Figure 0.1. A cat astronaut, mirroring the theme of the game.
Image by freepik. Retrieved from https://www.freepik.com/free-ai-image/cute-cat-space_82550977.htm.

Title of the Report “HUNGRY SPACECAT”

By Anita Pal (200208268)

(9478 words).

Link to the GitHub repository ♡

In partial fulfilment of the requirements for the degree of Bachelor of Science, BSc, Port Talbot, 2024

Field of study as it appears on the student record sheet: BSc Computer Science (Game Development)

ACKNOWLEDGEMENTS

I could have never started this degree without the support of my partner, David. You challenged me to become a better developer and made me believe I am not useless.

I also thank all the playtesters who offered criticism, advice, and support. You guys have been fantastic.

1. INTRODUCTION

1.1. Project Template

Hungry Space Cat builds on the 'Project Idea Title 1: Arcade Game' template.

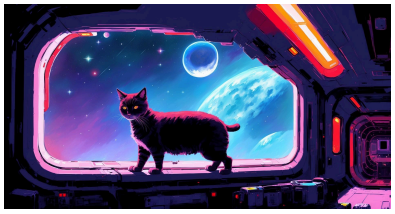


Figure 1.1. The game is cat-themed, as cats are fun, especially in space! [1].

1.2. Project Description

Hungry Space Cat is a 2D feline-inspired PC arcade game, as shown in Figure 1.1. Regarding gameplay, it takes inspiration from Namco's *Pac-Man*, a 1980 maze action game [2], and -- in its appearance -- *Asteroids*, a 1976 space shooter by Atari [3], but includes a more modernised look and feel. It uses arrow keys as controls to simulate an arcade game experience. ESC -- utilised for pausing -- is the only additional control employed for the game.

With its menu depicted in Figure 1.2, the game features a hungry astronaut cat pursuing purple space bugs. At the same time, it has to evade UFOs, flying hamburgers, ghost dolls, spaceships, or a combination thereof.

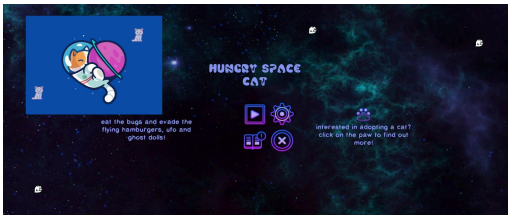


Figure 1.2. The menu scene has an astronaut cat image [4] and a space-themed background [5]. A white ghost cat [6] further enhances the menu's design. The overall aim is for the UI design to be user-friendly and easy to understand regarding controls and instructions – just like its main inspiration, *Pac-Man* [7].

Like in *Pac-Man*, the cat earns points for eating the space bugs while avoiding annoyed enemies that could spell the end of its feast (and life!) [8]. The player advances through increasingly challenging levels to eat all bugs without losing the cat's nine lives.

The more difficult levels include a spaceship shooting at the space cat character, which remains defenceless, with only its speed and wit allowing it to evade its opponent; the intention behind this gameplay is to mimic the deceptive simplicity of arcade games [9] while making the game fun to play and something users would want to return to [9, 10].

The background of *Hungry Space Cat* engages a space theme to pay tribute to *Space Invaders* and the previously mentioned *Asteroids*. Similarly to these titles, realism is not at the heart of *Hungry Space Cat.* Its design and gameplay use silly/cute elements to appeal to a broader audience via bright colours, happy music and relaxed gameplay to heighten positive emotions [7].

The game aims to be playable and fun without requiring much investment or time for the player to

understand its plot [9]. It has two modes: one that aims to be easy and relaxing, while the other – following the trajectory of arcade games becoming more complex over time [9] – poses more of a challenge, adding impulse to the player's movements and including more levels. Both modes promote learnability, allowing users to get used to the game's mechanics [7] while considering preferences such as slower speed.

1.3. Project Motivation

Academic/Personal Motivation

Arcade games—one of the earliest forms of electronic entertainment [10]—offer aspiring programmers a glimpse of the magic behind creating them [11]. *Pac-Man* uses pathfinding and chasing/tracking algorithms [10], e.g. the Pursuit-Evasion Game (PEG), which focuses on pursuers catching an evader as quickly as possible [12].

Not only do students benefit from learning about such algorithms [11], but researchers have proven Pac-Man only needs two ghosts to capture the player effectively [12]. Simultaneously, titles such as *Asteroids* need collision detection and distance calculation, allowing researchers and students to explore these topics visually [11]. Lastly, arcade games are practical tools for teaching children to learn new languages because they encourage players to interact directly with the material [13].

Hungry Space Cat addresses my affection and love for cats, giving me a platform to hone my skills in writing games to create ones accessible to different players.

Target Audience

The target audience of *Hungry Space Cat* mirrors those that Toru Iwatani, the creator of *Pac-Man*, intended for his game – in particular, he wanted it to be accessible to a diverse range of players [8]. Therefore, aligning with the fact that video games appeal to people of various ages [13], the game is suitable for people who:

- ★ Enjoy a game without requiring instruction manuals [14].
- ★ Wish to play a game with simple rules [10] and engaging gameplay [7, 10, 13].
- ★ With health issues preventing them from playing a game for an extended time.
- ★ Enjoy light-hearted games with cute creatures and bright colours.
- ★ Enjoy games that allow them to become part of a safe environment that heightens their emotions [13] and makes them happy [7].
- ★ A neurodiverse audience[1], as video games provide a continuous activity that allows them to grow and make mistakes [15].

Objectives

Hungry Space Cat aims to be/have:

- ★ Be intuitive to grasp, as arcade games are easy to play/pick up [9],
- ★ Elicit cheerful emotions via its design and gameplay [7].

[1] Please note that while someone on the spectrum has written the game and has considered design elements appropriate for a neurodiverse audience, it is not intended exclusively for autistic players.

- ★ Simple, enjoyable visuals [9] while defying the concept of fast-paced gameplay [13] by offering the player a choice between two modes, thus following the principality of accessibility of games such as *Asteroids* that had adjustable levels [7].
- ★ Provide accessibility/learnability features for players by using simple instructions and straightforward controls [7].
- ★ Serve to motivate players to explore other arcade games using nostalgic design elements [14].
- ★ Encourage people to adopt cats by having them actively engage with a cat character [13].

1.4. Report Overview

The literature review explores *Pac-Man* and *Asteroids*, concluding with an overview of feline-based games and a web application with interactive art/games.

The design chapter mentions the game's design, flow and mechanics, whereas the implementation chapter explains some of its features.

The evaluation chapter examines the overall project and its success. Furthermore, using data from playtesting sessions and peer-graded tasks, this chapter presents pie charts and word clouds to show the results visually.

The conclusion reflects on the experience, with some lessons for future game projects.

(954 words).

2. LITERATURE REVIEW

2.1. Pac-Man

History

A 2008 report revealed that 94 % of US consumers recognise *Pac-Man*, beating *Mario* in popularity [8]. According to its creator, Toru Iwatani – a game designer without formal training [8], the gentle gameplay of *Pac-Man* was intentional as, in the late 1970s, arcade centres only contained violent games focused on killing aliens [2]. He felt these arcade games were playgrounds for boys, leading him to develop a game for women and couples [2] with bright and joy-inspiring graphics [7].

A Japanese fairytale with a demon-eating creature protecting children from monsters inspired the creation of *Pac-Man*'s prototype [2]. In *Pac-Man*, the monsters from the fairytale became four ghosts [9], each with a unique personality [2, 16]. Furthermore, Toru Iwatani used the Kanji for eating – *taberu* 食 [17] – as a premise for the game and the one for mouth – *kuchi* 口 [18]-- with its square shape as the basis for the character's design [2, 8].

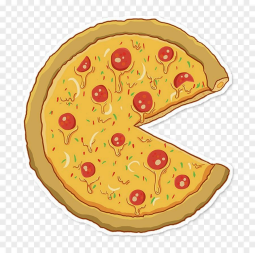


Figure 2.1. According to a popular urban myth, a pizza missing a piece influenced the shape of Pac-Man [8], as shown above [19].

Iwatani insisted on the simplicity of *Pac-Man'*s design—a yellow disc with a mouth resembling a pizza slice, as shown in Figure 2.1—to streamline the game and keep its gameplay simple without the player requiring a manual [8, 14]. Despite the simplicity, while playing the game, the player's strategy is crucial to winning it [20].

Further, Iwatani restricted the gameplay to a maze where players move in the following directions - up, down, left, and right [8]. As a maze game, *Pac-Man* contains one screen/playfield with little to no scrolling; multiple game levels keep the player's interest alive [20].

Core Mechanics

Pac-Man features a two-dimensional maze [21, 22]. Using a labyrinth relates to *Pac-Man's* characteristics as a maze game [20], where the player navigates the game character via a four-way joystick [21]. The maze has 240 non-flashing pills, each worth 10 points, and four flashing ones, referred to as energisers, worth 50 [22].

The game begins with the four ghosts stationed at the maze's centre, called the ghost house [21]. The ghosts are released individually and chase after Pac-Man, aiming to consume them [22]. Pac-Man

starts with three lives, losing one each time a ghost catches them [21, 22]. When a life is lost, the ghosts and Pac-Man return to their original positions, but any eaten dots remain [22].

When Pac-Man eats an energiser, the ghosts turn blue briefly, allowing Pac-Man to eat them and score points [19, 20]. Upon finishing a level, the game moves to the next one [22]. While the game is technically unlimited, a bug prevents the player from surpassing level 255 [22].

The speed of the ghosts is constant, except when travelling through a tunnel [21, 22]. The layout of the game remains the same [21]. However, each level increases in difficulty due to changes in Pac-Man's speed and the ghosts' behaviours [21], as shown in Figure 2.2 and elaborated on in Table 1.1. There are no modifications after reaching level 21 [22].

TABLE I
CHARACTERISTICS OF THE GHOSTS IN PAC-MAN [4].

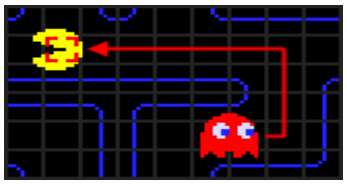
Colour	Orange	Blue	Pink	Red
Name	Clyde	Inky	Pinky	Blinky
Aggressiveness	20%	50%	70%	90%
Territory	Southwest	Southeast	Northwest	Northeast

Figure 2.2. The table shows the ghosts' personalities/aggressiveness levels [21].

Ghost Behaviour

Each of the ghosts has a target – a specific tile – assigned to them to reach [21]. The colour of a ghost's eyes indicates the direction it travels, which cannot change unless Pac-Man has eaten an energiser [20]. The ghosts have three modes – scatter, chase, or frightened [21, 22], with the predominant one being chase, where the spectres use Pac-Man's position to find their target tile [21]. When in scatter or chase mode, the ghosts travel in waves with a set timer, only pausing when in frightened mode [21].

Table 2.1. The table describes each ghost and its specific behaviours.

Name of Ghost and Screenshot(s)	**Core Behaviour**
Red Ghost / Blinky  **Figure 2.3.** The red ghost's target tile is where Pac-Man is located [22].	Starting outside the ghost house, the red ghost, described as 追いかけ, *oikake*, which means 'pursuer' in English, is the first threat [22]. It immediately starts following Pac-Man, as depicted in Figure 2.3 [22]. Blinky's speed increases when Pac-Man has 20 pellets left to eat [23]. Its scatter mode also changes, ensuring it remains in chase mode rather than targeting the tile at the upper right corner of the maze [22]. These behavioural shifts make Blinky the deadliest of the ghosts [23].
Pink Ghost / Pinky	The pink ghost starts inside the ghost house but exits immediately [22], moving around the maze's walls in an anti-clockwork pattern [23]. Although the ghost is nicknamed 'Pinky' [22, 23],

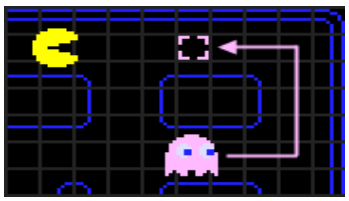
<table>
<tr>
<td> 

Figure 2.4. The pink ghost’s target tile tries to move four spaces ahead of Pac-Man [22].</td>
<td>the Japanese description of its personality 待ち伏せ, machibuse (ambusher), better suits its behaviour [22]. While Pinky does not move faster than the other ghosts, its target is where it anticipates Pac-Man will next be – displayed in Figure 2.4 [20].

More precisely, in chase mode, it looks at Pac-Man’s current position [20] and the four spaces ahead [22, 23] in an attempt to get in front of the player [22]. However, this behaviour only works if Pac-Man moves left, down, or right; an overflow error prevents it when the player moves upwards [22].

In scatter mode, Pinky moves towards the top left corner of the maze in an anti-clockwise, circling motion [23].</td>
</tr>
<tr>
<td>Blue Ghost / Inky

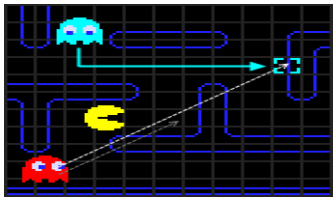
 

Figure 2.5. When calculating Inky's target, Pac-Man and Blinky’s positions are considered [22].</td>
<td>The blue ghost remains in the ghost house until Pac-Man has eaten 30 pellets [22]. Its Japanese description is 気紛れ, kimagure, meaning ‘whimsical’ in English [22].

Inky’s behaviour is difficult to predict as it uses Pac-Man’s and Blinky’s positions to calculate its target, as shown in Figure 2.5 [22].

It patrols the area close to the red ghost's location, becoming more random the further away it is from it [23]. In other words, if Blinky is close to the player, so will Inky [22]. In scatter mode, Inky guards the lower corners of the maze [23].</td>
</tr>
<tr>
<td>Orange Ghost / Clyde

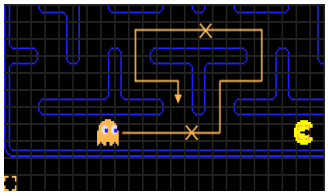
 

Figure 2.6. The image above shows how Clyde’s behaviour changes when close to Pac-Man [22].</td>
<td>The orange ghost is the last to leave the ghost house and does not move until a third of the pellets have been eaten in the first level [22]. The Japanese description for its behaviour translates to お惚け, otoboke or feigning ignorance’ [22].

While Clyde gives the impression of doing its own thing, it has two modes for targeting Pac-Man that it switches depending on its proximity to the player, which Figure 2.6 demonstrates [22].

It calculates how far it is from Pac-Man to determine its target. If it is eight spaces away, it behaves like Blinky, following the player directly; however, as soon as it is less than eight spaces from Pac-Man, its target changes to one of the ones it has in scatter mode, located at the bottom corner of the maze [22, 23].

As such, Clyde switches between following Pac-Man and retreating when it gets too close [22].</td>
</tr>
</table>

Evaluation

Due to its popularity, *Pac-Man* has not only inspired clones (e.g. *Cat Trax*) but has also been used in academic research [12, 16, 21]. *Pac-Man's* easy-to-understand yet increasingly challenging gameplay shows why arcade games are widespread; these aspects should remain integral in any project aiming to create one, as they keep players engaged [9, 10]. Furthermore, its universal appeal [8] reveals how arcade games can unite a diverse community of players [8, 12] when evoking feelings of nostalgia [14]. In terms of design, its simple controls and clear instructions make it an inspiration for modern games [7] in terms of overall playability. The game's colourful aesthetics also show how games can make players feel relaxed and engaged [7].

2.2. Asteroids

History

In December 1979 [3], Atari released the shoot-em-up *Asteroids* [24]. In line with its characteristics as a shooter arcade game [18], the player controls a spaceship on a single screen to evade asteroids [3, 24]. The game has an easy-to-learn but difficult-to-master gameplay that keeps people returning to it to beat it [24]. Not only did *Asteroids* make Atari a household name [3], but it sold 75,000 units, making it one of the best-selling coin-ops [22].

Beyond its financial success, *Asteroids* helped arcade games become popular among players from different walks of life [22], including professionals in their 30s and 40s who played it during their lunch break [22]. As Atari's best-selling game, it dominated not only the scene in arcades but also areas such as waiting rooms, bars and shopping malls, among other locations [3].

Created by Ed Logg and depicted in Figure 2.7, *Asteroids* was a response to an unsuccessful game in which players tried to shoot asteroids [3, 22]. Describing the game as dull, Ed suggested an alternative: players should blow up the asteroids instead [24].

During a conversation with Lyle Rains -- his boss at Atari, Ed stated in contrast to a successful title such as *Space Invaders* -- a static shooter [20] with one-directional controls that only let the player move left and right -- he envisioned a scrolling game that allowed for two-directional movements, creating player satisfaction [24] by giving them more free range movement [20]. The proposed game had simple yet addictive gameplay involving a flying spaceship and rocks that became smaller upon being hit until they disappeared altogether [3, 24].

Figure 2.7. Ed Logg poses next to *Gold Asteroid*s, created to celebrate building 50,000 units [16].

While Ed and Lyle agreed on the concept for *Asteroids*, they did not initially see eye to eye when it came to the game's format [3, 24] – Lyle wanted it to use raster, while Ed desired to utilise vector technology, which – according to him – had a higher resolution and allowed for more control [24]. Ed contacted the engineer Howard Delman [3, 24] to implement vector technology on its hardware. Previously, Delman had worked on *Lunar Lander* -- Atari's first game to use vector technology [24].

Game Design

While waiting for the hardware of *Asteroids*, Ed Logg documented the game's basic concepts [3], focusing on its design and tweaking various ship settings, such as its inertia, to find out what worked best in terms of gameplay [24].

As the Atari labs were open, engineers often walked between them and played games under development [24]. Consequently, Ed Logg received positive feedback from his colleagues [24]. However, Logg would also observe his colleagues playing and make amendments, such as moving the player away from the rocks to ensure that they hit more objects, as these increased the chance of collision [3, 24].

Despite there being no design processes in place, Ed sketched many versions of the ship, and – as noted by Mark Cerny – a colleague of Ed's at Atari, the secret to his success was that he planned the game, developing features in the correct order rather than focusing on complex algorithms [3]. Ed also engaged in feedback from two field playing testing sessions to fine-tune his game [24], making him one of the first game designers to realise the importance of gameplay. As stated before, he also ensured that the game was accessible, as it had a straightforward interface with simple instructions and allowed players to adjust the game's difficulty level to suit their needs [7].

Evaluation

While *Asteroids* has not enjoyed the same academic interest as *Pac-Man*, the game continues to be popular [3, 24]. Like *Pac-Man*, *Asteroids'* simple story and difficult-to-master gameplay show that these qualities make a successful arcade game [7]. Additionally, Ed Logg's approach to design — especially his emphasis on making the game learnable and accessible [7] – shows his innovative thinking – which feels inspirational.

2.3. Arcade-Based Cat Games

Cat Trax

Cat Trax (Figure 2.8), released in 1982 [27] or 1983 [28], is a clone of *Pac-Man*, featuring a cat on the run from three canines [27, 28]. To gain points, the kitty has to eat the catnip in the maze, and – anytime a green potion appears – it transforms into a dog catcher truck that sends dogs to a pound at the top of the screen [27, 28]. Once the potion wears off, the dogs start chasing the game character again [27, 28]. *Cat Trax* was developed for the Aradia 2001 [26] – an obscure home game system released in 1982 by UA.Ltd [27].

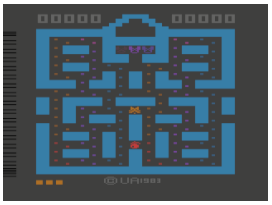


Figure 2.8. An image of the arcade game *Cat Trax* [27].

Cat Trax has a fun take on *Pac-Man*; however, it remains a clone with little to set it apart regarding creativity or originality. While games such as *Pac-Man* inspired *Hungry Space Cat*, it still aims to be something of its own.

Mappy

Released by Namco in 1983 [29, 30], *Mappy (マッピー)* is a side-scrolling maze game featuring cartoon-inspired cats and mice [29, 30]. It ran on the *Super Pac-Man* hardware modified to support horizontal side-scrolling [29, 30].

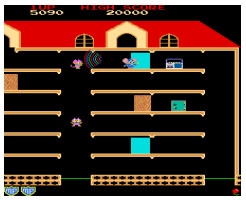


Figure 2.9. A screenshot of the arcade game *Mappy* [29].

In the game, shown in Figure 2.9, the player guides the police mouse, Mappy – a term derived from the Japanese nickname *mappo* – through a cat mansion to find stolen goods [29, 30]. To survive, Mappy must avoid the mansion's cats and traverse the building via trampolines [29, 30].

Mappy is more involved and complex than *Cat Trax*, but its story and motive are simple, and the slightly enhanced gameplay does not get in the way. *Hungry Space Cat* builds on Mappy's philosophy by extending a simple story with more elevated graphics, but these elements do not distract from the gameplay, emphasising its importance to players [25].

Pac Cat

Pac Cat by Divok, displayed in Figure 2.10, is an indie game with pixel graphics for mobile devices. In it, a cat has to eat all the points to pass the levels while running from bulldogs [31]. Reviews of the game describe it as cute but buggy, with the controls not working and the game being challenging from the get-go [31].

Consequently, *Pac-Cat* defeats the purpose that arcade game creators had in mind: for them to be simple but become more complex with increasing levels [8, 9]. *Hungry Space Cat* starts simple but becomes more difficult at each level, thus keeping in line with arcade game traits [9].

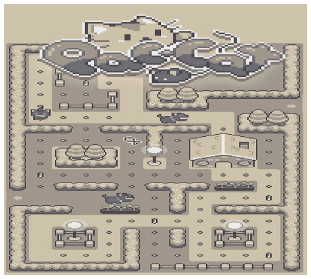


Figure 2.10. An image of the indie game *Pac Cat* [31].

2.4. Web-Based Game Project(s)

Space Cats

Space Cats is a web game application where users can play games and view interactive art, as shown in Figure 2.11.

Figure 2.11. The user can play one of the two games in *Space Cat* [32].

The application aims to bring people together who enjoy cute games. However, playtesters for the project described the gameplay of these two games as simple and lacking substance, which does not fit in with the concept of arcade games being challenging to master [9]. Using a web browser affected the games' graphics in size/resolution, further restricting gameplay.

(2500 words).

3. GAME DESIGN

3.1. Domain and Users

Hungry Space Cat's domain is arcade games with a simplistic design but challenging-to-master gameplay [9, 31]. Throughout their history, not only did the unsophistication of early arcade games allow the player to grasp all of a game's intricacies, but they also made them feel accomplished and eager to replay them to encounter new obstacles to solve [9, 33]. In contrast, modern games have excessive tutorials that affect a game's pacing, robbing players of the opportunity to feel challenged by cracking problems independently [33].

Taking cues from *Pac-Man,* with its diverse audience [8] and continuing appeal to people of all ages [13], the goal of *Hungry Space Cat* is to keep its players engaged in the story and want to return for more by providing them with increasingly challenging levels [9, 10]. As mentioned, the game also uses bright colours and cheerful gameplay to make players feel safe and upbeat [7]. In essence, the game's design elements—such as character and level aspects—are intended to be straightforward, with the goals and game mechanics similarly easy to gauge [9, 10].

Additionally, *Hungry Space Cat* considers accessibility, allowing players with diverse needs to play a more leisurely mode and contains menu options that hide more flashy features [7].

3.2. Core Mechanics and Gameplay

Core Mechanics

Table 3.1 outlines the core mechanics of *Hungry Space Cat*. With each new level, the number of bugs to eat increases, and the type of enemy changes, making the game less predictable. Players can choose between two modes, influencing how the space cat moves: the more difficult mode employs momentum. Using two modes makes the game playable and accessible to a broader range of players [8, 13].

Table 3.1. The core mechanics of the game.

Action	**Action Triggered**
In challenging mode, the player moves up, down, left, or right using momentum. In the more accessible mode, the player stops and starts instantaneously.	The cat navigates within the screen's boundaries, restricting its movements like the maze confides the player in *Pac-Man* [8].
The cat eats space bugs. The cat has no weapons or defences against the enemies, which are visually outlined via a pink shader [34] for clarity and learnability [7]. The player flashes red when injured by an enemy for the same reasons as the enemies' shader [7].	The cat gains 10 points for eating bugs, with the action similar to the player eating dots in *Pac-Man* [8], which should help players familiar with that game feel nostalgic [14].
Depending on the level, different enemies chase the cat; for example, four UFOs travel between the corners of the screen (Figure 3.6). The space cat must avoid them while hunting bugs	Like in *Pac-Man*, the cat loses nine lives, but the enemies do not run away when the pellets are gone [8]. Instead, the space bugs serve as a means to proceed to the next level.

(Figure 3.5). The player flashes cyan when eating a pellet, making it clear to the player that they have gained a point [7].	Further, *Hungry Space Cat* interprets the four ghosts as different enemies.

Gameplay

Figure 3.1. As the storyboard shows (using stock images from www.pexels.com), the concept of *Hungry Space Cat* is simple, adhering to the principle of arcade games being easy to play/pick up [9].

The gameplay of *Hungry Space Cats* – derived from an uncomplicated concept as shown in Figure 3.1 – is a sequence of:

- ★ The cat (Figure 3.4) enters the scene from any random location on the screen within its boundaries.
- ★ The cat moves up, down, left or right to look for bugs.
- ★ The UFOs enter the scene from the edges of the screen and then move back and forth within its boundaries.
- ★ The flying hamburgers enter the scene from different screen areas but follow the player.
- ★ The ghosts move based on wave points, while the UFOs – as mentioned previously – appear on the edges of the screen and move back and forth.
- ★ The spaceship that shoots follows the player.
- ★ Green skull UFOs move based on wave points.
- ★ The snails enter the scene from the edge of the screen and then move up, down, left and right.
- ★ Asteroids enter the scene and move around it.
- ★ Planets spawn randomly in a scene but do not affect the player.

Level transitions are indicated by a fade-in [35]; sound and visual effects—such as the player changing colour—indicate whether they have scored or lost a point; sound effects [36] also indicate whether the player has been injured or eaten a space bug.

3.3. Game Components

Tools

The game uses Unity's engine [36] and the C# programming language [37]. Unity [36] is a good choice, as many tutorials offer support and guidance, and previous modules also use the language. *Hungry Space Cat* also uses 2D to honour the original design of arcade games [9] and lure players familiar with the genre [14].

Components

Table 3.2 outlines the game components of *Hungry Space Cat*, each dealing with a game object and their related actions. These components do not reflect the implementation individually but form the basis for the game's structure.

Table 3.2. The table outlines the main game components.

Component	**Significance**
SpaceCat	The *SpaceCat* component deals with the player's movements and reaction to enemies regarding collision.
Enemies	The *Enemies* component handles the individual movements of each enemy and how they appear in the scene. This component also addresses the shooting behaviour of enemies.
Bugs	The *Bugs* component deals with the pickups' movement and reaction to encountering the player. It also deals with how they first appear in the scene.
AudioManager	The *AudioManager* component deals with the sound effects used during the game.
SceneLoadingManager	The *SceneLoadingManager* component deals with how the scenes are loaded.
ScoreManager	The *ScoreManager* component controls how the user's score and number of lives appear on the screen.
UI	The *UI* component deals with UI aspects, such as the *PauseMenu* or the *GameOver* scene.
Effects	The *Effects* component deals with enabling and disabling effects such as background scrolling.
Timer	The *Timer* component gives the game an internal countdown between levels, allowing the player to prepare for it.

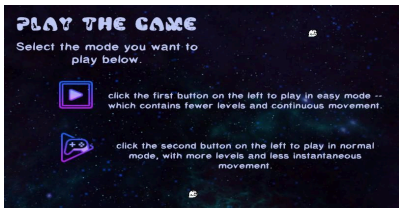
3.4. Game Design

Menu and UI Design

Figure 3.2. *Sudocats* boasts a colourful and cheerful UI interface.

The cat-based indie game *Sudocats*—shown in Figure 3.2 —inspired the menu design of *Hungry Space Cat*. In other words, the aim was to create a colourful, bright, and friendly UI. The reasoning behind a less simplistic interface was to give players more options and make the game stand out in terms of aesthetics. Table 3.3 shows further UI and menu design elements:

Table 3.3. Table 3.3 shows some of the menu and UI design sprites.

Name of Sprite	Description
 **Figure 3.3.** The current menu emphasises readability and accessibility [7].	Figure 3.3 shows that the current design prioritises providing buttons and text to help the player navigate throughout the game, using text and visual cues in the button design.
 **Figure 3.4.** Users can choose between an easy or hard mode, making the game more accessible [7].	The UI design of *Hungry Space Cat* prompts users to play the game, enables them to change its settings (Figure 3.3), and allows them to choose between two modes (Figure 3.4). In the *GameSettings* scene, the user can further adjust the player's speed, deal with specific effects such as the background effect, and turn off the audio.
 **Figure 3.5.** The menu scene contains a cat [38] and ghost sprite [39] to endear the design further.	*Hungry Space Cat* places images of cute sitting cats or ghost sprites in the menu to pay tribute to its feline mission of enticing players to adopt cats.
 **Figure 3.6.** The sleeping cat image fits the rest of the	Figure 3.6 shows a picture of a sleeping cat that displays when the player loses a life or finishes the game. The image plays into the theme of being endearing and cute.

feline-inspired UI design [40].	

Character Designs

Taking inspiration from *Pac-Man*, the character and level design are light-hearted [8] but of a higher quality. The game has a distinctly anime-inspired feel, paying tribute to Japanese culture. Apart from the UFOs used in the first level [41], all of the sprites come from *bevouliin*, a site that features game assets for game developers [4]. Using the same artist for sprites keeps the game's design consistent, as shown in Table 3.4.

Table 3.4. The table displays some of the sprites used for the enemy and character design.

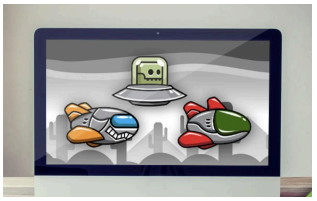
Name of Sprite	**Description**
 **Figure 3.7.** The astronaut cat [4].	As shown in Figure 3.7, the cute astronaut cat sprite matches the game's peaceful and soothing atmosphere, which matches its intention to be playable for a diverse audience [8, 14].
 **Figure 3.8.** The space bug that the space cat has to consume to earn points [4].	The bug sprite's design – as depicted in Figure 3.8 – is endearing, emphasising the game's cuteness and non-serious nature.
 **Figure 3.9.** An enemy UFO that moves back and forth at the corners of the screen [39].	Like the other enemy sprites, the enemy sprite displayed in Figure 3.9 is non-threatening, fitting in with the game's theme of light-hearted cuteness.
 **Figure 3.7.** Ghost-based UFOs and spaceships are among the many enemies that space cats have to deal with [4].	In the normal mode, the player encounters enemies such as ghost UFOs and spaceships that shoot at them, as shown in Figure 3.7.

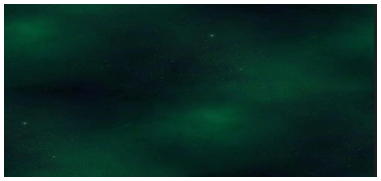
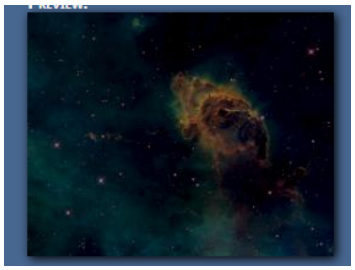
 **Figure 3.8.** Colour asteroids are further challenging in the last level of the normal mode [4].	Figure 3.8 depicts colourful rocks that serve as asteroids in the final level of the normal mode. The player has to avoid colliding with them to avoid losing points.

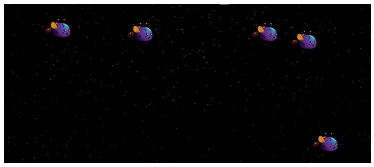
Level Design

As shown in Table 3.5, each game scene has a scrolling background, creating the illusion of the player being in space. Early game design versions involved a tilemap to draw borders around the screen. However, this design concept soon gave way to a more straightforward but effective space-related background [5] that required less work in the light of frequent changes.

Scenes involving a flying hamburger use a different background due to the need to avoid overlapping [42] and not being able to provide the scrolling background effect.

Table 3.5. Table 3.5 shows some of the sprites used for the level design.

Name of Sprite	**Description**
 **Figure 3.9.** The following background has another one on top of it [5].  **Figure 3.10.** The user can turn off the effect if they find it jarring.  **Figure 3.11.** Some scenes have a different background [42].	In Figure 3.9, the green background is laid on top of another similar one to provide the impression of being in space. Each of the three scenes features a differently coloured background [5]. The menu offers the option to reset the sprawling effect (Figure 3.10). Scenes involving flying hamburgers use the background [42] shown in Figure 3.11.

 **Figure 3.12.** The pickups are spawned randomly onto the screen.	To avoid confusion, the space bugs – shown in Figure 3.12 – are the only pickups available. Their amount changes depending on the level.
 **Figure 3.13** The lives and score text are at the top of the screen.	The number of lives (Figure 3.13) is represented by the number nine, with an image of a cat next to it – as a cat has nine lives, so does the player! The design intends to be cute and evoke nostalgia in players familiar with arcade games [14].
 **Figure 3.14.** The screenshot displays randomly spawning planets [43].	Figure 3.14 shows planets that spawn randomly in the scene and display text informing the user that the game is loading. The planets enhance the background further.

(1996 words).

4. GAME IMPLEMENTATION

4.1. Code Structure

Figure 4.1 shows the code structure of *Hungry Space Cat*:

Figure 4.1. The image describes the functionality of each category.

More specifically, as Table 4.1 outlines, there are seven categories that each serve a specific function – also elaborated on in Figure 4.1:

Table 4.1. The table provides an overview of the scripts used.

Category/Scripts Involved	**Purpose**
The *Animation* folder includes: - AnimatedSprite.cs - BackgroundSpriteScroller.cs - FadeAnimation.cs - SpriteEffects.cs	The *Animation* folder is responsible for the following: - Displaying arrays of sprites. - Handling the fading in and out of levels and other scene exits (e.g. game over). - The background scrolling effect is used within the game, apart from levels involving hamburgers. - Sprite effects, such as the character flashing cyan when scoring a point.
The *Audio* folder includes: - AudioPlayer.cs - HandleAudio.cs	The *Audio* folder takes care of the following: - Audio effects include picking up a coin or the player taking damage. - The muting and unmuting of sound. - The game's background music.
The *Controllers* folder includes two subdirectories - *EnemyControllers* and *OtherControllers*. *EnemyControllers* includes: - AsteroidController.cs	The *Controllers* folder handles the movement and collision of game objects, including enemies, pickups, and the player itself. More specifically, controllers deal with:

- FlyingHamburgerController.cs - FollowingSpaceshipController.cs - ShooterController.cs - SnailController.cs - UFOController.cs - WaveEnemyController.cs *OtherControllers* includes: - BugController.cs - SpaceCatController.cs	- How asteroids rotate in a scene. - How enemies follow a player. - How spaceships shoot bullets. - How snails and other enemies move. - How the player reacts to colliding with enemies. - How the pickups react when eaten by the player. - How the player reacts upon gaining a point or taking damage. - How the player moves depends on user input.
The *GamePlay* folder includes: - AdjustSpeed.cs - DealWithEffects.cs - HealthKeeper.cs - SceneLoaderManager.cs - ScoreKeeper.cs Timer.cs	The *GamePlay* folder deals with the following aspects of the game: - Adjusting the player character's speed, i.e., the space cat, by making them either slow or faster. - Turning on and off effects such as the background scrolling effect. - The calculation of the game score and the amount of lives available. - The changing of scenes depending on the game progression or the user input. - The internal timer ensures the game loads between levels.
The *Helpers* folder includes: - ControllerHelper.cs - SpawnerHelper.cs - WavesConfig.cs	The *Helpers* folder includes reusable functions used in the other directories, such as: - Clamping sprite movements. - Following the player - Flipping sprites. - Spawning objects into the scenes in waves.
The *Spawners* folder includes two subdirectories – *EnemySpawners* and *OtherSpawners*. *EnemySpawners* includes: - SpawnAsteroids.cs - SpawnUFOs.cs - SpawnWaveEnemies.cs *OtherSpawners* includes: - SpawnBugs.cs - SpawnPlanets.cs	The *Spawners* folder deals with: - The spawning of objects at the edges of a screen. - Spawning objects randomly within a scene.
The *UI* folder includes: - GameOver.cs - PauseMenu.cs - UIDisplay.cs	The *UI* folder deals with UI-specific functionalities such as: - Displaying the game over text and visuals when appropriate. - Displaying the pause game menu and its buttons. - Any text displayed during the transition of game levels, including the game over one.

Outside the *Scripts* folder is a *Tests* directory broken down into scene-specific sub-folders containing tests.	This folder includes: - Automated tests using Unity's play mode that check for basic functionality, such as loading game elements and prefabs used.

4.2. Animation Scripts

AnimatedSprite

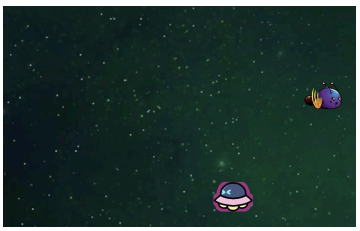


Figure 4.2. The UFO and space bug sprites use the *AnimatedSprite* script to show visual changes.

The *AnimatedSprite* script is responsible for animating the sprites, including how an array of them is displayed during the game to reflect changes in movement over time [44]. The script is attached to specific game objects, such as the space cat, enemy, and bugs, as shown in Figure 4.2 and explored in Table 4.2.

Table 4.2 The table explores some of the *AnimatedSprite* code.

Function/Code Snippet	**Purpose**

void MoveAnimationForward()
{
…
{
animationFrame++;
CheckAnimationFrameConditions();
}
}
	The following function increments the animation frame variable by one when the game object's SpriteRenderer component is enabled, and the animation frame conditions are fulfilled [44].
	
	A *SpriteRenderer* component is responsible for how a sprite is rendered and depicted on a screen [45].

void CheckAnimationFrameConditions()
{
…
if (animationFrame >= 0 && animationFrame < spritesArray.Length && !PauseMenu.isPaused)
{
spriteRenderer.sprite = spritesArray[animationFrame];
}
}
	The *CheckAnimationFrameConditions* function, called in the *MoveAnimationForward* method, checks that animation continues seamlessly if the sprite array is smaller than the animation frame by setting its current index to one of the animation frames [44]. It also checks that the game is running/not paused [46].
	
	If the animation frame is larger than the length of the sprite array and smaller or equal to 0, then it is set to zero so that the sprite animation can continue looping again [44].

BackgroundSpriteScroller

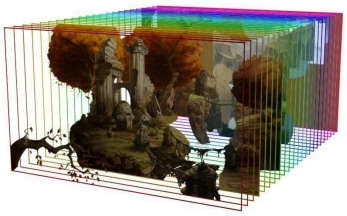


Figure 4.3. The image uses the layering method to show how parallax scrolling looks from the front [47].

The *BackgroundSpriteScroller* helps with parallax scrolling, in which background images move more slowly than forefront ones [47]. Parallax scrolling became popular in the early 1980s, along with video arcade games such as *Jump Bug* [47].

It uses layering, as shown in Figure 4.3, which involves multiple backgrounds and changing each layer's position by a different amount in the same direction [48]. The other method engages pseudo-layers using sprites drawn by hardware on top of layers [47].

Hungry Space Cat uses the layering process to honour arcade games, as shown in Table 4.3.

Table 4.3. The table describes the background scrolling effect.

Function/Code Snippet	**Purpose**

void Update()
{
if
(DealWithEffects.backGroundEffectsEnable
d)
{
_offset = moveSpeed * Time.deltaTime;
_backgroundMaterial.mainTextureOffset +=
_offset;
}
…
}
	Within the script's *Update function*, a private offset variable is multiplied by the move speed and *Time.deltaTime* to create smooth movement [48].
	
	Afterwards, the offset is added to the background material's texture offset to create the scrolling effect [48].
	
	A static boolean checks whether this option has been triggered (i.e. the user has to choose to turn off the effects in the *Game Settings* menu).

To ensure smooth movements with the scrolling backgrounds, Unity uses *Time.deltaTime*—displayed in Figure 4.4—which measures the interval between the current frame and the last one, ensuring that the animations and movement of game objects are consistent across platforms and different forms of hardware [49].

$$\text{Time.deltaTime} = \frac{1}{\text{frame rate}}$$

Figure 4.4. The formula helps normalise the calculation of the rate at which a machine renders a frame [49]. Calculating a game object's vector in Unity with the equation returns a fixed speed, no matter the frame rate [49].

4.3. Controllers

Controllers come from the MVC pattern, presented in Figure 4.5, which divides programming logic into three separate parts:

- ★ The model (the representation of information).
- ★ The view (interface).
- ★ The controller links the first two items together [50].

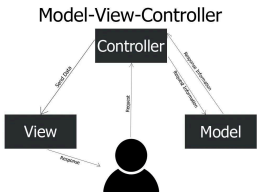


Figure 4.5. The image shows how the MVC pattern breaks down an application into separate parts that reflect the logic flow [50].

While *Hungry Space Cat* does not fully implement the MVC pattern, the scripts in the *Controllers* folder play an essential role in the movement of game objects (Figure 4.6).

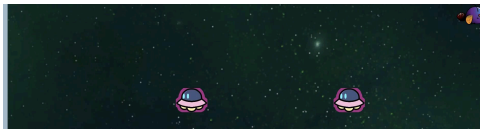


Figure 4.6. Code within the *Controllers* script folder ensures that the UFOs do not leave the boundaries but change directions when 'colliding' with it.

Controller-Script-Based Functions

Table 4.4 lists some of the functions across the *Controller's* scripts.

Table 4.4. The table lists some of the functions used in the *Controllers* directory.

Function/Code Snippet	**Purpose**

if (other.CompareTag("Boundary") &&
!PauseMenu.isPaused && Timer.timerFinished)
{
speed = -speed;
controllerHelper.FlipSprite(_spriteRenderer);
…
 	
}
	In the *UFOController*, using a reference to the *ControllerHelper* script, the UFO flips (Figure 4.6) and changes directions (as evidenced by the speed changing from a positive or negative value). The variable 0.4f represents the scale passed as an argument to the *FlipSprite* function.
	
	The code snippet is placed inside Unity's *OnTriggerExit2D* function, which handles how objects are triggered when another one leaves its trigger collider [51].
	
	Collision only occurs if the pause menu remains unactivated and the timer has finished (the game is not in the process of loading).

if (other.CompareTag("SpaceCat") && !_wasEaten && !PauseMenu.isPaused && Timer.timerFinished)
{
_wasEaten = true;
_audioPlayer.PlayPickupClip();
_scoreKeeper.ModifyScore(pointsForBugsEaten); gameObject.SetActive(false);
Destroy(gameObject);
}
	The following code snippet in the *BugController* script shows how the space cat object coming into contact with the bug leads to a series of consequences, which include:
	
	- The activation of the SFX audio pickup clip.
	- The score's modification [53].
	- The deletion of the bug itself [54].
	
	Unity's *OnTriggerEnter2D* function deals with how objects are triggered when they enter a trigger collider [52].
	
	As with the code snippet in the previous row, a collision occurs only when the game runs.

if (other.CompareTag("UFO") && !PauseMenu.isPaused && Timer.timerFinished)
{
…
if (_healthKeeper.GetLives() == 0)
{
CatDeath();
}
}
	The following code snippet, located with the *OnTriggerEnter2D* function of the *SpaceCatController* script, ensures that the space cat dies if its lives equal 0.
	
	As with the previous code snippet, this action only happens when the game is actively running.

4.4. Helper Scripts

ControllerHelper

The *ControllerHelper* script contains functions related to controllers, as shown in Figures 4.7 and Table 4.5:

Figure 4.7. The flying hamburgers pursue the space cat via its position in the scene.

Table 4.5 The table shows the various *Helper*-related functions.

Function/Code Snippet	**Purpose**
public void ClampSpriteMovements(Transform transform) { Vector2 newPos = new() { x = Mathf.Clamp(transform.position.x, minimumBounds.x, maximumBounds.x), y = Mathf.Clamp(transform.position.y,	One of the game prototypes used an empty boundary object to keep the player within the screen; however, that method was less effective than using Unity's *Clamp* function, which gives a minimum and maximum float value [55]. The *Clamp* function defines a scene's x and y bounds while keeping a game object within these

minimumBounds.y, maximumBounds.y)
};
…
}
	coordinates [55], as shown in the code snippet on the left.

public void MoveAwayFromPlayer(Transform target, Transform transform, float speed, int lives)
{
…
if (range > minDistance)
{
direction.Normalize(); // Keeps the length of the direction to one, thus constant.
float angle = Mathf.Atan2(direction.y, direction.x) * Mathf.Rad2Deg; // This allows for smoother rotation.
transform.SetPositionAndRotation(Vector2.MoveTowards(transform.position, target.transform.position, speed * Time.deltaTime), Quaternion.Euler(Vector3.forward * angle));
}
else
{
transform.position = Vector2.MoveTowards(transform.position, target.position, -1 * speed * Time.deltaTime);
}
	In the *MoveAwayFromPlayer* function, *Math.Atan* [56] is applied to the x and y coordinates of the direction vectors to calculate their inverse tangent before being converted to degrees; this helps smooth the rotation of the enemy following the player [57].
	
	Depending on the minimum distance, the object should move away from the target, and the speed and delta time is multiplied by -1 [58].

public void FollowPlayer(NavMeshAgent agent, Transform target)
{
if (agent != null)
{
agent.SetDestination(target.position);
}
…
}
	Based on the A*pathfinding algorithm, which involves finding the shortest path from one point to another while avoiding obstacles [59], the FollowPlayer uses the *NavMeshPlus* [60] library to prevent the flying hamburgers following the player from overlapping with each other, as shown in Figure 4.7.

4.5. Gameplay Scripts

DealWithEffects

The *DealWithEffects* script handles the triggering of effects; static boolean flags indicate whether a specific effect should be active or inactive, as shown in Table 4.6:

Table 4.6 The *DealWithEffects* class contains public methods for turning effects on or off throughout the game. Static variables allow for sharing among more than one class instance [61].

Function/Code Snippet	**Purpose**

public void DisableSpriteEffects() => backGroundEffectsEnabled = false;
 	
public void EnableSpriteEffects() => backGroundEffectsEnabled = true;
	The following functions are responsible for turning the sprawling background effect on or off, allowing the user to take control of the UI and animation.

Timer

The *Timer* script handles the loading of the game script between levels, as shown in Table 4.7:

Table 4.7. The table shows the *StartCountdown* function.

Function/Code Snippet	**Purpose**

public void StartCountdown()
{
timeLeft -= Time.deltaTime;
if (timeLeft < 0)
{
timerFinished = true;
}
}
	In the *Timer* script, a *timeLeft* value determines how long an internal countdown needs to run. A boolean flag indicates whether the counter has finished.

(2000 words).

5. EVALUATION

5.1. Evaluation Factors and Questions

Evaluation Factors

Hungry Space engaged in a mix of manual testing, automated unit testing and playtesting, favouring honest feedback over using questionnaires, with the project's success determined by the factors shown in Table 5.1:

Table 5.1. The table lists the evaluation factors discussed above.

Factors	**Meaning**
The playability of the game.	The truth of the following statements measures the playability of the game: - The game loads without any issues. The game moves smoothly from one level to the next, offering players a brief break to catch their breath. - The user pauses the game and resumes it without any hiccups. -The user can exit the game at any time. - The game ends with a game-over text that informs the player that the game has ended.
The enjoyability of the game.	While very subjective, the enjoyability of the game is determined by: - How users describe the game. - Whether it is easy to play without adverse effects like bugs or unexpected crashes. -The level of simplicity or difficulty of the game. - The atmosphere and look/feel of the game. - The accessibility features of the games.
The aesthetics/ look and feel of the game.	The aesthetics are measured by: - The appearance of the sprites and animation. - The way the graphics enhance the game. - The colour palette and its consistency across the game. - The way effects inform the user about specific actions.
The UI.	The UI's success is measured by: - The clarity regarding the purpose of the buttons. - The size and readability of the fonts. - The UI presentation forms a logical pattern. - The accessibility of the UI.
The way the project aligns with the original project	The criteria below measure the game's success

goals.	against the project guidelines: - The completeness of the game (which means that it has more than one level and becomes progressively more difficult). - The way the project adheres to the principles of an arcade game (easy to play and pick up simple controls). - The project's evolution in terms of bug-fixing and sticking to the original work plan—did the project stay on track?

Evaluation Questions

During peer-graded reviews taking place in weeks 14 and 18 of the module, the evaluation form asked users the following questions, which were answerable to the following criteria as displayed in Figure 5.1:

1. disagree
2. partially disagree
3. neither agree nor disagree
4. partially agree
5. agree

Figure 5.1. The reviewer can answer the question by selecting the criteria in the screenshot above. Image by Coursera. Retrieved from https://www.coursera.org/learn/uol-cm3070-computer-science-final-project.

★ Is the game playable?
★ Would you feel compelled to come back and play the game again?
★ Does the game look good?

Responses to Evaluation Questions

Table 5.2 showcases how people responded to the evaluation questions in the peer-graded tasks, using pie charts as a visual aid using Python [57]. GitHub hosts all the scripts for the relevant pie charts.

Table 5.2. The table outlines how peer reviewers responded to the game.

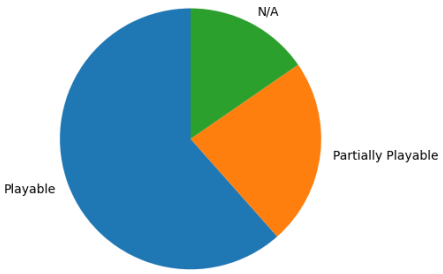
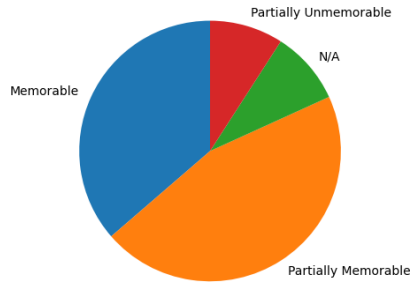
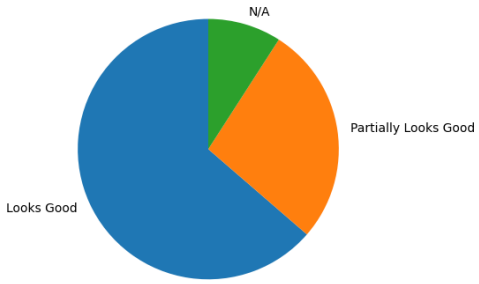
Pie Charts	**Result**
	Figure 5.2 reveals that eight out of eleven players felt the game was playable.

Figure 5.2: Most players felt the game was playable.	
 **Figure 5.3**: The majority of players felt the game was partially memorable.	As Figure 5.3 shows, five participants thought the game was only partially memorable, with four stating it was entirely so. One person had no opinion, while another thought the game was slightly unmemorable.
 **Figure 5.4**: Most players felt that the players looked good.	Figure 5.4 shows that seven players thought the game looked good, while three felt it partially did so. One person neither agreed nor disagreed.

5.2 Evaluation Graphs - Word Clouds

Using Python and an online version of JupyterLab, word clouds help better understand how playtesters feel about the game. GitHub hosts the scripts for these word clouds, focusing on adjectives to illustrate user feelings about the game.

Word Cloud - Entire Game

Figure 5.5 showcases the words utilised by playtesters to describe the game, with the overall impression suggesting that players found the game to be playable, pleasing and fun to play:

Figure 5.5. The word cloud is associated with the game in its entirety.

Word Cloud - Graphics

Figure 5.6. represents the sentiments utilised by playtesters to describe the game's graphics, which suggests that the game is cute and lovely to look at:

Figure 5.6. The word cloud is associated with the game's graphics.

Word Cloud - UI and Music

Figure 5.7. represents the sentiments utilised by playtesters to describe the game's UI and music, which, while decisive, did win favours with some players for its customisable and accessible features:

Figure 5.7. The word cloud is associated with the game's UI and menu.

Word Cloud - Gameplay

Figure 5.8. represents the adjectives utilised by playtesters to describe the game's gameplay, suggesting that – while needing more work – the game has managed to strike a balance between being fun to play and challenging at the same time:

Figure 5.8. The word cloud represents words associated with the game's gameplay.

5.3. Evaluation Results

Success Against Evaluation Factors

Table 5.2. The table evaluates the success factors in Table 5.1.

Factors	**Issues**	**Response / Success**
Playability of the game.	Playtesting uncovered numerous issues concerning the gameplay with the following bugs, with some of them listed below: - The collision hitboxes were imprecise, making the collisions seem unnatural. - The lives and score displays needed to be more consistent. - The game froze after hitting 'pause'. - Users wanted some time to prepare for the game between level transitions.	Much work and effort prioritised fixing gameplay-related bugs, with the most important ones fixed and retested before a new release. As shown in Figure 5.2, the game's playability has improved dramatically since its beginning, thanks to honest feedback from playtesters. However, there is no denying that, at its current stage, it still needs more work to move away from its more prototype-like impression.
Enjoyability of the game.	Generally, the consensus among players was that the game was fun, although some commented on the following: - The game's simplicity, with some stating it was too simple. Others felt the game could be made more accessible, with the cat's speed needing to be slower or the sprite effects turned off.	As this a subjective area to evaluate, it feels insincere to say that the responses to comments rendered the game better, as people can be fickle; however, the word cloud in Figure 5.5 suggests that the game is fun to play with enough of a challenge to appeal to seasoned players, but ones looking for a more relaxing game also getting some benefit out of it.

	- The game needed more levels and versatility, while others felt only a few scenes required. - Later feedback suggests that the game balances between being accessible and challenging.	
Aesthetics of the game.	Overall, comments regarding the aesthetics came across as positive, with users describing the game as ‘cute’. However, some playtesters requested the following features: - A transition between loading scenes. - A sprite effect when the player dies or gets injured. - A sprite effect when the player eats a bug. - A shader outline around the enemy sprites to help differentiate them from the player and pickups. - Resizing the sprites.	A fair amount of work went into improving the game's aesthetics, which—according to Figure 5.3—helped enhance its look and feel.
The UI.	Playtesters raised the following issues regarding the UI: - The buttons needed to be bigger/readable. - The game instructions required to be more specific. - The purpose of the buttons needed to be clarified. - The UI needed more accessibility features, such as allowing users to turn off flashing and background effects. - Other comments included the statement that the buttons needed to be more standardised.	To my surprise, the UI felt like one of the more controversial aspects of the game, with lots of back and forth in terms of what users liked/wanted to have. While the current menu seems pleasing to most, others will feel it needs more work. Thus, this area's overall success feels mixed. More data is required to provide a more conclusive answer, but the consensus—as shown in Figure 5.4— is that the menu is a partial success.
The alignment with the original project goals.	In terms of development, due to starting the design and development stages early, the	While the two modes depart from traditional arcade games, they still follow the pattern

	project was: - Completed ahead of time with much room left for playtesting and bug fixing. - Planned to incorporate arcade elements such as easy controls and increasingly tricky levels from the get-go. - Took user tastes and accessibility into consideration by providing two modes, allowing players to choose the difficulty level. - Players were allowed to adjust their speed, making it more difficult or accessible according to their wishes.	of—whatever the mode—each level having increasingly more challenging gameplay. Additionally, the game's development went smoothly, with enough time to iron out bugs and refactor the report as necessary.

(1499 words).

6. CONCLUSION

Overall, *Hungry Space Cat* has taught me more about creating games, especially ones with sprites and multiple levels.

6.1. Reflection on Issues

Lack of Experience in Game Art Design

One drawback was my lack of experience creating sprites and developing specific artistic visions. My lack of knowledge of sprite-related software caused several issues that prevented me from fulfilling some of the project's initial ideas.

Consequently, Hungry Space Cat showed that relying on assets created by others not only hindered the development of the initial project—that of an astronaut cat chasing bugs within a cat-shaped aircraft—but also led to changes due to the inability to make these artistic elements come to life, as specific techniques either proved ineffective or required more time than the project's development allowed.

Later, problems emerged when trying to create shaders that outlined a sprite. The lack of knowledge in this area resulted in using a simple shader made by someone else [34].

Player Expectations

After receiving feedback from playtesters, it became apparent that some players had particular expectations regarding how the game should look. An initial vision of using a Russian roulette approach to the game – where a player would click on a button to select one of three games – did not impress players, and further confusion about controls necessitated the creation of an instructions-related scene.

The menu also underwent numerous iterations due to players' complaints about its look and feel. It is the most contentious area of the game, and playtesters are still making suggestions about its appearance.

Lack of Experience in Games Development

As an unseasoned games developer, it took a lot of work to design a game from start to finish, having previously only worked on such projects in a group; however, focusing on more straightforward features first ensured that the game got worked on at all.

However, as feared, some feedback trickling through from the play sessions was that the game was too simple, and the preliminary report also contained some comments on the game requiring more advanced features.

6.2. Future Work

There are no current plans to continue *Hungry Space Cat* beyond this project's scope, but for any future work, the following points are worth considering (some of which helped finish this project on time):

- ★ Embrace playtesting early and often, though with the caveat that paid testers are more reliable than unpaid ones.

- ★ Document changes and understand the reason for these modifications.
- ★ Work on the project incrementally, with targeted breaks in between to avoid burnout.
- ★ Prioritise early deadlines, as these help deal with issues head-on rather than panicking at the last minute.
- ★ Do not be afraid to challenge unclear feedback and ask for clarification – this will help with the project and make the reviewer/tester feel more valued and trusted.
- ★ Give credit where it is due and ensure that everything is well-documented so you can quickly return to the source when needed.
- ★ Accept that, no matter how much you have worked on the project, there will always be improvements that can (and should) be made.
- ★ Understand that writing games is not just an art but an ever-growing, living skill set that needs time to develop/grow.

(529 words).

7. REFERENCES

[1] a2a5. 2023. Cat in space station (00184). (February 2023). Retrieved May 18, 2024 from https://www.deviantart.com/a2a5/art/Cat-in-space-station-00184-950887557

[2] Jamey Pittman. 2015. The Pac-Man Dossier. (August 2015). Retrieved February 12, 2024 from https://pacman.holenet.info

[3] Retro Gamer. 2009. The Making of Asteroids. *Retro Gamer* 68, (October 2009), 24-29.

[4] bevouliin. 2023. Bevouliin - Selling 2D Game Assets for Game Developers. (October 2023). Retrieved March 23, 2024 from https://bevouliin.com

[5] screamingbrainstudios. 2022. Seamless Space Backgrounds. (March 2022). Retrieved March 23, 2024 from https://screamingbrainstudios.itch.io/seamless-space-backgrounds

[6] kububbis. 2021. Kawaii Ghost Sprite Pack Neko Edition. (November 2021). Retrieved March 23, 2024 from https://kububbis.itch.io/kawaii-ghost-sprite-pack-neko-edition

[7] Scott Ellis. 2023. Unlocking the Secrets of Classic Arcade Games: What UX Tricks Can We Learn?. (November 2023). Retrieved July 27, 2024 from https://www.linkedin.com/pulse/unlocking-secrets-classic-arcade-games-what-ux-tricks-dba-llm/

[8] Retro Gamer. 2016. *Retro Gamer Book of Arcade Classics* (2nd. ed.). Imagine Publishing, Bournemouth, Dorset.

[9] Carl Therrien. 2017.“To Get Help, Please Press X” The Rise of the Assistance Paradigm in Video Game Design. In *Proceedings of DiGRA 2011 Conference: Think Design Play*, September 14-17, 2011, Utrecht School of the Arts, Hilversum, The Netherlands, 8 pages. https://dl.digra.org/index.php/dl/article/view/527/527

[10] Yuexian Gao and al. 2022. Nature of arcade games. In *Entertainment Computing* 41, Article 100469 (March 2022), 11 pages. https://doi.org/10.1016/j.entcom.2021.100469

[11] Katrin Becker and James R. Parker. 2005. All I Ever Needed to Know About Programming, I Learned From Re-writing Classic Arcade Games. In *Future Play, The International Conference on the Future of Game Design and Technology*, October 13-15, 2005, Michigan State University, East Lansing, Michigan, USA, 1-7. http://hdl.handle.net/1880/46707

[12] Renato Fernando dos Santos and al. 2021. Pac-Man is Overkill. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 24 - January 24, 2021, Las Vegas, NV, USA, 11652-11657. https://ieeexplore.ieee.org/document/9341274

[13] Adelina Moura. 2015. Using arcade games to engage students in the learning of foreign and mother languages. In *EAI Endorsed Transactions on e-Learning* 2, Issue 5 (March 2015), 1-14. https://eudl.eu/doi/10.4108/el.2.5.e2

[14] Nicolas Esposito. 2005. How Video Game History Shows Us Why Video Game Nostalgia Is So Important Now. In *Playing the Past*, March 18 - 19, 2005, Gainesville, Florida, USA, 11 pages. http://nicolasesposito.fr/publications_fichiers/esposito2005history.pdf

[15] Auroch Digital. 2022. Neurodiversity and the Games Industry (November 2022). Retrieved July 27, 2024 from https://www.aurochdigital.com/blog/2022/11/28/neurodiversity-and-the-games-industry

[16] Maximiliano Miranda, and al. 2017. Pac-Man or Pac-Bot? Exploring subjective perception of players' humanity in Ms. Pac-Man. In *Conference of the Spanish Association for Videogames Sciences*, June 30, 2017, Barcelona, Spain, 163-175. https://ceur-ws.org/Vol-1957/CoSeCiVi17_paper_17.pdf

[17] Nicolas. 2011. Kanji Card – 食. (April 2011). Retrieved February 12, 2024 from https://nihongoichiban.com/2011/04/10/jlpt-kanji-食

[18] Nicolas. 2011. Kanji Card – 口 – kuchi. (April 2011). Retrieved February 12, 2024 from https://nihongoichiban.com/2011/04/09/jlpt-kanji-口

[19] Computeach. Paper - Pacman Background. Retrieved from https://www.cleanpng.com/png-paper-pac-man-printing-pizza-food-bar-man-4033744/.

[20] Ari Feldman. 2001. *Designing Arcade Computer Game Graphics*. Wordware Publishing Inc., Plano, Texas.

[21] Philipp Rohlfshagen and al. 2017. Pac-Man Conquers Academia: Two Decades of Research Using a Classic Arcade Game. In *IEEE Transactions on Games* 10, 3 (December 2017), 233 - 256. https://ieeexplore.ieee.org/document/8207594

[22] Chad Birch. 2010. Understanding Pac-Man Ghost Behavior. (December 2010). Retrieved April 1, 2024 from https://gameinternals.com/understanding-pac-man-ghost-behavior

[23] Jason Brown. 2022. All Pac-Man Ghost Names and What They Do. (December 2023). Retrieved May 12, 2024 from https://retrododo.com/all-pac-man-ghosts

[24] Arcade Blogger. 2018. Atari Asteroids: Creating a Vector Arcade Classic. (March 2021). Retrieved May 11, 2024 from https://arcadeblogger.com/2018/10/24/atari-asteroids-creating-a-vector-arcade-classic

[25] Carlo Fabricatore. 2007. Gameplay and game mechanics design: a key to quality in videogames. In OECD Expert Meeting on Videogames and Education, October 29-31, 2007, ENLACES (MINEDUC Chile), Santiago de Chile, Chile, 18 pages. https://www.oecd.org/education/ceri/39414829.pdf

[26] LaunchBox. Games Database. CAT TRAX. Retrieved from https://gamesdb.launchbox-app.com/games/details/76418-cat-trax.

[27] atariprotos. Cat Trax. Retrieved from https://www.atariprotos.com/2600/software/cattrax/cattrax.htm.

[28] TrekMD. 2016. Cat Trax. (June 2016). Retrieved March 30, 2024 from https://www.retrovideogamer.co.uk/cattrax2600

[29] Namco Wiki. Mappy. Retrieved from https://namco.fandom.com/wiki/Mappy.

[30] RetroGames. Mappy - Nintendo NES system. Retrieved from https://www.retrogames.cz/play_008-NES.php.

[31] Divok. 2023. Pac Cat: retro cat. (August 2023). Retrieved February 17, 2024 from https://play.google.com/store/apps/details?id=com.Divok.PacCat&hl=en&gl=US

[32] HedonisticOpportunist. 2024. SPACE CATS - A GAME WEB APPLICATION. (January 2024).

Retrieved May 12, 2024 from https://github.com/HedonisticOpportunist/Space-Cats

[33] bdatg2. 2013. Simplicity. (December 2013). Retrieved February 20, 2024 from https://vgprobs.wordpress.com/simplicity

[34] Ronja. 2020. Sprite Outlines. (July 2020). Retrieved June 17, 2024 from https://www.ronja-tutorials.com/post/049-sprite-outline

[35] SaintBob. 2021. Free basic camera fade in script. (June 2021). (Retrieved May 1, 2024 from https://forum.unity.com/threads/free-basic-camera-fade-in-script.509423/.

[36] Unity. Unity Real-Time Development Platform | 3D, 2D, VR & AR Engine. Retrieved from https://unity.com.

[37] Microsoft. C# documentation. Retrieved from https://learn.microsoft.com/en-us/dotnet/csharp/.

[38] icons8. Cat symbols and icons in Cute Color Style, PNG, SVG Edition. Retrieved from https://icons8.com/icon/set/cat/dusk.

[39] kububbis. Kawaii Ghost Sprite Pack Neko Edition. Retrieved from https://kububbis.itch.io/kawaii-ghost-sprite-pack-neko-edition.

[40] Anyrgb. 8bit Color, Nyan Cat, pusheen, black Cat, pixel Art, cuteness, Kitten, pet, Cat, organ | Anyrgb. Retrieved from https://www.anyrgb.com/en-clipart-22sxz.

[41] kububbis. 2022. Kawaii UFO Sprite Pack. (January 2022). Retrieved March 23, 2024 from https://kububbis.itch.io/kawaii-ufo-sprite-pack

[42] Cuzco. 2010. Space background. Retrieved from https://opengameart.org/content/space-background.

[43] Kenny. 2023. Planets. Retrieved from https://kenney.nl/assets/planets.

[44] zigurous. 2023. AnimatedSprite. (November 2023). Retrieved April 2, 2024 from https://github.com/zigurous/unity-pacman-tutorial/commits/main/Assets/Scripts/AnimatedSprite.cs

[45] Unity Documentation. Sprite Renderer. Retrieved from https://docs.unity3d.com/Manual/class-SpriteRenderer.html.

[46] BMo. 2020. 6 Minute PAUSE MENU Unity Tutorial. Video. (May 2020). Retrieved April 28, 2024 from https://www.youtube.com/watch?v=9dYDBomQpBQ

[47] Wikipedia. Parallax scrolling. Retrieved from https://en.wikipedia.org/wiki/Parallax_scrolling.

[48] Gary Pettie. 2021. 17. Scrolling Background. (July 2021). Retrieved April 4, 2024 from https://gitlab.com/GameDevTV/unity2d-v3/laser-defender/-/blob/master/Assets/Scripts/SpriteScroller.cs

[49] educative. What is "Time.deltaTime" in Unity?. Retrieved from https://www.educative.io/answers/what-is-timedeltatime-in-unity.

[50] Joseph Spinelli. 2018. MVC Overview. (December 2018). Retrieved April 4, 2024 from https://medium.com/@joespinelli_6190/mvc-model-view-controller-ef878e2fd6f5

[51] Unity Documentation. MonoBehaviour.OnTriggerExit2D(Collider2D). Retrieved from

https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerExit2D.html.

[52] Unity Documentation. MonoBehaviour.OnTriggerEnter2D(Collider2D). Retrieved from https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerEnter2D.html.

[53] Rick Davidson. 2021. additional bug fix for coin pickup. (November 2021). Retrieved April 4, 2024 from https://gitlab.com/GameDevTV/unity2d-v3/tilevania/-/blob/master/Assets/Scripts/CoinPickup.cs

[54] Rick Davidson. 2021. 37 Scene Persist. (September 2021). Retrieved April 4, 2024 from https://gitlab.com/GameDevTV/unity2d-v3/tilevania/-/blob/master/Assets/Scripts/GameSession.cs

[55] Unity Manual. Mathf.Clamp. Retrieved from https://docs.unity3d.com/ScriptReference/Mathf.Clamp.html.

[56] Unity Manual. Mathf.Atan. Retrieved from https://docs.unity3d.com/ScriptReference/Mathf.Atan.html.

[57] MoreBBlakeyyy. 2022. Unity simple 2D Enemy AI Follow Tutorial. Video. (January 2022). Retrieved March 30, 2024 from https://www.youtube.com/watch?v=2SXa10ILJms

[58] maccabbe. 2016. GameObject1 move away when GameObject2 gets close. (February 2016). Retrieved June 17, 2024 from https://discussions.unity.com/t/gameobject1-move-away-when-gameobject2-gets-close/158522

[59] Matthew Jones. 2021. Pathfinding Algorithms: Part 1 — A* (July 2021). Retrieved July 27, 2024 from https://medium.com/geekculture/pathfinding-algorithms-part-1-a-2d9815ae1c64

[60] GitHub. NavMeshPlus. Retrieved from https://github.com/h8man/NavMeshPlus

[61] Full Stack Content Creator. 2023. Demystifying Static in C#: Understanding How it Works. (May 2023). Retrieved July 1, 2024 from https://medium.com/@javvadirupasri8/demystifying-static-in-c-understanding-how-it-works-d7b47a1acb32

[62] w3schools. Matplotlib Pie Charts. Retrieved from https://www.w3schools.com/python/matplotlib_pie_charts.asp.