# 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 the 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:
  C*haracter_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]