UNIVERSITY "ALEXANDRU IOAN CUZA" OF IAȘI

FACULTY OF COMPUTER SCIENCE

BACHELOR THESIS

Cat-astrophic Invasion

Proposed by

Alexandra-Mihaela Panaite

Session: July, 2025

Scientific coordinator

Conf. dr. Anca Vitcu

UNIVERSITY “ALEXANDRU IOAN CUZA” OF IAȘI

FACULTY OF COMPUTER SCIENCE

Cat-astrophic Invasion

Alexandra-Mihaela Panaite

Session: July, 2025

Scientific coordinator

Conf. dr. Anca Vitcu

Avizat,

Îndrumător lucrare de licență,

Conf. dr. Anca Vitcu.

Data: 01.06.2025 Semnătura: .....

Declarație privind originalitatea conținutului lucrării de licență

Subsemnata **Panaite Alexandra-Mihaela** domiciliată în **România, jud. Iași, mun. Iași, Sos. Națională, nr. 42F, bl. A4, et. 4, ap. 3**, născută la data de **13 octombrie 2003**, identificată prin CNP **6031013374539**, absolventă a **Universităţii „Alexandru Ioan Cuza" din Iaşi, Facultatea de Informatică**, specializarea **informatică în limba engleză**, promoția 2025, declar pe propria răspundere cunoscând consecințele falsului în declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art. 143 al. 4 și 5 referitoare la plagiat, că lucrarea de licență cu titlul ***Cat-astrophic Invasion*** elaborată sub îndrumarea domnului **Conf. dr. Anca Vitcu**, este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licenţă să fie verificată prin orice modalitate legală pentru confirmarea autenticității, consimțind inclusiv la introducerea conţinutului său într-o bază de date în acest scop. Declar că lucrarea de faţă are exact acelaşi conţinut cu lucrarea în format electronic pe care profesorul îndrumător a verificat-o prin intermediul software-ului de detectare a plagiatului.

Am luat la cunoştinţă despre faptul că este interzisă comercializarea de lucrări ştiinţifice în vederea facilitării falsificării de către cumpărător a calităţii de autor al unei lucrări de licenţă, de diplomă sau de disertaţie şi în acest sens declar pe proprie răspundere că lucrarea de faţă nu a fost copiată, ci reprezintă rodul cercetării pe care am întreprins-o.

Data: 01.06.2025 Semnătura: .....

Declarație de consimțământ

Prin prezenta declar că sunt de acord ca lucrarea de licență cu titlul ***Cat-astrophic Invasion***, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test, etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza" din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Absolvent **Alexandra-Mihaela Panaite**

Data: 01.06.2025 Semnătura: .....

Anexa IV

ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, ***Panaite Alexandra-Mihaela***.

Încheierea acestui acord este necesară din următoarele motive:

Îmi doresc să păstrez drepturile de autor asupra aplicației dezvoltate întrucât intenționez să continui dezvoltarea și îmbunătățirea acesteia și după examenul de licență, pentru a o include în portofoliul meu personal, ca exemplu relevant al competențelor acumulate în cadrul studiilor.

Iași, *data* 01.06.2025

Profesor îndrumător ***Anca Vitcu***

_____

(semnătura în original)

Absolvent ***Panaite Alexandra-Mihaela***

_____

(semnătura în original)

Decan,
Prof. Dr. Lenuța Alboaie

Contents

Motivation

My choice for the thesis topic represents a combinations of personal passion for this specific genre of video games, space top-down shooters, and my professional desire to gain and apply the knowledge needed to create innovative projects. Since my childhood, video games have been not only a form of entertainment, but also a source of inspirations that awakened my curiosity about the technology behind them.

Games such as ”Chicken Invaders” or ”Galaga” had a particularly strong influence on me in my childhood years. I was impressed by the fact that these games with simple designs and easy-to-understand controls still manage to offer a pleasant and captivating gameplay experience. Over time I began to wonder how were these games systems created from a technical perspective. This curiosity eventually led me to pursue my studies in Computer Science, thus had the opportunity to learn the basics of oriented-object programing, analyze artificial intelligence algorithms and familiarize myself with UX/UI design concepts.

But after all, something about the gameplay experience in my favorite genre of games was missing: the games had an almost cold atmosphere at times, lacking elements to bring a level of humanity. I realized how important it is to have an interactive NPC that is more than a decorative element or additional weapon. Thus I added in my project a Buddy AI companion, with a dynamic personality and reactions based on the context of the game, an element that brings, beside utility, more depth and originality to the gameplay experience.

Contributions

The Cat-astrophic Invasion project is the result of a long-term effors, during wich I wanted to create not only a functional game, but also an authentic and fun gaming experience that reflects both the technical skills I acquired during my studies as well as my passion for videogames.

My direct contributions include:

- **The development of the main game concept:** the idea of the game and the theme, the characters and the inegration of the inteligent helper NPC
- **Game mechanics and creating dynamic enemies:** I have designed a game system based on levels that get progressively difficult and a final boss fight, the enemies are procedurally generated and have different attack patterns.
- **Visual design of the game and User Interface Design:** I created most of the design and UI elements including character sprites, background elements, feedback elements and menus using Adobe Photoshop in a 2D style.
- **Buddy helper AI system implementation:** I created a NPC with adaptive behaviour based on the game context (player's health, number of enemie)
- **Testing and validating the game from a technical and usability perspective.**

My main goal was to create a game that reflects my creative and technical efforts and that is interactive and well-designed, enough to be considered a real and relevant game in the current context of the game industry.

I would also like to express my gratitude to Professor Dr. Anca Vitcu, my thesis coordinator for the support and guidance she provided me during the development of the application. Her valuable suggestions and constant feedback helped me grow and complete this project

Introduction

This thesis presents the design, development and testing of a 2d top-down (or "shoot'em up") game called "Cat-Astrophic Invasion", built using Unity game engine. Throughout this paper, both stages of game development - from design to implementation - will be presented along with important technical aspects such as the application architecture, game logic, technologies used, interactions between game objects and how is Unity used to integrate all these elements to create a complete and interactive video game.

This project aims not only to implement a functional software product, but also to apply basic concepts of computer science such as object-oriented programming, AI modeling, 2D animation and graphical interface design using open source tools. To have a memorable gameplay experience, the game has elements such as hand-drawn graphic assets, visual and sound effects and an intuitive user interface.

One of the main elements of the game is the artificial intelligence "mood based" system implemented for Buddy. It implements the basics of a Finite State Machine (FMS) and defines four main emotional states: Calm, Aggressive, Scared and Supportive. The player is also able to communicate directly with Buddy, thought text messages, using the ChatBox implemented using Gemini via Google Studio AI - an element that brings even more originality and creates a dynamic and interactive atmosphere. The transition from one state to another is defined by game events such as a decrease in the player's health or the approach of a large group of enemies. This behavior that adapts to the game context, enhances the gameplay experience and brings an additional narrative dimension, that maintains the player's interest throughout the game.

Chapter 1

Similar Games

1.1 Type of application implemented

The proposed application I implemented is a 2D top-down shooter game inspired by the classic Chicken Invaders style, but with an original theme and modern elements. Players control a character in a world invaded by aliens, and the main objective is to eliminate waves of enemies. In addition to similar apps in the industry, the game introduces a pretty new element: Buddy AI – a partner controlled by a form of artificial intelligence, capable of adapting its behavior depending on the game context. Buddy can take on four different moods: calm, aggressive, scared or supportive. These moods are determined by factors such as the player's health level or the number of enemies in the area, providing a more complex and realistic gaming experience. Also Buddy can communicate with the player with text messages through a ChatBox implemented using Gemini via Google AI Studio. This feature adds an interactive and personal dimension to the story. This makes Buddy a smart NPC character with his own personality, who can respond accordingly to player's messages.

1.2 Space Invaders (1978)

Space Invaders[1] in a classic arcane game created by the video game developer and engineer Tomohiro Nishikado and it was released in Japan in 1978 by Taito Corporation (a Japanese company that specializes in video games, toys, arcade cabinets, and game centers). This game is considered a pioneer of the fixed shooter genre (a

[1] https://en.wikipedia.org/wiki/Space_Invaders

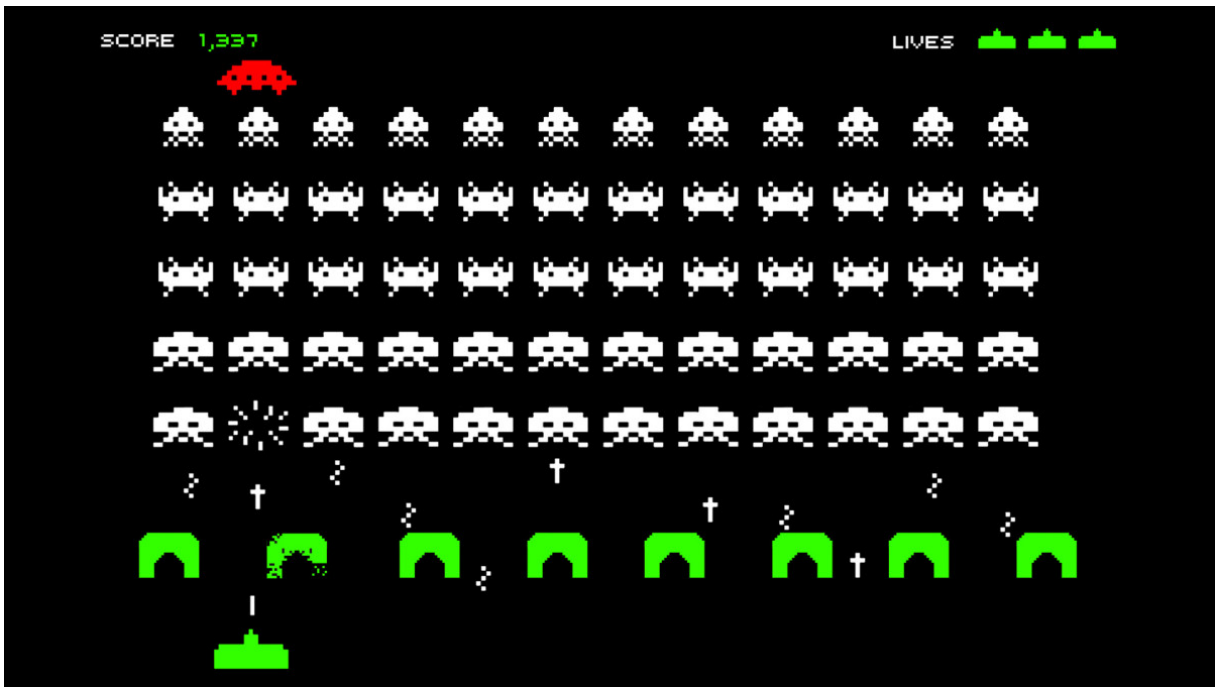

Figure 1.1: Screenshot from " Space Invaders (1978)".

subgenre of top-down shooter games, the player is restricted to a single axis of movement) and one of the most influential video games in history. The gameplay is simple: the player controls a laser that moves horizontally and the goal is destroying gradually descending rows of aliens, using destructible shields for protection. As enemies are eliminated, their speed increases, amplifying the difficulty. It was one of the first games to introduce the concepts of maximum scores, limited munition, multiple lives and continuous music, creating a tense interactive gaming experience. The game had a huge impact on the industry: in Japan it became so popular that it led to a temporary shortage of 100 yen coins. By 1982, Space Invaders had generated profits of over 3.8 billion dolars, becoming the most successful video game of its time and had contributed decisively to the development and popularization of the video game industry worldwide.

1.3 Chicken Invaders Series

Chicken Invaders[2], another 2D top-down shooter game, where the player controls a ship threatened by space chicken invasions, the main inspiration for my game

[2] https://en.wikipedia.org/wiki/Chicken_Invaders

Figure 1.2: Screenshot from "Chicken Invaders"

and one of the longest running series of video games developed. The series was created in Greece by Konstantinos Prouskas, and the first game was released on 24 July 1999 as a parody of the original Space Invaders. Although initially inspired by the classic Space Invaders, Chicken Invaders series manages to bring new and interactive elements such as weapon upgrades, power-up systems, themed missions, co-op modes, and a comical visual style. It also presents a very primitive form of buddy AI NPC, in a few levels at the beginning of the game the player has a little helper that shoots occasionally. The humor, wordplay and satirical theme not only provide a relaxing experience, but also a strong, memorable identity for players. For me, Chicken Invaders was a defining game that marked my childhood in all the good ways, it was one of my first interactions with the video game universe next to other worldwide known classics. For this reason, I wanted to pay my homage to this series by developing my own application that aspires to reach the same level of charm. I tried to include humorous elements, a pleasant design, and an original NPC companion, to recreate a gaming experience that transports players to the same cozy and fun atmosphere that I had in my childhood years.

1.4 Galaga

Figure 1.3: Screenshot from ”Galaga”.

Galaga[3] is a space shooter video game released by the Japanese company Namco. It is the direct successor to the 1979 game Galaxian and is considered to be one of the most influential and loved space shooter games. The player controls a spaceship and the goal is to eliminate the waves of enemies that move in formations and attack like kamikazes, creating explosive charges and attempting to capture the player's ship. This game brings a series of new unique elements for its genre at the time one of them being The Dual Fighter scene: if the enemy captures the player's ship, the player can recuperate it and combine it to his active ship creating a ”Dual Fighter” that shoots two missiles. Also the enemies don't just stant in a patter, like in Space Invaders, but move in a rotating ghraphic patters abd create dinamic attack ways.

1.5 Geometry Wars

Geometry Wars[4] is a videogame developed by Bizarre Creations that was released in 2003. It's a twin-stick shooter game (subgenre of top-down shooter, this means that

[3] https://en.wikipedia.org/wiki/Galaga

[4] https://en.wikipedia.org/wiki/Geometry_Wars

there are two controls: one for moving one for shooting) in wich the player is moving in a2D arena where he as to survive and eliminate as much enemies waves as possible. The design is retro-futuristic with intense neo colors and explosive visual efects.

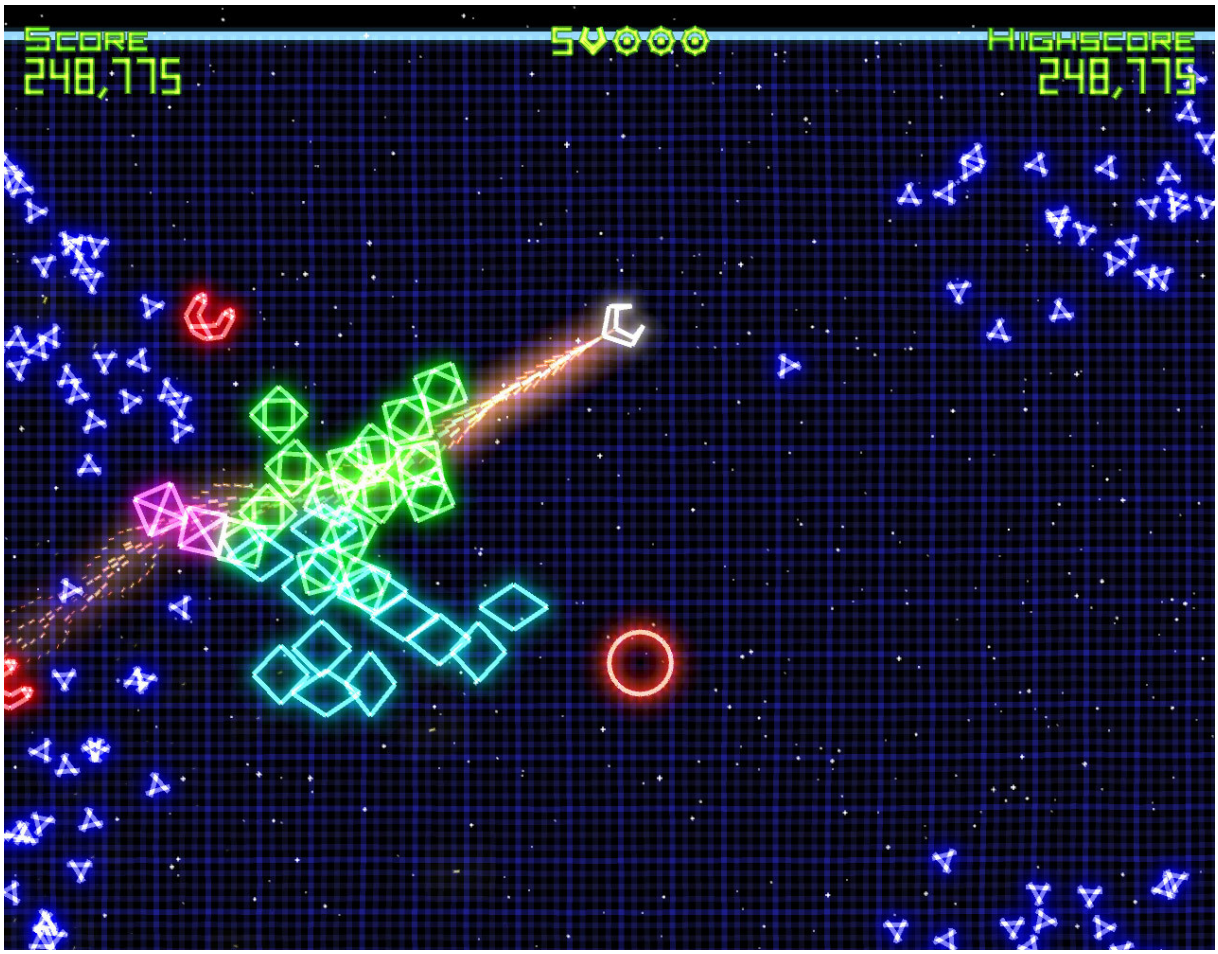

Figure 1.4: Screenshot from ”Geometry Wars”.

Chapter 2

Technologies Used

2.1 Unity - game engine

Unity[1] is one of the most well-known platforms used for developing 2D and 3D video games. It offers a balance between high performance, advanced features and ease of use even for beginners with no previous experience in game development, thanks to the multiple sources and tutorials that can be accessed easily. It is also one of the most accessible open source technologies and it is free for small teams and independent developers.

For this project is used the Unity 6000.0.46f1 version which is a stable and optimized version suitable for game demos.

When creating a 2D top-down shooters game , Unity provides all the features need for basic and advanced game mechanics:

- Controlling the X and Y axes of the main character moves using Rigidbody2D component;
- Shooting system - where each projectile is a pre-rendered object that can be dynamically created and animated;
- Collision detections using BoxCollider2D, CircleCollider2D, OnTriggerEnter2D or OnCollisionEnter2D;
- Implementing enemies behavior using an artificial intelligence system based on distance, player detection and collision response;
- Visual effects such as movement and animations (Animator, Animation Clips);

[1] https://en.wikipedia.org/wiki/Unity_(game_engine)

- Sound effects such as background musing and laser sounds using AudioSource or AudioClip.

These components can be integrated in Unity without installing any external libraries, simplifying the implementation.

Unity also allows for easy code integration with C#, the engine's main scripting language. It is also compatible with multiple third-party technologies such as Cloud services (Google Cloud), Artificial Intelligence and third-party APIs (Gemini API).

Everything mentioned above make Unity game engine the perfect fit for implementing the project's requirements.

2.2 C# - programming language

C#[2] is a object-oriented programing language developed by Microsoft as part of the .NET platform. It is one of the most used languages for web, mobile and game development. C# is the main scripting language and is fully supported for writing game logic.

Unity is designed to run in C#, that means that developers have full access to the Unity API, which is designed to be used in C#. All aspects of the game engine - movements, collisions, animations, keyboard input, sound effects- can be controled in C# using components provided by Unity.

Additionally, for students or developers that have a knowledge base from programming languages such as Java or C++, the transition to C# is easier. C# benefits from extensive documentation and sources of information - Stack Overflow, Unity forums and YouTube tutorials - that can be easily access to find answers and examples.

Therefore, choosing C# for the video game development in Unity was a natural decision.

2.3 Google Cloud and Gemini API

Gemini[3] is a set of AI models that can create and understand conversations with clarity, logic and context. Gemini is integrated with Google Cloud services to support

[2] https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[3] https://ai.google.dev/gemini-api/docs

developers who want to build chat applications, virtual assistants or other dynamic features.

In this project Gemini API is used to create a chat system that can be used by the player to interact with Buddy in real-time and provide advice, motivational messages or other responses from Buddy.

The reasons Gemini API is the suitable choice for this project are:

- The API provides a user-friendly REST interface, clear documentation and many usage examples in C# programming language
- Google Cloud support: since some components are already hosted in Google Cloud, Gemini API integration is secured using API key authentication and service accounts.
- Game Personalization: Gemini can emit different reactions based on the game state, player's progress or action.
- Easy to extend: Call states can be improved over time without changing the code but only by adjusting query parameters.
- Messages processing in Google Cloud: the Gemini API analyzes the context of the conversation and give an relevant response.
- Fast response time: for the player to have an authentic and dynamic experience while using the ChatBox the answers need to be received as fast as possible.

2.4 Adobe Photoshop

In this project, Adobe Photoshop played an esential role in the visual design process and was used to create and edit all the graphics in the game. The tool was used to create everything from user interface elements (buttons, icons, menu) to characters, enemies and backgrounds.

To properly organize the images, each object was created on a separate layer, and the final result was exported in a PNG format with transparent background for direct use in the Unity engine.

Photoshop is a world-known professional image editing and graphic design tool. It's high level of complexity an versatility made it the best choice.

Chapter 3

Application architecture and implementation process

3.1 General description of the application

This project implements a 2D top-down shooter game developed in unity. The main objective is surviving the battle with a alien invasion in space. The games adopts a three level progression system, with gradually increasing difficulty with each level. During the game players will continue to receive help and feedback from Budy, the AI partner, who provides visual, emotional and informational support.

The first game element the player interacts with, is the Main Menu screen, where he can Start the game, access High-Score archive, access the About screen presenting the game's story or exist the game, he also has the option to mute or unmute the background music.

Pressing the "Start" will directly enter the first level of the game. Here, players must survive for one minute against waves of procedurally generated enemies that appear at random times an positions to follow and attack the player. The difficulty is low to help the player familiarize with the controls and the overall dynamics of the game. This requires the player to dodge projectiles, eliminate enemies and maintain his health. Budy has the role to help the player by attacking enemies, providing basic guidance on movements, shooting and enemy behavior. If the player successfully pass the first level they will enter the second one, which becomes significantly difficult, besides waves of enemies, players must also dodge floating obstacles such as asteroids that appear and move randomly. These elements increase the complexity of

the game.The next level is also the final one, in which the player will engage in a final BossFight with a giant spider. This enemy cannot be defeated only by shooting continuously, but only by accurately hitting his weak points.

Whether the player defeats the BossLevel or is defeated in the game, the gaming experience ends with the GameOver screen where players can choose to enter their name to save their final score.

3.2 Application's Design

Initially, the artistic direction of the game was focused on pixel art ilustrations, which has become a common choice for indie games development, due to its simple look with a nostalgic effect. The initial version of the sprites was created in Photoshop using a pixel grid and hard brushes, inspired by classic games from the 1990s. Each character and background was hand-drawn with individual pixels for a more authentic look.

However, after initial feedback, I came to the conclusion that the pixel art style reduced the visual appeal and user experience. It was brought to my attention that players wish for a modern and aesthetic design.

In response to this feedback the design process took a different direction, from pixel art to hand-drawn 2D illustration style, which give the game a warmer and at the same time a more modern look. This change in visual design had a significant influence on the way the game is perceived, the hand-drawn characters and visual elements have even more personality now, thus creating a better gaming experience.

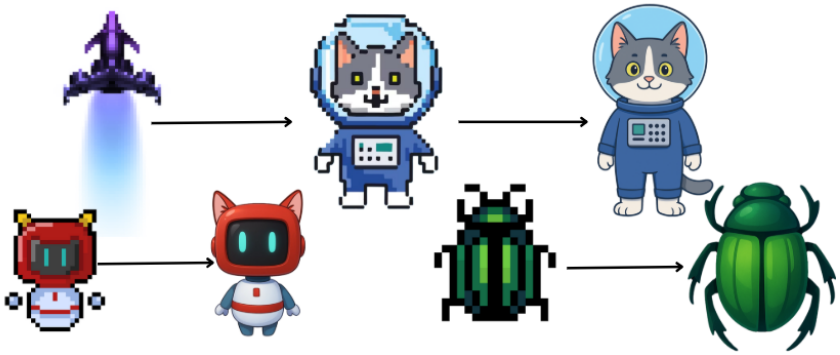

Figure 3.1: Game characters first and final appearance

3.3 Application's Architecture

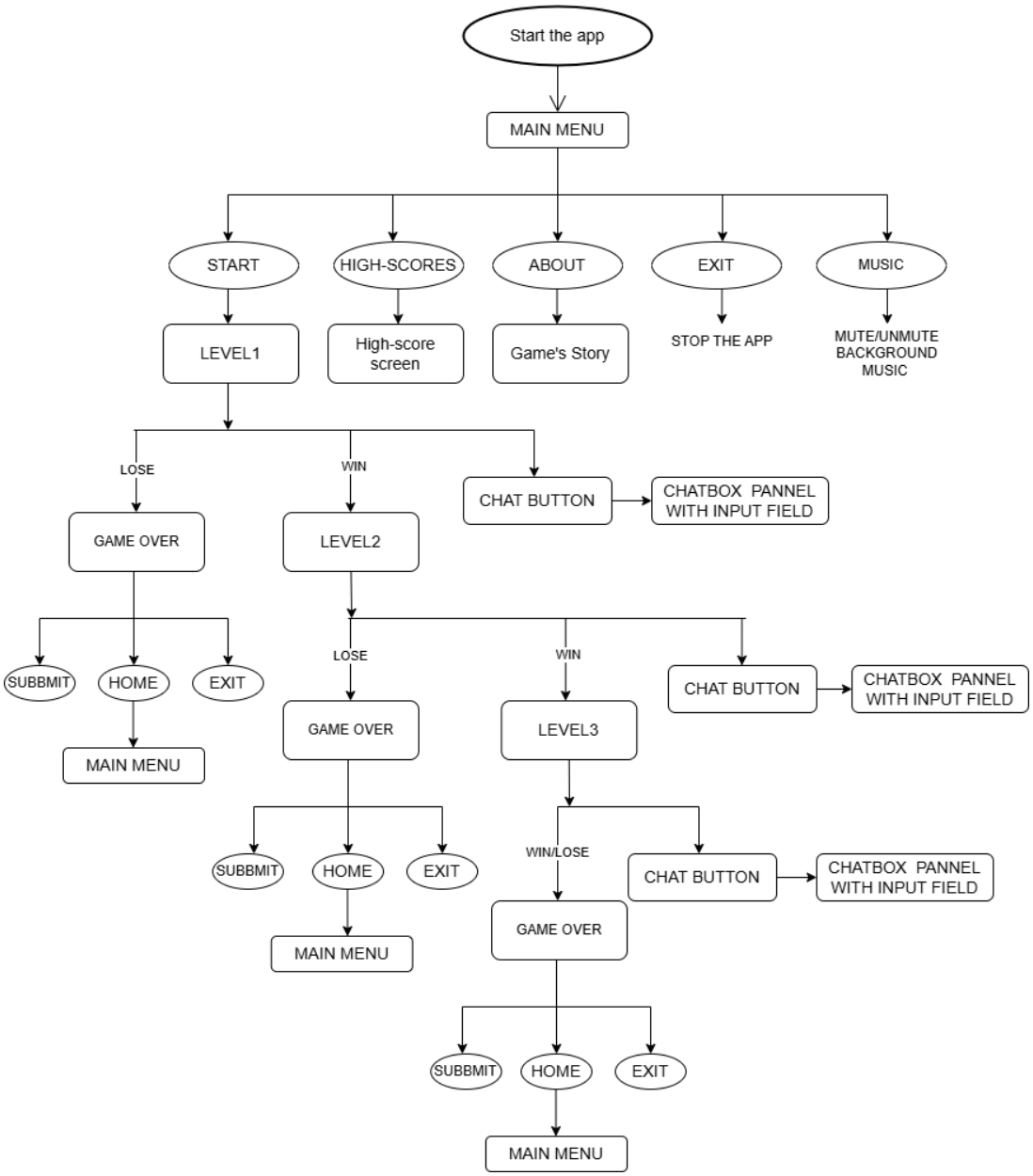

Figure 3.2: User Flow Diagram.

The software architecture is organized around the logical interaction between the user and the game. The diagram described below shows the entire user-flow, from the start through all possible scenarios and options until the game is finished.

This modular and decisional structure ensures a consistent and intuitive experience, allowing the user to play and manipulate the components in a consistent manner.

3.4 Implementing the game components

3.4.1 Player

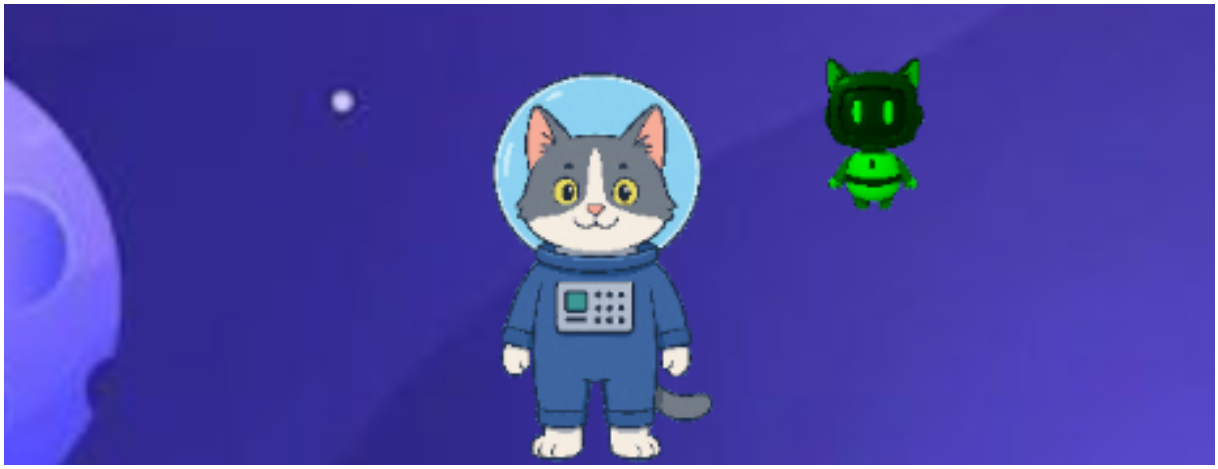

Figure 3.3: Felix with Buddy

The game's protagonist, Felix, is designed and implemented as a player-controlled spaceship, aiming to provide an intuitive, enjoyable and functional gaming experience. Its implementation is based on the principles of modularity and separation of responsibilities, which are essential for video game development.

Felix is logically divided into two main parts:

- MoveNAVA: responsible for movement and attacks
- PlayerHealth: responsible for life and damage management

Main input loop (Update() in MoveNAVA): This is the "heart" of the game, without it the player cannot be controlled at all.

```
void Update()
{
    HandleMovement(); // move ship
    HandleShooting(); // fire laser
}

void HandleMovement()
{
    if (Input.mousePresent && Input.GetMouseButton(0))
```

```
    {
        // move to mouse position
        // ... (convert coordinates and move)
    }
    // handle keyboard movement...
}

void HandleShooting()
{
    if (Input.GetKeyDown(KeyCode.Space) ||
    ↪  Input.GetMouseButtonDown(1))
    {
        ShootLaser(); // play sound and spawn laser
    }
}
```

This separation allows for easy testing, maintenance and extension, following good object-oriented programming practices.

For better accessibility, the player control supports two methods:

- Using the mouse: By pressing the left mouse button instantly teleports the spaceship to the cursor position. Coordinates are converted from screen space to game coordinates using Camera.main.ScreenToWorldPoint(). The Z axis is set to 0 to maintain position in a 2D plane.
- When using the keyboard: Use the ”Horizontal” and ”Vertical” axes (WASD / arrows). The movement is continuous, and the speed is affected by moveSpeed and Time.deltaTime to ensure consistency between images.

When it comes to shooting mechanics, Felix can attack with lasers:

- Shots are fired by pressing the spacebar or by clicking the right mouse button.
- Each laser is pre-created at the current position of the player.
- The corresponding sound effect is played using the SoundManager.

This system allows for quick reactions and auditory feedback for each action.

Health and Damage System (PlayerHealth):

Felix can take damage from: Collision with obstacles (asteroids) or collision with enemy projectiles.

Each effect reduces health by a configurable amount (damageAmount) and communicates with the GameManager to update the overall game state.

To prevent continuous life loss due to repeated collisions, a period of temporary invulnerability is activated after each hit, implemented using a coroutine that blocks damage for the duration of the invincibility (invincibilityDuration). It generates a flash effect using a SpriteRenderer, providing visual feedback to the player.

Temporary invincibility system:

```
private IEnumerator InvincibilityFrames()
{
    isInvincible = true;

    if (spriteRenderer != null)
    {
        for (int i = 0; i < 5; i++)
        {
            spriteRenderer.enabled = false;
            yield return new WaitForSeconds(...);
            spriteRenderer.enabled = true;
            yield return new WaitForSeconds(...);
        }
    }

    isInvincible = false;
}
```

A unique sound is played with each hit to emphasize the negative impact and visually signal to the player that one of his lives has been lost.

Interaction with other systems:

- GameManager: PlayerHealth communicates with GameManager to update remaining lifetime, determine game over time and for possible synchronization with user interface and other global systems.

- SoundManager: Both components (MoveNAVA and PlayerHealth) use SoundManager (a one-time model); MoveNAVA plays a gunshot sound and PlayerHealth plays a ”getting hit” sound.

2D Physics and Tags: Collisions are handled using OnTriggerEnter2D and OnCollisionEnter2D and the game objects are identified using tags, ”Enemy_Laser”, ”Enemy”.

The Felix player implementation is an important part of the game's development. Thanks to the modular design and careful integration with other systems (sound, physics, user interface), a coherent, stable and adaptable behavior is achieved.

3.4.2 Enemies

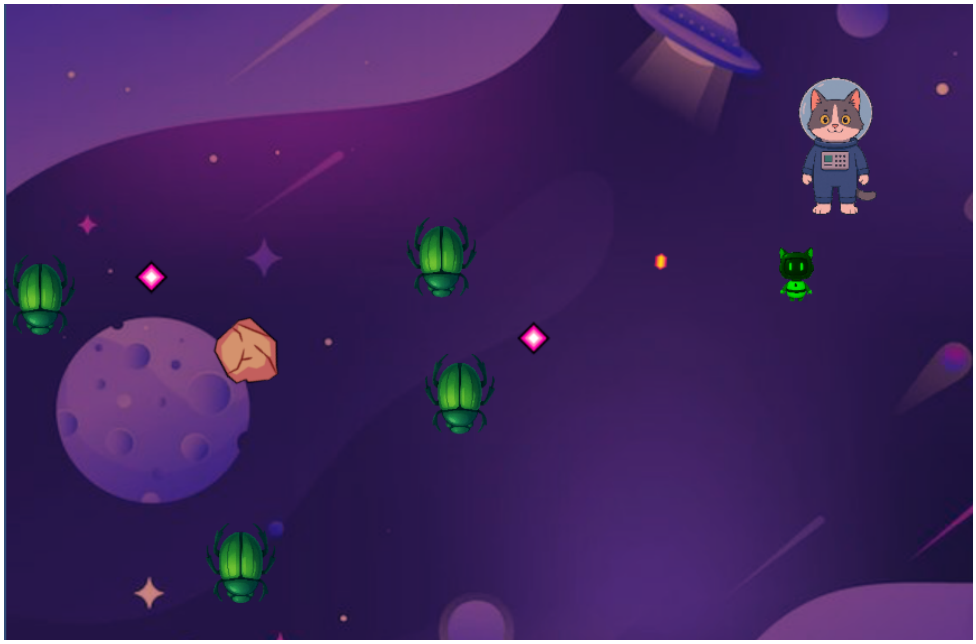

Figure 3.4: Enemies attacking - screenshot from Level 2

The enemy system is a critical element in creating an engaging and challenging game experience. In the game, enemies are designed to respond to the player's actions, move efficiently through the game environment and provide clear visual and auditory feedback, thus contributing to the game dynamics and balance.

The enemy system consists of two main components:

- Enemy.cs: Handles movement logic, tactical behavior, and attacks.
- EnemyHealth.cs: Handles the health system, damage feedback and enemy destruction.

This separation allows for modular development and efficient code maintenance, following the principles of object-oriented programming.

The enemy uses a zone system to determine its behavior:

- Pursuit Zone: If the player is too far away, the enemy moves closer.
- Attack Zone: If the distance is optimal, it stops and shoots.
- Retreat Zone: If the distance is too close, it retreats.

This logic is implemented by the CalculateBaseMovement() method, which returns a direction vector corresponding to the player's position:

```
private Vector2 CalculateBaseMovement()
{
    float distance = Vector2.Distance(transform.position,
    ↪  target.position);
    Vector2 dir = (target.position - transform.position).normalized;

    if (distance > stoppingDistance) return dir;
    if (distance < retreatDistance) return -dir;

    return Vector2.zero;
}
```

Obstacle Avoidance (Asteroids): The presence of asteroids complicates navigation. The enemy uses Physics2D.OverlapCircleAll to identify nearby obstacles and calculates a vector to avoid them:

```
private Vector2 CalculateAsteroidAvoidance()
{
    Vector2 avoid = Vector2.zero;
    var asteroids = Physics2D.OverlapCircleAll(transform.position,
    ↪  asteroidDetectionRadius);
    int count = 0;

    foreach (var col in asteroids)
    {
```

```
        if (col.CompareTag("Asteroid"))
        {
            // compute repulsion force...
            avoid += ...;
            count++;
        }
    }

    if (count > 0) avoid = avoid.normalized * avoidanceForce;

    return avoid;
}
```

Intelligent Attack System:

When the player is within range, the enemy fires projectiles at regular intervals. These attacks are accompanied by sound effects using the SoundManager, enhancing the experince and impact of combat.

Health System and Visual Feedback: The EnemyHealth component manages health reduction and triggers enemy death when health reaches zero:

```
public void TakeDamage(int damage)
{
    currentHealth -= damage;
    SoundManager?.Instance?.PlayEnemyHit();
    StartCoroutine(FlashEffect());

    if (currentHealth <= 0) Die();
}
```

To indicate when an enemy is hit, a white flash effect is used by modifying the sprite color:

Enemy Death: When an enemy dies, the following effects occur: distinct explosion sound is playedand the enemy is destroyed to free system resources.

Also, the enemies that leave the camera view are automatically destroyed. Null checks are implemented to avoid runtime errors.

Integration with Other Systems

- SoundManager: Fully integrated for audio feedback on hit and death.
- Physics2D: Used for projectile and obstacle detection via triggers.
- GameManager (optional): Can be extended to control the number of active enemies and manage difficulty scaling.

3.4.3 Buddy - Intelligent NPC

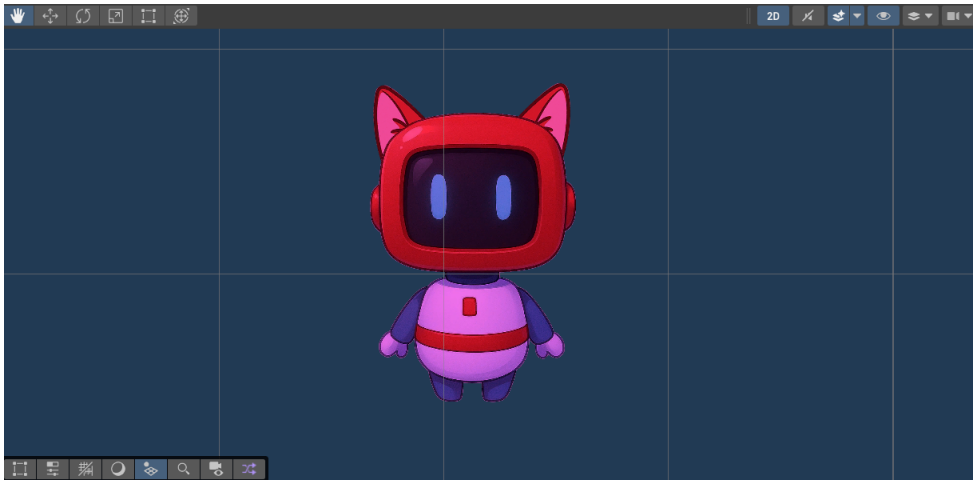

Figure 3.5: Buddy

In modern video games, artificial intelligence (AI) companions play a vital role in improving the player's gaming experience. These virtual characters are more than just background elements, they actively contribute to the gameplay. A well-designed AI companion can enhance the game's dynamics by offering assistance, reacting to the game environment, and adapting to changing circumstances in a realistic manner.

Next it will be presented the development process of the intelligent AI assistant named Buddy, specifically created to interact with both the player and the game environment. Buddy's system focuses on context-aware behavior and adaptability, allowing it to modify its actions in real time in response to threats, the player's health, or other game events. To achieve this, the system combines multiple AI strategies: Finite State Machines (FSMs) for global behavior management, behavior trees for specific task execution and decision-making logic rooted in real-time game context.

The internal architecture of the Buddy IA system is structured into multiple interrelated modules, each responsible for a distinct aspect of the AI's operation. This modular design promotes both clarity and flexibility in development and maintenance.

- Finite State Machine (FSM): Serves as the core controller, dictating the general behavioral state of the buddy. It transitions between states such as Calm, Aggressive, Fearful, and Supportive based on game context.
- Contextual Decision System: Evaluates game data, including the player's health, the number and proximity of the enemies, and other game events to determine which FSM state is most appropriate.
- Simplified Behavior Tree: Includes a set of predefined behaviors for each FSM state. This ensures that Buddy performs actions in a structured and predictable manner, enhancing player expectations and gameplay flow.
- Intelligent Combat System: Implements predictive targeting mechanisms, enabling Buddy to engage with the enemies effectively and naturally
- Fluid Tracking System: Manages Buddy's spatial positioning relative to the player.

Finite State Machine and Context-Aware Transitions:

At the base of Buddy's behavior is the BudyMood enumeration, which defines the four primary emotional or functional states Buddy can adopt:

```
public enum BudyMood { Calm, Aggressive, Fear, Support }
```

Transitions between these states are governed by a context-sensitive algorithm that monitors various game variables. For instance, if the player's health drops or multiple enemies are detected nearby, Buddy may switch to a supportive or fearful state.

```
void UpdateMood() {
    // evaluate context: player health, enemy count
    if (...) currentMood = BudyMood.Support;
    else if (...) currentMood = BudyMood.Fear;
    else if (...) currentMood = BudyMood.Aggressive;
    else currentMood = BudyMood.Calm;

    if (previousMood != currentMood)
        OnMoodChanged(previousMood);
}
```

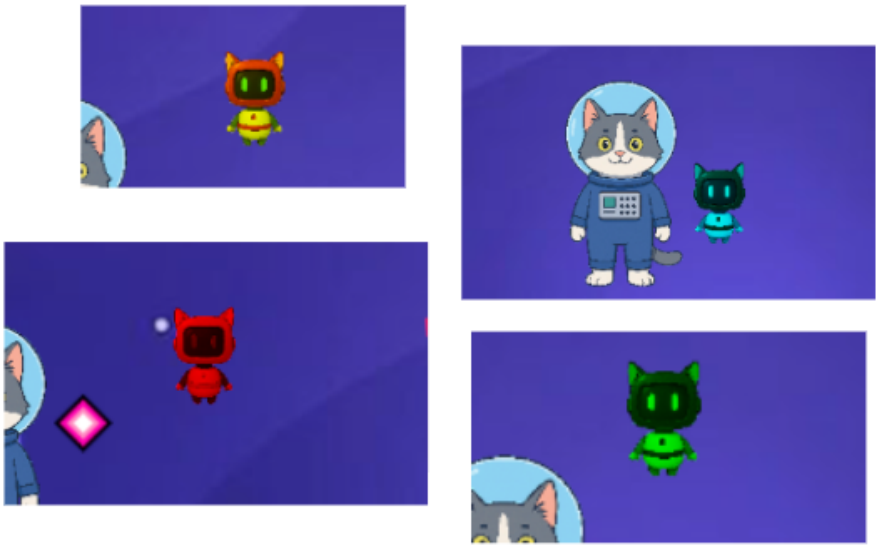

Figure 3.6: Buddy's states - Scared, Supportive, Aggresive and Calm

Once a new mood is determined, a corresponding behavior routine is executed to adapt Buddy's actions appropriately:

```
void ExecuteMoodBehavior() {
    switch(currentMood) {
        case BudyMood.Calm: CalmBehavior(); break;
        case BudyMood.Aggressive: AggressiveBehavior(); break;
        case BudyMood.Fear: FrightenedBehavior(); break;
        case BudyMood.Support: SupportiveBehavior(); break;
    }
}
```

Buddy is also capable of engaging enemies autonomously using predictive targeting. This involves identifying the closest enemy, calculating the appropriate direction and speed for projectiles, and visually adjusting bullet trajectories to simulate realistic shooting mechanics.

```
void ShootAtNearestEnemy() {
    // Select the nearest enemy
    // Create a bullet instance
    // Calculate direction and speed
    // Visually adjust the projectile
```

```
}
```

This system enhances gamesplay realism and makes Buddy seem intelligent and proactive.

To maintain its involvement and natural movement, Buddy tracks the player's position with smooth transitions using Unity's Vector3.Lerp function:

```
void Update() {
    if (target)
        transform.position = Vector3.Lerp(
            transform.position,
            target.position + offset,
            trackSpeed * Time.deltaTime);
}
```

Natural Language Command Integration

To further enhance interactivity, Buddy can interpret chat massages send by the player and change its behavior accordingly. This enables real-time control over Buddy's mood and actions:

```
void UpdateMoodFromChat(string msg) {
    // If message contains "attack" => mood = Aggressive;
    // If message contains "help" => mood = Fear;
    // ...
}
```

This feature adds depth to the player-companion relationship.

The architectural design of the Buddy system offers several notable benefits:

- Modularity: Each component is self-contained and focuses on a single responsibility, simplifying development and debugging.
- Flexibility: New behaviors, moods, or sensors can be integrated without major rewrites.
- Performance: The system avoids continuous evaluation by using periodic checks and conditional updates.

- Intuition: The state-based structure makes the AI's decisions easier for players to understand and anticipate.

The Buddy IA system exemplifies a modern, adaptive AI companion for video games, combining several intelligent systems to achieve responsive and engaging behavior. By utilizing FSMs for global state control, behavior trees for detailed action plans, and context-aware logic for decision making, Buddy can provide meaningfu support to the player. Its modular architecture allows easy expansion and fine-tuning, making it a robust foundation for future enhancements such as emotional simulation, cooperative problem solving, or advanced learning mechanisms.

3.4.4 Level 1

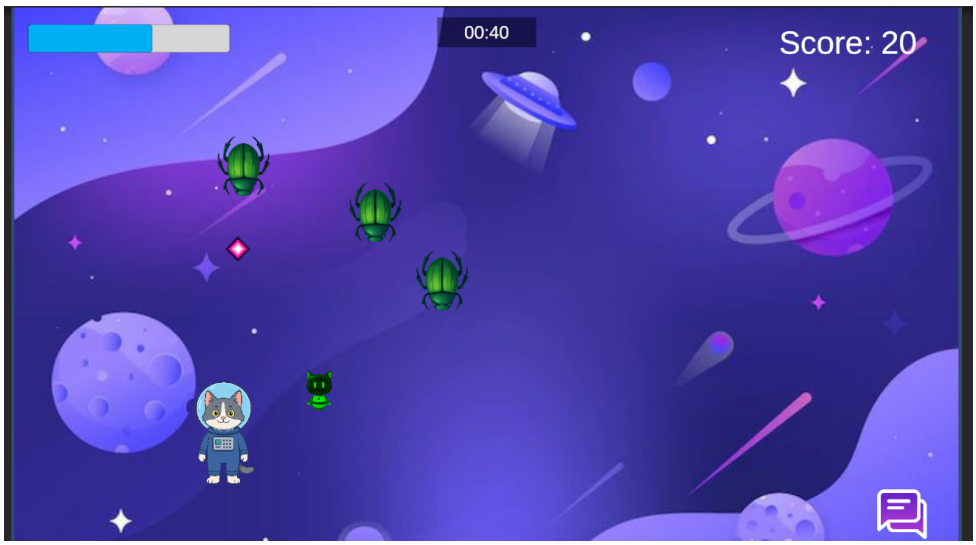

Figure 3.7: Level 1

The first level of the game is designed as an introductory stage in which the player is familiarized with the essential functionalities of the game and the controls. This level has a low difficulty, without complex obstacles or aggressive enemies, to allow the user to understand how the game works and adapt without pressure before reaching the following levels.

The goal of this level is to build the player's comfort and confidence, so that the transition to the next levels is natural and motivating. The visual design is friendly, using warm and bright colors. As the player progresses and understands the basic mechanics, a gradual visual transition to a darker red tone can be observed, which shows the increased difficulty.

As the player advances to the next level, the game visually signals this transition by changing the color palette to red tones, symbolically marking the danger and the intensification of the game.

3.4.5 Level 2

Upon reaching level two, the player is confronted with a series of dynamic obstacles in the form of asteroids. These add a new layer of difficulty, requiring the player's attention and skill.

From a technical point of view, the behavior and appearance of these asteroids is controlled by two main scripts: Asteroid.cs and AsteroidSpawner.cs. In Asteroid.cs,

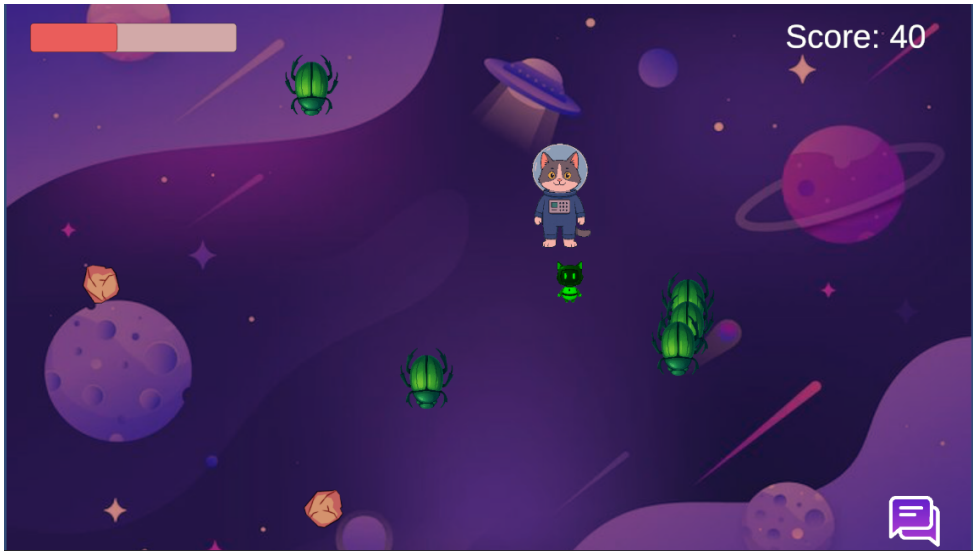

Figure 3.8: Level 2

each asteroid is an independent entity with its own direction of movement, speed and size. The possible sizes are: small, medium and large, each having a different impact on the player's health upon collision:

```
public enum AsteroidSize
{
    Small,
    Medium,
    Large
}
```

Each asteroid is initialized by the Initialize() method, which sets the direction

and speed. If an asteroid goes completely off screen, it is automatically destroyed to optimize performance.

In AsteroidSpawner.cs, the script is responsible for regularly spawning asteroids at random points off screen, on the sides or on top. At each instantiation:

- A prefab is randomly chosen from an array of prefabs
- The size is determined based on a predefined distribution
- The speed and direction are set with a small angular variation for realism

```
GameObject asteroid = Instantiate(asteroidPrefab, spawnPosition,
↪  Quaternion.identity);
asteroidScript.SetAsteroidSize(size);
asteroidScript.Initialize(moveDirection, speed);
```

In addition to the movement, the script also includes a slow rotation that provides a realistic and dynamic visual effect:

```
public class RotateAsteroid : MonoBehaviour
{
    public float rotationSpeed = 30f;
    void Update()
    {
        transform.Rotate(0, 0, rotationSpeed * Time.deltaTime);
    }
}
```

This component adds realism and contributes to the aesthetics of the game.

3.4.6 Boss Fight

To implement the Spider Boss enemy (the ”boss fight” level) in the video game i applied concepts such as game design, AI programming and handling visual and audio feedback. Next will be presented a detailed look at how the Spider Boss is implemented in a modular, scalable and intuitive way.

The Spier Boss enemy system consists of these main components:

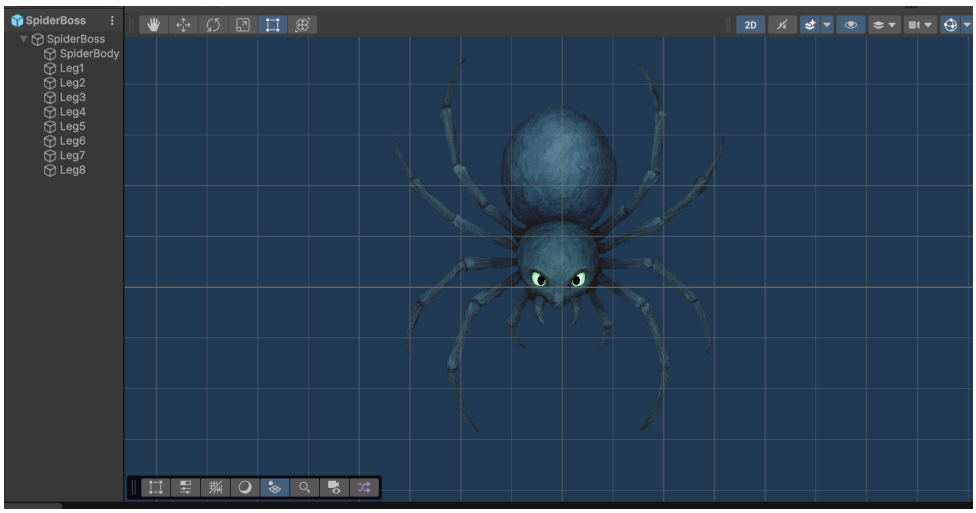

Figure 3.9: The Spider Boss Enemy.

- SpiderBoss.cs
- SpiderBodyCollider.cs
- SpiderLeg.cs

The SpiderBoss class manages the behavior of the enemy through a Finite State Machine (FSM), which follows the main phases of the game.

```
private enum BossState { Down, Fighting, Defeated }
private BossState currentState = BossState.Down;
```

The spider approaches or retreats based on the player's position, maintaining an optimal range distance.

```
void DistanceBasedMovement() {
    // Calculates the distance to the player
    // Threshold-Based Return Method(s)
}
```

This dynamic behavior increases tension and engagement during combat.

Destructible Leg System - SpiderLeg.cs

Each leg is implemented as an independent entity with a defined weak point, encouraging strategic gameplay. To support this mechanic, the code dynamically attaches a 2D CircleCollider2D component to each weak point, allowing precise detection of targeted hits. This modular approach ensures that each limb reacts individually to damage.

```
void CreateWeakPointCollider()
{
    // add a trigger collider at the weak point
    GameObject weakPointObj = weakPoint.gameObject;
    weakPointCollider =
     ↪  weakPointObj.AddComponent<CircleCollider2D>();
    weakPointCollider.isTrigger = true;
    ((CircleCollider2D)weakPointCollider).radius = weakPointRadius;

    // add a script to handle collisions on the weak point
    WeakPointHandler handler =
     ↪  weakPointObj.AddComponent<WeakPointHandler>();
    handler.parentLeg = this;
}
```

Figure 3.10: ScreebShot from Level3.

The boss becomes vulnerable only after all legs are destroyed. Until then, the body is protected. This feature is handeled by the SpiderBodyCollider.cs.

```
void Update() {
    // bodyCollider.enabled = spiderBoss.IsBodyVulnerable();
}
```

As the fight progresses, the Spider Boss dynamically changes behavior.

Adaptive Firing Rate:

```
float currentTime = Mathf.Lerp(val1, val2, ratio);
```

Modified Movement Speed:

```
movement *= (0.5f + 0.5f * ratio);
```

The balance of slower movement and faster attacks keeps players under constant pressure.

The Spider Boss can fire projectiles in circular waves.

```
void FireCircularWave() {
    // calculates angles
    // creates radially directed projectiles
}
```

Players must rely on dodging skills and precise timing to survive these sequences.

When the legs are all destroyed, the boss enters an enraged state with visual and behavioral changes.

```
IEnumerator EnragedMode() {
    // red flash + speed boost
}
```

This signals the final stage of the battle and adds dramatic tension.

Integration with Game Systems:

- Audio Feedback

```
SoundManager.Instance.PlayBossHit();
```

- Game Progression

```
gameManager.SpiderBossDestroyed();
```

These callbacks ensure smooth integration with other systems like scoring, quest progression, or achievements.

Boss Fight Flow Summary

1. Spider Boss descends dramatically into the scene.
2. Legs act as shields; the player must destroy them first by hitting their weakpoints.
3. Every leg destroyed increases rage and modifies boss behavior.
4. Once legless, the boss enters Enraged Mode.
5. The body becomes vulnerable, and the player can defeat the Spider.

The Spider Boss Level is a powerful example of modular enemy design. Using standard practices like state machines, autonomous components, hit detection, and audio/visual feedback, we can create complex and engaging combat scenarios.

3.4.7 Menus and screens

Main Menu

Figure 3.11: Main Menu Screen

Upon opening the app, the user is directed to the Main Menu screen, is a crucial component in terms of the appearance, aesthetics and user experience. From the Main Menu we can access the folowing screens:

High-Scores Screen: The High Scores screen is designed to encourage competition and motivate players to improve their performance. The top 10 scores are displayed, along with the players' names.

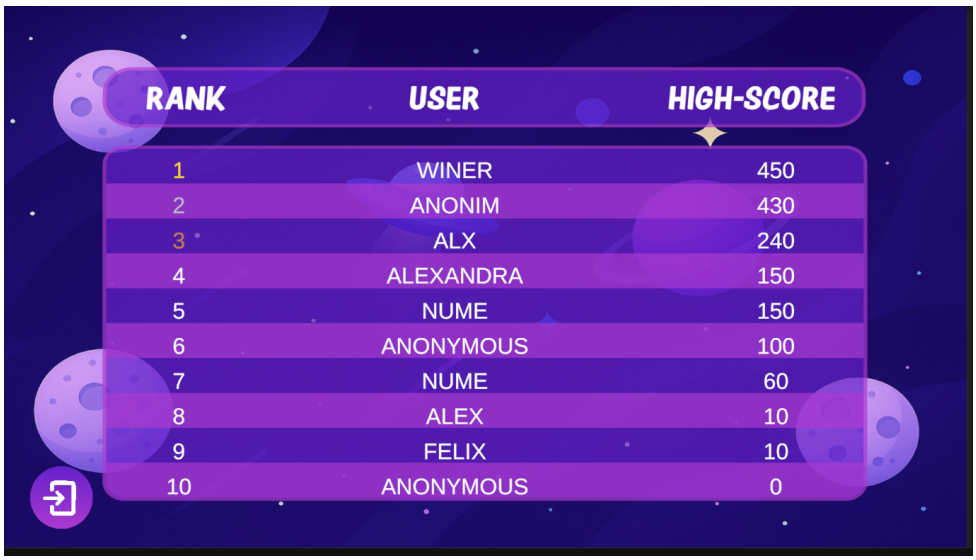

Figure 3.12: High Scores Screen

The data is managed by the ScoreManager.cs script, which implements a Singleton system to persist data between scenes. Scores are saved locally using Unity's PlayerPrefs system, which allows data to be stored even after the game is closed.

When saving a score, a ScoreEntry object is created:

```
ScoreEntry newEntry = new ScoreEntry(playerName, currentScore);
highScores.Add(newEntry);
```

Scores are converted to JSON format and stored:

```
string json = JsonUtility.ToJson(new
 ↪  SerializableList<ScoreEntry>(highScores));
PlayerPrefs.SetString(HIGHSCORES_KEY, json);
PlayerPrefs.Save();
```

When starting the game or accessing the High Scores screen, the system automatically reloads saved scores, this simple but effective system provides a smooth and competitive experience.

About Screen:

The about screen represents another important element when it comes to the visual appearance of the game and user experience, by adding humor and depth to the game. Story telling in a game can bring multiple benefits such as:

- Narrative context and deeper involvement: It makes the game's universe seem

alive and it creates an emotional connection between the player and the characters.Strengthening the game's personality

- The story reflects the personality of the game and plays a key role in branding, making the game recognizable and easy to describe in a few words.
- It represents a subtle gameplay guide by offering different insights.
- It gives the impression of a well-thought-out and completed project, it's not just a game, it's a product.

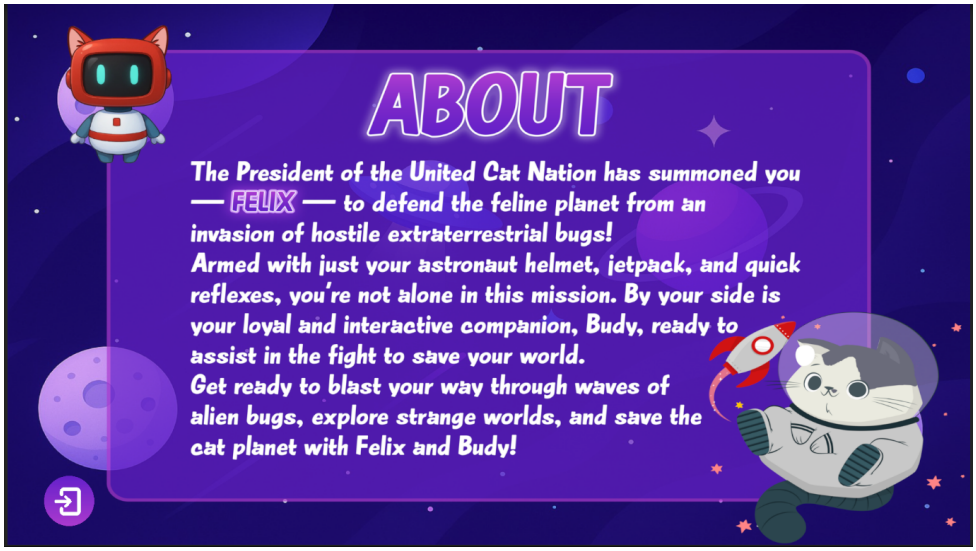

Figure 3.13: About Screen

Game Over Screen

The Game Over screen is an important element in the gaming experience, as it marks the end of the current session and offers the player multiple options:

- Enter name to save score
- Return to Main Menu screen
- Quit the game completely

The interface of this screen is designed to be clear and intuitive. In addition, it is an important moment for the player, as it reflects the performance achieved.

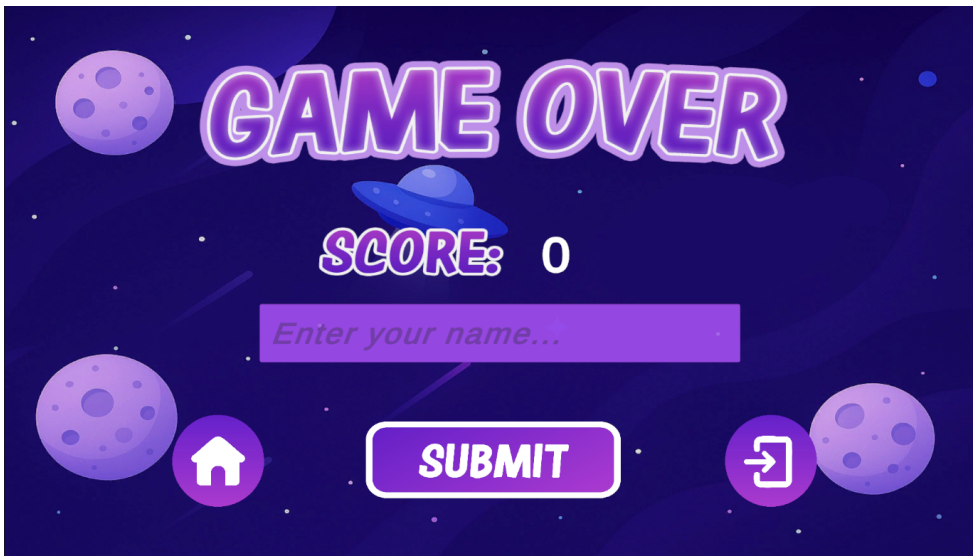

Figure 3.14: Game Over Screen

3.5 Chatbox - Gemini API integration

3.5.1 Purpose of the ChatBox

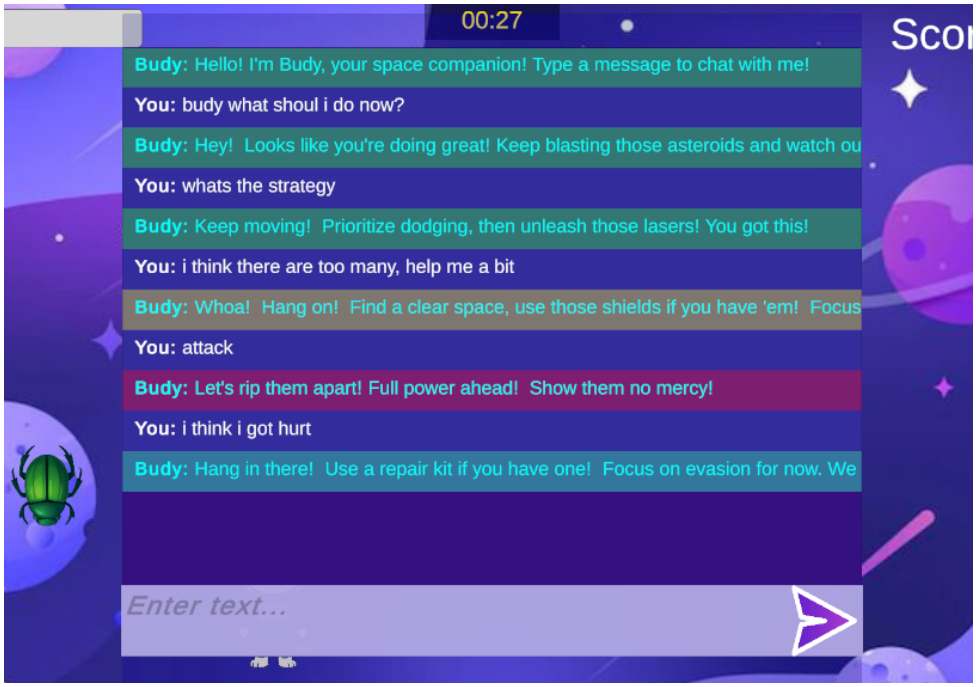

Figure 3.15: ChatBox

A new element for this genre of arcane games, implemented in Cat-Astrophic Invasion game, is the chat box component integrated with AI, which allows users to communicate directly with their Budy via text messages during the game. This functionality not only provides a chat interface, but also a real-time interactive feedback

mechanism that is customized to the context of the game and the emotional state of the simulated Budy.

The implementation is based on connecting Unity application to the Gemini AI service provided by Google Cloud using a REST API. Therefore, the AI's responses are influenced by the user's messages and the current game context (e.g. health level, level progress, lost states).

3.5.2 General Architecture

The functionality is organized in two main software components:

- ChatUiManager: Manages the graphical interface and user interactions.
- GeminiAIManager: Manages communication with the AI service and message processing logic.

This separation allows for a modular, easily extensible, and manageable architecture, where the interface is independent of the communication logic with the AI.

3.5.3 ChatUIManager Component – User Interface

The UI components include:

- chatPanel: the main panel that contains the entire chat
- SchatScrollRect: allows scrolling through messages
- chatContent: the container where messages are added
- inputField: the text field for entering messages
- Buttons for sending messages and activating/deactivating chat

Predefined messages are dynamically generated in the code, providing flexibility in customizing the visual appearance and changing the appearance of the message depending on Budy's state (aggressive, anxious, calm, etc.).

```
void CreateMessagePrefab()
{
    GameObject prefab = new GameObject("MessagePrefab");
    prefab.AddComponent<Image>().color = new Color(0, 0, 0, 0.3f);
```

```
    // Add text to prefab
    var textObj = new GameObject("Text");
    textObj.transform.SetParent(prefab.transform);
    var text = textObj.AddComponent<TextMeshProUGUI>();
    text.text = "Sample message";
    text.fontSize = 14;
    text.color = Color.white;

    // Configure layout and anchors...
    prefab.AddComponent<LayoutElement>().minHeight = 30;
    prefab.AddComponent<ContentSizeFitter>().verticalFit =
    ↪  FitMode.PreferredSize;

    messagePrefab = prefab;
    messagePrefab.SetActive(false);
}
```

To ensure a consistent chat experience, when opening the chat box, the game automatically pauses and the camera focuses on the chat interface. When the chat is closed, the game resumes, and all player controls are enabled again.

```
public void ToggleChat() => SetChatVisibility(!isChatVisible);

void SetChatVisibility(bool visible)
{
    isChatVisible = visible;
    chatPanel?.SetActive(visible);

    if (visible)
    {
        wasGamePausedBefore = Time.timeScale == 0f;
        Time.timeScale = 0f;
        DisablePlayerInput(true);
        inputField?.Select();
        inputField?.ActivateInputField();
```

```
    }
    else
    {
        if (!wasGamePausedBefore) Time.timeScale = 1f;
        DisablePlayerInput(false);
    }

    // update button text
    var text =
    ↪  toggleButton?.GetComponentInChildren<TextMeshProUGUI>();
    if (text != null)
        text.text = visible ? "Resume Game" : "Chat with Budy";
}
```

3.5.4 GeminiAIManager Component – Communication and AI Service

This component is responsible for:

- Contextual message generation (chat history + game state)
- Messaging to the Gemini API
- Reply management

API Configuration

```
[SerializeField] private string apiKey = "AIzaSyB...";
[SerializeField] private string model = "gemini-1.5-flash";
private const string GEMINI_API_URL =
↪  "https://.../gemini-1.5-flash:generateContent";
```

To maintain the consistency of the conversation, the system maintains a message history (the last 6 interactions) and system warnings that define the partner's personality.

The system is configured to respond in different ways depending on the language used by the player: sympathetic in dangerous situations, encouraging when the player succeeds, or aggressive when the player is about to attack.

Communicating with the Gemini API

```
private IEnumerator SendMessageCoroutine(string userMessage,
 ↪  Action<string> onResponse, Action<string> onError)
{
    conversationHistory.Add(new ChatMessage("user", userMessage));
    string fullContext = systemPrompt + moodContext +
     ↪  "\n\nConversation:\n";

    // add last 6 messages to fullContext
    for (int i = ...; i < ...; i++) { /* append conversation */ }

    // create and send JSON request
    var request = new GeminiRequest(new Content[] { new
     ↪  Content(fullContext) });
    string json = JsonConvert.SerializeObject(request);

    using (var webRequest = new UnityWebRequest(url, "POST"))
    {
        // set headers and uploadHandler...
        yield return webRequest.SendWebRequest();

        // handle response or error...
    }
}
```

Messages are sent asynchronously via coroutines to avoid blocking the main thread while waiting for a reply. The interface will display a ”typing” indicator, and the send button will be temporarily disabled to prevent spam.

Game context and adaptive feedback

Before sending the message to Gemini, the system adds context about the game state and current mood:

- The player’s health level
- if the game in a game-over state
- if the player is in the final level

- Budy's current state

This context will be combined with the player's message and sent to Gemini, thus providing a personalized response that reflects the intent of the message and the game state. Thus, Budy becomes more than just a chatbot – it acts as a practical friend, learning from the player's actions and difficulties.

Message Processing with Context

```
public void SendMessageWithContext(string userMessage,
 ↪  Action<string> onResponse, Action<string> onError = null)
{
    string context = GetGameContext();
    SendMessage($"{cleanMessage}\n\nGame context: {context}",
     ↪  onResponse, onError);
}

public string GetGameContext()
{
    var gm = FindObjectOfType<GameManager>();
    string context = gm != null
        ? $"Health: {gm.playerCurrentHealth}/{gm.playerMaxHealth}. "
         ↪  +
          $"GameOver: {gm.IsGameOver()}. Level3: {gm.isLevel3}."
        : "";

    if (budyAI != null) context += $" Budy mood:
     ↪  {budyAI.GetCurrentMood()}.";
    return context;
}
```

In addition to the basic functions, the chat system also has the following advanced functions:

- Auto scroll: Automatically scrolls to new messages
- History limit: Up to 50 messages to record performance
- Visual feedback: Messages will have different colors depending on the mood of the sender and recipient of the message

- Game integration: The chat prevents enemies from appearing in all views, allowing for a more relaxed conversation

The AI-integrated chat box feature is a special element of the game. Cat-Astrophic Invasion delivers an interactive, dynamic, and contextual experience. By combining an intuitive interface with an adaptive AI backend, the system creates the illusion of a real companion that can understand and respond to the player's needs. This approach adds a level of interactivity to 2D indie games and demonstrates the potential for integrating AI technology into video game storylines and dynamic design.

3.6 Challenges encountered during implementation

Developing Cat-Astrophic Invasion was a deep learning experience, especially since it was my first project in the field of video game development using Unity and C#.The challenges I faced in developing this project were many, but they contributed greatly to my growth as a developer. Every difficulty was an opportunity to learn, and the solutions implemented helped me gain a practical understanding of video game development, interacting with AI services and building complex functionalities.

The most significant challenge was that I had no prior experience with Unity or the C# language. I followed the official Unity documentation as well as regular video tutorials. I also experimented a lot with small prototypes to understand the basic concepts.

Integration with external APIs (Gemini AI) is also a challenge from multiple point of view such as:

- the need to understand the JSON request and response structure;
- sending data asynchronously without blocking the main thread;
- Handling network errors and situations where the API is not responding properly;
- Limit the length of threads and messages to avoid exceeding API limits.

Also, being my first interface developed in Unity, I encountered difficulties both in positioning and adapting UI elements to different resolutions, as well as in implementing functionalities such as dynamic message generation, automatic scrolling without visual errors, and aesthetic synchronization of the interface with the emotional state of the Budy AI.

Chapter 4

Evaluation of the application

4.1 Feedback received

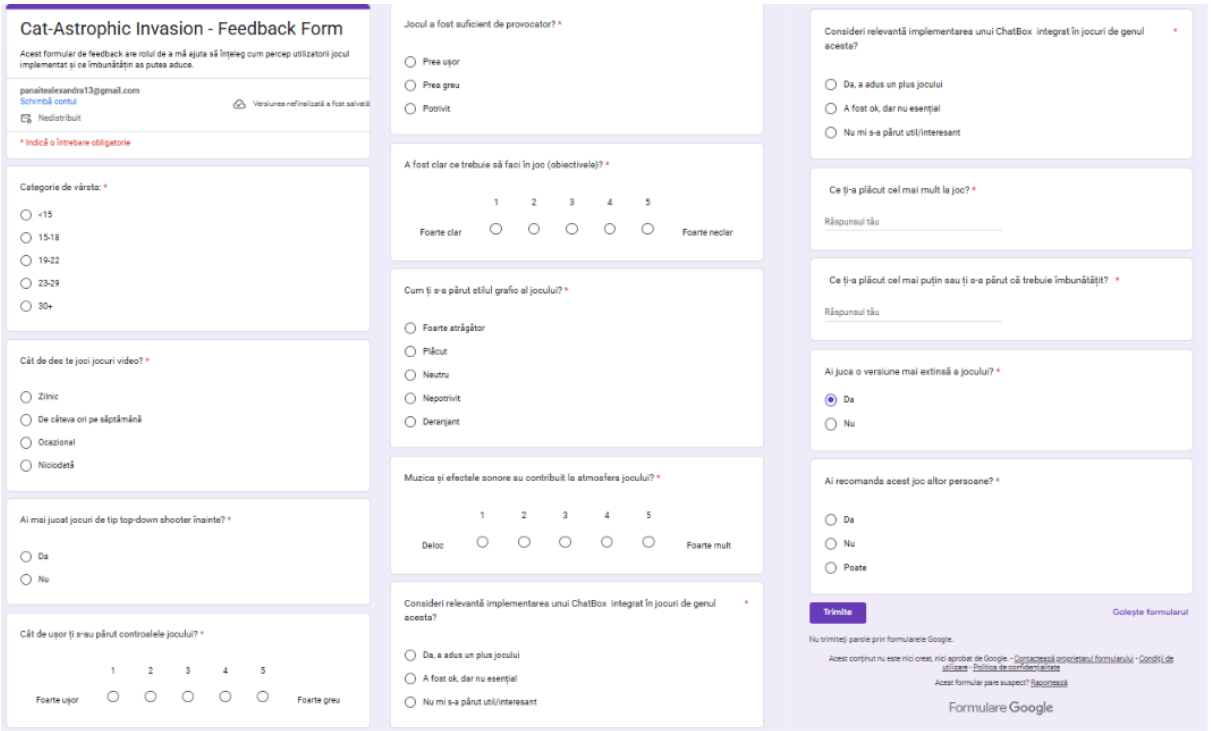

Figure 4.1: Feedback Form

Evaluation of software products through user feedback is an essential part of the development and continuous improvement process. By analyzing the responses received we get detailed insights into user perceptions, and identify strategic directions for the future evolution of the project. A simple feedback form was used to evaluate the game, by a total of 15 respondents. The form covers demographic data, video game

experience, user-friendly visual design, gameplay, and technical innovations such as AI-based ChatBox integration.

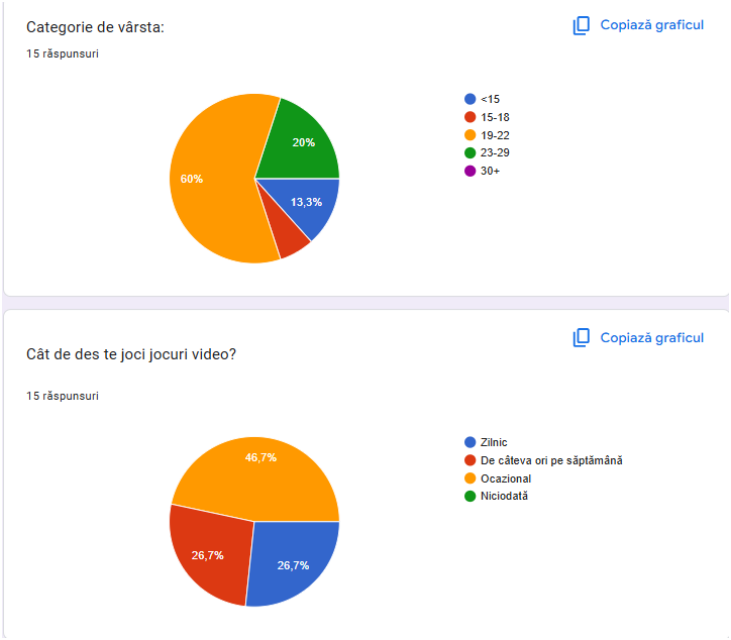

Figure 4.2: Feedback responses: demographic data

The data collected showed that the respondents were evenly distributed, but mostly between the ages of 19 and 22 years old.

This suggest that the game primarily appeals to a younger audience, which is in line with the proposed goals for top-down games with modern AI elements.

When asked how often users play video games, the majority answered ”sometimes.” The distribution gives a clear picture of the game’s accessibility to users of different experience levels.

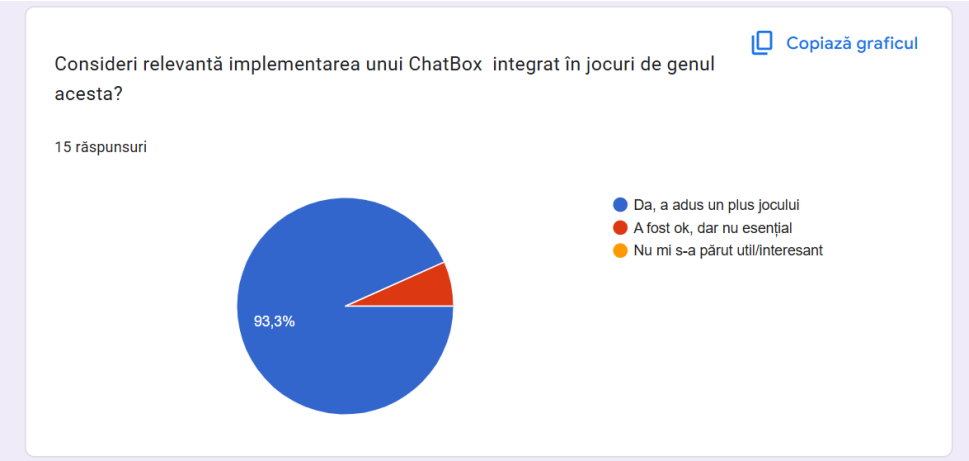

Figure 4.3: Feedback response: ChatBox feature

The focus of the analysis was the notion of an artificial intelligence chat system (ChatBox) implemented using the Google Gemini API.

- 93.3% of respondents acknowledged the relevance and usefulness of this task.
- Only 6.7% considered it a choice.

Quality feedback supports positive impact: ”The AI partner helped me because I was able to ask him what wasn’t obvious.” Its functionality has been proven to greatly enhance the gaming experience.

Also, very promising results were obtained in terms of acceptance and appropriate recommendations. 100% of users will play the expanded version of the game and recommend the game to others.

Figure 4.4: Feedback response

According to the independent opinion poll, aspects such as the design and Visual effects, gameplay and integration of the AI were greatly appreciated by users

While most of the feedback was positive, the following areas for improvement were identified.

Content Development

Users would enjoy a longer, more complex version of the game with more levels and characters.

More varied and challenging levels

New game mechanics, enemy types, or levels can be added to create a more varied experience.

This evaluation process showed that Cat-astrophic Invasion's gameplay was well received by its users and successfully integrated intuitive gameplay with AI elements, paving the way for new innovative and mechanical directions in future versions.

4.2 Further directions for implementation

Based on the feedback collected, several concrete directions for further development of the Cat-Astrophic Invasion game have been identified. These proposals aim both to improve the gaming experience for users and to exploit the technological potential of the Unity platform and integrated artificial intelligence.

Integrating a more elaborate game story

One of the key aspects of maintaining player interest in the long term is narrative construction. In the current version, the game's story is relatively linear and focused on progression between levels. An important direction for development is the introduction of a branching narrative, with meaningful player choices that influence the course of events. This approach would increase the game's replayability and allow for deeper customization of the experience.

Expanding the AI interface: voice responses and cinematic interactions

To improve player engagement in the story, it is proposed to integrate dynamically generated voice responses using text-to-speech technologies, so that the AI communicates with the player through voice. This can contribute to a stronger sense of emotional attachment to the companion and can replace traditional UI elements in some cases.

The AI could also have active interventions in cinematic scenes (cutscenes), providing comments, reactions or even influencing the story. Thus, the ChatBox would become a continuous and significant presence throughout the narrative.

Cat-astrophic Invasion was originally intended as an entertainment video game, and its original concept was to incorporate a conversational artificial intelligence assistant into the game, with potential applications adapted to various fields beyond the gaming industry. Although Cat-astrophic Invasion was originally conceived as an entertainment video game, its central concept — integrating a conversational AI companion into gameplay — has the potential to be adapted and leveraged in various other fields, beyond the gaming industry. Such as Interactive Education. The AI companion can be transformed into a virtual educational assistant that guides students

through gamified lessons, provides personalized explanations, or answers questions in real time.

Conclusions

The Cat-astrophic Invasion project represents more than just a game for me: it is a summary of the technical knowledge accumulated during my studies, an expression of the passion for interactive design and a personal challenge to complete a attractive and original product.

During the development process, I went through all the fundamental stages of a game development project, from idea and documentation, to implementation, testing and then optimization. I learned to manage the complexity of a project of this nature, combining game logic programming with visual elements and user experience.

Every aspect of the game was carefully thought out to contribute to an interactive gaming experience: from the levels with progressive difficulty and the presence of dynamic obstacles (such as asteroids), to the scoring mechanisms and the Buddy companion AI.

I believe that this project reflects my ability to transform an idea into a complete product, it is an achievement that I am proud of and marks an important point in my journey as a software developer and, potentially, as a game designer.

This game is the result of a sincere effort, supported by enthusiasm, curiosity, and the desire to create something that combines both technology and creativity – two essential aspects in any successful interactive product.

Bibliography

Books:

- Harrison Ferrone, *Learning C# by Developing Games with Unity*,Packt Publishing, 2021:

  https://books.google.ro/books/about/Learning_C_by_Developing_Games_with_Unit.html?id=OI22zgEACAAJ&redir_esc=y

- Jesse Schell, *The Art of Game Design: A Book of Lenses*,CRC Press, 2008:

  https://books.google.ro/books/about/The_Art_of_Game_Design.html?id=LP5xOYMjQKQC&redir_esc=y

Links:

- **Unity Manual – 2D Game Development:**

  https://docs.unity3d.com/6000.1/Documentation/Manual/Unity2D.html

- **Unity (game engine) - Wikipedia:**

  https://en.wikipedia.org/wiki/Unity_(game_engine)

- **Unity – C# Scripting:**

  https://unity.com/how-to/programming-unity

- **C# - Programming language:**

  https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

- **Unity and C# HTTP Requests:**

  https://docs.unity3d.com/Manual/web-request-sending-form.html

- **Google AI – Gemini API Documentation:**

  https://ai.google.dev/gemini-api/docs

- **Adobe Photoshop:**

  https://en.wikipedia.org/wiki/Adobe_Photoshop

- **Chicken Invaders:**

  https://en.wikipedia.org/wiki/Chicken_Invaders

- **Galaga Game:**

  https://en.wikipedia.org/wiki/Galaga

- **Space Invaders (1978):**

  https://en.wikipedia.org/wiki/Space_Invaders

- **Geometry Wars:**

  https://en.wikipedia.org/wiki/Geometry_Wars

- **Diagram drawing:**

  https://app.diagrams.net/