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 3](#_Toc156208165)

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

[1.2 Research carried out 4](#_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 6](#_Toc156208170)

[1.3 Identification of the prospective user(s) 6](#_Toc156208171)

[1.3.1 Interview 6](#_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 10](#_Toc156208179)

[1.4.3 Advanced Game 10](#_Toc156208180)

[1.4.4 Advanced GUI 11](#_Toc156208181)

[1.4.5 Load/Save 12](#_Toc156208182)

[1.4.6 Final Product 12](#_Toc156208183)

[1.5 Modelling diagrams 13](#_Toc156208184)

[1.5.1 High-level software object model 13](#_Toc156208185)

[1.5.2 Flowchart through program 14](#_Toc156208186)

[2 Design (12 marks) 16](#_Toc156208187)

[2.1 System design overview 16](#_Toc156208188)

[2.2 Data structures 18](#_Toc156208189)

[2.2.1 Tree 18](#_Toc156208190)

[2.2.2 Graph 18](#_Toc156208191)

[2.2.3 Stacks 19](#_Toc156208192)

[2.3 Algorithms 19](#_Toc156208193)

[2.3.1 Static Evaluation 19](#_Toc156208194)

[2.3.2 BRS 19](#_Toc156208195)

[2.3.3 Djikstra’s 21](#_Toc156208196)

[2.4 File structure and organisation 22](#_Toc156208197)

[2.5 User interface design 22](#_Toc156208198)

[2.5.1 Home Screen 22](#_Toc156208199)

[2.5.2 Game Screen and overlays 22](#_Toc156208200)

[2.6 User guide 24](#_Toc156208201)

[*3* Technical Solution *(42 marks)* 25](#_Toc156208202)

[3.1 Completeness Section 25](#_Toc156208203)

[3.2 Technical Skills Section 25](#_Toc156208204)

[3.2.1 Overview Guide 25](#_Toc156208205)

[3.2.2 Code listing 27](#_Toc156208206)

[4 System testing *(8 marks)* 28](#_Toc156208207)

[4.1 Test plan 28](#_Toc156208208)

[4.2 Testing Videos 32](#_Toc156208209)

[5 Evaluation *(4 marks)* 33](#_Toc156208210)

[5.1 Comparison of project performance against the objectives 33](#_Toc156208211)

[5.2 Effectiveness of the solution 33](#_Toc156208212)

[5.3 Analysis of user feedback 33](#_Toc156208213)

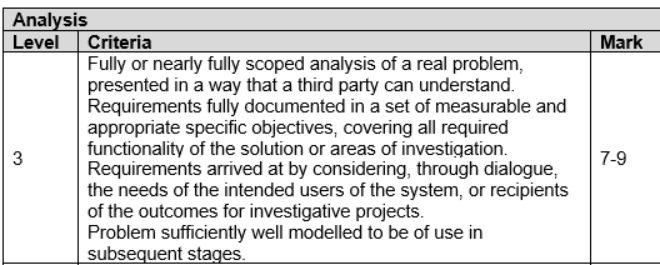
[5.4 Possible improvements 33](#_Toc156208214)

[6 Appendix 34](#_Toc156208215)

[6.1 Git log 34](#_Toc156208216)

[6.2 References to web sites or other resources used 34](#_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.

### 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 between 5 and 6 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 online 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 can not cheat by accessing the save file.

For networking, using HTTPS requests like GET and POST is an option for making a multiplayer game however this may end up being slow and therefore laggy. An alternative method would be to use web sockets which are much faster.

## 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.

This is a short paragraph that identifies the user and describes them and their need. It should include detailed interviews with the prospective user (not superficial yes/no questions) where the user’s opinion is sought in detail. Make specific links from the user’s requirements to the objectives of the project.

### 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.

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.

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 after rolling the dice, the resources spawn on the grid and the moving piece needs to go and collect them.

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? ***[Objectives 3, 16-18]***

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.

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? ***[Objective 24]***

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.

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.

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.

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***.

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 leaderboard so I can see my skill improving and it will make the game competitive with friends.

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 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 its 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 its 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.

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.

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: There should be a random chances of a recourse spawning at a location and this chances should be smaller if there is a city or village on its border.

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

These should cover all required functionality of the solution or areas of investigation ('appropriate' means the specific objectives are single purpose and at a level of detail that is without ambiguity)

NB These must be complex even if you are unable to code them all fully! They should be described in sufficient complexity as to meet the standard required of A level. Each user need should be identified as a measurable objective and then within that objective specific outcomes should be identified and listed. These should not include any add-ons of standard algorithms nor any code that will be generated by a GUI. This is the most important section for you to get right!!!!!!!!

### 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.

*[Terminal]* After a player has made their move they will be given the option to make a purchase or continue.

* 1. 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.
  2. If the player chooses continue, the next player will get a turn.
  3. After each players turn, if the player has reached the winning victory points the game will display the ranking of the players in the game to the console and should terminate.

### Basic GUI

* 1. The program should launch the game scene; a graphic game should open and at the start of the player’s turn, a player GUI overlay should be opened.
  2. The Grid will be formed of 19 tessellated hexagons with an image of the randomly generated resource in the centre of the hexagon. Both players should appear on the board with their colours.
  3. Resources generated around the player and their cities should be displayed on the screen.

###### Player GUI overlay 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 be enabled.
  2. Clicking on a node will disable all other nodes and move the player to this position.
  3. Clicking again on the move button will disable all nodes and should cancel the movement.

###### Shopping in more detail:

* 1. If the user presses the close button, the overlay should be closed.
  2. The user should be able to navigate through all of the settlements that they can purchase.
  3. Villages can only be purchased if it is connected to a road or if it passes the “Distance rule”
  4. Cities can only be purchased on a player’s village
  5. Roads can only be purchased if one end connects to a village or city
  6. Walls can be purchased as a connection and should block all enemies from passing. The wall should also take more moves to cross compared to an empty connection.
  7. If the player attempts to make a purchase in a position where this purchase cannot be made, an error should be made stating why they were unable to make this purchase.
  8. After a successful transaction, an image of the village, city, road or wall should be shown 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.
  2. If the user clicks on the cancel button, the GUI should close.
  3. If the user selects a player option, the trading overlay will be opened with this other player.
  4. 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.
  5. An overall indicator for the resultant change in resource count should also be displayed. Red and green should be used to indicate and increase and loss in a resource.
  6. 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.
  7. If the trade is accepted the resources should be exchanged as specified by the user in the overlay and the GUI should close.
  8. 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.
  2. 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.
  3. 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.
  4. 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 hidden.
  2. After the time reaches the final few seconds OR if the bot has completed its search, the best moves selected by the algorithm should be made graphically 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.
  2. 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.
  3. The entered player name must be alphanumeric and between 3 and 10 characters and the first player may not be an AI.
  4. Once the user submits these values, this player will be shown in the slot and the next button 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.
  7. In this overlay, colours taken by other players should appear as a player icon with a lock on it. These buttons will be disabled.
  8. 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. All villages and cities should have a flag with the player colour of the occupant.
  2. Once a player wins, the user will be sent to the victory scene where a message will be displayed congratulating the winner of the game. If the player clicks on the home button, they will be sent back to the menu page.
  3. After successfully completing a trade, card images should be exchanged between the player visually to indicate the trade completed.
  4. The player should visually travel along the shortest path when the user moves the current player to a new position.

###### Pause

* 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 options:
  2. Continue – 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 to change the application volume levels
  5. Mute background music – the background music playing will be stopped
  6. Mute SFX – sound effects will not be played

### Load/Save

###### Loading a Game

* 1. In the “Start Game” page, the user should 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. The main game should be serialized and stored in a file in the persistent data folder so it may be loaded again later.
  5. 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 they will be presented with the option to exit, which should send them back to the home page. The game should be saved to the slot they had selected at the start of the game.
  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 with 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. Game buttons should be animated and when traveling between scenes, a transition animation should be played.
  4. Sound effects should be heard for buttons, text displaying and animations. Background music should be heard.

## Modelling diagrams

E.g. E-R model, High level software object model, Data Flow Diagrams, mathematical models, the flow through the program.

### High-level software object model

A diagram of a flowchart

Description automatically generated

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

### Flowchart through program

A diagram of a flowchart

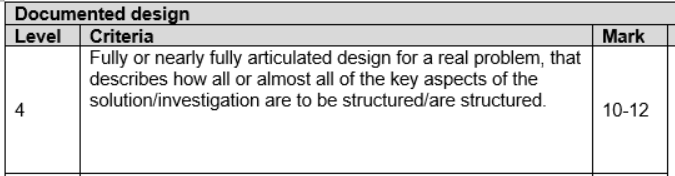
Description automatically generated

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)

Data flow diagram

A diagram of a software program

Description automatically generated with medium confidence

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). The **UI** interface defines a set of methods that must be implemented in any classes that implement the interface. This includes all of the functions that the main game class requires to operate. The main class for controlling the game is **TravelersOfCatan**. which creates the board and contains a list of the players in the game. **TravelersOfCatan** also defines all of the event based functions for the player to make moves on their turn. The **Board** class is a representation of the graph data structure as it contains all of the nodes in the game, along with the connections between each of them in an adjacency matrix. A **Node** object represents a single point on the grid where a player can reside, and where settlements can be built. These nodes contain a **Building** object which is a type of **Settlement** and contains information about the owner and type of building on a node. Likewise the **Connection** object contains the necessary information for a connection within the graph. A connection and a building are both settlements, so inherit from the **Settlement** abstract class, which has necessary information for all settlements. A **Board** also consists of **HexagonUnits** which are objects to represent the central hexagons in the mathematical model. These need to be referenced in order to determine what resources to give each player. The **HexagonUnit** has a **Resource** object which stores what resource is in each square and has a random generation constructor. The **TravelersOfCatan** game also has an array of the game’s **Player** objects. This is the definition of a human or computer character within the game. If the player is a computer, it will be an **AI** object instead which inherits from player but redefines the player’s input functions to call the BRS and Dijkstra’s algorithms to find the best moves.

// Add Unity Classes Here As Well : (

## 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 5 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, 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 diagram of a function

Description automatically generatedA screenshot of a cellphone

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. It does this by summing all of the victory points and resources of each player and returning the end value.

### 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.

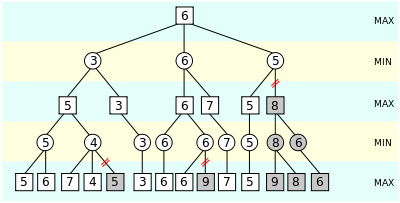
The generate moves function to deploy will scan over possible positions for the player and their possible purchases each round without considering all possible combinations of player movements. This will increase the efficiency of the algorithm exponentially as a player has up to three moves, each adding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

All of the files used in coding as well as created when playing the game?

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 access a saved game state.

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

|  |  |
| --- | --- |
| 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. |
| Unity | |
| 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. |
| SceneTransition.cs |  |
| JSON\_manager.cs |  |
| FileHandler.cs |  |
|  |  |
|  |  |

## 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 fsor 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.

A diagram of a game

Description automatically generated with medium confidence

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 required data folders. These files can be further zipped into a standalone executable file using WinRAR or other archiving tools. To execute the program from the command line, enter the file name which should end in .exe and the system should execute this file. Alternatively the user may launch the executable from the file explorer instead.

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.

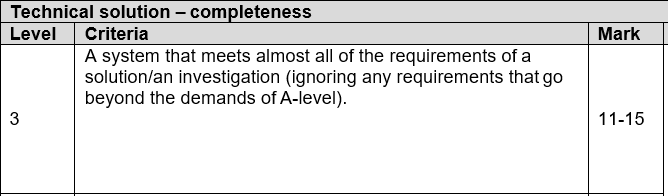
It was common to see algorithms appear without the reader having any real understanding as to how these fitted into the system and a few sketches or examples of the data to be used would help. So, for example, one student produced an excellent Sudoku solver which had some complex pseudo-code in the design 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)*

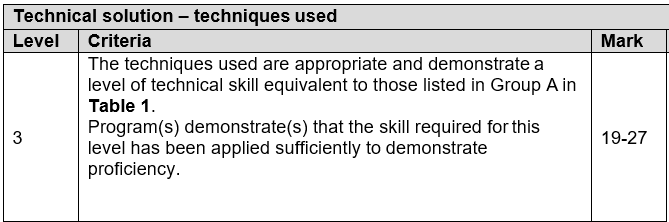
## Completeness Section

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?



## 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

You should produce a guide here that highlights techniques/ skills in sections A B and C with commentary. Your code should be commented to show where specific technical skills are being used e.g. # SKILL GROUP A – Graph Traversal

|  |  |  |  |
| --- | --- | --- | --- |
| **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. | TravelersOfCatan.cs |
| A | Recursive algorithms | BRS is a recursive algorithm that calls itself on each new position while altering the remaining depth to view future game states and determine the most optimal move for players. | AI.cs   * BRS() |
| A | Tree traversal | A tree structure has been used to represent the state of the position and is traversed by the recursive algorithm BRS. | AI.cs   * BRS() |
| A | Stack operations | A stack has been used to keep an ordered collection of moves within the AI move generation algorithm. | AI.cs  TravelersOfCatan.cs |
| A | Complex Mathematical Model | The board is stored as a graph of vector3 positions representing coordinates in cubic space which allows the positions of adjacent nodes to be calculated mathematically instead of storing this data as pointers. | UnityUI\_Game.cs  Board.cs |
| A | Complex data structure (Hash table) | A dictionary of dictionaries has been used to connect two adjacent nodes to their connection 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 |
| A | Complex user-defined use of OOP |  | Board.cs  TravelersOfCatan.cs |
| Inheritance |  | AI.cs  TravelersOfCatan.cs   * GameActions |
| Polymorphism |  |  |
| A | Dynamic generation of objects | AI move selection algorithm | AI.cs |
| A | Complex User-Defined Algorithm | ???????? | TravelersOfCatan.cs |
| B | Reading/Writing to text files | The file save and load function serializes a game object into JSON and writes this into a file after applying an XOR encryption so the user cannot change these values manually. | JSON\_manager.cs  FileHandler.cs |

#### Coding Style

|  |  |  |
| --- | --- | --- |
| **Coding Practice** | **Location in project** | **Description** |
| Whitespace | Everywhere |  |
| Variable Names | Namespace NEAGame |  |
| Commenting | Namespace NEAGame |  |
| Constants | TravelersOfCatan.cs  JSON\_manager.cs |  |
| Public Private Variables | Namespace NEAGame |  |

### Code listing

This should be fully commented and broken into suitable sections with subheadings. You can turn this section to landscape to make it easier to read. Ensure this is **fully annotated** with comments. Can use a website like hilite.me. Do not use ‘dark mode’ images or screenshots

#### TravelersOfCatan.cs

#### Board.cs

#### Player.cs

#### AI.cs

#### Connection.cs

#### Node.cs

#### UnityUI.cs

#### UnityUI\_Game.cs

#### UIManager.cs

#### GamePauseOverlay.cs

#### InventoryPopup.cs

#### ColorChoiceControl.cs

#### PlayerChoice.cs

#### PlayerNameInp.cs

#### PlayerUIOverlay.cs

#### PopupController.cs

#### SaveSelector.cs

#### ShopOverlay.cs

#### TradingInterface.cs

#### AudioManager.cs

#### CardCollection.cs

#### ConnectionAnimator.cs

#### MenuPage.cs

#### NodeButton.cs

#### PlayerAnimation.cs

#### PlayerSlot.cs

#### PlayersSetup.cs

#### PlayerTradeSlot.cs

#### SaveSlot.cs

#### TextAnim.cs

#### VictoryManager.cs

#### SceneTransition.cs

# System testing *(8 marks)*

## Test plan

Objectives have not been tested in order however all 74 objectives were tested in this plan.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Objective number | Test number | Purpose of test | Test data | Expected outcome | Reference to test result (timestamp) | Result |
| **Testing Game Creation** | | | | | | |
| 42 | 1 (part 1) | Test that the program runs the game in a graphical user interface. | N/A | Game loads to menu page |  | **PASS** |
| 43 | 2 | Test that the “Create Game” button sends the user to the game setup page | Click on “Start Game” | The game setup page should be loaded |  | **PASS** |
|  | Test that the back button goes back to the initial scene | Click on back button | Game loads to menu page |  | **PASS** |
| 44 | Select an Empty save to create a new game. | N/A | Player is sent to game creation page |  | **PASS** |
| 45, 46 | 3 | 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 |  |  |
| 47 | Ensure player name must be between 3 and 10 characters long | 12345678910 | Text is cropped when typing over 10 characters long and the submit button is disabled when the text is less than 3 characters. |  | **PASS** |
| 47 | 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** |
|  | Ensure the name is alphanumeric | Abc123! | The “!” is not accepted by the program |  |  |
| 49 | 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** |
| 50 | 4 | 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. |  |  |  | **PASS** |
| 51 | In this overlay, colours taken by other players should appear as a player icon with a lock on it. These buttons will be disabled. |  |  |  | **PASS** |
| **Testing Game Scene** | | | | | | |
| 52 + 8 | 5 | The continue button should load the game scene which appears with 19 hexagons with random resources. |  | Game scene loaded |  | **PASS** |
| 7 | The first player should be able to take their turn |  | The player one GUI is opened with buttons to control their turn. The first player’s name and colour are shown correctly. |  | **PASS** |
| 11 |  | 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** |
| 16 | Test that the Djikstras algorithm is working | N/A | Only nodes at a distance reachable by the current player are shown. |  |  |
| 18 | Test the move function can be cancelled by clicking again. | Click move button again | All node buttons should be deactivated. |  | **PASS** |
| 17 | Test that moving the player works. | Click move button  Click on random node | The player should visually travel to the selected node. |  | **PASS** |
|  |  | The shop button allows the current player to make a purchase | Click shop button | The shop overlay should be opened |  | **PASS** |
|  |  | Test the shop can be closed | Click close button | The shop overlay should be closed |  | **PASS** |
| 19 |  | Test that the user can purchase a city on an existing village | Open shop  Click on village | The user is charged and a city is constructed |  | **PASS** |
| 22 |  | Test that the user can not purchase a city on a node without a village | Open shop  Click on village | A popup should be displayed explaining the issue. |  | **PASS** |
| 23 |  | Test that the user can purchase a road from an existing building | Open shop  Click on road | The user is charged and a road is constructed |  | **PASS** |
|  | 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** |
|  |  | 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** |
|  |  | Test that a village can be made if it is connected to a road | Open shop  Click on village |  |  | **PASS** |
|  |  | Test that a village can be made if it passes the building rule | Open shop  Click on village |  |  | **PASS** |
|  |  | Test that a village can not be made on top of existing buildings | Open shop  Click on village |  |  | **PASS** |
|  |  | Test that zoom functionality works | Click Zoom | The board is zoomed out |  | **PASS** |
|  |  |  | 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** |
|  |  | 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 trade can still be cancelled | Click close | Overlay closes |  | **PASS** |
|  | Test the trade can not 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 |  | Resource cards should be seen travelling between players |  | **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 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 | Click end turn button | The overlay should hide most buttons |  | **PASS** |
|  |  | Test that the bot makes a move after its calculations | Click end turn and wait | The bot should make moves based on its calculations |  | **PASS** |
|  |  | Test the current player’s turn is forcibly ended when the timer runs out |  |  |  | **PASS** |
|  |  | Test that the bot makes a move even if its calculations are ongoing when there are a few seconds left |  |  |  | **FAIL** |
|  |  | Test that the flags are shown next to building bought by players |  |  |  | **PASS** |
|  |  | Test that the pause menu works and opens the pause GUI |  |  |  | **PASS** |
|  |  | Test that muting the background volume works |  |  |  | **PASS** |
|  |  | Test that muting the SFX works |  |  |  | **PASS** |
|  |  | Test that the volume slider works | Drag the volume slider around | The volume of the background music can be heard increase and decrease. |  |  |
|  |  |  |  |  |  |  |
|  |  | Test that a game can be saved….. |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  | Test that a save slot that has data can be loaded. |  |  |  |  |
|  |  | Test that |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  | Test that the volume slider works |  |  |  |  |
|  |  | Test that the background music can be muted |  |  |  |  |
|  |  | Test that the SFXs can be muted |  |  |  |  |
|  |  | Test that the game rules can be displayed |  | The game rules are shown in a popup |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## Testing Videos

Record your program being used, narrate each objective being shown to have been implemented. Give the objective in the audio of the recording. Provide bit.ly shortened links in the document to your videos which have been uploaded to youtube. Use a large monospace (e.g courier new) font with no underlining e.g.

https://bit.ly/abc0DEF12

# 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.

### 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 allowed the game to be instantiated with players on a board that contains resources and nodes.

### Basic GUI

The game scene was created

## Effectiveness of the solution

Overall the technical solution achieves the requirement of allowing the user to get immersed into the world of Catan. 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.

A drawback of the 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, user’s 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.

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. This is less optimal than some online multiplayer games of Catan

## Analysis of user feedback

This should include the original user who was involved at the analysis stage

Transcript + discuss transcript

## Possible improvements

Be verbose about what could be improved – make a full statement. Get user feedback for improvement

INCLUDE QUOTES FROM ADAM

Adding multiplayer

Event cards

# Appendix

## Git log

## References to web sites or other resources used

LeanTween

Unity Asset Store links:

Openart AI

