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. Please note our reference to the requirements has been done in the style: [Req. 13.a.i] referring to Requirement 13. a. i. in the Appendix_1.pdf.

New classes added

Leaderboard

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. Although the requirements [Req. 13.c.i] stated we should display the top 5 scores, we have only implemented the top 3. We decided to only include the top 3 as we believe that 3 should suffice and that it will become cluttered displaying more than 3 players who play 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.

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 in a txt file, so that this is not lost when you change screens.

How To Play & Storyline

We have implemented a how to play class and screen following [Req. 13.b.i]. We have added a storyline to this area following [Req. 12]. We have opted for a brief written narrative to give the user some context on how to play. Unfortunately, we were unable to implement the graphics as required

in [Req. 13.b.ii] and flipbook-style pages with detailed instructions [Req. 13.b.i]. This would have demanded a lot more work on a feature we thought was not crucial to the game play or the satisfaction of the player. Instead we opted for a link to a very easy follow pdf which is featured on our website. This pdf includes graphics and a a list of instructions to explain the game to the user.

Gamble / Bar

The gambling system is split into two sections, the frontend, and a backend. The backend has 2 important classes: "PlayRoulette" and "PlayLuckyDip". Both of these classes will return an integer 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. These features are clearly following the requirement [Req. 5.d].

The front end provides a simple interface for the player to play pub games [Req. 7.d], the player presses a button to gamble their money, this button calls the back end functions and allows the player to win or loose money based on a random lucky number. We have added an additional feature which also does not allow the player into the pub if they don't have sufficient funds to play and will kick the player out if they loose money and thereby their funds are below the minimum threshold.

Random Effects

The Random Effect class follows the previous requirement update set in Assessment 2 [Req. 6.a], 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 specific tile, and a solar flare which halves the resources of a player. These effects have been implemented with both a JFrame notification to inform the player of the consequences and also a visual graphic on the map.

Changes to classes:

Splash screen

The splash screen has been changed so that instead of displaying the "duck related team name" logo. It displays out Gandhi-Inc. logo.

Main menu

The main menu has been changed so that when you click the "How To Play" button it displays the How To Play screen and when you click the Leaderboard button it displays the leaderboard. We have also updated our name to the screen instead of Duck Related Team Name.

When a user clicks "Start", there are now JFrame popups which will allow the user to Enter their name and their desired college. This is compliant with [Req. 13.a.i] that allows a user to select a college. We have made another popup for player 2 [Req. 13.a.iii] to enter their details and enter their desired college but have taken note of [Req. 13.a.iv] where they can't select the same college.

We have not implemented an AI feature as this was not a clear requirement [Req. 3] and this also required a lot of additional work. Please see the justifications below for further details as to why we choose not to take on this task.

Tile

We have added to the drawBorder method, so that the Color object that is used is the College Color object. This is not a requirement but we have decided to add it to make it easier to distinguish which player owns which tile.

There are also now Setters and Getters for the wall, meteor and solar flare methods. These will state whether a tile has a wall or not and can then activate the image overlay on the tile if it returns true.

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. This follows the [Req. 11.a]. The upgrade levels of the assigned roboticon are also displayed and automatically updated after each upgrade following [Req. 11b].

Scoring

The requirement [Req. 4.a] 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. We convert the resources into the equivalent money value and then check which player has the most money at the end of the game.

Justifications for changes to requirements:

AI class

After referring to the requirements [Req. 3], 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. 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 each button calls a function that an AI can also use. Therefore, we have decided that the AI feature should be left out of the requirements.

Keyboard input

The game only has very simple tasks, therefore adding keyboard shortcuts [Req. 15.a] would introduce more complexity to the game than is necessary. Currently there are only a few buttons a player needs to press and having keyboard shortcuts and inputs would contradict [Req. 1] to have a non convoluted and easy to play game so for this reason, we have decided not to implement this feature. There are keyboard inputs required when entering a players name but this is the only time the keyboard is used.

Resource values

We have decided not to implement [Req. 14.b.iii] and [Req. 2.a.ii] since a variety of different landmarks would entail a variety of different bonuses, ultimately making the game harder to balance. This can lead to 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 [Req. 1] would not be met. These requirements would also give a major advantage to player 1 who is able to choose a tile first making player 2's disadvantage extremely unfair.