FACULTY OF SCIENCE, ENGINEERING AND COMPUTING

School of *Computer Science & Mathematics*

BSc DEGREE

IN

Computer Games Programming (Hons)

PROJECT REPORT

Zahra Fatima Islam Miah

K1940669

Feline Frenzy

21st June 2023

Supervisor: Francik Jarek

Kingston University London

Plagiarism Declaration

The following declaration should be signed and dated and inserted directly after the title page of your report:

Declaration

I have read and understood the University regulations on plagiarism, and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computer Science and Mathematics. All verbatim citations are indicated by double quotation marks ("…"). Neither in part nor in its entirety have I made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as their own work.

Date: 21st June 2023

Signature

Zahra Fatima Islam Miah

Table of Contents

Introduction **6**
Summary 6
Game Description 6
Aim of the Game 6
Objectives 6
Single-player mode 6
Multiplayer modes 7
Kitty-Cat Snatch 7
Prop-hunt based game 7
Personal Aims and Objectives 7
Summary of what is to come 7

State of the Art Review **9**
Similar Games 9
Hello Neighbour 9
Assassin's Creed Brotherhood 10
Petz: Catz 2(DS) 11
Nintendogs: Dalmatian and Friends 12

Development Tools and Technologies **13**
Engine and Programming Languages 13
Technologies such as Artificial Intelligence 13

Analysis, Requirements, Specification and Development methodology **14**
MoSCoW Analysis for Feline Frenzy 14
Potential Difficulty 15
Customisation 15
Caring for the Cat's 15
Prioritisation of Requirements 15
Development Methodology 16

Game Design **18**
Conceptual Design 18
Technical Design 21
Flow Charts 22
Implementation **23**
Single-Player Mode 23
Player Movement 23
Camera Script 23
Cat Behaviour 24
Bait 25
Character Customisation 26
Timer 30
Enemy AI 30
Level Loading and Menu 31
Multiplayer Mode 31
Prop-Hunt 31
Customisation Syncing 6
Character Customisation 33
Testing and Evaluation **35**
Black Box Testing 35
MoSCoW Review 39
Player Feedback 41
Prototype Content Overview 45
Protype 1 45
Protype 2 47
Protype 3 [Final Outcome] 49
Ethical, Legal, Social Issues and Data Security Issues **50**
PEGI Rating 50
Intellectual Property 50
Data Security 50
Representation and Ethical Issues 51

Critical Review and Conclusion **52**
Summary of Achievements 52
Testing Results 52
Time Management 52
What could have been done better 52
Personal Achievements and what was learned 52
Future Works 53
Appendix **54**
Project Proposal 54
Gantt Chart 54
Interim Demo [*Video*] 54
Bibliography **54**

Introduction

Summary

Feline Frenzy is a 3D arcade stealth game with both single-player and multiplayer functionality developed in Unity for PC. It is a first person stealth based game where you play as a frenzied owner whose cats have escaped their cat haven and infiltrated their neighbours' houses. It is set in the real world with people and cats.

Game Description

Feline frenzy will be a first-person stealth game where you play as a crazed cat individual whose dream is to live surrounded by cats. Your cat's have left your home to wander around in the streets and ended up in your neighbour's house. Use whatever means at your disposal to achieve the utopia of your dreams -trespass into your neighbour's house, lure in your cats with their favourite food, or sneak up behind them. However, retrieving your cats isn't as easy as it sounds! If you're seen by a cat, it will run away and alert the neighbour. Steer clear away and out of sight from the neighbours to avoid becoming a wanted criminal.

Aim of the Game

The primary aim of the game is to complete the level by finding and bringing the cats to the players base within the time limit. In multiplayer modes there will be both a competitive mode and collaborative modes. In the competitive multiplayer mode players must compete to catch the greatest number of cats before the time limit. In the competitive mode, players must work together to survive without being caught for a set amount of time. The number of multiplayer modes implemented is dependent on prioritisation and time dependencies and may be dropped or discarded closer to the project deadline.

Objectives

Single-player mode:

Each level will have a set amount of cats to catch, in the beginning levels there will be a small number 3 – 5, and in more difficult levels there can be up to 10 cats to find. If the player is seen by the owner, a chase sequence will be initiated, and the player must flee the scene until they are out of their line of sight or reach their base before they are caught. If they are caught by the owner, they are returned to their spawn point (*their base*), and any cat's being carried will be returned. In this scenario getting caught means game over and having to restart the infiltration into their home and grabbing the cats.

Players can place bait to entice the cats to come closer. The cats will actively seek food and head to the location. Players have a higher chance of capturing the cat if they place their favourite food. If a cat is approached prior to using a distraction, it will flee and alert the enemy of commotion. Players can also scavenge the house for more cat food. Cats will have randomised preference for food as well as personality if they are more approachable or on higher alert around strangers.

Once the player has successfully captured a cat, they must bring the cat back to their house without being seen. If the player is caught while still trespassing, they will drop the cat and return to their spawn point empty handed. Each level will have different cats and owners with different patrol points. To clear each level, the player must search the house for cats and bring them all home.

Neighbours [enemy] will chase the player if they are within their line of sight, if the player makes too much noise, or if the cat has alerted them. If the player moves out of the neighbour's line of sight, they will investigate the area before returning to their activities around the house (patrol points). If the player gets caught by the owner multiple times, wanted posters will be placed and notoriety (level of notoriety is invisible to the player) will be introduced. With high notoriety, neighbours will be on high alert and a new type of enemy will be introduced – the police. The police will begin to patrol the area. Players can remove these wanted posters to reduce the likelihood of getting caught. If the player gets caught, they will be thrown out of the house and return to their spawn point.

In order to help the player to catch the cats, a cat journal will be present (*See Technology for more information)* which displays information about the cats on the current level such as the cat's name, colour of the cat, preferred food, and personality.

Multiplayer modes:

Kitty-Cat Snatch

A party of 4 players compete to catch the greatest number of cats within the set time limit. Players will have to bring a cat back to their own designated base to keep it. Other players can sabotage and steal cats while they are being transported to their area.

Prop-hunt based game:

Once assigned to a lobby, players will each get a randomised role assigned to them – owner or cat thief. Thieves will then have to survive for the duration of the game (not get caught by the owner). The player posing as the owner (there will be one owner in each lobby) will need to catch each thief before the time limit is up. Owners can set traps for thieves to slow them down. Thieves will switch between predetermined objects depending on the map rather than use a raycast to check if the object can be used as a disguise, as raycasts are more costly to the CPU. This multiplayer mode has since been discarded. (*See Testing and Evaluation section*)

The aim of the planned project *Feline Frenzy* is to entertain the target audience – the idea of stealing cats is quite comical as cats are very independent animals who enjoy their time undisturbed napping in warm sunny places. The game is aimed both at people who own cats, and those who don't. People who don't own a cat can enjoy trying to catch one, and people who do will be familiar with the personality and attitude of the cats. The primary goal is to collect all the cats on each level and bring them to the base without being seen by the enemy (*cat's owner*).

Personal aims and objectives

My primary personal objective for Feline Frenzy is to make an enjoyable game where players can interact with each other via multiplayer modes and enjoy a separate single player game. I also aim to incorporate life-like behaviour in the cat's AI with functioning wander, seek and flee mechanics. The cats should be able to identify when the player comes too close and flee depending on if there is food or not in cat-like fashion.

Summary of what is to come

This report is divided into different sections and will cover the different stages of the project.

The first section is a state-of-the-art review, which will look at games similar to Feline Frenzy's mechanics and art-style as well which areas influenced the development of the game. The tools and technology used for development will also be discussed in this section.

The next section is an analysis of the requirements of this game and outlines the specifications as well as the initial MoSCoW analysis of this project and what areas have been implemented or discarded.

The third section will discuss the design of the game, both the conceptual and technical areas of the project. Appearances and aesthetics of the game will be covered in the conceptual section and in the technical section, the architecture and structure of components found in the game will be discussed.

The next section will cover detail regarding the implementation of Feline Frenzy and discuss how certain mechanics and features of the game were constructed.

The fifth component will be testing. This section discusses the testing performed during the developmental stages of the game, as well as any issues identified and how they were resolved. It will also cover areas which were improved upon after receiving player feedback.

The sixth part of the report analyses the legal, social, security and ethical issues which may be found within the game, how they were addressed, and the usage of software and assets.

The final section of the report is a critical review and conclusion of Feline Frenzy. Main achievements are highlighted as well as areas which could be improved upon in future updates.

State of the Art Review:

This section will look at games containing similar mechanics and themes to Feline Frenzy and analyse the impact on the final outcome.

Similar games

The following games had an influence on Feline Frenzy for the mechanics implemented, approach to AI behaviour or aesthetics.

Hello Neighbour:

[*Figure 1: Hello Neighbor (2018)*]

Hello Neighbor is a first-person stealth horror puzzle game where the player must investigate the house of their strange neighbour. The game is contains very similar mechanics to *Feline Frenzy* such as the enemy AI. The goal is to solve the puzzles without being caught by the neighbour, when the player is caught by the enemy AI they are returned to their house. The enemy has the ability to determine the route the player will take based off of percentages in the area as showcased during the AI rundown of the pre-alpha stage of the game's release (tinyBuildGAMES, 2017).

This value is constantly changing since the player can decide to find alternate routes. It is to be noted that the neighbour does not know the exact location of the player, but analyses objects in the scene and searches in areas where the player was last seen. As an added layer of complexity – the game features a trap setting mechanic which is triggered if the player is caught. While the player is respawning, traps have been placed in the last area the player was caught in – which when interacted with will alert the neighbour. This means that the player will have to either deactivate the traps or find a way around them.

In order to deactivate the traps, the player can throw items currently equipped at them, there are two types of traps which have different properties; security camera, makes a loud sound and causes the neighbour to head towards the location and bear traps, which freezes the players movement for

a short period of time. Each instance of getting caught will add another trap to the location – so there can be a large amount of traps preventing the player to access certain areas.

The chase sequence is made fun to play with additional mechanics – the neighbour will throw items at the player to slow them down or block their view.

Assassin's Creed Brotherhood:

[*Figure 2: Assassins Creed Brotherhood (2010)*]

Assassins' creed Brotherhood is a third person story orientated stealth video game. The games primary goals are to complete parts of the story through killing specific enemies while staying hidden and out of sight. The game features a mechanic similar to the stealth and line of sight behaviour implemented in *Feline Frenzy*.

Once the player has progressed a certain amount into the story and continues to kill their targets. They will start gaining a poor reputation. This makes it difficult for the player to blend in with their surroundings to complete the contract as civilians will flee and alert guards patrolling. Before this mechanic was introduced to the player, enemy guards will not attack on sight unless the player attacks them first.

After each kill, the player has a certain amount of reputation which will need to be reduced. Each kill is worth a certain amount of poor reputation, and in order to reduce it a number of wanted posters must be removed from the area. The more people are killed, the larger the number of wanted posters to remove before being able to do tasks such as kill, negotiate with businessmen or take a stroll in the streets.

Both games add a layer of complexity which have inspired various mechanics in *Feline Frenzy*. These include notoriety and wanted poster, addition of a second enemy after notoriety is above a certain amount, and the AI search and investigation behaviours. These mechanics will in turn bring a better experience to the player and make it more fun and enjoyable to play.

Petz: Catz 2 (DS):

[*Figure 3: Petz: Catz 2 DS (2007)*]

Catz 2 is a pet simulation game where the player takes care of a cat. It shares similar mechanics to be implemented in the proposed game when the player enters their home. At the start of the game the player can choose a cat from a variety of different breeds and name them.

The cat's hunger and thirst will drain when playing with the cat and completing obstacle courses. The player needs to pay attention to the cat's status which is always shown at the top of the DS screen (*Figure 5 top left*). The mood of the cat can change drastically if the hunger drains, their litter box has not been replenished or cleaned, or if they have not been given enough affection. The player cannot always train the cat, they must also pet the cat for a certain amount for it to remain happy. The cat can become upset or angry if the player does not care for it correctly or give it enough attention. If the cat is neglected it will take time before they will trust the player again.

When the player completes obstacle courses, they gain in game currency which can be used to buy costumes for the cat and new toys and food. Certain cat's with specified personalities will have different preferences for food and toys. The cat's current mood is also an indicator if the cat wants to play or nap.

The game encourages the player to earn the cats trust and take good care of it by giving achievements when they do so. When all the achievements have been completed, the player can unlock exotic breeds such as a lion, tiger or leopard cub. These exotic breeds are easier to play with to complete the obstacle course but are very playful – so the player will need to spend more time using the cat's energy.

Nintendogs: Dalmatian and Friends:

[*Figure 4: Nintendogs: Dalmatian and Friends (2005)*]

Nintendogs is another pet simulation game which shares similar mechanics to *Feline Frenzy*. At the start of the game, they can choose from a selection of breeds at a kennel. Each Nintendog game features a breed which can otherwise be bought in the in-game shop (*Labrador, Dalmatian, Chihuahua, Shiba Inu, and Dachshund*). The player can wash, brush, feed, walk, and train their dog for competitions. The player earns in-game currency through competing in three different competitions: agility, disc throwing and obedience.

The game uses voice commands to enable the player to call the dogs and teach them tricks. These tricks are then used in the obedience trial. The goal for each competition is to get the highest score (beat the NPC's dog) which earns the player 1st place. Earning 1st place in competitions in succession will increase both the difficulty and prize money. If the player comes in 3rd place (*which is last*) they will go back to the previous difficulty.

In game currency can be used to purchase food, toys, accessories, and another puppy. The player can have up to 3 dogs at a time. Dogs have different personalities which can be viewed when purchasing them. More energetic dogs will be good for agility and disc throwing competitions but poor for obedience trials.

The dog can become upset and angry with the player if they have been neglected (*If the player has not played the game in a certain amount of time*) or if they pull on the dogs' ears or tail. If the dog has been neglected, they will not respond to the player's voice or commands – which results in the dog forgetting tricks. The dog can get dirty if they have been outside or some time has passed – if this happens the player can give them a bath.

The player can view the dog's status (*Hunger, Thirst, Cleanliness*) when they click on their icon, they can also see the list of tricks they have been taught. This is similar to *Feline Frenzy's* cat journal.

The latest addition to the Nintendogs franchise includes cats (Nintendogs + Cats, 2011), the player cannot however purchase a cat from the start of the game, but only once they are able to afford one through in game currency. The cats cannot be taught tricks or enter competitions but only fed and pet.

Development Tools and Technologies:

Engine and Programming Language:

The game will be written using the Unity game engine with the C# language. Unity was chosen as opposed to other programmes and engines primarily for the robust physics engine. The user interface is much preferred over the Unreal Engine due to its simplicity. The Unreal Engine relies heavily on a visual scripting system (*blueprints*) as opposed to C#.

C# is very optimized and has reduced issues with memory due to having memory management through a built-in garbage collection. Garbage collection automatically reallocates valuable memory space used by objects which are no longer needed. (*Microsoft, 2022*)

For the multiplayer game modes, an online networking framework will be used. At the start of the project there were two options available: Photon Pun (*PUN 2 – FREE, 2018)* and Unity Netcode (*Netcode for GameObjects, 2022)*. Both allow for up to 20 players in one server, which is ideal for having 5 sets of 4 players in each lobby. Netcode selects a designated host from the list of players in the current lobby, all information such as messages and player position are sent to the host (acting as an intermediate) then through the relay server. Sending information this way takes far longer (*4 hops*).

If the current host leaves the game, the session will end – this can cause player frustration. With Photon however information is *not* sent through the host but directly to the relay server (2 *hops*) which means that messages are sent significantly faster to other players on the network. Unlike Netcode, if the current host leaves the game, the session will not end – instead another host is selected from the current list of players. As Netcode is still relatively new, there is not many tutorials or documentation about issues that could occur. Photon was used as the chosen framework for *Feline Frenzy*.

Technologies such as Artificial Intelligence:

The game must feature several non-playable characters (*NPC*) with independent behaviour systems (*Neighbours & Cats*). These enemies and NPC's will have behavioural AI such as seek and wander. The enemy agents (*neighbour*) will have a patrolling system implemented; they will also be able to investigate their surroundings for any changes in the environment. When a stimulus is detected, they will begin to investigate the area and known places the player has been seen in.

The wander AI will project a circle Infront of the agent and select a random point along the circumference. If no changes are detected, the agent will proceed to follow the point at random intervals, making the behaviour seem more realistic (*Francik, J. (2020*)).

A percentage will be applied to each area in the house, when the player walks through the area, the percentage is increased by a certain amount. Enemy agents will then traverse through the places with the highest percentages if they have detected the player. If the player is within their line of sight at this point, they will continue chasing the player until they are no longer in range. The agent will not immediately stop chasing once the player is out of sight, instead they will continue searching in areas they were last seen in. Once some time has past after a thorough investigation, they will return to their patrol route around the house (day to day activities).

Analysis, Requirements Specification and Development methodology

MosCow Analysis for Feline Frenzy

Mechanics and Features:

<table>
<tr><th colspan="2">Requirements (MoSCoW Analysis)</th></tr>
<tr><td>Must Have:
First person character controller
First person camera
Basic Cat AI
Basic Enemy AI - patrol route
Player base
Lay bait
Put down/ throw
Basic Models for player, cats, and enemy
Main Menu
Basic UI
Win Condition
Reset Player at house when caught by enemy
Score System
Timer</td><td>Should Have:
Procedurally generated cats - different materials, spawn points, preference for food and toys
Cat name generator
Stamina wheel
Complex Enemy AI - cone vision/line of sight, hearing sound
Complex Cat AI - wander, flee, seek
Each instance of getting caught adds difficulty
Wanted posters after getting caught
More enemies spawn after getting caught
Higher quality models for player, cats, and enemy with animations
Game over state
Variety of food and toys to lure in cats
Limited number of foods
Breakability of cat toys
Food-foraging
More than one type of enemy (initial enemy is owner, second is police)
Suspicion bar - where the enemy will investigate the area if the player has made noise or has broken objects in the area
Mini map
Multiplayer modes</td></tr>
<tr><td>Could Have:
Procedurally generated maps
Dashing mechanic</td><td>Won’t Have:
Leaderboards
Difficulty settings</td></tr>
</table>

• Pedigree breeds worth higher points • UI featuring which cats prefer which food and toys • Achievements • Mobile support with controls • Interactable objects • Player customization • Caring for cat's minigame (feed, brush, play)	• Micro-transactions

Potential Difficulty:

In more difficult levels, some cats will squirm and leap out of the players arms if they are carrying them for too long in true cat-like fashion. This will make it harder for the players to catch the cats. Cats which have escaped this way will not be interested in any toys or food for a set amount of time.

Customisation:

The player will be able to customise their character before starting the game (and in multiplayer lobbies). In both modes the player will only be able to see their hands and legs when moving around due to the view being set in first person. They can however see their reflection through some surfaces in the level.

Caring for the Cat's:

There will be an option for the player to care for the cats by entering their own home. Once they are inside, they can play with the cats they have retrieved, brush them, pet them, and feed them. After each level is cleared the player earns some coins which can be used to purchase more cat toys and beds from the shop. The coins earned give them player an incentive to progress through the levels.

Prioritisaton of requirements

MoSCoW	Features	Prioritisation
Must	First person character controller	1
	First person camera	1
	Basic Cat AI	1
	Basic Enemy AI - patrol route	2
	Player base	3
	Lay bait	2
	Put down/ throw	2
	Basic Models for player, cats, and enemy	3
	Main Menu	3
	Basic UI	3
	Win Condition	2
	Reset Player at house when caught by enemy	3
	Score System	3
	Timer	1

Should	Procedurally generated cats - different materials, spawn points, preference for food and toys	2
	Cat name generator	4
	Stamina wheel	4
	Complex Enemy AI - cone vision/line of sight, hearing sound	1
	Complex Cat AI - wander, flee, seek	1
	Each instance of getting caught adds difficulty	3
	Wanted posters after getting caught	3
	More enemies spawn after getting caught	3
	Higher quality models for player, cats, and enemy with animations	1
	Game over state	1
	Variety of food and toys to lure in cats	2
	Limited number of foods	1
	Breakability of cat toys	3
	Food-foraging	2
	More than one type of enemy (initial enemy is owner, second is police)	4
	Suspicion bar - where the enemy will investigate the area if the player has made noise or has broken objects in the area	3
	Mini map	2
	Multiplayer modes	2
Could	Procedurally generated maps	5
	Dashing mechanic	5
	Pedigree breeds worth higher points	5
	UI featuring which cats prefer which food and toys	4
	Achievements	5
	Mobile support with controls	4
	Interactable objects	4
	Player customization	2
	Caring for cat's minigame (feed, brush, play)	2

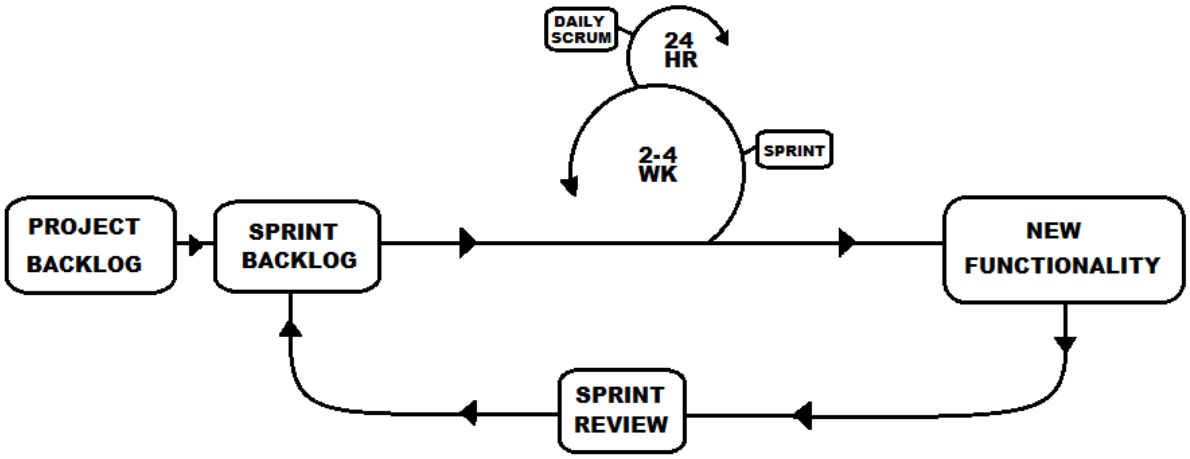
Development methodology

As discussed in the project proposal, Feline Frenzy was produced using the AGILE methodology throughout the duration of the project. The project has been managed through weekly and bi-weekly sprints alongside a continually updated project backlog.

The project backlog aided with planning out and implementing features for the current sprint. Primary mechanics were implemented during the first few sprints as they are a main requirement for the game. The agile method worked well together with the Gantt chart based off on how many sprints there could be within the set timeframe as well as the workload and intensity for each of these tasks.

The project contained some features which were dependent on another and so some had to be implemented before newer mechanics could be worked on. With the priority and intensity of each

task described in the backlog and in the Gantt chart (*see appendix*) agile aided greatly in removing many of the uncertain features before the final deadline.



[*Figure 5: Keith, C. (2011)*]

Game Design

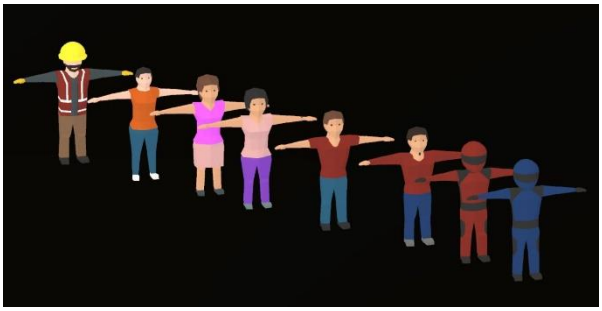
Conceptual design

Feline Frenzy is a 3D arcade style game which will be viewed in first person in single player mode and third person in multiplayer mode. The aesthetic of the game will be low-poly and cartoonlike. A mixture of assets were sued form the Unity Asset Store and produced by myself using Blender.

The assets sourced form the Unity Asset Store have a low-poly and simplistic art style.

[*Figure 6: Studio Billion (2022)*]

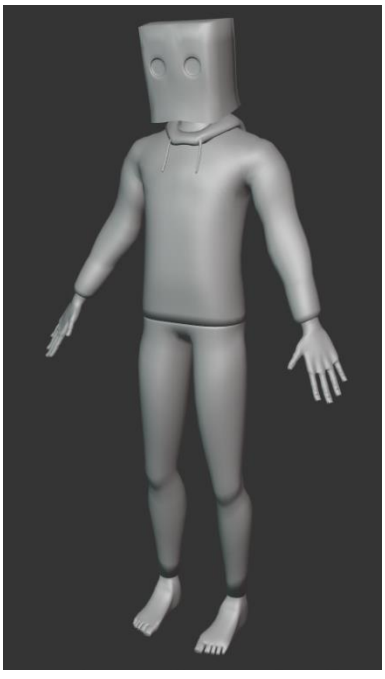
The assets which could be used for the level design. The simplistic feel and isometric aspect will fit with the desired art -style



[*Figure 7: Have Fun With Developing, (2021)*]

This pack comes with a variety of people as well as some assets for buildings. The people have been used for the owner.

Player Model: [Figure 8]



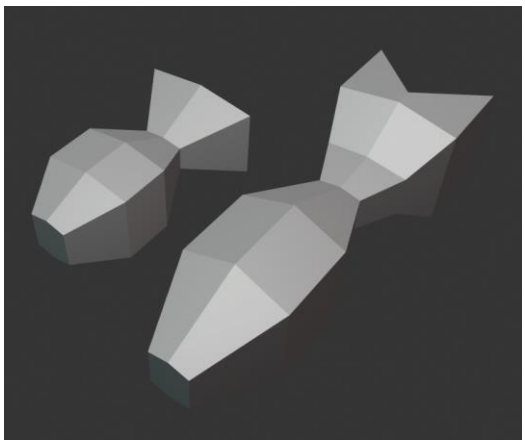
The player model was sculpted in Blender using box modelling techniques and later retopologising. The animations were added via Mixamo.

The player hides their face using different masks to prevent them from being recognised by their neighbours. The design is to come across as entertaining and humourous. The player does not have any shoes on to indicate the rush they were in when they left the house to retrieve their cats.

Cats: [Figure 9]

The cats were modelled in blender using box-modelling techniques and later rigged, weight painted and textured. The cat has different textures to represent a range of cats: black, ginger, tabby, grey, and calico. The cats are to have realistic proportions and a cute feel to them.

Bait:



[Figure 10]

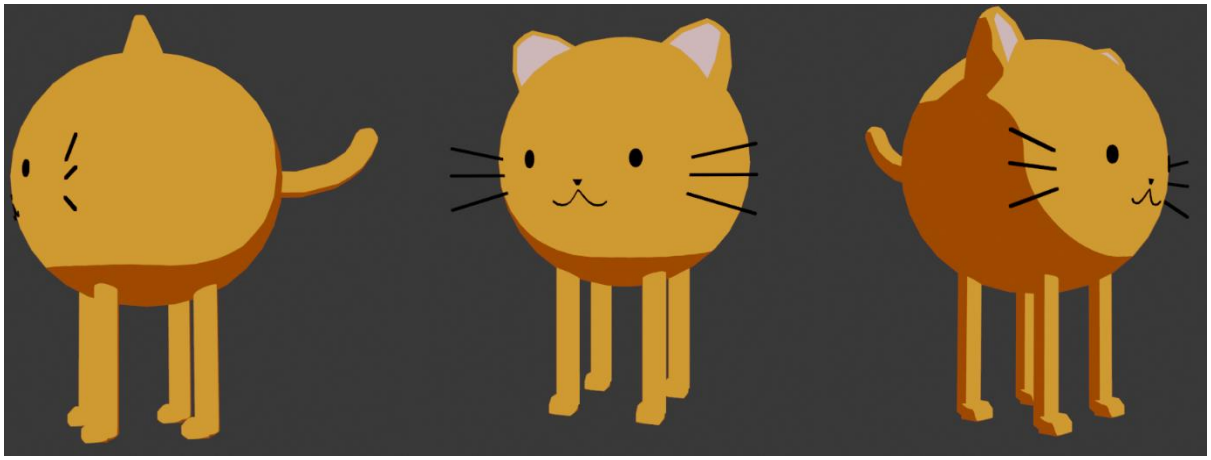
The bait model is very simple and there are two types: one regular fish and one premium branded fish.

Multiplayer Assets:

[Figure 11]

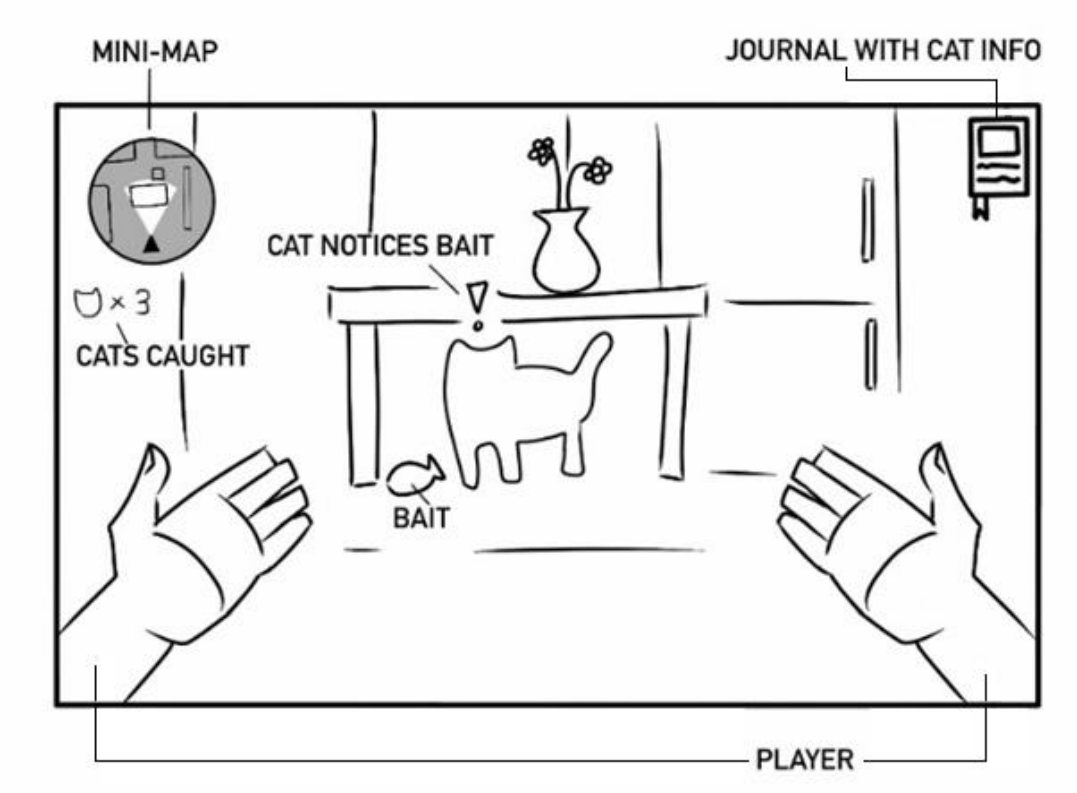
The multiplayer version for the player was made to be round and cute – to represent a miniature version of the owner.

Cat Model Multiplayer: [Figure 12]

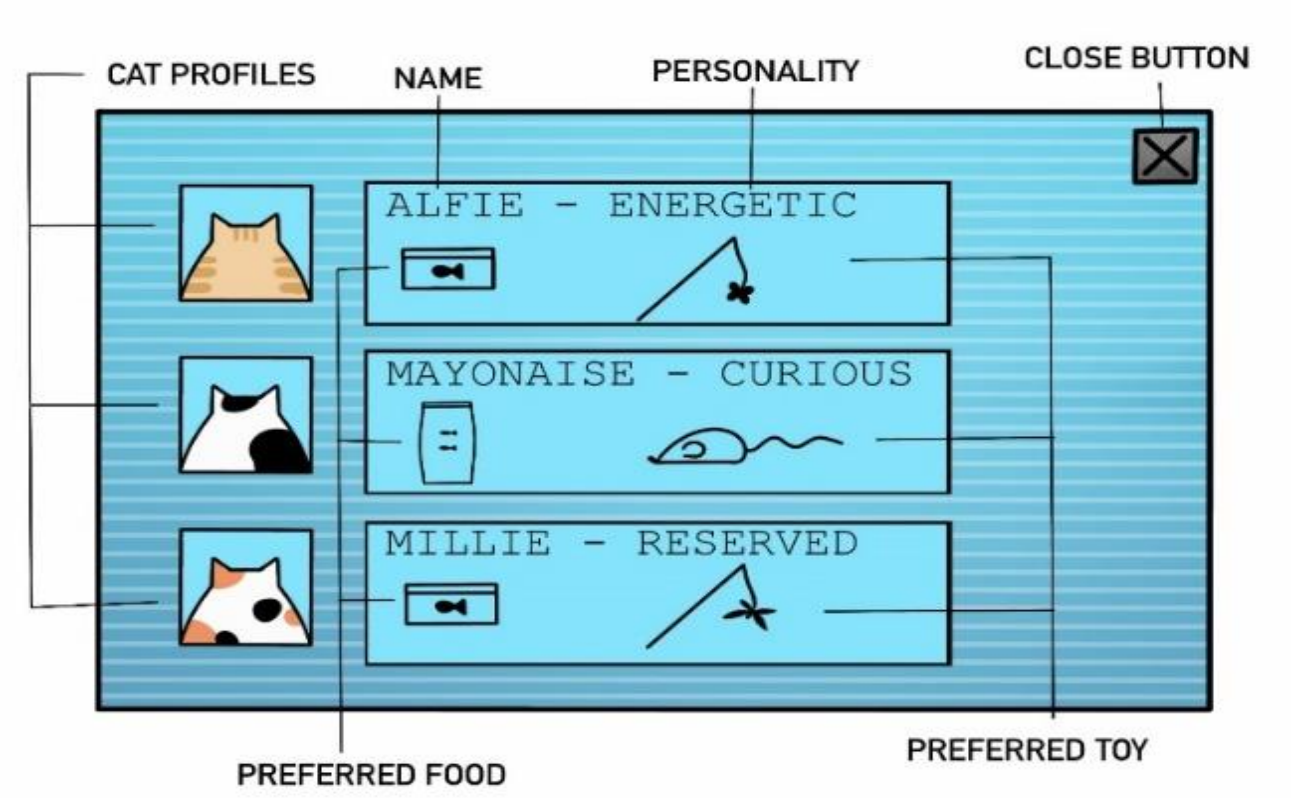


The cat model for multiplayer is to match with the smaller and simplified player. It has exaggerated proportions and a cute look to it to match the style of minigames often seen in pet games.

Technical design

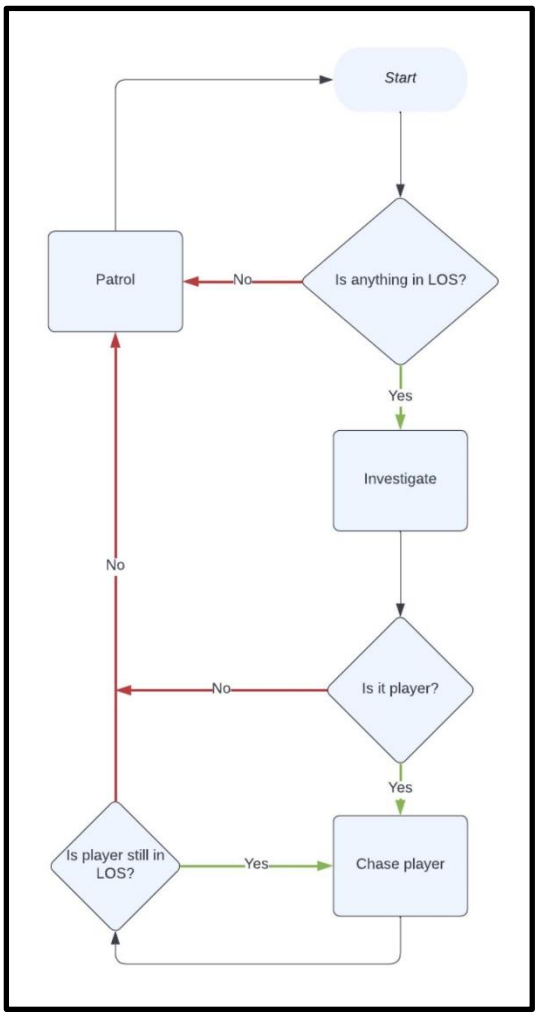


[*Figure 13: Single player game mode interface*]

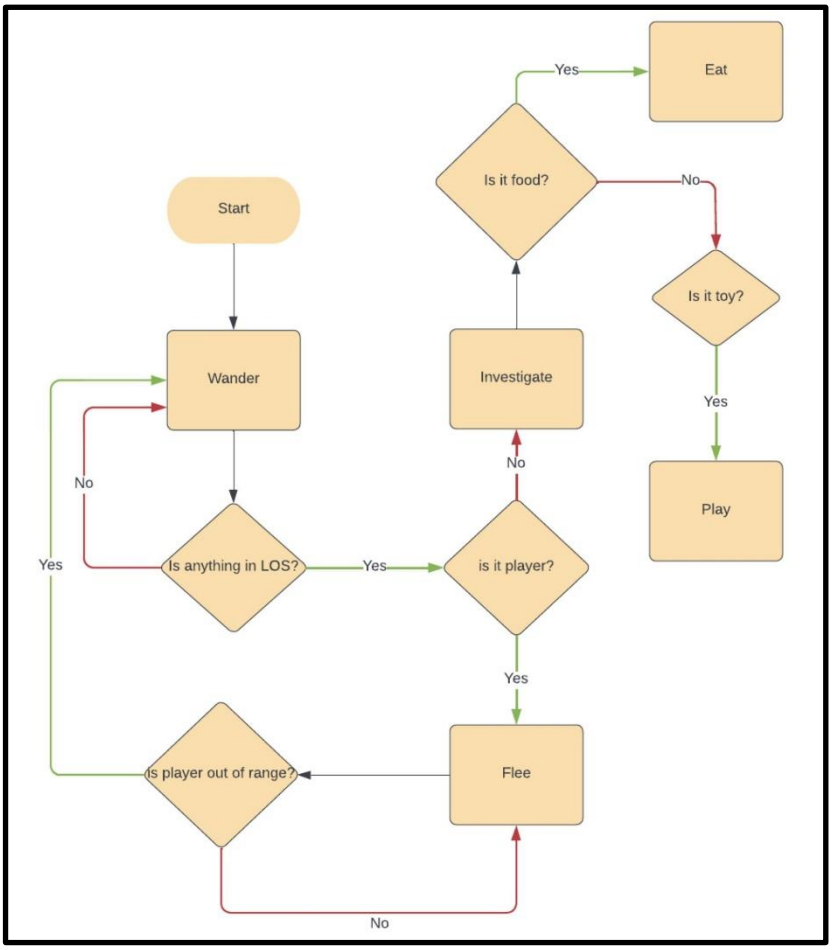


[*Figure 14: Planned interface containing details about each cat in current level*]

Flow chart of AI agents



[*Figure 15: Enemy flowchart*]



[*Figure 16: Cat flowchart*]

Implementation

Single-Player mode:

Player Movement [PlayerMovement]

```
private CharacterController characterController;
    private float moveSpeed = 5f;

    private void Awake()
    {
        characterController = GetComponent<CharacterController>();
    }

    private void Update()
    {
        // Get input for character movement
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");

        // Calculate movement direction
        Vector3 movementDirection = new Vector3(horizontalInput, 0f, verticalInput);
        movementDirection = transform.TransformDirection(movementDirection);
        movementDirection.Normalize();

        // Apply movement to the character controller
        characterController.SimpleMove(movementDirection * moveSpeed);
    }
```

The player uses a character controller rather than a rigidbody to prevent animations from causing issues. The player is moved depending on the direction they are currently facing.

Camera Script:

```
[SerializeField] private Transform followPos;
    [SerializeField] private GameObject PlayerBody;
    [SerializeField] private float mouseSensitivity;

    private float currentRotationX = 0f, currentRotationY = 0f;

    private void Update()
    {
        transform.position = followPos.position;

        // Get input for camera rotation
        float mouseX = Input.GetAxis("Mouse X");
        float mouseY = Input.GetAxis("Mouse Y");

        // Rotate the character horizontally
        PlayerBody.transform.Rotate(Vector3.up * mouseX * mouseSensitivity);

        // Rotate the camera vertically
        //transform.Rotate(Vector3.left * mouseY * mouseSensitivity);

        currentRotationX -= mouseY * mouseSensitivity;
        currentRotationX = Mathf.Clamp(currentRotationX, -90f, 90f);

        // Rotate the camera horizontally
        currentRotationY += mouseX * mouseSensitivity;
        currentRotationY = Mathf.Repeat(currentRotationY, 360f);

        transform.localRotation = Quaternion.Euler(currentRotationX, currentRotationY, 0f);
```

Cat Behaviour [Cat Behaviour]

The function *SetNewRandomTarget()* is used within the wander AI behaviour to select a random point within a sphere and set is as its target position, in order to prevent the cat from moving on the y axis, it is set to be the initial position.
SteerDirection is a private vector3 variable which takes in the current direction the cat should travel in – its value is adjusted depending on the current action of the cat. If the cat is currently wandering it will

```
private bool HasReachedTarget()
{
    return Vector3.Distance(transform.position, targetPosition) <= 1f;
}

private void SetNewRandomTarget()
{
    targetPosition = Random.insideUnitSphere * wanderRadius;
    targetPosition.y = transform.position.y;
}

private void RotateTowardsDirection()
{
    Quaternion targetRotation = Quaternion.LookRotation(steerDirection);
    transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime * rotSpeed);
}

private void MoveForward()
{
    transform.Translate(Vector3.forward * moveSpeed * Time.deltaTime);
}
```

The above functions are repeatedly called throughout the script to manage behaviour and AI. HasReachedTarget returns true if the cat is within range of the target position. The target itself changes depending on what the cat's current action is. If the cat is seeking then the target position is set to nearby bait, if the cat senses the player than the target is the opposite vector direction to them and if the cat is wandering than the target is set to be inside of a UnitSphere.

```
private void Wander()
    {
        steerDirection = Vector3.Lerp(steerDirection, (targetPosition - transform.position).normalized, Time.deltaTime * rotSpeed);

        RotateTowardsDirection();
        MoveForward();

        if (HasReachedTarget())
        {
            SetNewRandomTarget();
        }
    }
```

In order to prevent the cat from wandering in circles, a larger radius is used – however it still eventually walks in a circular sort of motion as it is continually retrieving a point within the sphere. A coroutine is used to occasionally pause the cats current movement within the wander state to ensure it selects a different

```
private void Seek()
{
    if(spawnedBaits.Count==0)
    {
        isWander = true;
        return;
    }

    int nearestPreferredBaitIndex = -1;
    float nearestPreferredBaitDistance = float.MaxValue;

    for (int i = 0; i < spawnedBaits.Count; i++)
    {
        GameObject bait = spawnedBaits[i];
        if (bait != null && i == prefbait)
        {
            float distanceToBait = Vector3.Distance(transform.position, bait.transform.position);

            if (distanceToBait < nearestPreferredBaitDistance)
            {
                nearestPreferredBaitIndex = i;
                nearestPreferredBaitDistance = distanceToBait;
            }
        }
    }

    if (nearestPreferredBaitIndex != -1)
    {
        GameObject nearestPreferredBait = spawnedBaits[nearestPreferredBaitIndex];
        targetPosition = nearestPreferredBait.transform.position;
        targetPosition.y = transform.position.y;

        steerDirection = Vector3.Lerp(steerDirection, (targetPosition - transform.position).normalized, Time.deltaTime * rotSpeed);

        RotateTowardsDirection();
        MoveForward();
    }
```

The seek function takes in bait spawned in the scene. As the bait is *instantiated after* runtime, the cat needs a way to check if it exists first – which is why the bait class contains a list to append spawned bait.

Another check is run within the seek function if the bait is its preferred one, if it is not then the cat will resume wandering. Once all the calculations of the target posision is completed, the MoveForward() and Rotate() functions are called to move the cat and rotate it towards the formulated target.

```
private void Flee()
    {
        targetPosition = player.transform.position;
        targetPosition.y = transform.position.y;
        steerDirection = Vector3.Lerp(steerDirection, (transform.position - targetPosition).normalized, Time.deltaTime * rotSpeed);

            RotateTowardsDirection();
            MoveForward();
    }
```

The flee function takes the players current position and subtracts it from the cats current position in order to move the cat away from the player rather than towards it.

Bait throwing and syncing [LayBait and Bait]

The Bait script sets a name for the bait and checks if a cat is currently touching it, if it is then the bait will be destroyed within the scene.

The LayBait script instantiates a prefab of the bait (dependant on what bait is currently selected) and projects it along a trajectory of the current forward vector of the camera. After the bait is instantiated it is added to a list of bait for the cat to be able to detect. Inside the script for the cat it retrieves the list of bait and filters out for the bait it prefers.

```
private void Update()
{

    if (Input.GetKeyDown(KeyCode.R))
    {
        ChangeBait();
    }

    if (currentbaitnum <= 0) { isthrowing = false; }
    if (Input.GetMouseButtonDown(0) && baitspawn.childCount == 0)
    {
        isthrowing = true;
    }

    if (isthrowing)
    {
        throwforce += Time.deltaTime * throwspeed;
        throwforce = Mathf.Clamp(throwforce, 0, maxThrowForce);

    }

    if (Input.GetMouseButtonUp(0) && isthrowing == true)
    {
        currentbaitnum -= 1;
        UpdateBait();
        isthrowing = false;

        GameObject bait = Instantiate(baitPrefab[currentbaitselected], baitspawn.position, Quaternion.identity);
        baitList.Add(bait);
        cat.GetComponent<CatBehaviour>().AddBait(bait);
        Rigidbody rb = bait.GetComponent<Rigidbody>();
        rb.velocity = cam.transform.forward * throwforce;
        throwforce = 0;
    }
}
```

The script checks for if the player has switched the current selected bait or not, depending on what is selected the object spawned will change.

Character customisation [CharacterCustomisation and DisplayCustomisation]

The character customisation script is relatively simple, it takes in canvas UI elements such as buttons to cycle through the players customisable options. A player prefs int is used for each customisable option and saves it once the player clicks the confirm button. It is important to note that the player's model does not delete the asset but hides it instead – this is to prevent issues at runtime with removing and deleting an object within a prefab, it also allows the player to be change their character at a later time in the game.

```
[SerializeField] private GameObject[] hats;
[SerializeField] private GameObject[] tops;
[SerializeField] private GameObject[] bottoms;
[SerializeField] private Material[] SkinMat;
[SerializeField] GameObject playerbody;

[SerializeField] Button Rhat, Lhat, Rtop, Ltop, Rbottom, Lbottom, Lskin, Rskin;
public float rotSpeed;
int currenthat = 0, currenttop = 0, currentbottom = 0, currentskin = 0;
public int hatID, topID, bottomID, skinID;
```

Prior to making the script, the structure of the player model is set up with all assets being unhidden in the hierarchy, the model has a base body which is rigged using mixamo and relevant accessories are then parented to the closest specified bone. This allows the player to have fully functioning animations with customisation. This was done in blender and Mixamo and then the model was imported as an fbx into Unity.

Each individual item which the player can swap out to is a separate game object – which has been parented to the closest bone within the armature.

The skin tone texture is a material applied to the character. The player has a separate material shared for each article of clothing, separate head accessories and skin. The skin material was separated and re-coloured for each skin tone. For each accessory the player cycles through an array of objects which are hidden and unhidden, whereas for skin tone the array changes the *material* of the players body.

```
void Start()
{
    SetAssetInactive();
    hats[0].SetActive(true);
    tops[0].SetActive(true);
    bottoms[0].SetActive(true);
    playerbody.GetComponent<SkinnedMeshRenderer>().material = SkinMat[0];

}

private void RotatePlayer()
{
    //float rotSpeed = 5;
    playerbody.transform.Rotate(0, Input.GetAxis("Horizontal") * rotSpeed, 0);// * Time.deltaTime, 0);
}
private void Update()
{
    RotatePlayer();
}
void SetAssetInactive()
{
    for (int i = 0; i < hats.Length; i++)
    {
        hats[i].SetActive(false);
    }

    for (int j = 0; j < tops.Length; j++)
    {
        tops[j].SetActive(false);
    }

    for (int k = 0; k < bottoms.Length; k++)
    {
        bottoms[k].SetActive(false);
    }
}
```

At the start of the game each of the customisable assets are set to be inactive, and then the first accessory to be visible and active (to prevent no items from being shown). The player is able to rotate their character around to see the changes made. The skin texture is set to the initial material in the array (it cannot be set to active).

```
//Head Accessory
    public void PreviousHat()
    {
        currenthat--;
        if (currenthat < 0) { currenthat += hats.Length; }
        CycleCustomisationHats();
    }
    public void NextHat()
    {
        currenthat++;
        if (currenthat > hats.Length - 1) { currenthat = 0; }
        CycleCustomisationHats();
    }
    void CycleCustomisationHats()
    {
        for (int i = 0; i < hats.Length; i++)
        {
            hats[i].SetActive(false);
            if (hats[currenthat] == hats[i])
            {
                hats[currenthat].SetActive(true);
                hatID = currenthat;
            }
        }
    }
```

The functions for the head, tops and bottoms follow the same structure with deactivating the current item in the array and incrementing the currnethat (trouser or bottom) and then calling a function to cycle through the list for each button press and display it on the screen for the player to see.

```
//Skin
    public void PreviousSkin()
    {
        currentskin --;
        if (currentskin < 0) { currentskin += SkinMat.Length; }
        CycleCustomisationSkin();
    }
    public void NextSkin()
    {
        currentskin ++;
        if (currentskin > SkinMat.Length - 1) { currentskin = 0; }
        CycleCustomisationSkin();
    }
    void CycleCustomisationSkin()
    {
        for (int i = 0; i < SkinMat.Length; i++)
        {
            if (SkinMat[currentskin ] == SkinMat[i])
            {
                playerbody.GetComponent<SkinnedMeshRenderer>().material = SkinMat[currentskin];
                skinID= currentskin ;
            }
        }
    }
```

For the skin change, it is similar except that the current material is changed to the next value.

These changes are then saved into the player prefs into their respective integers, and the scene is then loaded.

```
public void ConfirmChanges()
{
    PlayerPrefs.SetInt("Hat", hatID);
    PlayerPrefs.SetInt("Top", topID);
    PlayerPrefs.SetInt("Bottom", bottomID);
    PlayerPrefs.SetInt("Mat", skinID);

    //LoadScene:
    SceneManager.LoadScene("Level1");
}
```

The customisation script is attached to the canvas object in the customisation scene for the single player, while the display script is *directly* attached to any instance of the player in the level.

```
[SerializeField] GameObject[] hats, tops, bottoms;
[SerializeField] Material[] skin;
[SerializeField] GameObject playerbody;

    void Start()
    {
        SetAssetInactive();
        GetCustom();
    }


    void SetAssetInactive()
    {
        for (int i = 0; i < hats.Length; i++)
        {
            hats[i].SetActive(false);
        }

        for (int j = 0; j < tops.Length; j++)
        {
            tops[j].SetActive(false);
        }

        for (int k = 0; k < bottoms.Length; k++)
        {
            bottoms[k].SetActive(false);
        }

        playerbody.GetComponent<SkinnedMeshRenderer>().material = skin[0];
    }

    void GetCustom()
    {
        hats[PlayerPrefs.GetInt("Hat")].SetActive(true);
        tops[PlayerPrefs.GetInt("Top")].SetActive(true);
        bottoms[PlayerPrefs.GetInt("Bottom")].SetActive(true);
        playerbody.GetComponent<SkinnedMeshRenderer>().material = skin[PlayerPrefs.GetInt("Mat")];
    }
```

The order of the items in the array *must* be kept the same as how it is setup within the customisation scene. If not then the player will have different items set as to what they chose.

Timer

```
public float timeleft;
    public TextMeshProUGUI timertext;

    void Update()
    {
        if (timeleft > 0) { timeleft -= Time.deltaTime; }
        else { timeleft = 0; }

        DisplayTime(timeleft);

    }
    void DisplayTime(float timeToDsiplay)
    {
        if (timeToDsiplay < 0) { timeToDsiplay = 0; }

        float minutes = Mathf.FloorToInt(timeToDsiplay / 60);
        float seconds = Mathf.FloorToInt(timeToDsiplay % 60);
        timertext.text = string.Format("{0:00}:{1:00}", minutes, seconds);
    }
```

The timer script counts down from the inputted value (can be adjusted in the inspector per level). The remaining time is retrieved from this script into the gamewon and gamelose conditions in the game manager.

Enemy AI:

```
//Patrol
public Transform[] patrolPoints;
private int currentPoint;
private float dist_point;
public string task;

void Start()
{
    task = "patrol";
    isenter = false;
    transform.position = patrolPoints[0].position;
    currentPoint = 0;
    transform.LookAt(patrolPoints[currentPoint].position);
}

void NextPatrol()
{
    currentPoint++;
    if (currentPoint >= patrolPoints.Length) { currentPoint = 0; }
    transform.LookAt(patrolPoints[currentPoint].position);
}
void Patrol()
{
    transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(patrolPoints[currentPoint].position - transform.position), rotspeed * Time.deltaTime);
    transform.Translate(Vector3.forward * speed * Time.deltaTime);
}
void Update()
{
    dist_point = Vector3.Distance(transform.position, patrolPoints[currentPoint].position);
    if(dist_point < 1f) { NextPatrol(); }
    if(task == "patrol"){Patrol();}
    if(task == "chase"){ChasePlayer();}
    if (isenter) { time_enter += Time.deltaTime; }// ChasePlayer(); }
}
```

The enemy AI agent has different stated – as opposed to the cat which has AI wandering algorithm, the enemy patrols across a set number of points in the level instead by using an array. When the enemy reaches the point it targeted it will increment the index and move to the next slot.

```
void OnTriggerEnter(Collider other)
    {
        if(other.tag == "Player"){isenter = true;}
    }
    private void OnTriggerStay(Collider other)
    {
        if(other.tag == "Player")
        {
            if (time_enter >= 0.02f) { ChasePlayer(); }
        }
    }
    void OnTriggerExit(Collider other)
    {
        if(other.tag == "Player")
        {
            isenter = false;
            task = "patrol";
            time_enter = 0;
        }
    }

    void ChasePlayer()
    {
        task = "chase";
        transform.rotation = Quaternion.Slerp(transform.rotation, Quaternion.LookRotation(player.transform.position -  transform.position), rotspeed * Time.deltaTime);
        transform.position += transform.forward * speed * Time.deltaTime;
    }
```

The enemy will know If the player is within range by using a trigger as opposed to checking the distance or using a raycast. A Boolean value keeps track of the detection of the player, if the player is within sight then the enemy will leave patrol and begin chasing the player by claling ChasePlayer() and setting the current task to “Chase”.

Multiplayer Modes:

Prop-hunt:

While this mode has not been completed and has since been discarded the functionality of the mechanic has been implemented in a script and worked with prototype 2. (due to incompatibility issues with Photon).

The mechanic for switching between two different models in further updates the script would take different prefabs for each map for the player to use.

```
public class PlayerPropHunt: NetworkBehaviour
{

    //Prophunt
    [SerializeField] private GameObject[] props;
    [SerializeField] private GameObject playerbody;

    private int currentProp = -1;

    void Start()
    {
        HideProps();
    }

    public override void OnNetworkSpawn()
    {
        if (IsOwner) { LocalInstance = this; }
        transform.position = spawnPosList[(int)OwnerClientId];
    }

    private void Update()
    {
        if (!IsOwner)
        {
            return;
        }
        Movement();

        if (Input.GetKeyDown(KeyCode.Q))
        {
            if (NetworkManager.Singleton.IsServer)
            {
                PropDisplayClientRpc();
            }
            else
            {
                PropDisplayServerRpc();
            }
        }
```

Similar to the customisation script, the prop hunt script takes into account an array of props (which are set under the player body itself) and activated or deactivated via keypress. An integer keeps track of the current prop shown and cycles through the list until the index goes beyond the length of the prop array. A check is run if the client is the host (isserver) and sends an RPC.

```
[ServerRpc(RequireOwnership = false)]
void PropDisplayServerRpc(ServerRpcParams rpcParams = default)
{
    PropDisplayClientRpc();
}

[ClientRpc]
void PropDisplayClientRpc(ClientRpcParams rpcParams = default)
{
    PropDisplay();
}

void PropDisplay()
{
    if (currentProp == -1) { currentProp = 0; playerbody.SetActive(false); props[currentProp].SetActive(true); }
    else
    {
        props[currentProp].SetActive(false);
        currentProp++;
        if (currentProp < props.Length)
        {
            props[currentProp].SetActive(true);
        }
        else { RevertBackToPlayer(); }
    }

}
private void HideProps()
{
    for (int i = 0; i < props.Length; i++)
    {
        props[i].SetActive(false);
    }
}

private void RevertBackToPlayer()
{
    HideProps();
    currentProp = -1;
    playerbody.SetActive(true);
}
```

The server RPC calls a client RPC which in turn calls a function – this method is not ideal but it had worked as opposed to having a direct call for the host to sync all clients. Not implementing it this way had caused various errors and so two RPC function called was the solution.

The index is set to -1 to prevent the first prop (at index 0) from *not* being seen. When reinitialised to 0 after the maximum props were iterated though, it would skip the first prop at the next key press.

If this script was to be improved upon further, there would need to be a way to have different props, for the prototype there were 3 props per level, but having all props on the player prefab would have caused issues with slow fps as the vertex count of the model increases drastically the more props are currently on the player. Instantiating and destroying the props at runtime caused issues and so another option would have either been to have *separate* player models per selected map (this would have resulted in the players needing to quit the game to choose another map).

Character Customisation [CharacterCustomisationMP and DisplayCustomisationMP]

The character customsiation script for singleplayer mode has been duplicated and adjusted to the new model for the multiplayer levels.

For the multiplayer mode it was important to have the players customise their characters *before* they load into the game. This is to prevent syncing issues where the players would see otherplayers have the same outfit and character they have rather than the individual has chosen.

Inside DisplayCustomisationMP, the player sync's with the first player via an RPC function call.

```
    private PhotonView PV;
    public Material[] skin;
    public GameObject[] hats;
    public GameObject[] tops;
    public GameObject[] bottoms;
    public GameObject body;

    void Start()
    {
        PV = GetComponent<PhotonView>();
        if (PV.IsMine)
        {

            PV.RPC("RPC_ChangeSkin", RpcTarget.AllBuffered, PlayerPrefs.GetInt("Mat"));
            PV.RPC("RPC_ChangeHat", RpcTarget.AllBuffered, PlayerPrefs.GetInt("Hat"));
            PV.RPC("RPC_ChangeTop", RpcTarget.AllBuffered, PlayerPrefs.GetInt("Top"));
            PV.RPC("RPC_ChangeBottom", RpcTarget.AllBuffered, PlayerPrefs.GetInt("Bottom"));
        }
    }
```

Each player has the DisplayCustomisationMP script attached to ensure that they are all synced correctly.

```
[PunRPC]
void RPC_ChangeSkin(int whichskin)
{
    body.GetComponent<SkinnedMeshRenderer>().material = skin[whichskin];
}

[PunRPC]
void RPC_ChangeHat(int whichhat)
{
    for (int j = 0; j < hats.Length; j++)
    {
        hats[j].SetActive(false);
    }
    hats[whichhat].SetActive(true);
}

[PunRPC]
void RPC_ChangeTop(int whichtop)
{
    for (int i = 0; i < tops.Length; i++)
    {
        tops[i].SetActive(false);
    }
    tops[whichtop].SetActive(true);
}

[PunRPC]
void RPC_ChangeBottom(int whichbottom)
{
    for (int i = 0; i < bottoms.Length; i++)
    {
        bottoms[i].SetActive(false);
    }
    bottoms[whichbottom].SetActive(true);
}
```

Testing and Evaluation

Following the Agile development methodology, the game was tested throughout the duration of the project at set intervals. With minor changes to the prototype having smaller testing groups. After each mechanic was implemented, it was tested, once the mechanic worked as intended the next mechanic would be added.

Black Box testing

Test Name	Expected Output	Actual Output	Comments
Single Player Testing			
Player Movement	Player will be able to move using the arrow keys	As Expected	
Pick up Cats	The player should be able to pick up a cat by pressing on the E key. The cat should be parented to the players hand and not move. The player should not be able to pick up more than one cat at the same time.	The player cannot pick up a cat if they are wandering, sometimes the cat will continue to wander even when picked up. The player can pick up two cats.	Run a check to make sure the cat cannot wander, seek or flee by setting all Boolean values to false. Add another booelan value to check if it has been picked up. The mechanic now correctly functions unless the player repeatedly presses the key.
Drop Cats	The player should be able to put down a cat they are holding.	The player cannot drop a cat if they are carrying two.	This error occurred due to the pickup allowing for more than one cat to be picked up. It has since been fixed.
Camera Movement	Camera should be viewed withing a first-person perspective and rotate around the player's view. The p	As Expected	
Single player Customisation	The player should be able to press different buttons to cycle though the customisable options. The current material and accessory should be displayed at the time of clicking the button.	As Expected	
Single player Display Customisation	The selected accessories and materials should be	As Expected	

	displayed and seen on the player		
Cat Wander AI	The cat should wander randomly within a set area. The cat should occasionally stop and pause.	Cat wanders repeatedly in circles	A co routine runs to pause the cats movements for it to "think" about the next route to take. Implementing this shot wait time allowed the cat to not continually rotate in a circular manner.
Cat detection AI	The cat should be able to detect when the player is nearby and change states from wander to flee.	The cat will not detect the player after eating food	Another if statement clause has been added to check the Boolean value of isdetectplayer to false once it has eaten.
Cat Seek food	The cat should seek any bait laid out by the player	The cat will detect food but pause if the player is nearby	This has been fixed after adjusting the player detection.
Cat Seek preferred bait	The cat should only seek bait it prefers and ignore any other type of food.	As Expected	
Cat Flee	The cat should flee when they see the player (when the player is within their line of sight) The cat should not flee if it is currently eating or seeking food.	As Expected	
Enemy AI	Enemy should patrol along points on the map and if the player is within their line of sight then initiate a chase sequence until they are out of sight.	As Expected	
Laying Bait	The player should be able to lay bait by holding onto the mouse button. The player should be able to throw it further depending on how long the key was held on for.	As Expected	

Timer UI	The timer should countdown and decrease until it reaches 0.	As Expected	
Game Won State	A game win screen should pop up if the player has collected all cats on the level and the time is up.	As Expected	
Game Lose State	A game lose screen should pop up if the player has not collected all the cats and the time is up.	As Expected	
Multiplayer Testing			
Networking test	The player should be able to click on the multiplayer mode option to be told it is “connecting” or “loading” and setting up a connection.	As Expected	
Player Join room	The player should be able to join a lobby (*referred to as room*) which displays the current number of players	As Expected	
Player joins full room	The player should be able to enter a separate empty lobby	As Expected	
Player leaves room	The number of players in the room displayed in the UI should decrease. New players should be able to join the room.	As Expected	
Player Leaves Game	The game session should not end but continue until the conditions are met.	Players who are not the host are able to leave the game without ending the session or causing issues with the syncing of the timer. If the host leaves, the game does continue – however it does not end and players cannot win or lose as the timer will no longer sync.	Changing the host once they leave has been attempted but caused further issues with customisation.

Player Customisation	Player should be able to select customisable options and see the live change on the side.	As Expected	
Player Display Customisation and Sync	The player should be able to see their customised character as well as other players characters correctly.	Players who join later see other players customised avatars as what they themselves have set – only the host will correctly see other players as their customised option	An RPC call was used to sync the client's display with the host in order to make sure any player that joins later will correctly see the avatars via the host.
Prop-hunt – change into props	The player should be able to change into different items and hide the original player model while cycling through the list with E.	As Expected	The player is able to change between different objects and return to the original
Prop hunt – sync with other players	Other players should be able to see live changes to the player's current active mesh.	Many different issues occurred with syncing for different players: 1: Players do not see other players as different objects even if they change 2: Players will see other players as objects only when the player changes into a disguise themselves 3: Players will correctly see other players as objects when they disguise themselves however the order of props is not correct. If the player is currently a table and changes back to a player, player 2 will still see them as a table until they change back to the 1st of the array	*This was tested both with Photon Pun and Netcode. The issue was fixed using Netcode:* Two RPC functions were used in this instance. The first one calls the client RPC, which calls another function to revert back to the player body.
Player Animation	The player should have animations working and synced correctly	There is a significant time lag for when an animation plays and when other players see this happen	Time lag has not been fixed

Pickup cats	The player should be able to pick up cats and see other players with cats when they are moving	Time lag occurs Players can pick up cats	Instead of parenting the cat to the player it has a Boolean value which checks if it is "parented" or not by setting the current position to be in a players hand

Moscow Review

Many of the intended requirements have been met, there have been some which have not been attempted or completed due to time limits. Some have been discarded due to errors occurring with the methods used, while others have been implemented but adjusted (such as the procedurally generated cats).

Some of the "Could have" requirements were implemented after a majority of the "Should Have" were functioning. These were the character customisations and mobile support with controls. A smaller and simplified version of the singleplayer game has been ported to mobile. The caring for cats minigame was partially implemented with errors.

The "Should Have" requirements had multiple mechanics which could not be implemented on time or were not functional for the final prototype. One of the multiplayer modes "Prop Hunt" was also discarded after prototype 2 testing.

Nonetheless all "Must Have" requirements have successfully been implemented into the final project early in the development process.

Key:

Completed
Partially Implemented
Implemented – Not functioning
Discarded
Not Implemented

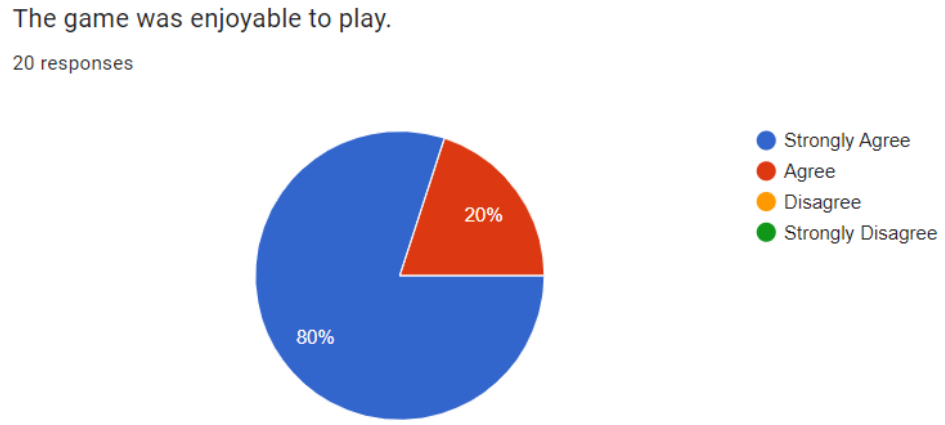
MoSCoW	**Mechanics**
Must Have	• First person character controller • First person camera • Basic Cat AI • Basic Enemy AI - patrol route • Player base • Lay bait • Put down/ throw

	• Basic Models for player, cats, and enemy • Main Menu • Basic UI • Win Condition • Reset Player at house when caught by enemy • Score System • Timer
Should Have	• Procedurally generated cats - different materials, spawn points, preference for food and toys • Cat name generator • Stamina wheel • Complex Enemy AI - cone vision/line of sight, hearing sound • Complex Cat AI - wander, flee, seek • Each instance of getting caught adds difficulty • Wanted posters after getting caught • More enemies spawn after getting caught • Higher quality models for player, cats, and enemy with animations • Game over state • Variety of food and toys to lure in cats • Limited number of foods • Breakability of cat toys • Food-foraging • More than one type of enemy (initial enemy is owner, second is police) • Suspicion bar - where the enemy will investigate the area if the player has made noise or has broken objects in the area • Mini map • Multiplayer modes
Could Have	• Procedurally generated maps • Dashing mechanic • Pedigree breeds worth higher points • UI featuring which cats prefer which food and toys • Achievements • Mobile support with controls • Interactable objects • Player customization • Caring for cat's minigame (feed, brush, play)

Won't Have	• Leaderboards • Difficulty settings • Micro-transactions

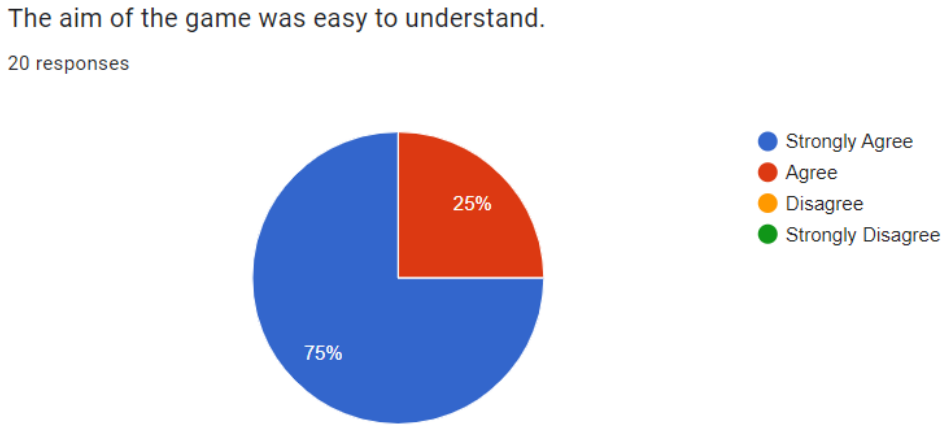
Player Feedback and prototype

A diverse group of 20 participants tested Feline Frenzy and were asked to complete a questionnaire. There were 7 questions for both single player mode and multiplayer mode.



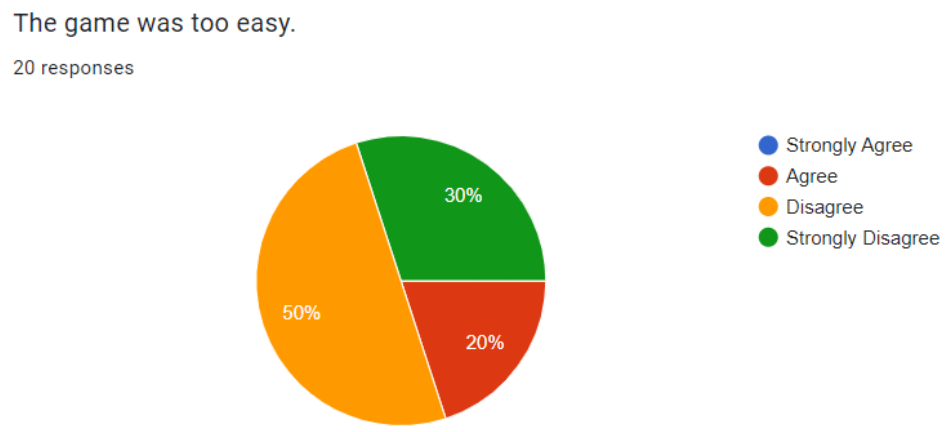
[Figure 17]

A majority of the testers agreed that the game was enjoyable to play.



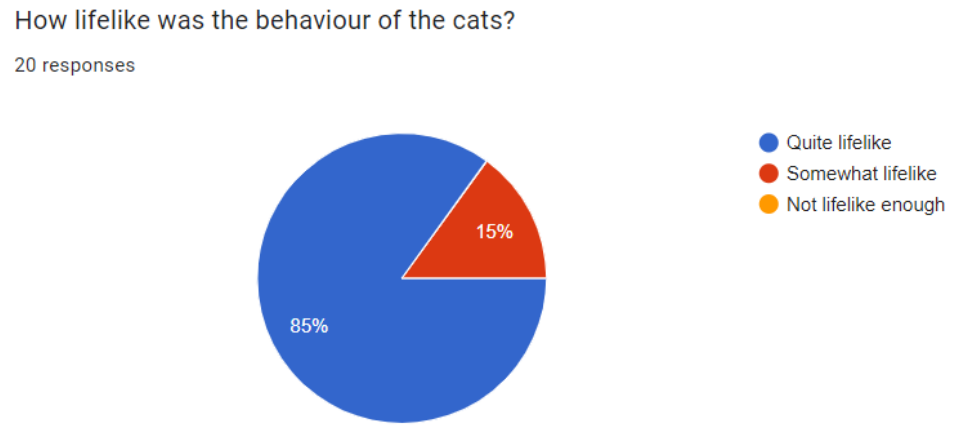
[Figure 18]

A majority of the players understood the main aim of the game.



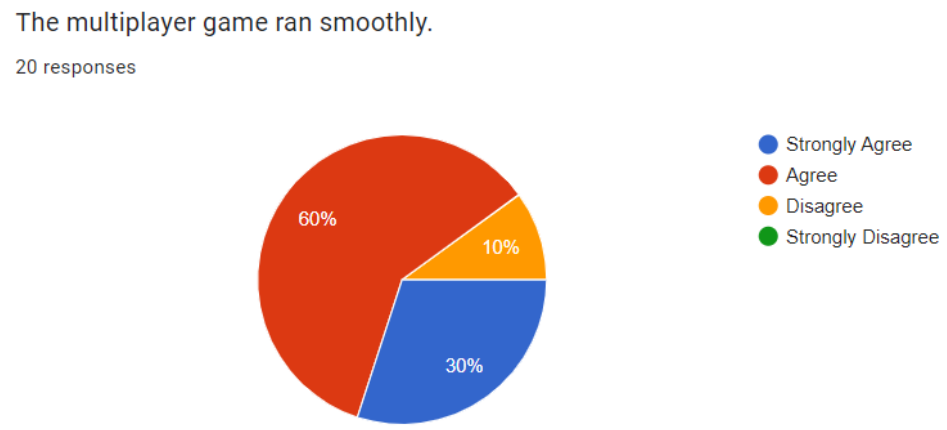
[Figure 19]

In regards to difficulty some players found it very hard, hard and easy. This may have been due to the time limit within the singleplayer level as there were only 3 minutes to find and catch all the cats.



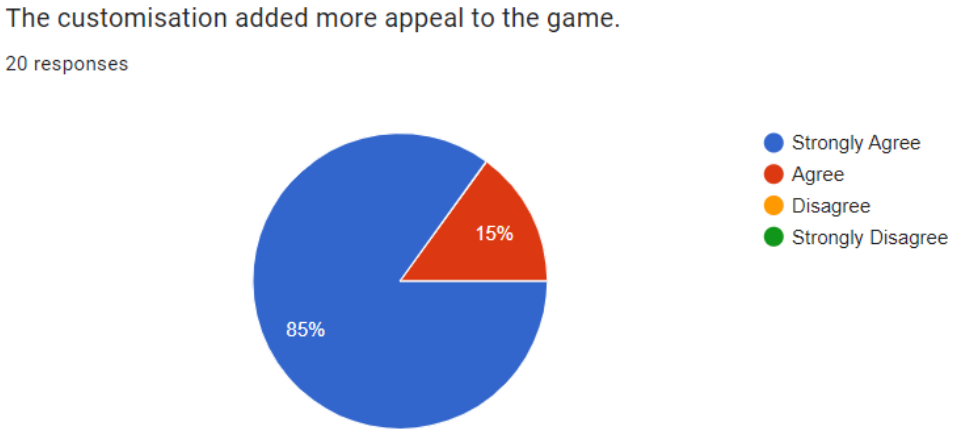
[Figure 20]

One of the primary goals of the game was to showcase realistic behaviour of the cats featured in the game, many of the players (85%) agreed that the cats were quite lifelike in behaviour.



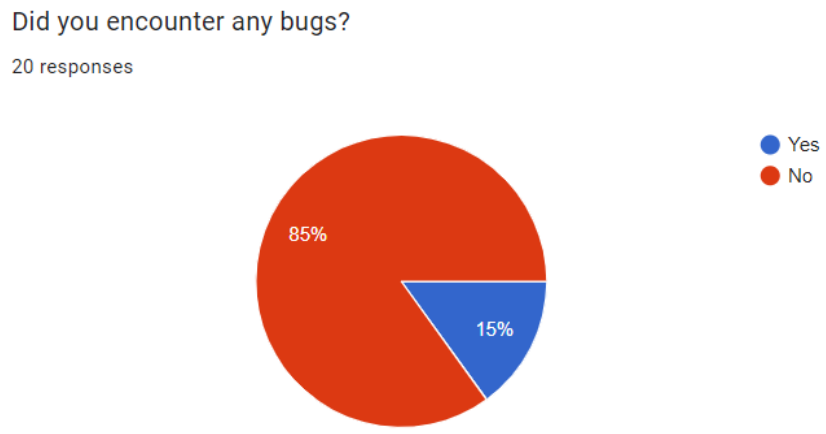
[Figure 21]

For questions regarding multiplayer modes a majority of answers had mixed opinions. There were issues with running smoothly and unexpected glitches which would cause players to exit the game.



[Figure 22]

Many players enjoyed the fact that they could customise their characters in both the single player version and the multiplayer mode.



[Figure 23]

> It glitched and wouldn't let me pick up a cat when I had none.

> The multiplayer session ended randomly

> For customisation, everyone had the same outfit in multiplayer mode

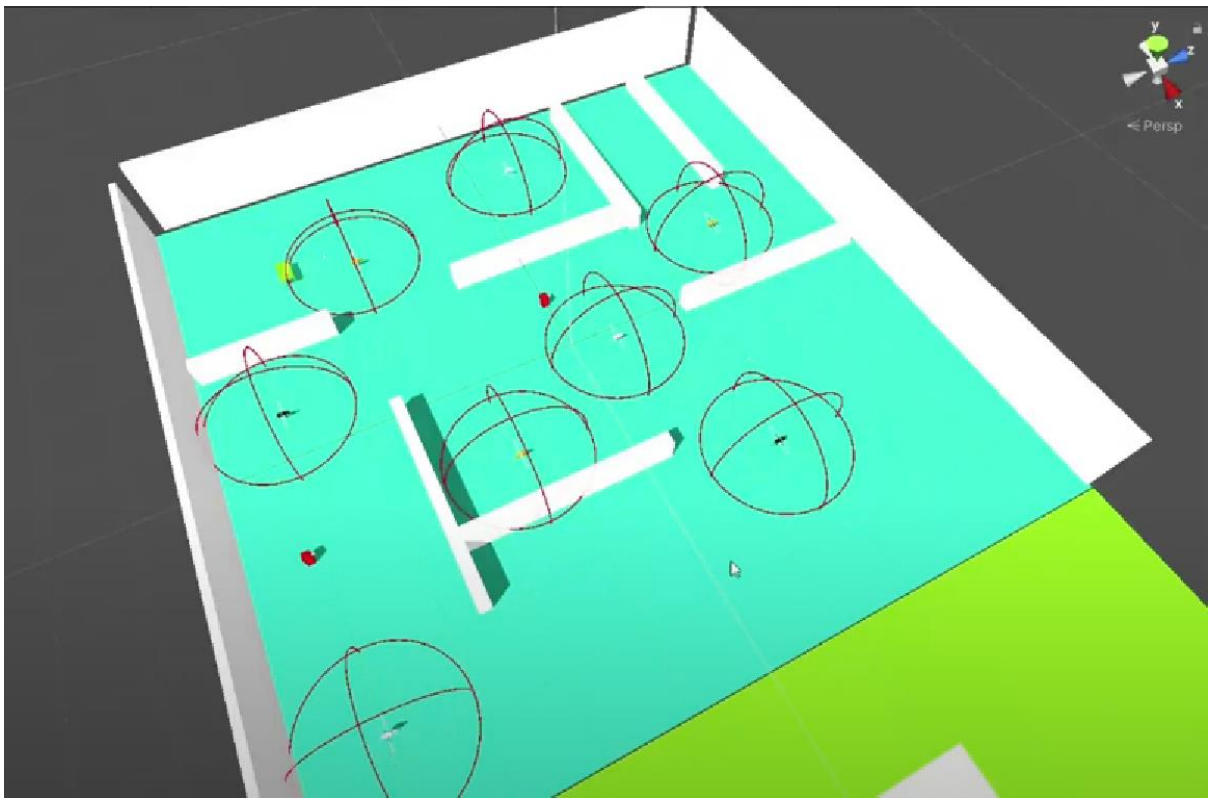
For improvements some users requested for more content, with 3 asking for a game to pet or take care of the cats. While the current petting minigame is faulty, there would be a future update to fix it and implement it.

> Maybe another multiplayer mode

Other request for improvement were to add another multiplayer mode as there is currently only one cooperative mode.

Prototype Content Overview

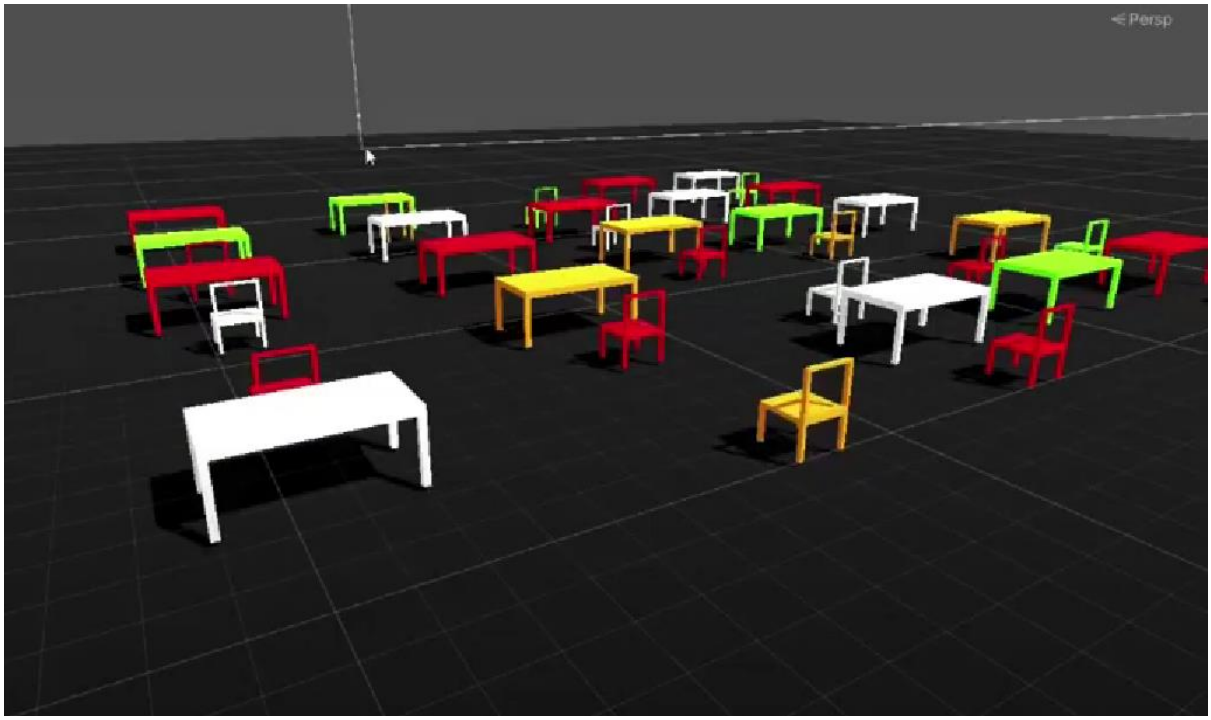
Prototype 1



[*Figure 24: Top down view of the single player level]*

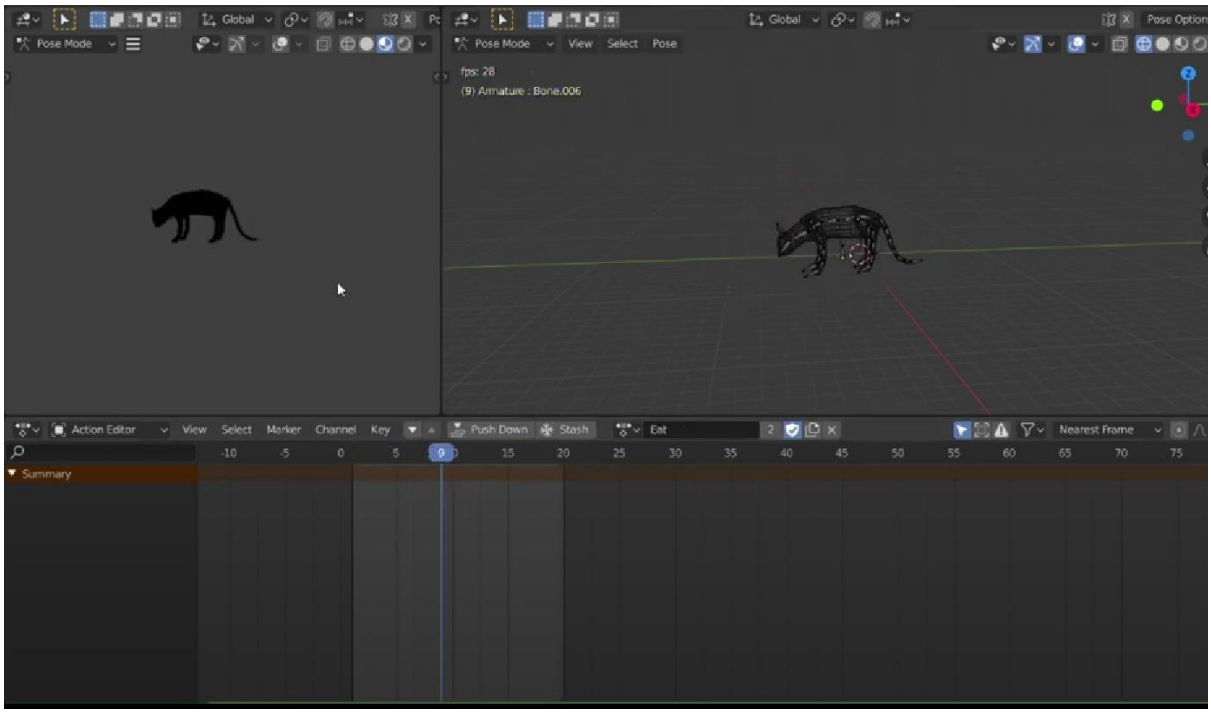
A small internal testing size of 5 were used to test out the functionality of the base mechanics of Feline Frenzy. During this development stage, all primary must have mechanics were implemented: Enemy AI, Cat wander behaviour, simple win conditions, laying bait, timer and score functionality.

All models were kept to be simple at this stage – with no level assets being added as of yet, walls were used to test obstacle avoidance and sensing of the AI agents. The base cat model was animated with a simple run animation in Blender and imported to the scene.



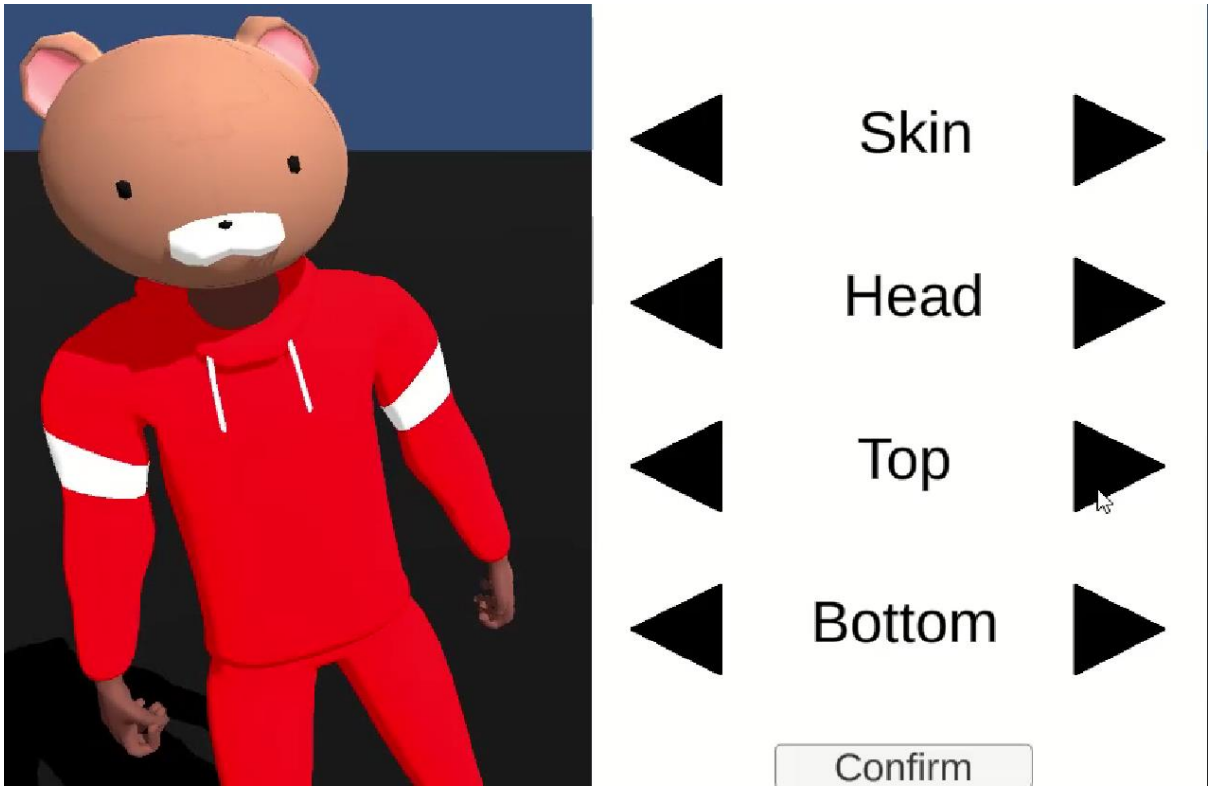
[*Figure 25: Base Level Design for Prop-Hunt Scene*]

The mechanics for prop hunt were also implemented at this stage once the must have requirements were completed. The player was able to switch between different items in the level and once the cycle has been completed – they revert back to the original body.



[Figure 26: Blender cat Model]

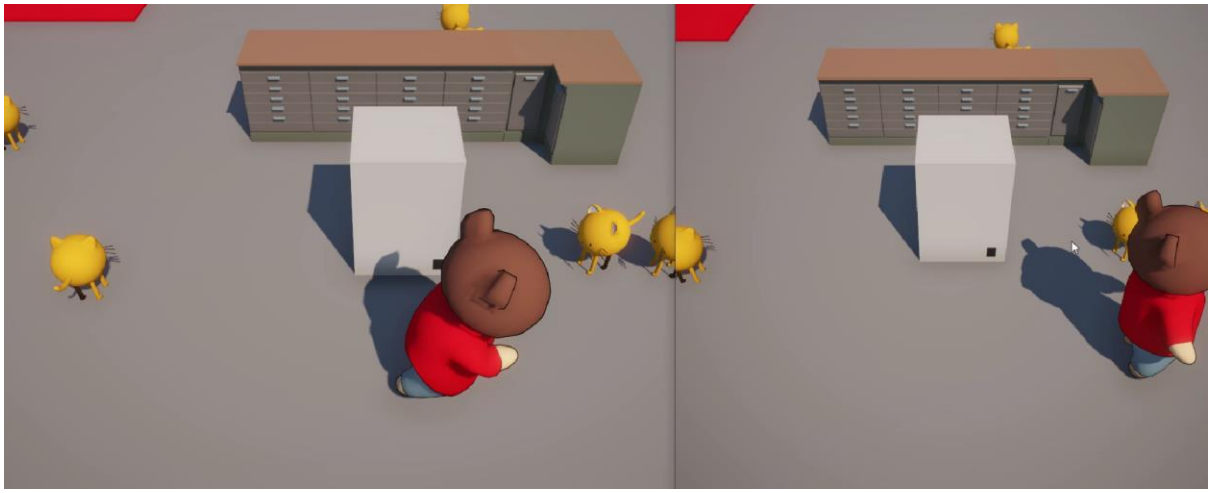
Prototype 2



[Figure 27: Player Customisation]

In Prototype 2 the player was able to customise their character in single player mode. A design choice was also made to keep a separate model for single player and multiplayer -with the multiplayer version being more cartoony and having a simpler form.

[*Figure 28: Prophunt Syncing]*



[Figure 29: Syncing of player movement and objects]

[Figure 30: Syncing of Player movement]

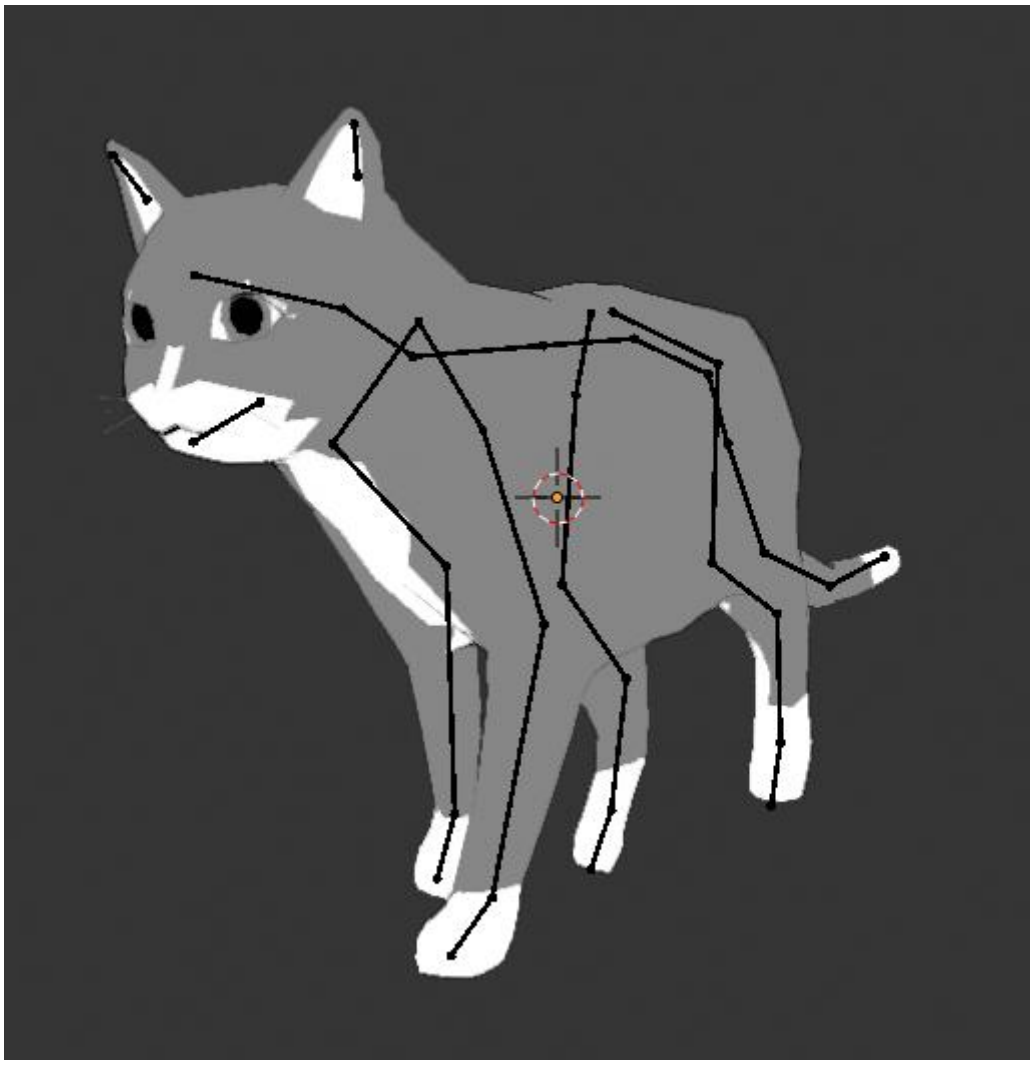
During the development of the second prototype, the multiplayer mechanics were polished.

Both Photon Pun and Unity Netcode have been used to test multiplayer connections and a server-client architecture was used in both instances. During prototype 2 the prophunt mechanics were polished further and player models were added. Netcode was used to implement this as photon had some issues with deactivating and activating gameobjects once instantiated.

After receiving feedback from players during the black box testing (*See Testing and Evaluation Section)* it was disabled in the gameplay as majority of the prophunt implementation is not entirely compatible with Photon Pun (which is what the remainder of connections utilise).

By this point a majority of the mechanics have been implemented and the remainder of the game just required some polish.

Prototype 3 [Final Outcome]



[Figure 31 New Cat Model]

The final prototype contains a new model for the cats found in the game, a majority of the time for the last few sprints were spent polishing up the scripts in the cat behaviour and creating a new cat model. This model contains face and shape keys which allow it to blink and open it's mouth. There was however not enough time to implement these into the animations currently added. The animations are more lifelike and positive feedback was received on the cat's model.

Ethical, Legal, Social issues and Data Security issues

PEGI rating:

Feline Frenzy takes inspiration from other similar games; the reputation mechanic from Assassins creed brotherhood (*Assassin's Creed Brotherhood (2017)*), the AI behaviour and trap mechanic in Hello Neighbor (*Hello Neighbor (2018*)) and the care mechanics in Catz 2(*Petz: Catz 2 DS (2007)*) and Nintendogs (*Nintendogs: Dalmatian and Friends (2005)*). The rating for each of these games are PEGI 3, PEGI 7, and PEGI 18. Catz 2 and Nintendogs have a low PEGI rating of 3 due to there being no scary imagery or sounds and content that is appropriate for young children. Hello Neighbor's PEGI rating is 7 due to it containing scary images and sounds for young children, the game contains no violence. Assassin's Creed Brotherhood has a PEGI of 18 due to graphic violence, language and nudity. Feline Frenzy will therefore be rated PEGI 7 as it does not contain violence or sinister themes, however it is also not suitable for PEGI 3 due to sounds and the chase sequence.

Intellectual property

The potential assets all have a free license for both personal use and commercial use as long as correct crediting is given. Assets made in Blender allows for both commercial and personal use under the General Public Licence (GPL).
A personal Unity license was used to produce Feline frenzy which allows for both commercial use and personal use if the amount earned from the game is under $100,000. Furthermore, the game does not currently have any in-store purchases or generate revenue. A personal license allows for Feline Frenzy to be sold commercially or contain in-app purchases in future updates if the revenue remains below the amount.

DaFont is a website which provides access to thousands of fonts for personal use, as well as those which can be purchased. Two fonts have been used and are free to use under the Creative Commons License. FreeSounds.org is another site which contains music and sound effects for both commercial used and personal use, sound effects used within Feline Frenzy fall under the Creative Commons License. The remainder of assets used withing Feline Frenzy have been downloaded from the Unity Asset Store and are permitted to be used royalty-free within the game.

All assets, tutorials, sources, and intellectual property used within the project have been fully cited and referenced.

Data security

Feline Frenzy incorporates multiplayer aspects which enable players to engage in competitive gameplay without storing sensitive player data such as IP addresses. The multiplayer connection is made through the Photon Pun2 framework, which facilitates multiplayer connectivity. The networking code itself does not store personal IP addresses or other sensitive information about the client, as a result, players can engage in seamless multiplayer interactions without concerns regarding data privacy.

The game operates on a server-client authoritative architecture where the server acts as an intermediary facilitating communication between players. The design choice ensures that no direct connection is establishes between players and therefore minimises potential security vulnerabilities.

The players are able to communicate with each other using the in-game chat function, it however does not store or retain any chat logs beyond a set amount of time within each game session. The game will store basic personal details for usernames chosen in multiplayer game mode but will not store them beyond the play session.

Representation and ethical issues

Feline Frenzy has a central theme of characters breaking into other people's homes which is presented in a light-hearted whimsical style. While the game is purely for entertainment, it is important to address concerns regarding the portrayal of illegal activities such as trespassing. A balance has been achieved whereby NPCs are introduced to prevent the player from breaking in.

Feline frenzy aims to provide players with inclusive customisable characters and offers a range of skin tones for the players to choose from, promoting diversity and inclusivity. There is however room for improvement in gender representation. In future updates to the game, Feline Frenzy aims to incorporate a female character base type for more players to immerse themselves in the game world.

Ethical:

The theme of the game is set in a human world featuring human characters (the neighbours and police), and cats. While the game is based around breaking in and retrieving cats, there is *no cruel* treatment towards them. The player *cannot* throw them or hit them but will gently pick them up and place them down. The goal is to retrieve them, not to mistreat them. When the player enters their home, they will be able to care for the cats. There will be no discrimination of race, gender, or religion. The player cannot harm the neighbours or other NPC's. The player will be able to customise their own character – choosing the skin tone from a variety of inclusive colours as well as other accessories. In future updates a banning feature will be implemented to prevent online harassment.

Critical review and conclusion

Summary of achievements

A 3D stealth based arcade game was created efficiently with designs and gameplay that met the expectations of the target audience. Although some major changes were made with the prop hunt multiplayer mode and some errors with the networking, there were several accomplishments achieved. The game successfully captures the behaviour of a cat and has lifelike walk and eating animations as well as idle resting ones as discussed in the report.

Testing results

The black box testing aided greatly in identifying key areas of the game which needed to be worked on as the outcome was not as expected. The game was tested with a range of users from age 9 – 28 of different backgrounds.

Time management

The project's time management was very well planned in the beginning of the project and continued to be achieving and meeting goals set within each sprint until February. During this time there were many circumstances which affected my availability for studies and having time to work on the project which resulted in large setbacks pushing back "Should Have" and "Could Have" requirements. Furthermore, there were major issues with pushing onto gitlab which resulted in some progress being lost repeatedly throughout the duration of the project – namely the corruption of files.

What could have been done better

Perhaps frequently updating and pushing to GitLab would have aided in preventing some issues with lost and corrupted files. Another area which would see improvement would be the networking – while the components are synchronised, there are areas which require more smoother transitions and better methods. As of now the positions of items are correct on most players screens but a $1/5^{th}$ of the time results in one player not being able to interact with the environment due to syncing correctly.

The AI development was a large struggle as the parameters for the cat had to be adjusted constantly though trial and error – after implementing the animations it had to be re-adjusted again.

At the start of the project I had attempted to implement multiplayer mechanics prior to adding in the network connection, while I thought it was a good idea at the start it resulted In the backend not working correctly and multiple scripts which needed to be hidden or deactivated to find small issues.

Personal achievements and what was learned

I believe all personal aims identified in the beginning of this report were met and up to standard. One achievement I am particularly proud of is the customisation script for both multiplayer and single player. I was also happy with the working prophunt mechanic of syncing with other players – while it unfortunately did not make it to the final product due to incompatibility with Photon, it was something which was difficult to achieve. The AI behaviour of the cat is another area which I am proud of as well as having a positive response from testers with the gameplay being fun and enjoyable.

The modelling and animation of the characters was another area which I had spent a lot of time in and was a challenge to retopologies them in order for animations to work smoothly. It was a big challenge, and I learnt a lot throughout the process.

I also learnt that setting up connection between two builds for multiplayer input needs to occur first prior to implementing any mechanics. As I redid the multiplayer side of the project several times, I now feel more confident with programming the backend of it. I will utilise the knowledge, techniques and concepts learned in future projects.

Future works

While the game is relatively complete and contains a majority of the requirements mentioned, there is still room for improvement or adding more mechanics to the game which could not be achieved due to time constraints.

One feature heavily requested was to add in a care mechanic for users to pet and interact with the cat outside fo the level. This is something which would be added in future updates, alongside other multiplayer modes which are collaborative as opposed to the competitive one to achieve a balance.

Appendix

Project Proposal available in PDF titled "ZAHRA MIAH PROJECT PROPOSAL".

Gantt chart available in PDF titled "ZAHRA MIAH GANTT CHART".

Interim Demo available in note file titled "ZAHRA MIAH INTERIM DEMO VIDEO LINK".

Interim Demo available to view at:
https://www.youtube.com/watch?v=F0R_Mkayj_Q&ab_channel=ZStor2

References:

Blender Foundation, (1994) *Blender* (2.8) [Computer program]. Available at: https://download.blender.org/release (13 October 2022).

Buttfield. P, Manning. J, Nugent. T, (2019). *Unity game development cookbook : essentials for every game, First edition. O'Reilly Media.*

Byl, P. (2012) *Holistic Game Development with Unity: An All-in-One Guide to Implementing Game Mechanics, Art, Design and Programming.* Routeledge.

Dickinson, C. (2019) *Unity Game Optimization: Enhance and extend the performance of all aspects of your unity games.* Packt Publishing.

DitzelGames (2018) *Unet vs Photon Engine - A comparison*. 26 August 2018. Available at: https://www.youtube.com/watch?v=xLECRl1eyGk (Accessed: 18 October 2022).

Dhule, M.(2022). *Exploring game mechanics : principles and techniques to make fun, engaging games. New York, NY : Apress*

Eerie Guest Studios (2018) *Hello Neighbor* [Video game]. tinyBuild. Available at: https://www.nintendo.co.uk/Games/Nintendo-Switch-games/Hello-Neighbor-1407177.html (Accessed: 9 October 2022).

Fanastudio (2018) *Black Bubble* [Font]. Available at: https://www.dafont.com/black-bubble.font? (Accessed: 14 October 2022).

Francik, J. (2020) *'Autonomous Agents and Behaviour Patterns'* [Presentation slides]. CI5525: 3D Graphics Programming and Artificial Intelligence. Available at: https://kingston.app.box.com/s/v9s4bdhwewr5o8hz4yqmxpp628ge70ts (Accessed: 3 October 2022).

Gibson, J. (2015). *Introduction to game design, prototyping, and development : from concept to playable game - with Unity and C#.* Addison-Wesley.

Goldstone, W. (2009). *Unity Game Development Essentials. [Unknown].*

Goldstone, W. (2011). *Unity 3.x game development essentials: game development with C# and Javascript : build fully functional, professional 3D games with realistic environments, sound, dynamic effects, and more. Packt Publishing.*

Have Fun With Developing, (2021). *Mini Low Poly Pack*. [Online] Available at: https://assetstore.unity.com/packages/3d/environments/mini-low-poly-pack-185471 (Accessed: 6 October 2022)

Keith, C. (2011) *Agile Game Development With Scrum*. Indiana: R R Donnelley.

Khatib, S. (2021). *Bag Face Pack*. [Online] Available at: https://assetstore.unity.com/packages/3d/characters/bag-face-pack-40222 (Accessed: 6 October 2022)

Kyaw, S. A.(2013). *Unity 4.x Game AI Programming, 1st Edition. Packt Publishing.*

Menard, M.(2012). *Game development with Unity. Boston, MA : Course Technology.*

Microsoft (2022) *Fundamentals of garbage collection.* Available at: https://learn.microsoft.com/en-us/dotnet/standard/garbage-collection/fundamentals (Accessed: 13 October 2022).

Nintendo EAD (2005) *Nintendogs: Dalmatian and Friends* [Video game]. Nintendo.

Nintendo EAD (2011) *Nintendogs + Cats* [Video game]. Nintendo. Available at: https://www.nintendo.com/store/products/nintendogs-cats-golden-retriever-and-new-friends-3ds/ (Accessed: 18 October 2022).

PEGI (2017) *Pegi Public Site*. Available at: https://pegi.info (Accessed: 9 October 2022).

Photon Engine, (2018) *PUN 2 – FREE.* (Version 2.41). [Online]. Available at: https://assetstore.unity.com/packages/tools/network/pun-2-free-119922 (Accessed: 18 October 2022).

Smith, M.(2021). *Unity 2021 cookbook : over 140 recipes to take your Unity game development skills to the next level, Fourth edition*. Packt Publishing

Studio Billion, (2022). *Free: House Interior*. [Online] Available at: https://assetstore.unity.com/packages/3d/props/interior/free-house-interior-223416 (Accessed: 6 October 2022)

tinyBuildGAMES (2017) *Hello Neighbor: AI Rundown*. 25 May 2017. Available at: https://youtu.be/Hu7Z52RaBGk (Accessed: 3 October 2022).

Ubisoft Montreal (2010) *Assassins' Creed Brotherhood* [Video game]. Available at: https://www.ubisoft.com/en-us/game/assassins-creed/brotherhood (Accessed: 9 October 2022).

Ubisoft Paris (2007) *Petz: Catz 2 (DS)*. [Video game]. Ubisoft.

Unity Technologies (2005) *Unity* (2019.4.40f1). [Computer program]. Available at: https://unity.com/download (Accessed: 13 October 2022).

Unity Technologies (2022) *Netcode for GameObjects*. [Online]. Unity Technologies. Available at:https://docs-multiplayer.unity3d.com/netcode/current/installation/install/index.html (Accessed: 18 October 2022).

Watkins, A.(2011). *Creating Games with Unity and Maya: How to Develop Fun and Marketable 3D Games.* Oxford:Routledge.

Watkins, *R.* (2016). *Procedural content generation for Unity game development : harness the power of procedural content generation to design unique games with Unity, 1st edition. Packt Publishing*