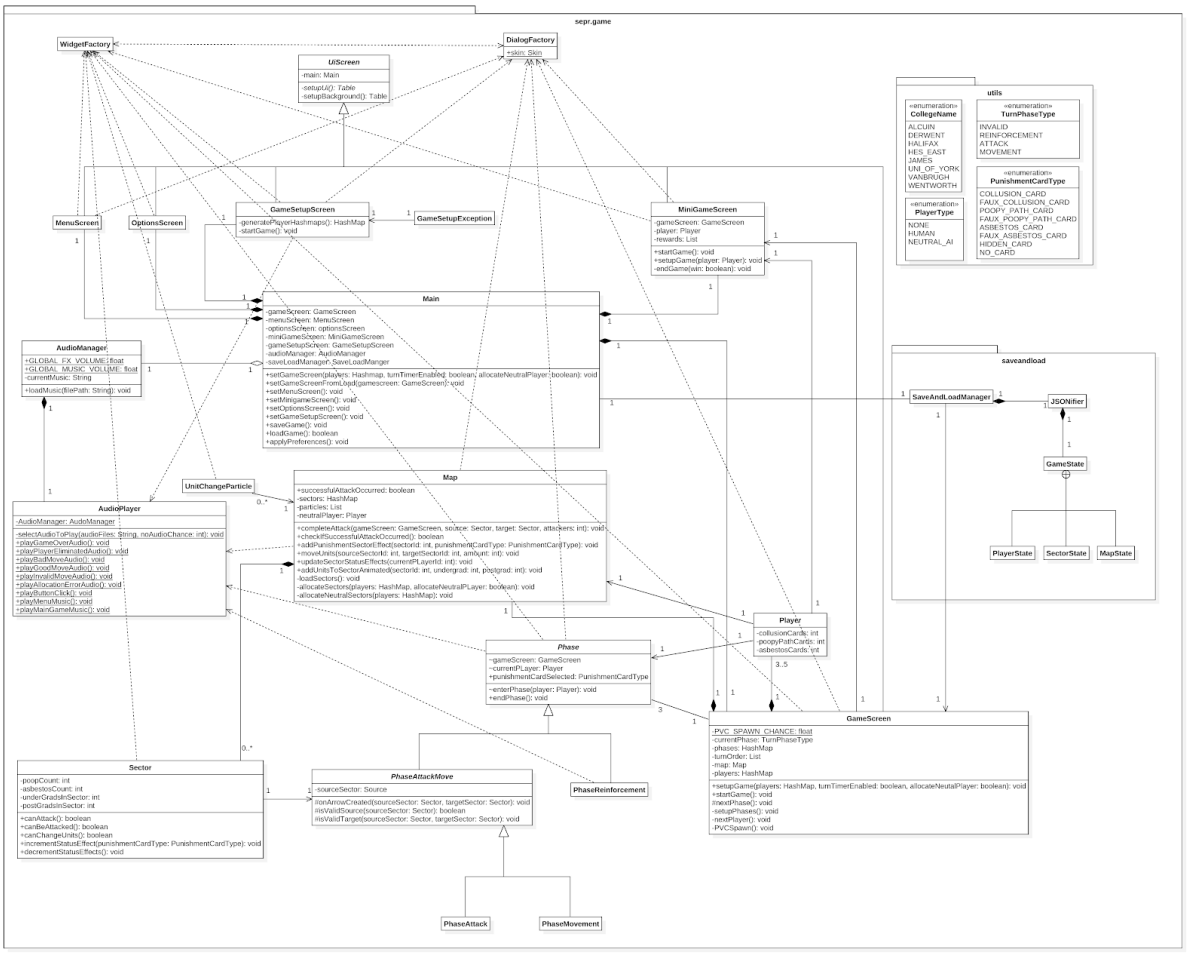
Architecture and Traceability Report

Diagram



The diagram above shows the concrete architecture of the game. Fields and methods key to the program's behaviour are included and less significant implementation details have been omitted to help readability. A full resolution version of this diagram is available on our website [1]. The diagram was produced using StarUML [2] and was designed using the UML 2.0 specification [3].

Architecture Justification

Overview of Architecture

The entire program is built around the Main class using five screens: MenuScreen, OptionsScreen, GameSetupScreen, GameScreen and MinigameScreen. The five screens inherit from UiScreen which is a class for performing the basic setup of the screens and they all store the same instance of Main which they can use to swap between each of the different screens. Being able to switch between the various screens is crucial to the design

of the program as it allows the user to be shown relevant information at the appropriate times. In particular the system is in place to satisfy requirement F15 by allowing the transition from the MenuScreen to the intermediary GameSetupScreen and then into the GameScreen to start a new game. Furthermore, the system is key to the implementation of the Pro-Vice-Chancellor minigame by enabling the transition from the GameScreen to MinigameScreen and back.

The core game logic is within the GameScreen class which is composed of “map: Map”, “players: HashMap” and “phases: HashMap”. Between these three elements the game state is represented and the gameplay is controlled.

The Map contains a HashMap of Sectors. These Sectors store game state and also key data for rendering the game map to the GUI, as required by F13.

The Player class stores data unique to each player in a game, for example their name and what college they belong to. This data is then used to construct the game’s GUI so that it is clear to users which sectors belong to which players.

The phases HashMap contains three objects an instance of PhaseReinforcement, PhaseAttack and PhaseMovement. These classes define how the program should react to player input and then call the appropriate methods in associated classes in order to handle these inputs.

Justification of Changes

Screens

In the code base we received the five screen classes: MenuScreen, OptionsScreen, GameSetupScreen, GameScreen and MiniGameScreen were all implemented independently despite containing lots of duplicate setup code. Therefore UiScreen was created which the other screens could inherit from. Due to this change large parts of duplicate code were able to be removed and all screens are setup in the same way so that any changes made to screen setup do not need to be replicated five times reducing the chance of errors being made when changing the screen setup.

Audio

Audio handling in the game is now handled by two classes: AudioManger and AudioPlayer, the latter has been added since we received the project. Prior to our work AudioManager was used to load sounds from files and played whichever sound/music file it was told and the logic for selecting what sound FX to play during the game was part of the main game logic. This lead to duplicate code for selecting which sound to play when there was multiple possible appropriate clips as this behaviour was rewritten each time it was needed. Plus, when wanting to select a sound FX from the same pool of clips this had to also be rewritten each time it was needed.

Therefore the AudioPlayer class was introduced to handle selecting which sound/music to play and the AudioManger is used for loading the audio from file and actually playing a file. “selectAudioToPlay(audioFiles: String, noAudioChance: int): void” was written for selecting what clip to play from a given list of files and this method could be used by the various “play<CLIP_TYPE>Audio()” methods.

Phases

Both the attack and move phase required creating an arrow from a source to target sector which was originally implemented in both the PhaseAttack and PhaseMovement classes. This lead to a large amount of code being duplicated, so arrow creation code was moved to the parent class PhaseAttackMove and the children implemented what to do when the player had drawn an arrow, (the “onArrowCreated(sourceSector: Sector, targetSector: Sector): void” method). Additionally the children also implement their own methods for evaluating if sectors are valid sources and targets for when creating an arrow, (“isValidSource(sourceSector: Sector): boolean” and “isValidTarget(sourceSector: Sector, targetSector: Sector): boolean”, as the conditions for this vary depending on the phase.

Due to a change in the design of the Pro-Vice-Chancellor minigame [4] it was no longer necessary to store persistent data across an entire game and therefore the class was removed, as the majority of the code was now redundant. “PVCSpawn(): void” was moved from the PVC class to GameScreen so that minigame could still be triggered.

Punishment Cards

After inheriting the project the client’s requirements changed, F20 and F21 now required the game to contain three Punishment Cards which the current player could choose to play if they held any. To accommodate this change variables were added to the Player class to store which cards the player holds, (“collusionCards: int”, “poopyPathCards: int” and “asbestosCards: int”), and the PunishmentCardType enumeration was created so that the different card types were able to be referred to.

Furthermore to support the player using these cards a variable, “punishmentCardSelected: PunishmentCardType” was added to the Phase class so that the card the player had chosen to play could be stored and the Map and Sector classes were modified to support applying these punishment card effect. Notably the Sector class had the “poopCount: int” and “asbestosCount: int” fields added for recording how long a sector was under a status effect for.

Postgraduate Unit

In order to satisfy the new requirement, F19, the Sector class was modified such that “unitsInSector: int” was split into “undergradsInSector: int” and “postgradsInSector: int” so that data on two unit types could now be stored.

References

[1] SEPR “Concrete Architecture UML Diagram” Risky Developments[Online]. Available: http://www.riskydevelopments.co.uk/documents/ConcreteArchitectureUMLDiagram.png [Accessed: May. 1, 2018].

[2] "Star UML," MKLab Co., Ltd, [Online]. Available: http://staruml.io/. [Accessed: Apr. 30, 2018]

[3] "About The Unified Modeling Language Specification Version 2.0," 7 2005. [Online]. Available: https://www.omg.org/spec/UML/2.0/Superstructure/PDF. [Accessed: Apr. 29, 2018]

[4] SEPR “Implementation Report” Risky Developments[Online]. Available: http://www.riskydevelopments.co.uk/documents/Impl4.pdf [Accessed: May. 1, 2018].