Software Engineering Project: Assessment 3
Gandhi-Inc. - Project Blind Eye

Implementation report

A number of changes have been made to the code, including new classes added and changes to existing classes. All these changes have been done to complete requirements outlined in the requirements document. We have also commented and refactored code that we have used, to make it easier for subsequent teams to use the code.

New classes added:

Leaderboard [REQ 13c]

To implement the leaderboard, we created 2 new classes: LeaderboardBackend and LeaderboardFrontend. The backend class handles storing the information and then retrieving the information from storage, the frontend handles displaying the information stored by the backend.

LeaderboardBackend:

This class has a number of methods to enable storage and retrieval of the top players that have played the game. The method "AddPlayerToLeaderboard" writes the player name and score to the file "GameSave.txt". This allows permanent storage of scores. To access the stored games the "ReturnsBestPlayer" method is used, this firstly creates an arraylist of string arrays of storing the name of each player and the score of each player. It then finds the best three scores from this list and returns an array of string arrays containing the best three players and their information.

LeaderboardFrontend:

This class handles displaying the best three players. To display this information, a separate screen is created which contains the information required. This new screen also stores the game state, so that this is not lost when you change screens.

How To Play:

We have implemented a how to play class. [REQ 13bi] introduces this class, where we need to add the game's context and teach a player how to play it through a list of instructions. We have primarily used graphics to convey the information based on [REQ 13bii] The graphics can be found on the how to play document on the website which the game links to.

Storyline

We created a storyline based on [REQ12] [REQ13bi] and implemented it into the splash-screen. We have also made it available on the "How to Play" document.

Gamble / Bar

[REQ 5D -d. Player can win or lose money via "gambling"]
[REQ 7D - d. The market will have a bar where money can be won or lost via gambling]
The gambling system is split into two sections, the front end, and a back end. The backend has 2 important classes: "PlayRoulette" and "PlayLuckyDip". Both of these classes will return a int relating to the amount of money won or lost if the game is played(positive if you win, negative if you lose). They use the java.util.random function to decide if a player wins or loses.
The front end provides a simple interface for the player to play pub games, the player presses a button to gamble their money, this button calls the back end functions.

The requirements document has been referenced in the style: [REQ1] to mean requirement number 1.

Random Effects

The Random Effect class follows the previous requirement update set in Assessment 2 [REF REQ 6] , which offers a variety of effects ranging from low impact to larger impact. We have introduced 3 different "random effects" to cover this variety of impact that was required by our customer. They include a Trump appearance - blocking all production on a specific tile, a Meteor shower which kills the roboticon on a tile, and a solar flare which stops production on a cell for just one round.

Changes to classes:

Splash screen:

The splash screen has been changed so that instead of displaying the "duck related team name" logo. It displays the storyline for the game. [REQ12] [REQ13bi]

Main menu:

The main menu has been changed so that when you click the "How to play" it displays the how to play screen and when you click the Leaderboard button it displays the leaderboard.

Tile:

drawBorder has been changed so that the Color object that is used is the College Color object.

List of owned tiles function
Set wall – this method is a Boolean setter that informs the tile that there is a wall sprite to be displayed
Set Meteor – this method this method is a Boolean setter that informs the tile that there is a meteor sprite to be displayed
Set SolarFlare – this method is a Boolean setter that informs the tile that there is a SolarFlare sprite to be displayed

The mouse-over feature has now been implemented so when the cursor is hovering above a tile the tile will now construct a display which will contain the tile number, the base production of that tile and the roboticon production. The upgrade levels of the assigned roboticon are also displayed and automatically updated after each upgrade.
Tile [REQ 11a]; Upgrades [REQ 11b]

Scoring:

[REQ 4a] states we should have a scoring system based off of resource amounts at the end of the game. The score is calculated using the sum of the value of all the assets that the player owns.

Justifications for changes to requirements:

AI class:

After referring to the requirements [reference: REQ3], we decided that implementing an AI class was not a feasible solution. The AI feature would require a complete rewrite of a lot of the game engine and therefore as it was not a necessary requirement, we decided not to include it. The previous architecture does not support an AI class as they require buttons to be pressed for events to occur. For an AI class to be implemented, we would have to rewrite all the architecture and make it so that the buttons call functions, and therefore the AI would be able to use these functions.

Leaderboard:

We decided to only include the top 3 players in our leaderboard. The requirements [REQ13ci] called for the top 5, however we believe that 3 should suffice and that it will become cluttered displaying everyone who plays the game. The idea of a top 3 also makes it more exclusive for those with the highest score and making it more rewarding to feature on the list.

Keyboard input:

The game only has 2 menus and therefore adding keyboard shortcuts [REQ15a] would introduce more complexity to the game than is necessary. This would contradict [REQ1], and for this reason, we have decided not to implement this feature.

Resource values:

We have decided not to implement [REQ14biii] and [REQ2aii] since a variety of different landmarks would entail a variety of different bonuses, ultimately making the game hard to balance, thus reducing enjoyability. Furthermore, maintaining a coherent relationship between the actual effect of the bonuses and the variety physical characteristics of the many different landmarks would be difficult as well. This would also mean that [REQ1] would not be met.