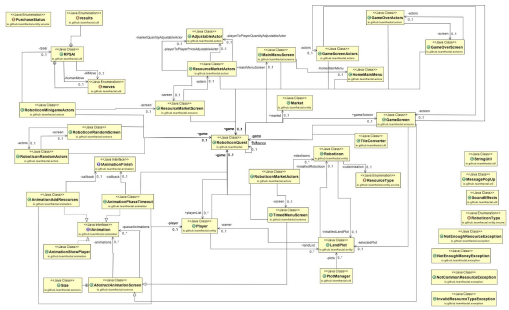
Architecture and Traceability Report

We justified our final architecture based on whether it meet our requirements. This is so that we can ensure traceability from our implementation through our architecture and through to our initial requirements.



Link to abstract architecture (Created using the eclipse plugin ObjectAid)
Below is how our concrete architecture fulfills our requirements:
Link to our final concrete architecture diagram (Created using the eclipse plugin ObjectAid)

Requirement 1.1 is fulfilled by the Player and RoboticonQuest classes, as the RoboticonQuest class can contain up to four instances of the player's class. Allow the game to be played between four human players.

Requirement 1.2 is fulfilled by the RoboticonQuest class, as it keeps track of the current turn and allows each player one turn per round.

Requirement 1.3 if fulfilled by the PlotManager class, as it contains a data on the game map including each plot. With plot being represented by the LandPlot class. The map contains at least 16 plots. Therefore fulfilling the requirement of the game having a minimum of at least 16 plots.

Requirement 1.4 is fulfilled by the LandPlot and Players classes, as the LandPlot class contains information on whether the plot has an owner or not. While the Player class contains information such as which plots the current player owns and is in control of. Therefore fulfilling the requirement that the system shall keep track of which player has acquired each plot.

Requirement 1.5 is fulfilled by the Player class, as it stores how much ore, money, energy and food each player has. Which therefore fulfills the requirement that the system shall maintain an inventory for each player.

Requirement 1.6 is fulfilled by the RoboticonQuest and the Player classes, as when the RoboticonQuest class starts the game, the Player class then adds an appropriate amount of money to start the game with to each player. Therefore fulfilling the requirement that the system shall provide a small amount of money to each player at the start of the game.

Requirement 1.7 is fulfilled by the Market class, as it contains details on how many food, energy, ore and roboticons it contains. This therefore fulfills the requirement that the system should maintain the markets inventory containing all the resources and the roboticons that the market is in possession of.

Requirement 1.8 is fulfilled by the RoboticonQuest and Market classes, as when RoboticonQuest calls for a game to be started the Market class then adds the appropriate 16 units of food, energy and 12 uncustomized roboticons to the markets inventory. Therefore fulfilling the requirement for the system allocating 16 units of food, energy and 12 uncustomized roboticons to the market.

Requirement 1.9 is fulfilled by the PlotManager and LandPlot classes, as they contain what type the each plot is, whether it be a hill or a water tile and their relevant production values. Which therefore fulfills the requirement that certain plots are better at producing a certain type of resource.

Requirement 1.10 is fulfilled by the RoboticonQuest and the PlotManager classes, as when the game enters the next turn the RoboticonQuest checks with PlotManager to see whether all the plots have been acquired. If so the game then enters the last round, followed by the game over

screen. Therefore fulfilling the requirement to stop the game once all the plots of land have been acquired.

Requirement 1.11 is fulfilled by the RoboticonQuest, Player and LandPlot classes, as when it a player enters the land acquisition phase the RoboticonQuest classes allows the Player to purchase a plot of land. It is then checked with LandPlot to ensure that no other player owns that plot, if that is the case RoboticonQuest calls for the next turn to start. Therefore fulfilling the requirement that at the start of each turn, allow the current player to acquire one plot unacquired land.

Requirement 1.12 is fulfilled by the Market class, as when it comes to the market phase of the game players can request to purchase roboticons from the Market class granted they have enough money. Therefore fulfilling the requirement that the system shall allow the player to purchase roboticons from the market.

Requirement 1.13 is fulfilled by the Roboticon class, as if the player has uncustomized roboticons they may purchase a specialise a roboticon to either food, ore or energy production through the Roboticon class. Therefore fulfilling the requirement that the system should allow player to customise their roboticons in exchange for money.

Requirement 1.14 is fulfilled by the TimedMenuScreen class, as it contains a timer which once it expires it calls for the next turn to occur which is used on the purchase, customization and placement round. Therefore fulfilling the requirement that there should be a fixed amount of time to purchase, customise and place roboticons.

Requirement 1.15 is fulfilled by the LandPlot class, as it allows for the installation of customised roboticons.

Requirement 1.16 is fulfilled by the RoboticonQuest and LandPlot classes, as when the RoboticonQuest ends a round it calls for the all the LandPlots to produce their respective resources, depending on whether they have roboticons and their customisations. Therefore fulfilling the requirement for each plot of land with a roboticon on it to produce resources for the player that owns it.

Requirement 1.17 is fulfilled by the RoboticonQuest, Market and ResourceMarketScreen, as when the RoboticonQuest calls for the market phase to begin, the Market which is interacted with through the ResourceMarketScreen allows players to buy and sell resources to each other and the market itself. Therefore fulfilling the requirement that all players should access the market simultaneously.

Requirement 1.19 is fulfilled by the Market class, as it allows players to sell resources in their inventory to the market for a set price, in exchange for money. Therefore fulfilling the requirement that players can sell resources in the market.

Requirement 1.20 is fulfilled by the Market class, as it allows players to buy resources in the market for a set price. Therefore fulfilling the requirement that players can buy resources in the market.

Requirement 1.21 is fulfilled by the RoboticonQuest class, as once it detects that players have marked that they have finished with the market it will proceed with the next round. Therefore fulfilling the requirement that the system should move to the next round once each player has indicated they are finished with the market.

Requirement 1.22 is fulfilled by the RoboticonQuest class, as it allows for the calculation of a player's final score based on the resources in their possession. Therefore fulfilling the requirement that the system will calculate the player's final scores.

Requirement 1.23 is fulfilled by the GameOver Screen class, as it displays all the players scores along with which player won. Which fulfills the requirement that the system should report who has won.

Requirement 1.24 is fulfilled by the RoboticonMinigameActors class, which allows players to gamble their money on a game of chance or in this case rock, paper, scissors when they access the market. Therefore fulfilling the requirement that players should be able to gamble some of their money away.

Requirement 1.25 is fulfilled by the RoboticonRandomActors and RoboticonRandomScreen classes, as they contain the eight random events which are randomly chosen every few rounds for players. Therefore fulfilling the requirement that the system should have at least 3 random events.

Requirement 1.26 is fulfilled by the RoboticonQuest and PlotManager classes, as when a game starts the RoboticonQuest calls for new unacquired plots. These are created by the PlotManager class which handles all the plots in the game. Therefore fulfilling the requirement that at the start of the game all plots are unacquired.

Requirement 1.27 is fulfilled by the Market class, as it allows for the creation of roboticons given it has enough ore in its inventory.

Requirement 1.28 is fulfilled by the GameScreenActors class, as it allows for players to 'catch the chancellor' randomly after they place their roboticons.

During development several changes were required to accommodate the new requirements change from the client. These being the addition of the 'Capture the Chancellor' and the requirement for four player support. However, during the additions of these new features and additions it became clearly that there was no need to modify the architecture. As our inherited project's architecture already lay down the groundwork for the requirements change. An example of this is how the only modification is how the relationship between RoboticonQuest and Player changed. Instead of there being two players for every RoboticonQuest, now there is four. As for the 'Capture the Chancellor' minigame, this was added in the form of an extra method in the GameScreenActors class, as we felt it was unnecessary to make additions to our architecture to accommodate the new addition. However some additions we made to the architecture were the classes MessagePopUp and SoundEffects. These were added for a number of reasons, such as to facilitate a new method of informing the user through pop-up windows. As well as allowing the game to have sounds effects. This results in the game giving the user another form of feedback through sound, as well as adding character and atmosphere to the game in general.

As well as justifying our architecture against our original requirements, we justified our final concrete architecture against our conceptual architecture. As this ensure traceability through every step of the design process.

In the conceptual architecture we created a number of use cases describing the steps a user might take to perform various tasks and then created UML diagrams (using Lucidchart) to visualize the behavior of the system and the interaction of its components according to these use cases. The use cases implemented many of the system requirements and helped to show the correctness of the model as it supports the necessary communication links between components. These can be viewed here: https://sepr-topright.github.io/SEPR/documentation/assessment4/UseCases.pdf

Player class in our conceptual model was implemented by the Player class in our final architecture. HumanPlayer and AIPlayer however were not implemented into our final architecture as the need for an AI player was removed. As such it made the need for two separate type of players (human and AI unnecessary) so in our final architecture HumanPlayer and Player were both rolled into a single class. Plot class was implemented by the LandPlot and the PlotManager classes in our final architecture. The reason for implementing the two different classes for Plot was due to implementation where we felt it would be easier to understand if there was a separate class that managed all the plots. The Inventory class was not implemented, however its two dependencies MarketInventory and PlayerInventory were implemented in the final architecture as Market and Player. We felt that it was not worth having separate classes for just the inventories of both Player and Market and instead thought it would be more clearly understandable if they were included in their respective classes. The Market was implemented in the final architecture as the Market class.

As just shown, our final architecture can be traced to our conceptual architecture, both of which fulfill our final requirements. As such we can ensure traceability from our requirements through to our architecture.