|  |
| --- |
| BHASVIC Computer Science |
| **Animalopoly** |
|  |

|  |
| --- |
| Dylan Barnard  [Date] |

Contents

[Analysis 2](#_Toc196816466)

[What goes in this section? 2](#_Toc196816467)

[Success Criteria 3](#_Toc196816468)

[Design 7](#_Toc196816469)

[What goes in this section? 7](#_Toc196816470)

[Checklist 8](#_Toc196816471)

[Development 9](#_Toc196816472)

[What goes in this section? 9](#_Toc196816473)

[Checklist 10](#_Toc196816474)

[Evaluation 11](#_Toc196816475)

[What goes in this section? 11](#_Toc196816476)

[Checklist 12](#_Toc196816477)

[References 13](#_Toc196816478)

[References Explanation 13](#_Toc196816479)

[Reference List 15](#_Toc196816480)

[AI References 16](#_Toc196816481)

[AI References Explanation 16](#_Toc196816482)

[AI Reference List 16](#_Toc196816483)

[Referencing FAQs 17](#_Toc196816484)

[What happens if I don’t cite a source I’ve used? 17](#_Toc196816485)

[What if I don’t realise, I’ve plagiarized? 17](#_Toc196816486)

[What are the potential consequences of plagiarism? 17](#_Toc196816487)

[If AI generated work isn’t marked, then why use it? 18](#_Toc196816488)

# Analysis

## What goes in this section?

In your actual programming project, the analysis section is a significant piece of work in which you justify why the program/product you have decided to create should be created. You’ll look at existing solutions and competitors to examine what their strengths and weaknesses are and what opportunities and threats this creates for your project. You’ll also use questionnaires and interviews with your potential stakeholders/target audience to understand their needs.

Once you’ve gathered this information, you’ll create success criteria that will inform your development and design tests that will ensure that you can evidence your progress towards meeting these success criteria.

You’ll complete the Analysis Phase of your programming project before the Summer Break so that you can then focus on design in September.

For Animalopoly, we’ve given you the success criteria to save time! The only thing you need to do is to prioritize them based on their importance:

* Green = Most Important, do this first!
* Yellow
* Red = Least Important, do this last!

## Success Criteria

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Success Criteria** | **Justification** | **Importance** |
| 1. Dice | | |  |
| 1.1 | Two Dice Implementation: Implement two dice that the player can roll, with each die generating a random number between 1 and 6. | Implementing two dice is fundamental to the game mechanics, as it determines how far players move on the board. This randomness adds excitement and variability to each turn, making the game more engaging. |  |
| 1.2 | Dice Roll Outcome: Display the total number of spaces the player should move based on the sum of the dice roll. | Displaying the number of spaces the player should move based on the dice roll ensures players understand their movement clearly. This transparency is crucial for maintaining the flow of the game and preventing confusion. |  |
| 1.3 | Double Roll Detection: Detect and confirm when two of the same dice have been rolled, triggering a special event. | Confirming when two of the same dice have been rolled is important because it triggers special events, such as drawing a card. This feature adds an extra layer of strategy and unpredictability, enhancing the overall gameplay experience. |  |
| 2. Board | | |  |
| 2.1 | Board Display: Create and display a text-based board with 26 spaces to the user. | Creating a text-based board with 26 spaces and displaying it to the user provides a visual representation of the game state. This helps players keep track of their positions and the overall progress of the game, making it more immersive. |  |
| 2.2 | Player Location Storage: Store and update the location of each player on the board after every move. | Storing the location of each player on the board is necessary for tracking player positions and game progress. It ensures that the game can accurately reflect each player's movements and interactions with the board spaces. |  |
| 2.3 | Current Player Position: Display the current position of each player on the board at all times. | Showing on the board where each player currently is helps players understand their current status in the game. This visibility is essential for strategic planning and decision-making during their turns. |  |
| 2.4 | Space Instructions: Provide specific instructions to the player based on the space they land on. | Informing the player what to do when they land on a specific space ensures players know the rules and actions required for each space. This clarity is vital for smooth gameplay and helps prevent misunderstandings. |  |
| 2.5 | Start Space Rewards: Automatically give the player £500 when they pass start and £1000 when they land on start. | Giving the player money when they pass (£500) or land on (£1000) start adds strategic elements and rewards for player movement. These incentives encourage players to keep moving and add excitement to reaching key points on the board. |  |
| 2.6 | Miss a Turn: Have the player miss their next turn when they land on the "miss a turn" space. | Having the player miss a turn when they land on the appropriate square introduces penalties and adds complexity to the game. This feature can change the dynamics of the game and requires players to adapt their strategies. |  |
| 3. Player | | |  |
| 3.1 | Playing Piece Selection: Allow players to select a text-based playing piece (e.g., \*, @, ?). | Allowing players to pick a text-based playing piece personalizes the game experience for each player. This customization makes the game more enjoyable and helps players feel more connected to their in-game character. |  |
| 3.2 | Player Name Storage: Store and display the player's name at the start of their turn. | Storing and announcing the player's name on their turn enhances player identification and engagement. It adds a personal touch to the game, making each turn feel more significant and interactive. |  |
| 3.3 | Money Management: Track and update the amount of money each player has throughout the game. | Storing and updating the amount of money each player has is central to the game mechanics and determining the winner. This feature is crucial for tracking financial transactions and ensuring the game progresses correctly. |  |
| 3.4 | Bankruptcy Notification: Notify players when they run out of money and are eliminated from the game. | Informing players when they run out of money indicates game status and player elimination. This notification is essential for maintaining the competitive aspect of the game and ensuring players are aware of their standing. |  |
| 3.5 | Winning Condition: Announce the winner when they are the last player remaining with money. | Announcing the winner when they are the last player with money defines the end goal and victory condition of the game. This clear objective helps players understand what they are striving for and adds excitement to the competition. |  |
| 4. Animals | | |  |
| 4.1 | Animal Assignment: Assign a unique animal to each space on the board. | Assigning an animal to each space on the board adds thematic elements and variety to the game. This feature enriches the gameplay experience by introducing unique attributes and interactions for each space. |  |
| 4.2 | Animal Information Storage: Store detailed information for each animal, including name/species, level, cost to stop/visit, cost to buy, and owner. | Storing information for each animal (name/species, level, cost to stop/visit, cost to buy, owner) provides detailed attributes for game interactions. This data is essential for managing ownership, costs, and upgrades within the game. |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 4.3 | Animal Purchase: Allow players to purchase animals at their set cost when they land on an unowned animal space. | Allowing players to purchase animals at their set cost when they have no owner introduces ownership and investment mechanics. This feature adds strategic depth, as players must decide when to invest in animals to gain advantages. |  |
| 4.4 | Animal Upgrade: Enable players to upgrade animals to increase their level at the same cost as the purchase price. | Allowing players to upgrade animals to increase their level at the same cost as purchase adds strategic depth and progression. This feature encourages players to enhance their assets, making the game more dynamic and competitive. |  |
| 4.5 | Stop Cost Notification: Inform players of the cost to stop on an owned animal space. | Informing players of the cost to stop on an owned animal space ensures players are aware of financial penalties. This transparency is crucial for strategic planning and helps players make informed decisions during their turns. |  |
| 5. Cards | | |  |
| 5.1 | Card Pack Creation: Create a pack of 20 cards with various scenarios where players gain or lose money. | Creating a pack of 20 cards with various scenarios where players gain/lose money adds random events and unpredictability to the game. This feature keeps the gameplay exciting and introduces new challenges and opportunities. |  |
| 5.2 | Random Card Draw: Implement a feature to randomly draw a card from the pack when needed. | Implementing the ability for a random card to be drawn when needed enhances game dynamics and player decision-making. This randomness adds an element of surprise and requires players to adapt to changing circumstances. |  |

# Design

## What goes in this section?

You need to include any whole project design/planning that your group did as well as the design/planning that you did for your objects – you do not need to include the design/planning that other members of your group did for their objects.

This should include:

* class diagrams
* flowcharts
* pseudocode
* data dictionaries

It should also include your test planning – remember that you need to have iterative testing, system testing and useability testing.

For each of these you need to plan **exactly** what test data should be used and **exactly** what output/result you would expect if that data were input.

Collaboration

A screenshot of a chat

AI-generated content may be incorrect.

We used Trello to coordinate tasks we had to complete and to communicate we used a group chat on teams. Trello was useful as we could see what was and what wasn’t completed, as well as the tasks in progress. (Trello, 2025)

Structure

Working with OOP, naturally divides the problem into a number of classes (smaller problems). This allows classes to be individually coded, allowing for ease of collaboration and less complex code solutions. The program will then operate through a ‘main’ or ‘game’ class which instantiates all objects and uses appropriate methods to create a cohesive program that abides by the success criteria. Here is a structure diagram summarising this explanation.

A diagram of a game

AI-generated content may be incorrect.

GUI plan

We intend to use swing to create a GUI, allowing for a more enjoyable user experience. This interface would consist of a visual board, showing animal names, player tokens and other relevant board squares. In addition, it will include pop up windows that are specific to individual players. This could be the initial process of letting players enter their name and token and also the instructions that correspond to each square, using buttons to allow the user to be able to say yes or no.

Contents:

* login window
* board – including labelled squares with animals, tokens etc…
* instructions
* yes or no input buttons
* roll dice button
* output windows – number rolled, cost of animal, cost of landing, money earned

In regard to accessibility considerations, the GUI will be black and white, leaving no bias to anyone with colour blindness. In addition, to accommodate for those with dyslexia, the font used will be sans serif. This is important as characters look less cursive and more disjointed, which makes them easier to differentiate.

Alternative

In the case that swing seems too difficult or time consuming to implement, there could be a comprise in that only the board would be displayed, and the rest could function through the terminal (if this is even possible). Otherwise, the program may just have to operate through the terminal fully and require that the players use a physical board and follow instructions.

Test plan

The test plan is a way of determining the success of the program, specifically the key functions of the game, defined by the requirements created in the analysis phase. In this case, the key game mechanics need to be tested; this could be rolling the dice, displaying the board or purchasing animals. In addition, the test plan is used to test the programs interactions with inputs, including unexpected ones. The program should validate inputs to ensure that adverse inputs, like using the wrong data type, don’t result in errors. All of this serves the purpose of checking the success of the program, improving usability and improving the maintainability of the code.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test | Input | Output | Expected Output | pass/fail |
| Inputting token | Any text-based char (!,?,/ ...) |  | Accepting token |  |
| Inputting  token | Any numerical  char (1,2,3...) |  | Prompting again |  |
| Inputting  token | More than one character |  | Prompting  again |  |
| Inputting  name | Any sequence of characters |  | Accepting name |  |
| Inputting  name | A sequence including a space |  | Prompting again |  |
| RollDice() | n/a |  | Sets a dice object to a random value 1-6 |  |
| Double roll detection within movePlayer() | n/a |  | If die1 equals die2, the player has the opportunity to collect a card |  |
| Board() - constructor | n/a |  | The constructor assigns an array list of animal objects and other spaces to a board |  |
| DisplayBoard() | n/a |  | The method displays a text based board |  |
| Player location within board | n/a |  | The player location is updated on the board following a dice roll |  |
| GiveInstructions() | n/a |  | The board displays instructions relevant to the square just landed on |  |
| Start space rewards using updateBalance() | n/a |  | The player when passing go or landing on go recieves either £500 or £1000 |  |
| Miss a turn | n/a |  | If a player object lands on a miss a turn square there attribute updates and on the following turn the game checks and skips their turn |  |
| Money management – e.g updateBalance() | n/a |  | A players balance is updated throughout the game |  |
| SetBankrupt() | n/a |  | A player is set to bankrupt and a relevant messaged if displayed |  |
| Winning | n/a |  | The main on each turn checks the bankrupt attribute of each player and determines whether there is a winner |  |
| Animal assignment to board within constructor | n/a |  | The board constructor assigns animal objects to relevant spaces |  |
| Animal information | n/a |  | The information of each animal is assigned to each object during the constructor |  |
| BuyAnimal() | n/a |  | The player is given the opportunity to buy the animal at the corresponding space |  |
| UpgradeAnimal() | n/a |  | The player is able to level up their animal and the cost to stop and level of the animal is updated |  |
| Stop cost notification | n/a |  | The player is notified if the space they land on requires a cost and allows them to pay |  |
| Card array list | n/a |  | The main creates an array list of card objects |  |
| Random card draw | n/a |  | In the event of a double the player can receive a randomly selected card from the pack |  |

Data Structure - Data dictionaries

Game:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Data type | Description for attributes | Methods |
| winner | string | Stores the winner of the game | game() - constructor |
| numOfPlayers | integer | Stores the number of players |  |
| Players | Array list | Stores the players in the board | movePlayer() |
| cards | characters | Stores the cards available | getCard() |
| cardMessages | characters | Stores the messages on each individual card | n/a |
| n/a | n/a | n/a | generateBoard() |

Dice:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Data type | Description for attributes | Methods |
| Value | Integer | Value of first die 1-6 | Dice() - constructor |
| n/a | n/a | n/a | rollDice() |

Board:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Data type | Description for attributes | Methods |
| board | Array list | Stores all 26 spaces | Board() - constructor |
| Players | Array list | Stores a list of all players | See Game |
| MissGo | Integer | Stores index of space missed | Missgo() |
| n/a | n/a | n/a | GiveMoney() |
| n/a | n/a | n/a | displayBoard() |
| n/a | n/a | n/a | Give Instructions |

Player:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Data type | Description for attributes | Methods |
| name | String | Stores players name | getName() |
| token | character | Stores chosen symbol piece | getToken()  setToken() |
| location | Integer | Stores position on board 0-25 | getlocation()  setlocation() |
| Balance | Integer | Stores current money held | UpdateBalance() |
| Bankrupt | boolean | Set to true if player has lost the game | getBankrupt()  setBankrupt() |
| missGo | boolean | Set to true if player misses a turn | getMissGo() |
| n/a | n/a | n/a | buyAnimal() |
| n/a | n/a | n/a | UpgrageAnimal() |

Animal:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Data type | Description for attributes | Methods |
| owned | String | Stores animal name | setOwned()  getOwned() |
| owner | String | Stores animal species | setOwner()  getOwner() |
| level | Integer | Stores current level of animal | getLevel()  setLevel() |
| costToBuy | Integer | Stores purchase cost | getCostToBuy()  setCostToBuy() |
| CostToStop | Integer | Stores cost to stop and visit animal | getCostToStop()  setCostToStop() |
| n/a | n/a | n/a | Animal() - constructor |

Cards:

|  |  |  |  |
| --- | --- | --- | --- |
| Attribute Name | Data type | Description for attributes | Methods |
| value | integer | Stores value of card | Card() - constructor |

Data structure justification:

The structure of the program consists of a series of collaborating classes. The game class contains the logic of the game within a main method while the other classes are used to instantiate relevant objects which are interacted with through the use of methods, including getters and setters. This structure therefore decomposes the problem, allowing for ease of collaboration and more simplistic code solutions. This in addition makes more maintainable code. The data dictionary itself lists the attributes and methods the program will most likely contain. This allows for a more cohesive integration of the program at the end because the contents of each class are clear.

Class diagram – made using smartdraw

(smartdraw, 2025)

A close-up of several blue and white boxes

AI-generated content may be incorrect.

Game class pseudocode

Public class Game

Private string winner

Private int numberOfPlayers

Private arraylist players

Public game()

NumOfPlayers = input(“How many players?”)

For I = 0 to NumOfPlayers

playerName = input(“Name: ”)

playerToken = input(“Token: ”)

players.add(new Player(playerName, playerToken))

Public main()

Game game = new Game()

Board board = new Board(game.players)

Dice dice1 = new Dice()

Dice dice2 = new Dice()

While winner == “”

For I = 0 to NumOfPlayers

player = players.get(i)

game.rollDice(player)

board.giveInstructions()

animal.upgradeAnimal()

public rollDice(Player player)

roll1 = dice1.roll()

roll2 = dice2.roll()

move = roll1 + roll2

// change players board location

print(“You rolled a “ + roll1 + “and a ” + roll2 + “. Move ” + move )

if roll1 == roll2

game.collectCard(Player pl ayer)

public collectCard(Player player)

card = input(“Do you want to draw a card? (yes or no): ”)

if card = yes

Card card = new Card()

If card.value < 0

player.lowerBalance(card.value)

if card.value > 0

player.raiseBalance(card.value)

Flowchart for Game

A diagram of a game

AI-generated content may be incorrect.

## Checklist

This is the top band marking criteria for the design section of your programming project – you should aim to tick off every statement on this list. You should also be clear about what evidence you have included for each statement and what page of your document this evidence is on.

|  |  |  |  |
| --- | --- | --- | --- |
| **I have…** | **y/n** | **As evidence, I have included…** | **On page…** |
| Broken the problem down systematically into a series of smaller problems suitable for computational solutions, explaining and justifying the process | y | A written explanation alongside a structure diagram | 9 |
| Defined in detail the structure of the solution to be developed. | y | A written explanation alongside a structure diagram | 9 |
| Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem. | y | My design section contains the algorithms for game only, but the connection to other algorithms is explained, forming a complete solution | 18,19,20 and 9 and 16 for the connection to other classes |
| Described, justifying choices made, the usability features to be included in the solution. |  | There is a GUI plan, which details usability features | 10 |
| Identified and justified the key variables / data structures / classes (as appropriate to the proposed solution) justifying and explaining any necessary validation. | y | There is a data dictionary for each class, alongside an explanation and justification at the end | 14,15,16 |
| Identified and justified the test data to be used during the iterative development of the solution. | y | There is a test plan proceeded by an explanation and justification | 11,12,13 |
| Identified and justified any further data to be used in the post development phase. | n | This wasn’t something we considered and will think to do so next time |  |

# Development

## What goes in this section?

This section is about explaining the process you went through to develop the program – you should be taking regular screenshots of your **fully annotated** code and then adding this to your development section with information about what you did and why you did it.

You also need to show evidence that you were regularly running and testing your programming – you should have an iterative test plan to follow but this will/should change and evolve as you are programming, and you start to encounter issues you didn’t predict.

It is **super** important that you include evidence of any failed tests, what you changed to make your code work and **why** these changes were important/the best solution for that situation. If you try to run your code and it doesn’t run then take a screenshot, add it to your development section, make the change, explain how you solved the problem, then add a screenshot of the fixed code.

Developing game class

I started by declaring attributes and creating the game constructor.

A screen shot of a computer program

AI-generated content may be incorrect.

This was the initial part of the code. For this portion of the code, I had to reference a previous homework repository for Java - (Github, java-homework-BHASVIC-DylanBarnard24, 2025). The purpose of this was to remind myself of the syntax for array lists and scanners. Initially, I was unsure of the data type the array list would contain, but then realised that you could specify objects or a class, which is the same as using Integer or String (in this case Player).

The next process was then, using the number of players, iterating that many times to instantiate all players.

A computer screen shot of a program code

AI-generated content may be incorrect.

This was my final code for the constructor, but to reach this I had to correct a few errors. At first, using the for loop, I had simple outputs and inputs as follows:

A screen shot of a computer code

AI-generated content may be incorrect.

This however was naive. The main concern was that to properly execute the player constructor, the token attribute had to be a char. As such, I thought to use a while loop to check whether the length was 1. In addition, I wanted to add try and catch, which I found in a previous Java task repository - (Github, BHASVIC-DylanBarnard24-Jack-Java/java fundementals/Bank.java, 2025). This was important because if the token variable was longer than 1 before it was casted to a char then there would be an error. The final problem raised in this portion was casting the token string of length 1 to a char. I initially attempted to use parse, thinking that there might be a parsechar() method. Upon finding out this wasn’t possible, I searched ‘how to cast a string to a char in java’. I found a relevant geeks for geeks page that offered a number of solutions - (geeksforgeeks, 2025). Due to the simplicity of the task as the string would already be length 1 using charAt(0) seemed most appropriate.

Running the code produces:

A screenshot of a computer program

AI-generated content may be incorrect.

In addition, an incorrectly inputted token results in another prompt:

A screenshot of a game

AI-generated content may be incorrect.

Next, I began the instantiated of all objects to prepare for the game. This all operated in a main method. I produced this code.

A computer screen shot of text

AI-generated content may be incorrect.

The code itself is fairly simple. During the process, I realised that I needed to create arraylists for both the card and animal class. This is because, in the case of the card class, the objects themselves need to be stored outside of the class as each object needs to be individual and can’t contain a list of other card objects within itself. The other query came when wondering how to pass in the arraylists to be board constructor. Initially I thought I would need getters to return the value of the lists, but I ended up realising that because the main method was within the game class that the instantiated game object was able to access the private attributes regardless.

The next process was working on the task of granting players their turn.

A computer code with text

AI-generated content may be incorrect.

The first while ensures that the game ceases when there is one player left. The winner attribute will take a value once all but one player is bankrupt; this will be determined at the end of the for loop once all players have taken their turns.

The for loop is then used to allow all players to have a turn through the use of indexing. This uses the .size() method to determine the number of players in the arraylist (the number of players).



The corresponding player is then accessed using the value of i as it updates and the get() method associated with an arraylist.



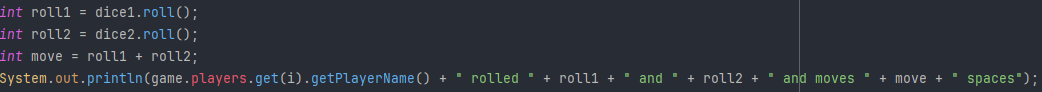
Once within the for loop, there are some initial checks to determine whether the player is valid to take their turn. If a player has to miss a go or is bankrupt, they are no longer able to take their turn.

A computer screen shot of text

AI-generated content may be incorrect.

This is determined using selection. Following the checks there is an else clause, which contains the logic for a players turn, as they have passed the checks.

The turn then starts with the player rolling their dice.



The program in the final statement outputs the moves the player should make. Following this, the rolls are compared to determine whether a double is rolled, in which case the player has the opportunity to draw a card. Thus, I developed a collectCard() method.

The method takes the parameters roll1, roll2 and playerName.



However, once finishing the method, I realised that the method didn’t require the parameters as it didn’t need to be called unless the values were equal.

A screen shot of a computer program

AI-generated content may be incorrect.

Within the method, the player is able to input whether they want to collect a card. To prevent errors, I decided to use input validation, which follows the same structure as that of the one used in the player constructor. The method prompts the player to enter 1 or 2 to choose. If they were to enter anything else, with the combination of the while loop and the try and catch, the method outputs a message and prompts again.

A screen shot of a computer code

AI-generated content may be incorrect.

Once a valid input is received, a Boolean is returned depending on whether they want to draw a card.

Within the main method, the method is called using an if statement.

A screen shot of a computer code

AI-generated content may be incorrect.

The if statement consists of only the method call as it returns a Boolean. Within the selection, a random number is generated to allow for a random card to be drawn. For this, I had to search ‘how to generate a random number in java?’. I found a relevant w3schools page - (w3schools, 2025). Using this page, I coded this following line.



Using this, the card can be found using the indexing of the cards arraylist. Then, the appropriate message can be output and the players balance can be updated, with getters and setters.



This results in an output like this:

A screenshot of a computer screen

AI-generated content may be incorrect.

The next thing I worked on was updating the instantiation of the animal objects. Initially, I had a for loop and just instantiated with no parameters.

A computer screen shot of text

AI-generated content may be incorrect.

However, the animal objects require distinct species and costs so this didn’t work. I thought the best solution would be to create arrays and use a for loop to iteratively instantiate, producing this code.

A screen shot of a computer screen

AI-generated content may be incorrect.

A black background with numbers and symbols

AI-generated content may be incorrect.

The instantiation then took the indexed parameters.



I also realised that there needed to be 24 animals rather than the initial 20.

The final portion of the game class was the method calling to complete the turn.

A screen shot of a computer screen

AI-generated content may be incorrect.

The upgradeAnimal() method was originally going to be an animal class method, but I realised an important aspect of the method was having a list of all the animal objects, which isn’t possible in the animal class. Therefore, I developed it in the game class.

A screen shot of a computer program

AI-generated content may be incorrect.

The first process of the method is determining the animal objects owned by an individual player, so they can only upgrade animals they own. To do this I created two arraylists – one for the name/species of the animal and one for their levels. Then, using a for loop, I added any animals owned by the player to the arraylists using selection.



This line, with so many method calls, seems complicated, but can be easily explained. The animals.get(i) simply returns the animal object at the given index. Then, the getOwner() method is called to return the owner attribute of the animal object, which should contain the player name. Finally, the this returned value is compared to the player’s name with the .equals() method to compare strings.

The next portion deduces whether the player wishes to or is able to upgrade an animal.

A screen shot of a computer program

AI-generated content may be incorrect.

The selection at the start checks whether the balance is less than 200. This is because the player needs to pay 200 to be able to upgrade their animal. Next, using the same try and catch structure the player is prompted with a choice and the upgrade variable is updated accordingly.

Following the decision the upgrade process begins.

A screen shot of a computer program

AI-generated content may be incorrect.

The selection at the start determines whether to start the upgrade. The first check is that the size of the animalOwned arraylist is greater than 0 to ensure that the player has an animal to upgrade. The second is that they decided they wanted to upgrade an animal.

First, using a for loop, the animals owned by the player are listed so they can decide.

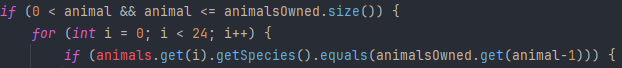
A screenshot of a computer program

AI-generated content may be incorrect.

The player then enters the number of the animal they wish to upgrade. To ensure the player entered a valid number, selection is used.



Following this, using the animal arraylist, the animal object is found according to their species – this is a similar process to the creation of the arraylist at the start of the method.



At the end of the selection, the animal is indexed accordingly as when the user was prompted, they had the option of 1 to the number of animals they own.

The next and final process is making sure that the animal isn’t already max level and if it isn’t the animal is upgraded, and the balance is updated.

A screenshot of a computer screen

AI-generated content may be incorrect.

I decided to add a method to reset the animals of any player that goes bankrupt, as I you don’t other players may end up having to pay the bankrupt player by landing on their animal.

A computer screen shot of text

AI-generated content may be incorrect.

This was a fairly simple method; it loops through the animals checking the owner. If the owner matches, the animal is ‘reset’ by using getters. The selection is used to reset the cost to stop. This works as following. If the animal is levelled up, the cost to stop is doubled, therefore the cost to stop has to be divided by a power of 2, according to their level.

Within the board class, I updated the giveInstructions() method with more functionality. To understand the method, first you must understand the logic of the board class itself.

A screenshot of a computer code

AI-generated content may be incorrect.

The attributes are made up of the players and animals arraylists created during the game constructor and then two more lists that make up the board. The board list keeps track of each players position on the board and the animalBoard arraylist keeps track of the animals. The animalBoard also contains null values at index 0 and 13, which are start and miss a go spaces.

A screen shot of a computer program

AI-generated content may be incorrect.

The method itself is large but follows simple logic. The initial lines are used to determine the nature of the space a player has landed on.

A screen shot of a computer code

AI-generated content may be incorrect.

The players index is found by using the location getter within the player class. Then the player is given instructions according their location. For example, is they are at index 13, they miss their next turn – using a setter.

A screen shot of a computer program

AI-generated content may be incorrect.

The else statement occurs in the case that the player is on any other indexes and that they are therefore on an animal. Within this, the animal object is checked to see if it is owned. If it isn’t the player is informed of the animals information and if they have enough money they have the opportunity to buy it using the purchaseAnimal() method.

A screen shot of a computer code

AI-generated content may be incorrect.

Within the next else clause, the animal object has an owner. There are then two more cases: it is owned by the current player or owned by another player. If the current player owns it they have no action to take. Otherwise, the player has landed on another players animal and they must pay that player.

A screen shot of a computer program

AI-generated content may be incorrect.

Upon writing this, I’ve seen that I used the wrong getter for the animal. I used getCostToBuy(), but the method should be getCostToStop() because they aren’t buying that animal but stopping on that space.



The getCostToStop() is times by negative 1. This is to make the value be subtracted form the player balance because the format of updateBalance() is += so the value has to be negative.

Here are some examples of the methods outputs.

Landing on an animal:

A screenshot of a computer

AI-generated content may be incorrect.

Landing on miss a go:

A screenshot of a computer

AI-generated content may be incorrect.

Landing on the start space:

A screenshot of a computer

AI-generated content may be incorrect.

Now that I’ve combined all classes, I started running the full program. While doing this I encountered the following error.

A screenshot of a computer

AI-generated content may be incorrect.

I think I’ve identified the error; the problem was a syntax error within the board constructor.

A computer screen shot of a program code

AI-generated content may be incorrect.

The line I had initially was this:



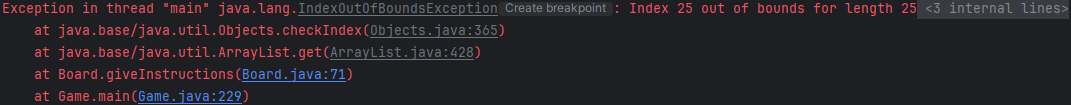
The problem that I think was occurring from this was that the index 13 shouldn’t be included and the operator should be greater than only (>), which is what I’ve changed in the code above. I believe that this created an error because there are 24 animals and if the index 13 is included they take up all indexes from 1 to 24. This therefore didn’t create the 25th index of animalBoard. Thus, if a player reached the 25th index of board, which does exist, there will be an index error because that index is supposed to be accessed in the animalBoard to find the relevant animal. As such, I believe that making the operator exclusive rather than inclusive should fix the problem as it should therefore create the 25th index containing an animal object that can be accessed.



(the line the error originated)

The test I did to identify if the problem was fixed was printing the 25th index of animalBoard. However, this again produced the same error.





I think I’ve realised the error in this case. That is a very similar issue.



The problem with this line was that in this case it should be inclusive of index 13. This is because the animals are added to board after index 0 is assigned to null and I accordingly added them indexed down.



This was the right idea, but it meant that the inequality had to become inclusive because to add index 12 at the right time the selection should include 13 (13-1=12).

Unfortunately, this again didn’t fix the problem. The problem was that the when index 13 was reached the else no longer occurred and therefore the null has to be added within the if.

A computer screen shot of a code

AI-generated content may be incorrect.

This worked as when I printed the 25th index the error was no longer produced and the animal (platypus) was outputted in the terminal.

The final portion of code the game required was a display board method. However, this method created some difficulties.

I started the method by creating a board arraylist.

A screen shot of a computer code

AI-generated content may be incorrect.

I also created a temporary arraylist. The purpose of this is to manage the event that there are multiple players at a single space. The board list is populated using a for loop. Within this loop, there is another loop to iterate through the players of the game. This is to accommodate for the previously mentioned scenario. Following the second for loop is a series of selection statements to add different strings to the board depending on the state of the space.

A screen shot of a computer program

AI-generated content may be incorrect.

The selection starts with an if statement determining whether there are any players on the current space.

A computer screen shot of text

AI-generated content may be incorrect.

The for loop is used to create a concatenated string of player tokens if there is more than one player. After this, selection is used to format the string to look good when the board is output.

A screenshot of a computer program

AI-generated content may be incorrect.

Within the else the rest of the spaces are populated. The selection the index is checked to determine whether the space is a special space like start or miss a go. As such, the indexes 0 and 13 are checked and an appropriate string is added. Otherwise, the current index is added to the board to make the spacing easy to identify.

A screen shot of a computer program

AI-generated content may be incorrect.

The final part of the method is now printing the populated board. The horizontal portions of the board are printed using for loops for efficiency, but I couldn’t identify a method to print the vertical in the same way, so I printed those lines individually.

Here is the output of the board:

A black background with numbers

AI-generated content may be incorrect.

There is one issue; if there were more than 3 players at a single space, which is possible, but very unlikely, the spacing would misalign for a single portion of the board. But because the is very unlikely, and I couldn’t determine a simple solution, I felt this was acceptable.

## Checklist

This is the top band marking criteria for the development section of your programming project – you should aim to tick off every statement on this list. You should also be clear about what evidence you have included for each statement and what page of your document this evidence is on.

|  |  |  |  |
| --- | --- | --- | --- |
| **I have…** | **y/n** | **As evidence, I have included…** | **On page…** |
| Provided evidence of each stage of the iterative development process for a coded solution, relating this to the breakdown of the problem from the analysis stage and explaining what I did and justifying why | y | The development log details my complete code development process. This includes explaining and justifying the important aspects of my code, with the use of screenshots and subsequent paragraphs. | Throughout the development log |
| Provided evidence of prototype versions of my solution for each stage of the process. | y/n | There are examples of aspects of my methods I have changed to improve my code solution. | 39 |
| Ensured the solution is well structured and modular in nature. | y | The program is well organised into different classes which the game class can combine using getters, setters and other methods, within a main method. | There is evidence of the game class collaborating with other classes throughout the development log but the general structure I better defined in the design phase as the development log only contains my code. |
| Annotated the code to aid future maintenance of the system. | n | I didn’t annotate my code that I created, but I did explain the purpose of my code within the development log, but not necessarily line by line. This is something I should of done and will remember to do so in the future. |  |
| Ensured all variables and structures are appropriately named. | y | Every variable used has a descriptive and relevant name, using camel case. | Evidenced throughout every piece of code |
| Included evidence of validation for all key elements of the solution. | y | There is input validation for many of the inputs, using try and catch. It is included in the following methods: game(), upgradeAnimal(), collectCard() | 24,28,31 |
| Shown my review at all key stages in the process. | n | There are examples of summations of my methods and things of that nature, but I wouldn’t say I was reviewing my code at all key stages in the process. |  |
| Provided evidence of testing at each stage of the iterative development process. | y/n | The nature of the class I worked on game didn’t constitute much iterative testing for some of the methods as they won’t work until all the code is combined, however for methods like rollDice(), the player constructor and giveInstructions() there is testing during the process, including output screenshots | 25,29,38-39 |
| Provided evidence of any failed tests and the remedial actions taken with full justification for any actions taken. | y | There is a section of the development log that evidences this: page 40-42 | 40-42 |

# Evaluation

## What goes in this section?

This section is about showcasing your system and usability testing results – it’s ok if your program doesn’t fully work (it probably won’t!).

Wherever possible the evidence in your testing table should be a screen recording of the program in action. The videos should be stored in the same folder as this document.

If it doesn’t fully work, then you actually have more to write about and show your evaluation skills – make sure that you comment on why you were unable to get the program fully working and how you could meet the missing criteria through further development.

Make sure to also think about whether the code you have created would be easy, difficult or somewhere in the middle for somebody else to maintain – have you coded in a weird way? Could you have been more modular? Used more inheritance?

Test plan

For the output portion of the test plan, there are many tests that use a screen recording for evidence. These screen recordings can be found in a file within the repository, and are identified by the stated name.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Test* | *Input* | *Output* | *Expected Output* | *pass/fail* |
| Inputting token | Any text-based char (!,?,/ ...) | Moves on to next player/accepting token | Accepting token | pass |
| Inputting  token | Any numerical  char (1,2,3...) | Moves on to next player/accepting token | Prompting again | pass |
| Inputting  token | More than one character | Invalid token and prompts again | Prompting  again | pass |
| Inputting  name | Any sequence of characters | Accepts and moves on | Accepting name | pass |
| Inputting  name | A sequence including a space | Moves on after accepting | Prompting again | fail |
| RollDice() | n/a | Works correctly | Sets a dice object to a random value 1-6 | pass |
| Double roll detection within movePlayer() | n/a | Outputs ‘you rolled a double’  Prompts for drawing a card | If die1 equals die2, the player has the opportunity to collect a card | pass |
| Board() - constructor | n/a | Works correctly | The constructor assigns an array list of animal objects and other spaces to a board | pass |
| DisplayBoard() | n/a | screen recording 1 | The method displays a text based board | Pass |
| Player location within board | n/a | screen recording 1 | The player location is updated on the board following a dice roll | pass |
| GiveInstructions() | n/a | screen recording 1 | The board displays instructions relevant to the square just landed on | pass |
| Start space rewards using updateBalance() | n/a | Landing on start - (screen recording) +screen recording 1 | The player when passing go or landing on go recieves either £500 or £1000 | Pass |
| Miss a turn | n/a | player going bankrupt and player missing turn  (screen recording) | If a player object lands on a miss a turn square there attribute updates and on the following turn the game checks and skips their turn | Pass |
| Money management – e.g updateBalance() | n/a | screen recording 1 | A players balance is updated throughout the game | pass |
| SetBankrupt() | n/a | player going bankrupt and player missing turn  (screen recording) | A player is set to bankrupt and a relevant messaged if displayed | pass |
| Winning | n/a | winning  (screen recording) | The main on each turn checks the bankrupt attribute of each player and determines whether there is a winner | pass |
| Animal assignment to board within constructor | n/a | works correctly | The board constructor assigns animal objects to relevant spaces | pass |
| Animal information | n/a | Outputs relevant info – screen recording 1 | The information of each animal is assigned to each object during the constructor | pass |
| BuyAnimal() | n/a | screen recording 1 | The player is given the opportunity to buy the animal at the corresponding space | pass |
| UpgradeAnimal() | n/a | screen recording 1 | The player is able to level up their animal and the cost to stop and level of the animal is updated | pass |
| Stop cost notification | n/a | landing on a players animal  (screen recording) | The player is notified if the space they land on requires a cost and allows them to pay | pass |
| Card array list | n/a | works correctly | The main creates an array list of card objects | Pass |
| Random card draw | n/a | card draw  (screen recording) | In the event of a double the player can receive a randomly selected card from the pack | Pass |

The completed the test plan highlights the success of the program. All tests, but one, were past. These tests were based on the success criteria of the program, so passing these tests can be seen as meeting the success criteria. The one failed test was validation for player names. This wasn’t something I prioritised, especially when there is more important input validation. The test plan itself contains a lack of inputs. This comes down to the fact that the program itself beyond player definition is a series of yes and no answers, so the opportunity for unexpected inputs is slim. However, the test plan still could include more inputs. The program is designed to validate these yes and no answers but the test plan doesn’t test that.

A white background with black text

AI-generated content may be incorrect.

This is a list of success criteria. All these criteria are fully met and are evidenced in the test plan, besides one, which is partially met. This is the animal upgrade. This was successfully created, however doesn’t exactly meet the criteria as an animal is supposed to be upgraded at the same cost as their purchase, where in the program there is a fixed cost for all animals. This is a problem that could be easily amended due to its simplicity. The code would be altered to, when upgrading the animal, using a getter to get the purchase cost and applying that rather than the fixed cost.

Maintainability

The program itself is fairly robust because of the design, especially as it is object-oriented. The maintainability of the program is complemented by the contents of the design phase and the comprehensive development log, which details the process of developing the program with explanations for maintainability. However, an aspect of this I overlooked was the annotation of my code. This would have provided a more in detailed explanation as it would be line by line. This would help a person follow the code easier and make decisions clearer. This is an important aspect to include in the future. Despite this, I would say the program itself is quite maintainable for the previously mentioned reasons.

Improvements

While testing the program, I identified a case which results in an infinite loop. This occurs when a player chooses to upgrade an animal but all animals they own are max level. This results in the player being prompted repeatedly to choose a ‘valid’ animal, but this can never happen as they are all max level. This problem could be fixed by within the upgrade animal method, including an initial check to ensure that all animals owned aren’t all max level. This could be done using iteration as a list of animals owned by a player is created. Thus, each of these animals could be checked and a flag could be made false if the animals are all max level.

Another improvement, mentioned in maintainability, is the use of comments or annotating the code. The benefit of this would be a more detailed explanation of the code compared to the development log. This would help aspects like maintainability.

The most substantial improve that could be made is the implementation of a GUI. This could significantly improve the user experience and the usability of the game, through a more visual interaction, making processes easier to follow – especially in comparison to long output messages in the terminal.

Usability

The usability of the program was perhaps an aspect that could be improved. The program itself ran fully through the terminal. The original plan had the possibility of a GUI, which would have provided better usability with the addition of buttons and a more visual representation of board. However, this didn’t come to fruition. This was because of time restrictions and a lack of knowledge of the implementation of a GUI. During the development of the code, I spent my time ensuring that the game would work in the terminal rather than thinking about how it could work with a GUI. This meant the GUI became less feasible, as the process would include altering previous code while implementing the GUI.

However, this being said, the program does operate well enough in the terminal, displaying relevant messages on player turns and displaying a text-based board. In addition to this, there is input validation to account for misinputs and things of that nature. In summation, the usability of the program could be improved but for the circumstances there were steps taken to make it sufficient. Further evidence of the usability can be found in the screen recordings of the program.

Text-based board:

A black background with numbers

AI-generated content may be incorrect.

Instructions/outputs:

A screenshot of a computer

AI-generated content may be incorrect.

## 

## Checklist

This is the top band marking criteria for the evaluation section of your programming project – you should aim to tick off every statement on this list. You should also be clear about what evidence you have included for each statement and what page of your document this evidence is on.

|  |  |  |  |
| --- | --- | --- | --- |
| **I have…** | **y/n** | **As evidence, I have included…** | **On page…** |
| Provided annotated evidence of post development testing for function and robustness | y | There is a test plan and surrounding paragraphs. | 49-52 |
| Provided annotated evidence for usability testing. | y | There is a test plan, including screen recordings that highlight program usability. | 49-52  + recording folder in repo |
| Used the test evidence to cross reference with the success criteria to evaluate the solution explain how the evidence shows that the criteria has been fully, partially or not met in each case. | y | There is a paragraph referring to success criteria after the test plan. | 51,52 |
| Provided comments on how any partially or unmet criteria could be addressed in further development. | y | There is a paragraph referring to success criteria after the test plan. | 52,52 |
| Provided evidence of the usability features justifying their success, partial success or failure as effective usability features. | y | There is a usability section in the evaluation | 52,53 |
| Provided comments on how any issues with partially or unmet usability features could be addressed in further development. | y | There is a usability section in the evaluation | 52,53 |
| Considered maintenance issues and limitations of the solution. | y | There is an improvements section in the evaluation | 52 |
| Described how the program could be developed to deal with limitations and potential improvements / changes. | y | There is an improvements section in the evaluation | 52 |
| There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated. | y | The evaluation is split into distinct and thorough explanations. |  |

# References

## References Explanation

You **must not** take anything you find online and include it (even with changes) directly into your project – you should be reading through the sources you use and using the knowledge gained to solve the problems.

The penalties for plagiarism can range from zero marks for your coursework to disqualification from **all** your A Levels (and everything in between) depending on the severity of the plagiarism.

To clear, it’s ok (encouraged even) for you to access websites books, and other resources (as long as they **aren’t** AI bots), when you get stuck as long as you use the knowledge acquired to then independently solve your problem.

You should try figuring out the problem first but if there is a pesky syntax error then you can’t solve then try and find a solution elsewhere. You just need to reference **every** source you look at.

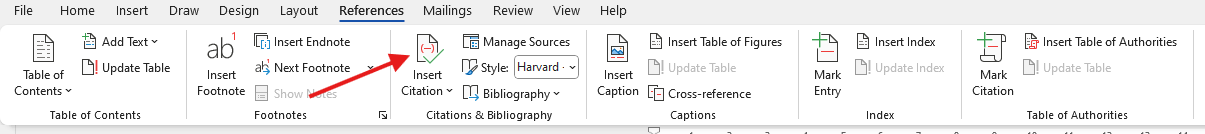
As you are writing this documentation you should include details about **every** website you access, **every** book that you read, and **every** other source that you access that can’t be so easily categorised.

Even if you don’t feel like you learned anything from the source, you **must** mention that you looked at it while trying to find the answer to something.

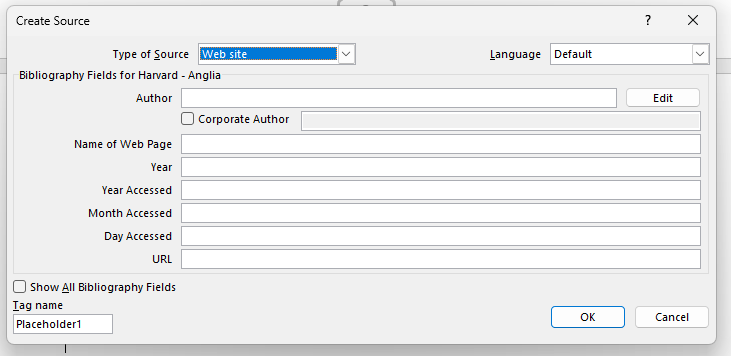
The easiest way to track this is to use the inbuilt References tab in Word – when you ever you need to cite that you’ve accessed a website you click the icon shown on the next page, select the type of source and the fill in the form – this will cause something like this: (Zucker, 2025) to appear in your workand the References section (below) to automatically update with details of the source.

If you reuse a source, you’ve already added then it appears on a drop-down menu to save time.

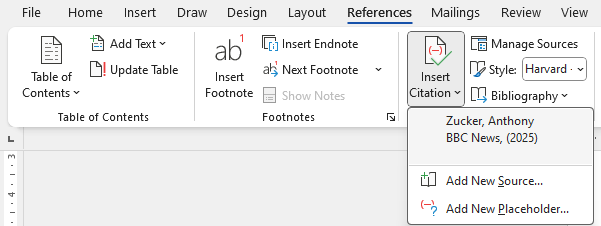
**References Section**



**Source Form**



**Reusing a Source**



# Reference List

geeksforgeeks. (2025, May 20). *How to Convert a String to a Character in Java?* Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/how-to-convert-a-string-to-a-character-in-java/

Github. (2025, May 20). *BHASVIC-DylanBarnard24-Jack-Java/java fundementals/Bank.java*. Retrieved from Github: https://github.com/BHASVIC-CompSci/BHASVIC-DylanBarnard24-Jack-Java/blob/main/java%20fundementals/Bank.java

Github. (2025, May 20). *java-homework-BHASVIC-DylanBarnard24*. Retrieved from Github: https://github.com/BHASVIC-CompSci/java-homework-BHASVIC-DylanBarnard24/blob/main/Main.java

smartdraw. (2025, May 15). *UML class diagram creator*. Retrieved from smartdraw: https://app.smartdraw.com/editor.aspx?templateId=3f94c5ba-46fb-46ac-80e2-8b13581b08db&flags=128

Trello. (2025, May 15). *Trello workspace*. Retrieved from Trello: https://trello.com/b/P5zmAI27/animalopoly

w3schools. (2025, May 26). *Java How To Generate Random Numbers*. Retrieved from w3schools: https://www.w3schools.com/java/java\_howto\_random\_number.asp

Zucker, A. (2025). *BBC News*. Retrieved April 29, 2025, from https://www.bbc.co.uk/news/articles/c5ypz7yx73wo

# AI References

## AI References Explanation

**AI should be your absolute last resort.**

As with other sources, any part of your work that comes directly from AI **will not be marked**, the big difference is that any work that is improved by AI **will also not be marked.**

This is not limited to code generation, if you use AI to help with any section of your project (Analysis, Design, Development, Evaluation) then you **need** to cite it and make it clear what was your work and what work was create via assistance from AI so that only **your** work is marked.

Examples:

* Using AI to reword your work – we will mark the original work shown in the prompt, not what the AI has generated.
* Generating code using AI – this should not even be considered as code generated via AI will not be marked
* Debugging code using AI – we will mark the original work shown in the prompt, not what the AI has debugged

As a result, citing AI sources is a little bit harder than a “normal” reference.

You need to include in your write up something like: “I was struggling to get the dice rolling mechanic to work and had to use ChatGPT to debug the code I had written.”

(ChatGPT 3.5 (https://openai.com/blog/chatgpt/), 25/01/2024.)

They key information here is

* The name of the AI bot
* The date the content was generated
* How you used the response

You should then include in your Refence Section screenshots of your **full** conversation/interactions with the AI.

* The prompt used
* The AI response

## AI Reference List

# Referencing FAQs

## What happens if I don’t cite a source I’ve used?

That won’t happen… you are going to be screenshotting your project so frequently that you’ll always have your documentation open, so you’ll just record that you’ve accessed the site as accessing the site.

With non-AI sources, if you’ve got dozens of sources and one of them is accidentally missing, then you’ve clearly shown understanding of the importance of referencing and that you’ve taken it seriously – that’s what matters, you won’t be penalised.

If you deliberately don’t cite a source, then it will be treated as plagiarism and investigated.

With AI sources, if you don’t cite AI usage then we will notice (it’s more obvious than you think) and we will have to investigate.

## What if I don’t realise, I’ve plagiarized?

What a stupid question!

If you can answer no to **all** these questions, then you haven’t plagiarized.

* Have you directly copied from a source?
* Have you copied from a source and changed the words?
* Have you deliberately not included a source for any reason?
* Have you used AI and not sourced it?

## What are the potential consequences of plagiarism?

All suspected plagiarism is fully investigated and will involve meeting with your class teacher and the Head of Department.

If you are found to have plagiarised, then the consequences will depend on the severity of the plagiarism but could include:

* Receiving zero marks for a section of your work
* Receiving zero marks for the entire coursework unit
* Disqualification from your OCR A Level Computer Science course
* Disqualification from **all** OCR qualifications – including in the future
* Disqualification from **all** your A Levels (extremely rare).

## If AI generated work isn’t marked, then why use it?

That’s sort of the point… but seriously there might be a very specific issue that you cannot solve that is preventing you from moving forward and accessing marks in other sections of the mark scheme. In this situation you should make sure that you have exhausted **literally every other option** before using AI.