[19] Gantt Chart [Online] http://lihq.me/Downloads/Assessment1/AppendixB.pdf

Appendix: Task Assignment Summary

Assessment 1:

- Ben Jarvis was assigned the requirements document
- Benjamin Grahamslaw was assigned the risks and mitigations report
- Brooke Hatton and Jason Mashinchi were assigned the Architecture report
- Joseph Shufflebotham and Vishal Soomaney were assigned the Methods report

Assessment 2:

- Benjamin Grahamslaw was assigned the GUI report & user manual
- Brooke Hatton, Jason Mashinchi & Joe Shufflebotham were responsible for the implementation of the game, testing and the design choices made.
- Other team members contributed towards implementing smaller features in the game
- Jason Mashinchi was assigned the testing report
- Brooke Hatton & Vishal Soomaney were assigned the architecture report
- Ben Jarvis & Benjamin Grahamslaw were assigned the document updates
- Joe Shufflebotham & Vishal Soomaney were assigned the implementation report

Architecture

Concrete Architecture

We have decided to continue using UML 2.X[1] to describe the architecture of our software project. The reason for this is that it allows us to create a diagram which provides a clear visualisation of the structure of the project. The diagram generated can be used to visualise how classes relate to each other, and to clearly see the attributes of each individual class, along with each of their methods.

We decided to use a class diagram instead of an object diagram since we wanted to show the classes our project consists of and what those classes were each capable of (methods). The object diagram would instead have shown the interaction between the classes at some point in run time.

At the start of the assessment, we scoped out which requirements and features we wanted to implement by the assessment deadline. We used the assessment brief provided to us and and took into account the amount of time we had until the assessment deadline to determine how far we wanted to take the project and what features we wanted to implement. As we began to plan out the implementation details, and started to make design decisions, it became clear that changes would need to be made to the architecture defined in the previous assessment.

For the first assessment, the abstract and concrete architecture were designed using the tool Draw.io [2], however we decided against using this tool for designing the latest version of the concrete architecture. This was because the complexity of our architecture has increased, and continuing to use Draw.io (a tool better suited for simpler diagrams) would make the task unnecessarily complex and time consuming.

Through some research we found out that we could save time by automatically generating UML class diagrams based on our code. We used the built in functionality of IntelliJ IDEA [3], the IDE we have been using. Once we had completed the implementation required for this assessment, we used IntelliJ to generate the UML diagram [4] which we then proceeded to thoroughly check for any errors or inconsistencies before annotating it and posting it below.

The diagram on the next page shows a summary of our code architecture, the full concrete architecture can be found at this link:

http://lihq.me/images/UML.png

Justification for concrete architecture

Update of abstract architecture

The implemented architecture has a few differences to the previously defined architecture. We aimed to keep the vast majority of the original abstract architecture[5] the sam e as the previous assessment, however some changes were necessary as the design and implementation process went on.

We wanted to make sure the structure of the game was clear and easy to understand before we started coding, with the aim of making our code more maintainable. While constructing our game, we regularly reviewed what we had already implemented and discussed how we would structure the classes that had yet to be implemented.

In our abstract architecture, we left out information about how the user interface of the game would be structured for implementation purposes, so this is a new addition to the concrete architecture over the abstract architecture. We modified the structure so that the Player[16] and NPC[17] now have a location in a room rather than in the map, this is explained further below.

Structure of concrete architecture

In the concrete architecture the Player[16] and NPC[17] classes now have a pointer to a room rather than to a location on the map, however the Map class still exists and constructs the transitions between the rooms. This was done due to the way we are handling the rooms in LibGDX, rather than having one continuous map file we have separated it out into rooms which all have their own coordinate grid.

To handle the coordinates of the Player[16] and NPCs[17] we have defined the Vector2Int[13] class. This describes the position of a Person in a room, because each room is made up of tiles. Vector2Int uses integers for the values of x and y (these each will match a tile in the room). We were going to use Libgdx's Vector2[14] class however this uses doubles which could cause issues as we do not want a person to stop between two tiles, so using integers simplifies the mechanics of our game).

LibGDX (the game engine) uses screens to split the application up. The screens we currently have are the MainMenuScreen[6], PauseScreen[7] and NavigationScreen[8], all of these screens extend the AbstractScreen[9] we created. The AbstractScreen[9] class was not defined in the original architecture, however we determined it useful for the purpose of defining the methods and attributes that all screen classes have in common.

For the UI we also added a package of 'Elements', these are small UI components that are used together to make up the UI of the game. Treating the UI as components allows us to improve maintainability as the code is separated. These elements allow us easily change them and to add relevant information in the game for the player.

To handle dialogue two classes were created, ConversationManagement [10] and SpeechboxManager [11]. These were added to separate the logic for creating conversations (generating the dialogue), and the logic for rendering the conversations to the screen, using SpeechBox[12] UI element.

The Player[16] and NPC[17] both extend a AbstractPerson[15] (previously called Person), as described in our original architecture diagram[5]. We have also added a Player Controller[18] which handles all user input. The Clue[19] class remains as shown in the original abstract architecture [5], the Player[16] has a list which contains the clues found at any given point in the game.

The original concrete architecture mentioned a N arrator[20] class, which we planned to use for guiding the player through the game. This was removed from the class diagram and wasn't implemented due to time constraints, however the only architectural change that would have been required to implement this would be a new Narrator class to handle the appropriate text and GUI elements.

There are also a few classes (Settings, Assets and UIHelpers) that aren't linked to anything - these contain constants and methods used across the game. Sharing code helps prevent unnecessary duplicate code and aids with maintainability. Finally, there is one other class DebugOverlay[28], which is useful for debugging the game, as it adds a GUI overlay showing collision properties of each tile.

Relating Concrete Architecture to Requirements

[1.1.1] "To start the game a main menu is needed" , several architectural changes were made to implement this, these included the creation of a MainMenuScreen[6] class which uses the Menu[23] class from the elements package to create the screen.

[1.1.2] "The game must be fully controlled by mouse and keyboard", we added a PlayerController class, which takes input from the user and controls the Player class; a StatusBar class which will handle the pressing of buttons on the bar, and a SpeechBox & SpeechBoxButton [25] classes that are used together to allow the user to view and continue conversations with the mouse.

[1.2.1] "There should be a way of suspending or pausing the game". The StatusBar[24] class was added, to include GUI for a pause button. The PauseScreen[7] class was created - this uses the Menu class and is an architectural change since it wasn't present originally. The logic for menus is contained within the Menu class, as much of the code is common between the pause and main menus.

[2.1.4]"The player must be able to navigate between rooms on the map", the Map[21] and Room[22] classes defined in the original concrete architecture allow this. To meet this requirement, we added a NavigationScreen [8] class, which loads the current room and handles player movement both within the room and between rooms by working in conjunction with the Room and Map classes.

[3.1.6] ”The killer and victim must be randomly selected each time the game begins from two sub-lists of killers and victims.”, This is achieved by choosing random killers and victims in the initialiseAllPeople() method in the GameMain[27] class. We considered creating a different class to handle this however we decided to keep it simple. Eventually, we plan to have a defined subset of NPCs that can be killers or victims, but this has not been implemented yet.

[3.1.7] ”The NPCs are assigned to rooms randomly ”. No architectural changes were required to implement this, the GameMain[27] class handles random assignment of NPCs and the Room and NPC classes keep track of NPC location.

[5.1.1] ”The clues are randomly assigned to rooms”. Clues are randomly assigned to rooms using the initialiseClues() method in the GameMain[27] class. An architecture change was made whereby the Vector2Int[13] class was created to define coordinates within the game, this is used to keep track of player, clue and NPC location.

[5.2.1] “There should be an inventory where clues can be placed by the player”, we initially had a separate Inventory class to keep track of clues, however we decided to simplify the architecture, by replacing it with a simple list in the Player class. At this stage we haven’t implemented the inventory.

[7.1.1] “The player must be able to interact with an NPC” , for this purpose the architecture was altered and the ConversationManagement[10], SpeechboxManager[11], SpeechBox and SpeechBoxB utton[25] classes were implemented, these allow for dialogue to appear upon interaction with an NPC. The UIHelpers[26] class was also created to handle all the shared methods and attributes that the SpeechBox, SpeechBoxButton[25] and StatusBar classes had in common.

Bibliography

[1] OMG Unified Modeling Language TM (OMG UML) Version 2.5 [Online] Available: http://www.omg.org/spec/UML/2.5/PDF/ [Accessed: 25/10/2016]

[2] Draw.io, “Flowchart Maker & Online Diagramming Software” [Online] Available: https://draw.io/ [Accessed: 25/10/2016]

[3] IntelliJ IDE [Online] Available: https://www.jetbrains.com/idea/ [Accessed: 18/01/2017]

[4] IntelliJ Viewing Diagram (UML Class diagram) [Online] Available: https://www.jetbrains.com/help/idea/2016.3/viewing-diagram.html [Accessed: 18/01/2017]

[5] Original abstract architecture [Online] Available: http://docs.lihq.me/en/1.0.0/architecture.html [Accessed: 22/01/2017

[6] JavaDocs reference to MainMenuScreen class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/MainMenuScreen.html [Accessed: 22/01/2017]

[7] JavaDocs reference to PauseScreen class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/PauseScreen.html [Accessed: 22/01/2017]

[8] JavaDocs reference to NavigationScreen class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/NavigationScreen.html [Accessed: 22/01/2017]

[9] JavaDocs reference to AbstractScreen class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/AbstractScreen.html [Accessed: 22/01/2017]

[10] JavaDocs reference to ConversationManagement class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/ConversationManagement.html [Accessed: 22/01/2017]

[11] JavaDocs reference to SpeechBoxManager class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/SpeechboxManager.html [Accessed: 22/01/2017]

[12] JavaDocs reference to SpeechBox class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/elements/SpeechBox.html [Accessed: 22/01/2017]

[13] JavaDocs reference to Vector2Int class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/models/Vector2Int.html [Accessed: 23/01/2017]

[14] Libgdx API "Vector2" class [Online] Available: https://libgdx.badlogicgames.com/nightlies/docs/api/com/badlogic/gdx/math/Vector2.html [Accessed: 23/01/2017]

[15] JavaDocs reference to AbstractPerson class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/people/AbstractPerson.html [Accessed: 23/01/2017]

[16] JavaDocs reference to Player class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/people/Player.html [Accessed: 23/01/2017]

[17] JavaDocs reference to NPC class [Online] Available: Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/elements/DebugOverlay.html [Accessed: 23/01/2017]

Implementation Report

The requirements we set out for ourselves, along with the requirements for assessment 2, unfortunately, haven't all been implemented.

Assessment 2 specific requirements The GUI:

The GUI for the game is partially implemented, in our overall requirements we specify several GUI related requirements.

[1.1.1] "To start the game there must be a main menu". This has been implemented and allows the user to start the game via the click of a Start Game button. When the button is pressed it moves to the NavigationScreen [1], which is the main screen for the game.

[1.1.2] "The game must be fully controlled by a mouse and keyboard". This requirement has been fully met all the buttons on the game are clickable by the mouse and the Player[2] is controlled by the keyboard. To manage the inputs we are using a InputMultiplexer [3]

[5.2.1] "There should be an inventory where clues can be placed by a player for future reference". This hasn't been implemented fully, at this stage we felt there were other GUI elements that needed to be implemented before we implemented the GUI for this, the code that stores Clues[4] when the user has picked them up does work, so it's just the GUI representation that is missing.

At most 6 detectives:

This has been fully implemented with 6 detectives being randomly distributed about the map each game instance. In regards to our overall requirements [3.1.2], "There must be at least 10 non playable characters (NPCs)". 6/10 non playable characters have been implemented and the code will allow for more without any modification other than the construct of the additional NPCs [5].

Question or accuse (but not ignore):

Both question and accuse have been partially been implemented the related overall requirements below explain this in more detail. [7.1.2] "The player must have the option of questioning an NPC". A Player[2]

has method so that it can question and a NPC [5] has methods that

respond about a Clue[4] that the Player[2] has found.

[7.1.6] "Each NPC must respond differently to questions based on the personality of the NPC and the personality level of the player". This was partially implemented with a lot of the groundwork in place for future development currently the Player[2] can select how they want to ask the question, the NPC [5] will then only respond with a helpful answer if you ask it in a way that matches their personality.

[2.1.2] "Must not be able to accuse a NPC unless enough evidence has been found", this has been partially implemented. Since the Player[2] class keeps track of the evidence or Clues[4] found so far, it checks[] to see if the

Player[2] has 4 or more Clues[4] if it doesn't the the option to accuse is not shown. However, [5.1.3] "The murder weapon clue must be found before the player can accuse any NPCs" has not been implemented as not all Clues[4] had been finalised at this stage.

At most one clue per room:

This requirement has been fully met, no more than one Clue[4] has been place in any given room. This has been completed by randomly placing the Clues[4] around the rooms. Not placing a Clue[4] in a room that already has a Clue[4] unless there are no empty rooms left.

[5.1.1] "There must be at least one clue in each room of the map". This has been partially implemented as there is one Clue[4] in 7/10 rooms meeting the requirements of assessment 2

Bibliography

[1] JavaDocs reference to NavigationScreen class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/NavigationScreen.html [Accessed: 22/01/2017]

[2] JavaDocs reference to Player class [Online] Available:

http://lihq.me/docs/JavaDocs/me/lihq/game/people/Player.html [Accessed: 23/01/2017]

[3] Github wiki reference the Libgdx InputMultiplexer [Online] Available: https://github.com/libgdx/libgdx/wiki/Event-handling#inputmultiplexer

[Accessed: 23/01/2017]

[4] JavaDocs reference to Clue class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/models/Clue.html [Accessed: 23/01/2017]

[5] JavaDocs reference to NPC class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/people/NPC.html [Accessed: 23/01/2017]

GUI Report

Main menu

The main menu will be the first screen that a player will see upon loading up the game. It will provide the user with the following options:

- **New game button** - this starts the game
- **Settings button** - See/change settings.
- **Exit button** - this closes the game

The buttons are all designed in the same way throughout the game. They are designed to be easy to distinguish from the background and for all text displayed on them to be easy to read.

Related requirements: 1.1.1

Realisation: Appendix F:1

In this assessment the main menu has these three buttons, however settings are not yet implemented.

Settings

The settings menu gives the player access to settings to modify the game experience. In our current version of the game the settings are not available, however they will eventually include: Music On/Off, Colour-Blind Mode On/Off and Fullscreen/Windowed

Realisation: N/A

Related requirements: 1.3.2, 1.3.3, 1.3.5

Pause menu

The pause menu will allow the player to access the settings menu, and also links back to the main menu.

The buttons in the pause menu are the same as the main menu.

Related requirements: 1.2.1

Realisation: Appendix F:2

In this assessment the pause menu contains buttons to resume the game, access settings and quit.

Main navigation screen

The main navigation screen contains two elements. While the player is playing the game they will be able to see their character on the map and move around from room to room. The character is displayed in the middle of the map for good playability. This screen also has an overlay at the bottom including a meter which displays the player's current temperament. A dialogue box appears upon interacting with an NPC, showing dialogue, or items.The in game overlay also contains links to the pause menu and the player's inventory as simple buttons similar to the main menu.

Related requirements: All map sections,2.1.4,2.2.1,all NPC sections.

Realisation: Appendix F:3

In this assessment we have a large explorable map of the RCH, the overlay includes buttons for inventory, pause menu and placeholders for score and the personality meter.There is also basic NPC interaction.

Inventory

Although the inventory is not currently implemented, the inventory includes the list of clues that the player has already found as well as a list of the characters encountered so far. Here the player can cross off people they are convinced are innocent.

It will also allow access to the map.

Related Requirements:4.1.2, 5.2.1, 5.2.2

Testing

Methods & Approaches

We decided to take a requirements focused view of testing, so that our tests are built around making sure the requirements were being met for the project, but also making sure that we tested key parts of our code that may not directly correspond to any requirement. We decided to use two types of testing, these can be seen below:

Acceptance Testing: this allows us to ensure the end product meets the requirements, and that a user can perform common scenarios successfully.

- Each test takes the form of a list of steps to carry out in the game.
- If a user can perform the list of steps successfully, the test passes, else it fails.
- These tests have to be run manually, because it would take too long to automate them and we have time constraints on the project.
- They are good for testing both functional and nonfunctional requirements.
- Can test functional requirements as the tests can confirm whether the game has the appropriate functionality to meet the requirement.
- Can test non-functional requirements as tests can confirm the project behaves as expected.
- Allows us to detect regressions during refactoring or other code changes.
- We use a **test-driven development** approach here, where the acceptance tests were written before the cod e, and more tests should pass as more of the requirements are implemented.

Unit Testing: this allows us to check that our code does what it is meant to.

- Takes into account both normal and edge cases to ensure normal behaviour and boundaries are handled correctly.
- The unit tests are written as the code is being implemented, due to their reliance on the code structure and design decisions.
- We have used both black box testing (doesn't take into account how code works) and white box testing (uses inner workings of code) when designing our unit tests.
- Allows us to detect regressions during refactoring or other code changes
- Written as JUnit unit tests in Java, because it is compatible with our project, easy to use, and has high quality documentation.
- These tests are automated, so whenever a commit is pushed to GitHub, the test script is run on CircleCI (our continuous integration server).
- We chose CircleCI for it's reporting abilities, as it provides a html website, and breakdowns of failing tests with debug logs where applicable, making it easy to diagnose problems.
- After the CI runs the unit tests, the result is pushed back to GitHub and updates our team Slack. GitHub is setup to block merging pull requests if a unit test fails.
- Continuous integration helps remove much of the effort required to run tests, and ensures that unit tests are taken into account when developing the project.
- These tests also alert us if the project fails to build with each commit, as the project is built by the CI server in order to run unit tests. This can let us know about possible code problems, like syntax errors.

Our software development process involves creating new branches for new features, and when a branch is ready, a pull request is created to merge the branch into the master branch. We've setup our pull request flow to block merging if the unit tests have failed, which helps us catch code issues sooner rather than later. Even within a couple of days of starting programming, the testing checks had saved us from merging code that looked good, but was in fact broken in a non-obvious way.

Test Report

Overall Statistics

- **48 tests**

- **18 failures**
- **63% successful**

Unit Tests

Below are the statistics generated by our unit test runner on CircleCI for the Assessment 2 release version of the code. These tests are functions within the code (although they don't get compiled into the final executable) that tests a small unit of code at a time. They are fully automated, and were automatically run when the appropriate commit was made to GitHub.

- **14 tests**
- **0 failures**
- **100% successful**

All of the unit tests in the project have passed, which indicates the absence of bugs in the tested code sections. More comments on what this result means are on the next page. The unit tests have test IDs indicated by 1, followed by their index in the list.

See Appendix E for a full listing of the unit tests.

Acceptance Tests

We have manually run through our acceptance tests to check the game works as intended, and to verify that the game fulfils the requirements. The results are shown below.

- **34 tests**
- **18 failures**
- **53% successful**

The vast majority of the acceptance tests passed, however a significant percentage failed because the appropriate code hasn't been implemented yet. Once the appropriate code has been written, these tests should pass when they are run again. More comments on what this result means are on the next page. The acceptance tests have test IDs indicated by 2, followed by their index in the list.

Failed Test IDs: 2.03, 2.05, 2.06, 2.07, 2.11, 2.12, 2.14, 2.16, 2.17, 2.20, 2.26, 2.28, 2.29, 2.30, 2.31, 2.32, 2.33, 2.34

See Appendix D for a full listing of the acceptance tests.

Test Information

The tests are associated with an appropriate requirement to allow for traceability, and to check that the code meets any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly tested, and some tests do not link directly to a requirement but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly work as intended.

Results & Evaluation

The unit tests for this project all passed. Most of the acceptance tests passed, however a significant percentage of the tests failed - these should pass once the relevant features have been implemented.

The test completeness is not perfect, nor is it possible to be. The unit tests only check their section of the code works as intended, and doesn't cover the integration of that code, or it's use within the entire project. What the unit tests do indicate, is that the specific code that they test works as intended, and they indicate the absence of bugs in that specific code. When combined with the acceptance tests, the test completeness is improved (but still not perfect) as not only are key functions of the code tested, but the overall product is tested to ensure it meets the requirements.

This project doesn't have perfect test correctness either. The unit tests could have bugs in them meaning bugs could be missed in the code, or the unit tests may not cover every edge case or normal use case that exists, which means bugs and issues could slip by undetected. The same sort of thing happens with acceptance testing, as typically a limited range of scenarios are tested, which may not account for the all of the very many possible ways of using the game. Also, the acceptance tests included in this project need to be run manually, which means that human error could occur and affect the overall correctness of the tests.

To improve our testing **completeness** and **correctness** , additional types of testing would be useful additions, such as fully automated end-to-end testing, or integration testing, along with more tests of any type. Introducing different types of tests would alert developers about the presence of a different kind of bug, which would allow identification of bugs that have previously been uncaught. Adding more tests would also decrease the chances of code regressions being missed, or other issues or bugs being missed during the testing process.

Testing material

Additional testing material can be found on the website (http://www.lihq.me).

This comprises of:

- **Executable test plan**:
  - Plan: http://docs.lihq.me/en/2.0.0/Assessment2/executableTestPlan.html
- ***Evidence of testing***:
  - Code: https://github.com/Brookke/Lorem-Ipsum/tree/master/game/tests/src
  - Continuous integration test script: https://github.com/Brookke/Lorem-Ipsum/blob/master/circle.yml
- ***Test design & results***:
  - Acceptance tests: http://lihq.me/Downloads/Assessment2/AppendixD.pdf
  - Unit tests: http://lihq.me/Downloads/Assessment2/AppendixE.pdf

Updates on Assessment 1

Requirements

What has changed, and why?

Based on feedback from the previous assessment, the entire requirements specification has been reworked and looked at.

- Almost all requirements have been changed and updated

- We have updated each requirement to make them more concise and objective, in order to reduce the "design pollution". This involved objectively reviewing each requirement and ensuring that they precisely specified the function or behaviour of the game
- The format has remained the same, as we still use the IEEE standard for system requirements
- We standardised and corrected our definitions of functional and nonfunctional requirements, and used these as a basis for categorising all of the requirements in the document. From the feedback given we were made aware that our understanding of this was incorrect, so made have changes to correct these problems .
- The idea and aims behind each of the requirements has remained largely unchanged
- Any small changes to the description of individual requirements was due to re-evaluation of what that requirement was attempting to achieve.
- The IDs of the requirements were modified to fit the changed categorisation
- We continued to ensure the requirements referenced appropriate risks

Link to updated version: http://docs.lihq.me/en/2.0.0/Assessment2/requirements.html

Link to version control diff: https://github.com/Brookke/Lorem-Ipsum/pull/116/files#diff-28d223c584badedf323df6c577820471

Methods

What has changed?

Although we as a group feel that sometimes change is necessary for a group to run smoothly, we have not had much change occur during the second assessment. However, we have agreed to change a few minor things, allowing us to operate more efficiently as a team.

- We are coding exclusively in IntelliJ while still ensuring that it can be opened, modified and run in Eclipse.
- The group leader has changed to Jason.
- Rather than 3 sub-groups, we are now operating more as singular units, sometimes pairing up to do more complicated tasks.
- As the process went on, we found it easier to think in terms of tasks we had to do, rather than spend valuable time deciding on S.M.A.R.T targets per person. The document has been changed to reflect that
- Systematic plan for Assessment 3 was added.
- Following feedback, we have added a greater focus on the methods we are using for software engineering, as such a large portion of the document has been rewritten or adapted to better fit the brief
- Improved section on how we are conducting meetings and sprints

Why has it changed?

- Due to team preference, no-one in our group is coding using eclipse. This is also in part due to it being easier to discuss problems to do with coding interface (such as not being able to import the project correctly at the start) as everyone will see the same UI.
- In our group we have had two leaders so far Brooke and Jason. The group unanimously decided that we would alternate leaders so that no one got overworked.
- Due to the nature of coding and the amount of code to write, for this assessment we decided to split the work more than previously discussed to get more done in the same time. By ensuring that all work is still checked,

this will ensure that we have more time spent on each part of the assessment so we will end up with a higher quality end product.

- As a group we have decided that we work well together and can decide on our own tasks and goals well so felt that S.M.A.R.T targets were unnecessary to help us construct targets for the sprint ahead
- We have elaborated on how we are going about software engineering using the agile-scrum methodology, and discussed further what we plan to do in our meetings and sprints. This should help the document better fit the brief.

Link to updated version: http://docs.lihq.me/en/2.0.0/Assessment2/methods.html

Link to version control diff: https://github.com/Brookke/Lorem-Ipsum/pull/116/files#diff-e0ebb380121eb0162b6d02872c6705b9

Risk Management

What has changed?

We feel that in the last assessment all major and minor risks were outlined well so we do not feel the need to add any more or change them in any way.

- We did improve the introduction documentation to address points raised in the feedback we received. This included greater detail into the risk identification process.
- We also introduced bullet points to aid with readability
- We added an owner column to the risks table

Why has it changed?

This document has been updated to include this information as we felt that it did not show enough detail in some elements of risk management.

Firstly the document did not describe how we actually determined the risks outlined in the document, therefore we felt it necessary to describe how we went about this.

Also the risks document did not show who actually had responsibility for the risks that we identified. It is important to know who is responsible so that if the risk does occur we know who will know how to deal with it, for example if there were conflicts in Git we would know who knows how to deal with them to resolve the issue quickly and efficiently.

Link to updated version: http://docs.lihq.me/en/2.0.0/Assessment2/riskAssessmentAndMitigation.html

Link to version control diff: https://github.com/Brookke/Lorem-Ipsum/pull/116/files#diff-0a977fd0f40fa7a9b531ba8b356ff907

Executable Test Plan

Unit Tests

This project is unit tested with JUnit. Tests are located in the `/game/tests` directory in the GitHub repository. For documentation on writing these tests, please see https://github.com/junit-team/junit4/wiki

Running Unit Tests

For every commit CircleCI runs all the included tests, however, we recommend that you run tests locally too before committing.

We have included a handy test configuration inside the repository that can be run from IntelliJ IDEA.

Adding Tests

- Create new class for tests under `/game/tests/src` When naming the class end the name with `UnitTests` for consistency e.g. `PlayerUnitTests`
- This class should extend `GameTester` this initialises the backend of the game so that tests run correctly.
- Import `org.junit.Test`
- Write a test function using assertions, and use `@Test` decorator above it
- See this page for examples of assertions: https://github.com/junit-team/junit4/wiki/assertions
- Run your tests locally and see if they pass!

CircleCI

Viewing Results

After tests have run the results are displayed in the “Test Summary” tab on CircleCI. This tab contains a summary of the testing result, along with any tests that have failed.

If the tests have failed and no test summary is provided, this normally means that the code doesn’t compile, or there is a problem with the test code. To gather more information, scroll down to read the console output from when the tests were run.

Also, CircleCI collects test “artifacts”, which are located in the “Artifacts” tab. This contains useful output files such as:

- HTML output: A website that provides a visual testing report with more details
- JUnit XML output: XML files for each class that provide raw data about each run test

Configuration

We have included a configuration file for setting up CircleCI tests in the root directory of the project. See `circle.yml`.

To setup CircleCI, you will need to link your GitHub account. Once this is done, you can add a project within CircleCI, which will automatically setup tests using the configuration file (`circle.yml`) mentioned earlier.

Whenever a commit is pushed to GitHub, CircleCI will run the tests and inform GitHub of the result, which will be displayed against each commit, and in any pull requests.

Acceptance Tests

We’ve also included a bunch of acceptance tests that can be run manually to ensure the game performs as expected, and meets the requirements. These have not been automated, so will need to be run by hand. These can be found in the Appendix for the Assessment 2 documents.

A: User Scenarios

Persona 1

David, 18 year old male at a Computer Science Open Day. He is not an avid gamer and his interests lie more in electronics.

Scenario

David is initially unsure as to how the game is played, however he is provided a tutorial by the narrator which explains the puzzle, how he is expected to solve the puzzle, how to move the main character, interact with clues and non player characters. After becoming accustomed to the keys and finding a clue, he is unsure as to what the next step is to solve the puzzle. He interacts with the narrator who subsequently provides hints as to the next course of action that will get the user closer to solving the puzzle. Near the end of the game he becomes stuck and spends a long time attempting to find out which of his 3 final suspects is the murderer, the narrator pops up on the side of the screen to ask whether the player requires assistance, David says “yes” and is told to speak to a character which holds the final piece of information required to solve the puzzle.

Persona 2

Louise, 17 year old female student at an open day. Enjoys playing puzzle games and is colour-blind.

Scenario

Although she is initially worried about missing a clue or finding it hard to read some of the text, she is able to go through the introductory tutorial without issue due to the text colour and background colour contrast in a way to avoid issue with most forms of colourblindness. She is able to see the clues quite clearly due to map tile colours and clue item colours being chosen in a way so as to not cause problems to most people will colourblindness. She is then able to solve the crime quicker than most and accuse the correct murderer of the crime on her first try thanks to her experience played puzzle games.

Persona 3

Ben, 19 year old second year Computer Science student. He is an avid gamer who takes games quite seriously and always tries to get the high score.

Scenario

He progresses through the tutorial quickly and is already accustomed to using keyboard keys to interact with the game. As he attempts to concentrate and solve the puzzle as quickly as possible so as to get the high score, the background noise begins to bother him. Fortunately he notices that there is a “turn music off” button in the sidebar of the game and this allows him to continue playing comfortably. Although he finishes the game with a good score, he is unable to beat the high score and wishes to play again, fortunately due to the game having a set of different characters from which murderers and victims are chosen randomly, he can play several times whilst going through a new, different and exciting storyline each time.

B: Project Plan (Gantt Chart)

We've produced a systematic plan for future assessments. This is in the format of a Gantt chart, created in Excel.

Download (AppendixB.pdf)

C: Survey Results

Figure 1

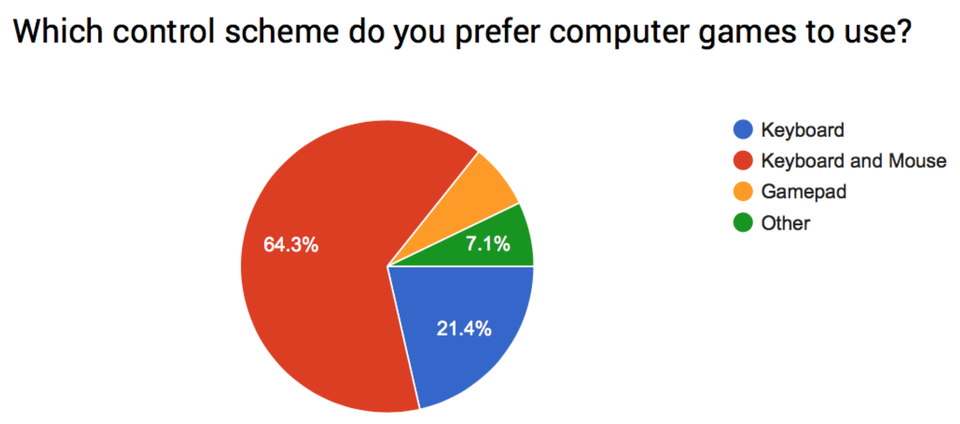
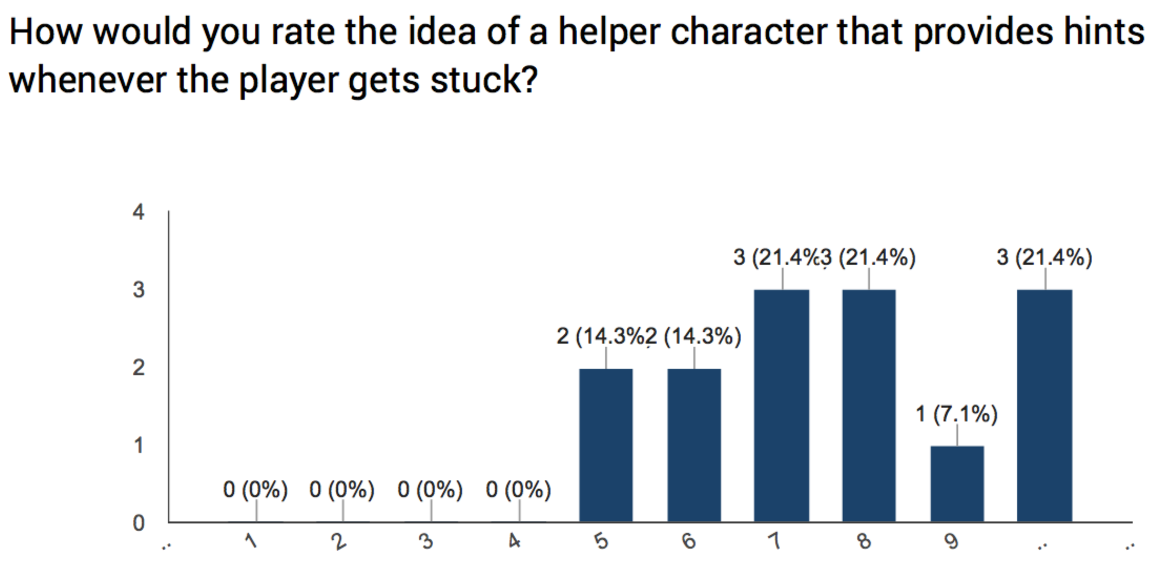
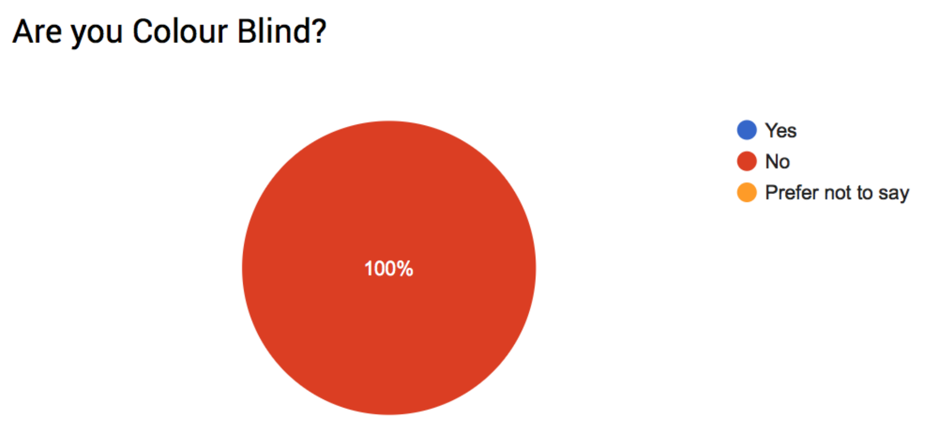


Figure 2



Bar Chart for Helper Character 1 = not helpful 10 = extremely helpful

Figure 3



Pie Chart Colour blind.

D: Acceptance Tests

Below is a table of the acceptance tests included within this project.

These tests need to be manually run using a copy of the game executable. They involve following a series of steps, verifying each of the assertions in the steps is true, and that the relevant GUI exists to allow the steps to be carried out. If all of the steps can be carried out, and their assertions are true, the test passes. If not, the test fails.

The acceptance tests are associated with an appropriate requirement to allow for traceability, and the tests aim to check that the code works for any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly unit tested, and some tests do not link up directly to a requirement, but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly function as intended.

Test Listing

ID	Test Steps	Req ID	Crit-ical-ity	Re-sult
2.01	Run game executable. Main menu is shown.	1.1.1	Low	Passed
2.02	Run game executable Start game using main menu Player can move around using arrow keys and "WASD" keys	1.1.2	High	Passed
2.03	Run game executable. Start game using main menu. Play game to completion. Repeat at least 3 times. Ensure the combination of selected killer, victim, murder location and murder weapon differ each time. Note: This failed as murder weapon clue is not implemented	1.1.3	Medium	Failed
2.04	Run game executable. Start game using main menu. Click the 'pause' button Pause menu is shown	1.2.1	Low	Passed
2.05	Run game executable. Start game using main menu. Player should move around using the gamepad Player can interact with Items and NPCs using the gamepad	1.3.1	Low	Failed
2.06	Run game executable. Ensuring speakers are turned on, the game plays music. Note: the music is not currently implemented	1.3.2	Low	Failed
2.07	Run game executable. Click on 'options' button. Uncheck the 'music' checkbox. Music stops playing. Note: the music is not currently implemented	1.3.3	Low	Failed
2.08	Run game executable. Start game using main menu. Interact with an NPC. The 'accuse' button is not visible. Collect 4 clues. Interact with an NPC. The 'accuse' button is now visible.	2.1.2	Medium	Passed
2.09	Run game executable. Start game using main menu. Player starts in a location. Close game. Run game executable. Start game using main menu. Player starts in same location	2.1.3	Low	Passed
2.10	Run game executable. Start game using main menu. Navigate the player to a doorway (marked by a carpet). Move through the doorway. Player is be in a different room.	2.1.4	High	Passed
2.11	Run game executable. Start game using main menu. Personality meter displays the current personality level. Note: the personality meter is not currently implemented	2.2.1	Medium	Failed
2.12	Run game executable. Start game using main menu. Navigate through all rooms of the map. There are 10 NPCs around the map. Note: this failed as for Assessment 2 only 6 NPCs have been implemented	3.1.2	High	Failed
2.13	Run game executable. Start game using main menu. Navigate through all rooms of the map. There are 10 distinct rooms that the player has traveled through.	4.1.1	High	Passed
2.14	Run game executable. Start game using main menu. Navigate through all rooms of the map. The player can find at least one clue in each room. Note: this failed as for Assessment 2 at most one clue is in each room	5.1.1	High	Failed
2.15	Run game executable. Start game using main menu. Find a clue. The player can interact with the clue.	5.1.2	High	Passed
2.16	Run game executable. Start game using main menu. Click the 'inventory' button. Inventory screen is shown. Note: the inventory is not currently implemented	5.2.1	Medium	Failed
2.17	Run game executable. Start game using main menu. Score is displayed in the status bar. Note: the score is not currently implemented	6.1.1	Low	Failed
2.18	Run game executable. Start game using main menu. Navigate the player to an NPC. The player should be able to talk to the NPC.	7.1.1	High	Passed
2.19	Run game executable. Start game using main menu. Interact with an NPC. Select 'Question' button. The player can question the NPC.	7.1.2	High	Passed
2.20	Run game executable. Start game using main menu. Interact with an NPC. Select 'Ignore' button. The player can ignore the NPC.	7.1.3	Medium	Failed
2.21	Run game executable. Start game using main menu. Pick up at least 4 clues Interact with an NPC. Select 'Accuse' button. The player can accuse the NPC.	7.1.4	High	Passed
2.22	Run game executable. Start game using main menu. Find a clue Interact with an NPC. Select 'Question' button. The player should be able to ask the NPC various questions.	7.1.5	High	Passed

2.23	Run game executable. Start game using main menu. Game runs at over 30 frames per second.	1.1.4	Medium	Passed
2.24	In Windows 10 Game executable run s.	1.1.5	High	Passed
2.25	In macOS Game executable runs.	1.2.2	High	Passed
2.26	In Mobile Game executable runs. Note: a mobile version is not currently implemented	1.3.4	Low	Failed

| implemented | | | |

2.28	Run game executable. Start game using main menu. Interact with an NPC. 'Accuse' button is not visible. Find and interact with the murder weapon. Interact with an NPC. 'Accuse' button is now visible. Note: the murder weapon clue is not currently implemented	5.1.3	Medium	Failed
2.29	Run game executable. Start game using main menu. Find a clue. Interact with clue. Click 'Inventory'. The clue appears in the Inventory. Note: the inventory is not currently implemented	5.2.2	High	Failed
2.30	Run game executable. Start game using main menu. Confirm the score decreases as time passes. Note: the score is not currently implemented	6.1.2	Medium	Failed
2.31	Run game executable. Start game using main menu. Make note of current score. Interact with an NPC. Accuse the NPC. Confirm the score has reduced. Note: the score is not currently implemented	6.1.3	Medium	Failed
2.32	Run game executable. Start game using main menu. Make note of current score. Interact with an NPC. Question the NPC. Confirm the score has reduced. Note: the score is not currently implemented	6.1.4	Medium	Failed
2.33	Run game executable. Start game using main menu. Ensuring personality score is not an extreme value, note down personality meter reading. Interact with an NPC. Question the NPC in a non-neutral way. If the question was positive, the personality meter is now higher. If the question was negative, the personality meter is now lower. Note: the personality meter is not currently implemented	7.1.7	Medium	Failed
2.34	Run game executable. Start game using main menu. Interact with an NPC. Accuse the NPC. Interact with the NPC again. The NPC 'refuses' to interact. Note: this is not currently implemented	7.1.8	Low	Failed

E: Unit Tests

Below is a table of the unit tests included within this project.

The unit tests are associated with an appropriate requirement to allow for traceability, and the tests aim to check that the code works for any associated requirements. Not all requirements have associated tests, and vice versa - this is because some requirements cannot be explicitly unit tested, and some tests do not link up directly to a requirement, but are still needed to ensure the code functions as intended.

There is a criticality measure against each test, for both acceptance and unit tests - this is to represent how important the test is to the overall function of the code. Criticality is on a scale - high criticality means that if that test fails, the project will not function at all; low criticality means that if the test fails, the project will still mostly function as intended.

ID	Test Name	Purpose	Criti- calit y	Class	Req ID	Re- sult
1.01	testDescri ption	Verifies clue descriptio n is returned correctly	Medium	Clue	5.1.2	passed
1.02	testEquali ty	Verifies clues can be compared correctly for equality	Medium	Clue	5.2.1	passed
1.03	testName	Verifies clue name is returned correctly	High	Clue	5.1.2	passed
1.04	testTileCo ordinates	Verifies clue has the correct coordinate for the map	High	Clue	5.1.1	passed
1.05	testPerson ality	Verifies the NPC personalit y is returned correctly	Medium	NPC	3.1.5	passed
1.06	testGetName	Verifies NPC name is returned correctly	Low	NPC	3.1.2	passed
1.07	doesPlayer Move	Verifies the player move function works correctly	High	Player	2.1.4	passed
1.08	testPlayer Personality	Verifies the player personality is set and get correctly	Medium	Player	2.1.1	passed
1.09	testPlayer Name	Verifies the player name is returned correctly	Low	Player	_	passed
1.10	testAddTra nsition	Verifies transitions between rooms can be added	High	Room	4.1.1	passed
1.11	testGetTra nsition	Verifies transitions between rooms can be performed	High	Room	4.1.1	passed
1.12	testMatRot ation	Verifies direction of room transition is correctly returned	Medium	Room	_	passed
1.13	testTrigge r	Verifies function to check if a tile is a trigger to perform an action	High	Room	_	passed
1.14	testWalkab le	Verifies that a tile returns the correct property for walkabilit y	Low	Room	_	passed

F: GUI Examples

Download (AppendixF.pdf)

Requirements

Introduction

First the team read the brief and associated documentation. Then we got together and discussed our ideas, coming up with a rough draft of basic and requirements. From these we discussed the more ambiguous points, such as what sort of view the player will use, and where the murder will take place. Then we picked out the most ambiguous parts that prevented us from continuing with basic designs and discussed them with our customer. After two interviews with our customer we finalised the requirements.

At the first group meeting we played cluedo to get a feel for a general detective game. This sparked great discussion, and allowed for great strides to be made on requirements very early on, by giving us an idea of what works in a game. This also allowed us to discuss ideas that would be interesting but more difficult to implement, which we left as possible expansion if overall time costs do not overrun. We also produced a number of user scenarios[1] which we used to help guide our thinking when producing requirements.

After discussing our ideas on clues for a bit, as a group we decided that the most effective way of deducing the murderer was to attempt a ‘guess who’ style. This way the player can cross off certain NPCs upon finding clues that confirm their alibis. This should allow the player to feel as though they are actually solving the crime, rather than just witnessing the solving.

The requirements are sorted into two tables, functional and non-functional. Functional requirements are things that the system does, so mainly requirements for when the game is running. Non-functional requirements are things that the system is, so mainly qualities the game must have, or objects that are hard coded.

The tables below are numbered using the following system:

- The first number is category (The bit in bold e.g. NPCs)
- The second number represents 'must(1), should(2), could(3)' (the priority of achieving the requirement)
- The third number is the requirements position in the list (to make it easier to find and refer to)

The requirements are laid out in tables based on the IEEE standard for system requirements for ease of access. By giving each requirement a row and a number all information is easy to find quickly in future. Any and all associated risks are discussed in the risks document and are referenced in the tables.

Some requirements are based on considerations of environmental assumptions of the game. Requirements based on hardware have all taken into account the systems that the game will run on. We produced a survey [2] which we got a small number of our target market to answer. From this we found that the preferred way of interaction with the game was keyboard and mouse. Although our survey produced no results of colourblind people, it is a condition apparent in 1 in 12 men and 1 in 200 women [3] so we still feel that it is a requirement that some people would benefit from, hence why it is listed as 'could'.

Functional Requirements

NPCs

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
1.1.1	The murderer and victim must be randomly selected each time the game begins from two sub-lists of murderers and victims.	Upon loading the game at least 2 times either or both the victim and murderer will change.	N/A - necessary requirement.	
1.1.2	The NPCs must spawn randomly into rooms so that each room has at least one NPC at the start of the game.	Upon loading of the game the NPCs will be spread around the map so that there is no room without an NPC.	NPCs will spawn in different rooms.	21

MAP

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
2.1.1	Rooms on the viewable map must be revealed only once the player has visited the respective rooms.	Walk through map and check that rooms are only revealed once visited.	All of the map will be visible from the start.	21

Dialogue/Interaction

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
3.1.1	The player must always get three choices of interaction with NPCs - Question, Accuse and Ignore.	Ensure while game is running the player can use any of these interactions with NPCs.	N/A - necessary requirement.	
3.1.2	All dialogue must change based on the characteristic s of the player NPC and NPC.	Not every NPC will respond in the same way to the same question, the options the player has to interact with the player changes throughout the game.	Each NPC has set lines of dialogue.	3
3.1.3	The game should have a method, for example an interface to allow the player to view the map, and some form a information on NPCs, clues found and other points of interest.	Check functionality as described throughout a play-through. Ensure that there is a clear way of displaying and closing interfaces.	N/A - otherwise the player cannot review information collected.	

Clues

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
4.1.1	Clues must help with the elimination process, but some will point to more than one NPC.	Ensure that all clues look meaningful, but some have false assumptions associated with them.	N/A - clues are necessary.	
4.1.2	The murder weapon must be found before the player is able to accuse any NPCs.	Make sure the player cannot accuse NPCs before they have found the murder weapon.	Murder weapon does not have to be found before the player is able to accuse NPCs.	21

Score

No.	Requirement	Success Criteria	Alternative Require-ments	Risk ID
5.1.1	The score must be raised when a clue is found.	If the player finds a clue, the score is raised.	There will be no scoring.	21
5.1.2	The score must be lowered for each wrong accusation and each question asked.	The score changes as required.	There will be no scoring.	21
5.3.1	The player could start with a predetermined score which is reduced for each second played.	Check that score reduces as time is spent playing the game.	There will be no scoring.	21

Other/System

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
6.3.2	IF (dependant on 6.3.1) the game has a soundtrack it must have a ‘sound on/off’ option.	Check that sound turns on/off when appropriate option chosen.	The game will have no soundtrack.	21

Non-Functional Requirements

NPCs

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
1.1.3	The game must have a cast of 10 NPCs (Non-Player NPCs).	The game contains 10 NPCs.	N/A - necessary requirement.	
1.1.4	The murderer must have a motive that becomes clear at some point in the game, not necessarily before they are accused.	Game functions as described.	The murderer will not have a clear motive.	21
1.1.5	There must be a narrator who acts as the tutorial and further help during gameplay.	The narrator talks to the player.	There will be no narrator.	3

MAP

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
2.1.2	The game must contain a game-map of 10 separate rooms, spread across the setting of 'The Ron Cooke Hub'.	The game will have 10 rooms that are accessible to the player.	N/A - necessary requirement.	
2.2.1	The room of the crime scene/murder room must be chosen randomly each time the game begins.	Upon loading the game at least 2 times the murder room will change.	The crime scene is always in the same place.	21

Dialogue/Interaction

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
3.1.4	The game must have multiple 'plot lines' .	Play through the game multiple times, checking that the plot lines differ each time.	N/A - necessary requirement.	
3.2.1	Some plotlines could be more intricate than others.	Play through the game and determine that some plotlines are more complicated .	The game line has similar plot lines.	3

Clues

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
4.1.3	There must be at least one clue to find in each room on the map.	Make sure that clues spawn in each room.	N/A - necessary requirement.	
4.2.1	Some ‘constant’ clues should be available, for example the guest sign in book in the central part of the map.	Check that consistent clues spawn in the correct place on at least 2 separate occasions .	There are no ‘constant’ clues.	21
4.2.2	Some rooms should have more than one clue e..g note left by victim/murder weapon.	Check that at least one room has at least one clue in.	There is only one clue per room.	21
4.3.1	The player could be able to interact with or pick up some items which are not clues.	Check the player can interact with some item and it not be listed as a clue.	All clues will be meaningful.	3

Score

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
5.1.3	The player must be scored on time taken, number of wrong accusations, number of questions asked and number of clues found.	Play through the game at least 3 times to check that scores add up as expected.	There will be no scoring system.	21
5.3.2	A list of high-scores could be stored on a server.	Check the server contains the high scores.	There will be a local list of high scores or no list of high scores.	3

Other/System

No.	Requirement	Success Criteria	Alternative Requirements	Risk ID
6.1.1	The game will be controlled by keyboard with mouse integration.	Determine the game is controlled as described.	N/A - necessary requirement.	
6.1.2	The game must play on a windows based system.	Determine the game runs on the system described.	N/A - necessary requirement.	
6.1.3	The game must be played in a 'top down' viewpoint, where the player is in the centre of the screen and the world moves around the player. The viewpoint is fixed zoom.	Check the game is viewed as described.	The game will be played in an improved viewpoint based on the reason for discarding this one.	5
6.2.1	The game should run smoothly on university computers.	Use frame-rate measuring software to obtain a frame-rate of at-least 30.	N/A - necessary requirement.	
6.3.1	The game could have a soundtrack.	Check that sound plays when game is running.	The game will not have a soundtrack.	21
6.3.3	The game could have a 'colour blind' setting.	When activated, the colourblind setting changes all textures in the game to ones that are easier for a colour-blind person to see.	The game textures will be designed with colour blindness in mind.	5
6.3.4	The game could be cross compatible on mobile (android) and Mac.	Make sure game runs on alternative systems.	The game will not be cross compatible.	5
6.3.5	The game could be controlled by a gamepad.	The game is controlled as described.	The game will not use a gamepad.	21

Bibliography

[1] Appendix A [online] docs.lihq.me/en/1.0.0/appendixA.html [Created 21/11/16]

[2] Appendix C [online] docs.lihq.me/en/1.0.0/appendixC.html [Created 21/11/16]

[3] Colour Blind awareness [online] http://www.colourblindawareness.org/colour-blindness/, [Accessed 3/11/16]

Risk Assessment and Mitigation

Introduction

Risk management is an important part of any project, we must prepare for what could happen during the course of the project in order to be able to quickly recover and stay on track. The risks which are shown below are relevant to the particular SEPR context and take into account the scale of the project and aim to cover only risks which are realistic within this context.

The risks are here presented in a tabular format. This table is split into 6 columns; the first column gives the risk an identification number (ID) for easy reference in our requirements and also if a risk does happen and we need to resolve it. The second column describes the risk itself. The third column gives an estimated likelihood of the risk occurring. To indicate the likelihood of each risk occurring we have use a high medium and low rating which is then also colour coded:

Likelihood

Low likelihood = This risk is not likely to occur. Roughly a 25% chance although some extreme risks could be a lot lower.

Medium likelihood = There is an equal chance of the risk occurring or not occurring. Roughly a 50% chance.

High likelihood = There is a good chance that this risk will occur . Roughly a 75% chance.

The fourth column describes the impact the risk would have on progress in the project. The fifth column shows the severity of the impact using this colour coordination:

Severity

Low severity = May mean a few hours extra work but nothing major.

Medium severity = Could add up to a week of extra work and may threaten a deadline.

High severity = A major set back which could affect the whole project.

The sixth and final column details how we will aim to avoid such a risk and deal with it.

The overall table is split into sections which group together similar risk such as software risks. Each section is then ordered by severity, highest first. Equal severity is ordered by likelihood. This table will be regularly consulted in an attempt to monitor the risks and try to ensure they do not occur and catch them early if they are occurring.

Table of risks

Software risks

ID	Description	Likelihood	Impact	Severity	Mitigation
1	Our game may be slow or unresponsive .	Medium	No one will want to play the game.	High	Improve efficiency of the game wherever possible and regularly check performance
2	Software library doesn't work or lacks a feature e.g. has a bug that stops the game from working, or is missing a feature required for the game to work.	Medium	We would struggle to implement the feature we want to add. We would also use up lots of time trying to solve the issue.	Medium	Test the elements of the library you plan to use beforehand. Also, make sure the library has an active community surrounding it and that bugs are fixed quickly. If it was early stages we could get a new library but this would require us to rewrite our code to work with the new library.
3	Code is hard to understand and seems too complex.	Low	Could cause bugs and makes bug fixing harder. Slows down the productivity of the group.	Medium	Use meaningful variable names and plenty of comments,bot h in code and in commit messages. Make sure code is reviewed by the majority of members before it gets merged into the repository.
4	Conflicts in git. Different members changing the same code.	High	May need to move code around and even rewrite.	Low	Make sure people work on separate elements by assigning them to different tasks and if not then make use of Gits tools.
5	Our own software doesn't work as intended.	High	Will need to bug fix. Loss of time and potentially productivity if that function or feature is the bottleneck of the game.	Low	This is a normal part of software development. We all make mistakes. However, before code is approved by the group we will use unit testing that will test key functions of the game as we develop them meaning that should a function break we will know about it before it's merged.

Hardware risks

ID	Description	Likelihood	Impact	Severity	Mitigation
6	Personal computer breaks long term or is lost.	Low	Could lose work and be unable to work.	Low	Ensure work is saved online to google drive cloud service and that code is stored on github. Department PC's should be accessible most days and have all the tools we need.
7	Personal computer crashes while working.	Medium	Potentially will have lose work, meaning you lose time doing it again.	Low	Save regularly, google docs[2] will do this for us. Regularly commit code to personal branches so that it stored elsewhere other than your PC .

Risks with people

ID	Description	Likelihood	Impact	Severity	Mitigation
8	A team member leaves the module or even the course.	Low	They may have only access to their work, also the rest of the team will have more to do.	High	As above store online but also try to keep each other motivated to avoid this.
9	A team member is ill/away for a week or two.	High	They might have been skilled in a certain area that no other member can do well. If they have the only access to work may get behind from it.	Medium	Hard to avoid, but we should store work online where everyone can access. If we work in pairs to complete tasks then there will be less of a chance of having one person who knows the most about one area.
10	Arguments within the team.	Medium	Disrupts the work of the team and prevents us moving forwards. Also, unpleasant for the team as a whole.	Medium	Try to avoid conflict but if necessary have proper debates perhaps using a mediator, do not keep issues hidden.
11	Lack of communication	Medium	Tasks may be done twice or not done at all.	Medium	Keep strong communicatio n using the tools we plan to use.
12	A team member does not do their work.	Medium	Could disrupt other members work and could make the other team members annoyed.	Low	Don’t give members too much work or work they cannot do, ensure that the team communicates well and regularly meets up to discuss how the work is going.

Risks with tools

ID	Description	Likelihood	Impact	Severity	Mitigation
13	Google drive servers stop working.	Low	Could lose/lose access to work that is stored there.	Medium	Store work locally , and on other services.
14	Central git repository [1] is lost in some way.	Low	Temporarily lose access to it.	Low	Keep up to date local copies so can be easily restored. We could host our own local copy should github go down.
15	Website hosting fails.	Low	Users lose access to the website.	Medium	The website files are stored on github and every team member has a local copy of the repository on their computer so we could bring the site back up on a different server. The site is also protected by cloud-flare[3] who will provide a cached version of the site if our host were to go down.

Requirements risks

ID	Description	Likelihood	Impact	Severity	Mitigation
16	Not including a requirement which is required by the customer.	Low	We let the customer down and have failed them.	High	Make sure key requirements are elicited from the customer so they get what they want.
17	A requirement could change / be added.	High	May need to rewrite code or add extra code to account for it. Extra time will be needed.	Medium	Our software architecture must be flexible and able to be changed easily.
18	Stating a requirement that we cannot actually achieve.	High	Let down the customer and also waste time.	Medium	Be sensible when deciding requirements , be sure you can achieve them.
19	Ambiguity in requirements	Medium	May end up making something which is not what was originally intended.	Medium	Ensure requirements are clear and check any ambiguities with the customer.
20	Choosing requirements that the customer doesn’t really want.	Medium	Waste time which could be spent on requirements they did want.	Low	Ensure you know which requirements the customer really wants and which can be ignored.

Estimation risks

ID	Description	Likelihood	Impact	Severity	Mitigation
21	Expect the team or a team member can do more than they actually can.	Medium	Work is not done or is done to an insufficient standard.	Medium	Give tasks that people can do and if they can't then help them. When working on difficult tasks work in pairs to complete the task meaning individual team members don't feel as overwhelmed by the task
22	We may underestimate how long it will take to do some work.	Medium	Work ends up taking longer than expected or not done to the standard it could be done. This could cause other areas of the project to suffer	Medium	Set realistic timings to do work and be realistic on how long a task will take. Account for unforeseen delays in our plan adding time where we can catch up.
23	Be too pessimistic about what we can achieve.	Medium	We end up with a product which is not as good as it could have possibly been.	Low	Push our limits but also stay realistic and within the requirements . If we have extra time then we can use it to enhance the product.
24	Distribute tasks incorrectly.	Low	Team over/under worked.	Low	Distribute tasks appropriatel y and tell others if feel over/under worked.

Bibliography

[1] GitHub [online] Available https://github.com [Accessed 01/11/2016]

[2] Google Drive [online] Available https://www.google.com/drive/ [Accessed 01/11/2016]

[3] Cloud Flare Available[online] https://www.cloudflare.com/ [Accessed 01/11/2016]

Methods and Planning

Overview

We plan to use an agile-scrum approach for our software engineering project, as this will allow our team to carry out work in a flexible manner, evaluating our progress throughout the project, and make any necessary changes to our plan as we go along.

Collaboration & Development tools

Throughout the design and development stages, we as a team will use a wide variety of software and tools to complete our tasks in the most efficient and appropriate way.

Team Communication

We will be using Slack for general team communication, it allows us to have separate chat channels for different aspects of the project, letting us keep all necessary information separated. Since separate discussions are split into separate channels, we are able to find important information more easily for future reference. The team engages in daily communication on Slack[1], we use this to consistently report our progress, ask questions on ambiguous issues and organise future in-person meetings. We plan to use integration between development tools and Slack to help us keep track of progress in one clear feed, in particular the GitHub[2] integration.

Finally, for online meetings we will be using Join.me. Join.me[3] is a free tool with voice, text and video chat as well as a screen sharing feature which allows a presenter to share their screens with other team members. This also allows for control of your screen can be passed on to other team members, allowing them to easily demonstrate ideas and make changes if necessary. Join.me allows efficient and effective team meetings to take place even whilst all members are in separate locations, looking ahead we expect this to be a very useful feature during the Christmas holiday period. An often overlooked benefit of Join.me is that it does not require users who are viewing to install software. This allows us to meet in a virtual environment even whilst travelling since we can simply use a public computer with a modern web browser to take part in the meeting.

Software Development

Through a detailed and constructive debate, the team decided that the project will be constructed using the Java[4] programming language. One reason that heavily influenced our decision was that Java is a language accessible to all members of the team, as well as the entirety of our cohort. This is due to Java being taught in the TPOP module in our First year. This fact is likely be a boon to the team once implementation begins and it should help increase development speed. We did give serious consideration to other programming languages such as C#[5] with Unity[6] however few in our team have any experience in the language thus using it would involve learning a new language which may take longer and cause problems later in the development process. Furthermore, from our understanding the vast majority of our cohort has little to no experience with C# therefore we believe they would be less inclined to choose our project at the end of assessment 2.

Another important reason for choosing Java was that a wide variety of libraries and frameworks for the purpose of game development are already freely available for use – we expect this to further speed up our development process. As different team members prefer different IDEs, we plan to ensure our code base is compatible with both Eclipse[12] and IntelliJ[13].

After engaging in detailed research, we obtained 4 main options of libraries/frameworks we could use: Swing[7], Mini2Dx[8], LibGDX[9], LWJGL[10]. Although LWJGL is a very good library for game graphics development, it is far too low level for us to create a high quality for the game within the time constraints we have been given. Swing can be used for simple 2D games however LibGDX and Mini2Dx provide more flexibility and game specific tools, making development easier. In the end decided to use the LibGDX Java game development framework. We chose LibGDX over Mini2DX because, although Mini2DX is based upon LibGDX it lacks some of the more powerful features that LibGDX has and we didn't want to limit ourselves so early on. LibGDX has support for all of the IDEs that we intend to use, it provides features that simplify several game development steps. One of the key feature is that it has support for tiled maps, we plan to use software called "Tiled"[11] that allows us to easily create a high quality 2D map and export it for use with the framework. This will save us massive amounts of time compared to alternative solutions, this can be used to improve gameplay and ensure that there are no bugs.

So as to ensure work is done in a balanced manner, we will engage in design process whilst the development stages are progressing. We will split the team up based on their skills and their preferences. The main tool we will be using to design the game assets is Piskel, it is a online tool that allows us to easily create game assets. We expect designs to start as paper prototypes that will be developed further as the project goes on. We will also be using Tiled to easily draw maps and rooms.

To ensure that several playability features we are considering are popular with the game's main audience, we conducted a survey using Google Forms asking questions regarding preferred input methods.

Code Management

Throughout the development process, we will be using GitHub as our code repository since it allows for easy collaboration. All members of the team will be contributing to the repository.

To manage development tasks, we plan to use the GitHub projects feature of our code repository. This lets us manage our tasks and code issues with a Kanban[14] board, and allows for items to be assigned to team members so everybody knows who's doing what. This allows us to keep our project management and code in the same place, which should encourage us to effectively keep track of what's happening as we're working on the code. GitHub also allows us to link issues to the project, and these are useful because team members can collaborate on design decisions and keep track of what's happening in the code.

Alongside GitHub projects, we plan to use git's branch and pull request functionality to help us manage multiple developers on the same project. For every pull request, we will ensure that a different developer performs a code review, and that all tests pass before approving the branch.

Tests will be added to our code as we write it, so that we maintain a high level of test coverage. We plan to use a continuous integration server to run all the tests every time a commit is made to GitHub, this will ensure that nobody accidentally breaks the codebase.

Team Organisation

Team organisation model

As mentioned above, we plan on utilising the traditional Agile-SCRUM[16] model when organising our team. In-person meetings are held multiple times a week, here each team member goes through the progress they have made since the previous meeting. This allows all members to have a clear understanding of how the project as a whole is progressing. The meetings also allow us to sort out any queries or worries that members may have. The team will be split into 3 sub teams, and we then discuss what each sub team should accomplish and what tasks each member or sub team should meet by the next team meeting. These constant updates allow us to account for any unforeseen events and alter our development process based on these events.

Team Roles

Through unanimous voting, we chose Brooke Hatton as our group leader. We determined that his prior experience in tasks similar to those we were undertaking and his general teamwork skills made him a natural leader for our group. We may potentially change our leadership during future assessments if the group feels it is necessary.

The reports in assessment 1 are assigned in the following manner:

- Ben Jarvis was assigned the requirements document
- Benjamin Grahamslaw was assigned the risks and mitigations report
- Brooke Hatton and Jason Mashinchi were assigned the Architecture report
- Joseph Shufflebotham and Vishal Soomaney were assigned the Methods report
- Vishal Soomaney was assigned the user stories and scenarios

Each member proofread all of the documents and contributed to all of the documents through comments and small edits.

Team Collaboration

We plan to use the tools above to help us with our collaboration on this project, in particular Slack and GitHub. During our meetings, the team will split up tasks evenly and we will be able to keep track of progress with the Kanban board in GitHub Projects and the Slack integration. As we have been using Github throughout the first assessment, there will be no confusion due to work platform transition when we use Github and Git regularly in the second assessment for game implementation.

During the project, we will determine specific and realistic goals that we expect to meet within certain timeframes. We have chosen to use SMART[16] goals (specific, measurable, achievable, realistic, time-bound) to help us meet our targets. We chose this as it will help us produce useful goals that we can use to guide our work, and meet our deadlines.

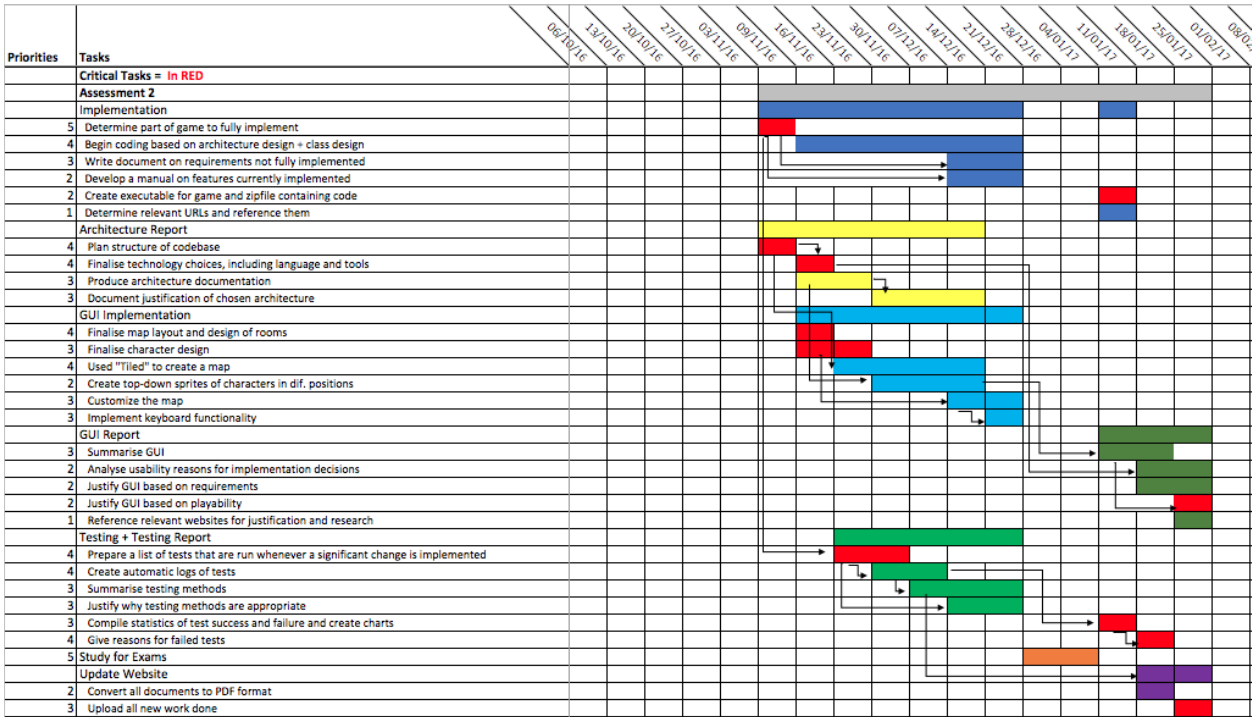
Systematic Plan

The focus of the team will fully switch to the second assessment on Wednesday 9th November, once the first assessment has been submitted. We plan on completing the second assessment by Tuesday Spring Week 2 giving us a week to account for any unexpected developments or fixing issues that happen to arise. It also gives us time to analyse, criticise and improve our own work thus improving the quality of our code and enhancing the functionality and efficiency. This will also ensure that we have very high quality documentation, once all group members are happy with the readability of our code we plan on requesting that students from other teams look at sections of our code to test readability. There will also be a 2 week rest from SEPR to account for time spent studying for exams and also the exam week itself.

A Gantt chart [18] containing the schedule for key tasks from all the assessments can be found in the appendix. This chart includes priorities, task dependencies and a critical path.

Assessment 2 Gantt Chart

Below is an extract of our Gantt chart related to Assessment 2. The full version can be seen on our website.



Priorities	Tasks
	Critical Tasks = In RED
	Assessment 2
	Implementation
5	Determine part of game to fully implement
4	Begin coding based on architecture design + class design
3	Write document on requirements not fully implemented
2	Develop a manual on features currently implemented
2	Create executable for game and zipfile containing code
1	Determine relevant URLs and reference them
	Architecture Report
4	Plan structure of codebase
4	Finalise technology choices, including language and tools
3	Produce architecture documentation
3	Document justification of chosen architecture
	GUI Implementation
4	Finalise map layout and design of rooms
3	Finalise character design
4	Used "Tiled" to create a map
2	Create top-down sprites of characters in dif. positions
3	Customize the map
3	Implement keyboard functionality
	GUI Report
3	Summarise GUI
2	Analyse usability reasons for implementation decisions
2	Justify GUI based on requirements
2	Justify GUI based on playability
1	Reference relevant websites for justification and research
	Testing + Testing Report
4	Prepare a list of tests that are run whenever a significant change is implemented
4	Create automatic logs of tests
3	Summarise testing methods
3	Justify why testing methods are appropriate
3	Compile statistics of test success and failure and create charts
4	Give reasons for failed tests
5	Study for Exams
	Update Website
2	Convert all documents to PDF format
3	Upload all new work done

For the Gantt chart showing the schedule for the rest of the SEPR project, please check appendix B or alternatively you can also see it on the website on this URL: http://lihq.me/Downloads/Assessment1/AppendixB.pdf

Bibliography

[1] Slack [Online] www.slack.com [Accessed 8/11/2016]

[2] Github [Online] www.github.com [Accessed 8/11/2016]

[3] Join.me [Online] www.join.me [Accessed 8/11/2016]

[4] Java [Online] www.oracle.com [Accessed 8/11/2016]

[5] C# [Online] www.msdn.microsoft.com [Accessed 8/11/2016]

[6] Unity [Online] www.unity.com [Accessed 8/11/2016]

[7] Swing [Online] www.oracle.com [Accessed 8/11/2016]

[8] Mini2DX [Online] www.mini2dx.org [Accessed 8/11/2016]

[9] libGDX [Online] www.libgdx.badlogicgames.com [Accessed 8/11/2016]

[10] LWJGL [Online] www.lwjgl.org [Accessed 8/11/2016]

[11] Tiled [Online] www.mapeditor.org [Accessed 8/11/2016]

[12] Eclipse [Online] www.eclipse.org [Accessed 8/11/2016]

[13] IntelliJ [Online] www.jetbrains.com [Accessed 8/11/2016]

[14] GitHub Projects - kanban board [Online] www.github.com [Accessed 8/11/2016]

[15] Piskel [Online] www.piskelapp.com [Acessed 8/11/2016]

[16] Waterfall to Agile: Flipping the Switch - Bhushan Gupta [Online] Available: http://www.uploads.pnsqc.org/2012/papers/t-21_Gupta_paper.pdf [Accessed 25/10/2016]

[17] Goal Oriented Requirements Engineering - Axel van Lamsweerde [Online] Available: http://dl.acm.org/citation.cfm?id=776930 [Accessed 04/11/2016]

[18] Gantt Chart [Online] Available: http://lihq.me/Downloads/Assessment1/AppendixB.pdf

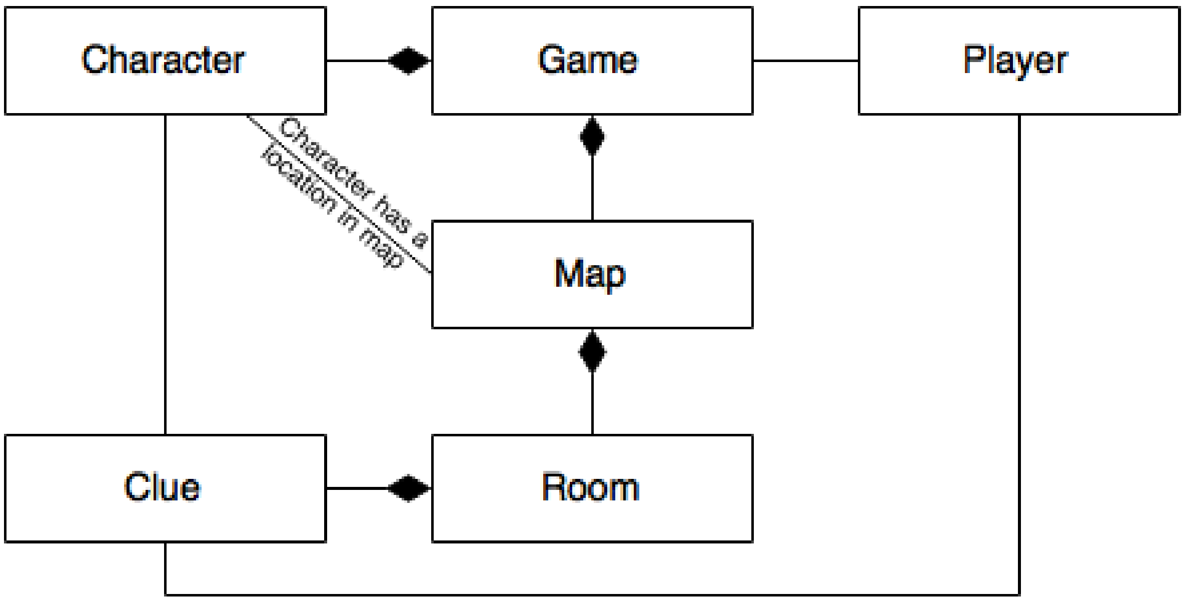
Architecture

Overview

To explain the architecture of our game we decided to use UML 2.X [1] as this allowed us to create a clear diagram of how our game should be architected. We have annotated where possible explaining the relationship between objects and explained some of the attributes. The tool we used to design the diagram was initially pen and paper however we later decided upon the use of Draw.io[2] as this allowed us to collaboratively design the achitechture online.

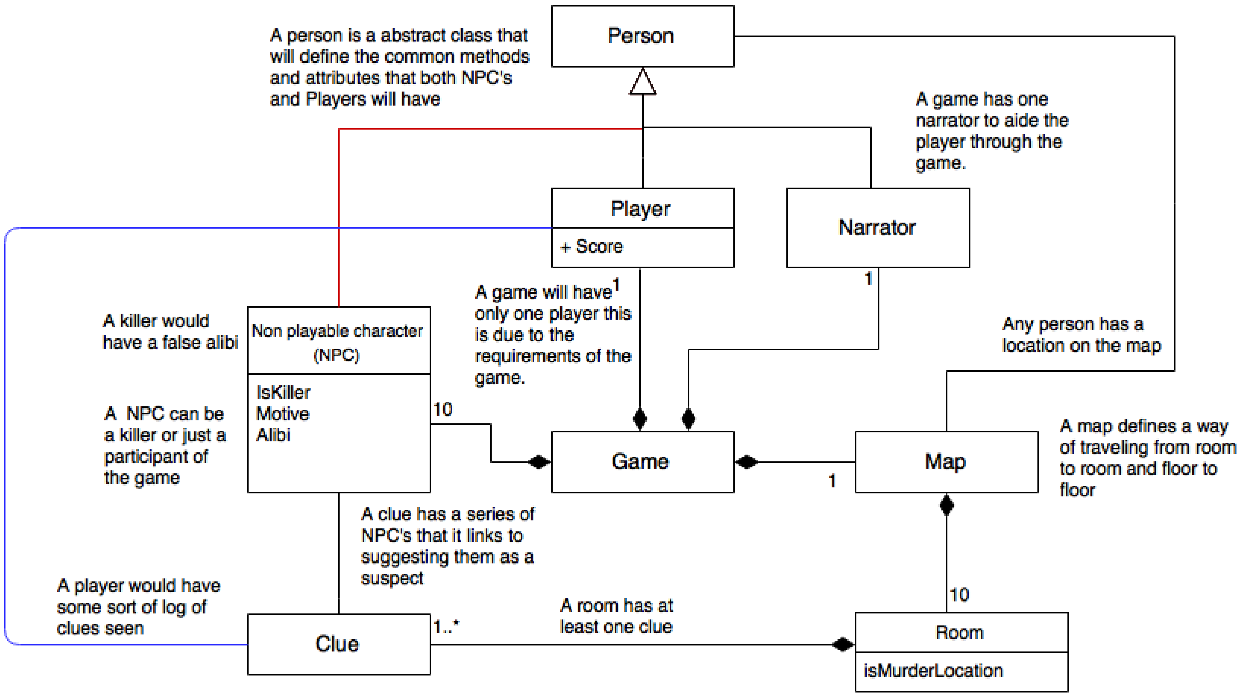
Before starting the design of the architecture we first sat down and worked out how the game would work in practice. Creating scenarios and planning how the game would play through with each scenario. This helped us decide what the main objects of the game would be.

We began designing a simplified UML diagram, showing only the classes and high level relationships, as this allowed us to work out the different objects we needed for the game, and clearly visualise how they relate to each other. This diagram was intentionally kept simple for clarity, and it allowed us to start thinking about the structure of the game in the simplest terms possible. This then gave us a clearer picture of how the structure of the game worked, and allowed us to move on to producing a more complex UML diagram that we can use for reference later on when implementing the game.



The structure that was presented by the simplified UML diagram served as a base for our final UML diagram. The UML diagram was produced based on our previous thinking, and it expanded upon the detail about the components and objects of the game we had planned before. We started map out the key classes and properties, and worked out how we could use abstract classes and inheritance to improve code reuse and aid with the implementation process.

Below is our UML diagram. We've used a few non-standard additions to the diagram to clarify our thinking. We've put comments across the diagram alongside relationship lines and classes to explain what we meant by the relevant parts. Also we have colour coded intersecting lines to show the differentiation between the two (we were unable to get "bridges" over the lines in our diagramming tool!)


A person is a abstract class that will define the common methods and attributes that both NPC's and Players will have
Person
A game has one narrator to aide the player through the game.
Player
+ Score
Narrator
1
A killer would have a false alibi
Non playable character (NPC)
IsKiller
Motive
Alibi
A game will have 1 only one player this is due to the requirements of the game.
Any person has a location on the map
A NPC can be a killer or just a participant of the game
10
Game
1
Map
A map defines a way of traveling from room to room and floor to floor
A clue has a series of NPC's that it links to suggesting them as a suspect
A player would have some sort of log of clues seen
10
A room has at least one clue
1..*
Clue
Room
isMurderLocation

Justification

The game will be made up of many classes, and these classes will have relationships to each other, as described previously. This structure is useful for the implementation process, as it allows relevant parts of the game to be isolated and available for reuse elsewhere. Having a strictly defined structure also aids with maintainability and ability to add new features, as developers can find the code they are looking for.

Game

A game is a class that provides the entry point to the game logic. It contains methods that render and initialise the game ready for use by a user, and manages the state of the game as it progresses. This class is necessary to contain the game logic, and will provide a common area to place shared game logic.

Person

A person is an abstract class that will define the common methods and attributes that both NPC's (see below) and a player will have. These include a relationship to the map to allow for common location attributes and methods to be shared, and other rendering methods. This will help prevent unnecessary code duplication, this will also allow us to expand the game further in the future if required adding new people that require the same functions.

Player

The game has one player, which represents the user interacting with the game. The player class extends the person class to allow for code reuse. The player contains attributes such as their score, and contains a list of clues the player has picked up. A player can also be in a room within the map. This class is necessary to contain logic related to the user, and to keep relevant user details and status for use within the game.

The player will contain details about previous types of questions asked to NPCs, this is to allow the development his/her personality as the game goes on.

Non-Playable Character (NPC)

A non-playable character (referred to as NPC) is one of the fictional people represented within the game. There will be 10 NPCs in the game. All NPCs have associated clues that point to them as a suspect.

At the start of a game, each NPC will be associated with a room within the map, and they can be questioned in that room by the player. The NPC contains the questioning data, including the possible questions that a player can ask and the NPC's response.

For every game, an NPC is selected as the killer from a subset of the NPCs that can be killers. The same process happens for selecting the victim. A subset of all the NPCs can be possible killers or victims - this is to reduce the number of storylines that need developing, and can be expanded later on.

All NPCs have a motive (a reason why they would kill the victim) and all NPCs except the killer have an alibi (what they were doing at the time of the murder). This is to allow the game to change the murder scenario every time it is played.

Narrator

There is one narrator in the game, we chose not to have the narrator as another NPC in the game as the narrator serves a different purpose, requiring different methods and attributes to a NPC. The purpose of the narrator is to introduce the game to the player, the control scheme and instructions on how the game works.

Map

This is a representation of the map within the game that a player can navigate around. A map contains many rooms arranged in a specific way, and provides methods and attributes that allow for navigation. There will be 10 rooms in a map, however the way the map class is designed it will allow more rooms to be added or taken away easily.

Rather than giving the game a series of rooms we decided to create a map class that would handle how these rooms linked to each other, this makes it easy to provide methods to the game and the player that mean if we were to add new rooms or change the map the game logic wouldn't need to be altered.

Room

A room is a place within a map that a player can visit. This class will contain methods and attributes that will allow other parts of the game to interact with the room where necessary.

At the start of the game, each room has a random NPC assigned to it, and one or more clues associated with the killer NPC assigned to it. The player can visit the room to gather information by talking to the assigned character, or finding the clues located in the room.

Clue

A clue is an object within the game that allows the player to narrow down their list of suspect NPCs. Every NPC has a set of clues, and these point to the NPC as a suspect. Some of these clues are shared amongst multiple characters, for example "a blond hair", that will allow the player to narrow down the set of suspects.

At the start of each game, every clue associated with the killer NPC is assigned to a room, so when the player visits a room they can collect any clues located in the room.

The clue class is essential to the game as it provides the mechanism for deducting which NPC is the killer in the game.

Bibliography

[1] OMG Unified Modeling Language TM (OMG UML) Version 2.5 [Online] Available http://www.omg.org/spec/UML/2.5/PDF/ [Accessed 25/10/2016]

[2] Draw.io, "Flowchart Maker & Online Diagramming Software" [Online] Available https://draw.io/ [Accessed 25/10/2016]

A: User Scenarios

Persona 1

David, 18 year old male at a Computer Science Open Day. He is not an avid gamer and his interests lie more in electronics.

Scenario

David is initially unsure as to how the game is played, however he is provided a tutorial by the narrator which explains the puzzle, how he is expected to solve the puzzle, how to move the main character, interact with clues and non player characters. After becoming accustomed to the keys and finding a clue, he is unsure as to what the next step is to solve the puzzle. He interacts with the narrator who subsequently provides hints as to the next course of action that will get the user closer to solving the puzzle. Near the end of the game he becomes stuck and spends a long time attempting to find out which of his 3 final suspects is the murderer, the narrator pops up on the side of the screen to ask whether the player requires assistance, David says "yes" and is told to speak to a character which holds the final piece of information required to solve the puzzle.

Persona 2

Louise, 17 year old female student at an open day. Enjoys playing puzzle games and is colour-blind.

Scenario

Although she is initially worried about missing a clue or finding it hard to read some of the text, she is able to go through the introductory tutorial without issue due to the text colour and background colour contrast in a way to avoid issue with most forms of colourblindness. She is able to see the clues quite clearly due to map tile colours and clue item colours being chosen in a way so as to not cause problems to most people will colourblindness. She is then able to solve the crime quicker than most and accuse the correct murderer of the crime on her first try thanks to her experience played puzzle games.

Persona 3

Ben, 19 year old second year Computer Science student. He is an avid gamer who takes games quite seriously and always tries to get the high score.

Scenario

He progresses through the tutorial quickly and is already accustomed to using keyboard keys to interact with the game. As he attempts to concentrate and solve the puzzle as quickly as possible so as to get the high score, the background noise begins to bother him. Fortunately he notices that there is a "turn music off" button in the sidebar of the game and this allows him to continue playing comfortably. Although he finishes the game with a good score, he is unable to beat the high score and wishes to play again, fortunately due to the game having a set of different characters from which murderers and victims are chosen randomly, he can play several times whilst going through a new, different and exciting storyline each time.

B: Project Plan (Gantt Chart)

We've produced a systematic plan for future assessments. This is in the format of a Gantt chart, created in Excel.

Download (AppendixB.pdf)

C: Survey Results

Figure 1

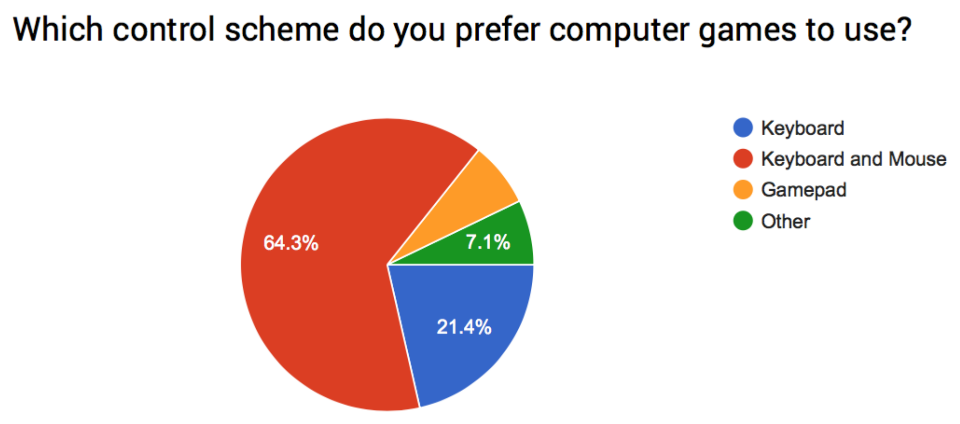
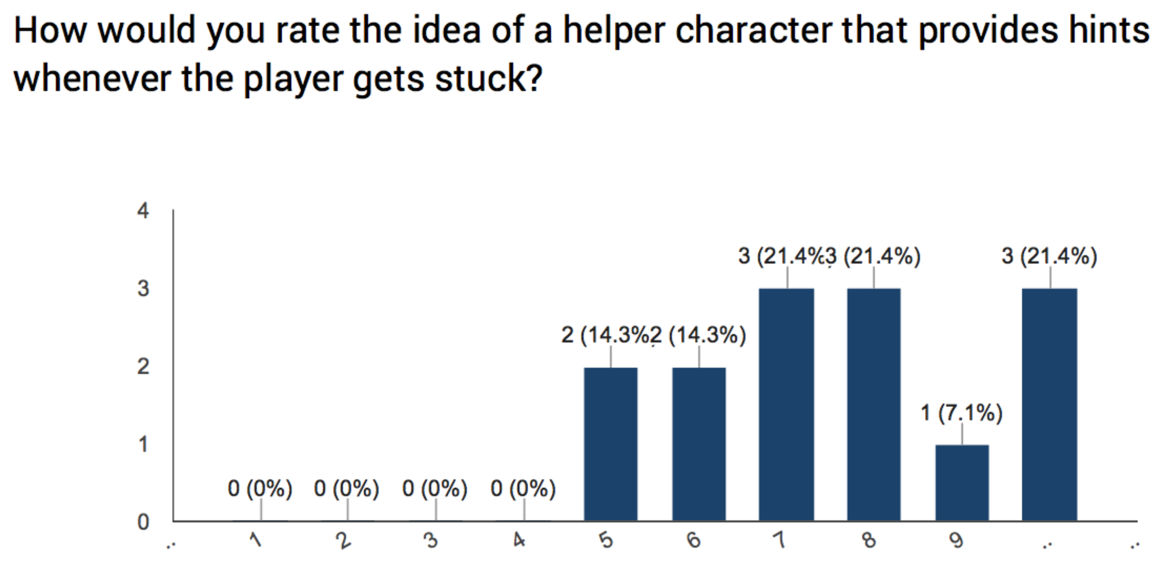
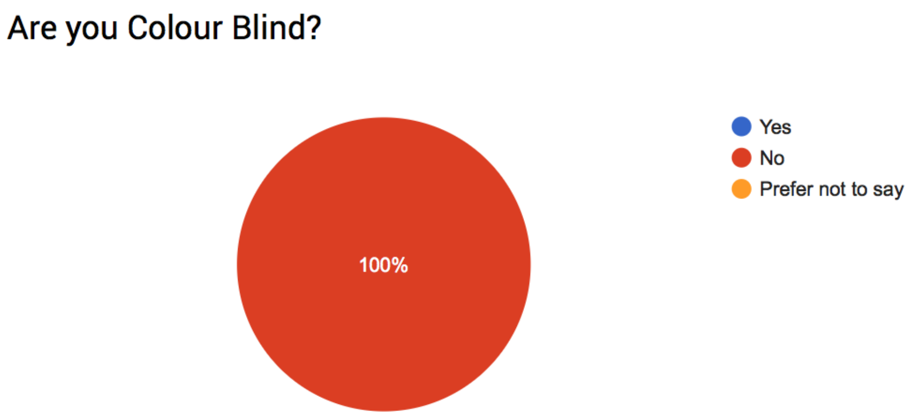


Figure 2



Bar Chart for Helper Character 1 = not helpful 10 = extremely helpful

Figure 3



Pie Chart Colour blind.