Architecture Report

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.

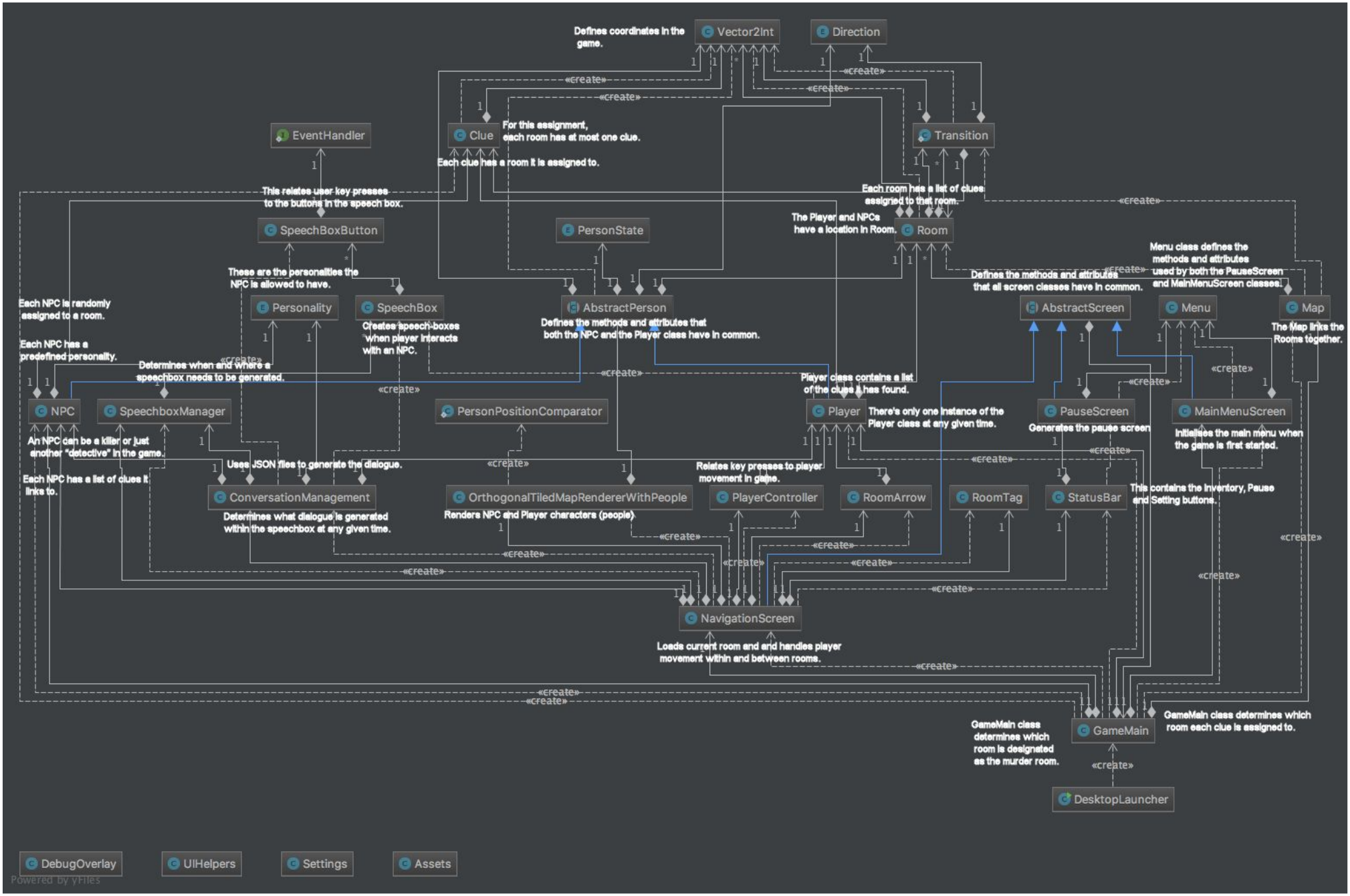
We decided to use a class diagram instead of an object diagram since we wanted to show the classes our project consists of and what those classes were each capable of (methods). The object diagram would instead have shown the interaction between the classes at some point in run time.

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


Defines coordinates in the game.
Vector2Int
Direction
«create»
EventHandler
Clue
For this assignment, each room has at most one clue.
Each clue has a room it is assigned to.
Transition
This relates user key presses to the buttons in the speech box.
Each room has a list of clues assigned to that room.
SpeechBoxButton
PersonState
The Player and NPCs have a location in Room.
Room
Menu class defines the methods and attributes used by both the PauseScreen and MainMenuScreen classes.
These are the personalities the NPC is allowed to have.
Defines the methods and attributes that all screen classes have in common.
Each NPC is randomly assigned to a room.
Personality
SpeechBox
AbstractPerson
AbstractScreen
Menu
Map
Creates speech-boxes when player interacts with an NPC.
Defines the methods and attributes that both the NPC and the Player class have in common.
The Map links the Rooms together.
Each NPC has a predefined personality.
Determines when and where a speechbox needs to be generated.
Player class contains a list of the clues it has found.
NPC
SpeechboxManager
PersonPositionComparator
Player
There's only one instance of the Player class at any given time.
PauseScreen
MainMenuScreen
Generates the pause screen
Initialises the main menu when the game is first started.
An NPC can be a killer or just another "detective" in the game.
Uses JSON files to generate the dialogue.
Relates key presses to player movement in game.
Each NPC has a list of clues it links to.
ConversationManagement
OrthogonalTiledMapRendererWithPeople
PlayerController
RoomArrow
RoomTag
StatusBar
This contains the Inventory, Pause and Setting buttons.
Determines what dialogue is generated within the speechbox at any given time.
Renders NPC and Player characters (people)
NavigationScreen
Loads current room and and handles player movement within and between rooms.
GameMain class determines which room is designated as the murder room.
GameMain class determines which room each clue is assigned to.
GameMain
DesktopLauncher
DebugOverlay
UIHelpers
Settings
Assets
Powered by yFiles

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 same 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, *ConversationManagemen*t *[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 *PlayerController[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 *Narrator[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 *SpeechBoxButton[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: http://lihq.me/docs/JavaDocs/me/lihq/game/people/NPC.html [Accessed: 23/01/2017]
[18] JavaDocs reference to PlayerController class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/people/controller/PlayerController.html [Accessed: 23/01/2017]
[19] JavaDocs reference to Clue class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/models/Clue.html [Accessed: 23/01/2017]
[20] Reference to Narrator from original architecture [Online] Available: http://docs.lihq.me/en/1.0.0/architecture.html [Accessed: 23/01/2017]
[21] JavaDocs reference to Map class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/models/Map.html [Accessed: 23/01/2017]
[22] JavaDocs reference to Room class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/models/Room.html [Accessed: 23/01/2017]
[23] JavaDocs reference to Menu class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/elements/Menu.html [Accessed: 23/01/2017]
[24] JavaDocs reference to StatusBar class [Online] Available:

http://lihq.me/docs/JavaDocs/me/lihq/game/screen/elements/StatusBar.html [Accessed: 23/01/2017]
[25] JavaDocs reference to SpeechBoxButton class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/elements/SpeechBoxButton.html [Accessed: 23/01/2017]
[26] JavaDocs reference to UIHelpers class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/elements/UIHelpers.html [Accessed: 23/01/2017]
[27] JavaDocs reference to GameMain class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/GameMain.html [Accessed: 23/01/2017]
[28] JavaDocs reference to DebugOverlay class [Online] Available: http://lihq.me/docs/JavaDocs/me/lihq/game/screen/elements/DebugOverlay.html [Accessed: 23/01/2017]