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].