1.0 Introduction

A list of requirements can be found in the acceptance testing document (see user stories and non-functional/functional requirements) on our website here [1]. Our code is available on our github repository: https://github.com/Tim020/Seprated-DRTN-Fractal.

1.1 Market

Market has been implemented to include only trading with the market itself and no peer-to-peer trades. Buying and selling from the market has been implemented using varying prices which are calculated using a formula, which is based on the amount of resources being generated each turn by the roboticons. All trading logic has been implemented to be in the Market class, as this logic is independent of any player and should be the same for all entities interacting with the market. Player interacts with the MarketUI class which then in turn communicates with the Market class. This change was made as we felt that the previous implementation was barely meeting the needs of the requirement (8) [1], i.e. prices based on resource abundance. Through imitating supply and demand economics in our market, we hope to provide improved playability and user enjoyment.

1.2 AI

As the previous AI was mostly logic based (selling half their resources every other turn) and in some cases purely random (for example tile acquisition was completely random), gameplay and enjoyability were negatively affected. Therefore to improve the players experience the AI class would have to be refactored. Following refactoring we ensured that the AI passed all of the test cases provided by the previous team, to ensure that we had not inadvertently affected the classes function as part of the whole system.

These changes accommodate the refactored Market by ensuring that the AI integrated the ability to gather historical data into its decisions on selling/buying and gambling. These prices are used in conjunction with a simple probabilistic model to predict the fluctuation of buying/selling prices, ensuring that the AI may make informed decisions on what and when to buy/sell. In addition to this, we've incorporated an element of 'chance', when the AI sees a high or a low in the price of the resource it is not guaranteed to sell/buy during that time. This in conjunction with a limit on how much of its resources the AI will sell, ensures that the AI is unable to bulk buy/sell its resources at times it predicts the market price has reached a minima/maxima, as within a dynamic market this could lead to frustrating results for the player (frequently crashing prices and an unstable economic model).

Tile acquisition is based on information take taken from the market, specifically the markets buying prices. The AI, similar to a player, chooses the tile that has the greatest monetary value at that turn in an attempt to make sound investments.

We have also chosen to limit the AI's ability to gamble, through enforcing that it may only gamble when it has not sold resources, it has over 100 currency and that a random float is greater than 0.5. This was considered a sensible way of limiting the AIs gambling, and by extension avoiding placing the AI in a situation where its functionality may be negatively affected by its lack of currency. For example, it could be that the AI lacks enough funds to buy tiles or roboticons and does not believe it should currently sell resources and as such may remain idle for a number of turns.

Through these methods we hope to limit the efficiency of the AI in the hopes of provide a more 'human-like' opponent, thus increasing the overall user experience. By developing a AI that is able to make mistakes (i.e. choosing not to sell/buy at optimal times), we ensure there are opportunity that a player can capitalize on. The team hope that this will increase the enjoyability of the game as having a fallible AI will likely lead the player to making more risks in the hope of making the most of any mistake - creating more exciting gameplay.

1.3 Multiple Players

We gained a new requirement to complete the game, allowing up to four players per game (requirement 12) [1]. We decided to limit the number of human players to one, with the remaining players being in AI form. The main justification for this was that the client has confirmed the intended use for the product will be on university open days. Having a single Human player would allow a current university student or member of staff to demonstrate the game to a large audience without forcing participation from anyone else. This will allow them to present the features of the game promptly.

The structure of the project we inherited from the previous team allowed us to go from a two player game to a three or four player game with ease. None of the architecture has to be changed to allow this change to happen. Instead, there were a few additional lines of code, changes to some parameters for some methods, and an update the the main menu UI, all of which allowed the player to play against one, two or three AI opponents.

1.4 Chancellor Phase

An additional requirement that we received before completing the game was to include a ‘capture the chancellor mode’ (requirement 13) [1]. In this mode the requirement states that the chancellor may appear randomly within a fifteen second period. We made a slight modification to this logic, making the chancellor appear for a total of three times, with a maximum of five seconds between them. The reason for this is that we believed that if the chancellor may not appear, a fifteen second wait was too long for the player. Secondly, we wanted to keep their attention to the screen throughout this chancellor phase, giving them multiple regular attempts in which they have a fair chance of clicking on the chancellor.

There was an ambiguity in the requirement, with it not being obvious when the chancellor phase would take place in relation to the other game phases. We chose to include it after resource generation, but before the market phase. This gave the player a chance to get some additional money before entering the market phase, which could come in useful when purchasing resources.

The full details of the implementation of the chancellor phase are described in the architecture report. New classes were created, ChancellorScreen and ChancellorActor. The screen class houses the actor instance, and contains the variables required for tracking time, for both counting down to when the chancellor sprite should be shown, to counting down to when it should be hidden again. The actor object is a child of the libgdx Image class, and therefore is used as the chancellor sprite. It also contains methods to hide the sprite, and show it in a random location on the screen. There is a call back from the actor to the screen when a user click is detected, which indicates the player successfully clicked on the chancellor. In the current version of the game, the player has three attempts to click on the chancellor, with the chancellor being displayed for exactly one second, and the maximum waiting time between each chancellor showing being five seconds.

The classes and method described above are used when the player whose turn it currently is is Human. In the case that the current player is an AI, no UI or chancellor logic is used, and instead a random number generator is used to determine whether to award the AI player in this phase or not. We had to modify the game phase enumeration to account for the chancellor phase, and then add this to the state machine.

Bibliography

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