

BHASVIC Computer Science

# Animalopoly

Suliman Momand  
[Date]

# **Contents**

Analysis .....	2
What goes in this section?.....	2
Success Criteria.....	3
Design.....	7
What goes in this section?.....	7
Checklist.....	11
Development.....	12
What goes in this section?.....	12
Checklist.....	33
Evaluation.....	35
What goes in this section?.....	35
Checklist.....	36
References .....	37
References Explanation .....	37
Reference List.....	39
AI References.....	40
AI References Explanation .....	40
AI Reference List.....	40
Referencing FAQs .....	41
What happens if I don't cite a source I've used? .....	41
What if I don't realise, I've plagiarized? .....	41
What are the potential consequences of plagiarism? .....	41
If AI generated work isn't marked, then why use it?.....	42

# **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.	
<b>4. Animals</b>			
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/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 usability 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.

### Development plan:

- Dice
- Animals
- Cards

I will do the dice class as it is the easiest class to do. I will create 2 different dice that will output a number from 1 to 6. the numbers for both dice. I will create 3 getters called getdie1(gets the value of the first dice), getdie2(gets the value of the second dice) and diceSum(gets the sum of both dices). DiceSum will be called in the player class to move the player the allocated number of spaces. The other 2 getters will be called in the game class to be displayed to the player. I will also make a method called isdouble() to see if both dices have rolled the same number. If they have i will make the player pick a card. This would need a getter from the card class that picks a random card.

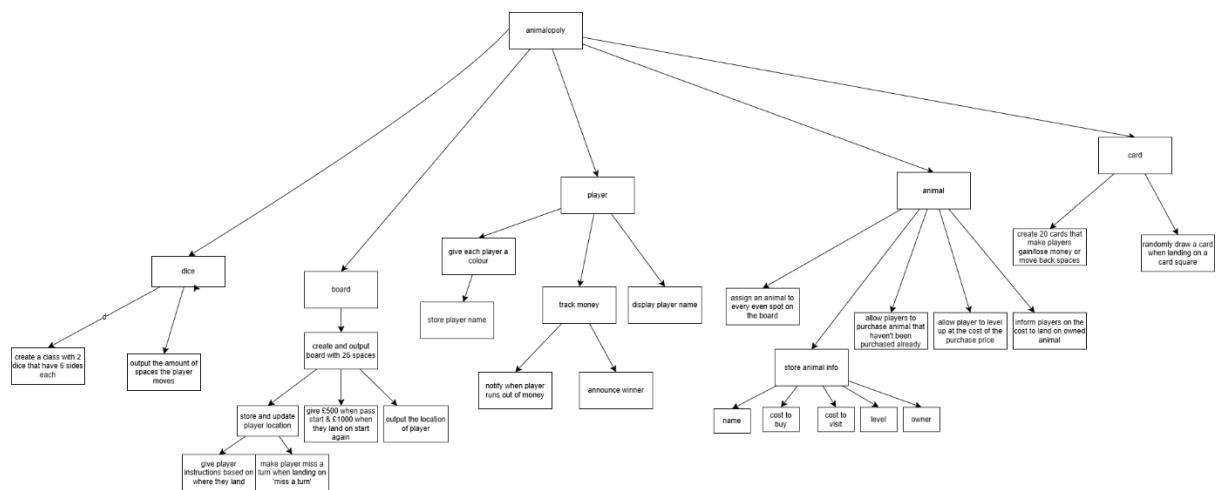
Then I will do the card class. In this class I will make an arraylist of cards. Each card will have a different effect on the players balance. I will make it so a card is randomly taken from the deck. Then I will make 2 getters for the description and the effect.

Lastly i will do the animal class as it is the class i will spend most of my time on. I will first store different attributes to the main animal class and make the pet classes have different types of the attributes. I will do an if statement to see if the player has enough money to buy the animal. Once the player reaches the animal again i will let

them upgrade their animal at the same price that they bought them. If the player lands on another players animal it will take money away from the player.

Lastly i will do the animal class as it is the class i will spend most of my time on. I will first store different attributes to the main animal class and make the pet classes have different types of the attributes. I will do an if statement to see if the player has enough money to buy the animal. Once the player reaches the animal again i will let them upgrade their animal at the same price that they bought them. If the player lands on another players animal it will take money away from the player.

## Structure Diagram



## Post-development plan

Testing:

Part tested	Pass/fail	Actions to take & why	Screenshot of code
eg. Moving player on board	fail	Player moved too far, need to edit values for moving to move correct spacing	

Tests to do:

Dice:

- Run tests to check the dice rolling system works and that it rolls two dice
- Test for passing the value of the rolls
- Test for confirmation of the rolls

Post development tests:

Animals:

- Have an animal on each space of the board
- Check correct data is stored
- Test purchasing animals and having them linked to a player
- Test for upgrading the level and setting a limit to upgrades
- Output the payment needed when a player lands on its space

Cards:

- Check each scenario for the 20 cards is stored properly

- Test that a card is picked at random and edits values correctly.

Part tested	Pass/fail	Actions to take & why	Screenshot of code
<b>eg. Moving player on board</b>			
<b>Draws a card when double rolled</b>			
<b>When a player loses, they are removed from the game</b>			
<b>Updates the balance correctly when someone lands on someone else's animal</b>			
<b>Displays what animal is on what tile</b>			
<b>Allows you to upgrade an animal</b>			
<b>Allows you to buy an animal if you can afford it</b>			
<b>Gives you £1000 if you pass go</b>			
<b>Able to input and choose your icons</b>			
<b>Displays the winner when they win the game</b>			

## 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

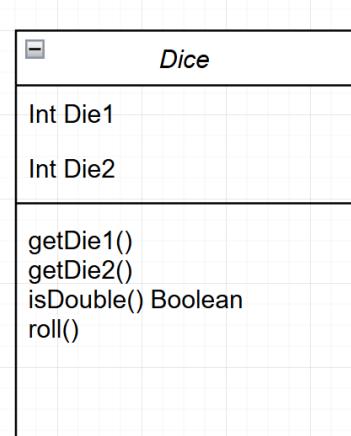
## What goes in this section?

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.

## Dice Class Diagram



I have changed the 'roll()' method to 'rollDice' to make what im doing more obvious.

Die1 & Die2 are the 2 dice in the game that are rolled and they output a random number.

## Pseudocode

```

1 public class Dice{
2     int die1=0
3     int die2=0
4
5     Public procedure roll() {
6         Die1 = RANDOM_INT(1, 6)
7         Die2 = RANDOM_INT(1, 6)
8         print "you rolled a" +getSum
9
10        if (isDouble()) {
11            print "YOU ROLLED A DOUBLE!"
12            drawCard()
13        }
14    }
15
16    int getDie1() {
17        return die1
18    }
19    int getDie2() {
20        return die2
21    }
22
23    Boolean isDouble() {
24        return die1 == die2
25    }
26
27    int getSum(){
28        return die1 + die2
29    }
30
31 }

```

This is the pseudocode I created for the dice class. I didn't know how to generate a random number between 2 values in pseudocode so I had to use (Lovelace, n.d.) to help me.

```

1 java.util.Random_
2 public class Dice { no usages
3     int die1 = 0; no usages
4     int die2= 0; no usages
5
6     public void rollDice() { no usages
7
8     }
9

```

A random number between 1 & 6 would be needed to be outputted by the dice but I didn't know how to generate a random number in Java so I had to go back to my source from earlier to help me.

```

import java.util.Random;

public class Dice { 1 usage
    private int die1; 3 usages
    private int die2; 3 usages

    public void rollDice() { 1 usage 1 related problem

    }
    public int getDie1() { no usages
        return die1;
    }

    public int getDie2() { no usages
        return die2;
    }

    public int getSum() { no usages
        return die1 + die2;
    }

    public boolean isDouble() { no usages
        return die1 == die2;
    }
}

```

I have now created all the getters I need. I have made 2 separate getters to return both die1 and die2. They are not needed for the code I'm creating but they are needed for the GUI that someone in my group has made to display the 2 dice.

I've also made a method that checks if a double has been rolled so that a card can be drawn for the player if it returns true.

now I need to create a method that rolls the dice.

```
public void rollDice() { 1 usage 1 related problem
    Random rand = new Random();
    die1 = rand.nextInt( bound: 6 ) + 1;
    die2 = rand.nextInt( bound: 6 ) + 1;
}
```

This makes it so that when rolled the die takes a random number between 1 and 6.

I then created a test to see if it returns the dices as different values each time and adds them together

```
public class Main {
    public static void main(String[] args) {
        Dice.rollDice();
        System.out.println("Die 1 is: " + Dice.getDie1());
        System.out.println("Die 2 is: " + Dice.getDie2());
    }
}
```

I had to make all die 1 & 2 static as well as my getters or else my code wouldn't work for some reason.

I ran 3 tests successfully. This is what they outputted:

```
C:\Users\smmom\.jdks\openjdk-24.0.1\bin\java
Die 1 is: 4
Die 2 is: 3
you rolled a: 7
Process finished with exit code 0
```

```
C:\Users\smmom\.jdks\openjdk-24.0.1\bin\java
Die 1 is: 6
Die 2 is: 2
you rolled a: 8
Process finished with exit code 0
```

```
C:\Users\smmom\.jdks\openjdk-24.0.1\bin\java
Die 1 is: 4
Die 2 is: 6
you rolled a: 10
Process finished with exit code 0
```

Now I needed to make it so that if a double is rolled, it does the drawCard() method (which hasn't been made as I haven't done the card class yet. So I will just make the drawCard() method output "Draw Card").

```

public static void drawCard() { 1 usage
    System.out.println("Draw Card");
}

public static void rollDice() { 1 usage
    Random rand = new Random();
    die1 = rand.nextInt( bound: 6 ) + 1;
    die2 = rand.nextInt( bound: 6 ) + 1;

    if(isDouble()) {
        drawCard();
    }
}

```

This makes it so that if both dice roll the same number, it will then draw a card from the deck(output “draw card” for now)

Now I need to test that it works.

I used the same test as earlier but it didn't output correctly.

```

Draw Card
Die 1 is: 2
Die 2 is: 2
you rolled a: 4

Process finished with exit code 0

```

I realised that the prints were meant to be in the rollDice() method to work.

This is what it looks like now:

```

public static void rollDice() { 1 usage
    Random rand = new Random();
    die1 = rand.nextInt( bound: 6 ) + 1;
    die2 = rand.nextInt( bound: 6 ) + 1;
    System.out.println("Die 1 is: " + Dice.getDie1());
    System.out.println("Die 2 is: " + Dice.getDie2());
    System.out.println("you rolled a: " + Dice.getSum());
    if(isDouble()) {
        drawCard();
    }
}

```

Main only has

Dice.rollDice(): now.

I tested it again and it worked.

```

Die 1 is: 3
Die 2 is: 3
you rolled a: 6
Draw Card

```

Now I need to do the card class so the drawCard() method actually does what it is meant to.

I forgot to annotate my code so I went back to the dice class and did it.

I also realised that I didn't need to output what both die rolled and that I only needed to output the sum of both dice

This is how my code looked at the end:

```
import java.util.Random;

public class Dice { 2 usages
    private static int die1; 4 usages
    private static int die2; 4 usages

    public static void rollDice() { //rolls the dice 1 usage
        Random rand = new Random();
        die1 = rand.nextInt(bound: 6) + 1;
        die2 = rand.nextInt(bound: 6) + 1;
        System.out.println("you rolled a: " + Dice.getSum()); //displays what has been rolled
        if(isDouble()) { // draws a card if player has rolled a double
            drawCard();
        }
    }

    public static int getDie1() { return die1; } //used in gui to display die1
    public static int getDie2() { return die2; } //used in gui to display die2
    public static int getSum() { return die1 + die2; }

    private static boolean isDouble() { return die1 == die2; } //checks if a double is rolled
    private static void drawCard() { System.out.println("Draw Card"); } //used in place for when i make the card class
}
```

Edit: went back to dice and removed all the static

```
import java.util.Random;

public class Dice { 3 usages
    private int die1; 4 usages
    private int die2; 4 usages

    public void rollDice(Card card, Player player, Dice dice) { //rolls the dice 1 usage
        Random rand = new Random();
        die1 = rand.nextInt(bound: 6) + 1;
        die2 = rand.nextInt(bound: 6) + 1;
        System.out.println("you rolled a: " + dice.getSum()); //displays what has been rolled
        if(isDouble()) { // draws a card if player has rolled a double
            System.out.println("you rolled a Double!!");
            card.drawCard(player);
        }
    }

    public int getDie1() { no usages
        return die1;
    } //used in gui to display die1
    public int getDie2() { no usages
        return die2;
    } //used in gui to display die2
    public int getSum() { 1 usage
        return die1 + die2;
    }

    private boolean isDouble() { return die1 == die2; } //checks if a double is rolled
}
```

**Card Class:**

**Class Diagram:**

dsa
List<Card>Cards
int random
int moneyEffect
string desc
getDesc()
cardDeck()
drawCard()
getEffect()

int randomCard(instead of int random) is used when picking a card,

The list is for the cards. Money effect is the amount the player gains/loses. Description is telling the player how much they gain/lose and it tells you what happened to the player for the change in balance.

## Pseudocode

```
public class Card {
    private int moneyEffect
    private String desc
    private Array list of cards // idk how to write that in pseudocode

    Card(String desc, int moneyEffect){
        this.desc = desc
        this.moneyEffect = moneyEffect
    }

    public procedure new cardDeck() {
        cards.add ("You receive $100 for helping save animals", +100)
        // create 19 more of these
    }

    public procedure new drawCard() {
        int randomCard = RANDOM_INT(cards.size)
        PRINT "you picked" +getDesc
    }

    public int getDesc() {
        return desc
    }
}
```

At first I was thinking about creating different if statements for each card e.g

If card[1] THEN

Balance + 100

But I knew doing it that way is highly impractical, so instead I made it so each card has the description and the moneyEffect stored in it.

I also forgot to create a method that actually takes away the money from the players balance.

## Java Code

I used (W3Schools, n.d.) to figure out how to properly use arraylists in my code as I have only ever really used them a couple times in coding.

```
import java.util.ArrayList;

public class Card {
    private int moneyEffect;
    private String desc;
    private ArrayList<Card> cards = new ArrayList<>();

    public Card(String desc, int moneyEffect) {
        this.desc = desc;
        this.moneyEffect = moneyEffect;
    }
    public void cardDeck() {
        cards.clear();
    }
}
```

The Card constructor is used for each card. This shows their description and the amount of money the player gains/loses.

Now I have to create 20 different cards with different scenarios where the player gains/loses money.

I decided that creating all those scenarios were very time consuming and not that important so I decided to use ChatGPT to create those scenarios for me.

(Anon., n.d.)

These are the scenarios it made for me

```
Got it! Here are 20 concise, animal-themed money gain/loss scenarios without mentioning specific animals:
1. You receive a grant for protecting wildlife — collect $150.
2. Your animal sanctuary earns a donation — collect $100.
3. You sell rare animal photos — collect $80.
4. A wildlife documentary pays you for access — collect $200.
5. Your pet training business has a booming season — collect $120.
6. You win a prize at the annual animal show — collect $75.
7. Your animal feed supplier offers a discount — collect $50.
8. A local zoo partners with you for funding — collect $110.
9. You receive a reward for rescuing a wild animal — collect $90.
10. You get sponsorship money for animal care — collect $130.
11. You must repair damage to your animal enclosure — pay $100.
12. Your vehicle is damaged during a wildlife trip — pay $120.
13. You pay a fine for improper animal care — pay $150.
14. Your veterinary bills exceed expectations — pay $90.
15. You pay extra for animal feed due to shortage — pay $80.
16. You are fined for breaking wildlife regulations — pay $130.
17. You cover cleanup costs for invasive species — pay $70.
18. You pay for damages after an animal escapes — pay $60.
19. You cover medical bills for an animal injury — pay $100.
20. You spend money on pest control for your habitat — pay $50.
```

Now to have to put this into my code.

This is what my code looks like now:

For each new card I used the constructor to show what the player owes/gains

```
public class Card { 2 usages
    private final ArrayList<Card> cards = new ArrayList<>(); 2 usages
    private final String desc; 1 usage
    private final int moneyEffect; 1 usage

    public Card(String desc, int moneyEffect) { 20 usages
        this.desc = desc;
        this.moneyEffect = moneyEffect;
    }

    public void cardDeck() { no usages
        cards.clear();
        cards.add(new Card(desc: "You receive a grant for protecting wildlife - collect $150.", moneyEffect: 150));
        cards.add(new Card(desc: "Your animal sanctuary earns a donation - collect $100.", moneyEffect: 100));
        cards.add(new Card(desc: "You sell rare animal photos - collect $80.", moneyEffect: 80));
        cards.add(new Card(desc: "A wildlife documentary pays you for access - collect $200.", moneyEffect: 200));
        cards.add(new Card(desc: "Your pet training business has a booming season - collect $120.", moneyEffect: 120));
        cards.add(new Card(desc: "You win a prize at the annual animal show - collect $75.", moneyEffect: 75));
        cards.add(new Card(desc: "Your animal feed supplier offers a discount - collect $50.", moneyEffect: 50));
        cards.add(new Card(desc: "A local zoo partners with you for funding - collect $110.", moneyEffect: 110));
        cards.add(new Card(desc: "You receive a reward for rescuing a wild animal - collect $90.", moneyEffect: 90));
        cards.add(new Card(desc: "You get sponsorship money for animal care - collect $130.", moneyEffect: 130));
        cards.add(new Card(desc: "You must repair damage to your animal enclosure - pay $100.", moneyEffect: -100));
        cards.add(new Card(desc: "Your vehicle is damaged during a wildlife trip - pay $120.", moneyEffect: -120));
        cards.add(new Card(desc: "You pay a fine for improper animal care - pay $150.", moneyEffect: -150));
        cards.add(new Card(desc: "Your veterinary bills exceed expectations - pay $90.", moneyEffect: -90));
        cards.add(new Card(desc: "You pay extra for animal feed due to shortage - pay $80.", moneyEffect: -80));
        cards.add(new Card(desc: "You are fined for breaking wildlife regulations - pay $130.", moneyEffect: -130));
        cards.add(new Card(desc: "You cover cleanup costs for invasive species - pay $70.", moneyEffect: -70));
        cards.add(new Card(desc: "You pay for damages after an animal escapes - pay $60.", moneyEffect: -60));
        cards.add(new Card(desc: "You cover medical bills for an animal injury - pay $100.", moneyEffect: -100));
        cards.add(new Card(desc: "You spend money on pest control for your habitat - pay $50.", moneyEffect: -50));
    }
}
```

Now I need to make it so the card is picked at random.

```
public static void drawCard() { 2 usages
    int Index = new Random().nextInt(cards.size());
    Card card = cards.get(Index);
}
```

this makes it so a random number from the card index is picked at random and then it gets the card chosen.

In my main I put Card.drawCard(). and then I ran it.

The code didn't run as the method drawCard was running infinitely without any exceptions so I needed to create an exception where there were no cards left.

```
public static void drawCard() { 2 usages
    if (cards.isEmpty()) {
        System.out.println("You don't have any cards in your deck!");
        return;
    }
    int Index = new Random().nextInt(cards.size());
    Card card = cards.get(Index);
    System.out.println("Card: " + card.desc);
}
```

I also made it print out the card that the player got.

I ran the code and it worked successfully

```
Card: A wildlife documentary pays you for access - collect $200.
```

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

Now I need to make it so that it effects the player's balance.

However, I need to make a getter for the balance as I'm not the one making the player class and I need to give myself a random balance as I'm not playing the full game.

This is how my code looked in the end.

```
import java.util.ArrayList;
import java.util.Random;
public class Card { 25 usages
    private static final ArrayList<Card> cards = new ArrayList<>(); 24 usages
    private final String desc; 2 usages
    private static int moneyEffect = 0; 2 usages
    static int balance = 1000; 2 usages

    public Card(String desc, int moneyEffect) { 20 usages
        this.desc = desc;
        this.moneyEffect = moneyEffect;
    }

    public static void cardDeck() { 1 usage
        cards.clear();
        cards.add(new Card(desc: "You receive a grant for protecting wildlife - collect $150.", moneyEffect: 150));
        cards.add(new Card(desc: "Your animal sanctuary earns a donation - collect $100.", moneyEffect: 100));
        cards.add(new Card(desc: "You sell rare animal photos - collect $80.", moneyEffect: 100));
        cards.add(new Card(desc: "A wildlife documentary pays you for access - collect $200.", moneyEffect: 200));
        cards.add(new Card(desc: "Your pet training business has a booming season - collect $120.", moneyEffect: 120));
        cards.add(new Card(desc: "You win a prize at the annual animal show - collect $75.", moneyEffect: 75));
        cards.add(new Card(desc: "Your animal feed supplier offers a discount - collect $50.", moneyEffect: 50));
        cards.add(new Card(desc: "A local zoo partners with you for funding - collect $10.", moneyEffect: 110));
        cards.add(new Card(desc: "You receive a reward for rescuing a wild animal - collect $90.", moneyEffect: 90));
        cards.add(new Card(desc: "You get sponsorship money for animal care - collect $130.", moneyEffect: 130));
        cards.add(new Card(desc: "You must repair damage to your animal enclosure - pay $100.", moneyEffect: -100));
        cards.add(new Card(desc: "Your vehicle is damaged during a wildlife trip - pay $120.", moneyEffect: -120));
        cards.add(new Card(desc: "You pay a fine for improper animal care - pay $150.", moneyEffect: -150));
        cards.add(new Card(desc: "Your veterinary bills exceed expectations - pay $90.", moneyEffect: -90));
        cards.add(new Card(desc: "You pay extra for animal feed due to shortage - pay $80.", moneyEffect: -80));
        cards.add(new Card(desc: "You are fined for breaking wildlife regulations - pay $130.", moneyEffect: -130));
    }

    cards.add(new Card(desc: "You cover cleanup costs for invasive species - pay $70.", moneyEffect: -70));
    cards.add(new Card(desc: "You pay for damages after an animal escapes - pay $60.", moneyEffect: -60));
    cards.add(new Card(desc: "You cover medical bills for an animal injury - pay $100.", moneyEffect: -100));
    cards.add(new Card(desc: "You spend money on pest control for your habitat - pay $50.", moneyEffect: -50));

    public static void drawCard() { 2 usages
        if (cards.isEmpty()) {
            System.out.println("You don't have any cards in your deck!");
            return;
        }
        int index = new Random().nextInt(cards.size());
        Card card = cards.get(index);
        System.out.println("Card: " + card.desc);
        setBalanceEffect();
        System.out.println("Balance: " + getBalance());
    }

    public static int getBalance() { 2 usages
        return balance;
    }

    public static void setBalanceEffect() { 1 usage
        balance = getBalance() + moneyEffect;
    }
}
```

Now I need to test if the player actually gains/loses money.

The test went wrong

```
Card: You are fined for breaking wildlife regulations - pay $130
Balance: 950

Process finished with exit code 0
```

There was clearly an error as the money it lost was wrong so I looked back at my cards and saw that one of my cards was wrong(so I fixed it) but not the one that was drawn.

I then started changing the parts that weren't working randomly until I ended up with:

```
    setbalanceEffect(card);  
  
private int moneyEffect = 0; 2 usages  
  
public static void setbalanceEffect(Card card) { 1 usage  
    balance += card.moneyEffect;  
}
```

So I reran the code.

```
C:\Users\smmom\jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program F.  
Card: You receive a reward for rescuing a wild animal - collect $90.  
Balance: 1090  
  
Process finished with exit code 0
```

It added the money correctly. Now I needed to test if it took away the money correctly.

```
C:\Users\smmom\jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program  
Card: You cover medical bills for an animal injury - pay $100.  
Balance: 900  
  
Process finished with exit code 0
```

It did. 😊

I then went back to my dice class to remove the placeholder for the drawCard() method and added the real one.

Now i needed to test if it draws a card when a double is rolled.

This is my main for the test:

```
public class Main {  
    public static void main(String[] args) {  
        Card.cardDeck();  
        Dice.rollDice();  
    }  
}
```

The test:

```
C:\Users\smmom\jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program F.  
you rolled a: 10  
you rolled a Double!!  
Card: You pay for damages after an animal escapes - pay $60.  
Balance: 940  
  
Process finished with exit code 0
```

It worked.

Now I have finished the development for card and dice. Only animal left for me.

Edit: I went back to my card class and stopped everything from being static.

I then tried to run my code but it didn't work as I didn't have a constructor for the card class with no arguments. So I made one.

```
public Card() {; 1 usage}
```

This is used in the main class so that I can call a new card and wont have to input an argument.

```
public class Main {
    public static void main(String[] args) {
        Player player = new Player(character: 'A', name: "Apple");
        Card card = new Card();
        card.cardDeck();
        card.drawCard(player);
    }
}
```

Card: You pay a fine for improper animal care - pay \$150.  
Balance: 1350  
Process finished with exit code 0

it then worked.

```
import java.util.ArrayList;
import java.util.Random;
public class Card {
    private final ArrayList<Card> cards = new ArrayList<>(); 24 usages
    private String desc; 2 usages
    private int moneyEffect = 0; 2 usages

    public Card() { 1 usage
    }

    public Card(String desc, int moneyEffect) { 20 usages
        this.desc = desc;
        this.moneyEffect = moneyEffect;
    }

    public void cardDeck() {...}
    public void drawCard(Player player) { 1 usage
        if (cards.isEmpty()) {
            System.out.println("You don't have any cards in your deck!");
            return;
        }
        int index = new Random().nextInt(cards.size());
        Card card = cards.get(index);
        System.out.println("Card: " + card.desc);
        player.setMoney(card.moneyEffect);
        System.out.println("Balance: " + player.getMoney());
    }
}
```

## Animal class

I need to choose 24 different animals

I decided to have 6 mammals, 6 fish, 6 reptiles, 6 birds.

I decided to make the stop cost be a quarter of the buy cost to make it not too expensive to land on someone elses animal

## Mammals:

Tiger – buy:600 stopcost:150

Lion – buy:500 stopcost:125

Dog – buy:300 stopcost: 75

Cow – buy: 200 stopcost:50

Horse – buy: 400 stopcost: 100

Sheep – buy: 150 stopcost:38

### **Fish:**

Clownfish – buy:300 stopcost: 75

Salmon – buy: 100 stopcost: 20

seahorse – buy 400 stopcost: 100

Pufferfish – buy 200 stopcost: 50

Shark – buy: 500 stopcost: 125

Dolphin – buy: 600 stopcost: 150

### **Reptiles:**

Snake – buy: 300 stopcost: 75

Crocodile buy: 400 stopcost: 100

Turtle – buy: 200 stopcost: 50

Chameleon – buy: 500 stopcost: 125

Lizard – buy: 100 stopcost: 20

Gecko – buy: 600 stopcost: 150

### **Birds:**

Eagle – buy: 600 stopcost: 150

Vulture – buy: 500 stopcost: 125

Owl – buy: 400 stopcost: 100

Parrot – buy; 300 stopcost: 75

Seagull – buy: 200 stopcost: 50

Chicken – buy: 100 stopcost: 20

For this class I have decided to start coding straight away as I will have to do a lot of coding (each class for each animal). I will be using inheritance to make coding easier. For help I went back to some old work I did earlier in the year to inspire me.

```
↓ public class Animal {  
    private String species; 2 usages  
    private int level; 2 usages  
    private int stopCost; 2 usages  
    private int buyCost; 2 usages  
    private Player Owner; 2 usages  
  
↓ public class Animal {  
    private String species; 2 usages  
    private int level; 2 usages  
    private int stopCost; 2 usages  
    private int buyCost; 2 usages  
    private Player Owner; 2 usages
```

These will be put in the animal constructor and inherited by each animal to store their info.

I then created the constructor and all the getters that will be needed.

```
public class Animal {  
    private String species; 2 usages  
    private int level; 2 usages  
    private int stopCost; 2 usages  
    private int buyCost; 2 usages  
    private Player Owner; 2 usages  
  
    public Animal(String species, int level, int stopCost, int buyCost, Player Owner) {  
        this.species = "";  
        this.level = 1;  
        this.stopCost = 0;  
        this.buyCost = 0;  
        this.Owner = null;  
    }  
  
    public int getLevel() { no usages  
        return level;  
    }  
    public String getSpecies() { no usages  
        return species;  
    }  
    public int getStopCost() { no usages  
        return stopCost;  
    }  
    public int getBuyCost() { no usages  
        return buyCost;  
    }  
    public Player getOwner() { no usages  
        return Owner;  
    }
```

```
    public boolean purchase(Player player) { no usages  
        if (player.getMoney() >= this.buyCost && owner == null) {  
            this.owner = player;  
            return true;  
        }  
        System.out.println("You cannot buy this animal");  
        return false;  
    }
```

This is used if the player wants to buy the animal that it lands on. It can only happen if there is no other owner for the animal and if the player has enough money to pay for the animal.

```
public boolean upgrade(Player player) { no usages
    if (player.getMoney() >= this.buyCost) {
        this.level += 1;
    }
    System.out.println("You cannot upgrade this animal");
    return false;
}
```

This is used if the player wants to upgrade the animal. They can only do it if their balance is greater than the buy cost.

```
public boolean upgrade(Player player) { no usages
    if (player.getMoney() >= this.buyCost) {
        this.level += 1;
        this.stopCost += stopCost;
    }
    System.out.println("You cannot upgrade this animal");
    return false;
}
```

I decided to make it so that the stop cost increases by 2 so there is an actual reason to level up. (may change later as it may be too much money).

I then decided that to make a new method called getInfo() so that we can see all the data for each animal.

```
public void getInfo() { no usages
    System.out.println("Species: " + this.species);
    System.out.println("Level: " + this.level);
    System.out.println("Stop cost: " + this.stopCost);
    System.out.println("Buy cost: " + this.buyCost);
    System.out.println("Owner: " + this.owner);
}
```

Now I need to create the method for notifying the player that they need to pay the stop cost, or if there is no owner for the animal.

```
public void stopNotification() { no usages
    if (owner != null) {
        System.out.println("You have landed on " + owner.getName() + "'s pet " + getSpecies() + " you have to pay " + stopCost);
    } else {
        System.out.println("You can buy this animal");
    }
}
```

I have now created the animal class:

```

public class Animal { no usages
    private String species; 3 usages
    private int level; 4 usages
    private int stopCost; 6 usages
    private int buyCost; 5 usages
    private Player owner; 7 usages

    public Animal(String species, int level, int stopCost, int buyCost) { no usages
        this.species = "";
        this.level = 1;
        this.stopCost = 0;
        this.buyCost = 0;
        this.owner = null;
    }

    public void stopNotification() { no usages
        if (owner != null) {
            System.out.println("You have landed on " + owner.getName() + "'s pet" + getSpecies() + " you have to pay" + stopCost);
        }
        else {
            System.out.println("You can buy this animal");
        }
    }

    public boolean purchase(Player player) { no usages
        if (player.getMoney() >= this.buyCost && owner == null) {
            this.owner = player;
            return true;
        }
        System.out.println("You cannot buy this animal");
        return false;
    }

    public boolean upgrade(Player player) { no usages
        if (player.getMoney() >= this.buyCost) {
            this.level += 1;
            this.stopCost += stopCost;
        }
        System.out.println("You cannot upgrade this animal");
        return false;
    }
}

```

```

public void getInfo() { no usages
    System.out.println("Species: " + this.species);
    System.out.println("Level: " + this.level);
    System.out.println("Stop cost: " + this.stopCost);
    System.out.println("Buy cost: " + this.buyCost);
    System.out.println("Owner: " + this.owner);
}

public int getLevel() { no usages
    return level;
}

public String getSpecies() { 1 usage
    return species;
}

public int getStopCost() { no usages
    return stopCost;
}

public int getBuyCost() { no usages
    return buyCost;
}

public Player getOwner() { no usages
    return owner;
}
}

```

Now I need to create all 24 child classes. (the way I did it may have been highly impractical).

The first child class I make will be used to test if the animal class actually works.

```

public class Tiger extends Animal { 2 usages

    Tiger () { 1 usage
        super( species: "Mammal", level: 1, stopCost: 150, buyCost: 600 );
    }
}

```

This is my tiger class. It uses 'super' to access the constructor from the animal class. Then it fills in all the information.

Now I'm going to test if I can get the information for the tiger

This will be my main for the test.

```

public class Main {
    public static void main(String[] args) {
        Tiger tiger = new Tiger();
        tiger.getInfo();
    }
}

```

I ran the code and it did not work as I thought it would

```
Species:  
Level: 1  
Stop cost: 0  
Buy cost: 0  
Owner: null  
  
Process finished with exit code 0
```

It only outputted what was in the animal constructor. So I changed it to this.

```
public Animal(String type, int level, int stopCost, int buyCost) {  
    this.type = type;  
    this.level = level;  
    this.stopCost = stopCost;  
    this.buyCost = buyCost;  
    this.owner = null;  
}
```

Now it should work as it allows the tiger to input its values for it.

```
Species: Mammal  
Level: 1  
Stop cost: 150  
Buy cost: 600  
Owner: null  
  
Process finished with exit code 0
```

it worked!

I realised that in my purchase() & upgrade() method, there is nothing that actually takes money away from the player.

So I fixed them both

```
public boolean purchase(Player player) {  
    if (player.getMoney() >= this.buyCost && owner == null) {  
        this.owner = player;  
        player.setMoney(player.getMoney() - this.buyCost);  
        return true;  
    }  
    System.out.println("You cannot buy this animal");  
    return false;  
}  
  
public boolean upgrade(Player player) {  
    if (player.getMoney() >= this.buyCost) {  
        this.level += 1;  
        this.stopCost += stopCost;  
        player.setMoney(player.getMoney() - this.buyCost);  
    }  
    System.out.println("You cannot upgrade this animal");  
    return false;  
}
```

I also removed static from all my classes.

Now I need to test if the stop cost notification works.

First I need to make a test to see that a player owns the tiger

```

public class Main {
    public static void main(String[] args) {
        Tiger tiger = new Tiger();
        tiger.getInfo();
        Player player= new Player( character: 'J', name: "john");
        if(tiger.purchase(player)) {
            System.out.println("You purchased a tiger!");
        }
        else {
            System.out.println("You don't have a tiger!");
        }
        tiger.getInfo();
    }
}

```

This is my test. If the player ends up owning the tiger it will print that it purchased it. I will then double check with the getInfo() method to see if it is the new owner.

```

Species: Mammal
Level: 1
Stop cost: 150
Buy cost: 600
Owner: null
You cannot buy this anim
You don't have a tiger!
Species: Mammal
Level: 1
Stop cost: 150
Buy cost: 600
Owner: null

```

It failed. It doesn't recognize the player as the owner. This is because the player has less money than the amount the tiger is worth.

I tried again but increased the money for the player by 500.

```

Species: Mammal
Level: 1
Stop cost: 150
Buy cost: 600
Owner: null
You purchased a tiger!
Species: Mammal
Level: 1
Stop cost: 150
Buy cost: 600
Owner: Player@30f39991

```

It worked.

Now I need to test that the stop cost notification worked.

```

public class Main {
    public static void main(String[] args) {
        Tiger tiger = new Tiger();
        Player player = new Player( character: 'S', name: "Suliman");
        tiger.purchase(player);
        tiger.stopNotification();
    }
}

```

This is the main I used to test it.

```
C:\Users\smmon\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\  
You have landed on Suliman's pet Mammal, you have to pay $150  
  
Process finished with exit code 0
```

It Worked!

The last thing I need to test is the upgrade() method.

I had to increase the players budget again by 500 so I can do it.

```
C:\Users\smmon\.jdks\openjdk-24.0.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.1.  
You cannot upgrade this animal  
Species: Mammal  
Level: 2  
Stop cost: 300  
Buy cost: 600  
Owner: Player@30f39991  
  
Process finished with exit code 0
```

The test kind of worked as the level and stop cost was increased, but it outputted that the player couldn't upgrade the animal.

Turns out that I forgot a 'return true' inside the if statement.

I reran the code.

```
C:\Users\smmon\.jdks\openjdk-24.0.1\bin\j  
Species: Mammal  
Level: 2  
Stop cost: 300  
Buy cost: 600  
Owner: Player@30f39991  
  
Process finished with exit code 0
```

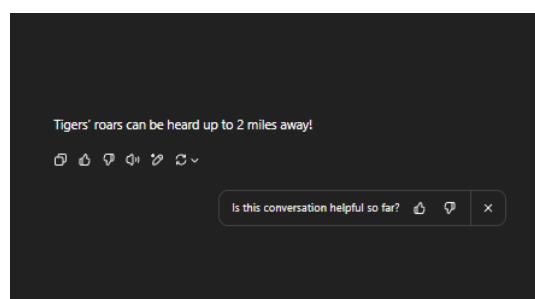
It Worked!

I also need to create a fun fact for each animal in the get info method.

I created a new string called 'funFact' and put it into my constructor for animal

```
public Animal(String type, int level, int stopCost, int buyCost, String funFact) {  
    this.type = type;  
    this.level = level;  
    this.stopCost = stopCost;  
    this.buyCost = buyCost;  
    this.funFact = funFact;  
    this.owner = null;  
}
```

I then used AI to generate the fun facts for me as it is a pretty time consuming process. (Anon., n.d.). This is the fact it gave me for the tiger.



I then tested it with it with this main

```
public class Main {
    public static void main(String[] args) {
        Tiger tiger = new Tiger();
        tiger.getInfo();
    }
}
```

```
C:\Users\summoner\Desktop\openJDK-24.0.1\bin>
Species: Mammal
Level: 1
Stop cost: 150
Buy cost: 600
Owner: null

Process finished with exit code 0
```

It didn't print the fun fact

I forgot to add the funfact to the getInfo() method.

```
Species: Mammal
Level: 1
Stop cost: 150
Buy cost: 600
Owner: null
FunFact: Tigers' roars can be heard up to 2 miles away!

Process finished with exit code 0
```

It worked.

```
public class Animal {
    private String type;
    private int level;
    private int stopCost;
    private int buyCost;
    private String funFact;
    private Player owner;

    public Animal(String type, int level, int stopCost, int buyCost, String funFact) {
        this.type = type;
        this.level = level;
        this.stopCost = stopCost;
        this.buyCost = buyCost;
        this.funFact = funFact;
        this.owner = null;
    }

    public void stopNotification() {
        if (owner != null) {
            System.out.println("You have landed on " + owner.getName() + "'s pet " + getSpecies() + ", you have to pay $" + stopCost);
        } else {
            System.out.println("You can buy this animal!");
        }
    }

    public boolean purchase(Player player) {
        if (player.getMoney() >= this.buyCost && owner == null) {
            this.owner = player;
            player.setMoney(player.getMoney() - this.buyCost);
            return true;
        }
        System.out.println("You cannot buy this animal!");
        return false;
    }
}
```

Now im done with testing I just need to create the other 23 child classes.

I did it as different subclasses of the class animal to use some inheritance and encapsulation.

I have now finished developing all my pets.

I also used the same source for the ai for each other qanimal but I forgot to take screenshots for each one.

Here it all is:

```

public boolean upgrade(Player player) { no usages
    if (player.getMoney() >= this.buyCost) {
        this.level += 1;
        this.stopCost += stopCost;
        player.setMoney(player.getMoney() - this.buyCost);
        return true;
    }
    System.out.println("You cannot upgrade this animal");
    return false;
}

public void getInfo() { 1 usage
    System.out.println("Species: " + this.type);
    System.out.println("Level: " + this.level);
    System.out.println("Stop cost: " + this.stopCost);
    System.out.println("Buy cost: " + this.buyCost);
    System.out.println("Owner: " + this.owner);
    System.out.println("FunFact: " + this.funFact);
}

public int getLevel() { return level; }

public String getSpecies() { 1 usage
    return type;
}

public int getStopCost() { no usages
    return stopCost;
}

public int getBuyCost() { no usages
    return buyCost;
}

public Player getOwner() { no usages
    return owner;
}
}

```

```

public class Tiger extends Animal { 2 usages

    Tiger () { 1 usage
        super( type: "Mammal", level: 1, stopCost: 150, buyCost: 600, funFact: "Tigers' roars can be heard up to 2 miles away!" );
    }
}

```

```

public class Lion extends Animal { no usages
    public Lion() { no usages
        super( type: "Mammal", level: 1, stopCost: 125, buyCost: 500, funFact: "Lions can run up to 50 mph");
    }
}

```

```

public class Dog extends Animal { no usages
    public Dog () { no usages
        super( type: "Mammal", level: 1, stopCost: 75, buyCost: 300, funFact: "Dogs' noses have up to 300 million scent receptors");
    }
}

```

```

public class Cow extends Animal { no usages
    public Cow() { no usages
        super( type: "Mammal", level: 1, stopCost: 75, buyCost: 300, funFact: "Cows have almost 360-degree vision");
    }
}

```

```

public class Horse extends Animal { no usages
    public Horse() { no usages
        super( type: "Mammal", level: 1, stopCost: 100, buyCost: 400, funFact: "Horses sleep standing up most of the time");
    }
}

```

```

public class Sheep extends Animal { no usages
    public Sheep() { no usages
        super( type: "Mammal", level: 1, stopCost: 38, buyCost: 150, funFact: "Sheep have excellent memories and can recognize faces for years");
    }
}

```

```

public class Clownfish extends Animal{ no usages
    public Clownfish(){ no usages
        super( type: "Fish", level: 1, stopCost: 75, funFact: 300, "Clownfish can change their gender if needed!");
    }
}

public class Salmon extends Animal { no usages
    public Salmon() { no usages
        super( type: "Fish", level: 1, stopCost: 20, buyCost: 100, funFact: "Salmon can jump up waterfalls to reach their spawning grounds");
    }
}

public class Pufferfish extends Animal { no usages
    public Pufferfish() { no usages
        super( type: "Fish", level: 1, stopCost: 50, buyCost: 200, funFact: "Pufferfish can puff up to three times their normal size to scare predators");
    }
}

public class Seahorse extends Animal {
    public Seahorse() {
        super( type: "Fish", level: 1, stopCost: 100, buyCost: 400, funFact: "Male seahorses are the ones that get pregnant and give birth");
    }
}

public class Shark extends Animal { no usages
    public Shark() { no usages
        super( type: "Fish", level: 1, stopCost: 125, buyCost: 500, funFact: "Dolphins have names for each other and can recognize themselves in mirrors");
    }
}

public class Dolphin extends Animal{ no usages
    public Dolphin(){ no usages
        super( type: "Fish", level: 1, stopCost: 150, buyCost: 600, funFact: "Dolphins have names for each other and can recognize themselves in mirrors");
    }
}

public class Snake extends Animal { no usages
    public Snake() { no usages
        super( type: "Reptile", level: 1, stopCost: 75, buyCost: 300, funFact: "Some snakes can \"hear\" by feeling vibrations through the ground");
    }
}

public class Crocodile extends Animal { no usages
    public Crocodile() { no usages
        super( type: "Reptile", level: 1, stopCost: 100, buyCost: 400, funFact: "Crocodiles can hold their breath underwater for over an hour");
    }
}

public class Turtle extends Animal { no usages
    public Turtle() { no usages
        super( type: "Reptile", level: 1, stopCost: 50, buyCost: 200, funFact: "Turtles can live for over 100 years");
    }
}

public class Chameleon extends Animal{ no usages
    public Chameleon(){ no usages
        super( type: "Reptile", level: 1, stopCost: 125, buyCost: 500, funFact: "Chameleons can move their eyes independently to see in two directions at once");
    }
}

public class Lizard extends Animal { no usages
    public Lizard() { no usages
        super( type: "Reptile", level: 1, stopCost: 20, buyCost: 100, funFact: "Some lizards can regrow their tails if they lose them");
    }
}

public class Gecko extends Animal { no usages
    public Gecko() { no usages
        super( type: "Reptile", level: 1, stopCost: 150, buyCost: 600, funFact: "Geckos can walk upside down on ceilings thanks to tiny hairs on their feet");
    }
}

```

```

public class Eagle extends Animal { no usages
    public Eagle() { no usages
        super( type: "Bird", level: 1, stopCost: 150, buyCost: 600, funFact: "Eagles have incredible eyesight-8 times sharper than humans");
    }
}

public class Vulture extends Animal { no usages
    public Vulture() { no usages
        super( type: "Bird", level: 1, stopCost: 125, buyCost: 500, funFact: "Vultures can eat rotten meat without getting sick");
    }
}

public class Parrot extends Animal { no usages
    public Parrot() { no usages
        super( type: "Bird", level: 1, stopCost: 75, buyCost: 300, funFact: "Parrots can mimic human speech and sounds");
    }
}

public class Owl extends Animal{ no usages
    public Owl(){ no usages
        super( type: "Bird", level: 1, stopCost: 100, buyCost: 400, funFact: "Owls can rotate their heads up to 270 degrees");
    }
}

public class Seagull extends Animal { no usages
    public Seagull() { no usages
        super( type: "Bird", level: 1, stopCost: 50, buyCost: 200, funFact: "Seagulls can drink both fresh and saltwater");
    }
}

public class Chicken extends Animal { no usages
    public Chicken() { no usages
        super( type: "Bird", level: 1, stopCost: 20, buyCost: 100, funFact: "Chickens can remember over 100 different faces");
    }
}

```

A few days later I realised that instead of making subclasses for each animal I could've just made an arraylist in the animal class. I realised this too late to actually make a change in the code.

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

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

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

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

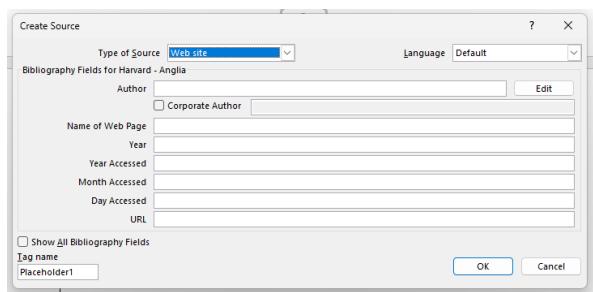
The easiest way to track this is to use the inbuilt References tab in Word – when you ever you need to cite that you've accessed a website you click the icon shown on the next page, select the type of source and the fill in the form – this will cause something like this: (Zucker, 2025) to appear in your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

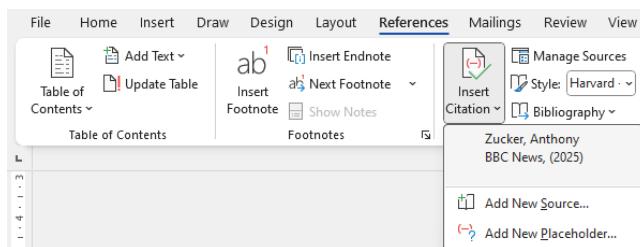
## References Section



## Source Form



## Reusing a Source



## Reference List

Anon., n.d. *ChatGPT*. [Online]

Available at: <https://chatgpt.com/c/683f586b-95a8-8000-b215-c8c3414f6447>

[Accessed 03 06 2025].

Lovelace, A., n.d. *Random Number Generation*. [Online]

Available at: [https://adacomputerscience.org/concepts/progcon\\_random\\_numbers](https://adacomputerscience.org/concepts/progcon_random_numbers)

[Accessed 03 06 2025].

W3Schools, n.d. *Java ArrayList*. [Online]

Available at: [https://www.w3schools.com/java/java\\_arraylist.asp](https://www.w3schools.com/java/java_arraylist.asp)

[Accessed 10 6 2025].

Zucker, A., 2025. *BBC News*. [Online]

Available at: <https://www.bbc.co.uk/news/articles/c5ypz7yx73wo>

[Accessed 29 April 2025].

# 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.