

BHASVIC Computer Science

Animalopoly

Jack Griffiths
[Date]

Contents

Analysis2

2

3

7

7

22

24

24

50

51

51

52

53

53

Reference List55

57

57

57

58

58

58

58

59

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:

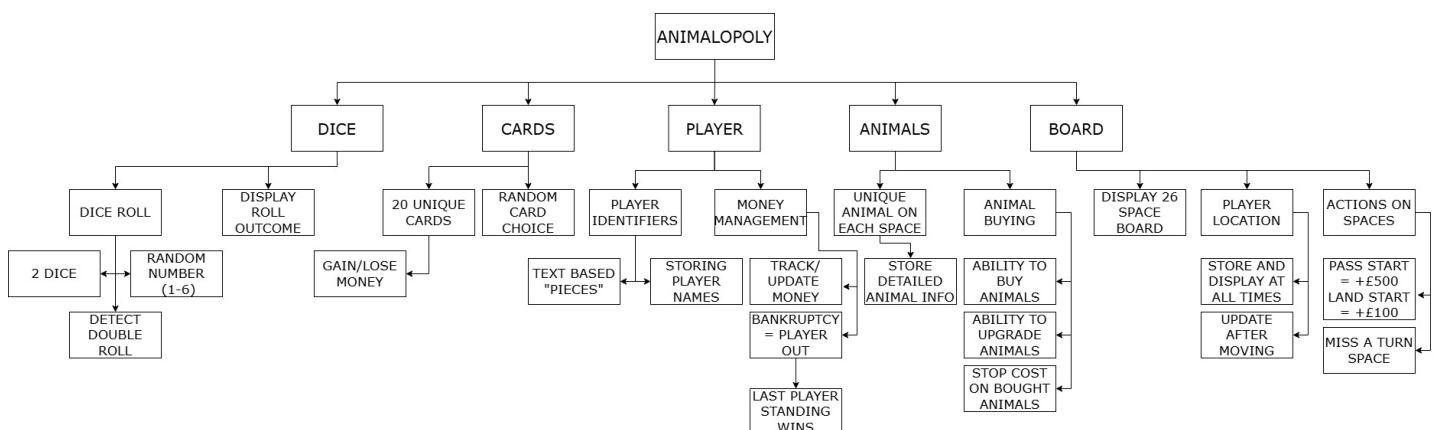
- class diagrams
- flowcharts
- pseudocode
- data dictionaries

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.

Whole group design:

Structure diagram:



As Animalopoly is divided into 5 main success criteria (the dice, cards, players, animals and the board) we initially decided to split the program into separate object orientated classes of those 5 types. This means that the entire program will be modular and can be worked on concurrently by

separate team members based on their strengths. This also increases data security as all methods and attributes of classes are encapsulated meaning that each object only modifies each attribute according to its defined methods in intended ways. Each class is further broken down into the different features they must contain which means when planning and developing a solution each feature of the class can be tackled individually which is easier to do as only small parts of the overall game are designed and developed at once.

The Dice class was identified to have 2 key functions which is to generate a random dice roll and detect if the roll generated was a double roll and to return and display on screen the outcome of that roll as defined in the success criteria. The generating of the dice roll can be handled by a suitable method which generates the two numbers and checks if they are equal and if they are then drawing a card and the displaying of the dice roll involves passing the rolled values into the GUI to then be displayed so as these are different processes they should be developed as separate methods.

The Card class has two distinct features which are to store 20 unique cards each containing a different unique effect and then a method to draw a random card when needed and apply the effect to the relevant player. As these are separate problems they can be tackled individually as the 20 cards can be stored in an array which contains the effect and value for each card and the method to draw a random card from the array is a separate problem to be developed after.

The Player class has two main features which are the player pieces/identifiers and the money management for each player. For the player identifiers, at the start of the game, a player object will be instantiated for each player in the game and each player will be asked to

input their name and the piece they wish to use which will then be stored. The pieces that are picked will be displayed on the board in the position of each player and updated when the player moves which will involve communication with the GUI and the player's name will be displayed on their turn. For money management, Each player's money should be stored in their respective object and updated during certain events in the game by a getter/setter method inside the player class. Additionally, each time the player's money is updated it should be checked to see if it is less than 0 and if so the player is removed from the game and is bankrupt. When this happens there will be an additional check of how many remaining players there are and if there is only a single player then that player is the winner and the game ends.

The Animal class has two main features which are to create a unique animal on each space of the board and the ability for individual players to buy/own/improve animals they land on. Each animal will store information such as its name, level, cost, etc and this will be able to be viewed by a player when they land on the animal's space and will be updated as the game progressed by getter/setter methods. When a player lands on an animal space that is not owned by any player, they will have the option to purchase that animal for a cost which will call a method to remove the amount of money necessary from the player object and change the animalOwner attribute to the name of the player who bought it. Any attributes owned by a player may be upgraded during the player's turn which increases the animal's level attribute by 1 and costs the same amount of money as to purchase the animal but it in turn greatly increases the stop cost attribute of the animal which can be done with an upgradeAnimal method inside the animal class. Finally, when a player lands on an animal space of an animal owned by another player, the stop cost of the animal will be removed from that player's money attribute and shall be given to the animal's owner and there will be a notification of the stop cost of an animal space if a player wishes to view it on their turn.

The board class has 3 main features which are to store 26 unique spaces and to display them on screen, Store and update the players' locations as the game progresses and to have two unique start/miss a turn spaces which have different effects. The 26 spaces will be stored in a text based 1D array since the board spaces are linear and can be identified by a single index and each index will store some information about the space such as the current player on the space and the type of space so that the relevant information can easily be retrieved from the board to the GUI to be displayed. The position of each player will be stored in the board in an index and there will be methods to update the player's positions by certain amounts of spaces and when this happens a check will be implemented to see what space the player has landed on and what action needs to be carried out because of that which will be easy to implement as all the relevant information to do that will be contained in the board array. To implement the start and miss a turn spaces their permanent location will also be stored in the board array and when a player's position enters that index if the index contains start then the relevant money will be added to the player's money attribute, if the player passes start the relevant money will also be added to the player's money index and if it is a miss a turn space then a missPlayerTurn method will be called which will mean that the player's next turn is skipped.

Development plan:

Oscar:

1. Creation of the Board class
2. Creation of the Player class
3. Joint Creation of the Game class

The board class is vital for the operating of all other classes, so it is important to be created as early as possible in the development process. The player class is also important for the game, but other classes rely less on it so should be completed afterwards. As both classes are large in scope it is sufficient work to be completed by one person.

Sam:

1. Creation of the Dice class
2. Creation of the Animal class
3. Joint creation of the Game class

The Dice class is relatively small and is important for some other classes so should be completed early in the development process. The animal class is large and important for the functionality of the game so should have a lot of time dedicated to it and be completed after the dice class. As this is one large class and two smaller classes this is a sufficient amount of work to be completed by one person.

Isaac:

1. Design and planning of the GUI
2. Creation of the GUI
3. Creation of the Cards class
4. Implementing the GUI with the events of the game

A graphical user interface is highly important for the accessibility of the game and conveying information in an engaging and understandable way but as it is a large task which involves creating a graphical design for each feature it is a sufficient task to be completed by one person. The Cards class is important to meet success criteria and improve the game, but few other classes rely on it so should be completed towards the end of the development process.

Once all individual classes are made Oscar and Sam will collaborate to create the Game class which controls the events of the game and allows the classes to work together. Additionally, Issac's GUI made using swift will be made to work with the game to update the screen when the game changes.

Post development test plan:

Test description	Test Type	Success criteria	Pass/Fail
Roll two dice and check the number generated.	Normal	Two dice successfully rolled and the result is between 2-12	
See if the total number of spaces to move after the dice roll is displayed to the player.	Normal	The total number of spaces to move will be displayed	
Check if there is a double roll then a special event will occur e.g. a card being drawn.	Normal	A special event occurs after a double roll	
See if there is a board with 26 spaces and that it is displayed onto the screen.	Normal	A 26 space board will be displayed to the users	
Roll the dice a few times and examine the players' positions.	Normal	The players' positions will be stored in the board and updated as the game goes on	
Roll the dice a few times and check if the players' positions on the board are visible.	Normal	All of the players' positions are visible on the board at all times	
Land on a space and view all information given to you about the current space.	Normal	All relevant rules and actions which can be taken on the current space will be presented on the screen.	
Roll the dice enough times to pass start.	Normal	£500 will be given to the player who passes start.	
Roll the dice to exactly land on start.	Normal	£1000 will be given to the player who lands on start.	

Roll the dice to land on the miss a turn space	Normal	The player who lands on the miss a turn space will miss a turn.	
Select unique playing pieces for each player.	Normal	Each player will be allowed to select a unique playing piece to be used during the game.	
Input a player name and play the game as that player.	Normal	The player's name will be stored and displayed at the start of each of their turns.	
Play the game as a player and view the amount of money owned throughout the game.	Normal	The player's money amount will be initialised and updated due to various events throughout the game	
Play the game and land on an animal space which is owned by another person.	Normal	The player's money is decreased and given to another player.	
Play the game and attempt to purchase an animal which cannot be afforded.	Erroneous	This should not be allowed as this would cause the player to become bankrupt intentionally.	
Play the game and be forced into bankruptcy.	Normal	The player should be notified that they are bankrupt and removed from the game.	

Play the game and become the last remaining player with money.	Normal	The player should be announced as the winner and the game should end.	
View the amount of animals on the board.	Normal	There should be 24 unique animals across the board with different attributes/interactions.	
Check the amount of information about each animal.	Normal	There should be information about name/species, level, cost to land on, cost to buy, owner, etc stored about each animal.	
Land on the space of an unowned animal you can afford and attempt to purchase it.	Normal	You should be able to purchase animals on different spaces on the board at their set cost.	
Attempt to upgrade the level of an animal you currently own.	Normal	You should be able to upgrade an animal (which increases the cost to land on it) for the same price as the animal originally cost.	
Check the cost to land on an animal space owned by another player.	Normal	The players should be able to check the cost of landing on another player's animals during their turn.	
Check that there are 20 cards generated to begin the game.	Normal	There should be 20 cards generated, each detailing a random scenario which has some effect on the game likely money gain/loss	

Roll 2 dice until you get a double.	Normal	A random card from the deck should be selected and its effect used on the player.	
-------------------------------------	--------	---	--

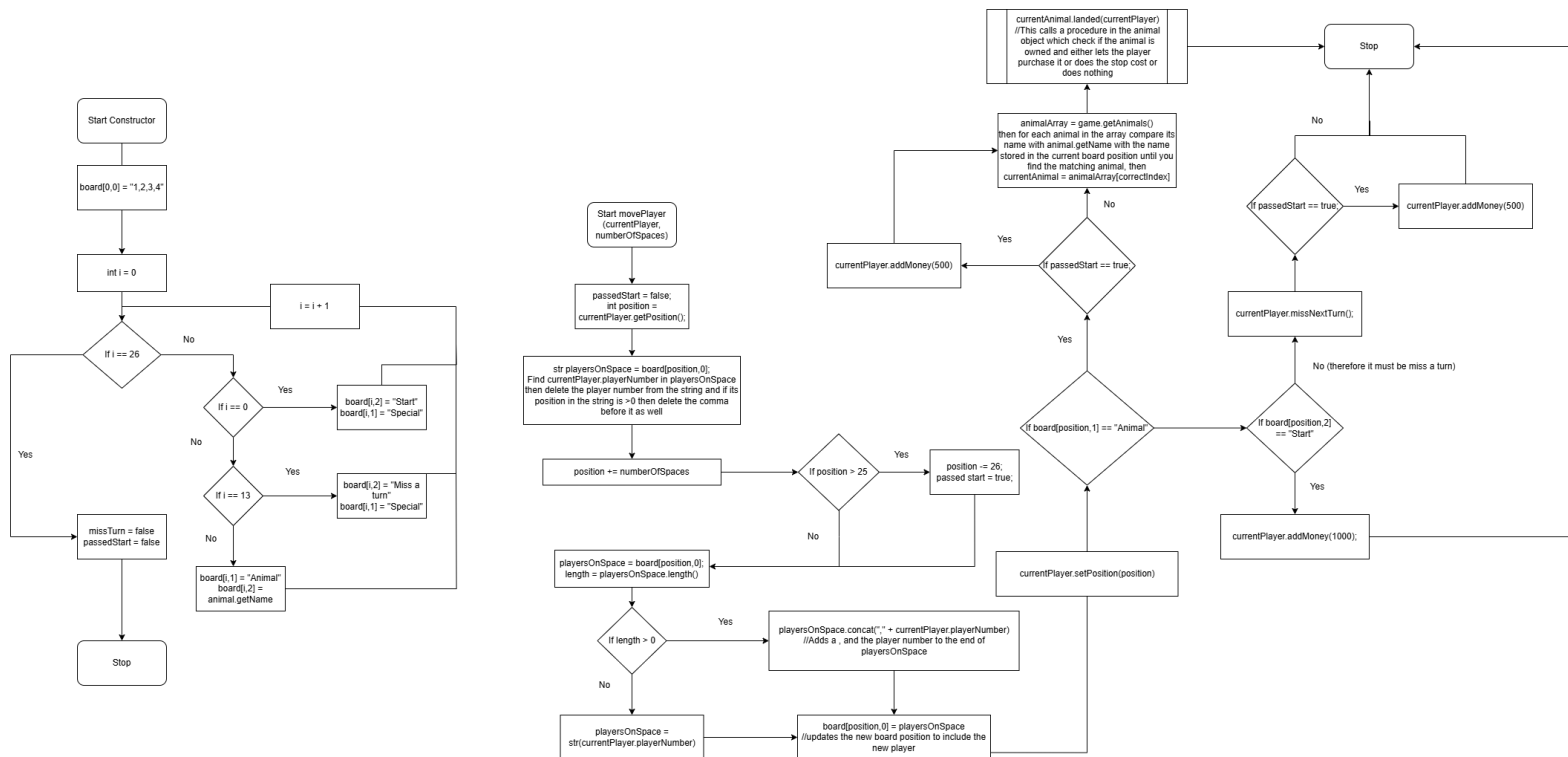
Individual design:

Initially, I followed the development plan and began planning the board class and immediately realised that in order to store all the relevant information about each space inside of the 1D board array it would be difficult to hold separate pieces of data such as the type of space and which players are on it inside of a single index so I decided to change the design of the board to be a 2D array such as the example below:

	[0] (Player positions)	[1] (Space type)	[2] (specific space)
[0] (space 0)	""	"Special"	"Start"
[1] (space 1)	"1,3"	"Animal"	"Elephant"
[2] (space 2)	"2"	"Animal"	"Tiger"

The player positions are stored as a string in each board space which will make it easy to give the positions of each player on each space to the GUI when displaying the board. The space type and specific space are also stored which will make it easy to decide what to do when landing on each space and the animal names can be used to locate each specific animal.

Flowchart for constructor and movePlayer methods for Board class (Sorry text is small you may need to zoom in):



Firstly when planning the board class, I began with the constructor. To initialise the board array firstly the index [0,0] will be initialised to contain the player numbers of all of the players in the game as the game needs to begin with all players on the start space. Then a for loop will iterate through all spaces in the board and instantiate the space type and specific space columns for that space for example making the 0th space Special in the 1st column and Start in the 2nd column and the 13th space will be miss a turn and the rest will be various animals. I wasn't quite sure how to show iteration in a flowchart so I searched how to on BBC Bitesize (BBC, 2025). I chose to do this using a for loop as there are a fixed number of spaces on the board and each unique space has a defined animal/special space which should be stored about the space in the array and for loops can repeat for a set number of times and keep track of which iteration is currently being done. After the board is completely filled initialise the

passedStart and missTurn attributes as false to be changed later if needed and then end the constructor.

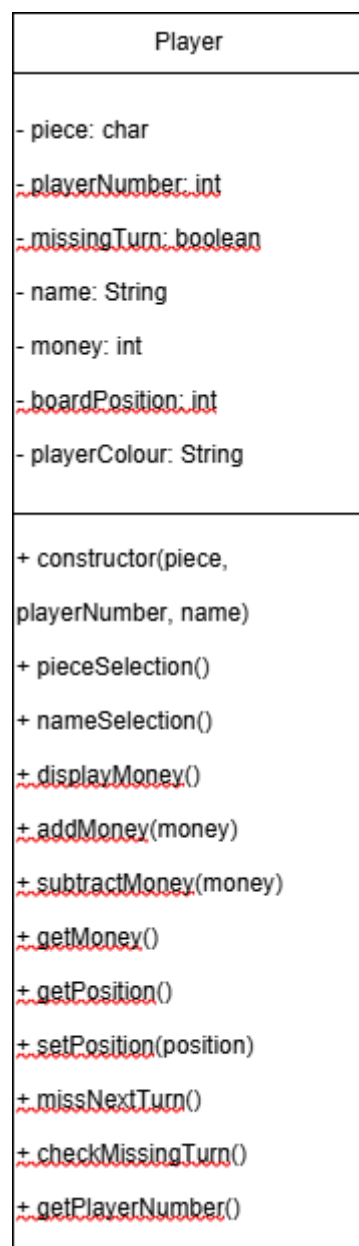
Next I created a flowchart for the movePlayer method which will be called after a dice roll to move the player the rolled amount of spaces across the board which is a key mechanic to be used in the game. Firstly, the player's position is retrieved using the getPosition method from the currentPlayer object. Then the "player positions" column is accessed for the board space that the player is in. The player's player number is found and removed from the string as they will be moving away from this space and any now unnecessary commas are also removed. Next add the number of spaces to move from the dice roll to the position to result in the new player position and if the player position is now greater than 25 then subtract 26 as the player has now passed start and needs to loop round the board again and set the passedStart attribute to true to be used later. Then find the new "player positions" string for the new board position and add the current player's player number into this string and then insert that string back into the board. I was unsure of which string function was used to add two strings together so I found the function using (W3 schools, 2025). Then use the currentPlayer object's setPosition method to set their position as the updated position. If the board position which has been landed on is an animal space, then get the array of animal objects and search through it until the animal with the matching name is found. Then call the landed method of that animal with the player as a parameter which will run the necessary code to deal with a player landing on an animal e.g. letting them purchase it if it is unowned or charging them the stop cost if it is owned by another player. If the board position is a start space, then give the player £1000. If the board position is a miss a turn space, then set the missingTurn attribute in the player object to true which will cause them to miss their next turn. If the player passed start and did not land on it then also give them £500.

Iterative test plan:

Test inputs	Additional code needed	Expected result	Test purpose	Pass/Fail
Input the number of players as 2 in Game class	Output position [0][0] of the board	The string "1,2" is outputted	To determine if a different number of players correctly changes the number of players initialised in the game	Pass
Input the number of players as 3 in Game class	Output position [0][0] of the board	The string "1,2,3" is outputted	To determine if a different number of players correctly changes the number of players initialised in the game	Pass
Input the number of players as 4 in Game class	Output position [0][0] of the board	The string "1,2,3,4" is outputted	To determine if a different number of players correctly changes the number of players initialised in the game	Pass
Input the number of players as 5 in Game class	Nothing	This value should be rejected, and the user should be asked to re-enter the number of players	To ensure there is correct input validation to only allow 2-4 players to play the game	Pass
Input the number of players as 1 in Game class	Nothing	This value should be rejected, and the user should be asked to re-enter the number of players	To ensure there is correct input validation to only allow 2-4 players to play the game	Pass

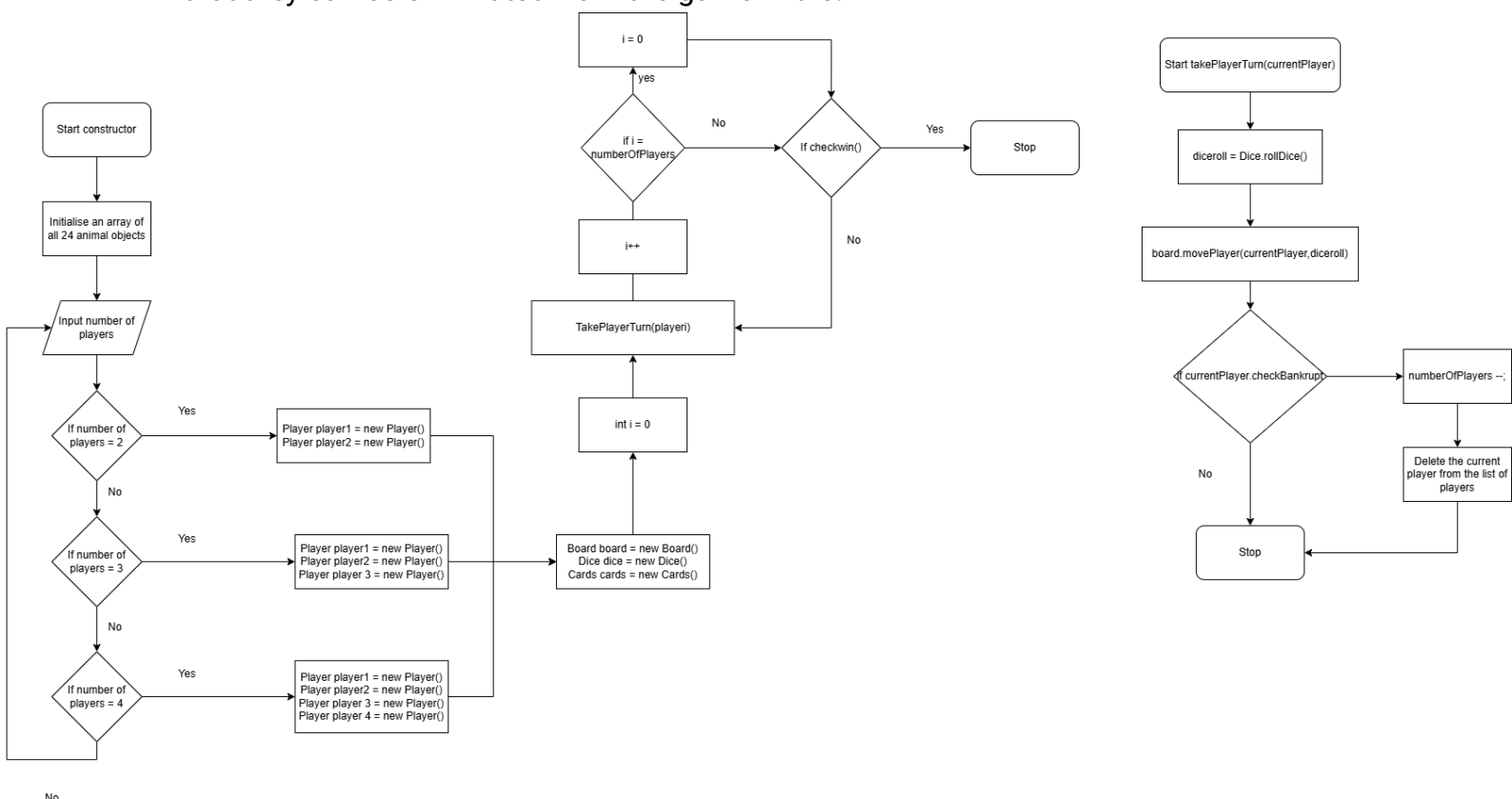
Input the number of players as 2 in Game class	Print(board[0][1])	"Special" printed	To ensure that the start position is classed as a Special space	Pass
Input the number of players as 2 in Game class	Print(board[0][2])	"Start" printed	To ensure that the start position is classed as a Start space	Pass
Input the number of players as 2 in Game class	Print(board[13][1])	"Special" printed	To ensure that the miss a turn position is classed as a Special space	Pass
Input the number of players as 2 in Game class	Print(board[13][2])	"Miss a turn" printed	To ensure that the miss a turn position is classed as a Miss a turn space	Pass
Input the number of players as 2 in Game class	Print(board[1][1])	"Animal" printed	To ensure that the start position is classed as a Animal space	Pass

The player class only needs to store attributes for each player and also contain getters and setters with no other logic, so I created the below class diagram for the player class:



I looked at (Ada computer Science, 2025) to make sure I formatted my class diagrams correctly. The player class has the attributes of the piece which is a symbol to be displayed on the board, the player number which will be used to identify each player separately, the missing turn attribute which will inform the game if the player needs to miss their next turn, their name as a string to be displayed when referencing the player which is much more personal than a player number, their money as an integer which is used to purchase animals and also to see if the player is bankrupt, the position in the board to make it easier for the board class to find and the colour of the player to be outputted on the GUI. It's constructor takes the piece, player

number, player colour and player name as parameters as these are different for each player and the rest of the attributes are initialised to starting values as they always begin the same as the money begins at 1500, missing turn as false and the board position as 0. The add and subtract money methods are used to update the player's money. I decided to use these instead of a setMoney method as the money is never set to a specific value and is only added to and subtracted from and this makes it easier to do so without also having to use the getter to get the money attribute from the player and add the number to that. There is also a getMoney method to retrieve the value of the Player's money. The getPosition method is used to return the position of the player when it is needed such as in the board class in the move player method. The setPosition method sets the position of the player to a new position such as when the player is moved. The setMissingTurn method sets the missing turn attribute to true which skips a player's turn or false which will not miss the next turn. The checkMissingTurn method checks if the current player is missing their next turn or not when the takePlayerTurn method is being run in the game class. The getPlayerNumber method gets the player's playerNumber which is used in the movePlayer method in the board class to update the position of the player. The checkBankrupt method checks if the player's money has dropped to less than 0 so that they can be eliminated from the game if it is.



The Game class is what initialises the game and repeatedly takes each turn. The constructor initialises a 24 index array of each animal that is present in the game.

Then, the number of player is inputted (and validated) and a different number of player objects are initialised depending on the number of players inputted. Initially I decided to leave it there but after thinking further how this would work I decided to also add each player object to an array of players so that they can be cycled through to be passed into the takePlayerTurn method. Next, the board, dice and cards are initialised to be used in the game. Finally, the takePlayerTurn method is run with each player iteratively and repeated until somebody wins where the game stops.

Next, the takePlayerTurn method takes the current player who's turn it is as a parameter and rolls a dice using the dice class to determine how far they will move which also checks if it is a double and draws a card if necessary and does the correct effect. After that, the board class does the move player method on the player with the number rolled by the dice to move the player in the board and to determine what happens when they land on a space. If the player is bankrupt after this then the number of players is decreased by 1 and the current player is deleted from the list of players and the method is then complete. There is also the checkWin method in the game class with simply checks if there is only one player left and if so announces them as the winner.

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			
Defined in detail the structure of the solution to be developed.			

Described the solution fully using appropriate and accurate algorithms justifying how these algorithms form a complete solution to the problem.			
Described, justifying choices made, the usability features to be included in the solution.			
Identified and justified the key variables / data structures / classes (as appropriate to the proposed solution) justifying and explaining any necessary validation.			
Identified and justified the test data to be used during the iterative development of the solution.			
Identified and justified any further data to be used in the post development phase.			

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.

Before development I looked at how strings work in java using (W3 schools, 2025) and also how to declare arrays in Java using (W3 schools, 2025).

```
public class Board { //nousages new*
    String[26][3] board; //Declares the board as a 2D array of Strings
    boolean passedStart; //Declares the passedStart attribute as a boolean 1usage

    public Board(int numberOfPlayers,Animal[] animalArray){ //The constructor for the Board class nousages new*
        switch(numberOfPlayers){ //Switch statement which begins the game with a different number of players depending on how many are in the game
            case 2:
                board[0][0] = "1,2"; //If there are 2 players, inserts player numbers 1 and 2 into the player positions column of the start space in the board array
            case 3:
                board[0][0] = "1,2,3"; //If there are 3 players, inserts player numbers 1, 2 and 3 into the player positions column of the start space in the board array
            case 4:
                board[0][0] = "1,2,3,4"; //If there are 4 players, inserts player numbers 1, 2, 3 and 4 into the player positions column of the start space in the board array
            default:
                System.out.println("Invalid number of player's entered")
        }
    }
}
```

I began development with the constructor for the board class as the development plan instructs that the board class must be completed first and the constructor is necessary for the functioning of the rest of the class. I began with following the flowchart for the constructor and the first step was to initialise the player positions index for the start space in the board array with the player numbers for all the players as they all begin in the start space. However, we have decided that Animalopoly should be playable for 2-4 players unlike in the flowchart so during development I

implemented a switch statement to initialise the start space with different numbers of players depending on how many were inputted to be playing. I used a switch statement for this as only a single variable (the parameter numberOfPlayers) was being compared, and it was easy to compare all possible numbers of players this way. I plan to implement validation of the number of players being inputted when the inputs are taken in at the start of the game so the default case should never be accessible but in case it is reached, an error message will be outputted. I forgot the syntax for this, so I researched the println syntax online (Geeks for Geeks, 2025). Next, I went to test this part of the constructor so I briefly implemented a prototype version of the main class (which will run the program and instantiate a game object) and the game class (which will set up the initial objects to be used in the game and also compute each turn of the game) with the help of (Jetbrains, 2025) so I could correctly set up the Main class.

Main class:

```
1 public class Main { new *
2     public static void main(String[] args) { new *
3         Game game = new Game();
4     }
5 }
6
```

Prototype Game class for testing:

```
public class Game { 2 usages new *  
  
    Animal[24] animalArray;  
  
    public static void Game(){ no usages new *  
        //Initialise animal array  
        for(int i=0;i<24;i++){    //Very basic initialisation of the array just sets the animal's name to a number  
            strI = (String) i;  
            animalArray[i] = new Animal(strI);  
        }  
        //end initialisation of animal array  
        int numberOfPlayers = -1;  
        while(numberOfPlayers > 1 && numberOfPlayers < 5) { //Repeats until a valid number of players is entered  
            System.out.println("Please input the number of players (2-4 players allowed)"); //Asks the player how many players are in the game  
            Scanner inputter = new Scanner(System.in);  
            String numberOfPlayersStr = inputter.nextLine(); //Takes the player's input as a string  
            numberOfPlayers = Integer.parseInt(numberOfPlayersStr); //casts the string to an integer  
        }  
  
        |  
  
        Board board = new Board(animalArray, numberOfPlayers); //Instantiates a board object with the animal array and number of player's as parameters  
    }  
  
}
```

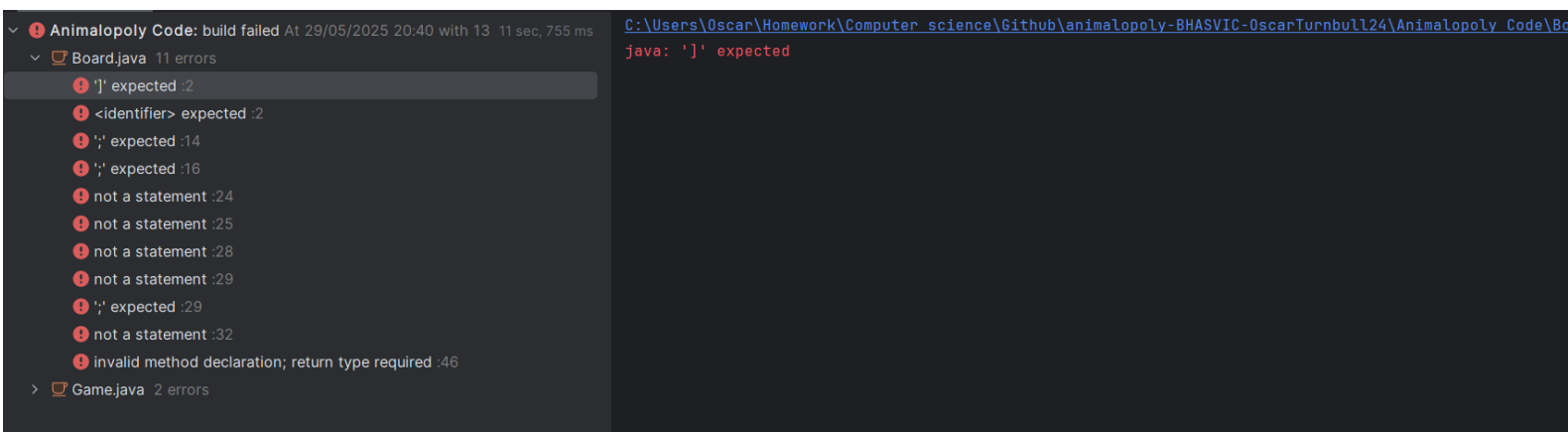
Prototype Animal class just for testing:

```
public class Animal { 3 usages new *  
    String name; 2 usages  
  
    public Animal(String theName){ 1 usage new *  
        name = theName;  
    }  
  
    public String getName(){ 2 usages new *  
        return name;  
    }  
}
```

Firstly I checked if you could pass an array into a method in Java using (GeeksforGeeks, 2025).

The Game class begins by creating an array of all 24 of the unique animal objects to be used primarily by the board class (Right now it simply fills the array with animal objects with their name being set as the index they have in the array but this is only to be used for testing that the board class works correctly and the actual full animal objects will be implemented later). Next, the user is asked how many players are in the game and this number is inputted which I researched how to do using (W3 schools, 2025) and assigned to the variable numberOfPlayers. Finally, a board class is constructed taking the number of players and the array of animals as parameters.

I attempted to run the code however I was met with many syntax errors:



The screenshot shows an IDE with two files: Board.java and Game.java. Board.java has 11 errors, and Game.java has 2 errors. The errors for Board.java are: 1. ']' expected :2, 2. <identifier> expected :2, 3. ';' expected :14, 4. ';' expected :16, 5. not a statement :24, 6. not a statement :25, 7. not a statement :28, 8. not a statement :29, 9. ';' expected :29, 10. not a statement :32, 11. invalid method declaration; return type required :46. The errors for Game.java are: 1. java: ']' expected, 2. java: ']' expected.

The first error was to do with my declaring of the board array which I had attempted to specify the size of the array when declaring incorrectly like this:

```
String[[26]][3] board; //Declares the board as a 2D array of Strings
```

I fixed this issue by looking at (GeeksforGeeks, 2025) for how to declare an array in Java and changed my code to this:

```
String[][] board = new String[26][3]; //Declares the board as a 2D array of Strings 11 usages
```

This fixed the issue as I now used the correct syntax for declaring an array.

The next three errors were because I had missed semi-colons on the ends of the following lines in the board class:

```
default:
    System.out.println("Invalid number of player's entered");
```

```
System.out.println(board[0][0]);
```

This line was added to test the board class which will be detailed later.

```
board[i][2] == "Miss a turn"
```

And so I updated them to:

```
default:
    System.out.println("Invalid number of player's entered");
}
System.out.println(board[0][0]);

for(int i=0;i<26;i++){ //Iterates through all spaces in the board to initialise their space type
    if(i==0){ // The first space is the start space so gains the space type Special and the name Start
        board[i][1] == "Special";
        board[i][2] == "Start";
    }
    if(i==13){ // The thirteenth space is the miss a turn space so gains the space type Special and the name Start
        board[i][1] == "Special";
        board[i][2] == "Miss a turn";
    }
}
```

The next syntax errors were due to using == for assignment incorrectly in the board class here:

```
for(int i=0;i<26;i++){ //Iterates through all spaces in the board to initialise their space type
    if(i==0){ // The first space is the start space so gains the space type Special and the name Start
        board[i][1] == "Special";
        board[i][2] == "Start";
    }
    if(i==13){ // The thirteenth space is the miss a turn space so gains the space type Special and the name Start
        board[i][1] == "Special";
        board[i][2] == "Miss a turn";
    }
    else{ //All other spaces are animal spaces so gain the space type Animal and the name will be the specific animal name
        board[i][1] == "Animal";
        if(i<13){ //
```

Which I fixed by changing these to single = signs:

```
for(int i=0;i<26;i++){ //Iterates through all spaces in the board to initialise their space type
    if(i==0){ // The first space is the start space so gains the space type Special and the name Start
        board[i][1] = "Special";
        board[i][2] = "Start";
    }
    if(i==13){ // The thirteenth space is the miss a turn space so gains the space type Special and the name Start
        board[i][1] = "Special";
        board[i][2] = "Miss a turn";
    }
    else{ //All other spaces are animal spaces so gain the space type Animal and the name will be the specific animal name
        board[i][1] = "Animal";
```

The final syntax error was due to incorrectly declaring the animal array like with the board array:

```
Animal[24] animalArray;
```

Which I also fixed by changing the code to:

```
Animal[] animalArray = new Animal[24]; 2 usages
```

After fixing these, when running the code, I was met with more errors in the game class:

```
! cannot find symbol variable strI :8
! incompatible types: int cannot be converted to java.lang.String :8
! non-static variable animalArray cannot be referenced from a static context :9
! cannot find symbol variable strI :9
! cannot find symbol class Scanner :15
! cannot find symbol class Scanner :15
! non-static variable animalArray cannot be referenced from a static context :22
```

The first two was due to incorrectly casting an integer to a String

```
strI = (String) i;
```

To fix this I searched online how to correctly cast an integer to a string in Java (GeeksforGeeks, 2025) and fixed the issue by correctly writing:

```
String strI = Integer.toString(i);
```

The next issue I got was “cannot find symbol class Scanner” Which confused me as I thought the class was included in Java so I revisited (W3 schools, 2025) and realised I forgot to import java.util.Scanner inside the game class so I then included that to fix the issue.

Next I had the issue “non-static variable animalArray cannot be referenced from a static context”. Initially I was unsure what the problem was as I had identified the error was in line 12 here:

```

6      Animal[] animalArray = new Animal[24]; 2 usages
7
8      public static void Game(){ no usages
9          //Initialise animal array
10         for(int i=0;i<24;i++){ //Very basic initialisatio
11             String strI = Integer.toString(i);
12             animalArray[i] = new Animal(strI);
13         }

```

I partially remembered what was meant by a static variable, so I researched online to make sure I understood them full (GeeksforGeeks, 2025). Firstly, I realised that I had forgotten to define the scope of many of my attributes so I marked most of them as private as they should only be accessed by methods in their own classes to increase data security by being able to validate all changes made to attributes such as:

```

private Animal[] animalArray = new Animal[24]; 2 usages

```

However, this did not fix the issue. I understood that the static context in which the array was being used had something to do with the assigning of a new animal object to each index in the array so I experimented with defining each animal object more properly to hopefully fix the issue as follows outside of the loop:

```

Animal animal1 = new Animal("strI");
animalArray[1] = animal1;

```

However this did not fix the problem. I then realised that I had declared the Game's constructor as static by mistake when trying to make the main class so by removing that and replacing it with this:

```

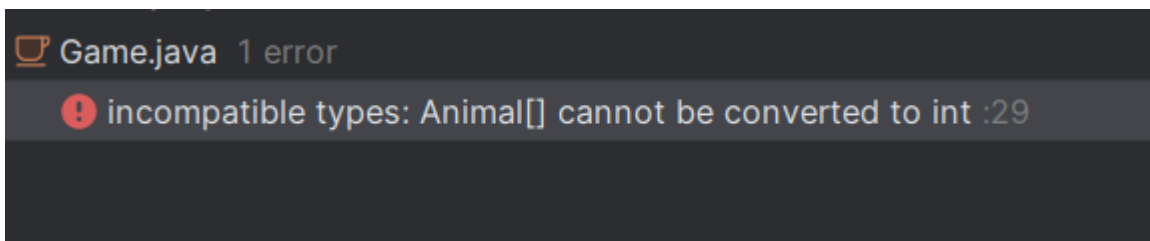
private Animal[] animalArray = new Animal[24]; 2 usages

public Game(){ no usages
    //Initialise animal array
    for(int i=0;i<24;i++){ //Very basic initialisation of the
        String strI = Integer.toString(i);
        animalArray[i] = new Animal(strI);
    }
}

```

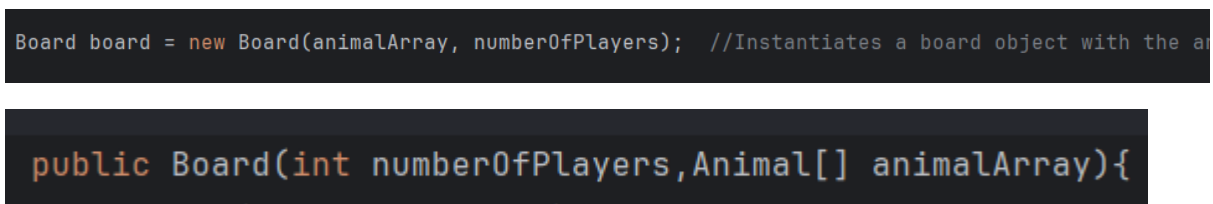
The issue was then fixed.

Finally, when running the program again I had this issue:



The screenshot shows an IDE window titled 'Game.java' with a red error icon and the message: 'incompatible types: Animal[] cannot be converted to int :29'. This indicates a type mismatch in the code at line 29.

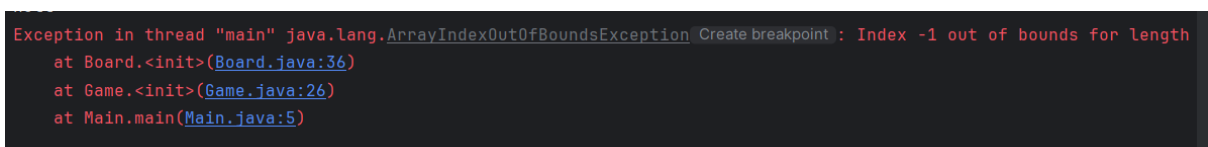
Which was due to me passing the arguments of the animal array and number of players in the wrong order into the board class:



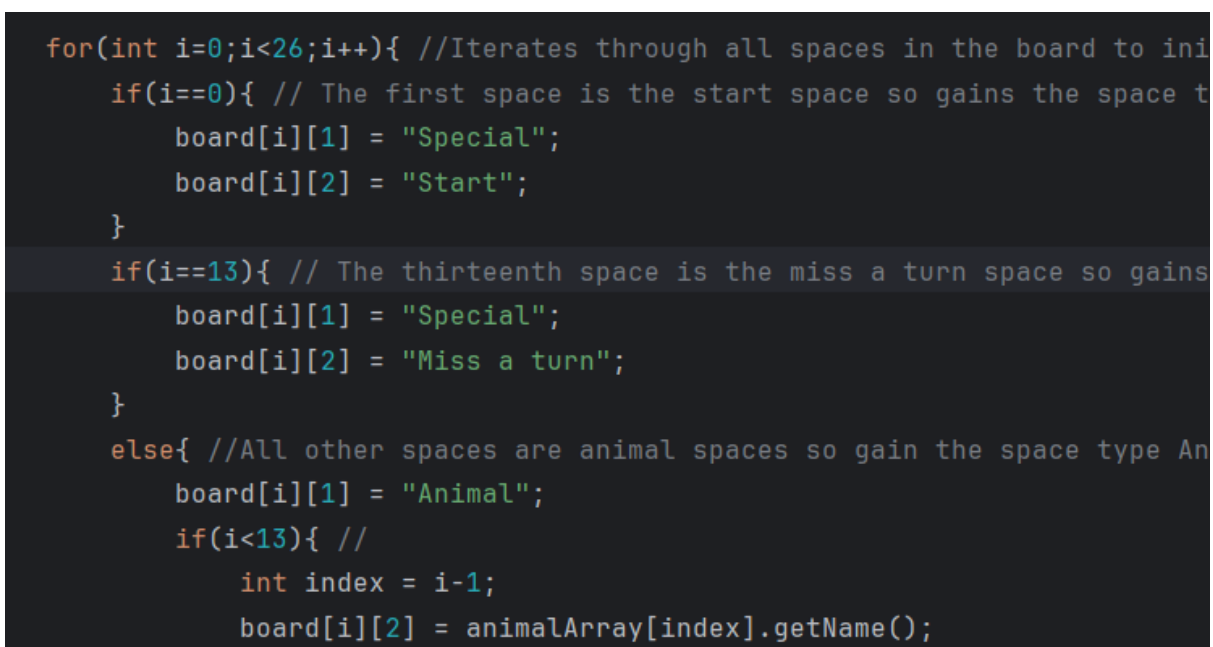
The screenshot shows two lines of Java code. The first line is a comment: '//Instantiates a board object with the a'. The second line is the constructor signature: `public Board(int numberOfPlayers, Animal[] animalArray){`. The parameters are in the wrong order for the intended logic.

So by switching their order in the board's initialisation this was fixed.

Next when running my code I got this error during the constructor for the board:



The screenshot shows a runtime exception: 'Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index -1 out of bounds for length'. The stack trace includes: 'at Board.<init>(Board.java:36)', 'at Game.<init>(Game.java:26)', and 'at Main.main(Main.java:5)'. This occurs because the code attempts to access an array at index -1.



The screenshot shows a loop for initializing a board. It iterates from i=0 to i=25. For i=0, it sets 'Special' and 'Start'. For i=13, it sets 'Special' and 'Miss a turn'. For other values of i, it sets 'Animal' and then tries to access 'animalArray[i-1].getName()'. When i=0, i-1 is -1, which causes the 'ArrayIndexOutOfBoundsException'.

After looking at these if statements the line where the error occurred was the final line here where the index -1 was out of bounds however this area of code should not be accessible if i=0 which would cause index = i-1 to be -1 but this occurred as the

second if(i==13) was not an else if which meant the else statement at the bottom ran if I was not 13 instead of not running when i is not 13 and i is not 0 so changing that to an else if fixed the issue.

Finally, the code ran with no syntax errors so I could follow the iterative development plan for the board's constructor and ensure that it meets the success criteria. Initially I ran the program however was not offered an opportunity to input the number of players and instead was met with:

```
Invalid number of player's entered  
null  
  
Process finished with exit code 0
```

I reviewed the area of my code where inputs are entered and realised that the while loop's condition was to repeat if the number of players is greater than 1 and less than 5 which would mean it would repeat if there was between 2-4 players

```
int numberOfPlayers = -1;  
while(numberOfPlayers > 1 && numberOfPlayers < 5) { //Repeats until a valid number of players i  
    System.out.println("Please input the number of players (2-4 players allowed)"); //Asks the  
    Scanner inputter = new Scanner(System.in);  
    String numberOfPlayersStr = inputter.nextLine(); //Takes the player's input as a string  
    numberOfPlayers = Integer.parseInt(numberOfPlayersStr); //casts the string to an integer  
}
```

However this should be the conditions to end the loop instead as anything not between 2-4 players should be made to be re-entered so I fixed this by changing the while loop to this:

```
while(numberOfPlayers < 2 || numberOfPlayers > 4) {
```

Running the code again allowed me to input a number of players. However when entering a valid number of players (2) I was met with

```
Please input the number of players (2-4 players allowed)  
2  
Invalid number of player's entered  
1,2,3,4  
  
Process finished with exit code 0
```

I then realised that my switch statements did not have a break; command which caused them to work incorrectly by researching (W3Schools, 2025)

```
switch(numberOfPlayers){ //Switch statement which begins the game with a dif
    case 2:
        board[0][0] = "1,2"; //If there are 2 players, inserts player number
    case 3:
        board[0][0] = "1,2,3"; //If there are 3 players, inserts player numb
    case 4:
        board[0][0] = "1,2,3,4"; //If there are 4 players, inserts player nu
    default:
        System.out.println("Invalid number of player's entered");
}
System.out.println(board[0][0]);
```

Which I changed to:

```
switch(numberOfPlayers){ //Switch statement which begins the game with a different number of
    case 2:
        board[0][0] = "1,2"; //If there are 2 players, inserts player numbers 1 and 2 into t
        break;
    case 3:
        board[0][0] = "1,2,3"; //If there are 3 players, inserts player numbers 1, 2 and 3 i
        break;
    case 4:
        board[0][0] = "1,2,3,4"; //If there are 4 players, inserts player numbers 1, 2, 3 an
        break;
    default:
        System.out.println("Invalid number of player's entered");
        break;
}
System.out.println(board[0][0]);
```

And then when running the code, it worked correctly:

```
Please input the number of players (2-4 players allowed)
2
1,2

Process finished with exit code 0
```

Next, I followed my iterative test plan for the constructor. Inputting 2 worked as expected so next I inputted 3 as the number of players:

```
Please input the number of players (2-4 players allowed)
```

```
3
```

```
1,2,3
```

```
Process finished with exit code 0
```

This outputted the correct player numbers and passed the test so next I inputted 4 as the number of players:

```
Please input the number of players (2-4 players allowed)
```

```
4
```

```
1,2,3,4
```

```
Process finished with exit code 0
```

This also outputted the correct player numbers as passed. Next to test the input validation for the number of players, I entered 5 as the number of players:

```
Please input the number of players (2-4 players allowed)
```

```
5
```

```
Please input the number of players (2-4 players allowed)
```

```
|
```

Which correctly rejected this incorrect value and asked the user to input a correct number of players. Next, I entered 1 as the number of players to also check the input validation:

```
Please input the number of players (2-4 players allowed)
```

```
1
```

```
Please input the number of players (2-4 players allowed)
```

```
|
```

This was also rejected so the input validation passed its tests.

Next, I began testing the board's initialisation during the constructor following the test plan. Firstly I tested if the start position was correctly classed as a special space by outputting `board[0][1]`.

I wasn't completely sure on the syntax for print statements so I looked online to confirm it (W3 Schools, 2025).

After checking the test this was outputted:

```
Please input the number of players (2-4 players allowed)
2
1,2
Special

Process finished with exit code 0
```

Which is correct so the test was passed.

Next, I tested if the start space was classed correctly as "Start" by outputting board[0][2]. After running the test, Start was outputted so the test was passed.

```
Please input the number of players (2-4 players allowed)
2
1,2
Start

Process finished with exit code 0
```

Next, I tested if the Miss a Turn space was correctly classed as special by outputting position [13][1] of the board. After running the test, Special was outputted so the test was passed.

```
Please input the number of players (2-4 players allowed)
2
1,2
Special

Process finished with exit code 0
```

Next, I tested if the Miss a Turn space was correctly classed as a Miss a Turn space by outputting position [13][2] of the board. After running the test, Miss a Turn was outputted so the test was passed.

```
Please input the number of players (2-4 players allowed)
2
1,2
Miss a turn

Process finished with exit code 0
```

Next, I tested if an Animal space was correctly classed as an Animal space by outputting position [1][1] of the board. After running the test, Animal was outputted so the test was passed.

```
Please input the number of players (2-4 players allowed)
2
1,2
Animal

Process finished with exit code 0
```

As the constructor has now passed all of its tests I moved on to the `takePlayerTurn()`. Firstly I looked at (W3 schools, 2025) again to view different methods which can be used with Strings. I also looked at (GeeksforGeeks, 2025) to find out how to delete a specific character inside of a string. I began coding the `takePlayerTurn` method by following the flowchart:

```
public void movePlayer(Player currentPlayer, int numberOfSpaces){ no usages
    passedStart = false; //Initialises passedStart as false as the player has not yet passed start
    int position = currentPlayer.getPosition(); //Gets the position of the space the player already was on
    String playersOnSpace = board[position][0]; //Gets the current players string from the board from the space the player is on
    int indexOfPlayer = playersOnSpace.indexOf(Integer.toString(currentPlayer.getPlayerNumber())); //Gets the index of the current player's player number in the string
    if(indexOfPlayer != 0){ //If the index is not 0 which means that the player's number is not the first in the list so the comma before it must be deleted as well
        deleteElement(playersOnSpace, index: indexOfPlayer - 1); //deletes the comma before the player number
        indexOfPlayer--;
        deleteElement(playersOnSpace, indexOfPlayer); //deletes the player number
    }
    else if (playersOnSpace.length() > 1){ //the player's player number is the first in the list but it is not the only one in the list
        deleteElement(playersOnSpace, index: indexOfPlayer + 1); //deletes comma after the player number
        deleteElement(playersOnSpace, indexOfPlayer); //deletes the player number
    }
    else{ //the player's player number is the only one in the list so there are also no commas
        deleteElement(playersOnSpace, indexOfPlayer); //deletes the player number
    }

    board[position][0] = playersOnSpace; //Updates the board to match the updated string
}
```

Firstly, the player's position is retrieved using the `getPosition` method from the `currentPlayer` object which is passed into the method as a parameter. Then the "player positions" column in the board is accessed for the position that the player is in. The player's player number is gotten and removed from the string using an additional `deleteElement` method I created to delete a specific character from a string as the player will be moving away from this space and any now unnecessary commas to separate the player numbers are also removed. I had not created the methods for the player class at this point, so they currently did not function.

```
private void deleteElement(String players, int index){ 4 usages
    players = players.substring(0,index) + players.substring( beginIndex: index + 1);
}
```

Here is the `deleteElement` method which sets an inputted string to the substrings before and after the index to be deleted which I learnt how to do from (GeeksforGeeks, 2025).

```
position += numberOfSpaces; //adds the dice roll to the player position
if(position > 25){ //if the position is greater than 25 then they have looped around the board and passed Start so their position must be set back to 0
    position -= 26;
    passedStart = true;
}

playersOnSpace = board[position][0]; //gets the current players string for the new board position
int length = playersOnSpace.length();
if(length > 0){ //if the length is greater than 0 then there is multiple player on the space so the new player number must be added with a comma
    playersOnSpace = playersOnSpace.concat( str: "," + Integer.toString(currentPlayer.getPlayerNumber()));
}
else{ // if the length is 0 then the player number can be added by itself
    playersOnSpace = Integer.toString(currentPlayer.getPlayerNumber());
}

board[position][0] = playersOnSpace; //updates the position on the board

currentPlayer.setPosition(position); //set's the player position attribute in the player to the new updated position
```

Next the number of spaces to move from the dice roll is added to the position to result in the new player position and if the player position is now greater than 25 then subtract 26 as the player has now passed start and needs to loop round the board again and set the `passedStart` attribute to true to be used later. Then the new "player positions" string for the new board position is found and the current player's player number is added into this string and then that string is inserted back into the board.

```

if (board[position][1].equals("Animal")){ //if the player lands on an animal space
    if(passedStart == true){ //if the player passed start add 500 to money
        currentPlayer.addMoney(500);
    }

    Animal currentAnimal = null;
    for(int j=0;j<24;j++){ //Loops through the animal array and find the animal which is the same as the one which has been landed on
        if(animalArray[j].getName().equals(board[position][2])){
            currentAnimal = animalArray[j];
        }
    }

    currentAnimal.landed(); //does the landed method in the animal object which asks does the relevant actions
}
else if(board[position][2].equals("Start")){ //if the player lands on the start space add 1000 to their money
    currentPlayer.addMoney(1000);
}
else{ //if they did not land on a start or animal space then they must have landed on a miss a turn space
    if(passedStart == true){ //if they passed start add 500 to their money
        currentPlayer.addMoney(500);
    }
    currentPlayer.missNextTurn(); //since they landed on a miss a turn space then do the missNextTurn method inside of the player object
}
}

```

If the board position which has been landed on is an animal space, then get the animalArray and search through it until the animal with the matching name is found. Then call the landed method of that animal with the player as a parameter which will run the necessary code to deal with a player landing on an animal. If the board position is a start space, then give the player £1000. If the board position is a miss a turn space, then set the missingTurn attribute in the player object to true which will cause them to miss their next turn using the missNextTurn method. If the player passed start and did not land on it then also give them £500.

Before I could test thing code, a lot of methods used rely heavily on the player class which was not yet developed so I decided to develop that class next so that the code is functional.

```

public class Player { 2 usages BHASVIC-OscarTurnbull24 *
|
    private char piece; 1 usage
    private int playerNumber; 2 usages
    private int money; 4 usages
    private boolean missingTurn; 3 usages
    private int boardPosition; 3 usages
    private String playerColour; 1 usage
    private String name; 1 usage

    public Player(char thePiece, int thePlayerNumber, String theColour, String theName){ no usages new *
        piece = thePiece;
        playerNumber = thePlayerNumber;
        playerColour = theColour;
        name = theName;
        money = 1500;
        missingTurn = false;
        boardPosition = 0;
    }
}

```

The player class has all attributes defined in the class diagram. It's constructor takes the piece, player number, player colour and player name as parameters as these are different for each player and the rest of the attributes are initialised to starting values as they always begin the same as the money begins at 1500, missing turn as false and the board position as 0.


```

public class Player { 28 usages  🡕 Oscar +1 *

    private char piece; 1 usage
    private int playerNumber; 2 usages
    private int money; 5 usages
    private boolean missingTurn; 3 usages
    private int boardPosition; 3 usages
    private String playerColour; 1 usage
    private String name; 1 usage

    public Player(char thePiece, int thePlayerNumber, String theColour, String theName){ 9 usages  🡕 Oscar
        piece = thePiece;
        playerNumber = thePlayerNumber;
        playerColour = theColour;
        name = theName;
        money = 1500;
        missingTurn = false;
        boardPosition = 0;
    }

    public void addMoney(int addedMoney){ 11 usages  🡕 Oscar
        money += addedMoney;
    }

    public void subtractMoney(int subtractedMoney){ no usages  🡕 Oscar
        money -= subtractedMoney;
    }

    public int getMoney(){ 1 usage  🡕 Oscar
        return money;
    }

    public int getPosition(){ 11 usages  🡕 Oscar
        return boardPosition;
    }

    public void setPosition(int newPosition){ 1 usage  🡕 Oscar
        boardPosition = newPosition;
    }
}

```

The add and subtract money methods are used to update the player's money. I decided to create these instead of a basic setMoney method as the money is never set to a specific value only added to and subtracted from and this makes it easier to

do so without also having to use the getter to get the money attribute from the player and add the number to that. There is also a `getMoney` method to retrieve the value of the Player's money. The `getPosition` method is used to return the position of the player when it is needed such as in the board class in the move player method. The `setPosition` method sets the position of the player to a new position such as when the player is moved.

```
public void setMissingTurn(boolean newValue){ 2 usages new *
    missingTurn = newValue;
}

public boolean checkMissingTurn(){ 1 usage 2 Oscar
    return missingTurn;
}

public int getPlayerNumber(){ 3 usages 2 Oscar
    return playerNumber;
}

public boolean checkBankrupt(){ 1 usage 2 BHASVIC-OscarTurnbull24
    if(money < 0){
        return false;
    }
    return true;
}
```

The `setMissingTurn` method sets the missing turn attribute to true which skips a player's turn or false which will not miss the next turn. The `checkMissingTurn` method checks if the current player is missing their next turn or not when the `takePlayerTurn` method is being run in the game class. The `getPlayerNumber` method gets the player's `playerNumber` which is used in the `movePlayer` method in the board class to update the position of the player. The `checkBankrupt` method checks if the player's money has dropped to less than 0 so that they can be eliminated from the game if so.

After developing the player class I ran the `movePlayer` method of the board class with an instantiated player however I was met with this error:

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "playersOnSpace" is null
    at Board.movePlayer(Board.java:73)
    at Game.takePlayerTurn(Game.java:168)
    at Game.<init>(Game.java:152)
    at Main.main(Main.java:4)

Process finished with exit code 1
```

I looked at the line at which the board class failed at in the code and realised that the .length method does not work on the current players column of the board object if the string is empty:

```
playersOnSpace = board[position][0]; //gets the current players string for the new board position
int length = playersOnSpace.length();
```

To fix this I included an if statement to set the length to 0 if the board position is null:

```
int length;
playersOnSpace = board[position][0]; //gets the current players string for the new board position
if(playersOnSpace == null){
    length = 0;
}
else {
    length = playersOnSpace.length();
}
```

Next, when running the code I was met with this error message:

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "Animal.getName()" because "this.animalArray[j]" is null
    at Board.movePlayer(Board.java:101)
    at Game.takePlayerTurn(Game.java:168)
    at Game.<init>(Game.java:152)
    at Main.main(Main.java:4)

Process finished with exit code 1
```

For these lines of code:

```
for (int j = 0; j < 24; j++) { //Loops through the animal array and find the animal which is the same as the one which has been landed on
    if (animalArray[j].getName().equals(board[position][2])) {
        currentAnimal = animalArray[j];
    }
}
```

I was initially confused as to how an index of the animal array could be null as I initialised each index of the array in the game class and passed that into the board

class however after viewing the constructor for the board class:

```
public Board(int numberOfPlayers, Animal[] theAnimalArray) { //The constructor for the Board class 1usage 2Oscar+1*
    switch (numberOfPlayers) { //Switch statement which begins the game with a different number of players depending on how many are in the game
        case 2:
            board[0][0] = "1,2"; //If there are 2 players, inserts player numbers 1 and 2 into the player positions column of the start space in the board array
            break;
        case 3:
            board[0][0] = "1,2,3"; //If there are 3 players, inserts player numbers 1, 2 and 3 into the player positions column of the start space in the board array
            break;
        case 4:
            board[0][0] = "1,2,3,4"; //If there are 4 players, inserts player numbers 1, 2, 3 and 4 into the player positions column of the start space in the board array
            break;
        default:
            System.out.println("Invalid number of player's entered");
            break;
    }
    System.out.println(board[0][0]);

    for (int i = 0; i < 26; i++) { //Iterates through all spaces in the board to initialise their space type
        if (i == 0) { // The first space is the start space so gains the space type Special and the name Start
            board[i][1] = "Special";
            board[i][2] = "Start";
        } else if (i == 13) { // The thirteenth space is the miss a turn space so gains the space type Special and the name Start
            board[i][1] = "Special";
            board[i][2] = "Miss a turn";
        } else { //All other spaces are animal spaces so gain the space type Board.Animal and the name will be the specific animal name
            board[i][1] = "Board.Animal";
            if (i < 13) { //
                int index = i - 1;
                board[i][2] = theAnimalArray[index].getName();
            } else {
```

```
                if (i < 13) { //
                    int index = i - 1;
                    board[i][2] = theAnimalArray[index].getName();
                } else {
                    int index = i - 2;
                    board[i][2] = theAnimalArray[index].getName();
                }
            }
        }
        passedStart = false;
    }
}
```

I realised I had used the Animal array to construct the board array however I had never set the animal array used inside the board class equal to the one passed into the board class as a parameter so I fixed the issue by writing this line of code:

```
passedStart = false;
animalArray = theAnimalArray;
```

After this the movePlayer method ran without crashing so that I could move onto the game class before testing the logic of the whole system in post development testing.

Finally, I developed the game class which creates the game and runs each turn.

```
import java.util.Scanner;

public class Game { 2 usages  ⚡ BHASVIC-OscarTurnbull24 +1 *

    private Animal[] animalArray = new Animal[24]; 2 usages
    private Player[] playerArray = new Player[4]; 4 usages
    private int numberOfPlayers; 11 usages
    private Dice dice; 2 usages
    private Board board; 2 usages
```

The game class has the attributes of an array of all the animal objects to be used in the game, an array of all of the player objects to be used in the game, an integer which is the number of players in the game so that the players can be initialised, the dice object and the board object to be used. I decided to instantiate all of the objects to be used in the game class so that they can all be directly accessed and controlled from a single place.

```
public Game(){ 1 usage  ⚡ BHASVIC-OscarTurnbull24 +1
    //Initialise animal array
    for(int i = 0; i < 24; i++){
        new Animal(((i * 100) + 300), 1);
    }
    //end initialisation of animal array
    Scanner inputter = new Scanner(System.in);
    numberOfPlayers = -1;
    while(numberOfPlayers < 2 || numberOfPlayers > 4) { //Repeats until a valid number of players is entered
        System.out.println("Please input the number of players (2-4 players allowed)"); //Asks the player how many players are in the game

        String numberOfPlayersStr = inputter.nextLine(); //Takes the player's input as a string
        numberOfPlayers = Integer.parseInt(numberOfPlayersStr); //casts the string to an integer
    }
}
```

The beginning of the game class is the same as before except the initialisation of the Animal array was changed by Sam to correctly work with the full animal object.

```

if(numberOfPlayers == 2) {
    System.out.println("PLAYER 1: Please input the piece you want to use");
    char Player1Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 1: Please input the colour you want to use");
    String Player1Colour = inputter.nextLine();

    System.out.println("PLAYER 1: Please input the name you want to use");
    String Player1Name = inputter.nextLine();

    Player player1 = new Player(Player1Piece, thePlayerNumber: 1, Player1Colour, Player1Name);

    System.out.println("PLAYER 2: Please input the piece you want to use");
    char Player2Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 2: Please input the colour you want to use");
    String Player2Colour = inputter.nextLine();

    System.out.println("PLAYER 2: Please input the name you want to use");
    String Player2Name = inputter.nextLine();

    Player player2 = new Player(Player2Piece, thePlayerNumber: 2, Player2Colour, Player2Name);
    playerArray = new Player[]{player1, player2};
}

```

Next, I did a series of if statements to initialise all of the players for each possible numbers of players. For each player, they are asked to input which text based playing piece they want to use which is then assigned to the correct variable and used in the construction of that player's player object. The same is also done with the colour of the pieces they want to use and the name of the player so that each player can feel uniquely and personally represented while playing the game. Then a player object is instantiated for each player and after all of this is done, the player array is set to contain each player who has been instantiated. The same idea is also used if both 3 and 4 players are chosen instead:

```

else if(numberOfPlayers == 3) {
    System.out.println("PLAYER 1: Please input the piece you want to use");
    char Player1Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 1: Please input the colour you want to use");
    String Player1Colour = inputter.nextLine();

    System.out.println("PLAYER 1: Please input the name you want to use");
    String Player1Name = inputter.nextLine();

    Player player1 = new Player(Player1Piece, thePlayerNumber: 1, Player1Colour, Player1Name);

    System.out.println("PLAYER 2: Please input the piece you want to use");
    char Player2Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 2: Please input the colour you want to use");
    String Player2Colour = inputter.nextLine();

    System.out.println("PLAYER 2: Please input the name you want to use");
    String Player2Name = inputter.nextLine();

    Player player2 = new Player(Player2Piece, thePlayerNumber: 2, Player2Colour, Player2Name);

    System.out.println("PLAYER 3: Please input the piece you want to use");
    char Player3Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 3: Please input the colour you want to use");
    String Player3Colour = inputter.nextLine();

    System.out.println("PLAYER 3: Please input the colour you want to use");
    String Player3Colour = inputter.nextLine();

    System.out.println("PLAYER 3: Please input the name you want to use");
    String Player3Name = inputter.nextLine();

    Player player3 = new Player(Player3Piece, thePlayerNumber: 3, Player3Colour, Player3Name);
    playerArray = new Player[]{player1, player2, player3};
}

```

For three players.

```

else if(numberOfPlayers == 4) {
    System.out.println("PLAYER 1: Please input the piece you want to use");
    char Player1Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 1: Please input the colour you want to use");
    String Player1Colour = inputter.nextLine();

    System.out.println("PLAYER 1: Please input the name you want to use");
    String Player1Name = inputter.nextLine();

    Player player1 = new Player(Player1Piece, thePlayerNumber: 1, Player1Colour, Player1Name);

    System.out.println("PLAYER 2: Please input the piece you want to use");
    char Player2Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 2: Please input the colour you want to use");
    String Player2Colour = inputter.nextLine();

    System.out.println("PLAYER 2: Please input the name you want to use");
    String Player2Name = inputter.nextLine();

    Player player2 = new Player(Player2Piece, thePlayerNumber: 2, Player2Colour, Player2Name);

    System.out.println("PLAYER 3: Please input the piece you want to use");
    char Player3Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 3: Please input the colour you want to use");
    String Player3Colour = inputter.nextLine();

    System.out.println("PLAYER 3: Please input the colour you want to use");
    String Player3Colour = inputter.nextLine();

    System.out.println("PLAYER 3: Please input the name you want to use");
    String Player3Name = inputter.nextLine();

    Player player3 = new Player(Player3Piece, thePlayerNumber: 3, Player3Colour, Player3Name);

    System.out.println("PLAYER 4: Please input the piece you want to use");
    char Player4Piece = inputter.next().charAt(1);

    System.out.println("PLAYER 4: Please input the colour you want to use");
    String Player4Colour = inputter.nextLine();

    System.out.println("PLAYER 4: Please input the name you want to use");
    String Player4Name = inputter.nextLine();

    Player player4 = new Player(Player4Piece, thePlayerNumber: 4, Player4Colour, Player4Name);
    playerArray = new Player[]{player1, player2, player3, player4};
}

else{
    System.out.println("Invalid number of players");
}

```

For four players and if the number of players is none of these then an error message is outputted however this should not happen under normal circumstances as the input is validated for the number of players.


```

board = new Board(numberOfPlayers, animalArray); //Instantiates a board object with the animal array and number of player's as parameters
dice = new Dice();
cards = new Cards();

int i = 0;

while(numberOfPlayers >= 1){
    takePlayerTurn(playerArray[i]);
    i++;
    if(i >= numberOfPlayers){
        i = 0;
    }
}
}

```

Next, a new board object, dice object and cards object are instantiated so that they can be used in each player's turn. Next, the players' turns are iteratively taken inside a while loop and when the final player is reached then the first player's turn is taken again which repeats until there is a single player left.

```

public void takePlayerTurn(Player currentPlayer, int index){ 1 usage 2 BHASVIC-OscarTurnbull24 *
    if(!currentPlayer.checkMissingTurn()) {
        int diceRoll = dice.diceRoll();
        board.movePlayer(currentPlayer, diceRoll);
        if (currentPlayer.checkBankrupt()) {
            numberOfPlayers--;
            for(int i=index; i<numberOfPlayers; i++){
                if(i <= numberOfPlayers -1){
                    playerArray[i] = playerArray[i+1];
                }
                else{
                    playerArray[i] = null;
                }
            }
        }
    }
    else{
        currentPlayer.setMissingTurn(false);
    }
}
}

```

The takePlayerTurn method firstly checks if the current player is missing their turn and if so then their turn is skipped and the missing turn attribute is set to false and if not then the turn is taken. During the turn, firstly the dice is rolled by the dice class which also deals with the possibility of rolling a double and drawing a card and the board then performs the move player method on the player using the amount rolled by the dice. Next, if the player is bankrupt after moving by landing on an animal owned by another player and losing money then this is checked using the player's checkBankrupt method and if they are then the total number of players is subtracted

by one so that the while loop which takes the turns can repeat for the correct number of players. Then, the player object of the current player who is bankrupt is deleted from the player array which is done by starting from the current player, setting the index of that player object to what is stored in the next index along so that all of the player objects ahead of the current player are shifted one index down removing the current player and setting the final index to null.

When I first ran the game class I was met with this error:

```
Please input the number of players (2-4 players allowed)
2
PLAYER 1: Please input the piece you want to use
*
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: Index 1 out of bounds for length 1 <7 internal lines>
    at java.base/java.lang.String.checkIndex(String.java:4930)
<4 folded frames>
```

I looked closely at the part “Index 1 out of bounds for length one” and realised that my input was of length 1 but the section of code which is responsible for inputting the player pieces took the character of index 1 of the user’s input however the index which I actually wanted to be set as the player’s piece was index 0 so I changed the following code:

```
System.out.println("PLAYER 1: Please input the piece you want to use");
char Player1Piece = inputter.next().charAt(1);
```

Into this:

```
System.out.println("PLAYER 1: Please input the piece you want to use");
char Player1Piece = inputter.next().charAt(0);
```

For each of the player piece inputs.

After running the code again the previous error was fixed and the code ran correctly allowing inputs of all of the attributes of all players and iteratively ran the takePlayerTurn method correctly. However I did not have time to test that the logic of the game class correctly functioned with the other classes created by the other members of my team or do post development testing of the game so if I was given more time or worked on another project like this again I would better organize the work done with each team member to ensure all stages of development were completed earlier to allow for post development testing and evaluation later.

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			
Provided evidence of prototype versions of my solution for each stage of the process.			
Ensured the solution is well structured and modular in nature.			
Annotated the code to aid future maintenance of the system.			
Ensured all variables and structures are appropriately named.			
Included evidence of validation for all key elements of the solution.			
Shown my review at all key stages in the process.			
Provided evidence of testing at each stage of the iterative development process.			
Provided evidence of any failed tests and the remedial actions taken with full justification for any actions taken.			

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?

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			
Provided annotated evidence for usability testing.			
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.			
Provided comments on how any partially or unmet criteria could be addressed in further development.			
Provided evidence of the usability features justifying their success, partial success or failure as effective usability features.			
Provided comments on how any issues with partially or unmet usability features could be addressed in further development.			
Considered maintenance issues and limitations of the solution.			
Described how the program could be developed to deal with limitations and potential improvements / changes.			
There is a well-developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated.			

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 be 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.

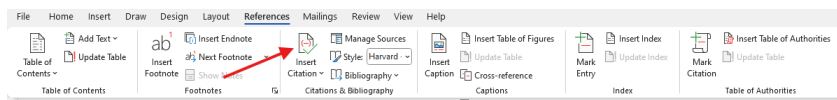
As you are writing this documentation you should include details about **every** website you access, **every** book that you read, and **every** other source that you access that can't be so easily categorised.

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: **Invalid source specified.** to appear in your work and 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.

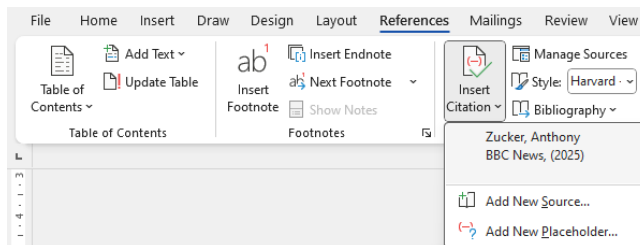
References Section



Source Form

A screenshot of the 'Create Source' dialog box in Microsoft Word. The 'Type of Source' is set to 'Web site'. The 'Language' is set to 'Default'. The 'Bibliography Fields for Harvard - Anglia' section includes fields for Author, Name of Web Page, Year, Year Accessed, Month Accessed, Day Accessed, and URL. The 'Show All Bibliography Fields' checkbox is checked. The 'Tag name' field contains 'Placeholder1'. The 'OK' and 'Cancel' buttons are at the bottom right.

Reusing a Source



Reference List

- Ada computer Science. (2025, 06 08). *Class definitions*. Retrieved from ada computer science: https://adacomputerscience.org/concepts/oop_class_diagrams
- BBC. (2025, 5 24). *Representing Iteration*. Retrieved from BBC Bitesize: <https://www.bbc.co.uk/bitesize/guides/zg46tfr/revision/4>
- Geeks for Geeks. (2025, 5 27). *System.out.println in Java*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/system-out-println-in-java/>
- GeeksforGeeks. (2025, 5 27). *How to Declare and Initialize an Array in Java?* Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/how-to-declare-and-initialize-an-array-in-java/>
- GeeksforGeeks. (2025, 6 1). *Java Convert int to String - Different Ways of Conversion*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/different-ways-for-integer-to-string-conversions-in-java/>
- GeeksforGeeks. (2025, 5 27). *Passing an Array to a Function in Java*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/how-to-pass-an-array-to-a-function-in-java/>
- GeeksforGeeks. (2025, 6 6). *Replace a character at a specific index in a String in Java*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/replace-a-character-at-a-specific-index-in-a-string-in-java/>
- GeeksforGeeks. (2025, 6 1). *Static Variables in Java*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/static-variables-in-java/>
- Jetbrains. (2025, 5 27). *Tutorial: Run a Java application*. Retrieved from Jetbrains: <https://www.jetbrains.com/help/idea/run-java-applications.html>
- W3 schools. (2025, 5 27). *Java Arrays*. Retrieved from W3 schools: https://www.w3schools.com/java/java_arrays.asp
- W3 Schools. (2025, 6 5). *Java Print Variables*. Retrieved from W3 Schools: https://www.w3schools.com/java/java_variables_print.asp
- W3 schools. (2025, 5 25). *Java String Methods*. Retrieved from w3schools: https://www.w3schools.com/java/java_ref_string.asp
- W3 schools. (2025, 5 27). *Java Strings*. Retrieved from W3 schools: https://www.w3schools.com/java/java_strings.asp
- W3 schools. (2025, 5 27). *Java User Input (Scanner)*. Retrieved from W3 schools: https://www.w3schools.com/java/java_user_input.asp

W3Schools. (2025, 6 2). *Java Switch*. Retrieved from W3Schools:
https://www.w3schools.com/java/java_switch.asp

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 created 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.)

The 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 Reference 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.