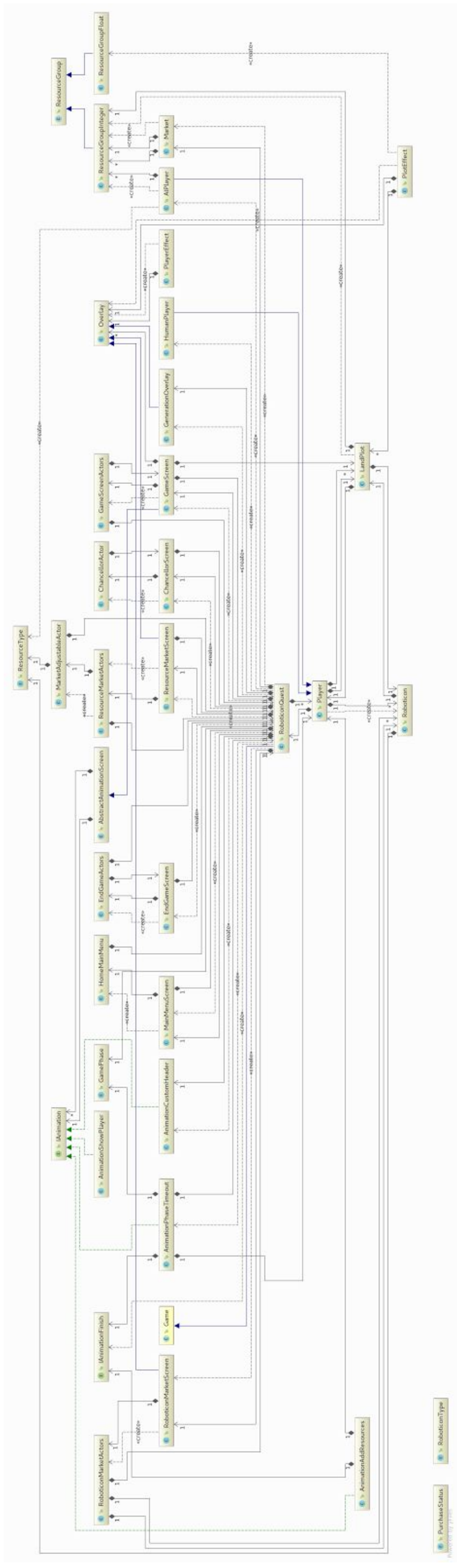
1.0 UML Diagram (Generated by Intellij)

Full detailed architecture can be here [1], abstract detail (like below) can be found here [2].



2.0 Architecture Discussion
A list of requirements can be found in the acceptance testing document (see user stories and non-functional/functional requirements) on our website here [3]

2.1 Market
The Market class was refactored in an attempt to better accommodate requirement 8 [3]. In addition, the static nature of the previous teams Market led to predictable gameplay, thus reducing a user's enjoyment. The Market's prices now function similar to supply and demand economics based on the total generation of resources. While the change in the generation of the prices, and the addition of a price history, alter how the Market functions it has no effect on the architectural structure of the class (or those that utilise this functionality).

2.2 AI
Much of the functionality within the AI class was refactored. This was undertaken in an attempt to improve the playability and enjoyability of the game, as we felt that the previous AI was lacking. The AI now makes logical decisions and assumptions on the changes in the market, resulting in more human-like gameplay. There has been little to no effect on the previous architecture, in fact the only changes made beside the refactoring of certain methods, were the addition of helper methods. As such the architecture created by the previous group has remained unchanged in the way individual classes relate to the AI player. While the relation between the Market and the AI has remained unchanged, the AI is now dependent on receiving accurate information of historical prices from the Market to make its decisions. This refactoring of the AI confirms to all requirements pertaining to the class (see requirements 4 and 8) [3] while improving the overall quality of the game, as such we felt justified both in the change to the class itself and the increased dependency on the accuracy of the Market.

2.3 Multiple Players
The new requirement to support up to four players did not require any change in the overall architecture of classes, as the structure of the code from the previous team was not hard coded for just two players (requirement 12) [3]. Instead only a few lines of code were modified or added to achieve support for any number of players. The main menu UI was updated to allow the player to create a two, three or four player game.

2.4 Chancellor Phase
An additional requirement was to implement a chancellor phase in the game, which occurs in between the resource generation and market phases for each player's turn (requirement 13) [3]. As this phase required additional UI elements to be shown, such as the chancellor sprite and phase description text, a new class was created (ChancellorScreen). This class inherited from the appropriate libgdx class (Stage), allowing it to get user input and have an update function called every frame (via GameScreen). An instance of the ChancellorScreen class is stored in the RoboticonQuest class, as this is where all the other screen instances are located.

There is also an additional class which was needed to fulfil the new requirement, ChancellorActor, which is a class representing an image. The reason for this image specialisation is due to the fact that the image needs to detect when it is clicked, and also be able to display itself in a random location each time it is shown. Therefore this class contains methods to show and hide the image, as well as calling back to ChancellorScreen when a click on the image is detected.

If the current player is Human, the state machine controlling the phases will display the chancellor screen when the game enters the chancellor phase, by calling a method on the screen object. When this happens the screen object will initialise all its timer and counter variables to default values, and generate a countdown time. Once the countdown time is over, the chancellor sprite is shown in a random location on the screen, by calling the show method on the actor. The chancellor then starts a timer, and if a click is not detected after the chancellor has been displayed for the maximum amount of time the chancellor is hidden. Then a new countdown is generated to show the sprite again, if they have additional attempts left to try and click on the chancellor. However, if they don't then a call back is made to the RoboticonQuest instance to move on to the next phase. On the other hand, if a click was detected then a method in the screen class is called, rewarding the player with money before calling a method to move on to the next game phase.

When it is the turn of an AI, no chancellor visuals are required as this will just waste time for the human players involved in the game. For this reason, the chancellor screen and actor classes are never used in an AI player's turn. Instead a function is called within the AI player's class, using a random number generator which gives them a chance of gaining a reward.

2.5 Resource Groups

The resource group classes are used to represent either the level of resources, or in plot and player multipliers. The ResourceGroup class is abstract, so that we may limit the use of its generalisations to certain number classes (i.e. integer and float). The functions of this superclass (ResourceGroup) provide the various getters and setters, with an abstract function for Sum(). The subclasses (ResourceGroupInteger, ResourceGroupFloat) provide operators and other functionality that could not be achieved in the superclass. We included this class for several reasons. Primarily because it proved verbose when applying operations to each individual resource (energy = expr; food = expr; ore = expr;). This was most evident in the use of plot/player effects, were previously an array was used to store the information, with these values being multiplied with the relevant resource variables of its target. Now, we simply use the mathematical operators that are present in the ResourceGroup class. Not only is this simpler, it is also a more intuitive means of representing the resources.

Bibliography

[1] SEPR, “full_architecture.png” [Online]. Available: https://seprated.github.io/Assessment4/full_architecture.png. [Accessed: 29 April 2017]

[2] SEPR, “abstract_architecture.png,” [Online]. Available: https://seprated.github.io/Assessment4/abstract_architecture.png. [Accessed: 29 April 2017]

[3] SEPR, “Acceptance Testing,” [Online]. Available: https://seprated.github.io/Assessment4/acceptance-testing.html. [Accessed: 29 April 2017]