GCE Computer Science (7517)

The Practical Project

|  |  |  |
| --- | --- | --- |
| Centre number |  | Centre name |
| 22133 | The Perse School |

|  |  |  |
| --- | --- | --- |
| Candidate’s full name |  | Candidate number |
| Abhinav Gupta | 2262 |

|  |
| --- |
| Project title |
| Travelers of Catan |

Contents

[*1* Analysis *(9 marks)* 3](#_Toc156208163)

[1.1 Background to/ Identification of the problem 3](#_Toc156208164)

[1.1.1 Game Rules 4](#_Toc156208165)

[1.1.2 Possible problems with board games 4](#_Toc156208166)

[1.2 Research carried out 5](#_Toc156208167)

[1.2.1 Board data structure 5](#_Toc156208168)

[1.2.2 AI algorithm options 6](#_Toc156208169)

[1.2.3 Storing files and online multiplayer 7](#_Toc156208170)

[1.3 Identification of the prospective user(s) 7](#_Toc156208171)

[1.3.1 Interview 7](#_Toc156208172)

[1.3.2 General Information 7](#_Toc156208173)

[1.3.3 General Game Rules 7](#_Toc156208174)

[1.3.4 User Account System 8](#_Toc156208175)

[1.3.5 Bots, difficulty and additional features 8](#_Toc156208176)

[1.4 Numbered measurable, appropriate specific objectives of the project 9](#_Toc156208177)

[1.4.1 MVP 9](#_Toc156208178)

[1.4.2 Basic GUI 9](#_Toc156208179)

[1.4.3 Advanced Game 10](#_Toc156208180)

[1.4.4 Advanced GUI 11](#_Toc156208181)

[1.4.5 Load/Save 11](#_Toc156208182)

[1.4.6 Final Product 12](#_Toc156208183)

[1.5 Modelling diagrams 12](#_Toc156208184)

[1.5.1 High-level software object model 12](#_Toc156208185)

[1.5.2 Flowchart through program 14](#_Toc156208186)

[2 Design (12 marks) 15](#_Toc156208187)

[2.1 System design overview 15](#_Toc156208188)

[2.2 Data structures 19](#_Toc156208189)

[2.2.1 Tree 19](#_Toc156208190)

[2.2.2 Graph 19](#_Toc156208191)

[2.2.3 Stacks 20](#_Toc156208192)

[2.3 Algorithms 21](#_Toc156208193)

[2.3.1 Static Evaluation 21](#_Toc156208194)

[2.3.2 BRS 21](#_Toc156208195)

[2.3.3 Djikstra’s 23](#_Toc156208196)

[2.4 File structure and organisation 23](#_Toc156208197)

[2.5 User interface design 26](#_Toc156208198)

[2.5.1 Home Screen 26](#_Toc156208199)

[2.5.2 Game Screen and overlays 27](#_Toc156208200)

[2.6 User guide 28](#_Toc156208201)

[*3* Technical Solution *(42 marks)* 29](#_Toc156208202)

[3.1 Completeness Section 29](#_Toc156208203)

[3.2 Technical Skills Section 29](#_Toc156208204)

[3.2.1 Overview Guide 29](#_Toc156208205)

[3.2.2 Code listing 32](#_Toc156208206)

[4 System testing *(8 marks)* 33](#_Toc156208207)

[4.1 Test plan 37](#_Toc156208208)

[4.2 Testing Videos 43](#_Toc156208209)

[5 Evaluation *(4 marks)* 44](#_Toc156208210)

[5.1 Comparison of project performance against the objectives 44](#_Toc156208211)

[5.2 Effectiveness of the solution 45](#_Toc156208212)

[5.3 Analysis of user feedback 45](#_Toc156208213)

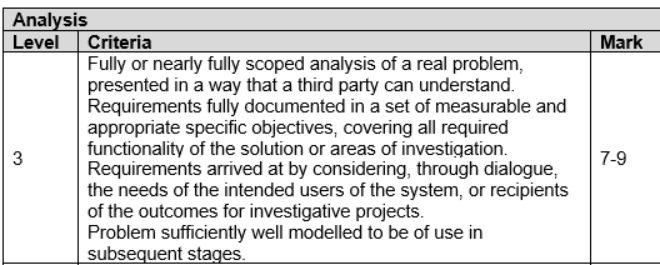
[5.4 Possible improvements 47](#_Toc156208214)

[6 Appendix 48](#_Toc156208215)

[6.1 Git log 48](#_Toc156208216)

[6.2 References to web sites or other resources used 53](#_Toc156208217)

# Analysis *(9 marks)*



In the analysis section we are looking for:

* A detailed description of the problem / investigation
* Clear evidence that research has been performed (dialogue)
* A clear set of objectives that will be useful across the later sections of the project
* Modelling of the proposed solution that will be of use to later design work

Having read the analysis stage:   
 Do you understand what the project is going to do?

Has the student set clear objectives that are detailed enough?

## Background to/ Identification of the problem

The game Settlers of Catan was created in 1995 in Germany, where players simulate settlers of new land, attempting to expand their individual empires. This is a turn based strategy board game between one and four players. Most notably, Catan is set on a hexagonal point-top tilemap, consisting of settlement pieces and resources.

An image of the board game is shown on the left. The problem identified lies intrinsically in the nature of the board game, and so digital solutions may be superior.

A screenshot of a game

Description automatically generated

This is an image of an online Catan alternative game which also supports live online multiplayer. This game also supports bots that allow the player to train and improve without needing to play other players.

### Game Rules

The Vanilla game rules require the player to roll a dice to determine where each resource is allocated. After setting up the board, the game begins from the player who roles the highest number on two die. A dice is rolled before each turn to determine which terrains produce a resource. After receiving resources, the player is given an opportunity to trade. The traditional game offers two different types of trading: domestic trading and maritime trading. In domestic trading, the player can offer other players their own resources in exchange for the other players’ resources. Maritime trading can occur at a harbour and is a fixed rate trade for any resource of the player’s choice. Finally in the building phase, players can construct settlements in specific locations. At a node (the joining point of three hexes) a village can be constructed, and later upgraded to a city. All villages need to be connected to the player’s starting capital city by roads, which must be connected to a settlement. Essentially this forces players to develop outwards from their capital. The costs of these settlements is fixed however may be changed to change the speed of the game.

There are also development cards and progress cards which can be bought or achieved. For example there is a card awarded for owning the longest sequence of roads which awards victory points.

This is still only an overview of the core rules in the game. The full instruction booklet that comes with the game can be found digitally here:

<https://www.catan.com/sites/default/files/2021-06/catan_base_rules_2020_200707.pdf>

Many of these rules could seem confusing at first which often repels newer players from immersing in this game. For example, many players do not fully understand how Maritime trading works or how to properly manage the development card system, so my solution may choose to avoid these areas.

### Possible problems with board games

Some other problems with the current board game is that small road and village play pieces can easily get lost, which makes it hard to play the game. In addition, dice can cause players to experience luck based events, which may not be fair if players are tactically superior but unlucky. There is also no way to improve at this game without carrying out independent research online or playing against oneself.

Is this a detailed description of the background/ identification? Is there evidence of an understanding of how the current system contributes to the problem? Is there evidence that the user is well understood and the context is understood? If a game or puzzle, have you described the rules in sufficient detail along with any alternative rule options that may exist? Is there a problem related to having to carry a physical game around or not having time to play a full game in one sitting and the associated issues with packing away and restarting? If a puzzle is there an issue with getting access to enough different puzzles to become proficient? Is there any way to help become better at the game or puzzle?

Generic problems with board games

## Research carried out

You must include detailed evidence of research into the background of the project including any knowledge acquisition for the problem domain. Include screen shots or diagrams of the **current system** and interview **transcripts** with the current user. Ensure questions are **detailed** and really scope out the full complexity of the new system and show how it can solve the problem. Identify any **key algorithms** that you may need to write – either ‘complex user defined’ ones to implement the rules of your program or known algorithms e.g. Minimax. Include an analysis of how deep (how many levels) you are likely to be able to go in a minimax implementation based on your product. Does your game allow for **repeat turns** and so would minimax need to be modified from its pure form? Identify any key **data structures** that might be useful in building your solution. Are there any **research papers or web articles** that describe a **playing strategy** in your product that you will be able to implement. Note that you are not implementing the algorithm, but identifying it as ‘needing to be implemented’. Make specific links from the research of existing solutions to the project objectives. Will your product use any networking (client/server) and if so what networking technology will you use (websockets?)

Add diagram to talk about algorithms and data structures

### Board data structure

A hexagons with numbers

Description automatically generatedWhen storing the board in a data structure, a weighted undirected graph seems to be the most suitable option as it is easy to store cells that are connected together by a weight. This graph may be stored as a Dictionary of positions as keys and the cell objects as values or they may simply be stored as cell objects in a list with their positions as a public property, as it will need to be accessed and updated from the main game.

The highlighted nodes on this figure represent the players’ starting locations to ensure the game is fair.

There are primarily two coordinate systems that can be employed when dealing with hexagonal grids. These are the row, column based coordinate system which acts like a map for the 2D coordinates of these hexagons on an x,y grid. The other option is the cubic system which takes each hexagons and considers the three possible directions of movement i,j,k. The following direction system may be used:

A pink cross with black letters

Description automatically generated with medium confidence

This cubic coordinate system allows for calculating the coordinates of terrain hexes from edge nodes by varying each i,j,k by one to give the new coordinates. This means that the mathematical model of the board in this manner is easy to manage as determining traversals or move legitimacy can be achieved through simple mathematical calculations on the position vectors. As outlined from this web page on hexagonal coordinates: <https://www.redblobgames.com/grids/hexagons/> there is an algorithm to convert this coordinate to odd-row coordinates which will be necessary for the graphical display of these nodes on the screen.

As for storing the board connection statuses (whether a road or wall exists and if so by which player), an adjacency list or matrix can be used. An advantage of an adjacency list is that it will take up less memory. However an adjacency matrix is very quick when looking up a specific connection between two nodes and that speed may be more important than memory if these lookups need to be made many times a second.

### AI algorithm options

When analysing multiplayer game computer bot algorithms, the main and most simple technique used is a Monte-Carlo Tree Search (MCTS). There are a few MCTS options that are considered to be the fastest and most optimized. The Max^n algorithm is an adaptation of the classic MiniMax algorithm that is extended to multiplayer by allowing the static evaluation of a position to be a n dimension vector. From extensive analysis it was proven that deep pruning is not possible in this algorithm due to the complexity of multiplayer game positions. This means that for the same time spent searching fewer positions get considered in comparison to other methods. An alternative method is the Paranoid method in which the computer is considered a maximising player and the other players are all minimising a single player’s score. It is shown that this approach can then undergo alpha beta pruning making it more efficient, despite it losing some precision as the best possible positions are often not searched for unless the correct minimising player is chosen. An algorithm that attempts to improve on both of these methods is the Best Reply Search algorithm. This algorithm approaches the problem in a similar way to Paranoid by classifying all other players as a single minimising player. However this time the algorithm searches moves from all opponent players on each turn, not just the moves of the current player. This means that many illegal positions can be reached which could be problematic for some games however should not affect Catan severely as positions are quickly changing. This means that BRS allows for alpha beta pruning like normal MiniMax which means it may be able to reach a depth of 4 or 5 moves with one second of evaluation. In addition experimental data collected on ResearchGate in 2013 has shown that BRS beats the other two approaches in all 8 games used.

<https://www.researchgate.net/figure/Results-of-max-n-vs-paranoid-vs-BRS_tbl1_259655439>

### Storing files and multiplayer

To store the board when the user is saving the game, the objects can be serialized into JSON or binary using built-in C# packages. After serialization to binary, the file can be encrypted using XOR encryption which will keep the data secure so players cannot cheat by accessing the save file.

## Identification of the prospective user(s)

My primary user is an upper sixth student Adam Smith (age 18) who enjoys playing board games with friends. He has found that he often does not get time to complete games of Catan with friends and they often get confused by some of the more complicated rules in the game. He wants a digital version of this game so he can fully immerse in the world, whilst being able to pause and return to the game later without needing to keep a board and all the pieces safe. The following transcript is from an interview conducted on June 14th 2023.

### Interview

### General Information

I: What annoys you about the games of the settlers of Catan?

P: The fact that I can't move around the board. Like I want to be more immersed in the world and it often gets boring after ten rounds. ***[Objective 3, 10]***

I: For the game and GUI, what colour schemes would you prefer to have.

P: More greens and yellows because that's less strain on eyes. The board itself should have a mix of colours depending on the resources, like lumber should be brown and metal should be silver.

I: How many players should it go up to?

P: It should be anywhere up to four players like the original game so it doesn’t become too cluttered.

I: What about music? Should there be any?

P: Some sound effects as well as background music would make it feel more like a videogame. ***[Objective 74]***

### General Game Rules

I: At the start of the game, where should each player’s capital city spawn?

P: Their capital city should spawn on a location on the edge of the board to make it more fair and challenging. ***[objective 1]***

I: What benefits are there from computer based games over board games?

P: There are pieces that you can lose and computer games are easier to save and come back to and play against people in a different location.

I: Would you think it would be better if you moved your entire empire or you had a piece which moves around and collects the resources?

P: I feel like it would be better to have a single piece that moves around, so that way you'd have to be able to prioritise your moves. ***[Objectives 3, 10]***

I: Should the gathering of materials still be based of a dice roll or since there is now a moving piece, should there be a different system?

P: There should be a new system where the resources spawn on the grid and the moving piece needs to go and collect them. ***[Objectives 2, 8]***

I: How long do you think each game should last and do you think there should be a time limit?

P: There shouldn’t be a time limit however I would like the games to last between ten to thirty minutes as it keeps the game

I: What new features would you specifically like the new game Travelers of Catan to do?

P: The base game should be the same, with the same objectives. But there should definitely be more features in the game to make it more competitive and difficult, such as the ability to move around the map. ***[Objectives 1-6]***

I: You mentioned moving around the map, how exactly should it work?

P: The moving piece should stop on a vertex on the hex grid so that it is next to three roads.

I: What resources should it collect from this position? All three?

P: I think so because it would also make the game go faster and the players need to strategize where the need to place their moving piece. ***[Objective 2, 9]***

I: Should there be any challenges to movement?

P: It would be cool if you could buy walls or barriers along with villages, cities and roads. And when you place a wall the enemies can’t move past there. You shouldn’t be able to travel on the enemies territory. ***[Objective 23]***

I: So then should you be able to travel past the enemies villages, cities and roads.

P: Definitely not traveling on the enemies roads or through their villages and cities.

I: In that case what should walls offer instead of making a village.

P: Walls shouldn’t need to be connected to your main empire so you can place a wall around a specific recourse and get a monopoly because this would open a whole new gameplay mechanic because you can expand your empire’s territory without needing to buy loads of roads and villages. ***[Objective 23]***

I: How should they be implemented?

P: Make walls appear perpendicular to roads blocking all enemy travel between the two adjacent nodes.

I: When should you be able to buy a new road, city, village or wall.

P: So you should be able to make new roads, villages and cities when your piece is anywhere as long as its connected directly to your capital. The walls should only be made at the place where your piece currently is.

I: Should your own moving pieces be able to travel through your walls?

P: Yes but it should take two moves instead of one to discourage players making loads of walls. ***[Objective 24]***

I: Should there still be a robber piece like in Settlers of Catan?

P: There should be a Highwayman piece that you can move when you roll a 7.

I: What should it do?

P: It should sit at the junction between roads, though not on a wall, village or city and when an enemy comes past, it should take all of their resources and give it to the person who moved the Highwayman there. ***[Objective 52]***

I: As discussed there is currently the road, village, city and wall to buy. Should there be anything else?

P: Maybe as the game progresses you can buy another resource collector so you have two moving pieces to control

I: What should the cost be for that?

P: Maybe instead of buying it with resources to stop people from just gathering lots of material instead of progressing the game, you should unlock a second one after creating two cities so it’s more balanced between collecting and upgrading.

I: Could it instead be after getting a certain amount of victory points which then translates to getting a certain number of cities of villages.

P: Definitely that sounds better so instead of just getting two cities it could also be one city and two villages give you a new collector.

### User Account System

I: Would you like to play with someone at the same computer as you or would it be nice to play remotely?

P: I mean, having options for both would be good, but if I'd have to pick one, then it would probably be same computer.

***This prompts the idea of multiplayer (pass around the device)***

I: What are your thoughts on having a leaderboard or a way to keep track of your progress as you play the game.

P: There should definitely be a way to look at your past victoried but I don’t really need a login system with an account leaderboard. ***[Objective 65]***

I: Would you always want to finish a game of Travelers of Catan. What are your thoughts on saving a game’s progress?

P: Yes, I would definitely want to be able to leave it for a bit and then be able to come back. ***[Objectives 63-70]***

### Bots, difficulty and additional features

I: Should the board generation have its own difficulty

P: I think it should generate a new board each game and there should be a separate difficulty for the board generation

I: You mentioned the board being a hex grid what should this look like?

P: The board is a grid of 19 hexagons like in the normal game with the recourse generators randomly distribute across the map. ***[Objective 8]***

I: What makes a board harder to play than another.

P: The resources should be more spread out so it’s harder to collect everything from just staying in one area of the board. There can also be fewer generators of one resource on the board so it’s more competitive to get. It would also make the game feel more natural and more competitive.

I: You mentioned the game should be up to four players. Should there be a one player mode or a general way to add bot opponents.

***This introduces the idea of the single player mode against the computer.***

P: Definitely, it would be fun to play with two players and two bots as well as the players can make an alliance to try and beat the bots. The bots should also have varying difficulty though. ***[Objectives 40, 41]***

I: What different factors should control the computer bot’s difficulty.

P: The harder bots should be able to strategize better which means planning moves ahead and knowing what resources it needs to collect next. It also should move the piece around the board in a more efficient way.

I: Should there be any way to team up in Travelers of Catan?

P: An alliance system would be pretty cool. You could then travel on the other person’s roads, villages and cities.

I: What are your thoughts on trading resources like in the original game?

P: It should definitely still be a feature. ***[Objectives 27-32]***

I: What about deciding if a bot should accept or start a trade?

P: The bots shouldn’t be able to start trades as that could get annoying but if you trade with a bot it should calculate if it’s a good or equal deal and accept it. ***[Objectives 33-35]***

I: What are your thoughts and talking back your moves?

P: I think that it’s probably a good feature as it means that you can undo mouse slips. However if you don’t need to think about your moves, the games could get boring with people taking back their moves.

I: How many moves should you be able to take back then?

P: Maybe just the moves from the current turn and not the previous turn.

I: What factors should affect the spawn rates of a resource?

P: The resources should be generated in the positions around the player.

I: Why does it need to be a smaller chance?

P: To discourage a player keeping their piece on their city and not moving.

I: Should there be any special features that occur during the gameplay?

P: Events that happen every five rounds would be cool like some resources spawning double or there being two Highwaymen.

I: How many events should there be per game?

P: They should keep happening until the game ends every five to ten rounds. Some ideas are changing the board like swapping resource regions.

## Numbered measurable, appropriate specific objectives of the project

### MVP

* 1. Running the program will load a new game and will log all of the hexagon positions along with the randomly generated resource. Two players will be instantiated in the game with hardcoded initial positions.
  2. At the start of a player’s turn they should receive resources based on their cities and their current position. This will be displayed to the player in text.
  3. On a player’s they may enter the coordinates of the position they wish to move to. After a player has made their move they will be given the option to make a purchase or continue.
  4. If they choose to purchase an upgrade, this will increment the players victory point count and decrease the relevant resources. However, If the player does not have enough resources to make a purchase, the program will not allow the player to complete this transaction.
  5. If the player now has achieved the winning number of victory points, the game should end and display the victor.
  6. If the player chooses continue, the next player will get a turn.

### Basic GUI

* 1. The program should launch the game screen; a graphic game should open and at the start of the player’s turn, a player GUI overlay should be opened.
  2. The game board will be formed of 19 tessellated hexagons with an image of the randomly generated resource. Both players should appear on the board with their player colour visible. At the start of each turn, the resources generated around the player and their cities should be displayed on the screen in an animation.

###### Player GUI overlay button options:

* 1. Zoom – the entire board should be shown instead of just the area around the current player.
  2. Moving – the node buttons adjacent to the current position should be activated
  3. Shopping – another GUI overlay should appear allowing the player to select which building to purchase. The resources required to purchase each building should also be displayed.
  4. Inventory Check – the current player’s resources should be displayed.
  5. End turn – the GUI will close and the GUI for the next player will be opened. The camera should also move to the new player’s position.
  6. Each round should have a timer that counts down from a defined starting time. Once this timer reaches zero the current player’s turn should forcibly end.

### Advanced Game

###### Movement in more detail:

* 1. If the user clicks on the move button, nodes that the current player is able to reach on their current turn should instead be enabled. Players can travel on their own roads consuming no moves but consume two moves when crossing their own wall. Players can not travel on other player’s connections.
  2. If the user clicks on one of these node buttons, all node buttons will be disables and the player should be moved to this position.
  3. Clicking again on the move button while the nodes buttons are enabled will disable all node buttons and should cancel the movement.

###### Shopping in more detail:

* 1. The user should be able to navigate through all of the settlements that they can purchase. If the user presses the close button, the overlay should be closed.
  2. If the user purchases an event card, they will be given one of the following options: Gain 1 victory point, Place a road, Place a village.

Settlements

* 1. If the user purchases a village, the village may only be purchased if it is connected to a road or if it passes the “distance rule” (not next to an existing settlement). If this condition is not met, an error should be displayed and nothing more should happen.
  2. If the user purchases a city, the city may only be purchased if the player stands on their own village. If this condition is not met, an error should be displayed and nothing more should happen.
  3. If the condition for a village or city is met, the node button where the player is standing should be activated and clicking on this button will complete the transaction. All villages and cities should have a flag with the player colour of the occupant.

Connections

* 1. If the user purchases a road, the road may only be purchased if the player stands on their own settlement or the player is standing next to their settlement and there is an empty connection for the road to be built. If this condition is not met, an error should be displayed and nothing more should happen.
  2. If the user purchases a wall, the wall may only be purchased there is an empty connection for the wall to be built. If this condition is not met, an error should be displayed and nothing more should happen.
  3. If the condition is met, the node buttons adjacent to the player’s position where the wall or road can be built should be activated and clicking on one of these buttons will complete the transaction.
  4. If the transaction was successful, an image of the village, city, road or wall should now be displayed at the position where it was purchased.

###### Trading:

* 1. A trading button should be added to the game overlay. If the user clicks on this button, an overlay should open offering the user a choice of all non-AI players that the current player may trade with. *This may be empty if there are no available options. If the user clicks on the cancel button, the GUI should close.*
  2. If the user selects a player option, the trading overlay will be opened with this other player.
  3. The user should be able to add and remove the current resource from the current player and this should be displayed as a number indicating the changes in resources. An overall indicator for the resultant change in resource count should also be displayed.
  4. The user should be able to navigate through all five resources. After reaching the final resource, the accept button should be activated allowing the user to accept the trade.
  5. If the trade is accepted the resources should be exchanged as specified by the user in the overlay and the GUI should close.
  6. If the trade is rejected, the GUI should close and nothing else should happen.

###### Undo actions

* 1. An undo button should be added to the game overlay. While there are actions in the current player’s turn that may be undone, the undo button should be activated. The button should begin deactivated.
  2. After moving the current player, the undo button should move the player back to their original node and should increment their remaining moves by the distance they travelled.
  3. After making a purchase, the undo should remove this purchase from the board and should refund the current player by the cost of the purchase.

###### AI

* 1. When the player adds a bot by ticking the bot checkbox in the add player overlay, the game must add an AI player to the list of players and once it is the bot’s turn, all player input buttons should be destroyed or disabled.
  2. Once the bot starts its turn, the game timer should continue counting down but the rest of the GUI should not change while the bot runs its recursive algorithm to select the best moves.
  3. After the time reaches the final five seconds or if the bot has completed its search, the best moves selected by the algorithm should be made by the AI on the board and the bot should end its turn.

### Advanced GUI

* 1. The program should launch a graphic game which opens on a “menu page” with the option to start a game, visit the about page, change the music settings or exit.

###### Start Game

* 1. If the user clicks on the “Start Game” button, the game will be sent to a game save slot selection scene which they should be able to exit back to the menu page.
  2. If the user selects a slot that has no save data, the user should be send to a game setup scene.

**Creating a New Game**

* 1. The first slot must have an add player button and any new slots that are created should begin displaying this button. If the user clicks on the button to add a player, an overlay will be opened prompting the user to enter this player's name and whether or not the player should be an AI bot.
  2. The entered player name must be alphanumeric and between 3 and 10 characters. The first player should not be allowed to tick the AI player box.
  3. If the user enters more than 10 characters, the new characters should be discarded. If the length of the name is less than 3 characters, the save button should be disabled.
  4. Once the user saves these values, this player will be shown in the current slot and the next slot will be made available until all four slots are filled in.
  5. If there are more than 1 players in the game, the remove button should be shown. If the user clicks on the remove player button, the last player added will be removed along with the newest add player button.
  6. If the user clicks on the colour button for any player, an overlay will appear allowing the user to change the colour for the player they had selected. In this overlay, colours taken by other players should appear as a player icon with a lock on it. These buttons will be disabled.
  7. The user should be able to adjust the number of victory points required for a player to win the game.
  8. The user should be able to adjust the time each player has per turn.
  9. If there are two or more players then the continue button should be activated and this button should sent the user to the game scene.

###### Game scene changes

* 1. Once a player wins, the user will be sent to the victory scene where a message should congratulate the winner of the game and show the final player rankings. If the player clicks on the home button, they will be sent back to the menu page.
  2. A highwayman should be added to the central position and on each player’s turn, they will be allowed to move the highwayman by one position. If the player crosses the highwayman, they will lose all of their resources.

###### Pause GUI

* 1. If the user clicks on the pause button, the game timer will stop and a pause overlay should be displayed giving the user the following options:
  2. Resume – the pause overlay will be destroyed and the game timer will continue.
  3. Exit – the user will be sent back to the home screen.
  4. Volume slider – the user will be able to move the volume slider from 0% to 100% to change the overall volume levels
  5. Mute Background music – the background music playing will be toggled
  6. Mute SFX – sound effects will be toggled

### Load/Save

###### Loading a Game

* 1. In the “Start Game” page, the user should now be able to select an existing save and the game should be loaded from the save. The game should continue from the last player’s turn with the remaining time left.
  2. When checking if a save slot has been used, a .json and .bin file should be accepted into the program, allowing the user to enter their own game states in JSON format as well as states serialized by the program stored in binary.
  3. If the user clicks the reset button, the game save should be deleted and allows the user to create a new game in its place.
  4. If a save is corrupted, a message should be shown to the user and the game should not attempt to load this save.

###### Saving a Game

* 1. During a game, when the user is paused if they click the “Exit” button, the game should now be saved to the slot they had selected at the start of the game in addition to sending them to the menu page.
  2. If the user closes the program abruptly in the middle of a turn, the game should still attempt to save to the selected slot.
  3. When the game is saved, the game should be serialized to a JSON file which is written to a binary file using XOR encryption that uses a secret key stored in the program.

### Final Product

* 1. On the menu page, clicking exit should close the program.
  2. On the menu page, clicking the about button should load the about page. From this page, if the user clicks the home button, the program should return to the menu page.
  3. On the menu page, clicking the settings button should load the pause overlay without the exit game button. Closing this overlay should return to the menu page.
  4. Game buttons should be animated and when traveling between scenes, a transition animation should be played.
  5. Sound effects should be heard for buttons, text displays and animations. Background music should also be heard.

## Modelling diagrams

### High-level software object model

An overview of the classes required to complete the task can be found to the left. The game is launched from the UI class. A TravelersOfCatan object is made for each game that is started, which contains information on the board and players. The board is made up of nodes and hexagon units

A diagram of a flowchart

Description automatically generated

### Flowchart through program

A diagram of a flowchart

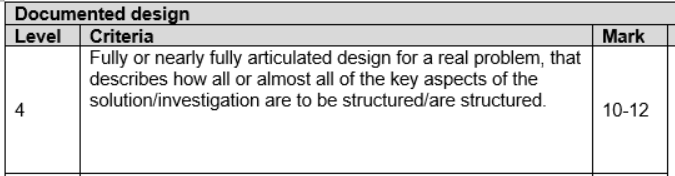
Description automatically generated

These two flowcharts demonstrate how the game will continue to alternate through players in an event based format. Every human players gets a fixed 5 minutes before their turn is forcefully ended as shown by the timer loop. After ending a turn, the game will move onto the next player and start their turn before terminating.

**Important read – Examiners Report**

Analysis There is encouragement that a student should gather details for the project from users via a dialogue of some form. Some of the interviews seen were very detailed and clearly gained relevant information for development of the project. Unfortunately, it was also common to see very short interviews which gathered no real requirements for the project to be assessed highly by centres. Students should be encouraged that in the interview it would be beneficial to ask probing questions to find out the real requirements of the user(s) and not just the kind of colours to be used or whether they like playing games. The analysis should contain a list of the objectives set by the student for their technical solution. It was pleasing to see many students provide a detailed list of objectives that indicated both the requirements to be met and the complexity that this might involve. Students who submitted vague and brief objectives would struggle to pick up high marks in the analysis section and it would also be common for the rest of the project to suffer slightly. Weak objectives also make awarding the completeness mark hard as consideration must also be placed into what an A-level student would be expected to achieve. The analysis section is to contain some modelling of the proposed system and it was pleasing to see students complete this in a variety of ways. Those projects that needed data processing usually included some discussion of the data required and DFD or ER diagrams. Students looking to produce a game sometimes struggled with the modelling section and also left the reader not understanding what their idea actually was. Students completing gaming projects could consider sketching out some ideas for the game and discussing the game flow as part of their modelling section.

# Design (12 marks)



In the documented design section we are looking for:

* An overview to the whole system design (module breakdown / objects / units / web pages …)
* Clear design & detail for some of the complex algorithms
* Detail for the data to be used (database design / data structure design)

Having read the documented design:

Do you understand how the project is going to work as a whole?

Do you have a clear understanding as to how some of the complex algorithms will work?

Do you understand how data will be processed / structured?

## System design overview

Description plus diagrams such as class diagram: For each class describe its purpose (high cohesion) and each public method (interface/low coupling)

Flowchart: Give general top-level flow of the system from running the program. May be more than one flow chart for different use cases.

Data flow diagram: Top level diagram showing how data moves between different parts of your software for different use cases e.g. logging in accesses database, saving access database.

Swim-lane diagrams for networking messages

Full UML diagram

One sentence description for every class (used in class comments)

Identifier key:

A group of icons with text

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A diagram of a computer

Description automatically generated

### Main NEAGame

*The game will be instantiated by whichever* ***UI*** *class is implemented when running the program (****TerminalUI*** *in the command line or* ***UnityUI*** *in the final version). This has been done because the game is event-based.*

The **UI** interface is used to define the methods that the game will use to interact with the user.

**TravelersOfCatan** us the main class for controlling the game. This includes creating the game board, players, and controlling the game loop.

The **Board** class is used to represent the graph of nodes and connections that make up the main fame board.

A **Node** represents a single point on the graph where a player or building can be placed.

A player may purchase a **Settlement** to place on the board. This is an abstract class which is inherited by **Building** and **Connection**.

A **Building** represents the status of a node on the board. It can be empty, a village or a city.

A **Connection** represents the status of a connection between two nodes on the game board.

A **Resource** represents the type of resource that a hexagon cell on the board produces.

A **Player** object represents a single player in the game.

The **AI** class inherits from Player and is used to represent an AI player in the game. This contains all of the AI’s logic and decision making.

The **GameAction** class represents an action that a player can make.

The **PlayerMove** class represents the game action of moving a player to a new node.

The **PlayerPurchase** class represents the game action of a player making a shop purchase.

*The implementation of the UI interface for the final product is the UnityUI.*

The **UnityUI** class is used to manage the Unity interface for the game. This class is used to manage the flow of the game and the user interface. It is a singleton class that creates and updates the game, and interfaces with all of the other UI classes.

### JSON and file handlers

The **JSON\_manager** class is used to manage the serialization and deserialization of the game to and from JSON.

A **FileHandler** object is used to manage the saving and loading of data to and from the files stored in the Unity persistent data path. This also has the option to encrypt the data before saving it to the file.

*A wrapper class was made for all of the main game classes that define a game. This is done so that a game could be serialized directly to and from JSON. The following classes were required:* ***GameWrapper, BoardWrapper, NodeWrapper, AllNodesWrapper, InventoryWrapper, PlayerWrapper, AdjMatrixWrapper, AdjMatrixBottomWrapper, SettlementWrapper, HexagonUnitWrapper, ResourceWrapper****. These all inherit from* ***JSONWrapper*** *which is an abstract class that defines the methods to serialize and deserialize an object to and from JSON.*

### Unity UI controllers

*The following classes have been made to controls Unity UI overlays and scenes which interact with the user. This includes calling events from the main UnituUI class along with animations and sound effect triggers.*

**ColorChoiceControls** is a class that manages the overlay which allows players to choose their colour.

**GamePauseOverlay** is a class that manages the pause and settings overlays.

**InventoryPopup** is a class that manages the inventory overlay that displays the current player’s resources.

**PlayerChoice** is the class that manages the overlay for selecting a player to trade with.

**PlayerNameInp** is the class that manages the overlay for inputting the player’s name and whether they are a bot.

**PlayerSetup** is the class that manages the player setup scene. It is used to display the player slots for the user to add or remove players.

**PlayerUIOverlay** is the class that manages the overlay UI for the player. This includes the buttons for game actions and pausing along with elements to display the player’s name, colour, score and time left.

**PopupController** is the class that manages the overlay to display a message to the user.

**SaveSelector** is the class that manages the game save selection overlay. It is used to display the save game slots for the user to load or create.

**ShopOverlay** is the class that manages the shop overlay. It is used to display the shop items for the user to purchase.

**TradingInterface** is the class that manages the trading overlay. It is used to display each resource along with the quantity to trade.

**VictoryManager** is the class that manages the victory scene.

### Unity animations

*The following classes have been made to temporarily store information in Unity or to add Unity animations and effects. They do not drive the game logic or process user inputs.*

**AudioManager** is the class that controls all audio for the game.

**CardCollection** animates a card on the screen to indicate the collection or trading of a resource.

**ConnectionAnimator** is a class attached to the connection game objects to control the connection between nodes.

**MenuPage** is the class that adds listeners to the buttons on the menu page.

**NodeButton** is the class that controls the node buttons on the game board. This also controls the settlement display on the nodes.

**PlayerAnimator** is the class that animates the player game object. This includes moving the player and bobbing the current player indicator.

**PlayerSlot** is the class that stores references to the UI elements of a player slot in the player setup scene.

**PlayerTradeSlot** is the class that stores references to the UI elements of a player slot in the trade partner selection overlay.

**SaveSlot** is the class that stores references to the UI elements of a game save slot, along with the ID number.

**SceneTransition** is a singleton class that handles the transition between scenes, including the cloud transition animation.

**TextAnim** is the class that handles the letter-by-letter animation of text objects.

## Data structures

Give examples of what data would be stored in the data structure in the context of your project. Do not be afraid to be verbose

Diagrams and descriptions of data structures

Include it all at different stages in the game (opening , middle game and end game)

Lots of images

Different game states

### Tree

A tree is a series of branches that form a fully connected network. My computer opponent uses a tree in the BRS MCTS algorithm. This tree originates from a single root point which is the current game position and branches over all possible moves made by the players in a maximum depth of 4 moves.

A diagram of a triangle

Description automatically generated

This is an example of the tree structure formed by the BRS algorithm. Each node represents the evaluation of a position (*the static evaluation calculation will be explained further below*), and the algorithm searches for the branch of the tree where both players make optimal moves. The tree is stored as a recursive callback of the BRS function, where each new branch is created by making another recursive call. Once a maximum tree search depth is reached, the tree gets evaluated from the leaves upwards until the root is reached.

### Graph

The board itself will be stored as a graph as discussed previously. This will allow the Node objects to store their positions in cubic coordinates which will allow the mathematical model to be optimized.

A screenshot of a cellphone

Description automatically generated

A diagram of a function

Description automatically generated

This is a zoomed in image of the game board proposed on page 5. The diagram on the left shows how a data structure of Node objects can be used to store the graph, where the connections between nodes can be stored in an *Adjacency Matrix*. The Node objects must store the position of the node however the rest of the information about them (such as the coordinates of the adjacent hexagons) will be calculated.

### Adjacency Matrix

The connections between nodes is stored in a dictionary of dictionaries, with all of the keys representing the nodes at the ends of a connection. An example of this can be shown:

A table with black text

Description automatically generated

The nodes on the first column and row represent the first and second keys used when indexing a connection from this dictionary of dictionaries. In this example, the words Empty, Wall and Road are used however in practice, these will be references to the specific connection object that is created by the board. This matrix will begin with empty connections, as no players have constructed any roads or walls. The “X” Symbol marks a specific connection that does not exist in the matrix, as a node cannot have a connection with itself.

### Stacks

A stack is a FILO data structure which stores values or events and pops them in the reverse order. This is exceptionally useful in reversing a sequence of moves, which is used by the computer bot algorithm in order to backtrack from the destination node to the current node. This will be implemented using the Stack<T> data structure which can be implemented from System.Collections built-in namespace.

## Algorithms

**Pseudocode** or similar for key algorithms essential to the success of the project. These MUST be linked into the how they fit into the project as a whole. Show how the algorithm would **affect** the **data structures** used in the context of your project. Give a **trace table/dry run** of sections of your key algorithms to **show** how they work.

### Static Evaluation

This is a simple algorithm that determines the static evaluation of a game state. This algorithm should sum the “scores” of all of the players within the game. The score may be calculated by the number of resources the player has along with their victory point count. This should be weighted to favour victory points as victory points demonstrate resources that have been spent to create and upgrade, and so are more important.

### BRS

The base BRS algorithm can be represented through the following pseudocode:

BRS(State, Alpha, Beta, Depth, playerTurn)

If (Depth <= 0)

Return StaticEval(State)

Moves -> []

If (playerTurn is Maximising)

Moves -> GenerateMoves(State, playerTurn)

turn -> Minimising

Else

Foreach (Opponent o in State)

Moves -> Moves + GenerateMoves(State, o)

turn -> Maximising

Foreach (Move m in Moves)

NewState = DoMove(State, m)

v -> -BRS(NewState, -beta, -alpha, depth-1, turn)

If v >= beta

Return v

alpha -> Max(alpha, v)

Return alpha

The BRS algorithm will be the most complex algorithm in my product. The BRS algorithm is employed by the AI opponent when trying to determine where to go and what to buy. The generate moves function I am using will therefore consider every board position along with ever possible legal purchase to cover all possible scenarios that could occur. This will ensure that the opponent makes the best possible moves with the random player variables and limited depth. When called, the initial value of alpha and beta will be plus and minus infinity.

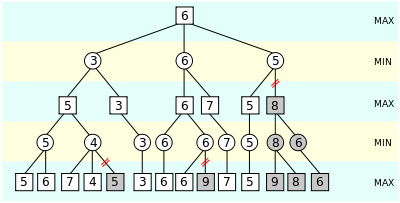
The generate moves function to deploy will scan over possible positions for the player and their possible purchases each round without considering all possible combinations of player movements. This will increase the efficiency of the algorithm exponentially as a player has up to three moves, each adding a new branch to the tree of possible moves and purchases.

A diagram of a triangle

Description automatically generated with medium confidence

This is the same example given above of a Tree data structure. From the bottom, the static evaluations of positions have been identified. In the middle layer, the maximising player will always choose the largest value. In the bottom left branches, the 3 is chosen as it is larger than the alternative branch with value 2. This repeats recursively until the root branch is reached. In this case, the highlighted path will be taken if the players make the best moves, meaning the evaluation of the current position is +3.

#### Alpha Beta Pruning



This diagram shows a tree of positions that the computer is evaluating. The static evaluation of these positions is indicated within the boxes and circles. Each layer represents the move changing from a maximising and minimising player. This graph contains an example of both shallow and deep pruning.

On the bottom left of the diagram, the grey five has been pruned. This interaction is circled in red on the diagram. This value may be pruned because the minimising player has already encountered a score of 5 when scanning the alternate branch in this position. As soon as they see a four, we know that whatever the last option is, the minimising player will chose this branch instead of the far left as they can guarantee a score of 4 or lower depending on the final value. Since the maximising player wants to chose the bigger number we therefore know that they will never go down the circled paths as the minimising player can assure a 4 or lower. Therefore no matter what the value shaded in grey is, the maximising player knows that going down the left path is better for them, so the value in the blue square is a 5 (the value from the left path evaluation).

Deep pruning works in a similar way however occurs when information is passed further down the tree, allowing for more complicated decision making, which is why the Max^N algorithm would not be suitable. However the BRS algorithm does permit a level of deep pruning as shown in the pseudocode when alpha and beta are passed down into the BRS algorithm.

### Djikstra’s

The pseudocode for Djikstra’s shortest path algorithm:

Djikstras(Graph, Start)

N -> Graph.Length

Dist[] = int[N]

Prev[] = Node[N]

Q = PriorityQueue()

Foreach Node S in Graph

Q.Push(S)

While Q is NOT empty

Current -> Node in Q with smallest distance

Remove Current from Q

Foreach Node O in Current.neighbours

NewDist -> Dist[Current] + GetConnection(Current, O)

If NewDist < Dist[O]

Dist[O] = NewDist

Prev[O] = Current

Return Dist, Prev

This algorithm will be used to determine the shortest path for the computer bot to get to its target destination by building up two arrays that can form a shortest path form the start node to any other node in the graph. To fully reconstruct the path from the starting node to the desired destination (which is determined by the BRS) a Stack can be implemented by starting from the end and pushing each precious node position onto the stack, until the starting node is reached. To then construct the correct order of moves, pop from the stack until it is empty, and the final element will be the end position.

## File structure and organisation

Show what files are going to be created and which software objects will be implemented in each file. Include folder structure for images, database file if used.

e.g JSON / bin serialization

Each game can be saved to a binary file that is encrypted using XOR encryption with a stored secret key. This means that players cannot edit a saved game state which will ensure that players do not cheat. These files are created in the permanent data folder on the user’s computer, which can be found here:

|  |  |
| --- | --- |
| Operating System | File Location |
| iOS | <Application\_Home>/Documents |
| Android | /storage/emulated/0/Android/data/AbhinavGuptaNEA/files |
| Windows | C:\Users\<username>\AppData\LocalLow\AbhinavGuptaNEA\TravelersOfCatan |
| Mac | ~/Library/Application Support/AbhinavGuptaNEA/TravelersOfCatan |

The game can generate up to four different save files. These are Save1, Save2, Save3 and Save4. This file extension of these files depends on whether encryption was enabled. If the file suffix is .json, then no encryption was used, so the user can view and edit the game JSON directly. If the file suffix .bin was used then the files are encrypted using bitwise XOR encryption so the user can not view or edit the game save.

A screenshot of a computer

Description automatically generated

Here are some images of my project structure from the Unity project assets view and the visual studio C# scripts view, along with a table containing the purpose of each of these files:

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

|  |  |
| --- | --- |
| File | Use Of File |
| namespace NEAGame | |
| TravelersOfCatan.cs | Controls a single game of Catan |
| Player.cs | Defines a single Player in a game of Catan including their resources, buildings and moves left. |
| AI.cs | Inherits from the Player class to define an AI bot that makes moves based on a recursive algorithm. |
| UIManager.cs | Contains the Interface for the game UI. |
| Connection.cs | Contains the definition, for a board connection between two nodes. |
| Node.cs | Contains the definition for a single node on the board. |
| Board.cs | Contains class definitions for the Board, Resources, Settlements and Buildings. The Board contains all of the nodes and connections. |
| global namespace | |
| UnityUI.cs | Controls the main Unity user interface and relays information to the TravelersOfCatan game object. |
| UnityUI\_game.cs | Controls the Unity user interface during the game scene including the GUI overlays for the player to interact with. This is a partial class of UnityUI. |
| SceneTransition.cs | Controls the loading of new scenes along with the scene transition animation. |
| JSON\_manager.cs | Controls all of the classes required for the JSON serialization of the main board. |
| FileHandler.cs | Controls the reading and writing of string messages to files. |
| Overlay controllers | |
| ColorChoiceControls.cs | Controls the overlay that allows the user to select a player’s color |
| GamePauseOverlay.cs | Controls the overlay for the in game Pause menu |
| InventoryPopup.cs | Controls the overlay that displays the player’s inventory |
| PlayerChoice.cs | Controls the overlay that allows the user to select a player to trade with |
| PlayerNameInp.cs | Controls the overlay that prompts the user for the player and offers an AI checkbox |
| PlayersSetup.cs | Controls the player setup overlay |
| PlayerUIOverlay.cs | Controls the main in-game player overlay |
| PopupController.cs | Controls the popup overlay |
| SaveSelector.cs | Controls the overlay that prompts the user for a game save |
| ShopOverlay.cs | Controls the shopping overlay |
| TradingInterface.cs | Controls the trading overlay between two selected players |
| VictoryManager.cs | Controls the victory scene once the game ends |
| Animation scripts | |
| AudioManager.cs | Controls the audio for the game |
| CardCollection.cs | Animates a card on the screen for the collection and trading of resources |
| ConnectionAnimator.cs | Controls the connection GameObjects in the game scene |
| MenuPage.cs | Controls button inputs on the menu page |
| NodeButton.cs | Controls the node button in the game scene |
| PlayerAnimator.cs | Controls and animates the player GameObject in the game scene |
| PlayerSlot.cs | Contains information for a single player slot in the player setup page |
| PlayerTradeSlot.cs | Contains information for a single player slot in the player trade page |
| SaveSlot.cs | Contains information for a save slot in the save menu overlay |
| TextAnim.cs | Animates text elements by making content appear letter-by-letter |

## User interface design

Provide wire-frame diagrams of the graphical user interface, positioning of buttons, different screens. Give any menu commands that would be available in a terminal interface

Sketches made before the UI is created

### Home Screen

A screenshot of a computer screen

Description automatically generated

This is a sketch for the home page with a background image showing the board game. There will be a few options to start a game, view the rules or go into game settings.

### Game Screen and overlays

A diagram of a diagram

Description automatically generated with medium confidence

This is a diagram illustrating the main game UI along with the player UI overlay for their turn. There will be buttons for all of their key actions along with keyboard shortcuts. The board will be displayer along with the ability to zoom in and out making it easy for the user to play.

A diagram of a inventory

Description automatically generated with medium confidence

There will be many more overlays for the game, for example viewing the player inventory. This overlay may look something like the diagram on the left with each card having the corresponding number of resources above it.

This diagram shows the overlay for trading with another player. It shows the two players who are trading and has all of the cards down the middle, with the count exchanged on either side for both players.

## User guide

Unity compiles projects into executable files, along with a required data folder. To begin the program, run the executable file produced from this compilation.

Write a user guide to say what software or modules the user will need to install or have available on their system, and how they should run your program from the command line

Instructions on how to start up the project ***not use it***

**Examiners report – an important read!**

It was pleasing to see good students carefully structure out the design of their technical solution. Effective use of diagrams to provide an overview of the whole system, key data requirements being identified and explained along with a breakdown of the complex parts leading to pseudo-code and/or code snippets would lead to a high mark. It was also common, however, to see a more random attempt at the design documentation including just pasting code across with no detail as to the design process or how it would link into the main system. So, for example, just providing stock algorithms for merge sort and binary search does not help the reader understand the design of the system.

Having a section titled ‘sample of SQL queries’ is not very beneficial in providing a reader an understanding as to how the system will work. Students would do better to design out a particular form/page and then discuss the algorithms required for that part of the system including the SQL queries to be used for that part. Students should be encouraged to think about the data to be used by the system. In a quiz system, for example, it would be beneficial to provide examples of the kind of question(s) to be asked. For a simulation it would be good to see how the formulas are to be used alongside, for example, a sketch of the trajectory of the projectile being modelled.

For a game a student could sketch out the grid or level and talk through, for example, the movement of any enemies.

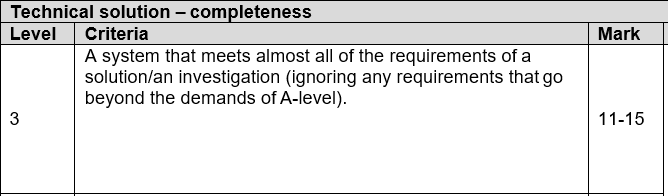
It was common to see algorithms appear without the reader having any real understanding as to how these fitted into the system and a few sketches or examples of the data to be used would help. So, for example, one student produced an excellent Sudoku solver which had some complex pseudo-code in the design section. This code was hard to understand but a few sketches of particular board layouts showing how the individual functions would perform would have really helped. It was common to see many students make use of well-known algorithms such as the merge sort. Just providing the pseudo code for this algorithm is not going to help their documented design mark. If the student talks about how this algorithm is going to be used by the system and integrated then this is beginning to pick up some credit. If the only pseudo-code or algorithm design a student attempts is based around merge sort, quick sort, binary search or other well-known algorithms without any attempt at looking at other parts of the system then the student should not be scoring highly in the documented design section.

# Technical Solution *(42 marks)*

## Completeness Section

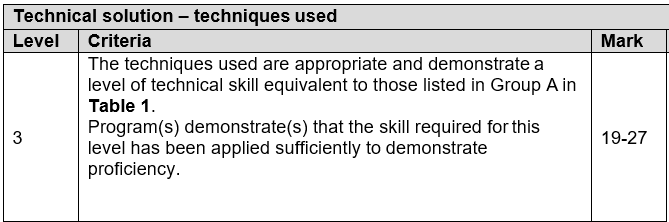
In the completeness section we are looking for a consideration as to:

* Has the project met the objectives set by the student in the analysis stage?
* How well have these objectives been met (consideration to HCI / features)?
* Does the technical solution match the original project background description?



My solution meets almost all of the requirements from the solution with exception to requirements 19 and 52. I focused the time I had to develop this project on enhancing the AI’s recursive algorithm and other areas which were more important for meeting the overall background problem.

## Technical Skills Section



In the technical skills section, we are looking for a marker to:

* Identify parts of the code where complexity is clearly evident and map to Group A/B/C  
  **[This can be helped by a student producing an overview guide]**
* Place consideration into the coding style and comment upon this
* Consider the overall effectiveness of the final solution (does it work how it should?)

### Overview Guide

#### Skills table

The following table demonstrates the skills that I have achieved within my project:

|  |  |  |  |
| --- | --- | --- | --- |
| **Technical Skill Group** | **Skill** | **Description** | **Location in project** |
| A | Graph traversal | The Djikstra’s algorithm is used to find the shortest distance of all nodes in the graph from the current player’s position. This is used to determine which nodes are reachable with the remaining moves so that these nodes can be displayed to the user when they attempt to move. | TravelersOfCatan.cs   * Dijkstras() |
| A | Recursive algorithms | BRS is used to recursively make a combination of a purchase and movement and analyse how effective these moves are for maximising the AI’s score. The AI will prioritise making purchases of buildings, then connections as these are more effective for increasing score. | AI.cs   * BRS() |
| A | Tree traversal | BRS is used to perform a depth first traversal of a tree of evaluations in positions it is searching. This ensures that the path chosen by the AI is the best possible move it can play with the given depth. | AI.cs   * BRS() |
| A | Stack operations | A stack has been used to keep an ordered collection of moves within the AI move generation algorithm. This is helpful for undoing moves or reversing the order of a sequence. | AI.cs   * BRS() * GenerateMoves()   TravelersOfCatan.cs   * DisplayAIMoves() |
| A | Complex Mathematical Model | The board is stored as a graph of vector3 coordinates representing nodes in cubic space which allows the positions of adjacent nodes to be calculated mathematically instead of storing this data as pointers. Algorithms are also needed to calculate central positions between nodes and convert between coordinate systems. | UnityUI\_Game.cs   * Line   Board.cs   * Line |
| A | Complex data structure | A dictionary of dictionaries has been used to associate two adjacent nodes to the connection between them using an adjacency matrix.  Another dictionary of dictionaries is used to store the costs of settlements based on Resource objects and their counts. | Board.cs   * Line   TravelersOfCatan.cs   * Line |
| A | Complex user-defined use of OOP: | The entire project uses Object-Oriented Programming by organising the design of the solution around data objects. |  |
| Interfaces | The user interface object of the main TravelersOfCatan class is an instance of the UI interface created. This interface was made to abstract the roles of the user interface whilst allowing the game to used defines methods to request user inputs. | UIManager.cs   * Line |
| Encapsulation | The main game classes are encapsulated to hide information and limit access to elements within the game. One example can be seen through the use of access modifiers used in the TravelersOfCatan class. | TravelersOfCatan.cs   * Line |
| Composition | Many has-a relations exist within the OOP structure of this project, which allow objects to interact with each other efficiently. An example of this is the Board class which contains lists and dictionaries of the Node, Resource and Connection objects that are needed to represent a game board. | Board.cs   * Line |
| Inheritance | The AI class inherits from a Player class as the AI needs to have all of the properties and methods a regular player has along with the additional functionality of the best move searching algorithms.  The Building and Connection classes inherit from Settlement which define some necessary attributes of these classes. | AI.cs   * Line   Board.cs   * Line |
| A | Dynamic generation of objects | The generate moves method in the AI creates a GameAction object of the type PlayerMove or PlayerPurchase depending on the type of move the player has made. Then in the TravelersOfCatan class, the code then identifies the type of the GameAction in order to compute them individually. | AI.cs   * GenerateMoves() |
| A | Complex User-Defined Algorithm | The BRS algorithm has been modified from the original design to account for dynamically adding and removing resources from passive collection. | AI.cs   * BRS() |
| B | Reading/Writing to files | The file save load function serializes the TravelersOfCatan object into JSON and writes this into a file after applying an XOR encryption so the user cannot change these values manually. | FileHandler.cs   * Save() * Load() |

#### Coding Style

The following table demonstrates some techniques that I implemented to follow good programming practices that were adhered to in my project:

|  |  |  |
| --- | --- | --- |
| **Coding Practice** | **Location in project** | **Description** |
| Whitespace and Indentation | Everywhere | Whitespaces have been used throughout the program to add spaces between words and symbols which aids in the code readability.  Indentation has been maintained throughout the code so that new blocks of code that are parts of different loops, methods or classes can clearly be identified. |
| Variable Names | Everywhere | The variable names used have been named appropriately so that it is clear what each variable is used for within the code. |
| Commenting | Everywhere | Comments have been added to all classes along with some complex methods and complex algorithms to explain to the reader/reviewer what sections of code are doing. |
| User of constants | TravelersOfCatan.cs  JSON\_manager.cs | Readonly variables have been employed to store critical information that does not change during the runtime of the program. This is a good practice as any hardcoded values can be altered from one place in the project. |
| Public Private Variables | Everywhere | Variables have been made public and private depending on the access requirement in the project. |

### Code listing

https://highlight.hohli.com/index.php

**Main Game**

#### TravelersOfCatan.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Numerics;
4. **using** System.Linq;
6. **namespace** NEAGame
7. {
8. */// <summary>*
9. */// <c>TravelersOfCatan</c> is the main class for controlling the game. This includes creating the game board, players, and controlling the game loop.*
10. ***/// </summary>***
11. **public** **class** TravelersOfCatan
12. {
13. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
14. */// The following variables are readonly meaning they can not be changed after first assigned. These control the game rules of Catan.*
15. ***/// These also take the form of dictionaries and nested dictionaries.***
16. */// Coding Style: Use of Constants*
17. */// Skill A: Complex Data Structure*
18. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
19. **public** **static** **readonly** IDictionary<**string**, **int**> VICTORY\_POINT\_CONVERTOR = new Dictionary<**string**, **int**>()
20. **{**
21. {"Road", 0},
22. {"Wall", 1},
23. {"Village", 3 },
24. {"City", 5 }
25. **};**
27. **private** **static** **readonly** **int** MAX\_MOVES = 3;
29. **private** **static** **readonly** Dictionary<**string**, Dictionary<Resource, **int**>> PURCHASE\_COST = new Dictionary<**string**, Dictionary<Resource, **int**>>()
30. **{**
31. {"Road", new Dictionary<Resource, **int**>() { { new Resource(1), 1 }, { new Resource(2), 1 }, { new Resource(3), 0 }, { new Resource(4), 1 }, { new Resource(5), 0 } } },
32. {"Wall", new Dictionary<Resource, **int**>() { { new Resource(1), 2 }, { new Resource(2), 0 }, { new Resource(3), 0 }, { new Resource(4), 2 }, { new Resource(5), 0 } } },
33. {"Village", new Dictionary<Resource, **int**>() { { new Resource(1), 1 }, { new Resource(2), 2 }, { new Resource(3), 0 }, { new Resource(4), 1 }, { new Resource(5), 1 } } },
34. {"City", new Dictionary<Resource, **int**>() { { new Resource(1), 1 }, { new Resource(2), 0 }, { new Resource(3), 3 }, { new Resource(4), 1 }, { new Resource(5), 2 } } }
35. **};**
37. **private** **static** **readonly** Vector3[] STARTING\_COORDS = new Vector3[] {
38. new Vector3( 0, 3, -2),
39. new Vector3( 1, -2, 3),
40. **new Vector3( 3, -1, -1),**
41. new Vector3(-2, 2, 2)
42. };
44. **public** **readonly** **int** WinningVictoryPoints;
45. **public readonly int StartingResourceCount;**
46. **public** **readonly** **float** TimePerMove;

49. **private** UI UserInterface;
50. **private int turn = 0;**
51. **private** Player currentPlayer;
52. **public** List<Player> gamePlayers = new List<Player>();
53. **public** Board board;
55. ***// Dijkstra's algorithm variables***
56. **private** Dictionary<Node, **int**> distance = new Dictionary<Node, **int**>();
57. **private** Dictionary<Node, Node> previous = new Dictionary<Node, Node>();
59. *// AI best move search variables*
60. **public bool isAICalculation;**
61. **public** Stack<GameAction> actions = new Stack<GameAction>();
63. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
64. */// Lots of the variables created above are private and can only be accessed by the methods in this class.*
65. ***/// This limits the scope of the variables to only the methods that need them, which is a good example of encapsulation.***
66. */// Skill A: Complex User-Defined OOP - Encapsulation*
67. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*

70. ***/// <summary>***
71. */// Constructor for a new game*
72. */// </summary>*
73. **public** TravelersOfCatan(UI ui, **int** win, **int** start, **float** maxtime)
74. {
76. UserInterface = ui;
77. WinningVictoryPoints = win;
78. StartingResourceCount = start;
79. TimePerMove = maxtime;
81. }
83. */// <summary>*
84. */// Constructor for loading a game from a save file*
85. ***/// </summary>***
86. **public** TravelersOfCatan(UI ui, GameWrapper game)
87. {
88. UserInterface = ui;
89. *// retrieve the game settings from the save*
90. **WinningVictoryPoints = game.winVictoryPoints;**
91. TimePerMove = game.timePerMove;
92. *// add all of the players back to the game*
93. **foreach** (PlayerWrapper pl **in** game.allPlayers)
94. {
95. **if (pl.isAI)**
96. gamePlayers.**Add**(new AI(pl, **this**));
97. **else**
98. gamePlayers.**Add**(new Player(pl));
99. }
101. board = new Board(game.board);
102. **foreach** (NodeWrapper n **in** game.board.nodes.\_Values)
103. {
104. Node node = new Node(n);
105. **foreach (Player pdl in gamePlayers)**
106. {
107. **if** (node.status.GetOccupant() == pdl.GetID())
108. {
109. pdl.addBuilding(node);
110. **break;**
111. }
112. }
113. }
114. *// load all of the connections made in the game so far*
115. **for (int i = 0; i < game.board.connections.\_Keys.Count; i++)**
116. {
117. Vector3 pos1 = new Vector3(game.board.connections.\_Keys[i].x, game.board.connections.\_Keys[i].y, game.board.connections.\_Keys[i].z);
118. **for** (**int** j = 0; j < game.board.connections.\_Values[i].\_Keys.Count; j++)
119. {
120. **Vector3 pos2 = new Vector3(game.board.connections.\_Values[i].\_Keys[j].x, game.board.connections.\_Values[i].\_Keys[j].y, game.board.connections.\_Values[i].\_Keys[j].z);**
121. Connection con = new Connection(game.board.connections.\_Values[i].\_Values[j]);
122. board.UpdateConnection(pos1, pos2, con);
123. UserInterface.UpdateConnection(board.GetNode(pos1), board.GetNode(pos2), con);
125. **foreach (Player pdl in gamePlayers)**
126. {
127. **if** (con.GetOccupant() == pdl.GetID())
128. {
129. pdl.addConnection(con);
130. **break;**
131. }
132. }
133. }
134. }
136. turn = game.turn;
137. currentPlayer = gamePlayers[turn];
138. **if** (HasWinner())
139. {
140. **FindWinner(); *// If the game is already won, display the winner***
141. }
142. **else**
143. {
144. UserInterface.BeginGame(game.timer);
145. **UserInterface.CreatePopup($"Resuming game from player {turn+1}'s turn");**
146. }
147. }
149. *// function to add a player to the game*
150. **public void AddPlayer(string Name, string color)**
151. {
152. **int** i = gamePlayers.Count;
153. gamePlayers.**Add**(new Player(playerNumber: i + 1,playerName: Name, color: color, origin: STARTING\_COORDS[i]));
154. }
156. *// function to add an AI to the game*
157. **public** **void** AddAI(**string** Name, **string** color)
158. {
159. **int** i = gamePlayers.Count;
160. **gamePlayers.Add(new AI(playerID: i + 1, name: Name, playerColor: color, home: STARTING\_COORDS[i], this));**
161. }
163. *// AI function to update the current player during the AI calculation*
164. **public** **void** UpdateCurrentPlayer(**int** id)
165. **{**
166. **foreach** (Player pdl **in** gamePlayers)
167. {
168. **if** (pdl.GetID() == id)
169. {
170. **currentPlayer = pdl;**
171. }
172. }
173. currentPlayer.moves = MAX\_MOVES;
174. }
176. *// get the cost of a structure*
177. **public** **static** Dictionary<Resource, **int**> GetCostOfUpgrade(**string** entityName)
178. {
179. **return** PURCHASE\_COST[entityName];
180. **}**
182. *// function to start a new created game*
183. **public** **void** startGame()
184. {
186. board = new Board();
188. **foreach** (Player current **in** gamePlayers)
189. {
191. Building capital = new Building("City", current.GetID());
192. board.GetNode(current.position).status = capital;
193. current.addBuilding(board.GetNode(current.position));
195. ***// give every player their starting resources***
197. **for** (**int** i = 1; i < Resource.RESOURCES.Length; i++)
198. {
199. current.addResource(new Resource(i), StartingResourceCount);
200. **}**
202. }
204. turn = 0;
205. **currentPlayer = gamePlayers[0];**
206. currentPlayer.moves = MAX\_MOVES;
207. UserInterface.BeginGame(-1f);
208. UserInterface.CreatePopup(gamePlayers.Count.ToString() + " players have joined the game");
210. **}**
212. *// function to get the current game time*
213. **public** **float** GetGameTime()
214. {
215. **return UserInterface.GetTimer();**
216. }
218. *// function to get the current player*
219. **public** Player GetCurrentPlayer()
220. **{**
221. **return** currentPlayer;
222. }
224. *// function to start the current player's turn + gather resources. also called to start a loaded game*
225. **public void StartTurn(float timeleft=-1f)**
226. {
227. **if** (timeleft == -1f)
228. {
229. gatherResources(currentPlayer.GetID());
230. **timeleft = TimePerMove;**
231. }
232. *// selection based on the class type of the current player*
233. **if** (currentPlayer.GetType() == typeof(Player))
234. {
235. **isAICalculation = false;**
236. }
237. **else** **if** (currentPlayer.GetType() == typeof(AI))
238. {
239. *// AI has the default time limit because when the new game is loaded, the AI calculation is restarted.*
240. **timeleft = TimePerMove;**
241. isAICalculation = **true**;
242. }
243. UserInterface.BeginTurn(timeleft);
244. }
246. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
247. */// <summary>*
248. */// Display's the AI's chosen moves after calculation*
249. */// <br/> Skill A: Stack Operations*
250. ***/// </summary>***
251. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
252. **public** **void** DisplayAIMoves()
253. {
254. currentPlayer = gamePlayers[turn];
255. **Stack<GameAction> selectedMoves = ((AI)currentPlayer).selectedMoves;**
256. Stack<GameAction> firstUndo = ((AI)currentPlayer).currentMove;
257. *// reverse order of stack*
258. Stack<GameAction> temp = new Stack<GameAction>();
259. **while** (selectedMoves.Count > 0)
260. **{**
261. temp.Push(selectedMoves.Pop());
262. }
264. **while** (firstUndo.Count > 0)
265. **{**
266. GameAction actToUndo = firstUndo.Pop();
267. UpdateCurrentPlayer(actToUndo.playerID);
268. UndoAction(actToUndo);
269. }
271. UpdateCurrentPlayer(gamePlayers[turn].GetID()); *// sets turn back to current player!*
272. isAICalculation = **false**;
273. **while** (temp.Count > 0)
274. {
275. **DoAction(temp.Pop());**
276. }
277. }
279. *// function to end the current player's turn and continue the game*
280. **public void EndTurn()**
281. {
282. actions.Clear();
283. **if** (HasWinner()) **return**;
284. turn++;
285. **if (turn >= gamePlayers.Count)**
286. {
287. turn = 0;
288. }
289. currentPlayer = gamePlayers[turn];
290. **currentPlayer.moves = MAX\_MOVES;**
291. StartTurn();
292. }
294. *// function to end the game when there is a winner.*
295. **protected void FindWinner()**
296. {
297. **foreach** (Player p **in** gamePlayers)
298. {
299. **if** (p.getVictoryPoints() >= WinningVictoryPoints)
300. **{**
301. UserInterface.HandleWinner(p);
302. **return**;
303. }
304. }
305. **}**
307. *// Check if there is a winner*
308. **public** **bool** HasWinner()
309. {
310. **return gamePlayers.Any(p => p.getVictoryPoints() >= WinningVictoryPoints);**
311. }
313. *// Generates resources around the player with the given ID*
314. **public** **void** gatherResources(**int** ID)
315. **{**
316. Player pdl = gamePlayers.Find(x => x.GetID() == ID);
318. **foreach** (Vector3 u **in** board.GetNode(pdl.position).GetHexNeighbours())
319. {
320. **if (board.GetHexAtPosition(u) != null)**
321. {
322. **if** (board.GetHexAtPosition(u).ToString() == "Empty") **continue**;
323. pdl.addResource(board.GetHexAtPosition(u));
324. **if** (!isAICalculation)
325. **UserInterface.ShowResource(u, board.GetHexAtPosition(u), Vector3.Zero);**
326. }
327. }
328. *// also get resources from all cities*
329. **foreach** (Node n **in** pdl.GetBuildings())
330. **{**
331. **if** (n.status.GetStatus() == "City")
332. {
333. **foreach** (Vector3 u **in** n.GetHexNeighbours())
334. {
335. **if (board.GetHexAtPosition(u) != null)**
336. {
337. **if** (board.GetHexAtPosition(u).ToString() == "Empty") **continue**;
338. pdl.addResource(board.GetHexAtPosition(u));
339. **if** (!isAICalculation)
340. **UserInterface.ShowResource(u, board.GetHexAtPosition(u), Vector3.Zero);**
341. }
342. }
343. }
344. }
345. **}**
347. *// Undo the resources gathered by the player with the given ID*
348. **public** **void** undoGatherResources(**int** ID)
349. {
350. **Player pdl = gamePlayers.Find(c => c.GetID() == ID);**
352. **foreach** (Vector3 u **in** board.GetNode(pdl.position).GetHexNeighbours())
353. {
354. **if** (board.GetHexAtPosition(u) != **null**)
355. **{**
356. **if** (board.GetHexAtPosition(u).ToString() == "Empty") **continue**;
357. pdl.removeResource(board.GetHexAtPosition(u));
358. }
359. }
361. **foreach** (Node n **in** pdl.GetBuildings())
362. {
363. **if** (n.status.GetStatus() == "City")
364. {
365. **foreach (Vector3 u in n.GetHexNeighbours())**
366. {
367. **if** (board.GetHexAtPosition(u) != **null**)
368. {
369. **if** (board.GetHexAtPosition(u).ToString() == "Empty") **continue**;
370. **pdl.removeResource(board.GetHexAtPosition(u));**
371. }
372. }
373. }
374. }
375. **}**
377. *// complete the trade between the current player and the other player*
378. **public** **void** CompleteTrade(Player otherPlayer, Dictionary<Resource, **int**> CurrentPlayerChange)
379. {
380. **foreach (var entry in CurrentPlayerChange)**
381. {
382. **int** change = Math.Abs(entry.**Value**);
383. **if** (entry.**Value** > 0)
384. {
385. **currentPlayer.addResource(entry.Key, change);**
386. otherPlayer.removeResource(entry.Key, change);
387. **for** (**int** \_ = 0; \_ < change; \_++)
388. UserInterface.ShowResource(otherPlayer.position, entry.Key, currentPlayer.position);
389. }
390. **else if (entry.Value < 0)**
391. {
392. currentPlayer.removeResource(entry.Key, change);
393. otherPlayer.addResource(entry.Key, change);
394. **for** (**int** \_ = 0; \_ < change; \_++)
395. **UserInterface.ShowResource(currentPlayer.position, entry.Key, otherPlayer.position);**
396. }
397. }
398. }
400. ***// method to deduct the cost of a settlement from the player's inventory***
401. **public** **void** ChargePlayer(**string** structure)
402. {
403. Dictionary<Resource, **int**> cost = GetCostOfUpgrade(structure);
404. **foreach** (**var** entry **in** cost)
405. **{**
406. currentPlayer.removeResource(entry.Key, entry.**Value**);
407. }
408. **if** (!isAICalculation)
409. {
410. **if (HasWinner())**
411. {
412. FindWinner();
413. }
414. **else** **if** (!currentPlayer.isPlayerAI())
415. **{**
416. UserInterface.CreatePopup("Purchase Successful");
417. }
418. }
419. }
421. *// method used in the UI to show the player what they need to buy a settlement*
422. **public** Dictionary<Resource, **int**> GetDifference(**string** structure)
423. {
424. Dictionary<Resource, **int**> res = new Dictionary<Resource, **int**>();
425. **Dictionary<Resource, int> diff = GetCostOfUpgrade(structure);**
426. **foreach** (**var** entry **in** diff)
427. {
428. res.**Add**(entry.Key, Math.Min(entry.**Value**, currentPlayer.getResources()[entry.Key] - entry.**Value**));
429. }
431. **return** res;
432. }
434. *// method to check if the player can afford a settlement*
435. **public bool CheckCosts(string structure)**
436. {
437. Dictionary<Resource, **int**> cost = GetCostOfUpgrade(structure);
438. **bool** canAfford = **true**;
439. **foreach** (**var** entry **in** cost)
440. **{**
441. **if** (currentPlayer.getResources()[entry.Key] < entry.**Value**)
442. {
443. canAfford = **false**;
444. }
445. **}**
446. **return** canAfford;
447. }
449. *// method to get possible locations for a player to build a road*
450. **public List<Node> tryPurchaseRoad()**
451. {
452. **if** (!CheckCosts("Road"))
453. {
454. **return** new List<Node>();
455. **}**
456. List<Node> viableLocations = new List<Node>();
457. Node otherPos;
458. **bool** canconnect = board.GetNode(currentPlayer.position).status.GetOccupant() == currentPlayer.GetID(); *// player must be standing on their own settlement to build a road*
459. **bool** allSlotsTaken = **true**;
460. **foreach (Vector3 vOther in board.GetNode(currentPlayer.position).GetNodeNeighbours() )**
461. {
462. **if** (board.GetNode(vOther) == **null**) **continue**; *// this is a null node (out of bounds*
464. Connection con = board.GetConnection(currentPlayer.position, vOther);
465. **otherPos = board.GetNode(vOther);**
466. **if** (con.GetStatus() != "Empty")
467. {
468. **continue**; *// can not build a road on existing connections of any sort*
469. }
470. **allSlotsTaken = false;**
471. **if** ((otherPos.status.GetOccupant() != currentPlayer.GetID()) && (otherPos.status.GetStatus() != "Empty"))
472. {
473. **continue**; *// The enemy controls the settlement at the end of this road*
474. }
475. **else if (!canconnect && (otherPos.status.GetOccupant() != currentPlayer.GetID()))**
476. {
477. **continue**; *// player must be connecting to their own settlement*
478. }
479. **else**
480. **{**
481. viableLocations.**Add**(otherPos);
482. }
483. }
484. **if** (!isAICalculation)
485. **{**
486. **if** (viableLocations.Count == 0)
487. {
488. **if** (allSlotsTaken)
489. {
490. **UserInterface.CreatePopup("All possible connections have already been made from your current position.");**
491. }
492. **else**
493. {
494. UserInterface.CreatePopup("One end of the road must be connected to your settlement. You can not purchase a road from your current position.");
495. **}**
496. }
497. **else**
498. {
499. UserInterface.GetUserNodeChoice(viableLocations.ToArray(), purchaseRoad);
500. **}**
501. }
502. **return** viableLocations;
503. }
505. ***// method to get possible locations for a player to build a wall***
506. **public** List<Node> tryPurchaseWall()
507. {
508. **if** (!CheckCosts("Wall"))
509. {
510. **if (!isAICalculation)**
511. UserInterface.CreatePopup("You can not afford this");
512. **return** new List<Node>();
513. }
515. **List<Node> viableLocations = new List<Node>();**
516. Node otherPos;
517. *// player must be standing on their own settlement to build a road*
518. **bool** canconnect = board.GetNode(currentPlayer.position).status.GetOccupant() == currentPlayer.GetID();
519. **foreach** (Vector3 vOther **in** board.GetNode(currentPlayer.position).GetNodeNeighbours())
520. **{**
521. **if** (board.GetNode(vOther) == **null**) **continue**; *// this is a null node (out of bounds)*
523. Connection con = board.GetConnection(currentPlayer.position, vOther);
524. otherPos = board.GetNode(vOther);
525. **if (con.GetStatus() != "Empty")**
526. {
527. **continue**; *// can not build a road on existing connections of any sort*
528. }
529. **else**
530. **{**
531. viableLocations.**Add**(otherPos);
532. }
533. }
534. **if** (!isAICalculation)
535. **{**
536. **if** (viableLocations.Count == 0)
537. {
538. UserInterface.CreatePopup("All possible connections have already been made from your current position.");
539. }
540. **else**
541. {
542. UserInterface.GetUserNodeChoice(viableLocations.ToArray(), purchaseWall);
543. }
544. }
545. **return viableLocations;**
547. }
549. *// method to see if the player can purchase a village*
550. **public Node tryPurchaseVillage()**
551. {
552. **if** (!CheckCosts("Village"))
553. {
554. **return** **null**;
555. **}**
556. Node otherPos;
557. Node current = board.GetNode(currentPlayer.position);
558. **if** (current.status.GetStatus() != "Empty")
559. {
560. **if (!isAICalculation)**
561. UserInterface.CreatePopup("You can not build a settlement on an existing settlement");
562. **return** **null**;
563. }
565. **bool DistanceRule = true;**
566. **bool** isConnecting = **false**;
567. **bool** road = **false**;
568. **foreach** (Vector3 vOther **in** board.GetNode(currentPlayer.position).GetNodeNeighbours())
569. {
570. **if (board.GetNode(vOther) == null) continue; *// this is a null node (out of bounds)***
572. Connection con = board.GetConnection(currentPlayer.position, vOther);
573. otherPos = board.GetNode(vOther);
575. **if (con.GetOccupant() != currentPlayer.GetID() && con.GetStatus() == "Road")**
576. {
577. *// can not build a settlement if an enemy road connects to this node*
578. road = **true**;
579. isConnecting = **false**;
580. **DistanceRule = false;**
581. **break**;
582. }
583. **else** **if** (con.GetOccupant() == currentPlayer.GetID() && con.GetStatus() == "Road")
584. {
585. ***// node is connected to a road owned by the player***
586. isConnecting = **true**;
587. }
589. **if** (otherPos.status.GetStatus() != "Empty")
590. **{**
591. *// breaches distance rule but may still be valid*
592. DistanceRule = **false**;
593. }
594. }
595. **if (isConnecting || DistanceRule)**
596. {
597. **if** (!isAICalculation)
598. UserInterface.GetUserNodeChoice(new Node[] { current }, purchaseVillage);
599. **return** current;
600. **}**
601. **else**
602. {
603. **if** (!isAICalculation)
604. {
605. **if (road)**
606. UserInterface.CreatePopup("You may not build a settlement on the end of an enemy road.");
607. **else**
608. UserInterface.CreatePopup("You may not build a settlement next to another settlement unless a road connects them.");
609. }
610. **return null;**
611. }
612. }
614. *// method to see if the player can purchase a city*
615. **public Node tryPurchaseCity()**
616. {
617. **if** (!CheckCosts("City"))
618. {
619. **return** **null**;
620. **}**
621. Node current = board.GetNode(currentPlayer.position);
622. **if** (current.status.GetStatus() == "Village" && current.status.GetOccupant() == currentPlayer.GetID())
623. {
624. **if** (!isAICalculation)
625. **UserInterface.GetUserNodeChoice(new Node[] { current }, purchaseCity);**
626. **return** current;
627. }
628. **else**
629. {
630. **if (!isAICalculation)**
631. UserInterface.CreatePopup("Player must be on their village to upgrade to a city.");
632. **return** **null**;
633. }
634. }
636. *// method to complete the purchase of a road at the given location*
637. **public** **void** purchaseRoad(Node other)
638. {
639. board.UpdateConnection(currentPlayer.position, other.position, new Connection(status: "Road", occupant: currentPlayer.GetID()));
640. **Connection con = board.GetConnection(currentPlayer.position, other.position);**
641. currentPlayer.addConnection(con);
642. currentPlayer.addVictoryPoints(VICTORY\_POINT\_CONVERTOR["Road"]);
643. actions.Push(new PlayerPurchase(currentPlayer.GetID(), currentPlayer.position, "Road", other.position));
644. ChargePlayer("Road");
645. **if (!isAICalculation)**
646. {
647. UserInterface.UpdateConnection(other, board.GetNode(currentPlayer.position), con);
648. }
649. }
651. *// method to complete the purchase of a wall at the given location*
652. **public** **void** purchaseWall(Node other)
653. {
654. **if** (board.GetConnection(currentPlayer.position, other.position).GetStatus() != "Empty")
655. **{**
656. **return**;
657. }
658. board.UpdateConnection(currentPlayer.position, other.position, new Connection(status: "Wall", occupant: currentPlayer.GetID()));
659. Connection con = board.GetConnection(currentPlayer.position, other.position);
660. **currentPlayer.addConnection(con);**
661. currentPlayer.addVictoryPoints(VICTORY\_POINT\_CONVERTOR["Wall"]);
662. actions.Push(new PlayerPurchase(currentPlayer.GetID(), currentPlayer.position, "Wall", other.position));
663. ChargePlayer("Wall");
664. **if** (!isAICalculation)
665. **UserInterface.UpdateConnection(other, board.GetNode(currentPlayer.position), con);**
666. }
668. *// method to complete the purchase of a village*
669. **public** **void** purchaseVillage(Node otherPos)
670. **{**
671. board.GetNode(currentPlayer.position).status = new Building("Village", currentPlayer.GetID());
672. currentPlayer.addBuilding(board.GetNode(currentPlayer.position));
673. currentPlayer.addVictoryPoints(VICTORY\_POINT\_CONVERTOR["Village"]);
674. actions.Push(new PlayerPurchase(currentPlayer.GetID(), currentPlayer.position, "Village"));
675. **ChargePlayer("Village");**
676. **if** (!isAICalculation)
677. UserInterface.UpdateSettlement(otherPos);
678. }
680. ***// method to complete the purchase of a city***
681. **public** **void** purchaseCity(Node otherPos)
682. {
683. board.GetNode(currentPlayer.position).status.UpgradeVillage();
684. currentPlayer.addVictoryPoints(VICTORY\_POINT\_CONVERTOR["City"]);
685. **actions.Push(new PlayerPurchase(currentPlayer.GetID(), currentPlayer.position, "City"));**
686. ChargePlayer("City");
687. **if** (!isAICalculation)
688. UserInterface.UpdateSettlement(otherPos);
689. }

692. *// method to get the possible locations for the player to move to*
693. **public** List<Node> attemptPlayerMove()
694. {
696. List<Node> viableLocations = new List<Node>();
697. Dijkstra(board, currentPlayer.position);
698. *// use a Linq to only select the nodes that are within the players movement range*
699. viableLocations = distance.**Where**(x => x.**Value** <= currentPlayer.moves).**Select**(x => x.Key).ToList();
700. ***// sort viable locations by distance descending (AI optimisation)***
701. viableLocations = viableLocations.OrderByDescending(x => distance[x]).ToList();
703. **if** (viableLocations.Count == 0 && currentPlayer.moves == MAX\_MOVES && !isAICalculation)
704. {
705. **UserInterface.CreatePopup("Something went wrong... Sending you to your capital");**
706. currentPlayer.position = currentPlayer.GetCapital();
707. Stack<Node> home = new Stack<Node>();
708. home.Push(board.GetNode(currentPlayer.origin));
709. UserInterface.UpdatePlayer(home);
711. }
712. **if** (!isAICalculation)
713. UserInterface.GetUserNodeChoice(viableLocations.ToArray(), MovePlayer); *// get the user to choose a location*
714. **return** viableLocations;
715. **}**
717. *// method to move the player to the given location*
718. **private** **void** MovePlayer(Node otherpos)
719. {
720. **if (!isAICalculation)**
721. {
722. Dijkstra(board, currentPlayer.position);
723. actions.Push(new PlayerMove(currentPlayer.GetID(), currentPlayer.position, otherpos.position));
724. Node current = otherpos;
725. **Stack<Node> path = new Stack<Node>();**
726. **while** (current != board.GetNode(currentPlayer.position))
727. {
728. path.Push(current);
729. current = previous[current];
730. **}**
731. UserInterface.UpdatePlayer(path); *// used to animate the player moving along the path*
732. currentPlayer.moves -= distance[otherpos];
733. }
734. currentPlayer.position = otherpos.position;
736. }
738. *// method to refund the player for a structure*
739. **public** **void** Refund(**string** structure)
740. **{**
742. Dictionary<Resource, **int**> cost = GetCostOfUpgrade(structure);
743. **foreach** (**var** entry **in** cost)
744. {
745. **currentPlayer.addResource(entry.Key, entry.Value);**
746. }
747. }
749. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
750. ***/// <summary>***
751. */// Method to calculate the shortest path to all nodes from the passed start position<br/>*
752. */// Skill A: Graph Traversal*
753. */// </summary>*
754. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
755. **public void Dijkstra(Board board, Vector3 start)**
756. {
757. distance = new Dictionary<Node, **int**>();
758. previous = new Dictionary<Node, Node>();
760. **Node[] gameBoard = board.GetAllNodes();**
762. List<Node> Q = new List<Node>();
763. *// Linq to get the nodes of all other players*
764. List<Node> occupied = gamePlayers.**Where**(gamePlayers => gamePlayers.GetID() != currentPlayer.GetID()).**Select**(gamePlayers => board.GetNode(gamePlayers.position)).ToList();
766. **foreach** (Node node **in** gameBoard)
767. {
768. distance.**Add**(node, **int**.MaxValue);
769. previous.**Add**(node, **null**);
770. **Q.Add(node);**
771. }
773. Node current = board.GetNode(start);
774. distance[current] = 0;
776. **while** (Q.Count > 0)
777. {
779. *// Linq to sort the list by distance and get the first element (Priority Queue Implementation)*
780. **current = Q.OrderBy(x => distance[x]).First();**
781. Q.**Remove**(current);
782. **if** (distance[current] == **int**.MaxValue) **break**; *// All remaining nodes are inaccessible so we can safely break*
784. **foreach** (**var** g **in** current.GetNodeNeighbours())
785. **{**
786. Node neighbour = board.GetNode(g);
787. **if** (neighbour == **null**)
788. {
789. **continue**;
790. **}**
791. **else** **if** ((neighbour.status.GetOccupant() != currentPlayer.GetID() && neighbour.status.GetStatus() != "Empty") || occupied.Contains(neighbour))
792. {
793. distance[neighbour] = **int**.MaxValue;
794. previous[neighbour] = **null**;
795. **continue;**
796. }
798. Connection con = board.GetConnection(current.position, g);
800. **int NewDist = distance[current] + con.GetWalkingCost(currentPlayer);**
801. **if** (NewDist < distance[neighbour])
802. {
803. distance[neighbour] = NewDist;
804. previous[neighbour] = current;
805. **}**
806. }
807. }
809. *// Prevent the player from moving onto their current position*
810. **distance[board.GetNode(start)] = int.MaxValue;**
811. }
813. *// method to undo the last action of the player*
814. **public** **void** UndoPlayerAction()
815. **{**
816. GameAction act = actions.Pop();
817. UndoAction(act);
818. }
820. ***// method to undo the given game action***
821. **public** **void** UndoAction(GameAction a)
822. {
823. **if** (a.type == typeof(PlayerMove))
824. {
825. **PlayerMove move = (PlayerMove)a;**
827. **if** (!isAICalculation)
828. {
829. *// if the player is undoing their move, animate back down the path taken*
830. **Node otherpos = board.GetNode(move.position);**
831. Dijkstra(board, move.newpos);
832. Node current = otherpos;
833. Stack<Node> path = new Stack<Node>();
834. **while** (current != board.GetNode(move.newpos))
835. **{**
836. path.Push(current);
837. current = previous[current];
838. }
839. currentPlayer.moves += distance[otherpos];
840. **UserInterface.UpdatePlayer(path);**
841. }
843. currentPlayer.position = move.position;
845. **}**
846. **else** **if** (a.type == typeof(PlayerPurchase))
847. {
848. PlayerPurchase purchase = (PlayerPurchase)a;
850. **if (purchase.status == "Road" || purchase.status == "Wall")**
851. {
852. *// remove the connection and refund the player*
853. Connection con = board.GetConnection(purchase.position, purchase.otherpos);
855. **currentPlayer.removeConnection(con);**
856. currentPlayer.addVictoryPoints(-VICTORY\_POINT\_CONVERTOR[con.GetStatus()]);
857. board.UpdateConnection(purchase.position, purchase.otherpos, new Connection());
858. Refund(purchase.status);
860. **if (!isAICalculation)**
861. UserInterface.UpdateConnection(board.GetNode(purchase.position), board.GetNode(purchase.otherpos), board.GetConnection(purchase.position, purchase.otherpos));
862. }
863. **else** **if** (purchase.status == "Village")
864. {
865. **currentPlayer.removeBuilding(board.GetNode(currentPlayer.position));**
866. currentPlayer.addVictoryPoints(-VICTORY\_POINT\_CONVERTOR["Village"]);
867. board.GetNode(purchase.position).status = new Building();
868. Refund("Village");
870. **if (!isAICalculation)**
871. UserInterface.UpdateSettlement(board.GetNode(purchase.position));
872. }
873. **else** **if** (purchase.status == "City")
874. {
875. **board.GetNode(purchase.position).status.DowngradeVillage();**
876. currentPlayer.addVictoryPoints(-VICTORY\_POINT\_CONVERTOR["City"]);
877. Refund("City");
879. **if** (!isAICalculation)
880. **UserInterface.UpdateSettlement(board.GetNode(purchase.position));**
881. }
882. }
883. **else**
884. {
885. **UserInterface.CreatePopup("Unknown Type");**
886. }
887. }
889. *// method to perform the given game action*
890. **public void DoAction(GameAction a)**
891. {
892. **if** (a.type == typeof(PlayerMove))
893. {
894. *// move the player*
895. **PlayerMove move = (PlayerMove)a;**
896. MovePlayer(board.GetNode(move.newpos));
897. }
898. **else** **if** (a.type == typeof(PlayerPurchase))
899. {
900. ***// make the purchase***
901. PlayerPurchase purchase = (PlayerPurchase)a;
902. **if** (purchase.status == "Road")
903. {
904. purchaseRoad(board.GetNode(purchase.otherpos));
905. **}**
906. **else** **if** (purchase.status == "Wall")
907. {
908. purchaseWall(board.GetNode(purchase.otherpos));
909. }
910. **else if (purchase.status == "Village")**
911. {
912. purchaseVillage(board.GetNode(purchase.position));
913. }
914. **else** **if** (purchase.status == "City")
915. **{**
916. purchaseCity(board.GetNode(purchase.position));
917. }
918. }
919. **else**
920. **{**
921. UserInterface.CreatePopup("Unknown Type");
922. }
923. }
924. }
926. */// <summary>*
927. */// The <c>GameAction</c> class represents an action that a player can make.*
928. */// </summary>*
929. **public** **abstract** **class** GameAction
930. **{**
931. **public** **int** playerID;
932. **public** Vector3 position;
933. **public** Type type;
934. }
936. */// <summary>*
937. */// The <c>PlayerMove</c> class represents the game action of moving a player to a new node.*
938. */// </summary>*
939. **public** **class** PlayerMove : GameAction
940. **{**
941. **public** Vector3 newpos;
943. *// contructor for the PlayerMove class that takes the origin, new position and player moving*
944. **public** PlayerMove(**int** playerID, Vector3 position, Vector3 newpos)
945. **{**
946. **this**.playerID = playerID;
947. **this**.position = position;
948. **this**.newpos = newpos;
949. type = GetType();
950. **}**
952. *// override the ToString method to return the move*
953. **public** **override** **string** ToString()
954. {
955. **return $"player {playerID} moved from {position} to {newpos}";**
956. }

959. }
961. */// <summary>*
962. */// The <c>PlayerPurchase</c> class represents the game action of a player making a shop purchase.*
963. */// </summary>*
964. **public** **class** PlayerPurchase : GameAction
965. **{**
966. **public** Vector3 otherpos;
967. **public** **string** status;
969. *// constructor for the PlayerPurchase class that takes the playerID, position, status and other position (if needed for purchase)*
970. **public PlayerPurchase(int playerID, Vector3 position, string status, Vector3 otherpos = new Vector3())**
971. {
972. **this**.playerID = playerID;
973. **this**.position = position;
974. **this**.otherpos = otherpos;
975. **type = GetType();**
976. **this**.status = status;
977. }
979. *// override the ToString method to return the purchase*
980. **public override string ToString()**
981. {
982. **if** (otherpos == new Vector3())
983. {
984. **return** $"player {playerID} Purchasing a {status} at {position}";
985. **}**
986. **else**
987. {
988. **return** $"player {playerID} Purchasing a {status} from {position} to {otherpos}";
989. }
990. **}**
991. }
992. }

#### Board.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Numerics;
4. **using** System.Linq;
6. **namespace** NEAGame
7. {
8. */// <summary>*
9. */// The <c>Board</c> class is used to represent the graph of nodes and connections that make up the main game board.*
10. ***/// </summary>***
11. **public** **class** Board *// A graph of nodes and connections that makes up the main game board.*
12. {
14. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
15. ***/// nested dictionary for the connections between nodes in the board implementing an adjacency matrix***
16. */// these data structures are also composed of other class objects*
17. */// Skill A: Complex Data Structure*
18. */// Skill A: Complex User-Defined OOP: Composition + Encapsulation*
20. **private Dictionary<Vector3, Resource> board = new Dictionary<Vector3, Resource>();**
21. **private** Dictionary<Vector3, Node> nodes = new Dictionary<Vector3, Node>();
22. **private** Dictionary<Vector3, Dictionary<Vector3, Connection>> connections = new Dictionary<Vector3, Dictionary<Vector3, Connection>>();
23. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*

26. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
27. */// <summary>*
28. */// Constructor for the <c>Board</c> class. This creates a new board with 19 hexagon cells and 54 nodes using a mathematical representation of the Catan board.*
29. */// This is more efficient than creating a 2D array of hexagons and nodes and allows for easier calculations of the positions of nodes and connections.*
30. ***/// Skill A: Complex Mathematical Model***
31. */// </summary>*
32. **public** Board()
33. {
34. **int** i = 0;
35. **for (int x = -2; x < 3; x++)**
36. {
37. **for** (**int** y = -2; y < 3; y++)
38. {
39. **for** (**int** z = -2; z < 3; z++)
40. **{**
41. **if** (x + y + z == 0)
42. {
43. Resource unit;
44. **if** (x == 0 && y == 0 && z == 0)
45. **{**
46. *// center of board is Empty in classical Catan*
47. unit = new Resource(0);
48. }
49. **else**
50. **{**
51. *// generate a random resource at each position in the hexagon grid*
52. unit = Resource.GetRandom();
53. }
54. board.**Add**(new Vector3(x, y, z), unit);
55. **i++;**
56. }
57. }
58. }
59. }
61. i = 0;
62. **for** (**int** x = -2; x < 4; x++)
63. {
64. **for** (**int** y = -2; y < 4; y++)
65. **{**
66. **for** (**int** z = -2; z < 4; z++)
67. {
68. **if** ((x + y + z == 1) || (x + y + z == 2))
69. {
70. **Node n = new Node(x, y, z);**
71. nodes.**Add**(new Vector3(x, y, z), n);
72. i++;
73. *// create a node at each position on the board using cubic coordinates*
74. }
75. **}**
76. }
77. }
78. }
79. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
81. *// constructor to create a board from the loaded serialized data*
82. **public** Board (BoardWrapper board) : **this**()
83. {
84. **int** i = 0;
85. **foreach (ResourceWrapper res in board.board.\_Values)**
86. {
87. Resource n = new Resource(res);
88. **this**.board[new Vector3(board.board.\_Keys[i].x, board.board.\_Keys[i].y, board.board.\_Keys[i].z)] = n;
89. i++;
90. **}**
92. i = 0;
93. **foreach** (NodeWrapper node **in** board.nodes.\_Values)
94. {
95. **Node n = new Node(node);**
96. nodes[new Vector3(board.nodes.\_Keys[i].x, board.nodes.\_Keys[i].y, board.nodes.\_Keys[i].z)] = n;
97. i++;
98. }
99. }
101. *// method to get the resource at a given position or return null if it doesn't exist*
102. **public** Resource GetHexAtPosition(Vector3 pos)
103. {
104. **if** (board.ContainsKey(pos))
105. **return board[pos];**
106. **else**
107. **return** **null**;
108. }
110. ***// method to get the node at a given position or return null if it doesn't exist***
111. **public** Node GetNode(Vector3 pos)
112. {
114. **if** (nodes.ContainsKey(pos))
115. **return nodes[pos];**
116. **else**
117. **return** **null**;
118. }
120. ***// method to get an array of all the nodes on the board***
121. **public** Node[] GetAllNodes()
122. {
123. **return** nodes.Values.ToArray();
124. }
126. *// method to get key value pairs of all the resources on the board with their positions*
127. **public** Dictionary<Vector3, Resource> GetResourcesOnBoard()
128. {
129. **return** board;
130. **}**
132. *// method to get the connection between two nodes*
133. **public** Connection GetConnection(Vector3 v1, Vector3 v2)
134. {
135. **try**
136. {
137. **return** connections[v1][v2];
138. }
139. **catch** (KeyNotFoundException)
140. **{**
141. **return** new Connection();
142. }
143. }
145. ***// method to update an existing connection between two nodes or create a new one if it doesn't exist***
146. **public** **void** UpdateConnection(Vector3 v1, Vector3 v2, Connection con)
147. {
149. **if** (connections.ContainsKey(v1))
150. **{**
151. **if** (connections[v1].ContainsKey(v2))
152. {
153. connections[v1][v2] = con;
154. }
155. **else**
156. {
157. connections[v1].**Add**(v2, con);
158. }
159. }
160. **else**
161. {
162. connections.**Add**(v1, new Dictionary<Vector3, Connection>());
163. connections[v1].**Add**(v2, con);
164. }
165. **if (connections.ContainsKey(v2))**
166. {
167. **if** (connections[v2].ContainsKey(v1))
168. {
169. connections[v2][v1] = con;
170. **}**
171. **else**
172. {
173. connections[v2].**Add**(v1, con);
174. }
175. **}**
176. **else**
177. {
178. connections.**Add**(v2, new Dictionary<Vector3, Connection>());
179. connections[v2].**Add**(v1, con);
180. **}**
181. }
183. *// method to serialize the board object directly to a JSON serializable BoardWrapper object*
184. **public** BoardWrapper SoftSerialize()
185. **{**
186. BoardWrapper b = new BoardWrapper();
187. b.board = new HexagonUnitWrapper(board);
188. b.connections = new AdjMatrixWrapper(connections);
189. b.nodes = new AllNodesWrapper(nodes);
191. **return** b;
192. }
194. }
196. */// <summary>*
197. */// A player may purchase a <c>Settlement</c> to place on the board. This is an abstract class which is inherited by <c>Building</c> and <c>Connnection</c>*
198. */// </summary>*
199. **public** **abstract** **class** Settlement
200. **{**
201. **protected** **int** id { **get**; **set**; }
202. **protected** **string**[] statuses { **get**; **set**; }
203. **protected** **int** occupantID { **get**; **set**; }
204. }
206. *///////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
207. */// <summary>*
208. */// A <c>Building</c> represents the status of a node on the board. It can be empty, a village or a city.<br/>*
209. */// Skill A: Complex User-Defined OOP - Inheritance*
210. ***/// </summary>***
211. *///////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
212. **public** **class** Building : Settlement
213. {
214. *// constructor for the building class*
215. **public Building(string i = "Empty", int o = -1)**
216. {
217. statuses = new **string**[] { "Empty", "Village", "City" };
218. id = Array.IndexOf(statuses, i);
219. occupantID = o;
221. }
223. *// constructor to create a building from the loaded serialized data*
224. **public** Building(SettlementWrapper sw)
225. **{**
226. statuses = new **string**[] { "Empty", "Village", "City" };
227. occupantID = sw.occupantID;
228. id = Array.IndexOf(statuses, sw.status);
229. }
231. *// method to upgrade a village to a city*
232. **public** **void** UpgradeVillage()
233. {
234. **if** (id == 1)
235. **{**
236. id++;
237. }
238. }
240. ***// method to check if a node is empty***
241. **public** **bool** IsEmpty()
242. {
243. **return** id == 0;
244. }
246. *// override the ToString method to return the status*
247. **public** **override** **string** ToString()
248. {
249. **if** (occupantID != -1)
250. **{**
251. **return** $"{statuses[id]} owned by Player {occupantID}";
252. }
253. **else**
254. {
255. **return $"{statuses[id]}";**
256. }
257. }
259. *// method to get the status of the building*
260. **public string GetStatus()**
261. {
262. **return** statuses[id];
263. }
265. ***// method to get the occupant of the building***
266. **public** **int** GetOccupant()
267. {
268. **return** occupantID;
269. }
271. *// method to undo the upgrade of a village to a city*
272. **public** **void** DowngradeVillage()
273. {
274. **if** (id == 2)
275. **{**
276. id--;
277. }
278. }
279. }
281. */// <summary>*
282. */// A <c>Resource</c> represents the type of resource that a hexagon cell on the board produces.*
283. */// </summary>*
284. **public** **class** Resource
285. **{**
286. **public** **static** **readonly** **string**[] RESOURCES = { "Empty", "Brick", "Sheep", "Ore", "Wood", "Wheat" };
287. **private** **int** id;
288. **private** **static** **readonly** Random rng = new Random();

291. *// constructor for the resource class using the resource id*
292. **public** Resource(**int** i = 0)
293. {
294. id = i;
295. **}**
297. *// constructor to create a resource from the loaded serialized data*
298. **public** Resource(ResourceWrapper res)
299. {
300. **id = res.id;**
301. }
303. *// override the ToString method to return the name of the resource*
304. **public** **override** **string** ToString()
305. **{**
306. **return** RESOURCES[id];
307. }
309. *// method to get the id of the resource*
310. **public override int GetHashCode()**
311. {
312. **return** id;
313. }
315. ***// method to create a random resource***
316. **public** **static** Resource GetRandom()
317. {
318. Resource temp = new Resource(rng.Next(1, RESOURCES.Length));
319. **return** temp;
320. **}**
322. *// override the Equals method to compare two resources*
323. **public** **override** **bool** Equals(**object** obj)
324. {
325. ***// used in comparisons to check if two resources are the same in dictionary lookups***
326. **if** (obj == **null** || GetType() != obj.GetType())
327. {
328. **return** **false**;
329. }
330. **Resource r = (Resource)obj;**
331. **return** r.id == id;
332. }
333. }
334. }

#### Player.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Numerics;
4. **using** System.Linq;
6. **namespace** NEAGame
7. {
8. */// <summary>*
9. */// A <c>Player</c> object represents a single player in the game.*
10. ***/// </summary>***
11. **public** **class** Player
12. {
13. *// Unity static color options*
14. [System.Serializable]
15. **public enum PlayerColors**
16. {
17. blue,
18. cyan,
19. green,
20. **grey,**
21. magenta,
22. red,
23. white,
24. yellow
25. **}**
27. **private** **int** victoryPoints;
28. **private** Dictionary<Resource, **int**> resources = new Dictionary<Resource, **int**>() {
29. { new Resource(1), 0 },
30. **{ new Resource(2), 0 },**
31. { new Resource(3), 0 },
32. { new Resource(4), 0 },
33. { new Resource(5), 0 }
34. };
36. **private** List<Node> buildings = new List<Node>();
37. **private** List<Connection> connections = new List<Connection>();
39. **protected** **int** playerNumber; *// used as a UID for the player allowing the same name*
40. **public readonly string playerName;**
41. **public** **readonly** Vector3 origin;
42. **public** **int** moves;
43. **public** Vector3 position;
44. **public** **string** color;
45. **protected bool isAI = false; *// gets changed by child AI class***
47. *// constructor for player*
48. **public** Player(**int** playerNumber, **string** playerName, Vector3 origin, **string** color)
49. {
50. **this.playerNumber = playerNumber;**
51. **this**.playerName = playerName;
52. **this**.origin = origin;
53. **this**.color = color;
54. victoryPoints = 0;
55. **position = origin;**
56. }
58. *// constructor for player that loads from a save file*
59. **public** Player(PlayerWrapper player)
60. **{**
61. playerNumber = player.playerNumber;
62. playerName = player.playerName;
63. origin = new Vector3(player.origin.x, player.origin.y, player.origin.z);
64. moves = player.moves;
65. **color = player.color;**
66. position = new Vector3(player.position.x, player.position.y, player.position.z);
67. resources = new Dictionary<Resource, **int**>();
68. **foreach** (**var** entry **in** player.resources.\_Keys.Zip(player.resources.\_Values, (k, v) => new { k, v }))
69. {
70. **resources.Add(new Resource(entry.k), entry.v);**
71. }
72. victoryPoints = player.victoryPoints;
73. }

76. *// checks if the player is an AI*
77. **public** **bool** isPlayerAI()
78. {
79. **return** isAI;
80. **}**
82. *// adds victory points to the player*
83. **public** **void** addVictoryPoints(**int** points)
84. {
85. **victoryPoints += points;**
86. }
88. *// gets the player's ID*
89. **public** **int** GetID()
90. **{**
91. **return** playerNumber;
92. }
94. *// adds a building to the player's list of buildings*
95. **public void addBuilding(Node building)**
96. {
97. buildings.**Add**(building);
98. }
100. ***// removes a building from the player's list of buildings***
101. **public** **void** removeBuilding(Node building)
102. {
103. **if** (buildings.Contains(building))
104. {
105. **buildings.Remove(building);**
106. }
107. **else**
108. {
109. **throw** new Exception("Player does not own this building");
110. **}**
111. }
113. *// gets the player's list of buildings*
114. **public** List<Node> GetBuildings()
115. **{**
116. **return** buildings;
117. }
119. *// gets the player's original position*
120. **public Vector3 GetCapital()**
121. {
122. **return** origin;
123. }
125. ***// adds a connection to the player's list of connections***
126. **public** **void** addConnection(Connection connection)
127. {
128. connections.**Add**(connection);
129. }
131. *// removes a connection from the player's list of connections*
132. **public** **void** removeConnection(Connection con)
133. {
134. **if** (connections.Contains(con))
135. **{**
136. connections.**Remove**(con);
137. }
138. **else**
139. {
140. **throw new Exception("Player does not own this connection");**
141. }
142. }
144. *// gets the player's list of connections*
145. **public int getVictoryPoints()**
146. {
147. **return** victoryPoints;
148. }
150. ***// get the number of moves the player has left***
151. **public** **int** getMovesLeft()
152. {
153. **return** moves;
154. }
156. *// adds a resource to the player's list of resources*
157. **public** **void** addResource(Resource resource, **int** amount = 1)
158. {
159. resources[resource] += amount;
160. **}**
162. *// removes a resource from the player's list of resources*
163. **public** **void** removeResource(Resource resource, **int** amount = 1)
164. {
165. **resources[resource] -= amount;**
166. }
168. *// gets the player's inventory*
169. **public** Dictionary<Resource, **int**> getResources()
170. **{**
171. **return** resources;
172. }
174. *// overrides the ToString method to return the player's name*
175. **public override string ToString()**
176. {
177. **return** playerName;
178. }
180. ***// Used to calculate the wealth of a player for the AI static state evaluation function***
181. **public** **int** GetWealth()
182. {
183. **int** wealth = 0;
184. **foreach** (KeyValuePair<Resource, **int**> resource **in** resources)
185. **{**
186. wealth += resource.**Value**;
187. }
188. *// victory points are weighted as they indicate resource have been spent*
189. wealth += victoryPoints \* 5;
191. *// int cast wealth*
192. **return** wealth;
193. }
194. }
195. **}**

#### AI.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Numerics;
4. **using** System.Linq;
6. **namespace** NEAGame
7. {
8. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
9. */// <summary>*
10. ***/// The <c>AI</c> class inherits from Player and is used to represent an AI player in the game. This contains all of the AI's logic and decision making. <br/>***
11. */// Skill A: Complex User-Defined OOP - Inheritance*
12. */// </summary>*
13. */////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
14. **class** AI : Player
15. **{**
16. **public** **enum** Turn
17. {
18. Max,
19. Min
20. **}**
21. **public** Stack<GameAction> selectedMoves = new Stack<GameAction>();
22. **public** Stack<GameAction> currentMove = new Stack<GameAction>();
23. **private** **readonly** **int** MAX\_DEPTH = 4;
24. **private** TravelersOfCatan gameRef;
26. *// Constructor for AI that creates a new player*
27. **public** AI(**int** playerID, **string** name, **string** playerColor, Vector3 home, TravelersOfCatan reference) : **base**(playerNumber: playerID, playerName:name, color: playerColor, origin:home)
28. {
29. isAI = **true**;
30. **gameRef = reference;**
31. }
33. *// Constructor for AI that loads a player from a save file*
34. **public** AI(PlayerWrapper player, TravelersOfCatan reference) : **base**(player)
35. **{**
36. isAI = **true**;
37. gameRef = reference;
38. }
40. ***// Method to get the static evaluation of the current game position based on player's wealth and victory points***
41. **public** **int** StaticEval()
42. {
43. **int** score = 0;
45. **foreach (Player pdl in gameRef.gamePlayers)**
46. {
47. **if** (pdl.GetID() == playerNumber)
48. {
49. **if** (pdl.getVictoryPoints() > gameRef.WinningVictoryPoints)
50. **{**
51. score += 1000;
52. }
53. score += pdl.GetWealth();
54. }
55. **else**
56. {
57. **if** (pdl.getVictoryPoints() > gameRef.WinningVictoryPoints)
58. {
59. score -= 1000;
60. **}**
61. score -= pdl.GetWealth();
62. }
63. }
64. **return** score;
65. **}**
67. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
68. */// <summary>*
69. */// Runs the Best Reply Search algorithm on the current game position to attempt to determine the best move to make.*
70. ***/// <br/>Skill A: Recursive Algorithm***
71. */// <br/>Skill A: Tree traversal*
72. */// <br/>Skill A: Complex User-Defined Algorithm*
73. */// <br/>Skill A: Stack Operations*
74. */// </summary>*
75. ***//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////***
76. **public** **int** BRS(**int** alpha=**int**.MinValue + 1, **int** beta=**int**.MaxValue, **int** depth=-1, Turn turn=Turn.Max)
77. {
78. **if** (depth == -1)
79. {
80. **depth = MAX\_DEPTH;**
81. }
83. List<List<GameAction>> AllMoves = new List<List<GameAction>>();
85. **if (depth == 0)**
86. {
87. *// this line was altered from the original code to fix a bug where the AI returns the negative of the static evaluation on all even depths*
88. **return** (**int**)Math.Pow(-1, MAX\_DEPTH) \* StaticEval();
89. }
91. **if** (turn == Turn.Max)
92. {
93. *// if it is the AI's turn, generate all possible moves the AI has*
94. AllMoves = GenerateMoves(**this**).ToList();
95. **turn = Turn.Min;**
96. }
97. **else** **if** (turn == Turn.Min)
98. {
99. *// if it is the player's turn, generate all possible moves that any player has*
100. **foreach (Player pdl in gameRef.gamePlayers)**
101. {
102. **if** (pdl.GetID() == playerNumber)
103. {
104. **continue**;
105. **}**
106. AllMoves.AddRange(GenerateMoves(pdl));
107. }
108. turn = Turn.Max;
109. }
111. *// sort the moves by their length to optimise alpha beta pruning*
112. *// this means that the moves that are most likely to be good are evaluated first*
113. AllMoves.Sort((x, y) => y.Count.CompareTo(x.Count));
115. **foreach (var m in AllMoves)**
116. {
118. gameRef.UpdateCurrentPlayer(m[0].playerID);
119. gameRef.gatherResources(m[0].playerID);
120. **gameRef.actions.Clear();**
121. **for** (**int** i = 0; i < m.Count; i++)
122. {
123. currentMove.Push(m[i]);
124. gameRef.DoAction(m[i]);
125. **}**
127. **int** v = -BRS(-beta, -alpha, depth-1, turn);
128. *// recursively calls the BRS algorithm to move to the next depth*

131. gameRef.UpdateCurrentPlayer(m[0].playerID);
132. **for** (**int** i = m.Count - 1; i > -1; i--)
133. {
135. **gameRef.UndoAction(m[i]);**
136. currentMove.Pop();
137. }
138. *// undoes the action to return to the previous game state*
139. gameRef.undoGatherResources(m[0].playerID);
141. **if** (v >= beta)
142. {
143. **return** v;
144. }
145. **if (v > alpha)**
146. {
147. **if** (depth == MAX\_DEPTH)
148. {
149. *// if the move is the best move found so far, store it in the selectedMoves stack*
150. **selectedMoves = new Stack<GameAction>(m);**
151. }
152. alpha = v;
153. }
154. }
155. **return alpha; *// returns the evaluation of the position***
156. }
157. *///////////////////////////////////////////////////////////////////////////////////////////*
158. */// <summary>*
159. */// Method to generate possible good moves on the passed player's turn*
160. ***/// <br/>Skill A: Dynamic generation of class objects***
161. */// <br/>Skill A: Stack Operations*
162. */// </summary>*
163. *///////////////////////////////////////////////////////////////////////////////////////////*
164. **public** IEnumerable<List<GameAction>> GenerateMoves(Player pdl)
165. **{**
166. *// get all combinations of making a purchase followed by moving to a new node*
167. **int** playerID = pdl.GetID();
168. gameRef.UpdateCurrentPlayer(playerID);
169. gameRef.gatherResources(playerID);
170. **List <Node> allMoves = gameRef.attemptPlayerMove();**
171. allMoves.**Add**(gameRef.board.GetNode(pdl.position));
172. **if** (gameRef.tryPurchaseCity() != **null**)
173. {
174. **foreach** (Node end **in** allMoves)
175. **{**
176. **yield** **return** new List<GameAction> {
177. new PlayerPurchase(playerID, pdl.position, "City"),
178. new PlayerMove(playerID, pdl.position, end.position)
179. };
180. **}**
181. }
183. **foreach** (Node n **in** gameRef.tryPurchaseRoad())
184. {
185. **foreach (Node end in allMoves)**
186. {
187. **yield** **return** new List<GameAction> {
188. new PlayerPurchase(playerID, pdl.position, "Road", n.position),
189. new PlayerMove(playerID, pdl.position, end.position)
190. **};**
191. }
192. }
194. **if** (gameRef.tryPurchaseVillage() != **null**)
195. **{**
196. **foreach** (Node end **in** allMoves)
197. {
198. **yield** **return** new List<GameAction> {
199. new PlayerPurchase(playerID, pdl.position, "Village"),
200. **new PlayerMove(playerID, pdl.position, end.position)**
201. };
202. }
203. }
205. **foreach (Node n in gameRef.tryPurchaseWall())**
206. {
207. **foreach** (Node end **in** allMoves)
208. {
209. **yield** **return** new List<GameAction> {
210. **new PlayerPurchase(playerID, pdl.position, "Wall", n.position),**
211. new PlayerMove(playerID, pdl.position, end.position)
212. };
213. }
214. }
216. **foreach** (Node end **in** allMoves)
217. {
218. **yield** **return** new List<GameAction> {
219. new PlayerMove(playerID, pdl.position, end.position)
220. **};**
221. }
222. gameRef.undoGatherResources(playerID);
223. }
224. }
225. **}**

#### Connection.cs

1. **using** System;
3. **namespace** NEAGame
4. {
5. ***///////////////////////////////////////////////////////////////////////////////////////////////////////////////////***
6. */// <summary>*
7. */// A <c>Connection</c> represents the status of a connection between two nodes on the game board.<br/>*
8. */// Skill A: Complex User-Defined OOP - Inheritance*
9. */// </summary>*
10. ***///////////////////////////////////////////////////////////////////////////////////////////////////////////////////***
11. **public** **class** Connection : Settlement
12. {
13. *// Constructor to create a connection from an id*
14. **public** Connection( **int** i = 0, **string** status = "", **int** occupant = -1)
15. **{**
16. statuses = new **string**[] { "Empty", "Road", "Wall" };
17. id = i;
18. **if** (status != "")
19. {
20. **id = Array.IndexOf(statuses, status);**
21. }
22. occupantID = occupant;
23. }
25. ***// Constructor to create a connection from a settlement wrapper***
26. **public** Connection(SettlementWrapper settlementWrapper)
27. {
28. statuses = new **string**[] { "Empty", "Road", "Wall" };
30. **occupantID = settlementWrapper.occupantID;**
31. SetStatus(settlementWrapper.status);
32. }
34. *// Method to get the occupant of the connection*
35. **public int GetOccupant()**
36. {
37. **return** occupantID;
38. }
40. ***// Method to set the occupant of the connection***
41. **public** **void** SetOccupant(Player p)
42. {
43. occupantID = p.GetID();
44. }
46. *// Method to get the status of the connection*
47. **public** **string** GetStatus()
48. {
49. **return** statuses[id];
50. **}**
52. *// Method to set the status of the connection*
53. **public** **void** SetStatus(**string** status)
54. {
55. **id = Array.IndexOf(statuses, status);**
56. }
58. *// Method to get the cost of walking on this connection*
59. **public** **int** GetWalkingCost(Player otherPlayer)
60. **{**
61. **if** (id == 0)
62. {
63. **return** 1;
64. }
65. **if (otherPlayer.GetID() == occupantID)**
66. {
67. **if** (id == 1)
68. {
69. **return** 0;
70. **}**
71. **else**
72. {
73. **return** 2;
74. }
75. **}**
76. **else** **if** (otherPlayer.GetID() != occupantID)
77. {
78. **return** **int**.MaxValue;
79. }
80. **throw new Exception("Invalid Player Number");**
81. }
83. *// override the ToString method to return a string representation of the connection*
84. **public** **override** **string** ToString()
85. **{**
86. **return** $"{statuses[id]} Owned by {occupantID}";
87. }
88. }
89. }

#### Node.cs

1. **using** System.Collections.Generic;
2. **using** System.Numerics;
4. **namespace** NEAGame
5. **{**
6. */// <summary>*
7. */// A <c>Node</c> represents a single point on the graph where a player or building can be placed.*
8. */// </summary>*
9. **public** **class** Node
10. **{**
11. **public** Vector3 position;
12. **public** Building status = new Building();
14. *// constructor for creating a node from a position*
15. **public Node(int x, int y, int z)**
16. {
17. position = new Vector3(x, y, z);
18. }
20. ***// constructor for creating a node from a node wrapper***
21. **public** Node(NodeWrapper node)
22. {
23. position = new Vector3(node.position.x, node.position.y, node.position.z);
24. status = new Building(node.status);
25. **}**
27. *// method to get the node neighbours of a node based on the parity of the position*
28. **public** IEnumerable<Vector3> GetNodeNeighbours()
29. {
30. ***// determine parity of position***
31. **int** sum = (**int**)(position.X + position.Y + position.Z);
32. **if** (sum % 2 == 1)
33. {
34. **yield** **return** position + new Vector3(1, 0, 0);
35. **yield return position + new Vector3(0, 1, 0);**
36. **yield** **return** position + new Vector3(0, 0, 1);
37. }
38. **else**
39. {
40. **yield return position + new Vector3(-1, 0, 0);**
41. **yield** **return** position + new Vector3(0, -1, 0);
42. **yield** **return** position + new Vector3(0, 0, -1);
43. }
44. }
46. *// method to get the hex neighbours of a node based on the parity of the position*
47. **public** IEnumerable<Vector3> GetHexNeighbours()
48. {
49. **int** sum = (**int**)(position.X + position.Y + position.Z);
50. **if (sum % 2 == 1)**
51. {
52. **yield** **return** position + new Vector3(-1, 0, 0);
53. **yield** **return** position + new Vector3(0, -1, 0);
54. **yield** **return** position + new Vector3(0, 0, -1);
55. **}**
56. **else**
57. {
58. **yield** **return** position + new Vector3(-1, -1, 0);
59. **yield** **return** position + new Vector3(0, -1, -1);
60. **yield return position + new Vector3(-1, 0, -1);**
61. }
62. }
64. *// method to check if the node is empty*
65. **public bool isEmpty()**
66. {
67. **return** status.ToString() == "Empty";
68. }
70. ***// override the ToString method to return the status of the node***
71. **public** **override** **string** ToString()
72. {
73. **if** (!status.IsEmpty())
74. {
75. **return $"{status} at {position}";**
77. }
78. **else**
79. {
80. **return position.ToString();**
81. }
82. }
83. }
84. }

#### UnityUI.cs

1. **using** System.Collections;
2. **using** UnityEngine;
3. **using** UnityEngine.UI;
4. **using** UnityEngine.SceneManagement;
5. **using NEAGame;**
6. **using** System.Linq;
8. *// First half of the UnityUI partial class which controls the overall flow of the game*
10. ***/// <summary>***
11. */// The <c>UnityUI</c> class is used to manage the Unity interface for the game. This class is used to manage the flow of the game and the user interface.*
12. */// It is a singleton class that creates and updates the game, and interfaces with all of the other UI classes.*
13. */// </summary>*
14. **public** **partial** **class** UnityUI : MonoBehaviour, UI
15. **{**
17. */// <summary>*
18. */// Singleton instance of the UnityUI class so that it can be accessed from other UI scripts*
19. */// </summary>*
20. **public static UnityUI Interface { get; private set; }**
22. [Header("Serialized Game View")]
23. **public** TravelersOfCatan game;
25. **private int selectedSave = -1;**
26. **private** **bool** isLoading = **false**;
28. *// Start is called before the first frame update*
29. **void** Awake()
30. **{**
31. **if** (**Interface** == **null**)
32. **Interface** = **this**;
34. DontDestroyOnLoad(gameObject);
35. **game = null;**
36. }
38. *// OnEnable and OnDisable are used to subscribe and unsubscribe from the sceneLoaded event*
39. **void** OnEnable()
40. **{**
41. SceneManager.sceneLoaded += NewScene;
42. }
43. **void** OnDisable()
44. {
45. **SceneManager.sceneLoaded -= NewScene;**
46. }
48. *// Method to prepare for a new scene being loaded*
49. **void** NewScene(Scene scene, LoadSceneMode mode)
50. **{**
52. **if** (scene.name == "Game")
53. {
54. SetupGameScene();
55. **if (isLoading)**
56. {
57. StartCoroutine(LoadGame());
58. }
59. **else**
60. **{**
61. game.startGame();
62. }
64. }
65. **else if (scene.name == "Hub")**
66. {
67. selectedSave = -1; *// deselect the save slot when returning to the hub*
68. }
69. **else** **if** (scene.name == "About+Rules")
70. **{**
71. GameObject.FindGameObjectWithTag("HomeButton").GetComponent<Button>().onClick.AddListener(() => {
72. AudioManager.i.Play("UIClick");
73. AudioManager.i.Stop("Write");
74. LeanTween.scale(GameObject.FindGameObjectWithTag("HomeButton"), new Vector3(1.2f, 1.2f, 1.2f), 0.2f).setEase(LeanTweenType.easeInOutSine).setLoopPingPong(1);
75. **GoHome();**
76. }); *// add a listener to the home button to go back to the hub from the about+rules page*
77. }
78. **else** **if** (scene.name == "Victory")
79. {
80. **AudioManager.i.Stop("Write");**
81. GameObject.FindGameObjectWithTag("HomeButton").GetComponent<Button>().onClick.AddListener(() => {
82. AudioManager.i.StopAll();
83. AudioManager.i.Play("UIClick");
84. LeanTween.scale(GameObject.FindGameObjectWithTag("HomeButton"), new Vector3(1.2f, 1.2f, 1.2f), 0.2f).setEase(LeanTweenType.easeInOutSine).setLoopPingPong(1);
85. **GoHome();**
86. }); *// add a listener to the home button to go back to the hub from the victory page*
88. **string**[] namesInVictoryOrder = game.gamePlayers.OrderByDescending(p => p.getVictoryPoints()).**Select**(p => p.playerName).ToArray();
89. FindObjectOfType<VictoryManager>().Setup(namesInVictoryOrder);
90. ***// order the players by victory points and pass the names to the victory manager***
91. saveCurrentGame();
92. }
93. }
95. ***// Used in testing and debugging to assert that a condition is true***
96. **void** UI.Assert(**bool** test)
97. {
98. **if** (!test)
99. {
100. **Debug.LogError("Assertion failed");**
101. Debug.**Break**();
102. }
103. }
105. ***// Method to go to the game scene***
106. **public** **void** CommenceGame()
107. {
108. SceneTransition.i.SendToScene("Game");
109. }
111. *// Called when the settings button is clicked on the home screen*
112. **public** **void** HomeScreenSettingsButton()
113. {
114. GamePauseOverlay overlay = Instantiate(PauseSettings).GetComponent<GamePauseOverlay>();
115. **overlay.RemoveSaveButton();**
116. }
118. *// Called when the start game button is clicked on the home screen*
119. **public** **void** LoadGameButton()
120. **{**
121. game = **null**;
122. StartCoroutine(DisplayLoadStates());
123. }
125. ***// Method to display the save states***
126. IEnumerator DisplayLoadStates()
127. {
128. SceneTransition.i.PlayAnimation();
129. **yield** **return** new WaitForSeconds(0.75f);
130. **SaveSelector overlay = Instantiate(GameSavePopup).GetComponent<SaveSelector>();**
131. overlay.Setup();
132. }
134. *// Method to select the game to load from save slot i*
135. **public void SelectGameToLoad(int i)**
136. {
137. isLoading = **true**;
138. selectedSave = i;
139. CommenceGame();
140. **}**
142. *// Method to create a new game in save slot i*
143. **public** **void** CreateNewGame(**int** i)
144. {
145. **isLoading = false;**
146. selectedSave = i;
147. SceneTransition.i.SendToScene("GameSetup");
148. }
150. ***// Method to load the game from the selected save slot***
151. IEnumerator LoadGame()
152. {
153. **try**
154. {
155. **JSON\_manager json = new JSON\_manager(selectedSave);**
156. GameWrapper gw = json.LoadGame();
157. game = new TravelersOfCatan(**Interface**, gw);
158. }
159. **catch** (System.Exception e) *// handle any potential exceptions in serialization or null references when loading the game*
160. **{**
161. Debug.Log(e); *// save has been corrupted*
162. LoadGameButton();
163. CreatePopup("Save has been corrupted");
164. }
165. **yield return null;**
166. }
168. *// Save the current game*
169. **void** saveCurrentGame()
170. **{**
171. **int** i = selectedSave;
172. JSON\_manager saver = new JSON\_manager(i);
173. saver.SaveGame(game);
174. }
176. *// Called by the pause menu to save the game and return to the menu page*
177. **public** **void** SaveAndExit()
178. {
179. saveCurrentGame();
180. **GoHome();**
181. }
183. *// Method to go to the menu page*
184. **public** **void** GoHome()
185. **{**
186. SceneTransition.i.SendToScene("Hub");
187. }
189. *// onclick method for the about button*
190. **public void AboutButton()**
191. {
192. SceneTransition.i.SendToScene("About+Rules");
193. }
195. ***// onclick method for the quit button***
196. **public** **void** QuitButton()
197. {
198. Application.Quit();
199. }
201. *// create a popup with the given message*
202. **public** **void** CreatePopup(**string** message)
203. {
204. Debug.Log(message); *// put message in unity console for debugging*
205. **PopupController overlay = Instantiate(AlertPopup).GetComponent<PopupController>();**
206. overlay.Setup(message);
207. }
209. *// interface method to create a popup*
210. **void UI.CreatePopup(string message)**
211. {
212. CreatePopup(message);
213. }
215. ***// Called when the program is closed by the user to attempt to save the current game.***
216. **public** **void** OnApplicationQuit()
217. {
218. **if** (game != **null** && selectedSave != -1) *// check if the user is in a game.*
219. {
220. **saveCurrentGame();**
221. }
222. }
223. }

#### UnityUI\_Game.cs

1. **using** System;
2. **using** System.Linq;
3. **using** System.Collections;
4. **using** System.Collections.Generic;
5. **using System.Threading;**
6. **using** UnityEngine;
7. **using** UnityEngine.Tilemaps;
8. **using** NEAGame;
10. ***// Second half of the UnityUI partial class, containing the game-specific methods and variables.***
12. [Serializable] *// This is serializable so it can be viewed in the Unity inspector*
13. **public** **partial** **class** UnityUI
14. {
15. **[Header("UI overlay prefabs and constants")]**
16. **public** GameObject NodePrefab;
17. **public** GameObject ConnectionPrefab;
18. **public** GameObject PlayerPrefab;
19. **public** GameObject PlayerUI;
20. **public GameObject inventoryPopup;**
21. **public** GameObject shoppingPopup;
22. **public** GameObject TradePopup;
23. **public** GameObject PlayerChoicePopup;
24. **public** GameObject AlertPopup;
25. **public GameObject PauseSettings;**
26. **public** GameObject CardObj;
27. **public** GameObject GameSavePopup;
28. **public** Tile[] resources = new Tile[6];
30. **public GridLayout gridLayout;**
31. **public** Tilemap tilemap;
32. **public** List<NodeButton> nodes;
34. **public** **float** Timer = 0.0f;
35. **public bool TimerActive = false;**

38. **private** PlayerUIOverlay overlay;
39. **private** NodeButton SelectedNode;
40. **private IEnumerator coroutine;**
42. *// Update is called once per frame*
43. **void** Update()
44. {
45. **if (TimerActive)**
46. {
47. Timer = Mathf.Clamp(Timer - Time.deltaTime, 0, **int**.MaxValue);
48. } *// updates the in game timer if it is active*
49. }
51. *// interface method to begin the new user's turn*
52. **void** UI.BeginTurn(**float** time)
53. {
54. AudioManager.i.Play("Success");
55. **Timer = time;**
56. TimerActive = **true**;
57. *// setup the player's main overlay*
58. overlay = Instantiate(PlayerUI).GetComponent<PlayerUIOverlay>();
59. **if** (game.GetCurrentPlayer().isPlayerAI())
60. **{**
61. overlay.SetAI();
62. StartCoroutine(BeginAI());
64. }
65. **else**
66. {
67. overlay.ShopInput.onClick.AddListener(OpenShop);
68. overlay.UndoInput.onClick.AddListener(game.UndoPlayerAction);
69. StartCoroutine(WaitForTurnToEnd());
70. **}**
71. overlay.PauseInput.onClick.AddListener(PauseButton);
72. overlay.ColorMe.color = textToColor(game.GetCurrentPlayer().color);
73. overlay.PlayerScore.text = game.GetCurrentPlayer().getVictoryPoints().ToString();
74. overlay.PlayerName.text = game.GetCurrentPlayer().playerName;
75. **GetPlayerGameObject(game.GetCurrentPlayer().GetID()).isCurrentPlayer = true;**
77. *// set camera position to player position on board!*
78. LeanTween.cancel(Camera.main.gameObject);
79. LeanTween.move(Camera.main.gameObject, GetPlayerGameObject(game.GetCurrentPlayer().GetID()).transform.position + new Vector3(0f, 0f, -10f), 0.3f).setEase(LeanTweenType.easeInSine);
80. **}**
82. *// method to begin the AI's turn*
83. IEnumerator BeginAI()
84. {
85. ***// call the AI BRS in a thread and wait for it to finish***
86. Thread t = new Thread(() => ((AI)game.GetCurrentPlayer()).BRS());
87. t.Start();
88. **while** (t.IsAlive)
89. {
90. **if (Timer <= 5f)**
91. {
92. **break**; *// if the timer is less than 5 seconds, break out of the loop and end the search. The best recorded move found so far will instead be used.*
93. }
94. **yield** **return** **null**;
95. **}**
96. FindAnyObjectByType<GamePauseOverlay>()?.CloseGUI(); *// fixes a sepcific bug where the player can get the AI two turns by exiting while the ai is making its moves*
97. t.Abort();
98. game.DisplayAIMoves();
99. **yield** **return** new WaitForSeconds(2f);
100. **EndTurn();**
101. }
103. *// method to setup the game scene*
104. **void** SetupGameScene()
105. **{**
106. gridLayout = FindObjectOfType<GridLayout>();
107. tilemap = FindObjectOfType<Tilemap>();
108. nodes = new List<NodeButton>();
109. }
111. *// async method to count down until the player is out of time for their turn*
112. IEnumerator WaitForTurnToEnd()
113. {
114. **bool** startedCD = **false**;
115. **while (Timer > 0.0f)**
116. {
117. **yield** **return** **null**;
118. **if** (Timer < 5f && !startedCD)
119. {
120. **AudioManager.i.Play("Countdown");**
121. startedCD = **true**;
122. }
123. }
124. EndTurn();
125. **}**
127. *// onclick method to pause the game*
128. **public** **void** PauseButton()
129. {
130. **TimerActive = false;**
131. StartCoroutine(GoToPause());
132. }
134. *// method to load the pause overlay*
135. **IEnumerator GoToPause()**
136. {
137. SceneTransition.i.PlayAnimation();
138. **yield** **return** new WaitForSeconds(0.5f);
139. Instantiate(PauseSettings);
140. **}**
142. *// function called when the player wishes to move*
143. **public** **void** OnPlayerMove()
144. {
145. **StopAllPlayerCoroutines();**
146. game.attemptPlayerMove();
147. }
149. *// method to get the time in a nice string format*
150. **public string GetTime()**
151. {
152. *// take float time of seconds and convert to string of minutes and seconds*
153. **int** minutes = Mathf.FloorToInt(Timer / 60F);
154. **int** seconds = Mathf.FloorToInt(Timer - minutes \* 60);
155. **string niceTime = string.Format("{0:0}:{1:00}", minutes, seconds);**
156. **return** niceTime;
157. }
159. *// interface method to get the time in seconds*
160. **float UI.GetTimer()**
161. {
162. **return** Timer;
163. }
165. ***// method for ending the player's turn***
166. **public** **void** EndTurn()
167. {
168. AudioManager.i.StopAll();
169. **if** (!overlay.isZoomed)
170. **{**
171. overlay.ZoomButton();
172. }
173. GetPlayerGameObject(**Interface**.game.GetCurrentPlayer().GetID()).isCurrentPlayer = **false**;
175. **StopAllPlayerCoroutines();**
177. *// Close all GUIS*
178. CloseAllGameUIs();
179. **Interface**.game.EndTurn();
180. **}**
182. *// method to close all game UIs*
183. **public** **void** CloseAllGameUIs()
184. {
185. **FindAnyObjectByType<InventoryPopup>()?.CloseGUI();**
186. FindAnyObjectByType<ShopOverlay>()?.CloseGUI();
187. FindAnyObjectByType<TradingInterface>()?.CloseGUI();
188. FindAnyObjectByType<PlayerChoice>()?.CloseGUI();
189. FindAnyObjectByType<PopupController>()?.CloseGUI();
190. **if (FindAnyObjectByType<PlayerUIOverlay>())**
191. Destroy(FindObjectOfType<PlayerUIOverlay>().gameObject);
192. }
194. *// method to find the connection gameobject between two nodes*
195. **public ConnectionAnimator FindConnectionGameObject(Node v1, Node v2)**
196. {
197. Vector3 searchingpos = GetConnectionGlobalPos(v1, v2);
198. **foreach** (ConnectionAnimator con **in** FindObjectsOfType<ConnectionAnimator>())
199. {
200. **if (con.transform.position == searchingpos)**
201. {
202. **return** con;
203. }
204. }
205. **return null;**
206. }
208. *// interface method to begin the game*
209. **void** UI.BeginGame(**float** timeleft)
210. **{**
211. StartCoroutine(WaitThenLoad(timeleft));
212. }
214. *// method to wait for the game to load before starting the first player's turn*
215. **IEnumerator WaitThenLoad(float timeleft)**
216. {
217. **while** (game == **null**)
218. {
219. **yield** **return** **null**;
220. **}**
221. **yield** **return** new WaitForEndOfFrame();
222. DisplayPlayers();
223. DisplayBoard();
224. **yield** **return** 0;
225. **game.StartTurn(timeleft);**
226. }
228. *// method to initialise the game board on the screen*
229. **public** **void** DisplayBoard()
230. **{**
231. **int** resourceID;
232. Vector3Int gridPos;
233. **foreach** (**var** entry **in** game.board.GetResourcesOnBoard())
234. {
236. resourceID = Array.IndexOf(Resource.RESOURCES, entry.**Value**.ToString());*// is the index of the resource in the list of resources*
237. gridPos = CubicToOddRow(entry.Key);
238. tilemap.SetTile(new Vector3Int(gridPos.x, gridPos.y), resources[resourceID]);
239. }
241. **foreach** (Node n **in** game.board.GetAllNodes())
242. {
244. NodeButton nodeui = Instantiate(NodePrefab, new Vector3(), Quaternion.identity, GameObject.FindGameObjectWithTag("NodeParent").transform).GetComponent<NodeButton>();
245. **nodeui.node = n;**
246. nodeui.NodePos = ConvertVector(n.position);
247. nodeui.transform.position = GetNodeGlobalPos(n);
248. nodeui.UpdateSettlement();
250. **nodes.Add(nodeui);**
251. }
252. }
254. *// method to initialise the players on the screen*
255. **public void DisplayPlayers()**
256. {
257. **foreach** (Player pl **in** game.gamePlayers)
258. {
259. PlayerAnimator playUI = Instantiate(PlayerPrefab, new Vector3(), Quaternion.identity, GameObject.FindGameObjectWithTag("PlayerParent").transform).GetComponent<PlayerAnimator>();
260. **playUI.player = pl;**
261. playUI.gameObject.name = pl.playerName;
262. playUI.GetComponent<SpriteRenderer>().color = textToColor(pl.color);
263. playUI.transform.position = GetNodeGlobalPos(game.board.GetNode(pl.position));
265. **}**
266. }
268. *// interface method to animate the player along a path of nodes*
269. **void** UI.UpdatePlayer(Stack<Node> path)
270. **{**
271. StartCoroutine(MovePlayerAlongPath(path));
272. }
273. IEnumerator MovePlayerAlongPath(Stack<Node> path)
274. {
275. **overlay.FinishMove();**
276. **while** (path.Count > 0)
277. {
278. Node otherNode = path.Pop();
279. Vector3 pos = GetNodeGlobalPos(otherNode);
280. **LeanTween.move(GetPlayerGameObject(Interface.game.GetCurrentPlayer().GetID()).gameObject, pos, 0.5f).setEase(LeanTweenType.easeInOutElastic);**
281. LeanTween.move(Camera.main.gameObject, pos + new Vector3(0f, 0f, -10f), 0.3f).setEase(LeanTweenType.easeInSine).setDelay(0.5f);
282. AudioManager.i.Play("UIEffect");
283. **yield** **return** new WaitForSeconds(0.55f);
284. }
285. **yield return null;**
286. }
288. *// interface method to update the connection between two nodes*
289. **void** UI.UpdateConnection(Node otherNode, Node current, Connection con)
290. **{**
291. **var** x = current.position;
292. **var** y = otherNode.position;
293. ConnectionAnimator conui = FindConnectionGameObject(current, otherNode);
294. **if** (conui == **null**)
295. **{**
296. conui = Instantiate(ConnectionPrefab, new Vector3(), Quaternion.identity, GameObject.FindGameObjectWithTag("ConnectionParent").transform).GetComponent<ConnectionAnimator>();
297. conui.UpdateConnection(ConvertVector(x), ConvertVector(y));
298. conui.transform.position = GetConnectionGlobalPos(current, otherNode);
299. }
300. **conui.connection = con;**
301. conui.UpdateDisplay();
302. }
304. *// method to get the color of the specified player*
305. **public Color GetPlayerColor(int playerID)**
306. {
307. **foreach** (Player pdl **in** game.gamePlayers)
308. {
309. **if** (pdl.GetID() == playerID)
310. **{**
311. **return** textToColor(pdl.color);
312. }
313. }
314. Debug.LogError("Failed to find color");
315. **return Color.clear;**
316. }
318. *// interface method to update the settlement on a node*
319. **void** UI.UpdateSettlement(Node otherNode)
320. **{**
321. **var** x = otherNode.position;
322. NodeButton nodeui = FindNodeGameObject(ConvertVector(x)); *// no need to instantiate as the nodes were made on startup*
323. nodeui.UpdateSettlement();
324. }
326. *// method to get the player gameobject for the specified player*
327. **public** PlayerAnimator GetPlayerGameObject(**int** playerID = -1)
328. {
329. **if** (playerID == -1)
330. **{**
331. playerID = game.GetCurrentPlayer().GetID();
332. }
333. **foreach** (**var** a **in** FindObjectsOfType<PlayerAnimator>())
334. {
335. **if (a.player.GetID() == playerID)**
336. {
337. **return** a;
338. }
339. }
340. **return null;**
341. }
343. *///////////////////////////////////////////////////////////////////////////////////////////////////////////////*
344. *// Used to represent and convert hexagonal grid of nodes and connection between cubic coordinates, odd row coordinates and global positions*
345. ***// Skill A: Use of Complex Mathematical Model***
347. **public** Vector3 GetConnectionGlobalPos(Node v1, Node v2)
348. {
349. HashSet<System.Numerics.Vector3> starthexes = new HashSet<System.Numerics.Vector3>();
350. **HashSet<System.Numerics.Vector3> endhexes = new HashSet<System.Numerics.Vector3>();**
351. **float** totalX = 0f;
352. **float** totalY = 0f;
353. **float** Z = 0f;
355. **foreach (var vec in v1.GetHexNeighbours())**
356. {
357. starthexes.**Add**(vec);
358. }
359. **foreach** (**var** vec **in** v2.GetHexNeighbours())
360. **{**
361. endhexes.**Add**(vec);
362. }
363. Vector3 center;
364. **foreach** (**var** HexPos **in** starthexes.Except(endhexes).Union(endhexes.Except(starthexes)))
365. **{**
366. center = GetHexGlobalPos(CubicToOddRow(HexPos));
367. totalX += center.x;
368. totalY += center.y;
369. Z = center.z;
370. **}**
371. **return** new Vector3(totalX / 2, totalY / 2, Z);
372. }
374. *// get the global position of a node*
375. **public Vector3 GetNodeGlobalPos(Node node)**
376. {
377. Vector3 center;
378. **float** totalX = 0f;
379. **float** totalY = 0f;
380. **float Z = 0f;**
381. **foreach** (**var** vec **in** node.GetHexNeighbours())
382. {
383. center = GetHexGlobalPos(CubicToOddRow(vec));
385. **totalX += center.x;**
386. totalY += center.y;
387. Z = center.z;
388. }
389. **float** avgX = totalX / 3;
390. **float avgY = totalY / 3;**
391. *// to get the center of the node, we average the x and y of the 3 hexes around it*
392. **return** new Vector3(avgX, avgY, Z);
393. }
395. ***// Get the global position of a hexagonal grid cell***
396. **public** Vector3 GetHexGlobalPos(Vector3 cellPos)
397. {
398. Vector3Int Pos = new Vector3Int((**int**)cellPos.x, (**int**)cellPos.y, 0);
399. **return** gridLayout.CellToWorld(Pos);
400. **}**
402. *// convert a System.Numerics.Vector3 to a Unity Vector3*
403. **public** **static** Vector3 ConvertVector(System.Numerics.Vector3 vec)
404. {
405. **return new Vector3(vec.X, vec.Y, vec.Z);**
406. }
408. *// convert the cubic coordinates of the hexagonal grid used in game calculations to odd row coordinates used in Unity*
409. **public** **static** Vector3Int CubicToOddRow(System.Numerics.Vector3 vec)
410. **{**
411. **int** col = (**int**)(vec.Z + ((vec.X - ((**int**)vec.X & 1)) / 2));
412. **int** row = (**int**)vec.X;
413. **return** new Vector3Int(col, row, 0);
414. }
415. ***//////////////////////////////////////////////////////////////////////////////////////////////////////////////***
417. *// convert a string to a color*
418. **public** **static** Color textToColor(**string** color)
419. {
420. **switch (color)**
421. {
422. **case** "blue":
423. **return** Color.blue;
424. **case** "cyan":
425. **return Color.cyan;**
426. **case** "green":
427. **return** Color.green;
428. **case** "grey":
429. **return** Color.grey;
430. **case "magenta":**
431. **return** Color.magenta;
432. **case** "red":
433. **return** Color.red;
434. **case** "white":
435. **return Color.white;**
436. **case** "yellow":
437. **return** Color.yellow;
438. **default**:
439. **return** Color.clear;
440. **}**
441. }
443. *// method to get the node gameobject at the specified position*
444. **public** NodeButton FindNodeGameObject(Vector3 pos)
445. **{**
446. **foreach** (NodeButton n **in** nodes)
447. {
448. **if** (n.NodePos == pos)
449. {
450. **return n;**
451. }
452. }
453. Debug.LogError(pos);
454. **return** **null**;
455. **}**
457. *// interface method to show a resource being given to the player*
458. **void** UI.ShowResource(System.Numerics.Vector3 u, NEAGame.Resource resource, System.Numerics.Vector3 endDest)
459. {
460. **StartCoroutine(AnimateResource(u, resource, endDest));**
461. }
462. IEnumerator AnimateResource(System.Numerics.Vector3 u, NEAGame.Resource resource, System.Numerics.Vector3 Dest)
463. {
464. Vector3 spawnpos;
465. **yield return new WaitForSeconds(0.5f);**
466. Vector3 destination = ConvertVector(Dest);
467. **if** (destination == new Vector3(0, 0, 0))
468. {
469. spawnpos = GetHexGlobalPos(CubicToOddRow(u));
470. **destination = overlay.InventoryInput.transform.position;**
471. }
472. **else**
473. {
474. destination = GetNodeGlobalPos(game.board.GetNode(Dest));
475. **spawnpos = GetNodeGlobalPos(game.board.GetNode(u));**
476. }
477. CardCollection card = Instantiate(CardObj, spawnpos, Quaternion.identity).GetComponent<CardCollection>();
478. card.SetCard(resource.GetHashCode(), destination);
479. }
481. *// interface method to get the user's choice of node*
482. **void** UI.GetUserNodeChoice(Node[] options, Action<Node> callback)
483. {
484. SelectedNode = **null**;
485. **foreach (Node choice in options)**
486. {
487. NodeButton node = FindNodeGameObject(ConvertVector(choice.position));
488. node.EnableButton();
489. }
490. **coroutine = WaitForNodeChoice(callback);**
491. StartCoroutine(coroutine);
492. }
494. *// method to wait for the user to choose a node*
495. **IEnumerator WaitForNodeChoice(Action<Node> callback) *// pass in function for moving vs buying***
496. {
497. **while** (SelectedNode is **null**)
498. {
499. **yield** **return** new WaitForSeconds(0.01f);
500. **}**
501. callback(SelectedNode.node);
502. SelectedNode = **null**;
503. }
505. ***// method to handle the user clicking on a node option***
506. **public** **void** OnNodeClick(NodeButton node)
507. {
508. SelectedNode = node;
509. **foreach** (NodeButton n **in** nodes)
510. **{**
511. n.DisableButton();
512. }
513. }
515. ***// method to stop all player coroutines***
516. **public** **void** StopAllPlayerCoroutines()
517. {
518. **foreach** (NodeButton n **in** nodes)
519. {
520. **n.DisableButton();**
521. }
522. **if** (coroutine != **null**)
523. {
524. StopCoroutine(coroutine);
525. **}**
526. }
528. *// method to display the inventory of the current player*
529. **public** **void** OpenInventory()
530. **{**
531. StopAllPlayerCoroutines();
532. InventoryPopup inv = Instantiate(inventoryPopup).GetComponent<InventoryPopup>();
533. **foreach** (KeyValuePair<Resource, **int**> entry **in** **Interface**.game.GetCurrentPlayer().getResources())
534. {
535. **inv.Display(entry.Key.ToString(), entry.Value);**
536. }
538. }
540. ***// method to open the player selection overlay for trading***
541. **public** IEnumerator OpenTrade()
542. {
543. StopAllPlayerCoroutines();
544. SceneTransition.i.PlayAnimation();
545. **yield return new WaitForSeconds(0.25f);**
546. PlayerChoice overlay = Instantiate(PlayerChoicePopup).GetComponent<PlayerChoice>();
547. List<Player> options = new List<Player>();
548. **foreach** (Player pl **in** **Interface**.game.gamePlayers)
549. {
550. **if (pl != Interface.game.GetCurrentPlayer() && !pl.isPlayerAI())**
551. {
552. options.**Add**(pl);
553. }
554. }
555. **overlay.Setup(options);**
556. **yield** **return** **null**;
557. }
559. *// method called when the player selects a partner to trade with*
560. **public void SelectPartner(Player pl)**
561. {
562. StartCoroutine(DisplayTrade(pl));
563. }
565. ***// function to display the trade overlay between players***
566. IEnumerator DisplayTrade(Player pl)
567. {
568. SceneTransition.i.PlayAnimation();
569. **yield** **return** new WaitForSeconds(0.1f);
570. **TradingInterface inv = Instantiate(TradePopup).GetComponent<TradingInterface>();**
571. inv.Setup(**Interface**.game.GetCurrentPlayer(), pl);
572. **yield** **return** **null**;
573. }
575. ***// function to register a trade between players***
576. **public** IEnumerator RegisterTrade(Dictionary<**int**, **int**> trades, Player other)
577. {
578. **yield** **return** new WaitForSeconds(1.25f);
579. Dictionary<Resource, **int**> GameDict = new Dictionary<Resource, **int**>();
580. **foreach (var entry in trades)**
581. {
582. GameDict.**Add**(new Resource(entry.Key + 1), entry.**Value**);
583. }
584. game.CompleteTrade(other, GameDict);
585. **}**


589. *// methods to handle the shop event*
590. **public void OpenShop()**
591. {
592. StopAllPlayerCoroutines();
593. StartCoroutine(DisplayShop());
594. }
595. **IEnumerator DisplayShop()**
596. {
597. ShopOverlay sv = Instantiate(shoppingPopup).GetComponent<ShopOverlay>();
598. **yield** **return** new WaitForSeconds(0.1f);
600. **}**
602. *// method to attempt a purchase from the shop*
603. **public** **void** AttemptPurchase(**string** name)
604. {
605. **switch (name)**
606. {
607. **case** "Road":
608. game.tryPurchaseRoad();
609. **break**;
610. **case "Wall":**
611. game.tryPurchaseWall();
612. **break**;
613. **case** "Village":
614. game.tryPurchaseVillage();
615. **break;**
616. **case** "City":
617. game.tryPurchaseCity();
618. **break**;
619. **default**:
620. **Debug.LogError("Invalid purchase name");**
621. **break**;
622. }
623. }
625. ***// interface method to handle the end of the game***
626. **void** UI.HandleWinner(Player winner)
627. {
628. AudioManager.i.StopAll();
629. AudioManager.i.Play("Victory");
630. **CloseAllGameUIs();**
631. SceneTransition.i.SendToScene("Victory");
632. }
633. }

#### UIManager.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Numerics;
5. **namespace NEAGame**
6. {
7. *//////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
8. */// <summary>*
9. */// The <c>UI</c> interface is used to define the methods that the game will use to interact with the user.<br/>*
10. ***/// Skill A: Complex OOP - Interface***
11. */// </summary>*
12. */// //////////////////////////////////////////////////////////////////////////////////////////////////////////////*
13. **public** **interface** UI
14. {
15. ***// gets the current time of the game in seconds***
16. **float** GetTimer();
17. *// offers the user a choice of nodes to select and then calls the method passing the selected node as a parameter*
18. **void** GetUserNodeChoice(Node[] options, Action<Node> method);
19. *// displays a message to the user*
20. **void CreatePopup(string message);**
21. *// called when the game is over to display the winner*
22. **void** HandleWinner(Player winner);
23. *// begin the turn of the current player*
24. **void** BeginTurn(**float** time);
25. ***// initialise the game screen***
26. **void** BeginGame(**float** Timer);
27. *// Updates the connection between two nodes*
28. **void** UpdateConnection(Node node1, Node node2, Connection con);
29. *// Updates the settlement of a node*
30. **void UpdateSettlement(Node otherNode);**
31. *// Updates the player's position over a path*
32. **void** UpdatePlayer(Stack<Node> otherNode);
33. *// Shows the user that a resource has been added to their inventory from a node*
34. **void** ShowResource(Vector3 u, Resource resource, Vector3 optional);
35. ***// tests a condition and raises an error if it is not met***
36. **void** Assert(**bool** test);
37. }
39. */// <summary>*
40. ***/// Discontinued when project became entirely Unity event-based***
41. */// </summary>*
42. **public** **class** TerminalUI { }
44. }

#### JSON\_manager.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** UnityEngine;
5. **namespace NEAGame**
6. {
7. */// <summary>*
8. */// The <c>JSON\_manager</c> class is used to manage the serialization and deserialization of the game to and from JSON.*
9. */// </summary>*
10. **class JSON\_manager**
11. {
12. **public** **readonly** **string**[] SAVE\_FILES = { "Save1", "Save2", "Save3", "Save4" };
13. **private** FileHandler fileHandler;
15. ***// Constructor for the JSON manager that takes in the save file number***
16. **public** JSON\_manager(**int** Save)
17. {
18. **string** fullpath = Application.persistentDataPath + "/" + SAVE\_FILES[Save];
19. fileHandler = new FileHandler(fullpath);
20. **}**
22. *// Checking if the save file exists*
23. **public** **bool** DoesGameExist()
24. {
25. **return fileHandler.IsMade;**
26. }
28. *// Clearing the save file*
29. **public** **void** ClearSave()
30. **{**
31. fileHandler.Delete();
32. }
34. *// Saving the game to a file*
35. **public void SaveGame(TravelersOfCatan game)**
36. {
37. GameWrapper gameWrapper = new GameWrapper(game);
38. *// save the game to a file in unity persistent data path*
39. **string** json = JSONWrapper<GameWrapper>.Dump(gameWrapper);
40. **fileHandler.Save(json);**
41. }
43. *// Loading the game from a file*
44. **public** GameWrapper LoadGame()
45. **{**
47. **string** json = fileHandler.Load();
48. GameWrapper gameWrapper = JSONWrapper<GameWrapper>.Load(json);
49. **return** gameWrapper;
50. **}**
51. }

54. */// <summary>*
55. ***/// <c>JSONWrapper</c> is an abstract class that defines the methods to serialize and deserialize an object to and from JSON.***
56. */// </summary>*
57. **public** **abstract** **class** JSONWrapper<T> **where** T : JSONWrapper<T>, new()
58. {
59. *// Method to serialize the game to JSON*
60. **public static string Dump(T t)**
61. {
62. **return** JsonUtility.ToJson(t);
63. }
65. ***// Method to deserialize the game from JSON***
66. **public** **static** T Load(**string** jsonString)
67. {
68. T t = new T();
69. JsonUtility.FromJsonOverwrite(jsonString, t);
70. **return t;**
71. }
72. }
74. *//////////////////////////////////////////////////////////////////////////////////////*
75. ***// Serializable Class Wrappers to be used for JSON Serialization and Deserialization***

78. [Serializable]
79. **public** **class** ResourceWrapper : JSONWrapper<ResourceWrapper>
80. **{**
81. **public** **int** id;
83. **public** ResourceWrapper() { }
85. **public ResourceWrapper(Resource resource)**
86. {
87. id = resource.GetHashCode();
88. }
89. }
91. [Serializable]
92. **public** **class** HexagonUnitWrapper : JSONWrapper<HexagonUnitWrapper>
93. {
95. **public List<Vector3> \_Keys = new List<Vector3>();**
96. **public** List<ResourceWrapper> \_Values = new List<ResourceWrapper>();
98. **public** HexagonUnitWrapper() { }
99. **public** HexagonUnitWrapper(Dictionary<System.Numerics.Vector3, Resource> board)
100. **{**
101. **foreach** (**var** entry **in** board)
102. {
103. \_Keys.**Add**(UnityUI.ConvertVector(entry.Key));
104. \_Values.**Add**(new ResourceWrapper(entry.**Value**));
105. **}**
106. }
107. }
109. [Serializable]
110. **public class SettlementWrapper : JSONWrapper<SettlementWrapper>**
111. {
113. **public** **string** status;
114. **public** **int** occupantID;
116. **public** SettlementWrapper() { }
117. **public** SettlementWrapper(Connection connection)
118. {
119. status = connection.GetStatus();
120. **occupantID = connection.GetOccupant();**
121. }
123. **public** SettlementWrapper(Building node)
124. {
125. **status = node.GetStatus();**
126. occupantID = node.GetOccupant();
127. }
128. }

131. [Serializable]
132. **public** **class** AdjMatrixWrapper : JSONWrapper<AdjMatrixWrapper>
133. {
135. **public List<Vector3> \_Keys = new List<Vector3>();**
136. **public** List<AdjMatrixBottomWrapper> \_Values = new List<AdjMatrixBottomWrapper>();
138. **public** AdjMatrixWrapper() { }
139. **public** AdjMatrixWrapper(Dictionary<System.Numerics.Vector3, Dictionary<System.Numerics.Vector3, Connection>> connections)
140. **{**
142. **foreach** (**var** entry **in** connections)
143. {
144. \_Keys.**Add**(UnityUI.ConvertVector(entry.Key));
145. **\_Values.Add(new AdjMatrixBottomWrapper(entry.Value));**
146. }
147. }
148. }
150. **[Serializable]**
151. **public** **class** AdjMatrixBottomWrapper : JSONWrapper<AdjMatrixBottomWrapper>
152. {
154. **public** List<Vector3> \_Keys = new List<Vector3>();
155. **public List<SettlementWrapper> \_Values = new List<SettlementWrapper>();**
157. **public** AdjMatrixBottomWrapper() { }
158. **public** AdjMatrixBottomWrapper(Dictionary<System.Numerics.Vector3, Connection> connections)
159. {
160. **foreach (var entry in connections)**
161. {
162. **if** (entry.**Value**.GetStatus() != "Empty")
163. {
164. \_Keys.**Add**(UnityUI.ConvertVector(entry.Key));
165. **\_Values.Add(new SettlementWrapper(entry.Value));**
166. }
167. }
168. }
169. }
171. [Serializable]
172. **public** **class** PlayerWrapper : JSONWrapper<PlayerWrapper>
173. {
174. **public** Vector3 position;
175. **public int playerNumber;**
176. **public** **string** playerName;
177. **public** **int** victoryPoints;
178. **public** **int** moves;
179. **public** **bool** isAI;
180. **public string color;**
181. **public** Vector3 origin;
182. **public** InventoryWrapper resources;
184. **public** PlayerWrapper() { }
185. **public PlayerWrapper(Player player)**
186. {
187. position = UnityUI.ConvertVector(player.position);
188. playerNumber = player.GetID();
189. playerName = player.playerName;
190. **victoryPoints = player.getVictoryPoints();**
191. moves = player.moves;
192. isAI = player.isPlayerAI();
193. color = player.color;
194. origin = UnityUI.ConvertVector(player.origin);
195. **resources = new InventoryWrapper(player.getResources());**
196. }
197. }
199. [Serializable]
200. **public class InventoryWrapper : JSONWrapper<InventoryWrapper>**
201. {
202. **public** List<ResourceWrapper> \_Keys = new List<ResourceWrapper>();
203. **public** List<**int**> \_Values = new List<**int**>();
205. **public InventoryWrapper() { }**
206. **public** InventoryWrapper(Dictionary<Resource, **int**> resources)
207. {
208. **foreach** (**var** entry **in** resources)
209. {
210. **\_Keys.Add(new ResourceWrapper(entry.Key));**
211. \_Values.**Add**(entry.**Value**);
212. }
213. }
214. }
216. [Serializable]
217. **public** **class** AllNodesWrapper : JSONWrapper<AllNodesWrapper>
218. {
219. **public** List<Vector3> \_Keys = new List<Vector3>();
220. **public List<NodeWrapper> \_Values = new List<NodeWrapper>();**
222. **public** AllNodesWrapper() { }
223. **public** AllNodesWrapper(Dictionary<System.Numerics.Vector3, Node> nodes)
224. {
225. **foreach (var entry in nodes)**
226. {
227. **if** (entry.**Value**.status.GetStatus() == "Empty") **continue**;
228. \_Keys.**Add**(UnityUI.ConvertVector(entry.Key));
229. \_Values.**Add**(new NodeWrapper(entry.**Value**));
230. **}**
231. }
232. }
234. [Serializable]
235. **public class NodeWrapper : JSONWrapper<NodeWrapper>**
236. {
237. **public** Vector3 position;
238. **public** SettlementWrapper status;
240. **public NodeWrapper() { }**
241. **public** NodeWrapper(Node node)
242. {
243. position = UnityUI.ConvertVector(node.position);
244. status = new SettlementWrapper(node.status);
245. **}**
246. }
248. [Serializable]
249. **public** **class** BoardWrapper : JSONWrapper<BoardWrapper>
250. **{**
252. **public** HexagonUnitWrapper board;
253. **public** AdjMatrixWrapper connections;
254. **public** AllNodesWrapper nodes;
256. **public** BoardWrapper() { }
258. }
260. **[Serializable]**
261. **public** **class** GameWrapper : JSONWrapper<GameWrapper>
262. {
264. **public** **int** winVictoryPoints;
265. **public float timePerMove;**
266. **public** **float** timer; *// controlled by UI*
267. **public** **int** turn;
268. **public** List<PlayerWrapper> allPlayers = new List<PlayerWrapper>();
269. **public** BoardWrapper board;
271. **public** GameWrapper() { }
273. **public** GameWrapper(TravelersOfCatan game)
274. {
275. **winVictoryPoints = game.WinningVictoryPoints;**
276. timePerMove = game.TimePerMove;
277. timer = game.GetGameTime();
278. turn = game.gamePlayers.IndexOf(game.GetCurrentPlayer());
279. **foreach** (Player player **in** game.gamePlayers)
280. **{**
281. allPlayers.**Add**(new PlayerWrapper(player));
282. }
283. board = game.board.SoftSerialize();
284. }
285. **}**
286. }

#### FileHandler.cs

1. **using** UnityEngine;
2. **using** System;
3. **using** System.IO;
5. ***/// <summary>***
6. */// A <c>FileHandler</c> object is used to manage the saving and loading of data to and from files stored in the Unity persistent data path. This also has the option to encrypt the data before saving it to the file.*
7. */// <br/>Credit to <seealso href="https://www.youtube.com/watch?v=KZft1p8t2lQ"/> for the tutorial on implementing this concept*
8. */// </summary>*
9. **public** **class** FileHandler
10. **{**
11. **private** **string** filepath = "";
12. **public** **bool** IsMade;
13. **private** **bool** useEncryption = **false**;
14. **private** **readonly** **string** encryptionCodeWord = "SomeRandomStringKeyToConvertIntoBinaryForXOR->aiuaeogmk3GJEK834FEJSAK->";
15. ***// practically speaking, this is not easily crackable, but it's not perfect as the same key is repeated and is not infinitely long***
17. **public** FileHandler(**string** filepath)
18. {
19. **this**.filepath = filepath;
20. **encryptionCodeWord += filepath; *// add the filepath to the encryption key to make it more unique***
21. **string** suffix = FindSuffix();
22. **if** (suffix == "")
23. {
24. IsMade = **false**;
25. **this.filepath += ".bin";**
26. useEncryption = **true**;
27. }
28. **else**
29. {
30. **IsMade = true;**
31. useEncryption = suffix == ".bin";
32. **this**.filepath += suffix;
33. }
34. }
36. *// method to find the suffix of the save file*
37. **string** FindSuffix()
38. {
39. **if** (File.Exists(filepath + ".json"))
40. **{**
41. **return** ".json";
42. }
43. **else** **if** (File.Exists(filepath + ".bin"))
44. {
45. **return ".bin";**
46. }
47. **else**
48. {
49. **return** "";
50. **}**
51. }
53. *// method to delete the file*
54. **public** **void** Delete()
55. **{**
56. **if** (IsMade)
57. {
58. File.Delete(filepath);
59. IsMade = **false**;
60. **}**
61. }
63. *///////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
64. */// <summary>*
65. ***/// function to load the data from the file***
66. */// Skill B: File Read/Write*
67. */// </summary>*
68. *///////////////////////////////////////////////////////////////////////////////////////////////////////////////////*
69. **public** **string** Load()
70. **{**
72. **string** dataToLoad = "";
73. **if** (File.Exists(filepath))
74. {
75. **try**
76. {
77. *// load the serialized data from the file*
78. **using** (FileStream stream = new FileStream(filepath, FileMode.Open))
79. {
80. **using (StreamReader reader = new StreamReader(stream))**
81. {
82. dataToLoad = reader.ReadToEnd();
83. }
84. }
86. *// optionally decrypt the data*
87. **if** (useEncryption)
88. {
89. dataToLoad = EncryptDecrypt(dataToLoad);
90. **}**
91. }
92. **catch** (Exception e)
93. {
94. Debug.LogError($"Error occured when trying to load data {e}");
95. **}**
96. }
97. **return** dataToLoad;
98. }
100. ***///////////////////////////////////////////////////////////////////////////////////////////////////////////////////***
101. */// <summary>*
102. */// function to save the data to the file*
103. */// Skill B: File Read/Write*
104. */// </summary>*
105. ***///////////////////////////////////////////////////////////////////////////////////////////////////////////////////***
106. **public** **void** Save(**string** data)
107. {
108. **try**
109. {
110. **Directory.CreateDirectory(Path.GetDirectoryName(filepath)); *// create the directory if it doesn't exist***
112. **if** (useEncryption)
113. {
114. *// encrypt the data before saving it*
115. **data = EncryptDecrypt(data);**
116. }
118. *// write the serialized data to the file*
119. **using** (FileStream stream = new FileStream(filepath, FileMode.Create))
120. **{**
121. **using** (StreamWriter writer = new StreamWriter(stream))
122. {
123. writer.Write(data);
124. }
125. **}**
126. }
127. **catch** (Exception e)
128. {
129. Debug.LogError("Error occured when trying to save data to file: " + filepath + "**\n**" + e);
130. **}**
131. }
133. */// <summary>*
134. */// Perform XOR encryption/decryption on the data<br/>*
135. ***/// Credit to <seealso href="https://stackoverflow.com/questions/2532668/help-me-with-xor-encryption"/> for the XOR encryption algorithm***
136. */// </summary>*
137. **public** **string** EncryptDecrypt(**string** data)
138. {
139. **string** key = encryptionCodeWord;
140. **int dataLen = data.Length;**
141. **int** keyLen = key.Length;
142. **char**[] output = new **char**[dataLen];
144. **for** (**int** i = 0; i < dataLen; ++i)
145. **{**
146. output[i] = (**char**)(data[i] ^ key[i % keyLen]);
147. }
149. **return** new **string**(output);
150. **}**
151. }

**Unity Animators and overlay controllers**

#### GamePauseOverlay.cs

1. **using** UnityEngine;
2. **using** UnityEngine.UI;
4. */// <summary>*
5. ***/// <c>GamePauseOverlay</c> is a class that manages the pause and settings overlays.***
6. */// </summary>*
7. **public** **class** GamePauseOverlay : MonoBehaviour
8. {
9. **public** Button Resume;
10. **public Slider Slider;**
11. **public** Button MuteBG;
12. **public** Button MuteSFX;
13. **public** GameObject Panel;
14. **public** GameObject PauseBtn;
15. **public GameObject Display;**
16. **public** GameObject Save;
17. **public** GameObject MusicParent;
18. **public** Sprite MusicUnmuted;
19. **public** Sprite MusicMuted;
21. *// Start is called before the first frame update*
22. **void** Start()
23. {
24. AudioManager.i.StopAll();
25. **Canvas myCanvas = GetComponent<Canvas>();**
26. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
27. myCanvas.worldCamera = Camera.main;
28. myCanvas.sortingLayerName = "UI";
29. myCanvas.sortingOrder = 700; *// VERY foreground*
30. **Slider.onValueChanged.AddListener((float v) => volumeChange(v));**
31. Slider.**value** = AudioManager.i.VolumeModifier;
32. *// Add listeners to the buttons*
33. Resume.onClick.AddListener(() =>
34. {
35. **AudioManager.i.Play("UIClick");**
36. CloseGUI();
37. });
38. MuteBG.onClick.AddListener(() =>
39. {
40. **AudioManager.i.ToggleMute(Background: true); *// Mutes background music***
41. UpdateMusicBtn(**true**);
42. });
43. MuteSFX.onClick.AddListener(() =>
44. {
45. **AudioManager.i.ToggleMute(Background: false); *// Mutes sound effects***
46. UpdateMusicBtn(**false**);
47. });
48. Save.GetComponent<Button>().onClick.AddListener(OnExit);
49. *// Set the mute buttons to the correct initial state*
50. **if (AudioManager.i.isMuted(Background: true))**
51. {
52. MuteBG.GetComponent<Image>().sprite = MusicMuted;
53. }
54. UpdateMusicBtn(**true**); UpdateMusicBtn(**false**);
55. **}**
57. *// The settings page is the pause page but with the save button removed*
58. **public** **void** RemoveSaveButton()
59. {
60. **Save.SetActive(false);**
61. }
63. *// Method called when the volume slider is changed by Unity*
64. **void** volumeChange(**float** x)
65. **{**
66. AudioManager.i.ChangeMasterVolume(x);
67. UpdateMusicBtn(**true**); UpdateMusicBtn(**false**);
68. }
70. ***// Method to close the pause overlay***
71. **public** **void** CloseGUI()
72. {
73. *//lean tween everything away*
74. LeanTween.scale(Save, new Vector3(), 0.5f).setEase(LeanTweenType.easeInCubic);
75. **LeanTween.scale(MusicParent, new Vector3(), 0.5f).setEase(LeanTweenType.easeInCubic);**
76. LeanTween.scale(Display, new Vector3(), 0.75f).setEase(LeanTweenType.easeInCubic).setDelay(0.1f);
77. LeanTween.rotateAround(Display, Vector3.forward, 360, 0.5f).setEase(LeanTweenType.easeInCubic).setDelay(0.1f);
78. LeanTween.scale(PauseBtn, new Vector3(), 0.75f).setEase(LeanTweenType.easeInCubic).setDelay(0.1f);
79. LeanTween.scale(Panel, new Vector3(), 0.75f).setEase(LeanTweenType.easeInCubic).setOnComplete(() =>
80. **{**
82. UnityUI.**Interface**.TimerActive = **true**;
83. Destroy(gameObject);
85. **});**
86. }
88. *// Method to update the music buttons*
89. **void** UpdateMusicBtn(**bool** background)
90. **{**
91. Sprite img = AudioManager.i.isMuted(background) ? MusicMuted : MusicUnmuted;
92. **if** (Slider.**value** == 0.0f)
93. {
94. img = MusicMuted;
95. **}**
96. **if** (background)
97. {
98. MuteBG.GetComponent<Image>().sprite = img;
99. }
100. **else**
101. {
102. MuteSFX.GetComponent<Image>().sprite = img;
103. }
104. }
106. *// onclick method for the save and exit button.*
107. **void** OnExit()
108. {
109. AudioManager.i.Play("UIClick");
110. **if (!LeanTween.isTweening(Save))**
111. {
112. LeanTween.scale(Save, Save.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInCubic).setLoopPingPong(1);
113. }
114. UnityUI.**Interface**.CloseAllGameUIs();
115. **UnityUI.Interface.SaveAndExit();**
116. }
117. }

#### InventoryPopup.cs

1. **using** UnityEngine;
2. **using** UnityEngine.UI;
3. **using** TMPro;
5. ***/// <summary>***
6. */// <c>InventoryPopup</c> is the class that manages the inventory overlay that displays the current player's resources.*
7. */// </summary>*
8. **public** **class** InventoryPopup : MonoBehaviour
9. {
10. **public GameObject Background;**
11. **public** Button CloseButton;
12. **public** TextMeshProUGUI Brick;
13. **public** TextMeshProUGUI Sheep;
14. **public** TextMeshProUGUI Ore;
15. **public TextMeshProUGUI Wood;**
16. **public** TextMeshProUGUI Wheat;
18. *// Start is called before the first frame update*
19. **void** Start()
20. **{**
21. Canvas myCanvas = GetComponent<Canvas>();
22. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
23. myCanvas.worldCamera = Camera.main;
24. myCanvas.sortingLayerName = "UI";
25. **myCanvas.sortingOrder = 500;**
26. CloseButton.onClick.AddListener(CloseGUI);
28. Background.LeanScale(new Vector3(0, 0, 0), 0);
29. CloseButton.gameObject.LeanScale(new Vector3(0, 0, 0), 0);
30. **LeanTween.scale(Background, new Vector3(7.710301f, 2.923337f, 2.923337f), 0.75f).setEase(LeanTweenType.easeInCubic).setDelay(0.7f);**
31. LeanTween.scale(CloseButton.gameObject, new Vector3(1, 1, 1), 0.75f).setEase(LeanTweenType.easeInCubic).setDelay(0.7f);
32. }
34. **public** **void** CloseGUI()
35. **{**
36. AudioManager.i.Play("UIClick");
37. *// add a close animation*
38. LeanTween.scale(CloseButton.gameObject, new Vector3(0, 0, 0), 0.75f).setEase(LeanTweenType.easeInCubic).setDelay(0.1f);
39. LeanTween.moveLocalY(Brick.gameObject, -1000, 0.75f).setEase(LeanTweenType.easeOutSine).setDelay(0.2f);
40. **LeanTween.moveLocalY(Sheep.gameObject, -1000, 0.75f).setEase(LeanTweenType.easeOutSine).setDelay(0.3f);**
41. LeanTween.moveLocalY(Ore.gameObject, -1000, 0.75f).setEase(LeanTweenType.easeOutSine).setDelay(0.4f);
42. LeanTween.moveLocalY(Wood.gameObject, -1000, 0.75f).setEase(LeanTweenType.easeOutSine).setDelay(0.5f);
43. LeanTween.moveLocalY(Wheat.gameObject, -1000, 0.75f).setEase(LeanTweenType.easeOutSine).setDelay(0.6f);
44. LeanTween.scale(Background, new Vector3(0, 0, 0), 0.75f).setEase(LeanTweenType.easeInCubic).setDelay(0.7f).setOnComplete(() => {
45. **Destroy(gameObject);**
46. });
47. }
49. **public** **void** Display (**string** Name, **int** count)
50. **{**
51. Vector3 TextScale = new Vector3(0.1395292f, 0.53154f, 0.53154f);
52. **switch** (Name)
53. {
54. *// display the resource count*
55. **case ("Brick"):**
56. Brick.gameObject.transform.localScale = new Vector3(0, 0, 0);
57. LeanTween.scale(Brick.gameObject, TextScale, 0.75f).setEase(LeanTweenType.easeOutElastic).setDelay(0.1f);
58. Brick.text = FormatInt(count);
59. **break**;
60. **case ("Sheep"):**
61. Sheep.gameObject.transform.localScale = new Vector3(0, 0, 0);
62. LeanTween.scale(Sheep.gameObject, TextScale, 0.75f).setEase(LeanTweenType.easeOutElastic).setDelay(0.15f);
63. Sheep.text = FormatInt(count);
64. **break**;
65. **case ("Ore"):**
66. Ore.gameObject.transform.localScale = new Vector3(0, 0, 0);
67. LeanTween.scale(Ore.gameObject, TextScale, 0.75f).setEase(LeanTweenType.easeOutElastic).setDelay(0.2f);
68. Ore.text = FormatInt(count);
69. **break**;
70. **case ("Wood"):**
71. Wood.gameObject.transform.localScale = new Vector3(0, 0, 0);
72. LeanTween.scale(Wood.gameObject, TextScale, 0.75f).setEase(LeanTweenType.easeOutElastic).setDelay(0.25f);
73. Wood.text = FormatInt(count);
74. **break**;
75. **case ("Wheat"):**
76. Wheat.gameObject.transform.localScale = new Vector3(0, 0, 0);
77. LeanTween.scale(Wheat.gameObject, TextScale, 0.75f).setEase(LeanTweenType.easeOutElastic).setDelay(0.3f);
78. Wheat.text = FormatInt(count);
79. **break**;
80. **}**
81. }
83. **private** **string** FormatInt(**int** count)
84. {
85. ***// return string in form XX***
86. **if** (count < 10)
87. {
88. **return** "0" + count.ToString();
89. }
90. **else**
91. {
92. **return** count.ToString();
93. }
94. }
95. **}**

#### ColorChoiceControl.cs

1. **using** System.Collections.Generic;
2. **using** UnityEngine;
3. **using** UnityEngine.Assertions;
4. **using** UnityEngine.UI;
6. */// <summary>*
7. */// <c>ColorChoiceControl</c> is the class that manages the overlay which allows players to choose their color.*
8. */// </summary>*
9. **public** **class** ColorChoiceControl : MonoBehaviour
10. **{**
12. **public** GameObject[] colorChoiceBtns;
13. **public** Sprite Taken;
14. **public** Sprite NotTaken;
15. **private string current = "clear";**
16. **private** **bool** isClosing = **false**;
18. [Header("Animation Objects")]
19. **public** GameObject panel;
20. **public GameObject CloseBtn;**
21. **public** GameObject ColParent;

24. *// Start is called before the first frame update*
25. **void Start()**
26. {
27. Canvas myCanvas = GetComponent<Canvas>();
28. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
29. myCanvas.worldCamera = Camera.main;
30. **myCanvas.sortingLayerName = "UI";**
31. myCanvas.sortingOrder = 600;
33. Vector3 init = panel.transform.localScale;
34. panel.transform.localScale = new Vector3(0, 0, 0);
35. **LeanTween.scale(panel, init, 0.5f).setEase(LeanTweenType.easeInSine).setDelay(0.5f);**
36. init = CloseBtn.transform.localScale;
37. CloseBtn.transform.localScale = new Vector3(0, 0, 0);
38. LeanTween.scale(CloseBtn, init, 0.5f).setEase(LeanTweenType.easeInSine).setDelay(0.5f);
39. CloseBtn.GetComponent<Button>().onClick.AddListener(CloseGUI);
40. **ColParent.GetComponent<CanvasGroup>().alpha = 0;**
41. LeanTween.alphaCanvas(ColParent.GetComponent<CanvasGroup>(), 1, 0.5f).setEase(LeanTweenType.easeInSine).setDelay(0.5f);
42. }
44. *// function to close the color choice overlay and save the current color*
45. **public void CloseGUI()**
46. {
47. **if** (isClosing)
48. **return**;
49. isClosing = **true**;
50. **FindObjectOfType<PlayersSetup>().NewColorSave(current);**
51. AudioManager.i.Play("UIClick");
52. *// Add a closing animation*
53. LeanTween.moveLocalY(ColParent, -1000f, 0.5f).setEase(LeanTweenType.easeOutSine).setDelay(0.1f);
54. LeanTween.rotateAround(CloseBtn , Vector3.forward, 360, 0.5f).setEase(LeanTweenType.easeShake).setDelay(0.0f);
55. **LeanTween.scale(CloseBtn , new Vector3(), 0.25f).setEase(LeanTweenType.easeShake).setDelay(0.25f);**
56. LeanTween.alphaCanvas(ColParent.GetComponent<CanvasGroup>(), 0, 0.75f).setEase(LeanTweenType.easeOutSine).setDelay(0.0f);
57. LeanTween.scale(panel, new Vector3(), 0.75f).setEase(LeanTweenType.easeOutSine).setDelay(0.25f).setOnComplete(() =>
58. {
59. Destroy(gameObject);
60. **});**
61. }
63. *// function to set up the color choice overlay with the available colors*
64. **public** **void** Setup(List<**string**> col, List<**int**> taken)
65. **{**
66. Assert.IsTrue(col.Count == colorChoiceBtns.Length);
67. **int** i = 0;
68. **foreach** (**string** c **in** col)
69. {
70. **colorChoiceBtns[i].transform.GetChild(0).GetComponent<Button>().onClick.AddListener(() => SetCurrent(c));**
71. colorChoiceBtns[i].transform.GetChild(0).GetComponent<Image>().color = UnityUI.textToColor(c);
72. **if** (taken.Contains(i))
73. {
74. colorChoiceBtns[i].GetComponent<Image>().sprite = Taken;
75. **colorChoiceBtns[i].transform.GetChild(0).GetComponent<Image>().sprite = Taken;**
76. colorChoiceBtns[i].transform.GetChild(0).GetComponent<Button>().interactable = **false**;
77. }
78. **else**
79. {
80. **colorChoiceBtns[i].transform.GetChild(1).gameObject.SetActive(false);**
81. }
82. i++;
83. }
84. }
86. *// onclick function to update the current color*
87. **public** **void** SetCurrent(**string** i)
88. {
89. AudioManager.i.Play("UIClick");
90. **if (current == i)**
91. **return**;
92. Color search = UnityUI.textToColor(current);
93. Color search2 = UnityUI.textToColor(i);
94. **foreach** (**var** btn **in** colorChoiceBtns)
95. **{**
96. **if** (btn.transform.GetChild(0).GetComponent<Image>().color == search)
97. {
98. btn.transform.GetChild(0).GetComponent<Image>().sprite = NotTaken;
99. btn.GetComponent<Image>().sprite = NotTaken;
100. **}**
101. **else** **if** (btn.transform.GetChild(0).GetComponent<Image>().color == search2)
102. {
103. btn.transform.GetChild(0).GetComponent<Image>().sprite = Taken;
104. btn.GetComponent<Image>().sprite = Taken;
105. **}**
106. }
107. current = i;
108. }
109. }

#### PlayerChoice.cs

1. **using** System.Collections;
2. **using** System.Collections.Generic;
3. **using** UnityEngine;
4. **using** UnityEngine.UI;
5. **using NEAGame;**
6. **using** TMPro;
8. */// <summary>*
9. */// <c>PlayerChoice</c> is the class that manages the overlay for selecting a player to trade with.*
10. ***/// </summary>***
11. **public** **class** PlayerChoice : MonoBehaviour
12. {
13. **public** GameObject ParentObj;
14. **public** GameObject CloseBtn;
15. **public GameObject PlayerSlotPrefab;**
16. **private** List<Player> options;
17. **private** **bool** found = **false**;
19. *// Close the GUI*
20. **public void CloseGUI()**
21. {
22. Destroy(gameObject);
23. }
25. ***// Start is called before the first frame update***
26. **void** Start()
27. {
28. Canvas myCanvas = GetComponent<Canvas>();
29. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
30. **myCanvas.worldCamera = Camera.main;**
31. myCanvas.sortingLayerName = "UI";
32. myCanvas.sortingOrder = 250;
33. CloseBtn.GetComponent<Button>().onClick.AddListener(CloseGUI);
34. }
36. *// Setup the GUI with the list of players*
37. **public** **void** Setup(List<Player> players)
38. {
39. options = players;
40. **StartCoroutine(AddPlayers());**
41. }
43. *// Add players to the GUI*
44. IEnumerator AddPlayers()
45. **{**
46. **foreach** (Player p **in** options)
47. {
48. PlayerTradeSlot go = Instantiate(PlayerSlotPrefab, ParentObj.transform).GetComponent<PlayerTradeSlot>();
49. go.PlayerColor.GetComponent<Image>().color = UnityUI.textToColor(p.color);
50. **go.PlayerName.GetComponent<TextMeshProUGUI>().text = p.playerName;**
51. go.ButtonObj.GetComponent<Button>().onClick.AddListener(() => Selected(p));
52. **yield** **return** new WaitForSeconds(0.1f);
53. }
54. }
56. *// Called when a player is selected*
57. **void** Selected(Player pl)
58. {
59. AudioManager.i.Play("UIClick");
61. **if** (found)
62. **return**;
63. found = **true**;
64. *// call reference with this player!*
65. **UnityUI.Interface.SelectPartner(pl);**
66. CloseGUI();
67. }
68. }

#### PlayerNameInp.cs

1. **using** UnityEngine;
2. **using** TMPro;
3. **using** UnityEngine.UI;
5. ***/// <summary>***
6. */// <c>PlayerNameInp</c> is the class that manages the overlay for inputting the player's name and whether they are a bot.*
7. */// </summary>*
8. **public** **class** PlayerNameInp : MonoBehaviour
9. {
10. **public string FinalName;**
11. **public** **bool** IsBot = **false**;
12. **public** Toggle toggle;
13. **public** TMP\_InputField inputField;
14. **public** Button button;
16. [Header("Tweening Object")]
17. **public** GameObject panel;
18. **public** GameObject banner;
19. **public** GameObject input;
20. **public GameObject botImage;**
21. **public** GameObject checkbox;
22. **public** GameObject doneBox;
24. **private** **const** **int** NAME\_MAX\_LENGTH = 10;
25. **private const int NAME\_MIN\_LENGTH = 3;**
27. *// Start is called before the first frame update*
28. **void** Start()
29. {
30. **Canvas myCanvas = GetComponent<Canvas>();**
31. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
32. myCanvas.worldCamera = Camera.main;
33. myCanvas.sortingLayerName = "UI";
34. myCanvas.sortingOrder = 1;
35. ***// animate the GUI to appear***
36. panel.transform.localScale = new Vector3(0, 0, 0);
37. Vector3 bannerOG = banner.transform.localScale;
38. banner.transform.localScale = new Vector3(0, 0, 0);
39. Vector3 inputOG = input.transform.localScale;
40. **input.transform.localScale = new Vector3(0, 0, 0);**
41. Vector3 doneOG = doneBox.transform.localScale;
42. doneBox.transform.localScale = new Vector3(0, 0, 0);
43. Vector3 botOG = botImage.transform.localScale;
44. botImage.transform.localScale = new Vector3(0, 0, 0);
45. **Vector3 checkOG = checkbox.transform.localScale;**
46. checkbox.transform.localScale = new Vector3(0, 0, 0);
47. LeanTween.scale(panel, new Vector3(1, 1, 1), 1f).setEaseOutBack();
48. LeanTween.scale(banner, bannerOG, 1f).setEaseOutBack();
49. LeanTween.rotateAround(banner, Vector3.forward, 360, 0.5f).setEase(LeanTweenType.easeInSine);
50. **LeanTween.scale(input, inputOG, 1f).setEaseOutBack().setDelay(0.1f);**
51. LeanTween.scale(botImage, botOG, 1f).setEaseOutBack().setDelay(0.2f);
52. LeanTween.scale(checkbox, checkOG, 1f).setEaseOutBack().setDelay(0.3f);
53. LeanTween.scale(doneBox, doneOG, 1f).setEaseOutBack().setDelay(0.4f);
54. }
56. *// Update is called once per frame*
57. **void** Update()
58. {
59. **if** (Input.GetKeyDown(KeyCode.**Return**))
60. **{**
61. **if** (button.interactable)
62. {
63. button.onClick.Invoke();
65. **}**
66. }
67. }
69. *// Method to close the GUI*
70. **public void CloseGUI()**
71. {
72. *// animate the GUI to close*
73. LeanTween.moveY(banner, banner.transform.position.y + 300f, 0.75f).setEaseInBack();
74. LeanTween.scale(panel, new Vector3(), 1f).setEaseInBack().setOnComplete(() => { Destroy(gameObject); });
75. **LeanTween.scale(input, new Vector3(), 0.75f).setEaseInBack().setDelay(0.15f);**
76. LeanTween.scale(botImage, new Vector3(), 0.75f).setEaseInBack();
77. LeanTween.scale(checkbox, new Vector3(), 0.75f).setEaseInBack();
78. LeanTween.scale(doneBox, new Vector3(), 0.75f).setEaseInBack().setDelay(0.25f);
79. LeanTween.rotateAround(doneBox, Vector3.forward, 360, 1f).setEase(LeanTweenType.easeInSine).setDelay(0.25f);
80. **}**
82. *// Method to sanitize the player name. This is called every character change in the input field by Unity.*
83. **public** **void** TextChange()
84. {
85. **string name = inputField.text;**
86. button.interactable = **true**;
87. *// ensure the name is not too long or too short*
88. **if** (name.Length > NAME\_MAX\_LENGTH)
89. {
90. **name = name.Substring(0, NAME\_MAX\_LENGTH);**
91. inputField.text = name;
92. }
93. **else** **if** (name.Length < NAME\_MIN\_LENGTH)
94. {
95. **button.interactable = false;**
96. }
98. **string** output = "";
99. *// sanitize the input to alphanumeric*
100. **foreach (char x in name.ToCharArray())**
101. {
102. **if** (**char**.IsLetterOrDigit(x))
103. {
104. output += x;
105. **}**
106. }
107. inputField.text = output;
108. }
109. }

#### PlayerUIOverlay.cs

1. **using** System.Collections;
2. **using** UnityEngine;
3. **using** UnityEngine.UI;
4. **using** TMPro;
5. **using Unity.VisualScripting;**
7. */// <summary>*
8. */// <c>PlayerUIOverlay</c> is the class that manages the overlay UI for the player. This includes the buttons for game actions and pausing along with elements to display the player's name, color, score, and time left.*
9. */// </summary>*
10. **public class PlayerUIOverlay : MonoBehaviour**
11. {
12. **public** **bool** isAI = **false**;
13. **public** **bool** isZoomed = **true**;
14. **private** **float** animationTimer;
15. **private float zoomCD = 0.0f;**
16. **private** **float** moveCD = 0.0f;
17. **private** **float** turnEndCD = 5f;
18. **private** **bool** isTryingToMove = **false**;
19. **private** Sprite buffer;
21. [Header("UI Elements")]
22. **public** Button MoveInput;
23. **public** Button ShopInput;
24. **public** Button InventoryInput;
25. **public Button TradeInput;**
26. **public** Button EndTurnInput;
27. **public** Button PauseInput;
28. **public** Button ZoomInput;
29. **public** Button UndoInput;
30. **public TextMeshProUGUI TimerText;**
31. **public** TextMeshProUGUI PlayerName;
32. **public** TextMeshProUGUI PlayerScore;
33. **public** TextMeshProUGUI PlayerMoves;
34. **public** Image ColorMe;
35. **public Sprite CancelImage;**
37. *// Start is called before the first frame update*
38. **void** Start()
39. {
40. **Canvas myCanvas = GetComponent<Canvas>();**
41. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
42. myCanvas.worldCamera = Camera.main;
43. myCanvas.sortingLayerName = "UI";
44. myCanvas.sortingOrder = 200;
45. **ZoomInput.onClick.AddListener(ZoomButton);**
46. MoveInput.onClick.AddListener(MoveButton);
47. EndTurnInput.onClick.AddListener(EndTurnButton);
48. InventoryInput.onClick.AddListener(OnInventory);
49. TradeInput.onClick.AddListener(OnTrade);
50. **}**
52. *// Update is called once per frame*
53. **void** Update()
54. {
55. **TimerText.text = UnityUI.Interface.GetTime();**
56. **if** (!isAI)
57. {
58. PlayerMoves.text = UnityUI.**Interface**.game.GetCurrentPlayer().getMovesLeft().ToString();
59. PlayerScore.text = UnityUI.**Interface**.game.GetCurrentPlayer().getVictoryPoints().ToString();
60. **UndoInput.interactable = UnityUI.Interface.game.actions.Count > 0;**
61. }
62. animationTimer += Time.deltaTime;
63. zoomCD = Mathf.Clamp(zoomCD - Time.deltaTime, -5f, 5f);
64. moveCD = Mathf.Clamp(moveCD - Time.deltaTime, -3f, 3f);
65. **turnEndCD = Mathf.Clamp(turnEndCD - Time.deltaTime, -3f, 3f);**
66. }
68. *// called by the AI to set the UI to AI mode. this disables all buttons except for zooming and pausing*
69. **public** **void** SetAI()
70. **{**
71. isAI = **true**;
72. Destroy(MoveInput.gameObject);
73. InventoryInput.interactable = **false**;
74. Destroy(EndTurnInput.gameObject);
75. **Destroy(ShopInput.gameObject);**
76. Destroy(UndoInput.gameObject);
77. Destroy(TradeInput.gameObject);
78. }
80. ***// method to handle a click on the move button***
81. **public** **void** MoveButton()
82. {
83. AudioManager.i.Play("UIClick");
84. **if** (LeanTween.isTweening(MoveInput.gameObject))
85. **{**
86. LeanTween.scale(MoveInput.gameObject, MoveInput.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
87. }
88. **if** (moveCD > 0.0f)
89. {
90. **return;**
91. }
92. isTryingToMove = !isTryingToMove;
93. **if** (isTryingToMove)
94. {
95. **moveCD = 0.5f;**
96. UnityUI.**Interface**.OnPlayerMove();
97. buffer = MoveInput.image.sprite;
98. MoveInput.image.sprite = CancelImage;
99. AlterAllGameButton(**false**);
100. **MoveInput.gameObject.SetActive(true);**
101. PlayerMoves.gameObject.SetActive(**false**);
102. }
103. **else**
104. {
105. **FinishMove();**
106. }
107. }
109. *// method to handle a click on the end turn button*
110. **public void EndTurnButton()**
111. {
112. **if** (turnEndCD > 0.0f)
113. {
114. **return**;
115. **}**
116. turnEndCD = 5f;
117. AudioManager.i.Play("UIClick");
118. **if** (!EndTurnInput.gameObject.IsDestroyed())
119. {
120. **LeanTween.moveLocalY(EndTurnInput.gameObject, 10, 0.5f).setEase(LeanTweenType.easeInBack);**
121. LeanTween.scale(EndTurnInput.gameObject, new Vector3(0f, 0f, 0f), 0.5f).setEase(LeanTweenType.easeInOutBounce).setOnComplete(() => {
122. Destroy(FindObjectOfType<PlayerUIOverlay>().gameObject);
123. UnityUI.**Interface**.EndTurn();
124. });
125. **}**
126. }
128. *// method to handle finishing or cancelling a move*
129. **public** **void** FinishMove()
130. **{**
131. **if** (!isAI)
132. {
133. **if** (buffer != **null**)
134. {
135. **MoveInput.image.sprite = buffer;**
136. buffer = **null**;
137. }
138. UnityUI.**Interface**.StopAllPlayerCoroutines();
140. **AlterAllGameButton(true);**
141. PlayerMoves.gameObject.SetActive(**true**);
142. }
143. }
145. ***// method to alter the state of all game buttons (used to disable all buttons when a player is moving etc.)***
146. **public** **void** AlterAllGameButton( **bool** state, **bool** changeZoom = **false** ) *// only called by player movement and shopping*
147. {
148. MoveInput.gameObject.SetActive(state);
149. ShopInput.gameObject.SetActive(state);
150. **InventoryInput.interactable = state;**
151. TradeInput.gameObject.SetActive(state);
152. EndTurnInput.gameObject.SetActive(state);
153. UndoInput.gameObject.SetActive(state);
154. **if** (changeZoom)
155. **{**
156. ZoomInput.gameObject.SetActive(state);
157. }
158. }
160. ***// method to handle the zoom button click***
161. **public** **void** ZoomButton()
162. {
163. AudioManager.i.Play("UIClick");
164. **if** (!LeanTween.isTweening(ZoomInput.gameObject))
165. **{**
166. LeanTween.scale(ZoomInput.gameObject, ZoomInput.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
167. }
169. **if** (zoomCD > 0.0f)
170. **{**
171. **return**;
172. }
173. **float** inDist = 150.0f;
174. **float** outDist = 300.0f;
175. **isZoomed = !isZoomed;**
176. zoomCD = 0.85f;
177. animationTimer = 0.0f;
178. **if** (isZoomed)
179. {
180. **StartCoroutine(ZoomLerp(inDist));**
181. }
182. **else**
183. {
184. StartCoroutine(ZoomLerp(outDist));
185. **}**
186. }
188. *// method to smoothly lerp the camera zoom*
189. IEnumerator ZoomLerp(**float** endDist)
190. **{**
191. **float** totalTime = 0.5f;
192. **float** startDist = Camera.main.orthographicSize;
193. **float** t = 0.0f;
194. **while** (animationTimer < totalTime)
195. **{**
196. t = animationTimer / totalTime;
197. Camera.main.orthographicSize = Mathf.Lerp(startDist, endDist, t);
198. **yield** **return** 0;
199. }
200. **}**
202. *// method to handle the inventory button click*
203. **public** **void** OnInventory()
204. {
205. **SceneTransition.i.PlayAnimation();**
206. **if** (FindObjectOfType<InventoryPopup>() != **null**)
207. {
208. **return**;
209. }
210. **AudioManager.i.Play("UIClick");**
211. LeanTween.scale(InventoryInput.gameObject, InventoryInput.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
212. UnityUI.**Interface**.OpenInventory();
213. }
215. ***// method to handle the trade button click***
216. **public** **void** OnTrade()
217. {
218. **if** (FindObjectOfType<TradingInterface>() != **null**) *// might need to be changed to a bool*
219. {
220. **return;**
221. }
222. AudioManager.i.Play("UIClick");
223. **if** (!LeanTween.isTweening(TradeInput.gameObject))
224. {
225. **LeanTween.scale(TradeInput.gameObject, TradeInput.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);**
226. }
228. StartCoroutine(UnityUI.**Interface**.OpenTrade());
229. }
230. **}**

#### PopupController.cs

1. **using** UnityEngine;
2. **using** TMPro;
3. **using** UnityEngine.UI;
5. ***/// <summary>***
6. */// <c>PopupController</c> is the class that manages the overlay to display a message to the user.*
7. */// </summary>*
8. **public** **class** PopupController : MonoBehaviour
9. {
10. ***// Start is called before the first frame update***
11. **public** GameObject MainText;
12. **public** GameObject CancelBtn;
13. **public** GameObject TextBG;
14. **public** GameObject Header;
16. *// Start is called before the first frame update*
17. **void** Start()
18. {
19. Canvas myCanvas = GetComponent<Canvas>();
20. **myCanvas.renderMode = RenderMode.ScreenSpaceCamera;**
21. myCanvas.worldCamera = Camera.main;
22. myCanvas.sortingLayerName = "UI";
23. myCanvas.sortingOrder = 900;
24. CancelBtn.GetComponent<Button>().onClick.AddListener(CloseGUI);
25. ***// do some animation for the popup to appear***
26. Vector3 init;
27. init = TextBG.transform.localScale;
28. TextBG.transform.localScale = new Vector3(0, 0, 0);
29. LeanTween.scale(TextBG, init, 0.5f).setEase(LeanTweenType.easeOutSine).setDelay(0.15f);
30. **init = Header.transform.localScale;**
31. Header.transform.localScale = new Vector3(0, 0, 0);
32. LeanTween.scale(Header, init, 0.5f).setEase(LeanTweenType.easeOutSine);
33. init = CancelBtn.transform.localScale;
34. CancelBtn.transform.localScale = new Vector3(0, 0, 0);
35. **LeanTween.scale(CancelBtn, init, 0.5f).setEase(LeanTweenType.easeOutSine);**
36. LeanTween.rotateAround(CancelBtn, Vector3.forward, 360f, 0.5f).setEase(LeanTweenType.easeOutSine);
37. LeanTween.rotateAround(Header, Vector3.forward, 360f, 0.5f).setEase(LeanTweenType.easeOutSine);
39. }
41. *// method to close the popup*
42. **public** **void** CloseGUI()
43. {
44. *// do some tweening*
45. **AudioManager.i.Play("UIClick");**
46. AudioManager.i.Stop("Write");
47. LeanTween.scale(Header, new Vector3(0, 0, 0), 0.75f).setEase(LeanTweenType.easeInElastic);
48. LeanTween.scale(CancelBtn, new Vector3(0, 0, 0), 0.5f).setEase(LeanTweenType.easeInElastic);
49. LeanTween.rotateAround(TextBG, Vector3.forward, 360f, 0.5f).setEase(LeanTweenType.easeInElastic);
50. **LeanTween.scale(TextBG, new Vector3(0, 0, 0), 0.75f).setEase(LeanTweenType.easeInElastic).setDelay(0.15f).setOnComplete(() => Destroy(gameObject));**
51. }
53. *// method to update the text of the popup*
54. **public** **void** Setup(**string** content)
55. **{**
56. MainText.GetComponent<TextMeshProUGUI>().text = content;
57. }
58. }

#### SaveSelector.cs

1. **using** NEAGame;
2. **using** UnityEngine;
3. **using** TMPro;
4. **using** UnityEngine.UI;
6. */// <summary>*
7. */// <c>SaveSelector</c> is the class that manages the game save selection overlay. It is used to display the save game slots for the user to load or create.*
8. */// </summary>*
9. **public** **class** SaveSelector : MonoBehaviour
10. **{**
12. **public** **string** SaveFileName;
13. **public** SaveSlot[] SaveSlots;

16. [Space(10)]
17. [Header("Animation objs")]
18. **public** GameObject panel;
19. **public** GameObject cancelBtn;
20. **public GameObject SaveBtnParent;**
22. *// Start is called before the first frame update*
23. **void** Start()
24. {
25. **Canvas myCanvas = GetComponent<Canvas>();**
26. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
27. myCanvas.worldCamera = Camera.main;
28. myCanvas.sortingLayerName = "UI";
29. myCanvas.sortingOrder = 800;
30. ***// add listener for close button click***
31. cancelBtn.GetComponent<Button>().onClick.AddListener(() => {
32. AudioManager.i.Play("UIClick");
33. **if** (!LeanTween.isTweening(cancelBtn))
34. {
35. **LeanTween.scale(cancelBtn, cancelBtn.transform.localScale, 0.1f).setEase(LeanTweenType.easeInCubic).setLoopPingPong(1);**
36. }
37. CloseGUI();
38. });
39. }
41. *// method to close the save selector*
42. **public** **void** CloseGUI()
43. {
44. AudioManager.i.Play("UIClick");
46. LeanTween.scale(cancelBtn, Vector3.zero, 0.3f).setEase(LeanTweenType.easeInCubic);
47. LeanTween.scale(SaveBtnParent, Vector3.zero, 0.5f).setEase(LeanTweenType.easeInCubic);
48. LeanTween.rotateAround(SaveBtnParent, Vector3.forward, 180, 0.25f).setEase(LeanTweenType.easeInCubic).setDelay(0.15f);
49. LeanTween.scale(panel, Vector3.zero, 0.6f).setEase(LeanTweenType.easeInCubic).setOnComplete(() => { Destroy(gameObject); });
51. }
53. *// method to setup the save selector*
54. **public** **void** Setup()
55. **{**
56. panel.SetActive(**true**);
57. cancelBtn.SetActive(**true**);
58. **for** (**int** i = 0; i < SaveSlots.Length; i++)
59. {
60. **JSON\_manager json = new JSON\_manager(i);**
61. **bool** hasData = json.DoesGameExist();
62. SaveSlot slot = SaveSlots[i];
63. slot.ID = i;
64. **if** (hasData)
65. **{**
66. *// allows the user to load the save*
67. slot.StatusText.GetComponent<TextMeshProUGUI>().text = "Load Save";
68. slot.SlotText.GetComponent<TextMeshProUGUI>().text = "Save " + (i + 1);
69. }
70. **else**
71. {
72. *// allows the user to create a new save*
73. slot.StatusText.GetComponent<TextMeshProUGUI>().text = "Create New";
74. slot.SlotText.GetComponent<TextMeshProUGUI>().text = "Save " + (i + 1);
75. **slot.Reset.SetActive(false);**
76. }
77. slot.Background.GetComponent<Button>().onClick.RemoveAllListeners();
78. slot.Reset.GetComponent<Button>().onClick.RemoveAllListeners();
79. slot.Background.GetComponent<Button>().onClick.AddListener(() => BtnClick(slot.ID, hasData));
80. **slot.Reset.GetComponent<Button>().onClick.AddListener(() => ResetClick(slot.ID));**
81. }
82. }
84. *// onclick for the reset button*
85. **void ResetClick(int i)**
86. {
87. AudioManager.i.Play("UIClick");
88. JSON\_manager json = new JSON\_manager(i);
89. json.ClearSave();
90. **Setup();**
91. }
93. *// onclick for the save slots*
94. **void** BtnClick(**int** i, **bool** hasData)
95. **{**
96. **if** (hasData)
97. {
98. UnityUI.**Interface**.SelectGameToLoad(i);
99. }
100. **else**
101. {
102. UnityUI.**Interface**.CreateNewGame(i);
103. }
104. CloseGUI();
105. **}**
106. }

#### ShopOverlay.cs

1. **using** UnityEngine;
2. **using** TMPro;
3. **using** UnityEngine.UI;
4. **using** NEAGame;
6. */// <summary>*
7. */// <c>ShopOverlay</c> is the class that manages the shop overlay. It is used to display the shop items for the user to purchase.*
8. */// </summary>*
9. **public** **class** ShopOverlay : MonoBehaviour
10. **{**
11. *// Start is called before the first frame update*
12. **int** currentID = 0;
14. **string**[] shoppingOrder = new **string**[] { "Road", "Village", "Wall", "City" };
15. **public Sprite[] shoopingImages = new Sprite[4];**
16. **public** GameObject[] texts = new GameObject[5];
17. *// same order as the resource class. for referece this is: brick, sheep, ore, wood, wheat*
19. [Space(10)]
20. **[Header("Animation objects")]**
21. **public** Button purchase;
22. **public** GameObject LeftBtn;
23. **public** GameObject RightBtn;
25. **private float Buffer = 0.0f;**
27. *// Start is called before the first frame update*
28. **void** Start()
29. {
30. **Canvas myCanvas = GetComponent<Canvas>();**
31. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
32. myCanvas.worldCamera = Camera.main;
33. myCanvas.sortingLayerName = "UI";
34. myCanvas.sortingOrder = 700;
35. **purchase.onClick.AddListener(() => OnPurchase());**
36. purchase.gameObject.GetComponent<Image>().sprite = shoopingImages[currentID];
37. UpdateDisplayCounts();
38. AudioManager.i.Play("Purchase");
39. }
41. *// Update is called once per frame*
42. **void** Update()
43. {
44. **if** (purchase.interactable)
45. **{**
46. **if** (Input.GetKeyDown(KeyCode.LeftArrow))
47. {
48. OnCarousel(**false**);
49. }
50. **else if (Input.GetKeyDown(KeyCode.RightArrow))**
51. {
52. OnCarousel(**true**);
53. }
54. }
55. **if (Input.GetKeyDown(KeyCode.Escape))**
56. {
57. CloseGUI();
58. }
59. **if** (Input.GetKeyDown(KeyCode.**Return**))
60. **{**
61. OnPurchase();
62. }
63. }
65. ***// FixedUpdate is called once per frame***
66. **private** **void** FixedUpdate()
67. {
68. Buffer = Mathf.Max(0, Buffer-Time.deltaTime);
69. }
71. *// Attempt to purchase the item*
72. **void** OnPurchase()
73. {
74. UnityUI.**Interface**.AttemptPurchase(shoppingOrder[currentID]);
75. **CloseGUI();**
76. }
78. *// Close the GUI*
79. **public** **void** CloseGUI()
80. **{**
81. LeanTween.scale(purchase.gameObject, new Vector3(0, 0, 0), 0.75f).setEase(LeanTweenType.easeOutCirc).setOnComplete(() => Destroy(gameObject));
82. }
84. *// Moves to the next item in the carousel*
85. **public void OnCarousel(bool right)**
86. {
87. **if** (Buffer > 0)
88. {
89. **return**;
90. **}**
91. Buffer = 1.2f;
92. purchase.interactable = **false**;
93. AudioManager.i.Play("UIClick");
94. GameObject btnToTween;
95. **if (right)**
96. {
97. btnToTween = RightBtn;
98. currentID++;
99. }
100. **else**
101. {
102. btnToTween = LeftBtn;
103. currentID--;
104. }
105. **if (!LeanTween.isTweening(btnToTween))**
106. {
107. LeanTween.scale(btnToTween, btnToTween.transform.localScale \* 1.2f, 0.1f).setEase(LeanTweenType.easeOutCirc).setLoopPingPong(1);
108. }
109. currentID %= 4; *// wrap around the current ID*
111. Vector3 startpos = purchase.gameObject.transform.position;
112. **int** dir = right ? 1 : -1;
113. *// animate the purchase image across the screen and update the costs.*
114. LeanTween.moveLocalX(purchase.gameObject, startpos.x + (dir\*1000f), 0.5f).setDelay(0.1f).setEase(LeanTweenType.easeInCirc).setOnComplete(() =>
115. **{**
116. purchase.gameObject.transform.position = new Vector3(- (dir\*2000), 0, 0) + startpos;
117. purchase.gameObject.GetComponent<Image>().sprite = shoopingImages[currentID];
118. UpdateDisplayCounts();
119. LeanTween.move(purchase.gameObject, startpos, 0.5f).setEase(LeanTweenType.easeOutCirc);
120. **});**
121. }
123. *// Update the display counts*
124. **void** UpdateDisplayCounts()
125. **{**
126. **bool** canBuy = **true**;
127. **foreach** (**var** entry **in** UnityUI.**Interface**.game.GetDifference(shoppingOrder[currentID]))
128. {
130. **int index = entry.Key.GetHashCode() - 1;**
131. **if** (entry.**Value** >= 0)
132. {
133. texts[index].GetComponentInChildren<TextMeshProUGUI>().color = Color.white;
134. texts[index].GetComponentInChildren<TextMeshProUGUI>().text
135. **= TravelersOfCatan.GetCostOfUpgrade(shoppingOrder[currentID])[entry.Key].ToString();**
136. }
137. **else**
138. {
139. canBuy = **false**;
140. **texts[index].GetComponentInChildren<TextMeshProUGUI>().color = Color.red;**
141. texts[index].GetComponentInChildren<TextMeshProUGUI>().text = entry.**Value**.ToString().Trim('-');
142. }
143. }
145. **if (canBuy)**
146. {
147. purchase.interactable = **true**;
148. }
149. **else**
150. **{**
151. purchase.interactable = **false**;
152. }
153. }
154. }

#### TradingInterface.cs

1. **using** System.Collections.Generic;
2. **using** UnityEngine;
3. **using** NEAGame;
4. **using** UnityEngine.UI;
5. **using TMPro;**
7. */// <summary>*
8. */// <c>TradingInterface</c> is the class that manages the trading overlay. It is used to display each resource along with the quantity to trade.*
9. */// </summary>*
10. **public class TradingInterface : MonoBehaviour**
11. {
12. **public** Button accept;
13. **public** Button cancel;
14. **public** Button left;
15. **public Button right;**
16. **public** Sprite[] Cards;
18. [Header("Players")]
19. **public** GameObject PName1;
20. **public GameObject PName2;**
21. **public** GameObject PCol1;
22. **public** GameObject PCol2;
23. **public** Button increase;
24. **public** Button decrease;
25. **public GameObject PTrade1;**
26. **public** GameObject PTrade2;
27. **public** GameObject Card;
28. **public** GameObject POverallVal1;
29. **public** GameObject POverallVal2;
31. [Header("Animation")]
32. **public** GameObject PanelText;
34. *//Dictionaries used to store the current trades and the max gain/loss for each resource so player's can't trade more than they have*
35. **Dictionary<int, int> MaxGain = new Dictionary<int, int>();**
36. Dictionary<**int**, **int**> MaxLoss = new Dictionary<**int**, **int**>();
37. Dictionary<**int**, **int**> CurrentTrades = new Dictionary<**int**, **int**>();
39. **int** currentResource = 0;
40. **int overallVal = 0;**
41. Player other;
43. *// Start is called before the first frame update*
44. **void** Start()
45. **{**
46. Canvas myCanvas = GetComponent<Canvas>();
47. myCanvas.renderMode = RenderMode.ScreenSpaceCamera;
48. myCanvas.worldCamera = Camera.main;
49. myCanvas.sortingLayerName = "UI";
50. **myCanvas.sortingOrder = 250;**
51. cancel.onClick.AddListener(() =>
52. {
53. AudioManager.i.Play("UIClick");
54. CloseGUI();
55. **});**
56. accept.onClick.AddListener(AcceptTrade);
57. accept.interactable = **false**;
58. left.onClick.AddListener(Left);
59. right.onClick.AddListener(Right);
60. **increase.onClick.AddListener(AddToCurrent);**
61. decrease.onClick.AddListener(GiveToEnemy);
63. }
65. ***// method to close the trading interface***
66. **public** **void** CloseGUI()
67. {
68. *// tween everything out of existance*
69. Destroy(POverallVal1);
70. **Destroy(POverallVal2);**
71. LeanTween.scale(PName2.transform.parent.gameObject, new Vector3(0, 0, 0), 0.5f).setEase(LeanTweenType.easeInOutElastic);
72. LeanTween.scale(PName1.transform.parent.gameObject, new Vector3(0, 0, 0), 0.5f).setEase(LeanTweenType.easeInOutElastic);
73. LeanTween.scale(accept.gameObject, new Vector3(0, 0, 0), 0.5f).setEase(LeanTweenType.easeInOutElastic);
74. LeanTween.scale(cancel.gameObject, new Vector3(0, 0, 0), 0.5f).setEase(LeanTweenType.easeInOutElastic);
75. **LeanTween.moveLocalY(cancel.gameObject, -100, 0.2f).setEase(LeanTweenType.easeOutQuint);**
76. *//LeanTween.rotateAround()*
77. LeanTween.moveLocalX(left.gameObject, -600f, 0.5f).setEase(LeanTweenType.easeOutBack);
78. LeanTween.moveLocalX(right.gameObject, 600f, 0.5f).setEase(LeanTweenType.easeOutBack);
79. LeanTween.alphaCanvas(PanelText.GetComponent<CanvasGroup>(), 0f, 0.5f).setEase(LeanTweenType.easeOutBack).setDelay(1f).setOnComplete(() => Destroy(gameObject));
80. **}**
82. *// method to setup the trade interface*
83. **public** **void** Setup(Player current, Player other)
84. {
85. **this.other = other;**
86. PName1.GetComponent<TextMeshProUGUI>().text = current.playerName;
87. PName1.SetActive(**true**);
88. PName2.GetComponent<TextMeshProUGUI>().text = other.playerName;
89. PName2.SetActive(**true**);
90. **PCol1.GetComponent<Image>().color = UnityUI.textToColor(current.color.ToString());**
91. PCol2.GetComponent<Image>().color = UnityUI.textToColor(other.color.ToString());
92. **foreach** (**var** entry **in** current.getResources())
93. {
94. CurrentTrades.**Add**(entry.Key.GetHashCode()-1, 0);
95. **MaxLoss.Add(entry.Key.GetHashCode()-1, entry.Value);**
96. }
97. **foreach** (**var** entry **in** other.getResources())
98. {
99. MaxGain.**Add**(entry.Key.GetHashCode()-1, entry.**Value**);
100. **}**
102. *// set Ptrade texts*
103. UpdateGUI();
104. left.interactable = **false**;
106. }
108. *// method to update all of the trade quantity information*
109. **void** UpdateGUI()
110. **{**
111. *// tween the card around*
112. Card.GetComponent<Image>().sprite = Cards[currentResource];
113. **if** (CurrentTrades[currentResource] > 0)
114. {
115. ***// if the current player is gaining this resource***
116. PTrade1.GetComponent<TextMeshProUGUI>().text = "+" + CurrentTrades[currentResource].ToString();
117. PTrade2.GetComponent<TextMeshProUGUI>().text = "-" + CurrentTrades[currentResource].ToString();
118. }
119. **else** **if** (CurrentTrades[currentResource] < 0)
120. **{**
121. *// if the current player is giving this resource*
122. PTrade2.GetComponent<TextMeshProUGUI>().text = "+" + Mathf.Abs(CurrentTrades[currentResource]).ToString();
123. PTrade1.GetComponent<TextMeshProUGUI>().text = CurrentTrades[currentResource].ToString();
124. }
125. **else**
126. {
127. *// if the current player is not trading this resource*
128. PTrade1.GetComponent<TextMeshProUGUI>().text = "0";
129. PTrade2.GetComponent<TextMeshProUGUI>().text = "0";
130. **}**
132. **if** (overallVal > 0)
133. {
134. *// if the current player is getting more than they are giving*
135. **POverallVal1.GetComponent<TextMeshProUGUI>().text = "+" + overallVal.ToString();**
136. POverallVal1.GetComponent<TextMeshProUGUI>().color = Color.green;
137. POverallVal2.GetComponent<TextMeshProUGUI>().text = "-" + overallVal.ToString();
138. POverallVal2.GetComponent<TextMeshProUGUI>().color = Color.red;
139. }
140. **else if (overallVal < 0)**
141. {
142. *// if the current player is giving more than they are getting*
143. POverallVal2.GetComponent<TextMeshProUGUI>().text = "+" + Mathf.Abs(overallVal).ToString();
144. POverallVal2.GetComponent<TextMeshProUGUI>().color = Color.green;
145. **POverallVal1.GetComponent<TextMeshProUGUI>().text = overallVal.ToString();**
146. POverallVal1.GetComponent<TextMeshProUGUI>().color = Color.red;
147. }
148. **else**
149. {
150. ***// if the trade is even***
151. POverallVal1.GetComponent<TextMeshProUGUI>().text = "0";
152. POverallVal1.GetComponent<TextMeshProUGUI>().color = Color.white;
153. POverallVal2.GetComponent<TextMeshProUGUI>().text = "0";
154. POverallVal2.GetComponent<TextMeshProUGUI>().color = Color.white;
155. **}**
156. decrease.interactable = (-CurrentTrades[currentResource] < MaxLoss[currentResource]);
157. increase.interactable = (CurrentTrades[currentResource] < MaxGain[currentResource]);
158. }

161. *// onclick on the add for current player*
162. **void** AddToCurrent()
163. {
164. AudioManager.i.Play("UIClick");
165. **if (!LeanTween.isTweening(increase.gameObject))**
166. {
167. LeanTween.scale(increase.gameObject, increase.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
168. }
169. overallVal++;
170. **CurrentTrades[currentResource] ++ ;**
171. UpdateGUI();
173. }
175. ***// onclick on the add for the other player***
176. **void** GiveToEnemy()
177. {
178. AudioManager.i.Play("UIClick");
179. **if** (!LeanTween.isTweening(decrease.gameObject))
180. **{**
181. LeanTween.scale(decrease.gameObject, decrease.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
182. }
183. overallVal--;
184. CurrentTrades[currentResource] -- ;
185. **UpdateGUI();**
186. }
188. *// rotate carousel to the left*
189. **void** Left()
190. **{**
191. accept.interactable = **false**;
192. right.interactable = **true**;
193. AudioManager.i.Play("UIClick");
194. **if** (!LeanTween.isTweening(left.gameObject))
195. **{**
196. LeanTween.scale(left.gameObject, left.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
197. }
198. currentResource--; **if** (currentResource == 0) { left.interactable = **false**; }
199. UpdateGUI();
200. **}**
202. *// rotate carousel to the right*
203. **void** Right()
204. {
205. **left.interactable = true;**
206. AudioManager.i.Play("UIClick");
207. **if** (!LeanTween.isTweening(right.gameObject))
208. {
209. LeanTween.scale(right.gameObject, right.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
210. **}**
211. currentResource++; **if** (currentResource == 4) { right.interactable = **false**; accept.interactable = **true**; }
212. UpdateGUI();
213. }
215. ***// onclick on the accept button***
216. **void** AcceptTrade()
217. {
218. AudioManager.i.Play("UIClick");
219. **if** (!LeanTween.isTweening(accept.gameObject))
220. **{**
221. LeanTween.scale(accept.gameObject, accept.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.0f).setLoopPingPong(1);
222. }
223. StartCoroutine(UnityUI.**Interface**.RegisterTrade(CurrentTrades, other));
224. CloseGUI();
225. **}**
226. }

#### AudioManager.cs

1. **using** System;
2. **using** UnityEngine;
4. */// <summary>*
5. ***/// <c>AudioManager</c> is the class that controls all audio for the game.***
6. */// Credit to Brackeys for the tutorial on this script: <seealso href="https://www.youtube.com/watch?v=6OT43pvUyfY"/>*
7. */// </summary>*
8. **public** **class** AudioManager : MonoBehaviour
9. {
10. ***// Singleton instance of the AudioManager class so that it can be accessed from all other scripts***
11. **public** **static** AudioManager i;
12. **public** Sound[] sounds;
13. **public** Sound[] BackgroundMusic;
14. [Range(0f, 1f)] **public** **float** VolumeModifier = 1f;
15. **private bool mutedSFX = false;**
16. **private** **bool** mutedBG = **false**;
18. *// Awake is called before Start*
19. **void** Awake()
20. **{**
22. **if** (i == **null**)
23. {
24. i = **this**;
25. **}**
26. **else**
27. {
28. Destroy(gameObject);
29. **return**;
30. **}**
31. DontDestroyOnLoad(gameObject);
33. *// load all sound effects*
34. **foreach** (Sound s **in** sounds)
35. **{**
36. s.source = gameObject.AddComponent<AudioSource>();
37. s.source.clip = s.Clip;
38. s.source.volume = s.Vol \* VolumeModifier;
39. s.source.pitch = s.Pitch;
40. **s.source.loop = s.Loop;**
41. }
42. *// load all background music*
43. **foreach** (Sound s **in** BackgroundMusic)
44. {
45. **s.source = gameObject.AddComponent<AudioSource>();**
46. s.source.clip = s.Clip;
47. s.source.volume = s.Vol \* VolumeModifier;
48. s.source.pitch = s.Pitch;
49. s.source.loop = s.Loop;
50. **}**
51. }
53. *// Start is called before the first frame update*
54. **private** **void** Start()
55. **{**
56. Sound s = BackgroundMusic[0];
57. s.source.Play();
58. }
60. ***// Update is called once per frame***
61. **void** Update()
62. {
63. AudioListener.volume = VolumeModifier;
64. }
66. *// Plays the sound with the given name*
67. **public** **void** Play(**string** name)
68. {
69. Sound s = Array.Find(sounds, sound => sound.name == name);
70. **if (s == null)**
71. {
72. Debug.Log(name + "not found");
73. **return**;
74. }
75. **else if (mutedSFX)**
76. {
77. **return**;
78. }
79. s.source.Play();
80. **}**
82. *// Stops the sound with the given name*
83. **public** **void** Stop(**string** name)
84. {
85. **Sound s = Array.Find(sounds, sound => sound.name == name);**
86. **if** (s == **null**)
87. {
88. **return**;
89. }
90. **s.source.Stop();**
91. }
93. *// Stops all sounds*
94. **public** **void** StopAll()
95. **{**
96. **foreach** (Sound s **in** sounds)
97. {
98. **try**
99. {
100. **s.source.Stop();**
101. }
102. **catch** {;}
103. }
104. }
106. *// Adjusts the game volume*
107. **public** **void** ChangeMasterVolume(**float** vol)
108. {
109. VolumeModifier = vol;
110. **}**
112. *// Toggles the mute for SFX or background music depending on the bool parameter*
113. **public** **void** ToggleMute(**bool** Background)
114. {
115. **if (Background)**
116. {
117. mutedBG = !mutedBG;
118. Sound s = BackgroundMusic[0];
119. **if** (mutedBG)
120. **{**
121. s.source.Stop();
122. }
123. **else**
124. {
125. **s.source.Play();**
126. }
127. }
128. **else**
129. {
130. **mutedSFX = !mutedSFX;**
131. **if** (mutedSFX)
132. {
133. StopAll();
134. }
135. **}**
136. }
138. *// Returns whether the game is muted or not*
139. **public** **bool** isMuted(**bool** Background)
140. **{**
141. **if** (Background)
142. {
143. **return** mutedBG;
144. }
145. **else**
146. {
147. **return** mutedSFX;
148. }
149. }
150. **}**
152. *// Class that holds the information for a sound*
153. [System.Serializable]
154. **public** **class** Sound
155. **{**
157. **public** **string** name;
158. **public** AudioClip Clip;
160. **[Range(0f, 1f)]**
161. **public** **float** Vol = 1f;
163. [Range(1f, 3f)]
164. **public** **float** Pitch = 1f;
166. **public** **bool** Loop = **false**;
168. [HideInInspector]
169. **public** AudioSource source;
171. }

#### CardCollection.cs

1. **using** System.Collections.Generic;
2. **using** UnityEngine;
4. */// <summary>*
5. ***/// <c>CardCollection</c> animates a card on the screen to indicate the collection or trading of a resource.***
6. */// </summary>*
7. **public** **class** CardCollection : MonoBehaviour
8. {
10. **public List<Sprite> cards = new List<Sprite>();**
12. *// Method to set the card to a specific resource and move it to a location on screen as an animation for collection and trading*
13. **public** **void** SetCard(**int** resource, Vector3 Dest)
14. {
15. **GetComponent<SpriteRenderer>().sprite = cards[resource];**
16. *// Animate the card to its destination. This is either the inventory button or the trading partner*
17. **float** delay = Random.Range(0.5f, 1f);
18. **float** duration = Random.Range(0.7f, 1.1f);
19. LeanTween.move(gameObject, Dest, duration).setEase(LeanTweenType.easeOutBounce).setDelay(delay).setOnComplete(() =>
20. **{**
21. AudioManager.i.Play("Ding");
22. LeanTween.scale(gameObject, Vector3.zero, 0.5f).setEase(LeanTweenType.easeInBounce).setDelay(0.15f).setOnComplete(() =>
23. {
24. Destroy(gameObject);
25. **});**
26. });
27. }
28. }

#### ConnectionAnimator.cs

1. **using** UnityEngine;
2. **using** NEAGame;
4. */// <summary>*
5. ***/// <c>ConnectionAnimator</c> is a class attached to the connection game object to control the connection between nodes.***
6. */// </summary>*
7. **public** **class** ConnectionAnimator : MonoBehaviour
8. {
9. **public** Connection connection;
10. **public GameObject Road;**
11. **public** GameObject Wall;
13. *// Use this for initialization*
14. **public** **void** UpdateConnection(Vector3 n1, Vector3 n2)
15. **{**
17. *// Animate images based on the direction of this connection so that they are always facing the right way*
19. **if** (n1.y == n2.y && n1.z == n2.z)
20. **{**
22. }
23. **else**
24. {
25. **if (n1.x == n2.x && n1.y == n2.y)**
26. {
27. Wall.GetComponent<SpriteRenderer>().flipX = **true**;
28. Road.transform.Rotate(0, 0, 60);
29. }
30. **else**
31. {
32. Road.transform.Rotate(0, 0, -60);
33. }
34. }
35. **}**
37. *// Method to update the display of the connection*
38. **public** **void** UpdateDisplay()
39. {
40. **switch (connection.GetStatus())**
41. {
42. **case** "Road":
43. Road.SetActive(**true**);
44. Wall.SetActive(**false**);
45. **break;**
46. **case** "Wall":
47. Road.SetActive(**false**);
48. Wall.SetActive(**true**);
49. **break**;
50. **default:**
51. LeanTween.scale(gameObject, new Vector3(0, 0, 0), 0.5f).setEase(LeanTweenType.easeOutBack).setOnComplete(() => { Destroy(gameObject); });
52. **break**;
53. }
54. }
55. **}**

#### MenuPage.cs

1. **using** UnityEngine;
2. **using** UnityEngine.UI;
4. */// <summary>*
5. ***/// <c>MenuPage</c> is the class that adds listeners to the buttons on the menu page.***
6. */// </summary>*
7. **public** **class** MenuPage : MonoBehaviour
8. {
10. **public Button about;**
11. **public** Button load;
12. **public** Button close;
13. **public** Button settings;
15. ***// Start is called before the first frame update***
16. **void** Start()
17. {
18. *// Add listeners and sound effects to the buttons on the menu page*
19. about.onClick.AddListener(() => {
20. **AudioManager.i.Play("UIClick");**
21. UnityUI.**Interface**.AboutButton();
22. });
23. load.onClick.AddListener(() => {
24. AudioManager.i.Play("UIClick");
25. **UnityUI.Interface.LoadGameButton();**
26. });
27. close.onClick.AddListener(() => {
28. AudioManager.i.Play("UIClick");
29. UnityUI.**Interface**.QuitButton();
30. **});**
31. settings.onClick.AddListener(() =>
32. {
33. AudioManager.i.Play("UIClick");
34. UnityUI.**Interface**.HomeScreenSettingsButton();
35. **});**
36. }
37. }

#### NodeButton.cs

1. **using** UnityEngine;
2. **using** UnityEngine.UI;
3. **using** NEAGame;
5. ***/// <summary>***
6. */// <c>NodeButton</c> is the class that controls the node buttons on the game board. This also controls the settlement display on the nodes.*
7. */// </summary>*
8. **public** **class** NodeButton : MonoBehaviour
9. {
10. **public Vector3 NodePos;**
11. **public** Node node;
12. **public** GameObject flag;
13. **public** Button btn;
14. **public** SpriteRenderer img;
15. **[InspectorName("Village")]**
16. **public** Sprite village;
17. [InspectorName("City")]
18. **public** Sprite city;

21. *// Start is called before the first frame update*
22. **void** Awake()
23. {
24. img.enabled = **false**;
25. **btn.onClick.AddListener(OnClick);**
26. btn.gameObject.SetActive(**false**);
27. }
29. *// method to enable the current node button*
30. **public void EnableButton()**
31. {
32. btn.gameObject.SetActive(**true**);
33. btn.transform.localScale = new Vector3(0, 0, 0);
34. LeanTween.scale(btn.gameObject, new Vector3(1, 1, 1), 0.75f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.1f);
35. **btn.interactable = true;**
36. }
38. *// method to disable the current node button*
39. **public** **void** DisableButton()
40. **{**
41. btn.interactable = **false**;
42. btn.gameObject.SetActive(**false**);
43. }
45. ***// method to handle the click event on the node button***
46. **public** **void** OnClick()
47. {
48. UnityUI.**Interface**.OnNodeClick(**this**);
49. }
51. *// method to update the settlement beneath the node button*
52. **public** **void** UpdateSettlement()
53. {
55. **var sc = img.transform.localScale;**
56. img.enabled = **true**;
57. flag.SetActive(**true**);
58. **if** (node.status.GetStatus() == "Village")
59. {
60. **flag.GetComponent<SpriteRenderer>().color = UnityUI.Interface.GetPlayerColor(node.status.GetOccupant());**
61. img.sprite = village;
62. }
63. **else** **if** (node.status.GetStatus() == "City")
64. {
65. **flag.GetComponent<SpriteRenderer>().color = UnityUI.Interface.GetPlayerColor(node.status.GetOccupant());**
66. img.sprite = city;
67. }
68. **else**
69. {
70. **LeanTween.scale(img.gameObject, new Vector3(0, 0, 0), 0.75f).setEase(LeanTweenType.easeInOutElastic).setOnComplete(() =>**
71. {
72. flag.SetActive(**false**);
73. img.enabled = **false**;
74. });
75. **}**
76. img.transform.localScale = new Vector3(0, 0, 0);
77. LeanTween.scale(img.gameObject, sc, 0.75f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.5f);
78. }
79. }

#### PlayerAnimation.cs

1. **using** NEAGame;
2. **using** UnityEngine;

5. ***/// <summary>***
6. */// <c>PlayerAnimator</c> is the class that animates the player game object. This includes moving the player and bobbing the current player indicator.*
7. */// </summary>*
8. **public** **class** PlayerAnimator : MonoBehaviour
9. {
11. **public** Player player;
12. **public** **bool** isCurrentPlayer = **false**;
13. **public** GameObject indicator;
15. **Vector3 indpos;**
17. *// Start is called before the first frame update*
18. **void** Start()
19. {
20. **indpos = indicator.transform.localPosition;**
21. }
23. *// Update is called once per frame*
24. **void** Update()
25. **{**
26. **if** (isCurrentPlayer && !LeanTween.isTweening(indicator))
27. {
28. *// bob the indicator up and down using LeanTween*
29. LeanTween.moveLocalY(indicator, indpos.y - 0.15f, 1.5f).setEaseInOutSine().setLoopPingPong();
30. **}**
31. **else** **if** (!isCurrentPlayer)
32. {
33. *// stop bobbing the indicator*
34. LeanTween.cancel(indicator);
35. **indicator.transform.localPosition = indpos;**
36. }
37. }
38. }

#### PlayerSlot.cs

1. **using** UnityEngine;
3. */// <summary>*
4. */// <c>PlayerSlot</c> is the class that stores references to the UI elements of a player slot in the player setup scene.*
5. ***/// </summary>***
6. **public** **class** PlayerSlot : MonoBehaviour
7. {
9. **public** GameObject AddButton;
10. **public GameObject Parent;**
11. **public** GameObject PlayerName;
12. **public** GameObject Color;
13. }

#### PlayersSetup.cs

1. **using** System;
2. **using** System.Linq;
3. **using** System.Collections.Generic;
4. **using** UnityEngine;
5. **using UnityEngine.UI;**
6. **using** NEAGame;
7. **using** TMPro;
9. */// <summary>*
10. ***/// <c>PlayersSetup</c> is the class that manages the player setup scene. It is used to display the player slots for the user to add or remove players.***
11. */// </summary>*
12. **public** **class** PlayersSetup : MonoBehaviour
13. {
14. **public** Button **Remove**;
16. **public** PlayerSlot[] playerAdders;
17. [SerializeField] **private** List<PlayerTemplate> players = new List<PlayerTemplate>();
18. [SerializeField] **private** GameObject newPlayerGUI;
19. [SerializeField] **private** GameObject ColorSelector;
21. **public** Button back;
22. **public** Button cont;
23. **public** Slider WinVP;
24. **public** Slider MaxTime;
25. **public GameObject WinVPstr;**
26. **public** GameObject MaxTimestr;
28. **private** Button AddPlayer;
29. **private** PlayerNameInp overlay;
30. **private ColorChoiceControl colorChoice;**
31. **private** List<**string**> colors = new List<**string**>();
33. **private** **float** RemoveCD = 0.0f;
34. **private** **bool** isGetting;
35. **private bool submittedName;**
36. **private** **bool** gettingColor;
37. **private** **int** playerChangingColor;
39. *// Start is called before the first frame update*
40. **void Start()**
41. {
43. **for** (**int** i = 0; i < 4; i++)
44. {
45. **playerAdders[i].gameObject.SetActive(false);**
46. }

49. cont.onClick.AddListener(**Continue**);
50. **cont.interactable = false;**
51. back.onClick.AddListener(Back);
52. **Remove**.onClick.AddListener(RemovePlayer);
53. SetupPlayerAdder();
55. **foreach (Player.PlayerColors playerColors in Enum.GetValues(typeof(Player.PlayerColors)))**
56. {
57. colors.**Add**(playerColors.ToString());
58. }
59. WinVP.onValueChanged.AddListener((**float** **value**) => { WinVPstr.GetComponent<TextMeshProUGUI>().text = **value**.ToString(); });
60. **MaxTime.onValueChanged.AddListener((float value) => { MaxTimestr.GetComponent<TextMeshProUGUI>().text = value.ToString() + "s"; });**
62. }
64. *// Method to setup the player adder buttons*
65. **void SetupPlayerAdder()**
66. {
67. **if** (players.Count == 4)
68. {
69. **return**;
70. **}**
71. playerAdders[players.Count].transform.localScale = new Vector3();
72. playerAdders[players.Count].gameObject.SetActive(**true**);
73. LeanTween.scale(playerAdders[players.Count].gameObject, new Vector3(1, 1, 1), 0.5f).setEase(LeanTweenType.easeInOutElastic).setDelay(0.5f);
74. AddPlayer = playerAdders[players.Count].AddButton.GetComponent<Button>();
75. **AddPlayer.onClick.AddListener(PlayerAddButton);**
77. isGetting = **false**;
78. }
80. ***// Method to add a new player to the game***
81. **public** **void** PlayerAddButton()
82. {
83. **if** (isGetting)
84. {
85. **return;**
86. }
87. isGetting = **true**;
88. overlay = Instantiate(newPlayerGUI).GetComponent<PlayerNameInp>();
89. overlay.gameObject.GetComponent<Canvas>().worldCamera = Camera.main;
90. **overlay.gameObject.GetComponent<Canvas>().sortingLayerID = 2;**
91. overlay.GetComponent<Canvas>().sortingLayerName = "UI";
92. overlay.gameObject.GetComponent<Canvas>().sortingOrder = 1000;
93. overlay.button.onClick.AddListener(SaveNewPlayer);
94. **if** (players.Count == 0)
95. **{**
96. overlay.toggle.interactable = **false**;
97. *// The first player is always a human*
98. }
99. submittedName = **false**;
100. **}**
102. *// Method to check if a color is already taken by another player*
103. **bool** checkIfColorIsTaken(**string** color)
104. {
105. **return players.Any(p => p.color == color);**
106. }
108. *// Method to get the first available color for a new player*
109. **string** FirstAvailableColor()
110. **{**
111. **foreach** (**string** color **in** colors)
112. {
113. **if** (!checkIfColorIsTaken(color))
114. {
115. **return color;**
116. }
117. }
118. **return** "clear"; *// never happens as there are always more colors than players*
119. }
121. *// Method to open the color change overlay allowing a player to change their color*
122. **void** ColorChangeButton(**int** index)
123. {
124. AudioManager.i.Play("UIClick");
125. **if (gettingColor)**
126. **return**;
127. gettingColor = **true**;
128. colorChoice = Instantiate(ColorSelector).GetComponent<ColorChoiceControl>();
129. *// use linq to get a list of the color indexes that are taken already by other players*
130. **List<int> indexes = players.Where(p => players.IndexOf(p) != index).Select(p => colors.IndexOf(p.color)).ToList();**
131. colorChoice.Setup(colors, indexes);
132. colorChoice.SetCurrent(players[index].color);
133. playerChangingColor = index;
135. **}**
137. *// Method to save the new color for a player*
138. **public** **void** NewColorSave(**string** choice)
139. {
140. **gettingColor = false;**
141. players[playerChangingColor].color = choice;
142. playerAdders[playerChangingColor].Color.GetComponent<Image>().color = UnityUI.textToColor(choice);
143. }
145. ***// Method to update the continue button to be interactable if there are enough players***
146. **void** updateContinue()
147. {
148. cont.interactable = players.Count > 1;
149. }
151. *// Method to save a new player to the game and close the input overlay*
152. **void** SaveNewPlayer()
153. {
154. AudioManager.i.Play("UIClick");
155. **if (submittedName) return;**
156. submittedName = **true**;
157. **string** name = overlay.inputField.text;
158. **bool** isbot = overlay.toggle.isOn;
160. **overlay.CloseGUI();**
161. **int** index = players.Count;
162. LeanTween.scale(playerAdders[players.Count].AddButton.gameObject, new Vector3(), 0.5f).setEase(LeanTweenType.easeInOutElastic).setOnComplete(() =>
163. {
164. **Remove**.interactable = **true**;
165. **playerAdders[index].PlayerName.GetComponent<TextMeshProUGUI>().text = name;**
166. playerAdders[index].PlayerName.GetComponent<TextAnim>().ResetTool();
167. playerAdders[index].Parent.gameObject.SetActive(**true**);
168. playerAdders[index].AddButton.SetActive(**false**);
169. **string** col = FirstAvailableColor();
170. **playerAdders[index].Color.GetComponent<Image>().color = UnityUI.textToColor(col);**
171. playerAdders[index].Color.GetComponent<Button>().onClick.AddListener(() => { ColorChangeButton(index); });
172. players.**Add**(new PlayerTemplate { name = name, ai = isbot, color = col.ToString() });
173. updateContinue();
174. SetupPlayerAdder();
175. **});**
176. }
178. *// Onclick method on the back button to go back to the home screen*
179. **private** **void** Back()
180. **{**
181. UnityUI.**Interface**.GoHome();
182. }
184. *// Update is called once per frame*
185. **void Update()**
186. {
187. RemoveCD -= Time.deltaTime;
188. }
190. ***// Method to remove a player from the game***
191. **void** RemovePlayer()
192. {
193. AudioManager.i.Play("UIClick");
194. **if** (RemoveCD > 0.0f)
195. **return;**
196. RemoveCD = 0.8f;
198. LeanTween.scale(**Remove**.gameObject, **Remove**.transform.localScale \* 0.8f, 0.1f).setEase(LeanTweenType.easeOutCirc).setLoopPingPong(1);
200. **players.RemoveAt(players.Count-1);**
201. updateContinue();
202. **if** (players.Count == 0)
203. {
204. **Remove**.interactable = **false**;
205. **}**
206. **if** (players.Count < 3)
207. {
208. LeanTween.scale(playerAdders[players.Count+1].gameObject, new Vector3(), 0.5f).setEase(LeanTweenType.easeInOutElastic);
209. }
210. **SetupPlayerAdder();**
211. LeanTween.rotateAround(playerAdders[players.Count].Parent, Vector3.forward, 360, 0.5f).setEase(LeanTweenType.easeInOutElastic).setOnComplete(() => {
212. playerAdders[players.Count].Parent.SetActive(**false**);
213. playerAdders[players.Count].AddButton.SetActive(**true**);
214. LeanTween.scale(playerAdders[players.Count].AddButton, new Vector3(1, 1, 1), 0.5f).setEase(LeanTweenType.easeInOutElastic);
215. **});**
216. isGetting = **false**;
217. }
219. *// Onclick method on the continue button to start the game*
220. **public void Continue()**
221. {
222. AudioManager.i.Play("UIClick");
223. *// For now the user may not change the number of starting resources as it gives players and unfair advantage over the AI*
224. UnityUI.**Interface**.game = new TravelersOfCatan(UnityUI.**Interface**, (**int**)WinVP.**value**, 1, MaxTime.**value**);
225. **foreach (PlayerTemplate p in players)**
226. {
227. **if** (p.ai)
228. {
229. UnityUI.**Interface**.game.AddAI(p.name, p.color);
230. **}**
231. **else**
232. {
233. UnityUI.**Interface**.game.AddPlayer(p.name, p.color);
234. }
235. **}**
236. UnityUI.**Interface**.CommenceGame();
237. }
239. */// <summary>*
240. ***/// Class used to store temporary player data while the game is being setup***
241. */// <br/> This is used to create a <c>Player</c> object when the game is started*
242. */// </summary>*
243. **class** PlayerTemplate
244. {
245. **public string name;**
246. **public** **string** color;
247. **public** **bool** ai;
248. }
249. }

#### PlayerTradeSlot.cs

1. **using** UnityEngine;
3. */// <summary>*
4. */// <c>PlayerTradeSlot</c> is the class that stores references to the UI elements of a player slot in the trade partner choice overlay.*
5. ***/// </summary>***
6. **public** **class** PlayerTradeSlot : MonoBehaviour
7. {
9. **public** GameObject ButtonObj;
10. **public GameObject PlayerName;**
11. **public** GameObject PlayerColor;
12. }

#### SaveSlot.cs

1. **using** UnityEngine;
3. */// <summary>*
4. */// <c>SaveSlot</c> is the class that stores references to the UI elements of a game save slot, along with the ID number.*
5. ***/// </summary>***
6. **public** **class** SaveSlot : MonoBehaviour
7. {
9. **public** GameObject Background;
10. **public GameObject Reset;**
11. **public** GameObject SlotText;
12. **public** GameObject StatusText;
13. **public** **int** ID;
15. **}**

#### TextAnim.cs

1. **using** System.Collections;
2. **using** UnityEngine;
3. **using** TMPro;
5. ***/// <summary>***
6. */// <c>TextAnim</c> is the class that handles the letter-by-letter animation of text objects.*
7. */// </summary>*
8. **public** **class** TextAnim : MonoBehaviour
9. {
10. **public TextMeshProUGUI textObj;**
11. **public** **bool** muted = **false**;
12. **public** **float** timeBtwnChars = 0.1f;
13. **public** **float** timeBtwnWords = 0.125f;
14. **public** **float** timeBtwnSentences = 0.15f;
16. **string** endText;
18. *// OnEnable is called whenever the object is enabled*
19. **public** **void** OnEnable()
20. **{**
21. StartCoroutine(TextVisible());
22. }
24. *// Method to reset the text object*
25. **public void ResetTool()**
26. {
27. textObj = GetComponent<TextMeshProUGUI>();
28. endText = textObj.text;
29. }
31. *// Method to add a letter-by-letter animation to the text object*
32. **private** IEnumerator TextVisible()
33. {
34. **yield** **return** 0;
35. **ResetTool();**
36. **if** (!muted)
37. AudioManager.i.Play("Write");
38. textObj.ForceMeshUpdate();
39. **int** totalVisibleCharacters = endText.Length;
40. **int counter = 0;**
42. **while** (**true**)
43. {
44. textObj.maxVisibleCharacters = counter;
46. **if** (counter >= totalVisibleCharacters)
47. {
48. **break**;
49. }
51. **if** (endText.ToCharArray()[counter] == ' ')
52. **yield** **return** new WaitForSeconds(timeBtwnWords);
53. **else** **if** (endText.ToCharArray()[counter] == '.')
54. **yield** **return** new WaitForSeconds(timeBtwnSentences);
55. **else**
56. **yield** **return** new WaitForSeconds(timeBtwnChars);
57. counter += 1;
59. }
60. **AudioManager.i.Stop("Write");**
61. }
62. }

#### VictoryManager.cs

1. **using** UnityEngine;
2. **using** TMPro;
4. */// <summary>*
5. ***/// <c>VictoryManager</c> is the class that manages the victory scene.***
6. */// </summary>*
7. **public** **class** VictoryManager : MonoBehaviour
8. {
10. **public GameObject MainText;**
11. **public** GameObject[] VictoryImages;
13. *// Method to set up the victory screen*
14. **public** **void** Setup(**string**[] names)
15. **{**
16. **string** endText = "";
17. **int** i = 0;
18. **foreach** (**string** name **in** names)
19. {
20. **VictoryImages[i].SetActive(true);**
21. endText += name + "**\n**";
22. i++;
23. }
24. MainText.GetComponent<TextMeshProUGUI>().text = endText;
25. **}**
26. }

#### SceneTransition.cs

1. **using** System.Collections;
2. **using** UnityEngine;
3. **using** UnityEngine.SceneManagement;
5. ***/// <summary>***
6. */// <c>SceneTransition</c> is a singleton class that handles the transition between scenes, including the cloud transition animation.*
7. */// </summary>*
8. **public** **class** SceneTransition : MonoBehaviour
9. {
10. **public static SceneTransition i { get; private set; }**
11. **public** Animator animator;
13. *// Start is called before the first frame update*
14. **void** Start()
15. **{**
16. **if** (i == **null**)
17. {
18. i = **this**;
19. }
20. **else**
21. {
22. Destroy(gameObject);
23. }
24. animator = GetComponent<Animator>();
26. DontDestroyOnLoad(gameObject);
27. SceneManager.sceneLoaded += (scene, mode) => OnNewScene();
28. }
30. ***// public method to play the scene transition animation***
31. **public** **void** PlayAnimation()
32. {
33. StartCoroutine(AnimationSameScene());
34. }
36. *// Coroutine to play the scene transition animation*
37. IEnumerator AnimationSameScene()
38. {
39. GetComponentInChildren<Animator>().SetTrigger("Exit");
40. **yield return new WaitForSeconds(1.25f);**
41. GetComponentInChildren<Animator>().SetTrigger("Enter");
42. }
44. *// Method to set the canvas to the new scene*
45. **void OnNewScene()**
46. {
47. StartCoroutine(WaitForNewScene());
48. }
50. ***// Wait for the new scene to load before setting the canvas to the new scene***
51. IEnumerator WaitForNewScene()
52. {
53. **yield** **return** new WaitForEndOfFrame();
54. Canvas myCanvas = GetComponent<Canvas>();
55. **myCanvas.renderMode = RenderMode.ScreenSpaceCamera;**
56. myCanvas.planeDistance = 100;
57. myCanvas.worldCamera = Camera.main;
58. myCanvas.sortingLayerName = "UI";
59. myCanvas.sortingOrder = 12000;
60. **GetComponentInChildren<Animator>().SetTrigger("Enter");**

63. }
65. ***// Send the player to the scene with the given name***
66. **public** **void** SendToScene(**string** sceneName)
67. {
68. **if** (sceneName == "Game")
69. {
70. **StartCoroutine(LoadGameScene());**
71. }
72. **else**
73. {
74. StartCoroutine(LoadScene(sceneName));
75. **}**
76. }
78. *// Coroutine to load the scene with the given name*
79. IEnumerator LoadScene(**string** sceneName)
80. **{**
81. GetComponentInChildren<Animator>().SetTrigger("Exit");
82. **yield** **return** new WaitForSeconds(1f);
83. AsyncOperation asyncLoad = SceneManager.LoadSceneAsync(sceneName);
84. **while** (!asyncLoad.isDone)
85. **{**
86. **yield** **return** **null**;
87. }
88. }
90. ***// Coroutine to load the game scene***
91. IEnumerator LoadGameScene()
92. {
93. AudioManager.i.Play("GetSetGo");
94. GetComponentInChildren<Animator>().SetTrigger("Exit");
95. **yield return new WaitForSeconds(1.5f);**
96. AsyncOperation asyncLoad = SceneManager.LoadSceneAsync("Game");
97. **while** (!asyncLoad.isDone)
98. {
99. **yield** **return** **null**;
100. **}**
101. }
102. }

# System testing *(8 marks)*

## Test plan

Objectives have not been tested in order however all 64 objectives are tested in this plan excluding objectives 19 and 52 which were not met.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Objective number | Test number | Purpose of test | Test data | Expected outcome | Reference to test result (timestamp) | Result |
| **Starting program** | | | | | | |
| 36 | 1 | Test that the program runs the game in a graphical user interface. | Run Program | Game loads to menu page | Will do once its on Youtube! | **PASS** |
| 37 | Test that the “Start Game” button sends the user to the save slot scene | Click on “Start Game” | The game save selection overlay should be opened |  | **PASS** |
| Test that the back button goes back to the initial scene | Click on back button | Game loads to menu page |  | **PASS** |
| **Testing Game Creation** | | | | | | |
| 38 | 2 | Select an Empty save to create a new game. | Select Save 1 | Player is sent to game creation page |  | **PASS** |
| 39 | Test that the “Add Player” button is functional | Click add player button | An overlay is opened to request the player name and if they are an AI |  | **PASS** |
| 40 | Ensure player name must be between 3 and 10 characters long | “ABCDEFGHIJK” | Text is cropped when typing over 10 characters long (the “k” should not be accepted) and the submit button is disabled when the text is less than 3 characters. |  | **PASS** |
| Ensure the name is alphanumeric | “Abc123!” | The “!” is not accepted by the program |  | **PASS** |
| Ensure at least one player is not an AI | Click on AI checkbox | The first player may not be an AI however any further players will be given the option to check the AI box. |  | **PASS** |
| 41 | Remove players from the game | Click on remove player button | If there 1 or more players the remove player button should be active and when pressed should remove the last player. |  | **PASS** |
| 42 | 3 | Test the player can change colour by clicking colour button | Click player colour | Colour change overlay opened |  | **PASS** |
| In this overlay, colours taken by other players should appear as a player icon with a lock on it. These buttons will be disabled. | N/A | Taken colours can not be selected by other players. |  | **PASS** |
|  | The user should be able to change the amount of time each player has on their turn. | Drag slider | Can take any value between 10s and 300s |  | **PASS** |
|  | The user should be able to change the number of victory points a player needs to win. | Drag slider | Can take any value between 15 and 50 |  | **PASS** |
| **Loading a game** | | | | | | |
|  | 4 | Test that a game can be loaded from the save slots | Click start game button  Load Save 1 | The user should be sent back to the game scene at the moment they left. The last player and time left should be maintained |  | **PASS** |
|  | Test that the data was saved to the permanent save data folder on the computer |  | The game was saved to a file “Save1.bin” in the persistent data folder |  | **PASS** |
|  | Test that if a corrupted file is loaded, the error is caught and a message is displayed | Edit file  Load Save 1 | A message is shown telling the user the file was corrupted |  | **PASS** |
|  | Test that a save can be reset | Click the Reset button on Save 1 | The save is deleted and the user may now create a new game in this slot |  | **PASS** |
| **In Game Testing** | | | | | | |
| 1, 8, 43 | 5 | The continue button should load the game scene which appears with 19 hexagons with random resources. | Load Save 3 (testing file) | Game scene loaded |  | **PASS** |
| 3 | The first player should be able to take their turn | N/A | The player one GUI is opened with buttons to control their turn. The first player’s name and colour are shown correctly. |  | **PASS** |
| 9 | Test that zoom functionality works | Click Zoom | The board is zoomed out |  | **PASS** |
| 44 | Test that the flags are shown next to building bought by players | N/A | The player’s city flag matches the player colour |  | **PASS** |
| 10 | 6 | The move button allows the current player to move to a new node. | Click move button | Node buttons appear over the nodes where the player may travel to. |  | **PASS** |
| 15 | Test that the Dijkstra’s algorithm is working | N/A | Only nodes at a distance reachable by the current player are shown. |  | **PASS** |
| 17 | Test the move function can be cancelled by clicking again. | Click move button again | All node buttons should be deactivated. |  | **PASS** |
| 16, 46 | Test that moving the player works. | Click move button  Click on random node | The player should visually travel to the selected node. |  | **PASS** |
| 11 | 7 | The shop button allows the current player to make a purchase | Click shop button | The shop overlay should be opened |  | **PASS** |
| 18 | Test the shop can be closed | Click close button | The shop overlay should be closed |  | **PASS** |
| 19 | Test that a village can be made if it is connected to a road | Open shop  Click on village | Village is shown in position and cost is deducted from player |  | **PASS** |
| Test that a village can be made if it passes the building rule | Open shop  Click on village |  |  | **PASS** |
| Test that a village cannot be made on top of existing buildings | Open shop  Click on village |  |  | **PASS** |
| 20 | Test that the user can purchase a city on an existing village | Open shop  Click on city | The user is charged and a city is constructed |  | **PASS** |
| Test that the user cannot purchase a city on a node without a village | Open shop  Click on city | A popup should be displayed explaining the issue. |  | **PASS** |
| 21 | Test that the user can purchase a road going to an existing building | Open shop  Click on road | The user is charged and a road is constructed |  | **PASS** |
| Test that the user cannot purchase a road if they cannot connect one end to a building | Open shop  Click on road | A popup should be displayed explaining the issue. |  | **PASS** |
| 22 | Test that the user can build a wall | Open shop  Click on wall | There are options to build walls |  | **PASS** |
| Test that connections cannot be bought on top of existing connections | Open shop  Click on wall | There is no option to override the existing connection |  | **PASS** |
|  | 8 |  | Click Inventory | The current players resources are displayed correctly |  | **PASS** |
|  |  | Close Inventory | The Inventory overlay is closed |  | **PASS** |
|  | Test that the user can undo movements | Make a move  Undo | The player should return to the original position and their moves left should be increased |  | **PASS** |
|  | Test that the user can undo purchases | Buy village  Buy road  Undo  Undo  Inventory | The player should be refunded and the board should be reset to the original state |  | **PASS** |
|  | 9 | Test that the user can select a trading partner | Click trading button | Only non-AI opponents should be displayed as options |  | **PASS** |
|  | Trading can be closed | Click close | The overlay closes |  | **PASS** |
|  | Test that user can set resources to be traded | Open trade with player | Each resource stores a value that is being traded |  | **PASS** |
|  | Test the overall indicator | Add some resources to both sides | The overall indicator should display the overall change in resource count |  | **PASS** |
|  | Test the player cannot gain more resource than the opponent has | Open trade  Add resources | The user cannot add more than the opponent has |  | **PASS** |
|  | Test the player cannot lose more resource than the current player has | Open trade  Add resources | The user cannot lose more than the current player has |  | **PASS** |
|  | Test the trade cannot be accepted until the last resource | N/A | Button should be disabled |  | **PASS** |
|  | Test the trade can be accepted | Click accept button | Both players should gain and lose the resources determined by the trade |  | **PASS** |
|  | There should be a visual indication of the trade completing | N/A | Resource cards should be seen travelling between players |  | **PASS** |
| 5, 45 | 10 | Test that once a player reaches the winning number of victory points, they are sent to the win scene | Build settlements | User sent to victory scene |  | **PASS** |
|  | Test that the game is saved once it is won |  |  |  |  |
| 14 | 11 | Test the current player’s turn is forcibly ended when the timer runs out | Create new game with 20 seconds per round | Player’s turn ends and GUI closes |  | **PASS** |
| 2 | Test that at the start of a player’s turn, they can see the resources around them get collected | N/A |  |  | **PASS** |
| 13 | Test that the end turn button is functional | Click end turn button | The current player’s turn should be ended |  | **PASS** |
| **AI** | | | | | | |
|  | 12 | Test the player can add AI opponents | Create game  Add player1  Add bot1 (AI)  Start game | The opponent should be added as an AI |  | **PASS** |
|  | Test that on the bot’s turn the player cannot use player overlay buttons | N/A | The overlay should hide most buttons |  | **PASS** |
|  | Test that the bot makes a move after its calculations | Load game 4  Click end turn and wait | The bot should make moves based on its calculations |  | **PASS** |
|  | Test that the bot makes a move even if its calculations are ongoing when there are a few seconds left | Setup game with 120 seconds per move and 3 players | Bot should make best moves |  | **PASS** |
| Setup game with 20 seconds per move and 3 players | Bot should make move based on limited calculation time |  | **PASS** |
| **Game pause / Saving** | | | | | | |
|  | 13 | Test that the pause menu works and opens the pause GUI |  | The pause overlay should be opened |  | **PASS** |
|  | Test the resume button works | Click “Resume” | The game should return to the game scene and the timer should resume |  | **PASS** |
|  | Test that a game can be saved by pressing the “Exit” button in the game save overlay | Click “Exit” | The game should go to the menu page and the game should be saved to the slot selected at the start |  | **PASS** |
|  | Test that if the user closes a game manually, the game still attempts to save | Close game abruptly  Reopen game  Reopen save | The game still saved |  | **PASS** |
| **Final Product** | | | | | | |
| 67 | 15 | On the menu page, clicking about will open an about page | Click “About” | User should be sent to about page with a button to return home |  | **PASS** |
| User can return home from about page | Click “Home” | User should be sent back to menu page |  | **PASS** |
| 68 | 16 | The user should be able to change the sound settings of the game from the home page | Click “Settings” | The pause scene should be loaded without the exit button |  | **PASS** |
| Test the overlay can be closed | Click home buttons | Return to menu page |  |  |
| 57 | Test that muting the background volume works |  | The background music is muted |  | **PASS** |
| 58 | Test that muting the SFX works |  | Button clicks and text display SFX are muted |  | **PASS** |
| 56 | Test that the volume slider works | Drag the volume slider around | The volume of the background music can be heard increase and decrease |  | **PASS** |
| 66 | Sound effects can be heard | Click buttons and go to about page | Sound effects can be heard |  |  |
| 69 | 17 | Game buttons should be animated | Click buttons | User should be able to see button animations |  | **PASS** |
| A transition animation should be played when a new scene is loaded | Go to a new page | User should see a transition animation |  | **PASS** |
| 70 | On the menu page, clicking exit will close the program | Click exit button | Program should terminate |  | **PASS** |

## Testing Videos

A video of the results from this testing plan can be found from the link below:

http://bit.ly/3OJ3w19

# Evaluation *(4 marks)*

## Comparison of project performance against the objectives

Overall, a majority of the objectives that I had initially intended to complete have been met, excluding the addition of event cards and the highwayman. These are more complicated game rules which could be implemented into the game in the future.

Whilst most of the objectives were met, two were skipped to keep the game intuitive and fast-paced. Due to a limited time frame for development, I instead focused on getting the AI algorithm efficient and on making the GUI immersive for the user. The objectives missed include the removal of the highwayman from the original concept along with event cards which should be bought from the shop. These objectives had a minimal overall effect on the project as a whole as there are many other game rules and features which were incorporated.

### MVP

All of the objectives from the MVP have been met or were superseded by later objectives. The game does not require a terminal so any reference to console inputs are later replaced by buttons or text. This stage allowed the game to be instantiated with players on a board that contains resources and nodes which is essential to creating the minimum viable product.

### Basic GUI

All of the objectives from the basic graphical user interface have been met. The game scene correctly displays a board of Catan and loads the players into the game. The game overlay is intuitive to use and all options work as planned. Overall the UI is very effective and allows the player to easily navigate the game.

### Advanced Game

In this section, two of the objectives were not met as mentioned at the start of 5.1. The rest of the objectives have been met which added the functionality to the game. The player can move to nodes within their reach using the Dijkstra’s algorithm which reduces the number of buttons the player would need to press. The player can purchase four different upgrades; in addition to the three upgrades in regular Catan, a wall has been added to block players from moving and developing adding a new tactical element to the game.

### Advanced GUI

All of the objectives in the advanced graphical interface were met. This allows the user to create a new game with the ability to add up to four players with custom names and colours along with a customisable required winning victory points and time per round. This means that the user can change the game to suit their needs by allowing games to be fast and slow paced. This stage was crucial to making the product feel like a game as it allows customisation which also helps immerse the players in the game.

### Load/Save

All of the objectives in the loading and saving stage of the product have been met. This allows the user to save existing games so they may be loaded later. This meets a key requirement for allowing the user to save their progress in order to continue later meaning that players do not need to worry about losing all of their progress if they cannot finish a game. The project completes this task effectively by storing the game in an encrypted format so the games cannot be altered and the games are saved to a permanent data folder meaning that even if the user deletes and reinstalls the project, the data will still be saved.

### Final Product

The objectives from this stage have all been completed and are very important to making the project feel like a videogame version of Settlers of Catan. The sound effects and animations give the project a feeling of immersion as all elements have interactable elements, which will help keep users engaged.

## Effectiveness of the solution

Overall the technical solution achieves the requirement of allowing the user to get immersed into the world of Catan. I am very pleased that a majority of the objectives have been completed successfully and all of the major requirements by the primary user were completed. The GUI is intuitive and well animated with sound effects to give the user the feeling of a videogame, opposed to a board game. The solution has passed a majority of the objectives as described in the previous section 5.1 and the objectives that were not met had a limited impact on the effectiveness of the solution, as there are new game rules to replace those that were skipped.

A drawback of the original board game is the lack of ability to save and load a game which can mean that players do not want to start a game knowing that they may not be able to finish it. However in the online game, users can create accounts to play multiplayer games however there is still limited functionality for saving and loading existing games. Completing the Loading and Saving objective in my project helps resolve this issue and allows players to take a break from the game and resume at a later date.

I am pleased with the implementation of the actual game; I think that the player’s inventory checking, moving, shopping and undoing are very effective. However the main game overlay could be confusing to new players so the game could benefit from tutorials or user guides that teach new players how to navigate the game. I also think that the trading system is not very effective as two players must agree on the deal in person before then entering their decision into the game, which could be altered to make this process more efficient.

I am happy with the customization of the game as there are many settings for the game player and the individual games themselves. I am very pleased with how the implemented AI algorithm works as it generates convincingly good moves and works very efficiently despite the complexity of the game. However one drawback from the current solution is the difficulty of certain games where the board was generated unfairly, or if the user would like to play a weaker or stronger AI.

Overall, the multiplayer functionality is not as effective as it could have been as users need to pass their device around and trading must be agreed on in real life without any back and forth negotiation within the game. My solution is similar to the board game implementation however I feel that online multiplayers have a better solution for this problem.

## Analysis of user feedback

**P**rimary user

**I**nterviewer

### Overall thoughts

I: What are your overall thoughts with the GUI?

P: The game definitely immerses me in the experience of playing a videogame of Catan.

I: How is the customization in each game?

P: It looks like it works really well. It allows for lots of variability between different rounds and allows you to play short or long games depending on what you want.

### UI and ease of use

I: How do the sound effects and GUI meet your expectations?

P: The GUI is quite important for making it feel like a game and it looks really good. It is much better than I was expecting given the time frame you had for development, as it has lots of good animations such as the clouds. The sound effects fit what you are attempting to do, as in they reflect my actions playing the game. They are really good.

I: how intuitive do you feel the game is to a new user?

P: I think that it is quite easy to learn while playing but since there are so many options it might be a bit overwhelming for completely new players so a tutorial or guide might be beneficial.

***One improvement may be the addition of a tutorial***

### Meeting requirements

I: How well has the bot met your requirements?

P: The bots are effective at their job. Bots appears to make reasonable moves that could be seen in a real game but these moves don’t seem perfect so it is possible to learn to be better than the bot which is the purpose of adding them.

I: You mentioned that you wanted to be able to save games to come back to them later. How well has this been implemented?

P: Looking at it now, it definitely works well. This will be perfect for storing a game after game night to resume it next time.

I: How well has the trading feature been implemented?

P: It allows you to trade all resources at the same time which is more efficient than the normal game in most cases which is very efficient. The indicator works very well as it conveys the entire trade in a simple and compact way. It’s a bit annoying that the players need to agree outside the game.

***One improvement could be to allow players to accept, reject or edit trades on their own turns***

I: What are your thoughts on the implementation of game timers to make the game faster-paced?

P: I think this works really well as it can be used for all levels of experience in players so they can take their time.

I: What are your thoughts on removing the random number resource generation because I decided to remove that from the base game.

P: I can see why you removed it as it helps make the game more skill based instead of praying for random luck, which is what I had asked for.

I: I didn’t get around to adding the highwayman and event cards, how much do you think that effects the gameplay?

P: Without event cards, the gameplay is slightly limited as there are fewer features however there are also new additions like the moveable player and walls so the gameplay is definitely still operational. I think I am happy you removed the highwayman as the new movement mechanic defeats the purpose of the highwayman concept in the original game so this is better.

I: Has the move undo feature met your expectations?

P: The features works well, you can undo your moves on your turn and allows you to re-make them if you feel you made a sub-optimal decision or you made a mouse-slip.

### Suggested improvements

I: Can you suggest any improvements to the game creation process?

P: A difficulty scale for either the bots or for the board generation could be a good addition to also allow players to determine how hard games are.

***Varying the depth of the AI search could effectively add a difficulty level to the bots***

I: The game is currently local multiplayer meaning you need to pass around the device. Is this a good system?

P: I think this works nicely as it emulates the feeling of a normal boardgame of Catan. It makes people have to gather around to play which is nice however if friends are not free to meet up it may make playing difficult.

***Adding a web server could allow players across the world to play the same game on the Internet***

## Possible improvements

### Functional changes and improvements

One problem is that the game difficulty cannot be customized. As said by the primary user in the feedback “*A difficulty scale for either the bots or the board generation could be a good addition*”. The current method for generating a board simply assigns each position on the board a random resource however a better implementation may be having a fixed number of each resource and randomly assigning each a unique position, to ensure that there are the correct numbers of resources on the board. In addition to this, a weight could be added to make resources generate closer or farther apart which would make the board generation subject to customisation. In addition, the AI BRS algorithm works well however the maximum depth has been fixed at 4, which can cause the AI to work slowly in certain scenarios. It would be more efficient to implement an algorithm like iterative deepening which would allow the algorithm to continue searching for better moves while not overrunning the time it has on its turn. This could also be altered to allow the bot a maximum depth depending on the difficulty of the bot, as a higher depth means the bot can make a more logical move based on future positions.

The trading mechanic can be altered so that the user can offer, edit, accept, or reject trades on their turn. This way, on a player’s turn they can offer a trade to another player and they must wait for the other player’s turn for them to accept the trade. In addition if the other player is not satisfied there could be an option to edit the trade and re-offer it to the original player. This will make it easier for players to trade in the game and will also make the game less interruptive for other players not involved in the trade.

### Further/Additional Improvements

One potential addition is the implementation of a tutorial system to teach newer players how to operate the game. As my primary user said “*since there are so many options it might be a bit overwhelming for completely new players*”. This could occur during the first time the user starts a game, and a file could be used to register that the tutorial has been played. This tutorial could be an interactive walkthrough of the overall game UI along with a breakdown of the game rules.

The board could also be expanded to allow for even more players in the future by making the board size dynamically generated by the Board class. This could be done by changing the for loops that generate the board to use a value calculated from the number of players inputted. The original board game has this functionality through an expansion pack.

The game could also support an online multiplayer mode as recommended by the primary user. As said by the primary user: “*if friends are not free to meet up it may make playing difficult*”. This could be implemented by creating a web server to control the clients in a game and would forwards web sockets sent on each players turn to all other players in order to keep the games synchronized. This could be implemented quite easily through the use of the serializable GameAction objects that are created when a user makes a move. These actions could be serialized into JSON and sent through a web request or web socket, which could be echoed to all other connected clients.

# Appendix

## Git log

The following screenshots from GitHub show my repositories commit history, which demonstrates how the project was decomposed and progress on completing the objectives was achieved over time.

A white rectangular object with blue lines

Description automatically generated A white rectangular object with a blue line

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white rectangular object with blue lines

Description automatically generated A white rectangular object with a black border

Description automatically generated

A white rectangle with blue lines

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white rectangular object with a blue stripe

Description automatically generated with medium confidence

A screenshot of a phone number

Description automatically generated

A white lined paper with blue lines

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A white rectangular object with a black border

Description automatically generated with medium confidence

## References to web sites or other resources used

This project was constructed using the Unity Editor (2021.3.31f1). Along with Unity which aided the visual development of the game scenes, Visual Studio was used as a C# coding IDE which includes a built-in Unity plugin. In addition, GitHub Copilot was used in order to assist the programming process.

**Useful sites for learning:**

The following StackOverlow pages and YouTube tutorials were beneficial for learning certain techniques used in C# and Unity to complete specific tasks:

This video was helpful for learning how to implement an Audio manager to my project:

<https://www.youtube.com/watch?v=6OT43pvUyfY>

This page provided a simple XOR encryption algorithm to perform bitwise encryption on a character array:

<https://stackoverflow.com/questions/2532668/help-me-with-xor-encryption>

This video is a tutorial for implementing a file handler in Unity to save and load JSON to files.

<https://www.youtube.com/watch?v=KZft1p8t2lQ>

Other StackOverflow pages along with the relevant language documentations were also used to debug errors during the programming process.

**Mathematical hexagon grid model:**

<https://www.redblobgames.com/grids/hexagons/>

This website was helpful for learning different methods of implementing a coordinate system for tessellated hexagons. This also taught me how to convert between different coordinate systems as used by my mathematical model of the board and by Unity’s tilemap rendering.

**Unity Asset Store links:**

Apart from using several built-in Unity extensions such as TextMeshPro, several external assets were imported from the Unity Asset Store, which are listed below:

**LeanTween** was used to animate elements within the user interface to make the game feel more aesthetically pleasing and interactive.

<https://assetstore.unity.com/packages/tools/animation/leantween-3595>

**Hyper casual mobile GUI** was chosen as it is free and contains high quality button and UI image assets.

<https://assetstore.unity.com/packages/2d/gui/hyper-casual-mobile-gui-268659>

**Free casual pack SFX** was chosen as it is free and contains a variety of high quality sound effects to make the game feel more interactive.

<https://assetstore.unity.com/packages/audio/sound-fx/free-casual-pack-sfx-197054>

**Fatality FPS gaming font** was chosen as it is a free and high resolution font which scales well with Unity’s TextMeshPro.

<https://assetstore.unity.com/packages/2d/fonts/fatality-fps-gaming-font-216954>

**Clean Vector Icons** was chosen as it is a large collection of free and high quality vector icons which are used as buttons in many UI pages.

<https://assetstore.unity.com/packages/2d/gui/icons/clean-vector-icons-132084>

**Simple Button Set 02** was chosen as it contains free button background images with hover and selection effects.

<https://assetstore.unity.com/packages/2d/gui/icons/simple-button-set-02-184903>

**Openart AI**

<https://openart.ai/>

This website was used to generate free assets through generative artificial intelligence. This specific tool does not claim copyright over the generated content.