Architecture

3.1 Conceptual models

We have created 2 separate conceptual models for our architecture: a flowchart [1] to describe the flow of the game from the player's perspective, and a UML Class diagram [2] to describe the structure of the game. These models are visualisations to help us and others to understand the structure of our software both in a literal sense of code modules and classes, but also in terms of the flow of the software experience. The flowcharts help our design to be more user-centred, and they will serve as a continuous reminder of the direction that development should progress towards.

These models are intended to be abstract, and can be adjusted if necessary in accordance with our Agile work approach. For example, the methods shown in our UML class diagram are not concrete and will be subjected to change as our project develops and requirements change. For the most part, the methods in these classes are generalisations, and will consist of many smaller methods when implemented. Similarly, the attributes of these classes can also be amended in the future should requirements suggest it. Our flowchart is simplistic and has deliberately ignored some exceptional cases and routines, (for example, any menu input other than 'Start Game'), as we intend it to be used as a guide to the user's experience rather than a concrete structure or plan to follow.

**We have abbreviated Ron Cooke Hub to "RCH"*
**We have abbreviated Non-Player Character to "NPC"*
**Requirements are numbered. Associated system requirements are the number followed by a letter.*

3.1.1 Tools used to create architecture representation

The class diagram and flowchart were both created in Microsoft Visio and outline the architecture for the game. Microsoft Visio was used because the team has previous experience with the program, and it provides incredibly useful templates to make UML Class diagrams and flowcharts. This meant that less time was spent learning an obscure software package and more time was spent planning and building the diagrams.

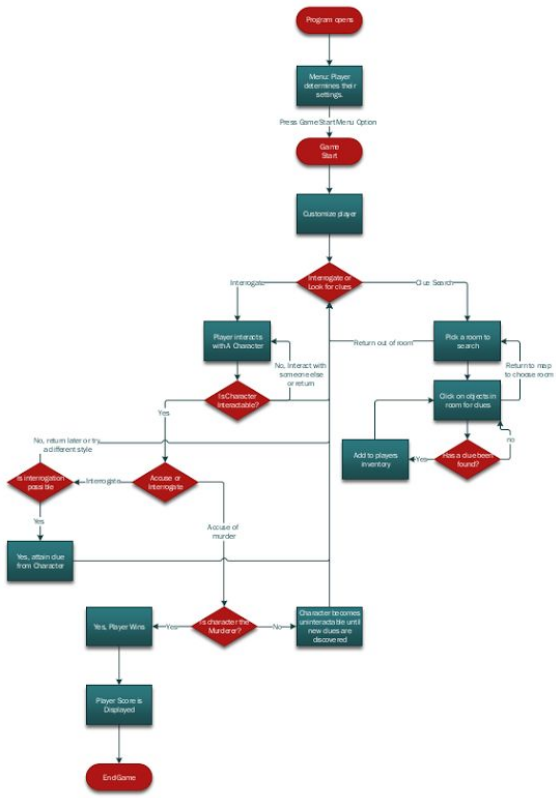

Figure 1: Gameplay Flowchart [1]

3.1.2 Gameplay Flowchart

The flow chart depicted in Figure 1 (above) starts at "Game Start"; a state that assumes the user has successfully launched our software, and is presented with a main menu. When they select "New Game" the user will be prompted with a brief introduction, which will include information such as the victim's name and the option for the user to manipulate their detective's personality. The player will then have the freedom to move around the Ron Cooke Hub, and decide who to interrogate or where to search for clues. The two mechanisms for gaining clues are somewhat dependent on each other, and should not be seen as entirely separate processes (although the user has complete freedom to move where they want when they want, and interview who they want in any order). Some more detailed information on how these processes will work from the user's perspective is depicted below.

- Interrogation
    - The user will move their detective around the RCH until they find the NPC that they wish to interrogate. If the NPC has recently been falsely accused of murder then the player will be unable to interact with them.
    - Once the player has found an NPC with which they can interact and they open up dialogue, the detective will have several different methods of interrogation to choose from depending on their personality. Through this kind of interrogation, the user can gain clues and information.
    - If the player thinks they have gathered enough information, for example the murder weapon and the motive clue, then they can accuse the NPC of the murder.
    - If the player accuses the correct character, they have won and the game will end. It will then show the user their score and the overall story of the murder. If they are incorrect then their detective will be “locked out” from the NPC they accused, who can no longer be questioned until new evidence is found by the player/detective. The detective will then exit the dialogue and return to the room they were in.
    - The interrogation will depend on the interrogation style, the character’s personality and the clues found so far. Certain NPCs will give the player different information depending on the interrogation style.

- Clue Search
    - The player will move from room to room and can find clues by inspecting furniture and objects within the room by clicking on them with the mouse. If the detective discovers a clue it will be added to the player’s inventory, where it is stored to be examined for future reference. These collected clues can then be used while questioning or accusing characters around the RCH.
    - The player will find a motive clue and a weapon, both of which are essential to the plot of the story, and to a correct accusation.

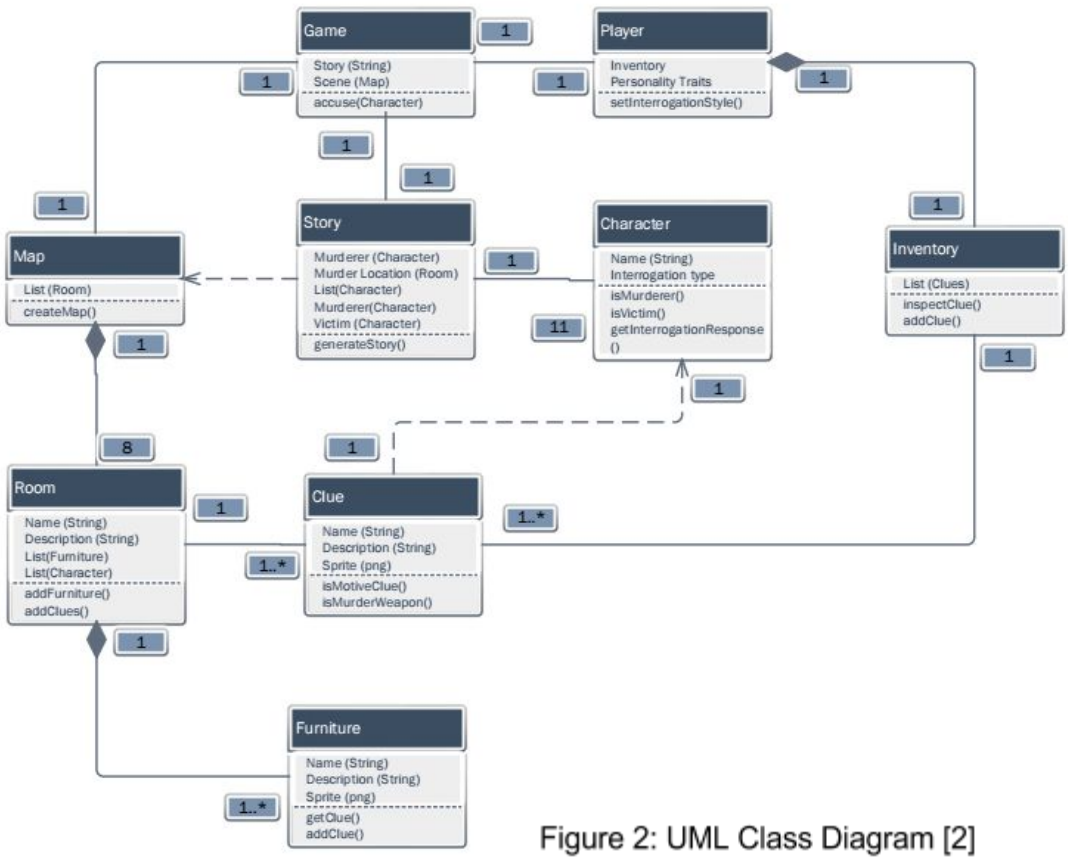

Figure 2: UML Class Diagram [2]

3.2 Justification of conceptual models

The conceptual models were made while taking into consideration our requirements [3]. These requirements are listed below and with them is a brief description as to how this affected our design choices.

Req 1: "Each playthrough should be different in some way to the user."

Many aspects of the game have been split out into their own classes that will help facilitate the process of creating a unique game for the player. Characters, Clues, Rooms and Furniture will all be instanced randomly at the beginning of the game from a pool of possible types. These elements will be linked using the Story class. The Story class itself stores:

- 10 characters, including the murderer and the victim as well as the other 8 NPCs who provide clues. (System Requirement 8a)
- Murderer - randomly allocated from set of 10 characters.
- Victim - randomly allocated from set of 10 characters.
- Murder location, randomly chosen from the set of rooms.

The function generateStory will create the story of the game from these details, including generating relevant clues and, in accordance with Other System Requirement 18 (*"The system should randomly distribute the clues when the game is restarted."*) and System Requirement 1a, putting them in furniture and characters, and putting all of the characters and furniture in rooms.

Req 4: "*The user must be able to customise the personality of their character."*

As described in the flowchart [1], when the player starts a game they will be prompted to customise their detective and choose their interrogation style (System Requirement 4a). The personality traits of the user will be stored in the Player class.

Req 5: "The user must be able to navigate to all rooms in the RCH."

The Map class will store all the rooms for that instance of the game. The room type, ordering and location will be randomly generated at the start of the game. The Map class is designed so that the user does not interact directly with a room but instead it goes through the map which then interacts with the room.

Req 6: "The user should be able to find and collect clues within the Ron Cooke Hub."

Clues will have their own class containing their names, descriptions, and sprites. Unlike the previous classes, these are modified at initialisation to be in more in line with the murderer (for example a "bloodied hair" might be a "bloodied blond hair" if the blond-haired "Mr Smith" was the murderer). Exactly one clue will be a motive clue, and exactly one clue will be the murder weapon (setting the responses of isMotiveClue and isWeapon respectively).

Searching for Clues is outlined in the flowchart. The Rooms can have Furniture objects assigned to them and also may have Clues that are in plain sight. When a piece of Furniture is instantiated it may be randomly assigned a Clue object for the player to find (System Requirement 6b). When the player interacts with the Furniture in the game and finds the Clue, the function getClue will add the clue to the player's inventory and remove it from the furniture (System Requirement 6c).

Req 7: "The user should be able to review the evidence that they have already found."

The Inventory class will store the clues that the user has found so the user can keep track of them. The inspectClue method is used to interact with the stored clues so the player can see the information relevant to each clue.

Req 8: "The user should be able to see and interact with other game characters and objects to help them in their quest for evidence."

Each Room object will contain a name, description, Furniture, and Characters. Storing Rooms as separate objects is necessary as the details of each room will be generated at runtime, and will be completely different for each room. The Furniture and Character objects may contain clues. Clues will be added during the creation of the rooms via the randomly called addClues subroutine. Functionally, Characters will work similarly to furniture; they may or may not hold a clue (System Requirement 8c). To obtain the Clue, the player will need to interrogate them.

Req 9: "*The user should be able to question NPCs in different ways, and will receive varying amounts of help from them, depending on their question choice."*
NPCs will be represented as Character objects within our game, and each will have several basic properties including their name, description and spritesheet. They will also contain some less basic attributes, such as the location of the file they will get their responses from, and how they will react to different kinds of interrogation. Each NPC will store a flag as to whether they have been incorrectly accused of the murder. This flag will denote whether the NPC is able to be interrogated by the detective or not. If an NPC is accused of murder, its flag will set to *true* and would be reset once new evidence has been found (System Requirement 11a). As mentioned above, the responses that the NPC gives will vary depending on the questioning style used (System Requirement 9d), and if the player uses the right investigation style for the NPC, they could also obtain a clue.

Req 10: "The user should be able accuse NPCs of being the murderer, with a motive clue and a weapon".
The player can try to accuse NPCs of being the murderer at any point (system requirement 10a), but will only be successful if they have at least a certain number of clues, including the motive clue and the murder weapon. If the player meets the basic clue requirements (having found the motive clue and murder weapon, System Requirement 12b) and has correctly identified the murderer, then the player beats the game (System Requirement 12a) and the end state of the game is triggered. If they are incorrect or they do not have the required clues, the NPC accused becomes non-interactable (System Requirement 10b) until the player obtains another clue, at which point the player can interact with the NPC again.

System Requirements:
In our architecture, the Game class will be the overarching class from which the game runs. This class will contain all the appropriate instances that are needed for the playthrough and methods to reset the game as required in the system requirements. The Game class will handle accusations, and will contain all of the main loops that the game runs on, including rendering and input handling.

3.3 Bibliography

[1] H. Cadogan, W. Hodkinson, A. Pearcy, T. Fox, S. Davison and C. Hughes, "Flowchart", Wedunnit!, 2016. [Online]. Available: https://henrycadogan.github.io/Clued-up/ass1/webfiles/Flowchart.pdf [Accessed: 8- Nov- 2016].

[2] H. Cadogan, W. Hodkinson, A. Pearcy, T. Fox, S. Davison and C. Hughes, "ClassDiagram", Wedunnit!, 2016. [Online]. Available: https://henrycadogan.github.io/Clued-up/ass1/webfiles/ClassDiagram.pdf [Accessed: 8- Nov- 2016].

[3] H. Cadogan, W. Hodkinson, A. Pearcy, T. Fox, S. Davison and C. Hughes, "Req1", Wedunnit!, 2016. [Online]. Available: https://henrycadogan.github.io/Clued-up/ass1/webfiles/Req1.pdf [Accessed: 8- Nov- 2016]