Code Documentation

Game.cs

The Game script is the centre of the project and initialises the basics of the game. When the script begins it firsts reads the csv file in which holds the information regarding the board, a range of lists are created in order to hold each section of information separately then each list is filled with the relevant information. The board initialisation then occurs creating an array of the Tile class length of the board. If a tile can be bought a new TileStreet object is created inputted with the relevant data from the initial csv sheet. While other types of card are assigned other Tile Types under the Tile abstract Class. Players and cards are then initialised with an array of six players being created all with their positions set to Tile 0, all the cards are shuffled, and the first player moves beginning the game.

The Update() method deals with the player positions on the visual board. It goes through each player in the array and if their position on the board doesn't equal the position in their class, which should be just after they roll the dice, it calls a Player method to set their animated position to the new location. The player token GameObject is animated to move to the new location. Once the classes location and physical location match up the LandingMethod() of that tile is called. The final part of the Update method checks if the game has finished, if active players falls to 1 the game ends.

Player.cs

The player class is just initialised with its id number.

The first method is move(), this simulates the rolling of the dice, a random number between 1-6 is chosen twice, the results are added together and if the player is not in prison the SetPosition method is called to set the players new location.

GetId() and GetPosition() are two classes which just return the classes variables.

SetPosition() takes the integer representing the new position as input. If it is greater than the boards length the CrossGo() method will be called, explained below.

GetAnimatedPosition() and SetAnimatedPosition() are again just getters and setters for the player scripts variables.

CrossGo() called when the player crosses the go tile will add 200 to the players balance.

isActive() check if the player has a balance above 0 and has not surrendered.

Tile.cs

Tile is an abstract class.

It holds all the variables containing the information pulled from the input csv file, such as the name and various prices. In is initializes in the Game script when making the board.

It holds a abstract method for landingAction() of which every type of tile reacts differently.

The Buy() method is called to when the current player buys the tile they are positioned on. It first sets up the new rent after buying, then reduces the current players balance by the required amount, then assigns the tile owner variable with the players id, in int form, and vice versa with the tile id in the players card list.

The Auctbuy() is called by the TileStreet script after the auction process has taken place, Auction.cs. It takes two variables in int form, first being the id of the player who won the auction and the second being the amount of money they bid. Like above, the method first sets the new rent, although this time sets the tile id to that of the input variable, the wining player. Then removes the winning bid amount from that players balance and assigns them the card.

The ResetPrice() method is called to set the new rent of a tile in times where it needs changing, for example when a property is upgraded or mortgaged. Using if statements it checks if the Tile variable "mortgaged" is false, if it is false it checks the "numHouses" variable and sets the rent as the variable which holds the price taken from the csv file. Is mortgaged is true it set the rent to 0.

TileDraw.cs

TileDraw() will be called when a player land on a tile where a potluck or opportunity knocks card will be picked up.

When the player lads the LandingAction() is called which enables the Cont.cs script, explained below, this tells the player they will be picking up the card. When they click continue the Update() method's if statement become true and the Card method Draw() is called, then the next players turn begins with nextp().

TileFreeParking.cs

Free parking only has one method excluding its constructor. This tile holds a balance of money that when the player lands will add that balance to the player.balance and reset this tiles balance to 0.

TileGo.cs

TileGo is associated with the Go tile on the board and as a result its only purpose, when the Landing action is called, is to call the player method CrossGo().

TileStreet.cs

The tile street constructor takes 10 inputs, the first ,i, is the id while the next nine inputs are all information pulled from the csv board file in Game.cs. These are all immediately assigned to the appropriate Tile.cs variables and the remaining constructor variables are set to their beginning amount. The next part of code sets up the Tile object on the board, for every TileStreet.cs initialized an accompanying game object is created on the board decorated with the information of the class.

LandingAction() is called when a player lands on a tile, it immediately pauses the game and starts checking the information from the tile. An if statement first checks if the card is owned, if not owned the owned variable will be "99".If the tile is owned it then has to check if the current player is the one who owns it, is so it called the Choice.cs script. If another player owns the tile it calls the Cont.cs script and charges the player the required rent and gives it to the owner. If the card is not owned at all the Buy.cs script is called.

The Update() class holds the code for what happens after the scripts above finish. It constantly checks the above scripts for when they are finished and if they change their finished variable, it activates one of the various if statements which carried out the next part, from calling Buy(), getting the results of an auction, to just turning off panels and resuming the game and calling the nextp().

Nextp() sets up the dice roll UI and calls the game to continue to the next players turn.

ColourFromString() is a method used in the constructor when creating the physical Tile Game Object, the input file takes the colour of the tile as a String, so the method just takes this string and sets it to the required format for unity to be able to use.

Buy.cs

The Buy script is called by the TileStreet script when a tile is unowned. It holds many Text and Game Object files as it creates the Buying UI. Firstly, it activates the relevant Game Object panels and sets the text and relevant buttons, Yes or No. It also creates a visual representation in form of a card for the tile. It shows the player all the tiles information, all set in Text game objects; the info is pulled from the variables of the Tile Class.

UpgradeCost() is a method that appears in a few scripts as a way checking, using the tiles colour, what its upgrade cost should be, Its used here to show the upgrade cost on the UI for the player.

The next method NoButPress() is called when the user clicks on the no button, It changes the visible text and sets a variable that activates something in this classes update() method.

The YesButPress() does mostly the same as no only this time checks if the current player actually has the balance to buy this card.

ContButPress() is called when the player clicks the continue button which become visible after either no or yes is clicked, it sets the choice made variable with the yes/no answer which lets the TileStreet class know the script has finished.

Update() simply checks that when a Yes/Mo choice is made those buttons are deactivated and the continue button is activated.

Auction.cs

The Auction class is called in TileStreet.cs Update(), when the Buy script returns the player not choosing to buy the property. This script controls the relevant UI Game Objects to run and takes input from all players. It begins by activating the UI Panels, buttons, Text and an input box for players to input their bid for the property. Each player one by one will enter a bid and click confirm.

The confirm() method is called every time the player hit the confirm button after entering their bid. It firstly checks if the entered bid is under that players total balance. If not askes for them to re-enter a new value. It the number is fine it converts the string to an int and places the bid into an array. Then an if statement checks if all players have entered, if not the text is changed accordingly and the above happens again for the next player. If all players have entered the entry box is deactivated and the array holding the bids is searched through for the highest. The highest bid and player are set to variables which can be reached by the TileStreet class and the finished variable is set to true so the TileStreet class also knows the auction process has been completed.

Choice.cs

Choice script is called from TileStreet.cs LandingAction() if the player lands on a tile they already own the script is activated. On enabling it will activate the corresponding UI. A Panel containing text and two buttons Pass and Upgrade.

The method PassButPress() is called when the player clicks the pass button, it changes the text and sets a variable saying it's been clicked.

They method UpgradeButPress() does the same as above.

ContbutPress() activates after either one of the above has been clicked, its activated when the player clicks the continue button.

Update() checks if pass or upgrade has been clicked, if so deactivates those buttons and activate the continue button.

<u>**Cont.cs**</u>

The Cont script can be called at different times throughout the code, it is a simple display message and continue button. It has an Enum variable Type which is needed by the script on creation to run the appropriate UI and, on finish for the right script to read the output. On enable it simply makes the base UI elements described above appear.

The script contains a SetText() method, this method always needs to be called after activating and requires the text that needs to be shown and the appropriate Enum as inputs to set up the UI. In case that the Enum Type is "pay" a card showing the tiles information is shown as well as the original panel, this is mostly the same as the card which is shown in the Buy() script but shows other relevant information to situation such as who the owner of the card currently is and the current rent.

ContButPress() is called when the continue button is press, it will turn off the card created above and set finished to true for other script to continue.

UpgradeCost() is described in Buy.cs

<u>**Roll.cs & Sell.cs**</u>

These two scripts are connected to two buttons which appear on the scene individually. Roll and Sell

The roll button OnClick() just calls the Cont.cs Script to show the next players roll.

The Sell button OnCllick activates the Properties UI menu which in turn activates the ButtonListControl.cs and ButtonListButton.cs.

<u>**ButtonListControl.cs**</u>

This script ButtonListControl is enabled when the properties menu has been activated. It controls the initialization of the ButtonListButton objects, for every property the current player owns the script will produce a scroll list of buttons, each one with the name of the property. As well as on closing the menu it will delete all these buttons.

<u>**ButtonListButton.cs**</u>

This script is held in the button object which will be cloned for every property owned.

The first method is setText() this takes three inputs from the control class, the string to be displayed on the button , the id of the property and the actual card tile. It sets these to the relevant variables and sets the text.

OnClick() is called when the button is clicked by the player, it enables the mort.cs script with the cards id, for the purpose of selling, upgrading or mortgaging.

The OnMouseOver() method is connected to another event listener so when a player hovers their pointer over a button, a card set up in the same fashion as Buy()/Cont() displays all the relevant information regarding that property.

The OnMouseExit() method just sets all the text elements as empty strings when the pointer moves off of the button associated with that property.

UpgradeCost() is described in Buy.cs

Mort.cs

Mort is the script which deals with the selling/mortgaging and upgrading/downgrading of properties. It is activated when a player clicks on one of their properties in the properties menu. It enables a UI menu similar looking to the buy menu asking if the player would like to sell or upgrade their property.

SetParam() is called by the ButtonListButton () giving the Mort script the cards id which allows the Mort script to search through Game.board, find the tile and get information and check the variables on how many house are currently on the property, its colour and whether this player controls all cards of the same colour.

OnEnable() just makes the first set of buttons and panel/text visible while hiding the second set of buttons.

FirstSellButPress() Is called when the player clicks sell when prompted if they want to sell or upgrade. It first checks if the number of house on the tile is more than 0, if so immediately disables the sell/upgrade buttons, enables the continue button, removes a house from that tile, adds upgrade cost amount to the players balance and updates the rent by calling ResetPrice(). If there are no houses present on the tile it deactivates the sell/upgrade buttons and activate two new buttons asking Mortgage/Sell and asks the player if the want to sell or mortgage their property.

UpgradeButPress() is called on click of the upgrade button. This immediately checks if the player actually owns all of the properties of this colour, this was checked in SetParam(). If true, deactivates the Sell/upgrade buttons, activates the continue button, decreases the players balance by the upgrade cost, adds a house to the tile and updates the tiles rent. If the player doesn’t own all the properties of that colour activates the continue button as above and tells the player, they can’t upgrade yet.

MortButPress() is called when the player clicks the mortgage button in the second menu after sell has been pressed. It first checks that tile to see what the Tile.mortgaged variable is set to, if true the game lets the player know that the property is already mortgaged and they can’t do it again. Or in the case its not mortgaged adds half of the original cost to the players balance, sets the tiles mortgaged variable to true and calls that tiles ResetPrice() which should now set the rent to 0.

SellButPress is called in the second menu and is clicked when the player confirms they want to sell the property rather than mortgage. It checks whether the Tile is already mortgaged and adjusts the players balance accordingly, adding the full cost when mortgaged is false or half the cost for when mortgaged is true. Then sets owner back to unowned “99”, sets mortgaged to false, removes that card from the players card list.

contButPress() is called after all of the above through the usual update method, and when clicked disables the menu.

Update() here just deactivates the second set of button when they have been clicked and enables the above continue button.

UpgardeCost() is a method to check the actual upgrade cost of a property depending on the input colour, it is used in the above methods to see how much to give or task from a players balance when dealing with upgrades/downgrades.