# Hungry Space Cat

A Game Based on the 'Project Idea Title 1: Arcade Game' Template

Anita Pal (20020826)
ap383@student.london.ac.uk

## Contents



## List of Figures

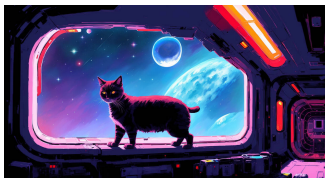

**Figure 1.** The game is cat-themed, as cats are fun, especially in space! [1].

## Acknowledgments

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. Absolutely fantastic.

. Link to the public GitHub repository. ♥

# 1 INTRODUCTION

## 1.1 Project Template

*Hungry Space Cat* builds on the 'Project Idea Title 1: Arcade Game' template. The following chapter describes the project's main objectives, motivation and prospective audience.

## 1.2 Project Description

*Hungry Space Cat* is a 2D feline-inspired PC arcade game, as shown in Figure 1. Regarding gameplay, it takes inspiration from Namco's *Pac-Man*, a 1980 maze action game [17], and - in its galaxy-themed setting and ability to shoot at enemies - *Asteroids*, a 1976 space shooter by Atari [18].

However, unlike these two games, it includes a more modernised look and feel. Moreover, it emphasises customisability and accessibility to stand out from other arcade games by being available to diverse players, such as those on the spectrum.

The game features a hungry astronaut cat pursuing purple space bugs. To keep chasing the adorable insects, the cat must simultaneously evade UFOs, flying hamburgers, asteroids, ghost dolls, spaceships, snails, or a combination of these opponents. It uses arrow keys as controls to simulate an arcade game experience. ESC is used for pausing, and the space bar activates player shooting. These two keyboard keys are the only other controls employed in the game.

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!) [2]. The player has to advance through increasingly challenging levels to eat all the floating bugs without losing the cat's nine lives. The player can activate a laser beam that destroys their enemies, but they may also keep it turned off if they enjoy a less demanding

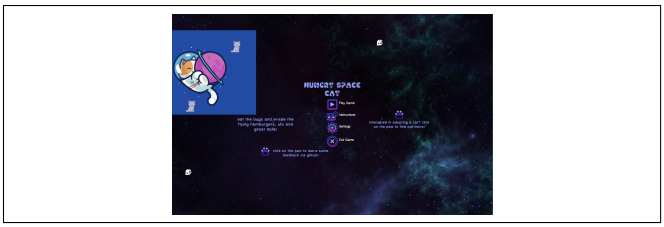


**Figure 2.** The menu's design emphasises vivid colours and has a feline touch.

game. The laser beam is just one of the features that the user can adjust according to their priorities.

Therefore, to allow players to choose the difficulty of the game they want to play, *Hungry Space Cat* has two modes: one that aims to be easy and relaxing, while the other - following the trajectory of arcade games becoming progressively complex over time [9] - poses more of a challenge, adding impulse to the player's movements and containing numerous levels. Both modes promote learnability for users to grow familiar with the game's mechanics [19] while considering preferences such as slower speed or the ability to shoot at enemies.

The more difficult mode mimics the deceptive simplicity of arcade games [9] while making the game fun and something users would want to return to [9, 20]. Meanwhile, the simple mode offers less seasoned players the ability to enjoy the game without having too many hurdles placed in their direction. It is also shorter than the challenging mode, thus requiring less effort and time. Both modes address the game's aim to be playable and fun without requiring much investment or time for the player to understand its plot [9].

The background of *Hungry Space Cat* engages a space theme to pay tribute to interstellar-related games such as *Space Invaders* and *Asteroids*. Similarly to these titles, realism is not at the heart of the game. Instead, its design and gameplay use silly/cute elements to appeal to a broader audience via radiant colours, happy music and relaxed gameplay to heighten positive emotions [19].

Befitting the game's dedication to cute design, the menu scene - as depicted in Figure 2 - has an astronaut cat image [10] and a space-themed background [21]. A white ghost cat [22] further enhances the menu's design. The overall goal for the UI design is to be user-friendly and easy to understand [19].

## 1.3 Project Motivation

### 1.3.1 Academic/Personal Motivation.

. Arcade games - one of the earliest forms of electronic entertainment [20] - offer aspiring programmers a glimpse of the magic behind creating them [23]. *Pac-Man* uses pathfinding and chasing/tracking algorithms [20], e.g. the Pursuit-Evasion Game (PEG), which focuses on pursuers catching an evader quickly [24].

Not only do students benefit from learning about such algorithms [23], but researchers have proven that Pac-Man merely needs two ghosts to capture the player [24]. Furthermore, titles like *Asteroids* use collision detection and distance calculation, enabling researchers and students to explore these topics visually [23]. Lastly, arcade games are practical tools for teaching children to learn new languages because they encourage them to interact with the material [25].

*Hungry Space Cat* addresses my affection for cats, allowing me to hone my coding skills and create games accessible to players who love felines.

### 1.3.2 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 mixed range of players [2]. Consequently, aligning with the fact that video games appeal to people of various ages [25], the game is suitable for players who:

- Enjoy a game without requiring instruction manuals [26].
- Wish to play a game with simple rules [19] and engaging gameplay [19, 20, 25].
- 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 positively influences their emotions [25], and makes them happy [19].
- A neurodiverse audience, as video games provide a continuous activity allowing them to grow and make mistakes [27].

### 1.3.3 Objectives.

. *Hungry Space Cat* aims to be/have the following criteria:

- Be intuitive to grasp, as arcade games are easy to play and pick up [9].
- Elicit cheerful emotions via its design and gameplay [19].
- Simple, enjoyable visuals [19] while defying the concept of fast-paced gameplay [25] by offering the player a choice between two modes, thus following the principle of accessibility of games such as *Asteroids* with adjustable levels [19].
- Provide accessibility/learnability features for players by using simple instructions and straightforward controls [19].
- Serve to motivate players to explore other arcade games using nostalgic design elements [26].
- Encourage people to adopt mousers by having them engage with a cat character [25].

. (922 words.)

# 2 LITERATURE REVIEW

The following chapter explores accessibility in game design before discussing and evaluating previous work.

## 2.1 Accessibility and Design

### 2.1.1 Why Accessibility?

. User design is essential as games have become popular with countless people, including those with disabilities [28]. Games enrich these individuals' lives by giving them a platform to feel empowered by a game's narrative [28]. Often, video games are the only venue available to people with disabilities that permits them to stand on an equal footing with their non-disabled peers [28].

At the same time, games function as entertainment, stress relief, and contributors to feelings of well-being [29], further highlighting the importance of considering all kinds of users when designing them. The increasing seniority of players is another fact that touches upon the issue of accessibility, as with an advanced age come specific health issues, such as decreased vision, mobility, and hearing [28].

2.6 billion players worldwide play digital games, and this popularity has prompted calls for titles to be more inclusive and address the needs of their ever-growing audience [28].

Making games available to players with special needs would help fill a gap in the market, aiding not only the players themselves but the wider community, such as developers who could learn to understand how a game could be elevated beyond an all-purpose design while promoting a more fruitful discussion on the manifold representations of disability [28]. Such talks would prove eye-opening in a society obsessed with good looks.

Beyond a commercial need for accessible games, organisations such as the European Union with the *European Accessibility Act* instruct software to have basic accessibility requirements [28]. In the US, games need components that facilitate accessibility regarding communication, presentation and controls [28].

Furthermore, social justice forums argue that people with disabilities are citizens who have the same right to consume entertainment as everyone else [28]. The UN Convention similarly mandates this point of view with the *Rights of Persons with Disabilities Act* [28].

### 2.1.2 Current Challenges.

. Despite being aware of accessibility, modern game design frameworks do not address the complex needs of disabled players, including the changes and customisations they make to a particular game to be able to play it [28].

The issue with adapting games for a disabled audience is that many accessible games were made with a particular group of players in mind. Thus, it is difficult for other game developers to adapt these specific techniques to their games, as they cannot easily be generalised [28].

Moreover, the problem with the approach above is that many developers adopt a singular quality of a disability, ignoring the nuanced complexity of users who come with varying degrees of disability or suffer from multiple at once [28].

Additionally, game developers are realising that players with neurodiversity may not require simplified controls but a selective approach to how their emotions are aroused, as heightened ones, such as fear or surprise, could stop them from playing a game altogether [28].

Another fact that renders modern frameworks problematic is that they separate commercial concerns from those with accessibility, overlooking the fact that disabled players enjoy and wish to participate in popular games [28].

Also, while guidelines exist that attempt to streamline the accessibility of video games, a checklist - while offering reassurance - cannot help when a game's feature fails to meet its criteria, especially when dealing with less minimalistic aspects [28].

Guidelines, likewise, suffer from being limited, mainly when it comes to newer technology, which not only interacts differently with a disability but also requires an update of the guidelines themselves [28]. Considering the size of the guidelines available, this poses a problem for the people who write these guidelines and the developers themselves, who may feel overwhelmed by the sheer amount of what is present [28].

Lastly, guidelines cannot predict how an individual player will react to a particular accessibility feature [28], as there may be a mismatch between the developer's vision and the player's expectations.

## 2.2 Arcade Games

### 2.2.1 Arcade Game Design.

. Arcade games feature a simplistic design but challenging-to-master gameplay [9]. Throughout their history, not only did the unsophisticated nature of early arcade games allow the player to grasp all of a game's intricacies, but it made the player feel accomplished and eager to replay them to encounter new obstacles to solve [9, 30]. 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 [30].

Taking cues from *Pac-Man*, with its diverse audience [2] and continuing appeal to people of all ages [25], arcade games aim to keep their players engaged in the story and want to return for more by providing them with gradually challenging levels [9, 20].

*Pac-Man* uses dazzling colours and cheerful gameplay to make players feel safe and upbeat [19]. The game's design elements - such as character and level aspects - are intended

**Figure 3.** According to a popular urban myth, a pizza missing a piece influenced the shape of Pac-Man [2], as shown above [3].


**Figure 4.** Ed Logg poses next to *Gold Asteroids*, created to celebrate building 50,000 units [4].

to be undemanding, with the goals and game mechanics similarly easy to gauge [9, 20].

#### 2.2.2 Pac-Man.

. A 2008 report revealed that 94 % of US consumers recognise *Pac-Man*, beating *Mario* in popularity [2]. According to its creator, Toru Iwatani – a game designer without formal training [2], the gentle gameplay of *Pac-Man* was intentional as, in the late 1970s, arcade centres only contained violent games focused on killing aliens [17]. He felt these arcade games were playgrounds for boys, leading him to develop a game for women and couples [17] with beaming and joy-inspiring graphics [19].

A Japanese fairytale with a demon-eating creature protecting children from monsters inspired the creation of *Pac-Man's* prototype [17]. In *Pac-Man*, the monsters from the fairytale became four ghosts [9], each with a unique personality [17]. Furthermore, Toru Iwatani used the Kanji for eating – *taberu* 食 [31] - as a premise for the game and the one for mouth - *kuchi* □ [32]- with its square shape as the basis for the character's design [2, 17].

Iwatani insisted on the clearness of *Pac-Man*'s design—a yellow disc with a mouth resembling a pizza slice, as shown in Figure 3—to streamline the game and keep its gameplay simple without the player requiring a manual [25, 33]. Despite the simplicity, while playing the game, the player's strategy is crucial to winning it [33].

Further, Iwatani restricted the gameplay to a maze in which players move in the following directions: up, down, left, and right [2]. As a maze game, *Pac-Man* contains one screen or playfield with little to no scrolling; multiple game levels keep the player's interest alive [33].

#### 2.2.3 Evaluation.

. Due to its coherent controls and easy-to-understand story, *Pac-Man* is a popular game with many players, including those new to arcade games. *Pac-Man's* graphics not only offer escapism due to their unsophistication but are easy on the eyes because they do not contain flashy graphics or involve having too many events concurrently on the screen.

Beyond that, *Pac-Man* is uncomplicated to get into but becomes challenging over time due to the multiple ghost personalities and their speed changes [17]. As such, it offers more seasoned players the ability to advance through levels while less skilled ones feel compelled to return for more due to its addictive gameplay. Its plot does not provoke negative emotions or come with nasty surprises, making it soothing for neurodiverse players.

While *Pac-Man* does not address more advanced accessibility features, such as those for people with upper limb disabilities who cannot use two or more buttons simultaneously [28], it provides visual and acoustic feedback that allows players with visual impairments, for example, to comprehend what is going on within the game without having to resort to assistive technology.

However, one drawback is that its graphics are very colourful, rendering the number of enemies daunting for people with cognitive issues or those sensitive to bright tones.

#### 2.2.4 Asteroids.

. In December 1979 [18], Atari released the shoot-em-up *Asteroids* [4]. In line with its characteristics as a shooter arcade game [18], the player controls a spaceship on a single screen to evade asteroids [4, 18]. The game has an easy-to-learn but difficult-to-master gameplay that keeps people returning to it to beat it [4].

Beyond its financial success, *Asteroids* helped arcade games become popular among players from various walks of life, including professionals in their 30s and 40s who played it during their lunch break [4]. The game was created by Ed Logg, depicted in Figure 4. *Asteroids* was his response to an unsuccessful game where players tried to shoot asteroids [4, 18]. Describing the game as dull, Ed suggested an alternative: players should blow up the asteroids instead [4].

During a conversation with Lyle Rains, his boss at Atari, Ed stated that in contrast to a successful title such as *Space Invaders*—a static shooter [33] with one-directional controls that only lets the player move left and right—he envisioned a

scrolling game that allowed for two-directional movements, creating player satisfaction [4] by giving them more free-range movement [33].

Despite no design processes in place, Ed sketched many versions of the ship [18]. As noted by Mark Cerny, a colleague of Ed's at Atari, the secret to his success was that he planned the game carefully, developing features in the correct order rather than focusing on complex algorithms [18].

Ed also engaged in feedback from two field playing testing sessions to fine-tune his game [4], making him one of the first game designers to realise the importance of gameplay. He also ensured that the game was accessible, as it had a straightforward interface with understandable instructions and allowed players to adjust the game's difficulty level to suit their needs [19].

#### 2.2.5 Evaluation.

. What makes *Asteroids* stand out compared to other classic arcade games is the creator's attention to detail in game design and how he approached the entire game's development process. Not only did he listen to player feedback, but he emphasised the importance of having multiple rounds of playtesting. For aspiring game developers, such an approach should be an inspiration and something they should take to heart when designing a game.

Even more so than *Pac-Man*, *Asteroids* offers clean graphics that immerse a player in the game without worrying about understanding the storyline or concept. The controls, while multiple, are straightforward to grasp and do not require too much time to get used to.

Additionally, the black-and-white graphics are colourblind friendly while respecting players—such as people on the spectrum—who may prefer a less involved look to their game. The player also receives acoustic feedback with distinguishing sounds that help them understand when an enemy is approaching or the spaceship has been damaged.

### 2.3 Arcade-Based Cat Games

#### 2.3.1 Cat Trax.

. *Cat Trax* (Figure 5), released in 1982 [5] or 1983 [34], is a clone of *Pac-Man*, featuring a cat on the run from three canines [5, 34]. 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 [5, 34].

Once the potion wears off, the dogs start chasing the game character again [5, 34]. *Cat Trax* was developed for the Aradia 2001 [35] – an obscure home game system released in 1982 by UA.Ltd [5].

#### 2.3.2 Evaluation.

**Figure 5.** An image of the arcade game *Cat Trax* [5], showing the dogs and playable cat character.

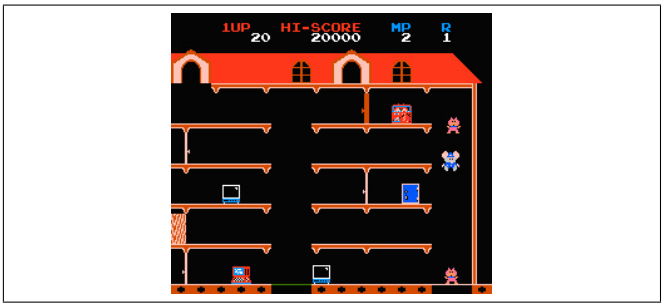


**Figure 6.** A screenshot of the arcade game *Mappy*, showing the trampolines inside the cat mansion [6].

. *Cat Trax* has a fun take on *Pac-Man*; however, it remains a clone with little to set it apart regarding creativity or originality, making it unenticing for players looking for something special or new. Therefore, it remains unlikely that anyone beyond a fascination for *Pac-Man* copies would revisit the game more than once.

Furthermore, what makes it less accessible than *Pac-Man* is that the cat and dog enemies are less suited to the maze than in the original game. While the circular design of Pac-Man makes it easy to distinguish the player from the ghosts, in *Cat Trex*, the cat and dog characters overlap, making the game confusing for players who need help differentiating these visuals.

The game feels disjointed because the appearance of a dog-catching trucker adds more colourful elements to the screen, thus creating more 'noise'. Players with vision impairment may find these features overwhelming.

#### 2.3.3 Mappy.

. Released by Namco in 1983, *Mappy* (マッピー) is a side-scrolling maze game featuring cartoon-inspired cats and mice [6, 36]. It ran on the Super Pac-Man hardware modified to support horizontal side-scrolling [6, 36].

In the game, shown in Figure 6, the player guides the police mouse, Mappy – a term derived from the Japanese nickname *mappo* – through a cat mansion to find stolen goods [6, 36]. To survive, Mappy must avoid the mansion's cats and traverse the building via trampolines [6, 36].

#### 2.3.4 Evaluation.

**Figure 7.** An image of the indie game *Pac Cat* [7] which involves a cat in a maze.

. *Mappy* is more involved and complex than *Cat Trax*, but its story remains simple. The enhanced gameplay does not interfere with the game's objective; however, it feels like the player requires more skill to play the game, which could deter those who prefer a less challenging piece of entertainment, especially since there is no option to opt for a simpler version.

The game's visuals also involve flashing objects and more things happening on screen, which could discourage people with epilepsy or impaired cognitive skills from wanting to continue to play. Moreover, the fact that a lot happens at once on the screen could make it hectic and faster-paced for those who prefer the gentler gameplay of *Pac-Man*.

#### 2.3.5 Pac Cat.

. *Pac Cat* by Divok, displayed in Figure 7, is an indie game with pixel graphics for mobile devices. In it, a cat has to eat all the points to advance through the levels while running from bulldogs [7].

#### 2.3.6 Evaluation.

. Reviews of the game describe it as cute but buggy, with the controls not working and the game being challenging from the get-go [7]. Consequently, *Pac-Cat* defeats the purpose that arcade game creators had in mind: for their games to be simple but become more complex with increasing levels [9].

Besides, challenging controls might make the game inaccessible or difficult to an older audience, especially those who struggle with mobile devices and may need more time to adjust to the controls. Additionally, the game does not offer users any options to make it customisable according to their needs.

### 2.4 Space Cats - Web-Based Cat Games

### 2.5 Space Cats

. *Space Cats* is a web game application where users can play games and view interactive art, as shown in Figure 8. The application aims to bring together people who enjoy cute games.

#### 2.5.1 Evaluation.

**Figure 8.** The user can play one of the two games in *Space Cats* [8].

. Playtesters for the project described the gameplay as too easy, which does not fit the concept of arcade games being challenging to master [9]. Using a web browser affected the games' graphics in size/resolution, stopping a significant portion of the intended audience from engaging with the game in the first place. Consequently, the game did not fulfil its objective of being playable to a feline-obsessed audience.

. (2399 words.)

## 3 GAME DESIGN

The following chapter explores the design aspects of *Hungry Space Cat*, including its core mechanics, game components, and the two modes from which players can choose to play the game.

### 3.1 Domain and Users

. *Hungry Space Cat*'s domain is arcade games. While it focuses on offering customisable features, the intended audience is all-embracing, aiming to entertain players from contrasting backgrounds with varying game-playing skills. Therefore, the game provides the traditional trajectory of its levels, which are more complex over time for those who enjoy overcoming obstacles, while providing an accessible and relaxing mode for players who do not.

### 3.2 Core Mechanics

With each new level, the type of enemy changes, making the game less predictable and more versatile. Players can choose between two modes that influence how the space cat moves: the more difficult mode employs momentum, while the easier one does not include it.

Two modes make the game playable and accessible to many players, but it also offers customisation. In other words, players can choose between the difficulty level and effects on display while adjusting the speed, audio volume, and ability to shoot at enemies.

#### 3.2.1 Player Movement.

. In the challenging mode, the player moves up, down, left, or right using momentum. In the more accessible mode, the player stops and starts instantaneously. Consequently, the player moves faster in that mode, while in the other mode, they are slower. The left, up, down or right arrow keys control the player's movement in both modes.

**Figure 9.** 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 and pick up [9].

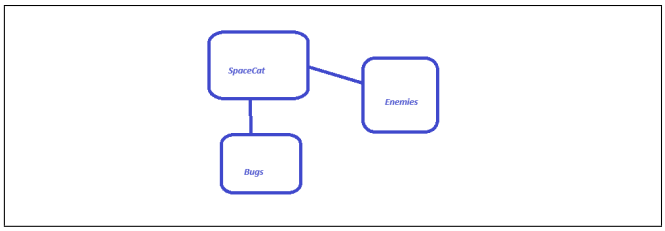


**Figure 10.** The three-game objects form the basis of the game, allowing the player to be a space cat that eats bugs and tries to avoid enemies.

Triggered by the player input, the cat navigates within the screen's boundaries, restricting its movements like the maze confides the player in *Pac-Man* [2]. Moreover, depending on the character's direction, the space cat's sprite flips to the left or right.

#### 3.2.2 Pickups.

. The cat eats purple space bugs that move within the boundaries of the screen. The cat gains points for eating the bugs; the player advances to the next level if all bugs have been eaten. When devouring a bug, the space cat sprite flashes green if the player has enabled that effect.

#### 3.2.3 Player Shooting.

. The space bar can activate laser beams if the player triggers the shooting functionality. The space cat's position matches the direction of the beams. The player can earn points for shooting at the enemies who disappear from the screen upon being hit. Unlike the bugs, however, shooting all the enemies does not enable the player to move to the next level.

#### 3.2.4 Enemies.

. Depending on the level, differing enemies chase the cat; for example, four UFOs travel between the corners of the screen in the first one. The space cat must avoid them while hunting for bugs. In more advanced levels, enemies that spawn off-screen appear in waves.

When attacked by an enemy, the player flashes red. If the player gets attacked nine times, the cat loses its life, leading to a game over. The only way to make enemies vanish is by shooting at them.

### 3.3 Gameplay

The gameplay of *Hungry Space Cats* - derived from an uncomplicated concept, as shown in Figure 9- is a sequence of:

- The space cat enters the scene from any random location on the screen within its boundaries.
- Bugs appear on the screen in any random location on the screen within its boundaries.
- The bugs move up and down.
- The cat moves up, down, left or right to look for space bugs.
- If the option is turned on, the space cat can shoot at enemies using the space bar.
- 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, avoiding colliding with each other using a navigation mesh.
- The ghosts move based on wave points.
- The spaceship that shoots follows the player.
- The second spaceship that shoots also 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 then move around the screen.
- Planets spawn randomly in a scene but do not affect the player.

### 3.4 Tools Used

The game uses the Unity engine and C# programming language. For automated testing, the project utilises Unity's testing framework. *Hungry Space Cat* also engages in 2D development to honour the original design of arcade games.

### 3.5 Game Object Components

The following paragraphs summarise the main playable or opposing game components of *Hungry Space Cat*, each dealing with a game object and its related actions. As depicted in Figure 10, the game contains three character-related game objects that drive its main actions.

#### 3.5.1 SpaceCat.

. The *SpaceCat* component deals with the player's movements and reactions to enemies regarding a collision. In all cases, a collision with an enemy leads to the space cat losing points and its eventual death if its nine lives have run out.

If enabled, this component addresses the cat's shooting behaviour and is always responsible for the space cat's actions when dealing with the pickups. In the game, a green

**Figure 11.** The space cat sprite [10] is cute and colourful.

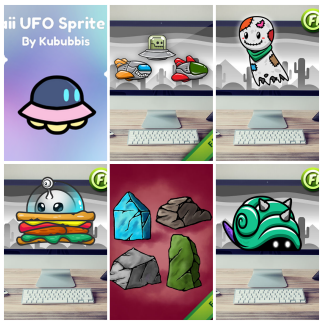


**Figure 12.** A collage depicting the eleven enemies that - apart from the angry UFOs [11] - come from the same artist [10].

astronaut cat sprite [10] represents the *SpaceCat* component, as shown in Figure 11.

#### 3.5.2 Enemies.

. The *Enemies* component handles each enemy's movements and appearance in the scene. It also addresses – as in the case of the spaceships – their shooting behaviour, which causes the player to incur damage if hit by a bullet. This component further causes the player to be injured if they collide with an enemy.

In total, there are eleven enemies – shown in Figure 12– consisting of:

- Cute, angry UFOs [11].
- Flying hamburgers [10].
- Two non-identically coloured spaceships [10].
- A skeleton-based UFO [10].
- Snails [10].
- Four disparate asteroids [10].

#### 3.5.3 SpaceBugs.

. The *SpaceBugs* component – shown in Figure 13– deals with the pickups' movement and reaction to encountering the player. It also deals with how they first appear in the scene. The player earns points for eating all the bugs and advances to the next level once they have done so.


**Figure 13.** The purple space bug the player must consume to earn points [10].

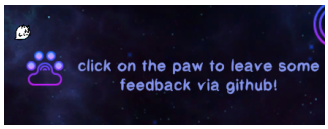


**Figure 14.** Clicking the link above leads to a GitHub account.

### 3.6 Other Game Components

The other components in the game are responsible for background actions such as audio, sprite effects and the UI interface. The following paragraphs summarise these components and describe their functionality in more detail.

#### 3.6.1 AudioManager.

. The *AudioManager* component deals with the sound effects used during the game and the volume handling. More explicitly, regarding volume handling, this component is responsible for muting and unmuting any sound played during the game.

#### 3.6.2 SceneLoadingManager.

. The *SceneLoadingManager* component deals with how the scenes are loaded. More precisely, this component handles:

- The loading of scenes from the menu, including the accessible and challenging modes.
- The loading of scenes from the *Resume Game* button when the game is paused.
- Loading any external sites when the *Adopt a Cat* or *Leave Feedback* buttons are pressed, as shown in Figure 14.
- Loading new levels.
- Loading the *GameOver* scene when the player loses or finishes the game.
- Loading the *MainMenu* scene if the player returns to the menu.
- Exiting the game.

#### 3.6.3 ScoreManager.

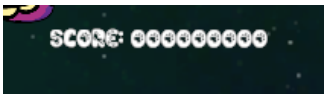


**Figure 15.** The score updates when the user eats a bug or shoots an enemy.

**Figure 16.** The player loses a life when colliding with an enemy or getting hit by a bullet.

. The *ScoreManager* component controls the user's score, displayed in Figure 15. The *ScoreManager* modifies the score when the player consumes a bug or shoots at an enemy. If the player collides with an enemy and loses a life, this component updates the score on the screen display. When the player starts a new game, this component resets the score.

#### 3.6.4 HealthKeeper.

. The *HealthKeeper* component manages the player's number of lives (Figure 16). If the player collides with an enemy, the number of lives is updated and resets when the game ends.

#### 3.6.5 UI.

. The *UI* component handles the following:

- The game's game over text and the user's final score display.
- Setting the pause menu components active when the user presses the ESC key.
- Updating the speed display when the user selects an integer value using the *AdjustSpeed* slider.
- Displaying the text that tells the player the game is loading between level transitions.
- Displaying the text that tells the player they will be redirected to the next level after eating all the bugs.
- Displaying the player's score and number of lives.

#### 3.6.6 Effects.

. The *Effects* component deals with effects and animations, in particular the ones listed below:

- The background scrolling.
- The player flashes magenta when they get hit by an enemy.
- The player flashes cyan after eating a bug.
- The fade-in-and-out animation occurs when the user exits the game or switches between levels.
- The animation of sprites, such as the player and enemies.

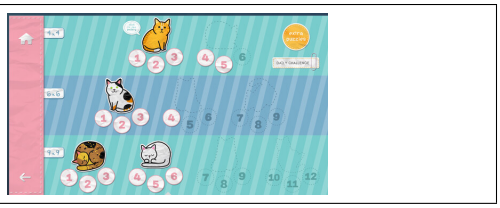

**Figure 17.** *Sudocats* [12] boasts a colourful and cheerful UI interface.

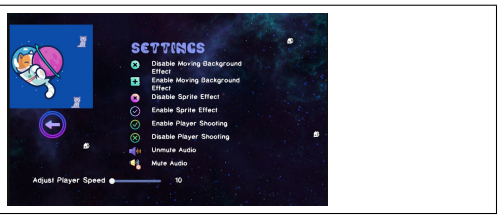


**Figure 18.** The UI aims to be functional and cutesy.

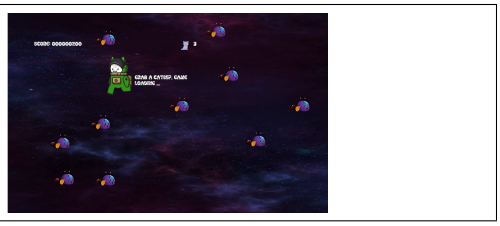

**Figure 19.** All levels share common elements to avoid nasty surprises for the player.

#### 3.6.7 Timer.

. The *Timer* component gives the game an internal countdown between levels, allowing players to prepare for new ones.

### 3.7 Menu and UI Design

. The cat-based indie game *Sudocats* - shown in Figure 17 - inspired the menu design of *Hungry Space Cat*. In other words, the aim was to create a colourful, lively, and friendly UI. The reasoning behind a less simplistic interface was to make the game stand out in terms of aesthetics. In particular, *Hungry Space Cat* wants to be unique, fun, and charming.

As shown in Figure 18, the UI and menu design aims to:

- Elicit cheerful emotions by using cute images and bright buttons, engaging in colours such as purple or green.
- Provide users with visual and textual clues as to the purpose of the buttons.
- Emphasise the feline nature of the game by placing cat images where the users can see them.

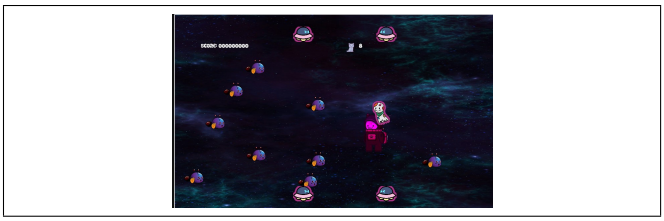

**Figure 20.** The last level involves a combination of enemies from the first level and some ghost dolls coming in waves.

### 3.8 Level Design

The level design of *Hungry Space Cat*, depicted in Figure 19, aims to be consistent, providing the following elements for all levels:

- A space-themed background.
- A score and number of lives text display that uses feline-inspired elements.
- The appearance of space bugs.
- The appearance of enemies.
- The ability to customise the speed of the player.
- The ability to choose whether the player wants to shoot at enemies.

Using similar elements at each level offers reassurance and gives the game a soothing atmosphere. Calming and peaceful gameplay provides the player with a safe space and allows them a form of escapism. Yet, as the following subsections showcase, players require a challenge, which is why all individual levels contain differences.

#### 3.8.1 Accessible Mode.

. The accessible mode consists of three levels, with the speed of the space cat determined by instantaneous stopping and starting.

For the first level, the player must face four angry-looking UFOs that move between the scenes' borders. The second, more challenging one involves four flying hamburgers pursuing the player.

The third level (Figure 20) is the most difficult. It consists of the return of the UFOs from the first level but also contains ghost dolls that come and attack the player in waves.

#### 3.8.2 Challenging Mode.

. The challenging mode has six levels, with the first three repeating the accessible mode. However, in this mode, the player moves using momentum.

The fourth level includes some planets as pretty background features but also adds the ghosts that come in waves from the third level. New enemies, snails that move up, down, left, and right, appear to spice things up. The fifth level introduces a new enemy: a spaceship that shoots at the player while following them. Moreover, skull-based UFOs appear in waves.

**Figure 21.** The last level of the normal mode is a scene of chaos.

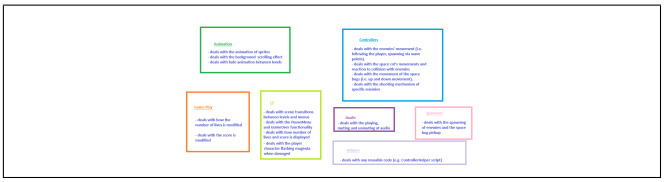

**Figure 22.** There are seven categories that each serve a specific function or purpose.

The last level - shown in Figure 21- is the most chaotic of all levels, incorporating four asteroids of assorted colours and sizes moving across the boundaries of the screen. To add even more mayhem and anarchy, snails from the fourth level show up, along with a spaceship from the fifth level that chases the player and shoots at them.

. (1829 words.)

## 4 GAME IMPLEMENTATION

The following chapter first provides an overview of the project's structure before exploring some code snippets and implementation concepts in more detail.

Please consult GitHub for the entire implementation; discussing the entire codebase is beyond the scope of this chapter.

### 4.1 Code Structure

Figure 22 shows that *Hungry Space Cat's* code structure comprises seven directories responsible for a specific functionality. The following subsections showcase each of the areas more closely.

#### 4.1.1 Animation.

. The *Animation* folder is responsible for a subset of features, ranging from:

- Displaying arrays of sprites and their change in animation over time.
- Handling the fading in and out of levels and other scene exits (e.g. game over).
- Sprite effects, such as the character flashing cyan when scoring a point.

#### 4.1.2 Audio.

. The *Audio* folder takes care of the following:


**Figure 23.** A button click is all it takes for the player to choose between one of two modes.

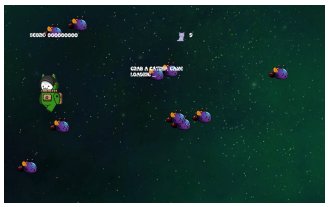

**Figure 24.** When transitioning between levels, a text informs the user that the game is loading.

- Audio effects, e.g. picking up a coin or the player taking damage.
- SFX effects including enemy and player shooting.
- Muting and unmuting the audio, as shown in Figure 18.
- The game's background music.

#### 4.1.3 Controllers.

. The *Controllers* folder includes two subdirectories - *EnemyControllers* and OtherControllers.

The *Controllers* folder handles the movement and collision of game objects, including enemies, pickups, and the players themselves. More specifically, controllers deal with:

- How asteroids rotate and move in a scene.
- How enemies follow a player.
- How spaceships shoot bullets at the player.
- How snails and other enemies move.
- How the player reacts to colliding with enemies.
- How the player reacts to being shot by spaceships.
- How the enemies react to being shot by the player.
- How the pickups react when eaten by the player.
- How the player reacts upon gaining a point or taking damage.
- How the player reacts to dying.
- How the player moves in a scene depending on the user input.
- How the player shoots at enemies depending on the user input.

#### 4.1.4 GamePlay.

. The *GamePlay* folder deals with the following aspects of *Hungry Space Cat* and its gameplay:

- Adjusting the player character's speed, i.e., the space cat, by making them either slow or faster.
- Turning on and off the background scrolling effect.
- Tuning on and off the sprite's flashing effects.
- The calculation of the game's score and the amount of lives available.
- Modifying the game's score when the player eats a bug or shoots at an enemy.
- Modifying the number of lives when the player gets hit by a bullet or collides with an enemy.
- The loading of scenes depending on the game's progression or the user input (e.g. ending the game when the user presses the *Exit* button).
- Ensuring an internal timer helps the game load between levels, allowing players to prepare for them.
- Turning on and off the player's ability to shoot at enemies.
- Handing any loading of scenes triggered by button clicks on the menu, as shown in Figure 23.

#### 4.1.5 Helpers.

. The *Helpers* folder includes reusable functions found in the other directories, such as:

- Clamping sprite movements.
- Initialising bounds for the clamping of movements.
- Following the player.
- Moving away from the player.
- Flipping sprites.
- Spawning objects into the scenes in waves.
- Instantiating objects into the scenes using an object pool.
- Instantiating objects into the scene without using an object pool.

#### 4.1.6 Spawners.

. The Spawners folder - consisting of the subdirectories *EnemySpawners* and *OtherSpawners* - deals with:

- The spawning of objects at the edges of a screen.
- Spawning objects randomly within a scene.

#### 4.1.7 UI.

. 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, as shown in Figure 25.
- Any text displayed during the game's level transitions, as shown in Figure 24.
- Displaying menu buttons and feline-related images.
- Dealing with the display of the player's speed value (18) and the slider that adjusts it.

**Figure 25.** The pause menu components are loaded when the user presses the ESC key.

```
void MoveAnimationForward()
{
    if (!spriteRenderer.enabled)
    {
        return;
    }
    else
    {
        animationFrame++;
        CheckAnimationFrameConditions();
    }
}
```

**Figure 26.** The *MoveAnimationForward* function first checks that a *SpriteRenderer* component is enabled before verifying the animation frame conditions.

- Displaying the score and 'number of lives' components on the screen, shown in Figure 16.

### 4.2 The AnimatedSprite Script

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. The script is attached to all game objects that use multiple sprites. The implementation is based on a *Pac-Man* tutorial [37].

The *MoveAnimationForward* function increments the animation frame variable by one when the game object's *SpriteRenderer* component is enabled, and the animation frame conditions are fulfilled—i.e. the *CheckAnimationFrameConditions* method runs without issues. A *SpriteRenderer* component is responsible for how a sprite is rendered and depicted on a screen [38]. The function is displayed in Figure 26.

The *CheckAnimationFrameConditions* function, called in the *MoveAnimationForward* method (Figure 26), 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. It also checks that the game is running/not paused.

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. The code snippet is shown in Figure 27.

### 4.3 The BackgroundSpriteScroller Script

The *BackgroundSpriteScroller* helps with parallax scrolling (Figure 28), in which background images move more slowly than forefront ones [13]. Parallax scrolling became popular

```
void CheckAnimationFrameConditions()
{
    if (animationFrame >= spritesArray.Length && animationShouldLoop && !PauseMenu.isPaused)
    {
        animationFrame = 0;
    }

    if (animationFrame >= 0 && animationFrame < spritesArray.Length && !PauseMenu.isPaused)
    {
        spriteRenderer.sprite = spritesArray[animationFrame];
    }
}
```

**Figure 27.** The *CheckAnimationFrameConditions* method checks for animation frame conditions to ensure the animation can continue looping.

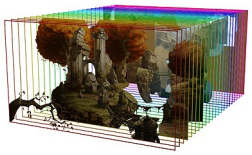

**Figure 28.** The image uses the layering method to show how parallax scrolling looks from the side [13].

```
void Update()
{
    if (DealWithEffects.backGroundEffectsEnabled)
    {
        _offset = moveSpeed * Time.deltaTime;
        _backgroundMaterial.mainTextureOffset += _offset;
    }
}
```

**Figure 29.** The code above is responsible for creating the scrolling effect.

in the early 1980s, along with video arcade games such as *Jump Bug* [13].

There are two methods for background scrolling: layering, which involves multiple backgrounds and changing each layer's position by a different amount in the same direction [13]. The other method engages pseudo-layers using sprites drawn by hardware on top of layers [13]. *Hungry Space Cat* uses layering to honour the design of arcade games.

Within the *BackgroundSpriteScroller* script's *Update* function (Figure 29), modelled after an implementation from a 2D game tutorial by GameDev.tv [39], a private offset variable is multiplied by the move speed and *Time.deltaTime* to create smooth movement.

Afterwards, the offset is added to the background material's texture offset to create the scrolling effect. A static boolean checks whether this option has been triggered (i.e. the user has chosen to turn off the effects in the *Game Settings* menu, as shown in Figure 18).

To ensure smooth movements with the scrolling backgrounds, Unity uses *Time.deltaTime* - displayed in Figure 30 - 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 varied forms of hardware [14].

### 4.4 The SpriteEffects Script

The *SpriteEffects* script handles the space cat's flashing a different colour when the player eats a bug or is injured by an enemy through collision or shooting. The code in Figure 31

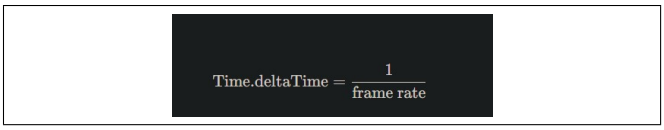


**Figure 30.** The formula helps normalise calculating the rate at which a machine renders a frame [14]. Calculating a game object's vector in Unity with the equation returns a fixed speed, no matter the frame rate [14].

```
public IEnumerator DisplayDeathEffect(SpriteRenderer spriteRenderer, float damageDelay)
{
    spriteRenderer.color = Color.magenta;
    yield
    return new WaitForSeconds(damageDelay);
    spriteRenderer.color = Color.white;
}

public IEnumerator DisplayScoreEffect(SpriteRenderer spriteRenderer)
{
    spriteRenderer.color = Color.cyan;
    yield
    return new WaitForSeconds(0.4f);
    spriteRenderer.color = Color.white;
}
```

**Figure 31.** The sprite functions above exist to make the player flash magenta or cyan depending on certain game events.

```
public class DealWithEffects : MonoBehaviour
{
    public static bool backGroundEffectsEnabled = true;
    public static bool spriteEffectsEnabled = true;

    public void DisableMovingBackgroundEffect() => backGroundEffectsEnabled = false;
    public void EnableMovingBackgroundEffect() => backGroundEffectsEnabled = true;
    public void DisableSpriteEffects() => spriteEffectsEnabled = false;
    public void EnableSpriteEffects() => spriteEffectsEnabled = true;
}
```

**Figure 32.** The *DealWithEffects* script helps the user enable and disable animation-related effects.

does not adhere to the traditional purpose of the *IEnumerator* in C#, which fetches a current value from a collection [40] and only allows for forward cursor movement when iterating through it [41].

In Unity, using a coroutine and the keyword *yield* suspends the method until the next frame or a set amount of time passes (e.g., WaitForSeconds()) [42]. Usually, a standard void method applies all of its functions in a single frame [42]. A coroutine is used to flash sprite effects after ensuring a delay between the actions, triggering the change of the player's colour and displaying the change of colour effect itself.

### 4.5 The DealWithEffects Script

The *DealWithEfects* class contains public methods for turning effects on or off throughout the game. Static variables are created once as a copy and then shared [43].

The functions in Figure 32 control the disabling of the sprawling background and sprite effects, allowing the user to take control of the UI and animation. Giving back power to the player regarding such features enables them to make the game more playable, especially as the flashing sprite effects and the sprawling backgrounds can be too much for some people.

### 4.6 The ObjectPool Script

Object pooling improves an application's performance by instantiating and destroying an array of objects on demand [44]. It is a design pattern that reduces the burden on the CPU

```
void CreatePoolOfObjects()
{
    _objectPool = new GameObject[_poolSize];
    for (int i = 0; i < _objectPool.Length; i++)
    {
        _objectPool[i] = Instantiate(objectPrefab, transform);
        _objectPool[i].SetActive(false);

    }
}
```

**Figure 33.** The *CreatePoolOfObjects* function creates the underlying instances of objects that the players may wish to use later.

```
public GameObject GetPooledObjects()
{
    for (int i = 0; i < _objectPool.Length; i++)
    {
        if (_objectPool[i].activeInHierarchy == false)
        {
            _objectPool[i].SetActive(true);
            return _objectPool[i];
        }
    }

    return null;
}
```

**Figure 34.** The *GetPooledObjects* function sets objects to active and inactive per the game's needs.

by pre-creating the game objects needed before gameplay [44]. By doing that, there is no need to create new objects or destroy old ones [44].

The *ObjectPool* script is based on an implementation in a Unity tutorial [44] and another for a 3D Game tutorial by GameDev.tv [45].

As shown in Figure 33, the size of the object pool is determined during the array's creation in the *CreatePoolOfObjects* function. Afterwards, a for loop instantiates the required objects and sets them to inactive so they are not viewable before the required gameplay.

In the *GetPooledObjects* (Figure 34) function, which is activated in the *Awake* method and returns a game object, an object from the pool is fetched from a for loop after being set to active. This only occurs if the object in the hierarchy is set to false, as - otherwise - an object that is already in use would be activated, thus defeating the purpose of using an object pool.

The function above is used for spawning space bugs; however, the implementation is not a singleton, which would have been even more effective than using several object pools across multiple scenes.

Implementing such a solution had to be abandoned due to the project's time constraints and the relatively late addition of the object pool. Moreover, although an attempt was made, bullet creation does not use an object pool because it proved inefficient.

### 4.7 The FollowPlayer Script

The *ControllerHelper* function contains several methods for moving objects, but they are placed within a single file for reusability. Code reusability follows the principle of DRY and helps make the implementation more beneficial for future projects.


**Figure 35.** The *FollowPlayer* function enables an object to follow a given target.

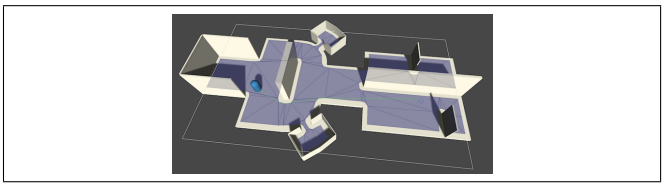

**Figure 36.** A *navmesh* does not have equally sized nodes [15].


**Figure 37.** *FixedUpdate* is used for the movement in the *FlyingHamburgerController* script, as the flying hamburgers have rigid bodies attached, and this function is in synch with the game's physics system [16].

One of these functions—the *FollowPlayer* one (Figure 35)—involves a game object following another depending on its position. The implementation is based on a video tutorial using Navmesh in Unity 2D [46].

Unity's navmesh system is based on AI pathfinding [47]. Navmesh—or navigation mesh—is an abstract data structure that helps agents find their way through complicated spaces by representing the world using polygons [48], which describe the surface game objects can walk on [15].

Navmeshes are more beneficial than nodes arranged in a grid pattern, requiring fewer nodes [15]. Having too many nodes can feel superfluous when designing a landscape needing only a few areas the play can navigate around [15]. Figure 36 shows that navmeshes consist of nodes of miscellaneous sizes based on the game's needs [15].

As the nodes in a navigation mesh are smaller, they can more accurately describe the environment [15]. Furthermore, they are smaller because no space is wasted on empty notes, which makes them faster for AI pathfinding [15]. AI pathfinding deals with locating the shortest path between two routes while avoiding obstacles [49].

In *Hungry Space Cat*, the navigation mesh prevents the flying hamburgers—representing the enemies in the second level in both the accessible and challenging mode—from colliding with each other while following the player.

The *FollowPlayer* function is called in the *FlyingHamburgerController* script in the *FixedUpDate* function (Figure 37), which deals with how physical objects move in the game [16]. Unlike the *Update* function, it is frame rate independent and called at timed intervals [16].

Before the *FollowPlayer* function can be called, an if statement checks that the game is running, that the space cat's target position is present and that the flying hamburgers have a *Rigidbody* attached, which controls their position using physics simulation [50].

. (1983 words.)

# 5 GAME EVALUATION

The following chapter used Python scripts to create word clouds and pie charts to visualise results. All the scripts can be found on GitHub. Additionally, JupyterLite and W3Schools generated the images from the Python code used as figures throughout this chapter.

## 5.1 Evaluation Factors

*Hungry Space* evaluates its quality using manual testing, automated unit testing, and playtesting. In playtesting, honest feedback was favoured over questionnaires. The factors described in the following subsections have determined the project's success.

### 5.1.1 The Game's Playability.

. The truth of the following statements measures the playability of the game:

- The game loads without any issues.
- The game transitions from one level to another, providing players with a well-deserved breather in between.
- The user pauses the game and resumes it without any hiccups.
- The user can exit the game without any issues.
- The game ends with a game-over text that informs the player that it has ended.

### 5.1.2 The Game's Enjoyability.

. While subjective, the enjoyability of the game is judged by:

- How users describe the game and its overall gameplay.
- The absence of bugs or unexpected crashes.
- The degree of simplicity or difficulty of the game.
- The accessibility of the game.
- The atmosphere and the look and feel of the game.

### 5.1.3 The Game's Aesthetics.

. The aesthetics of the game are measured by:

- The appearance of the sprites and their animation.
- The way the graphics enhance the game.
- The colour palette and its consistency across the game.
- The way effects inform the user about actions and their consequences.

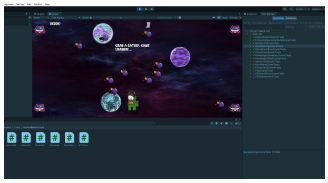

**Figure 38.** *Hungry Space Cat* uses automated tests to check for the presence of given elements in each scene.

#### 5.1.4 The Game's UI.

. The UI's success is evaluated by:

- The clarity regarding the purpose of the buttons.
- The way the buttons and the appearance of the UI fit together.
- The size and readability of the fonts.
- The accessibility of the UI.

#### 5.1.5 The Game's Automated Unit Tests.

. The success of the game's (Figure 38) automated unit tests is decided by the validity of the following statements:

- All text elements are present in the scenes.
- All buttons are present within the menu scenes.
- The game scenes contain the space cat character prefab.
- The game scenes contain the required enemies and script prefabs.
- The game scenes contain the proper *Navmesh* components, if applicable.
- The game scenes contain the pause menu components.
- The game scenes contain the components of the score and 'number of cat lives'.

While automated tests cannot guarantee the absence of bugs or replace playtesters, they provide fast feedback when changes are made to the game and - thus - provide confidence that any previously working features were not broken. The visual feedback offered by Unity's play mode is beneficial, as the tests are run in the editor itself and help a developer understand what is working.

#### 5.1.6 The Game's Progress.

. The criteria below measure the game's success against the project guidelines and deadline:

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

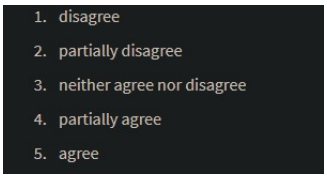


**Figure 39.** The reviewers could answer questions by selecting the criteria in the screenshot.

### 5.2 Evaluation Questions

During peer-graded reviews taking place in weeks 14 and 18 of the module, an evaluation asked users the following questions, which were answerable to the following criteria as displayed in Figure 39: The evaluation questions comprise the following:

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

The purpose of using open-ended questions that could still be answered by 'yes/no' was to encourage players to write their own narratives about the gameplay. Consequently, the feedback would be more than just metrics but a story with a life of its own.

### 5.3 Playtesting Sessions

More than twenty playtesting sessions took place within the module's lifespan. Players, ranging from two to five at a time, included volunteers from the module, acquaintances from a gaming-related *Discord* channel, and - at one point - two professional game testers from Playcocola.

To avoid infusing players with my biases, they were given free rein for feedback, as the focus lay on gathering their true feelings regarding the gameplay and other aspects of *Hungry Space Cat*. Playtesting sessions started early in the semester, with records kept regarding each session to keep track of bugs and any other feedback mentioned.

While few playtesters played the game throughout the entire playtesting rounds, feedback proved valuable, as important insights, such as the menu requiring game instructions and the gameplay needing more customisation options, came to fruition because of them.

### 5.4 Evaluation Results

The overall consensus for *Hungry Space Cat* —as the following sections will explore—was positive, with players describing the gameplay as fun. The game's look and feel also garnered favourable feedback. The following subsections reflect on gameplay, UI and music, graphics and the game itself.

#### 5.4.1 Gameplay.


**Figure 40.** The word cloud above is a collection of adjectives players have used to describe the gameplay during the playtesting and peer-graded sessions.

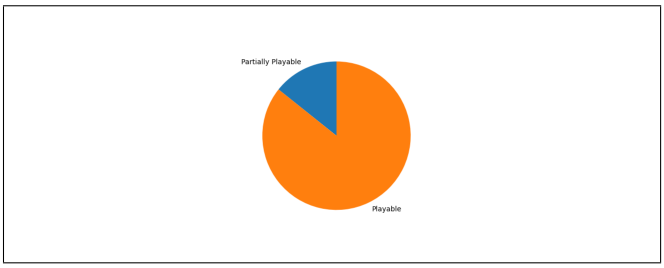


**Figure 41.** The chart above comes from week 18, where seven people responded, with the majority stating that the game was playable.

. As the word cloud in Figure 40 indicates, the overall narrative about the gameplay was that it was fun, good and entertaining. Additionally, the words 'challenging' and 'playable' crop up often, suggesting that - as an arcade game - *Hungry Space Cat* has fulfilled its objective of becoming more difficult with each level [9].

However, while the word cloud suggests cautious optimism, the pie chart (Figure 41) from the peer-graded evaluation sessions implies room for improvement in the gameplay, especially since a small percentage of the players said the game was partially playable. While week 18 showed progress in playability, the small pool of players - seven, to be exact - makes even one person's statement that the game was partially playable worrying, specifically since the previous round contained merely eleven people responding to the evaluation.

Written feedback from both the peer-graded comments and playtesting sessions suggests that people would like more levels and even more range of enemies, which is not possible due to the project's relatively short development span.

Overall, the gameplay seems successful, but more data would be required to capture the nuances behind why the game is playable or unplayable, especially in light of differing tastes and preferences.

### 5.4.2 Game Graphics.

***A.*** s Figure 42 suggests, the graphics of *Hungry Space Cat* are nice, cartoony, and velvety, indicating that the game has a pleasant appearance. The only negative comment regarding


**Figure 42.** The word cloud above captures adjectives players have used to describe the game's look and feel during playtesting and peer-graded evaluation sessions.

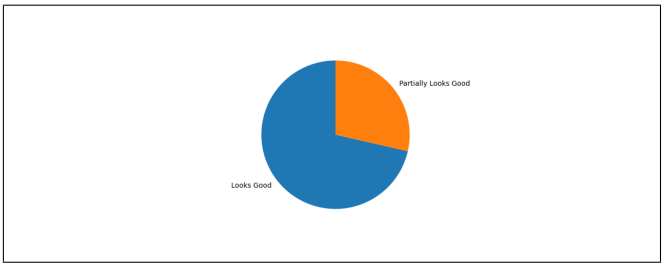


**Figure 43.** In week 18, seven players thought the game looked good, while two thought it only partially did so.


**Figure 44.** As with previous word clouds, the word cloud above uses a collection of words that players have used to describe the game's UI and music.

the game's graphics came from one player, who stated that the sprite for the space cat character was unappealing.

### 5.4.3 Game UI and Music.

. While the pie chart (Figure 43) from week 18 shows that two out of seven players thought the game looked partially good, the response to the UI has enjoyed a tumultuous history. Much of the game's early development focused on improving the buttons' layout, adjusting the audio's customisability and devising a less clunky menu design.

Early feedback from playtesters emphasised the game's clumsy menu and lack of coherency regarding the buttons' purpose; however, as the word cloud in Figure 44 showcases, improvements were made to make the UI more straightforward. To some degree, the menu seems to look acceptable to players, even though more data would be required to reach a broader consensus on this subject.

Meanwhile, the music has enjoyed consistently positive feedback, with playtesters stating it was good.

### 5.4.4 The Entire Game Itself.

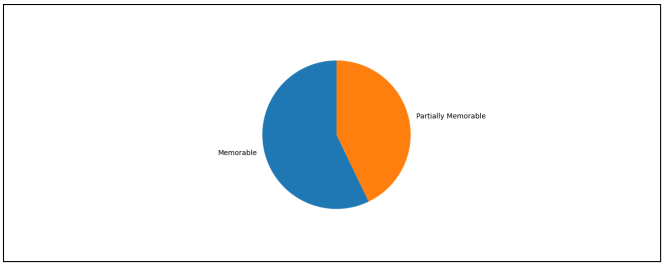


**Figure 45.** While memorability is subjective, four out of nine players thought the game was partially memorable.


**Figure 46.** The word cloud above is a compilation of all the words players have used to describe the game.

. Figure 45 shows that four out of nine players thought the game was partially memorable, suggesting they felt bored or only wanted to play it once. As the term 'memorability' is subjective and the pool of playtesters for this evaluation was small, it can be hoped that more data would indicate a better (or worse!) outcome.

However, the word cloud concerning the entire game (Figure 46) portrays a more hopeful outcome than the pie chart earlier. It tells a story of a fun, simple, good and nice game. While those adjectives are not accolades of greatness, they suggest that *Hungry Space Cat* is, at least, a decent prototype.

Of course, the word cloud comprises all the words used to describe the previous word clouds, meaning that the results are inferred. That is why - as with prior sections - the evaluation should be viewed critically, as more data would be needed to produce more precise results.

### 5.5 Overall Evaluation

From the perspective of the game being finished on time, *Hungry Space* is a success, as the development of the core features has gone according to plan. From the players' point of view, the game is playable and fun, even if not the most memorable, which hints at success but also gives room for future work.

. (1454 words.)

## 6 CONCLUSION

The following section summarises the conclusions drawn from creating *Hungry Space Cat*, including any planned future work.

#### 6.0.1 Personal Development.

. Given that I was inexperienced as a game developer and had only previously worked on another game as part of a team, I am proud of the outcome of this project. It met all its deadlines despite full-time work and other academic commitments.

Not only that, but I embarked on this degree as an unseasoned video game player who knew little and appreciated even less the complexities of creating a game. I had only ever heard about Unity, and it was only through this course that I understood how it worked. To see an idea go from being a rough outline in my head to a playable game has been a bit baffling and humbling but also rewarding.

*Hungry Space Cat* is a working game prototype with a look and feel precisely as feline-inspired as initially envisioned. Furthermore, the gameplay is cute, and I have come a long way since first undertaking the project. Not only can I tackle the programming of a game using various tools independently, but I have grown more confident doing so.

#### 6.0.2 Future Work.

. Some areas of *Hungry Space Cat* could benefit from code refactoring regardless of the working prototype. For example, as mentioned in the implementation section, the object pool should be a singleton for better code reuse and more efficient performance. With lack of time no longer a decisive factor in the game's development, I can research more game design and development patterns, helping me refine the object pool code and look into polishing other scripts.

Code that needs refactoring includes the overall structure that does not adhere to any programming patterns, as the project focused on completing it on time. Thus, future work will involve polishing the code while ensuring it becomes more elegant and readable. Unity itself offers e-books that teach developers about game programming patterns.

Moreover, to help lessen my reliance on game artists, I plan to learn more about designing my own sprites and shaders. Learning more about game design would help *Hungry Space Cat* feel more original, grow my skill set, and enable me to further grasp the secret behind making a good game. It would also feel more gratifying, as there is an undeniable beauty in creating your own assets.

#### 6.0.3 Beyond This Project.

. Beyond this project, I want to explore other game engines like Unreal or Godot. Learning about these engines would expand my knowledge of game creation and give me the tools to become a more well-rounded developer.

In addition to learning more game engines, I would like to know about UX and UI concepts to make better menus that are accessible and aesthetically pleasing. In that vein, an increased exposure to web design and applications may also decrease my deficits in this area.

. (474 words.)

## References


[1] Mike Lenz. 2023. *Cat in space station (00184).* DeviantArt. Retrieved August 19, 2024 from https://www.deviantart.com/a2a5/art/Cat-in-space-station-00184-950887557

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

[3] ouno. 2023. *PACMAN PIZZA.* pampling. Retrieved August 21, 2024 from https://www.pampling.com/disenos/122081-Pacman-Pizza

[4] Tony Temple. 2021. *Atari Asteroids: Creating a Vector Arcade Classic.* The Arcade Blogger. Retrieved August 21, 2024 from https://arcadeblogger.com/2018/10/24/atari-asteroids-creating-a-vector-arcade-classic/

[5] atariprotos. 2023. *Cat Trax.* AtariProtos. Retrieved August 21, 2024 from https://www.atariprotos.com/2600/software/cattrax/cattrax.htm

[6] RetroGames. 2023. *Mappy.* RetroGames. Retrieved August 21, 2024 from https://www.retrogames.cz/play_008-NES.php

[7] Divok. 2023. *Pac Cat: retro cat.* Google Play. Retrieved August 21, 2024 from https://play.google.com/store/apps/details?id=com.Divok.PacCat&hl=en&gl=US

[8] HedonisticOpportunist. 2024. *SPACE CATS - A GAME WEB APPLICATION.* Coding Black Females. Retrieved August 21, 2024 from https://github.com/HedonisticOpportunist/Space-Cats

[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* (Utrecht School of the Arts, Hilversum, The Netherlands). Digital Games Research Association, Hilversum, The Netherlands, 8 pages. https://dl.digra.org/index.php/dl/article/view/527/527

[10] Bevouliin. 2023. *Bevouliin - Selling 2D Game Assets for Game Developers.* Bevouliin. Retrieved August 19, 2024 from https://bevouliin.com

[11] kububbis. 2022. *Kawaii UFO Sprite Pack.* itch.io. Retrieved August 22, 2024 from https://kububbis.itch.io/kawaii-ufo-sprite-pack

[12] Devcats. 2023. *Sudocats.* Steam. Retrieved August 23, 2024 from https://store.steampowered.com/app/1725640/Sudocats/

[13] Wikipedia. 2023. *Sprite Renderer.* Wikipedia. Retrieved August 24, 2024 from https://en.wikipedia.org/wiki/Parallax_scrolling

[14] educative. 2023. *What is "Time.deltaTime" in Unity?* educative.io. Retrieved August 24, 2024 from https://www.educative.io/answers/what-is-timedeltatime-in-unity

[15] Aron Granberg. 2024. *Getting Started with the A* Pathfinding Project - Part 2 - Navmeshes.* Wikipedia. Retrieved August 25, 2024 from https://arongranberg.com/astar/docs/getstarted2.html

[16] John French. 2023. *How to use Fixed Update in Unity.* Game Dev Beginner. Retrieved August 25, 2024 from https://gamedevbeginner.com/how-to-use-fixed-update-in-unity/

[17] Jamey Pittman. 2015. *The Pac-Man Dossier.* holenet. Retrieved August 19, 2024 from https://pacman.holenet.info

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

[19] Scott Ellis. 2023. *Unlocking the Secrets of Classic Arcade Games: What UX Tricks Can We Learn?* LinkedIn. Retrieved August 19, 2024 from https://www.linkedin.com/pulse/unlocking-secrets-classic-arcade-games-what-ux-tricks-dba-llm/

[20] Yuexian Gao, Chang Liu, Naying Gao, Mohd Nor Akmal Khalid, and Hiroyuki Iida. 2022. Nature of arcade games. *Entertainment Computing* 41 (2022), 100469. https://doi.org/10.1016/j.entcom.2021.100469

[21] screamingbrainstudios. 2022. *Seamless Space Backgrounds.* itch.io. Retrieved August 20, 2024 from https://screamingbrainstudios.itch.io/seamless-space-backgrounds

[22] kububbis. 2021. *Kawaii Ghost Sprite Pack Neko Edition.* itch.io. Retrieved August 20, 2024 from https://kububbis.itch.io/kawaii-ghost-sprite-pack-neko-edition

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

[24] Renato Fernando dos Santos and al. 2020. Pac-Man is Overkill. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Las Vegas, NV, USA). IEEE, Las Vegas, NV, USA, 11652–11657. https://doi.org/10.1109/IROS45743.2020.9341274

[25] Adelina Moura. 2015. Using arcade games to engage students in the learning of foreign and mother languages. *EAI Endorsed Transactions on e-Learning* 2, 2 (3 2015), 13 pages. https://doi.org/10.4108/el.2.5.e2

[26] Nicolas Esposito. 2005. How Video Game History Shows Us Why Video Game Nostalgia Is So Important Now. In *Playing the Past.* nicolasesposito, 60205 Compiègne Cedex, France, 11 pages.

[27] Auroch Digital. 2022. *Neurodiversity and the Games Industry).* Auroch Digital. Retrieved August 19, 2024 from https://www.aurochdigital.com/blog/2022/11/28/neurodiversity-and-the-games-industry

[28] Paul Cairns, Christopher Power, Mark Barlet, and Greg Haynes. 2019. Future design of accessibility in games: A design vocabulary. *International Journal of Human-Computer Studies* 131 (2019), 64–71. https://doi.org/10.1016/j.ijhcs.2019.06.010 50 years of the International Journal of Human-Computer Studies. Reflections on the past, present and future of human-centred technologies.

[29] Pallavi Sodhi, Audrey Girouard, and David Thue. 2023. Accessible Play: Towards Designing a Framework for Customizable Accessibility in Games. In *Companion Proceedings of the Annual Symposium on Computer-Human Interaction in Play* (Stratford, ON, Canada) *(CHI PLAY Companion '23).* Association for Computing Machinery, New York, NY, USA, 49–55. https://doi.org/10.1145/3573382.3616075

[30] bdatg2. 2013. *Simplicity.* WordPress. Retrieved August 21, 2024 from https://vgprobs.wordpress.com/simplicity/

[31] Nicolas. 2011. *Kanji Card – taberu.* NIHONGO ICHIBAN. Retrieved August 21, 2024 from https://nihongoichiban.com/2011/04/10/jlpt-kanji-éč§/

[32] Nicolas. 2011. *Kanji Card – kuchi.* NIHONGO ICHIBAN. Retrieved August 21, 2024 from https://nihongoichiban.com/2011/04/09/jlpt-kanji-ăŕč

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

[34] TrekMD. 2016. *Cat Trax.* LaunchBox. Retrieved August 21, 2024 from https://www.retrovideogamer.co.uk/cattrax2600

[35] LaunchBox. 2023. *Cat Trax.* LaunchBox. Retrieved August 21, 2024 from https://gamesdb.launchbox-app.com/games/details/76418-cat-trax

[36] Namco Wiki. 2023. *Mappy.* Namco Wiki. Retrieved August 21, 2024 from https://namco.fandom.com/wiki/Mappy

[37] Adam Graham. 2023. *AnimatedSprite.cs.* GitHub. Retrieved August 24, 2024 from https://github.com/zigurous/unity-pacmantutorial/blob/main/Assets/Scripts/AnimatedSprite.cs

[38] Unity. 2023. *Sprite Renderer.* Unity Documentation. Retrieved August 24, 2024 from https://docs.unity3d.com/Manual/class-SpriteRenderer.html

[39] Gary Pettie. 2021. *17. Scrolling Background.* GameDevTV. Retrieved August 24, 2024 from https://gitlab.com/GameDevTV/unity2d-v3/laser-defender/-/blob/master/Assets/Scripts/SpriteScroller.cs

[40] Tariq Siddiqui. 2022. *IEnumerable and IEnumerator in C#.* CodeGuru. Retrieved August 24, 2024 from https://www.codeguru.com/csharp/ienumerable-ienumerator-c-sharp/

[41] Laks Tutor. 2023. *C# IEnumerable and IEnumerator: An In-depth Exploration from Basics to Advanced.* Medium. Retrieved August 24, 2024 from https://medium.com/@lexitrainerph/c-ienumerable-and-ienumerator-an-in-depth-exploration-from-basics-to-advanced-

b7217fd15e1c
[42] Adam Reed. 2021. *How To Use IEnumerator Coroutines In Unity!* Medium. Retrieved August 24, 2024 from https://adamwreed93.medium.com/how-to-use-ienumerator-coroutines-in-unity-b2fc70d06de1
[43] geeksforgeeks. 2024. *Static keyword in C#.* GeeksForGeeks. Retrieved August 27, 2024 from https://www.geeksforgeeks.org/static-keyword-in-c-sharp/
[44] Unity Technologies. 2022. *Introduction to Object Pooling.* Unity Learn. Retrieved August 25, 2024 from learn.unity.com/tutorial/introduction-to-object-pooling
[45] Gary Pettie. 2020. *17. Object Pools.* GameDevTV. Retrieved August 25, 2024 from https://gitlab.com/GameDevTV/CompleteUnity3D/realmrush/-/blob/master/Assets/Enemy/ObjectPool.cs
[46] Rootbin. 2023. 2024 AI Pathfinding: Unity 2D Pathfinding with NavMesh tutorial in 5 minutes. Retrieved August 25, 2024 from https://www.youtube.com/watch?v=HRX0pUSucW4&t=221s
[47] Ethan Martin. 2023. *Game Dev: Unity/C# — What is Navmesh?* Dev Genius. Retrieved August 25, 2024 from https://blog.devgenius.io/game-dev-unity-c-what-is-navmesh-68a63d63360b
[48] Wikipedia. 2023. *Navigation mesh.* Wikipedia. Retrieved August 25, 2024 from https://en.wikipedia.org/wiki/Navigation_mesh
[49] MAKINDE DAVID CRAIG. 2023. *Unity's AI Pathfinding: Exploring the Artificial Intelligence side of the Unity Engine.* Medium. Retrieved August 25, 2024 from https://medium.com/our-internship-journey/unityengines-ai-pathfinding-exploring-the-artificial-intelligent-side-of-the-unity-engine-1404a3ab76f5
[50] Unity. 2023. *Rigidbody.* Unity Documentation. Retrieved August 25, 2024 from https://docs.unity3d.com/ScriptReference/Rigidbody.html

. (9061 words.)