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 SPACE CAT”

By Anita Pal (200208268)

Please click on the relevant links to view some figures hosted on Google Drive, such as design documents and the work plan. The caption underneath the image highlights them in blue. Some items are also viewable in the appendix.

Tables and references are left-aligned, while any other text is justified. The text uses single-spaced spacing.

Please note that ‘Pac-Man’ or ‘the player’ uses them/their pronouns to avoid making assumptions about gender, etc.

The overall word count for this document is 6000, excluding the title page, table of contents, appendix and references.

1. INTRODUCTION

1.1 Project Template

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

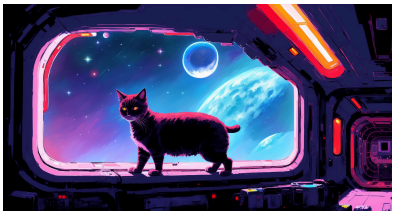


Figure 1.1. The game is cat-themed, as cats are fun, especially in space! [1].

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.

The game—depicted in Figure 1.2—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.

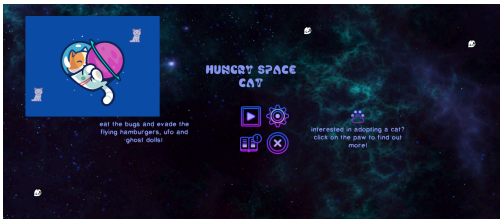


Figure 1.2. The menu scene has an astronaut cat image [4] and a space-themed background [5]. A white ghost cat [6] further enhances the menu's design.

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!) [7]. The player advances through increasingly challenging levels to eat all bugs without losing the cat's nine lives.

The more difficult levels include spaceships shooting at the space cat character that remains defenceless, with only its speed and wit allowing it to evade them; the intention behind this gameplay is to mimic the deceptive simplicity of arcade games [8] while making the game fun to play and something users would want to return to [8, 9].

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,* with its design and gameplay using silly/cute elements to appeal to a broader audience.

The game aims to be playable and fun without requiring much investment or time for the player to understand its plot [8]. 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 [8] – poses more of a challenge, adding impulse to the player's movements and including more levels.

1.3 Project Motivation

Academic/Personal Motivation

Arcade games—one of the earliest forms of electronic entertainment [9]—offer aspiring programmers a glimpse of the magic behind creating them [10]. *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 [11].

Not only do students benefit from learning about such algorithms [10], but researchers have proven *Pac-Ma*n only needs two ghosts to capture the player effectively [11]. Simultaneously, titles such as *Asteroids* need collision detection and distance calculation, allowing researchers and students to explore these topics visually [10]. Lastly, arcade games are practical tools for teaching children to learn new languages because they encourage players to interact directly with the material [12].

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 [7]. Therefore, aligning with the fact video games appeal to people of various ages [12], the game is suitable for people who:

- ★ Enjoy a game without requiring instruction manuals [13].
- ★ Wish to play a game with simple rules [9] and engaging gameplay [9, 12].
- ★ 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 [12].

Objectives

Hungry Space Cat aims to:

- ★ Be a game that is easy to play/pick up, as arcade games are [8].
- ★ Have simple, enjoyable visuals [8] while defying the concept of fast-paced gameplay [12], instead offering the player a choice between two modes.
- ★ Serve as a motivation for players to explore other arcade games using nostalgic design elements [13].
- ★ Encourage people to adopt cats by having them actively engage with a cat character [12].

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 deals with its prototype.

(779 words).

2. LITERATURE REVIEW

2.1 Pac-Man

Pac-Man's History

A 2008 report revealed that 94 % of US consumers recognise *Pac-Man*, beating *Mario* in popularity [7]. According to its creator, Toru Iwatani – a game designer without formal training [7], 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 one for women and couples [2].

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, 14]. Furthermore, Toru Iwatani used the Kanji for eating – *taberu* 食 [15] – as a premise for the game and the one for mouth – *kuchi* 口 [16]-- with its square shape as the basis for the character's design [2, 7].

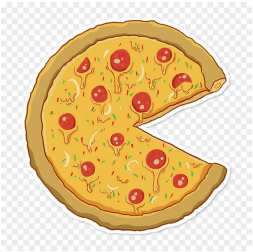


Figure 2.1. According to a popular urban myth, a pizza missing a piece influenced the shape of Pac-Man [7], as shown above [17].

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 [7, 13]. Despite the simplicity, while playing the game, the player's strategy is crucial to winning it [18].

Further, Iwatani restricted the gameplay to a maze where players move in the following directions - up, down, left, and right [7]. As a maze game, *Pac-Man* contains one screen/playfield with little to no scrolling; multiple game levels keep the player's interest alive [18].

Core Mechanics

Pac-Man features a two-dimensional maze [19, 20]. Using a labyrinth relates to *Pac-Man's* characteristics as a maze game [18], where the player navigates the game character via a four-way joystick [19]. The maze has 240 non-flashing pills, each worth 10 points, and four flashing ones, referred to as energisers, worth 50 [20].

The game begins with the four ghosts stationed at the maze's centre, called the *ghost house* [19]. The ghosts are released individually and chase after Pac-Man, aiming to consume them [20]. Pac-Man starts with three lives, losing one each time a ghost catches them [19, 20]. When a life is lost, the ghosts and Pac-Man return to their original positions, but any eaten dots remain [20].

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 [20]. While the game is technically unlimited, a bug prevents the player from surpassing level 255 [20].

The speed of the ghosts is constant, except when travelling through a tunnel [19, 20]. The layout of the game remains the same [19]. However, each level increases in difficulty due to changes in Pac-Man's speed and the ghosts' behaviours [19], as shown in Figure 2.2 and elaborated on in Table 1.1. There are no modifications after reaching level 21 [20].

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

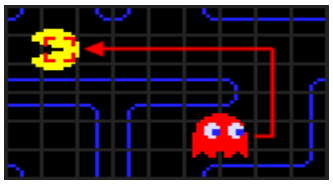
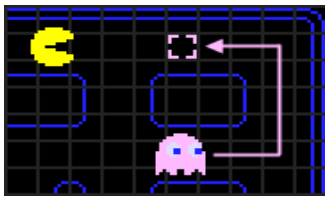
Colour	Orange	Blue	Pink	Red
Name	Clyde	Inky	Pinky	Blinky
Aggressiveness	20%	50%	70%	90%
Territory	Southwest	Southeast	Northwest	Northeast

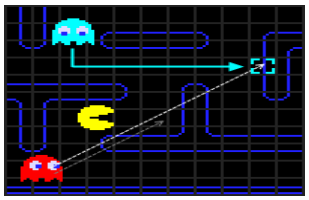
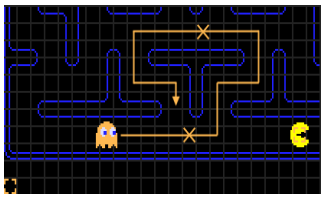
Figure 2.2. The table shows the ghosts' personalities/aggressiveness levels [19].

Ghost Behaviour

Each of the ghosts has a target – a specific tile – assigned to them to reach [19]. 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 [19, 20], with the predominant one being chase, where the spectres use Pac-Man's position to find their target tile [20]. When in scatter or chase mode, the ghosts travel in waves with a set timer, only pausing when in frightened mode [19].

Table 1.1. 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 [20].	Starting outside the ghost house, the red ghost, described as 追いかけ, *oikake,* which means 'pursuer' in English, is the first threat [20]. It immediately starts following Pac-Man, as depicted in Figure 2.3 [20]. Blinky's speed increases when Pac-Man has 20 pellets left to eat [21]. Its scatter mode also changes, ensuring it remains in chase mode rather than targeting the tile at the upper right corner of the maze [20]. These behavioural shifts make Blinky the deadliest of the ghosts [21].
Pink Ghost / Pinky  **Figure 2.4.** The pink ghost's target tile tries to move four spaces ahead of Pac-Man [20].	The pink ghost starts inside the ghost house but exits immediately [20], moving around the maze's walls in an anti-clockwork pattern [21]. Although the ghost is nicknamed 'Pinky' [20, 21], the Japanese description of its personality 待ち伏せ, *machibuse* (ambusher), better suits its behaviour [20]. 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 [20, 21] in an attempt to get in front of the player [20]. However, this behaviour only works if Pac-Man moves left, down or right, with an overflow error preventing it when the player moves upwards [20]. In scatter mode, Pinky moves towards the top left

	corner of the maze in an anti-clockwise, circling motion [21].
Blue Ghost / Inky  **Figure 2.5.** When calculating Inky's target, Pac-Man and Blinky's positions are considered [20].	The blue ghost remains in the ghost house until Pac-Man has eaten 30 pellets [20]. Its Japanese description is 気紛れ, *kimagure*, meaning 'whimsical' in English [20]. 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 [20]. It patrols the area close to the red ghost's location, becoming more random the further away it is from it [21]. In other words, if Blinky is close to the player, so will Inky [20]. In scatter mode, Inky guards the lower corners of the maze [21].
Orange Ghost / Clyde  **Figure 2.6**. The image above shows how Clyde's behaviour changes when close to Pac-Man [20].	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 [20]. The Japanese description for its behaviour translates to お惚け, *otoboke* or feigning ignorance' [20]. 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 [20]. 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 [20, 21]. As such, Clyde switches between following Pac-Man and retreating when it gets too close [20].

Evaluation

Due to its popularity, *Pac-Man* has not only inspired clones (e.g. *Cat Trax*) but has also been used in academic research [11, 14, 19]. *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 [8, 9]. Furthermore, its universal appeal [7] reveals how arcade games can unite a diverse community of players [7, 12] when evoking feelings of nostalgia [13].

2.2 Asteroids

Asteroids' History

In December 1979 [3], Atari released the shoot-em-up *Asteroids* [22]. In line with its characteristics as a shooter arcade game [18], the player controls a spaceship on a single screen to evade asteroids [3, 22]. The game has an easy-to-learn but difficult-to-master gameplay that keeps people returning to it to beat it [22]. 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, depicted in Figure 2.7, *Asteroids* was a response to an unsuccessful game where players tried to shoot asteroids [3, 22]. Describing the game as fun, Ed suggested an alternative: players should blow up the asteroids instead [22].

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 [18] 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 [22] by giving them more free range movement [18]. 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, 22].

Figure 2.7. Ed Logg poses next to *Gold Asteroids*, 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, 22] – Lyle wanted the game to use raster, while Ed desired to utilise vector technology, which – according to him – had a higher resolution and allowed for more control [22]. Ed contacted the engineer Howard Delman [3, 22] to use vector technology on hardware, who had worked on *Lunar Lander* -- Atari's first game to use vector technology [22].

Asteroids' 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 [22].

As the Atari labs were open, engineers often walked between them and played games under development [22]. Consequently, Ed Logg received positive feedback from his colleagues [22]. 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, 22].

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 [22], making him one of the first game designers to realise the importance of gameplay.

Evaluation

While *Asteroids* has not enjoyed the same academic interest as *Pac-Man*, the game continues to be popular [3, 22]. Like *Pac-Man*, *Asteroids'* simple story and difficult-to-master gameplay show that

these qualities make up a successful arcade game. Additionally, Ed Logg's approach to design and simple yet addictive gameplay shows how important these aspects are when developing arcade games [23].

2.3 Arcade-Based Cat Games

Cat Trax

Cat Trax (Figure 2.8), released in 1982 [24] or 1983 [25], is a clone of *Pac-Man*, featuring a cat on the run from three canines [25, 26]. 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 [25, 26]. Once the potion wears off, the dogs start chasing the game character again [19, 20]. *Cat Trax* was developed for the Aradia 2001 [24] – an obscure home game system released in 1982 by UA.Ltd [25].

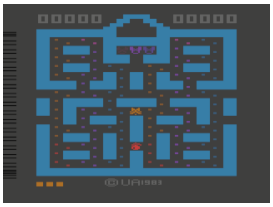


Figure 2.8. An image of the arcade game *Cat Trax* [25].

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 [27, 28], *Mappy (マッピー)* is a side-scrolling maze game featuring cartoon-inspired cats and mice [27, 28]. It ran on the *Super Pac-Man* hardware modified to support horizontal side-scrolling [27, 28].

Figure 2.9. A screenshot of the arcade game *Mappy* [27].

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 [27, 28]. To survive, Mappy must avoid the mansion's cats and traverse the building via trampolines [27, 28].

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 [23].

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 [29]. Reviews of the game describe it as cute but buggy, with the controls not working and the game being challenging from the get-go [29].

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 [7, 8]. *Hungry Space Cat* starts simple but becomes more difficult at each level, thus keeping in line with arcade game traits [8].

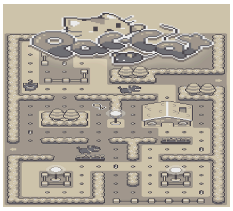


Figure 2.10. An image of the indie game *Pac Cat* [29].

2.4 Web-Based Game Project(s)

Space Cats

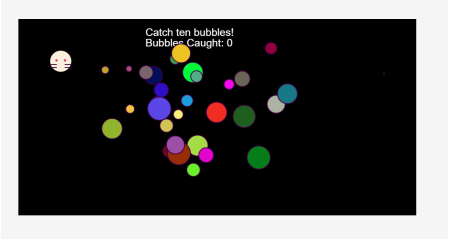


Figure 2.11. The user can play one of the two games in *Space Cat* [30].

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

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 [8]. Using a web browser affected the games' graphics in size/resolution, further restricting gameplay.

(2419 words).

3. DESIGN

3.1 Domain and Users

Hungry Space Cat's domain is arcade games with a simplistic design but challenging-to-master gameplay [8, 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 [8, 31]. 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 [31].

Taking cues from *Pac-Man* with its diverse audience [7] and continuing appeal to people of all ages [12], 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 [8, 9]. 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 [8, 9].

3.2 Core Mechanics and Gameplay

Core Mechanics

Table 2.1 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 [7, 12].

Table 2.1. The core mechanics of the game.

Action	**Action Triggered**
In difficult mode, the player moves up, down, left, or right using momentum. In easy 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* [7].
The cat eats space bugs. The cat has no weapons or defences against the enemies.	The cat gains 10 points for eating bugs, with the action similar to the player eating dots in *Pac-Man* [7], which should help players familiar with that game feel nostalgic [13].
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 (Figure 3.5).	Like in *Pac-Man*, the cat loses nine lives, but the enemies do not run away when the pellets are gone [7]. Instead, the space bugs serve as a means to proceed to the next level. Further, *Hungry Space Cat* interprets the four ghosts as different enemies.

Gameplay

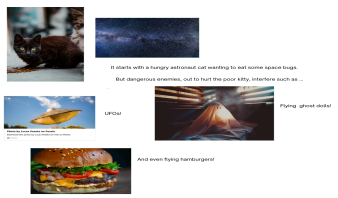


Figure 3.1. As the storyboard shows (using stock images from www.pexels.com), the concept of *Hungry Space Cat* is simple, adhering to the principle of arcade games being easy to play/pick up [8].

The gameplay of *Hungry Space Cats* – derived from a simple 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 many spacecraft floating in the scene enter from the edges of the screen before disappearing at the corner. These spaceships also shoot at the player.

3.3 Game Components

Tools

The game uses the Unity's engine [32] and the C# programming language [33]. Unity [32] is a good choice, as many tutorials offer support and guidance. *Hungry Space Cat* also uses 2D to honour the original design of arcade games [8] and lure players familiar with the genre [13].

Components

Table 2.2 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 2.2. 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.

3.4 Game Design

Menu and UI Design

Figure 3.2. The current menu emphasises readability.

The UI design of *Hungry Space Cat* prompts users to play the game, enables them to change its settings (Figure 3.2), and allows them to choose between two modes (Figure 3.3).

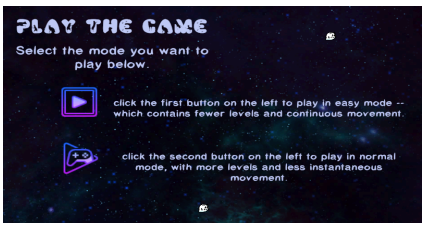


Figure 3.3. Users have the option to choose between an easy or hard mode.

Player, Pickup and Enemy Designs

Taking inspiration from *Pac-Man*, the character and level design are light-hearted [7] 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 [34], 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 2.3.

Table 2.3. The table displays some of the sprites used for the enemy and character design.

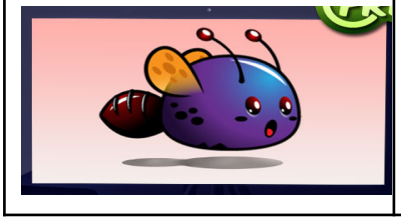
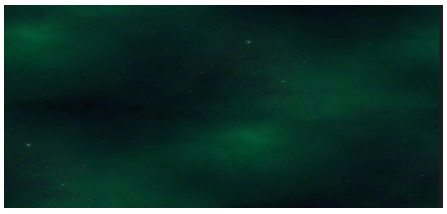
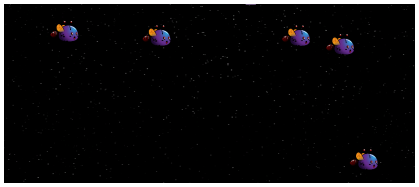
Name of Sprite	**Description**
 **Figure 3.4.** The astronaut cat [4].	As shown in Figure 3.4, the cute astronaut cat sprite matches the game's peaceful and soothing atmosphere, which matches its intention to be playable for a diverse audience [7, 13].
	The bug sprite's design – as depicted in Figure 3.5 – is endearing, emphasising the game's cuteness and non-serious nature.

Figure 3.5. The space bug that the space cat has to consume to earn points [4].	
 **Figure 3.6.** An enemy UFO that moves back and forth at the corners of the screen [34].	Like the other enemy sprites, the enemy sprite displayed in Figure 3.6 is non-threatening, fitting in with the game's theme of light-hearted cuteness.

Level Design

As shown in Table 2.4, 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.

Table 2.4. Table 2.4 shows some of the sprites used for the level design.

Name of Sprite	Description
 **Figure 3.7.** The following background has another one on top of it [5].  **Figure 3.8.** The user can turn off the effect if they find it jarring.	In Figure 3.7, 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 3.8).
 **Figure 3.9.** The lives and score text are at the top of the screen.	The number of lives (Figure 3.9) 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 [13].
 **Figure 3.10.** The pickups are spawned randomly onto the screen.	To avoid confusion, the space bugs – shown in Figure 3.10 – are the only pickup available. Their amount changes depending on the level.

3.5 Project Management

Work Plan

Figure 3.11 defines a work plan following the report's structure; the reasoning is that software development projects generally follow a similar life cycle [49]:

Project Stage	Time Frame	Key Goals	Milestone
The discovery and literature review phase represents the planning stage in a software project [86].	This stage should be completed within the first two to four weeks of April.	Based on the chosen template, a background search helps examine gaps in the market. At the same time, the literature review enhances understanding of the subject and makes informed decisions about the project's objectives.	The proposal and literature review are comprehensive and complete and have helped with the project's design.
The design stage.	The following stage should take place after the planning stage and take about two to three weeks. It should finish around the middle of May to allow for the first iteration of the development process to take place simultaneously. In the design stage, test plans that determine the success of a feature shall also be decided on. In particular, unit testing of an arcade shall be researched to help make the testing more extensive.	In the design stage, various models are considered before a final one is selected. Once the model has been chosen, a sketch of the game and some simple character designs are created. Afterwards, the critical components of the game architecture are described to help understand how everything comes together. The game's flow is also established, with diagrams present to help understand how the system and user interact.	The rough outline of the game components and the rudimentary character designs serve as a means to start developing the functioning prototype.
The implementation stage represents the building stage of a software project's lifecycle [87].	While this stage will take place after the design stage, around the middle of May, it is an iterative process that should last until August. In other words, development will consist of 'sprints' focusing on a specific project feature [87]. Each sprint will be about a week long [87].	The goal of the implementation phase is to get the project working incrementally, with the ultimate goal of completing the project on time.	Each sprint ends with the completion of the feature.
The testing stage.	As with the implementation stage, this is an iterative process after each feature is developed. This stage will be closely tied to the implementation stage, starting around the end of May and lasting till August.	The testing stage will help evaluate the project's success by combining manual testing and user play sessions. These play sessions will include other users,	The testing stage will improve the project.

Figure 3.11. The project's work plan.

Gantt Chart

Figure 3.12 depicts the project's Gantt chart and its four stages.

	A	B	C	D	E	F	G
	TASK NAME	April	May	June	July	August	September
	Planning						
	Design						
	Implementation						
	Evaluation						

Figure 3.12. The project's Gantt chart.

Evaluation Strategy

Figure 3.13 displays a test plan that performs manual regression testing to ensure the game's most critical features work. The actual result and Pass/Fail-related rows are left blank, as the test plan is for demonstrative purposes only.

Action	Expected Result	Actual Result	Pass / Fail
Upon loading the game, the user sees the menu.	The menu contains buttons, the title of the game and an image of the cat.		
The user is navigated to the first level when clicking the play game button.	The game scene is loaded correctly.		
When clicking any individual play button, the user is directed to its specific game scene.	The specific scene is loaded correctly.		
The cat cannot cross the boundaries of the screen.	The cat can only move within the boundaries of the scene.		
The bugs disappear when the space cat collides with them.	The cat can devour the bugs.		
The cat loses a life when colliding with one of the aliens.	The cat loses points.		
The SFX sound occurs when a cat picks up a coin or loses a life.	The sound clip plays with no issues.		
The score and health status text are updated whenever appropriate.	The UI displays of the scores and cat's health are updated promptly.		
When returning to the menu, the audio background of the game is muted.	The only audible background sound is the menu-specific one.		

Figure 3.13. A test plan for manual regression testing.

Along with the evaluation plan, the game relies on frequent playtesting with users of different ages and backgrounds [12]. Players will not be given questionnaires but instead encouraged to observe any bugs as they see them; this allows for an open forum, with any bug fixes emphasising those that affect gameplay [23].

(1535 words).

4. FEATURE PROTOTYPE

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.

4.2 Animation Scripts

AnimatedSprite

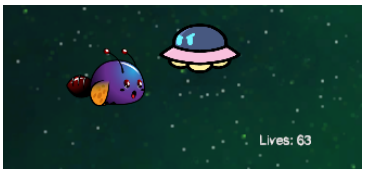


Figure 4.2. The UFO and space bug sprites use the *AnimatedSprite* script to show visual changes.

The 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 [35]. The script is attached to the game objects, such as the space cat, enemy, and bugs, as shown in Figure 4.2 and explored in Table 4.1.

Table 4.1. 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 have been met (see the *CheckAnimationFrameConditions* function in the next row below) [35].
	
	A *SpriteRenderer* component is responsible for how a sprite is rendered and depicted on a screen [36].

void CheckAnimationFrameConditions()
{
…
if (animationFrame >= 0 && animationFrame < spritesArray.Length && !PauseMenu.isPaused)
{
	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 [35]. It also checks that the game is

spriteRenderer.sprite = spritesArray[animationFrame];
}
}
	running/not paused [37].
	
	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 [35].

BackgroundSpriteScroller

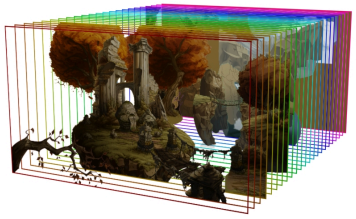


Figure 4.3. The image uses the layering method to show how parallax scrolling looks from the front [32].

The *BackgroundSpriteScroller* helps with parallax scrolling, in which background images move more slowly than forefront ones [38]. Parallax scrolling became popular in the early 1980s, along with video arcade games such as *Jump Bug* [38]. 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 [38]. The other method engages pseudo-layers using sprites drawn by hardware on top of layers [36].

To honour arcade games, *Hungry Space Cat* uses the layering process, as shown in Table 4.2:

Table 4.2. The table shows how the background scrolling effect is achieved.

Function/Code Snippet	**Purpose**

void Update()
{
if (Effects.backgroundEffectEnabled)
{
_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 [39].
	
	Afterwards, the offset is added to the background material's texture offset to create the scrolling effect [39].
	
	A static boolean checks whether this option has been triggered (the user has to choose to turn off the scrolling background effect in the *GameSettings* 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 [40].

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

Figure 4.4. The formula helps normalise the calculation of the rate at which a machine renders a frame [34]. Calculating a game object's vector in Unity with the equation returns a fixed speed, no matter the frame rate [40].

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 [41].

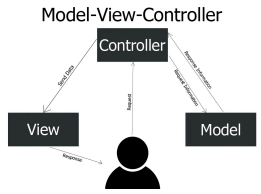


Figure 4.5. The image shows how the MVC pattern breaks down an application into separate parts that reflect the logic flow [41].

While *Hungry Space Cat* does not fully implement the MVC pattern, the scripts in the *Controller* folder are responsible for updating the enemies' movements, pickups, and the player. It also deals with how game objects react to others, especially regarding collision and ensuring that the UFOs do not leave the scene's boundaries (Figure 4.6).

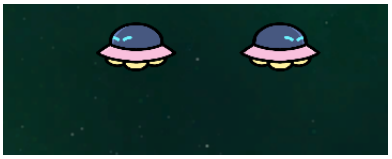


Figure 4.6. Code within the *Controller* script folder ensures that the UFOs do not leave the boundaries but change directions when 'colliding' with it.

Controller-Script-Based Functions

Table 4.3 lists some of the functions across the *Controller* scripts.

Table 4.3. The table lists some of the functions used across the *Controller* directory.

Function/Code Snippet	**Purpose**

if (other.CompareTag("Boundary") && !PauseMenu.isPaused)
{
speed = -speed;
_controllerHelper.FlipSprite(transform, _body);
…
}
…
	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 code snippet is placed inside Unity's *OnTriggerExit2D* function, which handles how objects are triggered when another one leaves its trigger collider [42].

if (other.CompareTag("SpaceCat") && !_wasEaten && !PauseMenu.isPaused)
{
_wasEaten = true;
_audioPlayer.PlayPickupClip();
_scoreKeeper.ModifyScore(pointsForBugsEaten);
	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 [44].

gameObject.SetActive(false);
Destroy(gameObject);
}
	- The deletion of the bug itself [45].
	
	The following code snippet is located in Unity's *OnTriggerEnter2D* function, which deals with how objects are triggered when they enter a trigger collider [43].

if (other.CompareTag("UFO") && !PauseMenu.isPaused)
{
…
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.

4.5 Helper Scripts

ControllerHelper

The *ControllerHelper* script contains functions related to actions that make a game object behave in a certain way under a specific condition, as shown in Figures 4.7 and Table 4.4:

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

Table 4.4. The table shows the *FollowPlayer* function.

Function/Code Snippet	**Purpose**

public void FollowPlayer(Transform target, Transform transform, float speed, int lives)
{
…
Vector2 direction = target.position - transform.position;
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));
…
	Initial game prototypes did not include rotation towards the player. Normalising the direction between the player and the target position helps give a constant distance [46]; this helps avoid bugs when rotating the flying hamburgers towards the player (Figure 4.7) [47].
	
	Further, the *Math.Atan* function [48] is applied to the x and y coordinates of the direction vectors to calculate their inverse tangent before being converted to degrees; this helps make the rotation of the enemy following the player smoother [48].

4.5 Prototype Evaluation

The current prototype is a fully fleshed-out game that has tried to fulfil the following criteria:

- ★ Straightforward story that does not require any manual or background knowledge.
- ★ Simple to understand graphics and gameplay.
- ★ Levels become increasingly complex, with the user having the option to choose how challenging the game becomes.

Early playtesters described the game as having cute and fun visuals, which adheres to the project's goal of being adorable. However, there are bugs regarding sound replaying between menus. Moreover, one or two more levels are planned for the normal mode to make the game more versatile. In addition, any bugs found in future playtesting sessions will be fixed if feasible.

(1267 words).

7. APPENDIX

Evaluation Plan

Table A.1. An evaluation plan that showcases some test cases.

Action	**Expected Result**	**Actual Result**	**Pass / Fail**
Upon loading the game, the user sees the menu.	The menu contains buttons, the title of the game and an image of the cat.		
The user is navigated to the first level when clicking the play game button.	The game scene is loaded correctly.		
When clicking any individual play button, the user is directed to its specific game scene.	The specific scene is loaded correctly.		
The cat cannot cross the boundaries of the screen.	The cat can only move within the boundaries of the scene.		
The bugs disappear when the space cat collides with them.	The cat can devour the bugs.		
The cat loses a life when colliding with one of the aliens.	The cat loses points.		
The SFX sound occurs when a cat picks up a coin or loses a life.	The sound clip plays with no issues.		
The score and health status text are updated whenever appropriate.	The UI displays the scores, and the cat’s health is updated promptly.		

When returning to the menu, the audio background of the game is muted.	The only audible background sound is the menu-specific one.	

Work Plan

Table A.2. The table below displays the game's work plan.

Project Stage	**Time Frame**	**Key Goals**	**Milestone**
The discovery and literature review phase represents the planning stage in a software project [49].	This stage should be completed within the first two to four weeks of April.	Based on the chosen template, a background search helps examine gaps in the market. At the same time, the literature review enhances understanding of the subject and makes informed decisions about the project's objectives.	The proposal and literature review are comprehensive and complete and have helped with the project's design.
The design stage.	The following stage should take place after the planning stage and take about two to three weeks. It should finish around the middle of May to allow for the first iteration of the development process to take place simultaneously. In the design stage, test plans that determine the success of a feature shall also be decided on. In particular, unit testing of an arcade shall be researched to help make the testing more extensive.	In the design stage, various models are considered before a final one is selected. Once the model has been chosen, a sketch of the game and some simple character designs are created. Afterwards, the critical components of the game architecture are described to help understand how everything comes together. The game's flow is also established, with diagrams present to help understand how the system and user interact.	The rough outline of the game components and the rudimentary character designs serve as a starting point for developing the functioning prototype.
The implementation stage represents the building stage of a software project's lifecycle [49].	While this stage will take place after the design stage, around the middle of May, it is an iterative process that should last until August. In other words, development will consist	The goal of the implementation phase is to get the project working incrementally, with the ultimate goal of completing the project on time.	Each sprint ends with the completion of the feature.

	of ‘sprints’ focusing on a specific project feature [50]. Each sprint will be about a week long [50].		
The testing stage.	As with the implementation stage, this is an iterative process after each feature is developed. This stage will be closely tied to the implementation stage, starting around the end of May and lasting till August.	The testing stage will help evaluate the project's success by combining manual testing and user play sessions. These play sessions will include other users,	The testing stage will improve the project.

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

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] 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

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

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

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

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

[19] 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

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

[21] 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

[22] 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

[23] 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

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

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

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

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

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

[29] 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

[30] HedonisticOpportunist. 2024. SPACE CATS - A GAME WEB APPLICATION. (January 2024). Retrieved May 12, 2024 from https://github.com/HedonisticOpportunist/Space-Cats

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

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

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

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

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

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

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

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

[39] 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

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

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

[42] Unity Documentation. MonoBehaviour.OnTriggerExit2D(Collider2D). Retrieved from https://docs.unity3d.com/ScriptReference/MonoBehaviour.OnTriggerExit2D.html.

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

[44] 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

[45] 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

[46] VoidNUL. 2021. What is the correct way for an enemy to chase player?. (February 2021). Retrieved March 30, 2024 from https://stackoverflow.com/questions/65784085/what-is-the-correct-way-for-an-enemy-to-chase-player

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

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

[49] Alexandra. 2024. What Is SDLC? Understand the Software Development Life Cycle. (February 2024). Retrieved March 20, 2024 from https://stackify.com/what-is-sdlc

[50] Wrike. What Is a Sprint in Agile?. Retrieved from https://www.wrike.com/project-management-guide/faq/what-is-a-sprint-in-agile/.