Architecture Report

2.1 Concrete Architecture - Structure of Code

One major change we made in between Assessment 1 and the implementation in Assessment 2 was the software tools and IDE used to develop our software. Initially we planned on using Java, however we found that using the Unity Engine [1] with C# scripts would help our game to be much more advanced in the allowed time constraint. Unlike Java, C# heavily favours component-based architectures and thoroughly relies upon object composition over inheritance. For example, *every* object in the Unity editor is of class *GameObject,* and other types of object are created through the composition of *GameObject* with other classes. As a result, the structure of our software is highly dependant upon the inner workings of the Unity Engine and as this part of the code is abstracted from us we have elected not to describe any components (including Event Triggers, Sprites, and UI elements) in this document - they would only make sense in the context of the Unity project, which is far too large to document in its entirety.

We have, however, detailed the operation and structure of our C# scripts, which is best interpreted by examining the following diagrams. We chose to follow the UML 2.5 [2] standard in our diagrams as it is widely known and simple to interpret, yet can represent complex structures.

As the Unity Engine automatically destroys all objects within a scene when a new scene is loaded, we use the DontDestroyOnLoad method to preserve certain objects so they persist throughout the game. This is declared within the 'Story.Awake()', 'Character.Awake()', Detectives.makePersistant.Start() methods, resulting in a maximum of one continuous instance of each of those classes. As a result of this, when the following explanations of our concrete architecture reference 'the Story' or 'Detective' (and by composition, 'Inventory'), there is only a single object to which they could reference.

The linked UML Class and Sequence diagrams explain both relationships and data-flow between classes/ objects. They were created in a free software package named StarUML [3], which allows simple and quick linking of classes and addition of properties/ methods.

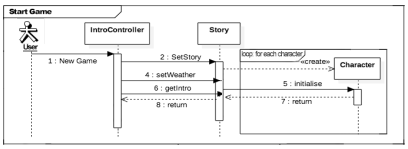

Figure 1. UML Sequence diagram for when the player starts a new game (clicks 'New Murder' from the main menu). View larger version at [http://wedunnit.me/webfiles/ass2/sequence-diagram-new.png].

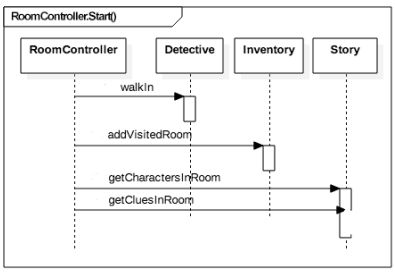

Figure 2. UML Sequence diagram for when the detective walks into another room (RoomController.Start(), triggered automatically by the Unity Engine). View larger version at [http://wedunnit.me/webfiles/ass2/sequence-diagram-room.png].

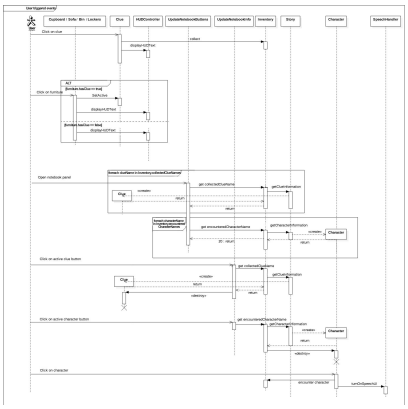

Figure 3. UML Sequence diagram for when the player starts a new asynchronous event. View larger version at http://wedunnit.me/webfiles/ass2/sequence-diagram-event.png].

Figure 4. UML Class Diagram available at: http://wedunnit.me/webfiles/ass2/class-diagram.png

The basic structure of our code can be explained in terms of *Story*, *RoomController*, *Character* and *Clue* classes. When a new game begins the *Story* object is instantiated *Story.setStory()* is called (see *Figure 1*) and sets a reference to which characters are in each room, which clues are in each room etc. Each Unity scene contains an instance of *RoomController*, that (depending on which scene it is instantiated from) instantiates the relevant *Clue* and *Character* objects by calling *Story.getCluesInRoom()* and *Story.getCharactersInRoom()* with appropriate arguments (see *Figure 2*).

Most of the other features of the game are started by Events including mouse clicks & keyboard presses. By default, Unity automatically provides event handlers to each scene, and any *GameObject* with an attached '*Collider'* component will automatically trigger the OnMouseClick() method of any *Script* component attached to the *GameObject.* A character is represented in a scene as a *GameObject* with attached *BoxCollider* component and *Character* script component that handles any clicks to the object. Clue objects are represented in scenes in an identical fashion, except the script is the Clue class. See *Figure 3.* for examples of the sequence of calls when certain events are triggered.

2.2 Concrete Architecture - Systematic Justification

How concrete architecture builds from abstract architecture:

Our conceptual architecture [4] was a very brief outline of our initial design idea, where the primary focus was to ensure we would meet all our requirements [5] and not neglect important features of our game. Abstract architecture is very appealing initially, as it does not go into much detail and therefore flexible, however it cannot be used to demonstrate the true implementation of a system. Our concrete architecture is an in-depth representation of the structure of our code, and builds upon our abstract model by adding properties and methods to each of the classes that exist in our code. Naturally, during the implementation phase, many features and ideas morph/ adapt for either practical or visionary reasons, causing the original model to no longer be relevant.

Changes made from abstract architecture:

There are several reasons for the changes that have occurred in our architectures. Obviously, changing from a Java to a Unity project plan yielded the largest change in architecture as the whole philosophy and structure of the environment was overturned. Another reason for the changes were that some of the required methods and associations were overlooked in the conceptual architecture, for example the *Room* class did not have an association with *Story*, despite each Room depending on certain elements of the *Story*.

The following is a list of classes in our conceptual architecture that have been changed in some way in the concrete architecture, followed by a reason why.

The *'Map'* class has been renamed to *'MapController'* and now only associates with the *Detective* class, as the *Map.updateMapButtons()* method requires access to the *Story.visitedRooms* property. This allows the user to navigate around the RCH, and fulfils system requirement 5c.

The *'Player'* class has been renamed to *'Detective'*. This is justified because the human player triggers events in a *'Detective'* instance. The *'Detective'* class instantiates the *'Inventory'* class as a component of itself, which stores collected Clue.name & Character.name properties.

'UpdateNotebookButtons' & *'UpdateNotebookInfo'* classes were added to our architecture to allow the user to review their progress. These classes are added as a component of certain elements within the HUD in the Unity editor and are triggered in a similar way, fulfilling system requirement 7a.

The *'Room'* class was renamed *'RoomController'* as the room itself would be encapsulated in a Unity scene. Each room scene in Unity contains a *RoomController* script such that *RoomController.Start()* is triggered when the scene is loaded. It calls a *Detective* object to walk in, gathers data from *Story* and randomly initialises relevant furniture, Clue, Character objects that should be within each room, as in *Figure 2.* It also assigns

locations for Clue and Character objects in the scene, making each playthrough different as of system requirement 1.

The *'Story'* class is instantiated in the 'IntroAndInit' scene in our Unity Project with a call to the *'Story.setStory*()' method that decides upon properties of the murder fulfilling system requirement 1a. The *Story* class also holds constructors for all possible *Clue* objects in *'Story.getClueInformation*()'. Within the *Story* there are a total of 8 Clue objects, thereby fulfilling system requirement 6, which are randomly distributed in *Story.setClueLocations()*, which fulfils system requirement 18. These clues are initially set by *'Story.clueNames'*, and are instantiated by *Story.getClueInformation()* on request of *RoomController* or *UpdateNotebookInfo*.

The *'Game'* class was removed because of our switch from a Java based solution to Unity. The game loop is abstracted away and handled entirely by the Unity Engine.

'Furniture' in our abstract architecture has been replaced by separate *Bin*, *Sofa*, *Lockers* & *Cupboard* classes (instantiated in *RoomController*) where a *Clue* may be hidden in the game. This fulfils system requirements 6a and 8b. The change was made for simplicity and ease of programming, and we planned to combine these into a single *'Furniture'* class, however our time constraints led us to prioritise other aspects of the game that would push our game closer to our client's mental model and satisfy the requirements, rather than improving the structure of incredibly simple classes.

The *'SpeechHandler'* class was added to allow the player to interrogate characters, and provides a mechanism for reading dialogue text documents in *'ImportSpeech'* that respond to user inputs in the UI. This fulfils requirement 8b, 9a, 9b, 9c, and 9d, and is triggered from the *'Character.OnMouseClick'* event.

The *'HUDController'* class was added to help reduce the presence of a so called 'God Class', and to further separate independent aspects of the software. A *HUDController* component is present in every room of the game and controls all interactive panels and buttons (eg. Notebook or Map) that can be clicked on by the user fulfilling system requirement 5b.

[1] "Game Engine," *Unity*. [Online]. Available: https://unity3d.com/ . [Accessed: 24-Jan-2017].
[2] "UML 2.5 Diagrams Overview", *Uml-diagrams.org*, 2017. [Online]. Available: http://www.uml-diagrams.org/uml-25-diagrams.html . [Accessed: 23- Jan- 2017].
[3] "StarUML", StarUML. [Online]. Available: http://staruml.io/ . [Accessed: 24-Jan-2017].
[4] H. Cadogan, S. Davison, T. Fox, W. Hodkinson, C. Hughes and A. Pearcy, "Architecture Report", *Wedunnit!*, 2016. [Online]. Available: http://wedunnit.me/webfiles/ass1/Arch1.pdf . [Accessed: 24- Jan- 2017]
[4] H. Cadogan, S. Davison, T. Fox, W. Hodkinson, C. Hughes and A. Pearcy, "Software Requirements Specification", *Wedunnit!*, 2016. [Online]. Available: http://wedunnit.me/webfiles/ass1/Req1.pdf . [Accessed: 24- Jan- 2017]