2024/2025

SHADOW CANNON

BY:
CLAUDIA MARIA CARBONI (535421)
SUSANNA MAZZOCCHI (535996)

Computer Programming - Bachelor of Science in Artificial Intelligence

1. Introduction and Objectives

Narrative Concept and Context

“Shadow Cannon” is conceived as a unique gaming experience that combines strategy, physics, and precision. The player, in the role of the last Master of the Shadow Cannon, must embark on a journey through four thematic realms – from Hell, through Darkness and Cursed Lands, to Space – to defeat a darkness that threatens to devour entire worlds. This narrative path guides the user towards a stimulating and engaging gaming experience and also determines the evolution of the game.

Main Objectives:

Progressive Difficulty and Complexity

We created four levels where the difficulty and complexity of the mechanics progressively increase, making the game more challenging. This allows the player to gradually prepare before facing more complex challenges.

Diverse Weapons and Mechanics

We implemented three different types of weapons (bomb, bullet, and laser) and physical mechanics (parabolic motion, collisions, reflections) to enrich the gaming experience. Each weapon is designed to be useful in specific situations, encouraging the player to think strategically.

Separate Interface and Logic

We used Python and the Kivy framework to develop an interface separate from the game logic. This made it easier to make modifications and collaborate.

Scoring System and Hall of Fame

We created a scoring system based on completion time and a Hall of Fame to encourage users to improve their best scores.

Distinct Visual Themes

Each level features a distinct visual theme that contributes to a sense of progression and discovery. Despite the thematic differences, the design maintains a stylistic coherence that unifies the experience.

Challenges and Teamwork

At the beginning, we encountered many difficulties, especially in managing collisions and implementing laser reflections. However, working together was extremely stimulating. Teamwork allowed us to continuously refine our skills and improve our project over time. Additionally, the constant exchange of ideas helped maintain high code quality and overcome the challenges we faced.

Incremental Development

We developed our project incrementally, following these main steps:

1. **Initial Prototype**: We started by implementing the essential functionalities - a cannon, a simple projectile, and a target. This prototype allowed us to identify potential problems and start getting hands-on experience.
2. **Basic Mechanics**: Next, we implemented the physics system for projectile movement, basic collisions, and target detection.
3. **Level Expansion**: Only after perfecting the first level did we begin implementing the subsequent levels, adding new mechanics (shotgun, laser, bats, mirrors, reflections) incrementally.
4. **Optimization and Bug Fixing**: Once all levels were completed, we focused on optimization, balancing difficulty, and fixing bugs.

Separation of Logic and Interface

One of the strengths of our project was the clear separation between Python code (logic) and KV files (interface). Kivy encourages this separation, and we took full advantage of it. This way, we achieved cleaner and more organized code, allowing us to modify the interface without changing the logic and vice versa.

2. Architecture and Structure of the Game

ScreenManager and Navigation

The heart of the application is Kivy's ScreenManager, which manages navigation between the different screens of the game. Each screen is implemented as a class that extends Screen.

```
class MenuApp(App):
    def build(self):
        sm = ScreenManager()
        sm.add_widget(MainMenu(name='mainmenu'))
        sm.add_widget(Level1Game(name='level1game'))
        sm.add_widget(Level2Game(name='level2game'))
        sm.add_widget(Level3Game(name='level3game'))
        sm.add_widget(Level4Game(name='level4game'))
        sm.add_widget(HallOfFame(name='halloffame'))
        return sm
```

This architecture has several advantages:

1. Each screen manages only its own behavior.
2. Transitions between screens are automatically handled by the ScreenManager.
3. Adding new screens (such as additional levels) is simple and does not require changes to the existing structure.

Game System

The basic gameplay logic is centered around four main components, which interact with each other in a coordinated manner:

1. **Cannon**: The player's control point, with adjustable parameters (angle and speed) that determine the projectile's trajectory. The cannon is implemented as a rotating image, with the rotation determined by the selected angle.

```
def drop(self, dt):
  # Update cannonball position
    new_x = self.cannonball.pos[0] + self.velocity_x
    new_y = self.cannonball.pos[1] + self.velocity_y
    self.cannonball.pos = (new_x, new_y)

    # Apply gravity
    self.velocity_y -= 0.98

    # Check for collisions
    self.check_collisions()
```

2. **Projectiles**: Different types (bomb, shotgun, laser) with distinct physical behaviors. Each type of projectile is implemented as a specialized class or as specific behavior within the level.

```
# Different projectiles
def fire_cannon(self):
    if self.shots_left <= 0:   # Check if shots are available
      self.show_game_over_popup()
      self.reset_level()  # Reset the level when there are no shots left
      return

    if self.current_weapon == "cannon":
      self.fire_single_shot()
    elif self.current_weapon == "shotgun":
      self.fire_shotgun()
```

```
def fire_shotgun(self):
    for offset in [-10, 0, 10]:
        self.create_pellet(self.angle + offset)


def fire_laser(self):
mirror_collision = self.check_laser_collision(start_x, start_y, end_x, end_y, self.mirror)
```

3. **Obstacles**: Elements that block the path to the target, each with specific behaviors:
    - **Destructible Rocks**: Can be destroyed by the bomb.
    - **Perpetual** (Indestructible): Block any projectile.
    - **Bats**: Enemies that require the shotgun.
    - **Mirrors**: Reflect lasers according to the laws of optics.
    - **Target**: The objective to hit to complete the level, implemented with dedicated collision detection. It can be hit by all types of weapons.

Game Flow

The game flow is designed to be intuitive yet challenging and is divided into four steps:

- **Preparation**: The player adjusts the cannon parameters, considering the obstacles and the target's position. This phase requires analysis and strategy.
- **Execution**: The player selects the type of projectile (which varies depending on the level) and fires it, activating the physical simulation.
- **Adaptation**: Based on the results, the player adjusts the strategy for the next attempt. The limited number of shots encourages thoughtful choices.
- **Completion**: When the target is hit, the level ends, the time is recorded, and the player can advance to the next level.

3. Implementation of Physics and Game Mechanics

Standard Projectile Movement (Cannon)

For the basic projectile, we implemented simplified physics of parabolic motion, based on the equations of motion of a body under the influence of gravity. The position is updated at regular intervals (60 times per second) to ensure smooth movement. At each frame, gravity is applied to the vertical component of the velocity, and the new position is calculated. Calibrating the values of gravity and speed required numerous testing sessions to find the right balance between physical realism and playability.

Collision System

The collision system uses a **bounding box** approach to detect intersections between elements. We chose this method for its computational efficiency and relative simplicity of implementation, while maintaining adequate precision for the type of game. This collision approach checks if the bounding rectangles of the two objects (projectile and obstacle/target) overlap. If they do, the appropriate response is triggered: destruction of the rock, error message for the indestructible obstacle, or level completion for the target.

Example:

```
def check_collision(self, projectile, target):

ball_left = self.cannonball.pos[0]
ball_right = ball_left + self.cannonball.size[0]
ball_bottom = self.cannonball.pos[1]
ball_top = ball_bottom + self.cannonball.size[1]

# Collision with the ghost target
ghost_left = self.ghost.pos[0]
ghost_right = ghost_left + self.ghost.size[0]
ghost_bottom = self.ghost.pos[1]
ghost_top = ghost_bottom + self.ghost.size[1]

if (ball_right > ghost_left and ball_left < ghost_right and
        ball_top > ghost_bottom and ball_bottom < ghost_top):
    self.hit_target()
    self.reset_cannonball()

        # Collision with destructible obstacles

        for obstacle in self.obstacles[:]:

            if obstacle.pos[0] != -1000:  # If the obstacle is not destroyed

                obs_left = obstacle.pos[0]

                obs_right = obs_left + obstacle.size[0]

                obs_bottom = obstacle.pos[1]

                obs_top = obs_bottom + obstacle.size[1]


                if (ball_right > obs_left and ball_left < obs_right and

                        ball_top > obs_bottom and ball_bottom < obs_top):

                    obstacle.pos = (-1000, -1000)  # Destroy the obstacle

                    self.reset_cannonball()
```

Shotgun Mechanics

The shotgun, introduced in the second level, offers a different gameplay experience compared to the standard projectile. It generates three smaller projectiles that move simultaneously at slightly different angles, creating a fan pattern. The update of the shotgun projectiles is similar to that of the standard projectile but applied to all active projectiles simultaneously. A unique feature of the shotgun is that it is the only effective weapon against bats, a type of enemy introduced in the second level. This specification has been implemented in the collision system.

Laser Reflection System

The laser, introduced in the third level, represents the most advanced mechanic in the game. Unlike ballistic projectiles, the laser follows a linear path and can bounce off mirrors according to the laws of optics (angle of incidence = angle of reflection). The implementation of the laser is based on a simplified ray-tracing system.

4. User Interface, Visual and Audio Feedback

Menu

The main menu was designed to introduce the game's atmosphere and provide access to all features in a clear and intuitive way. It includes options to start a new game, access the Hall of Fame, and view game instructions.

Intuitive User Interface

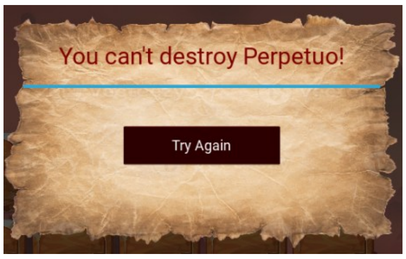
We created an interface that is easily understandable and consistent across the various levels.

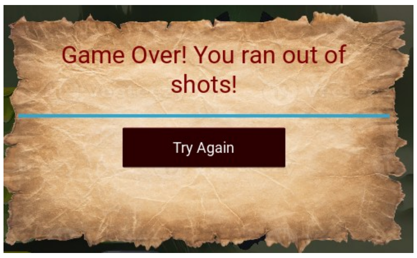
Game Levels

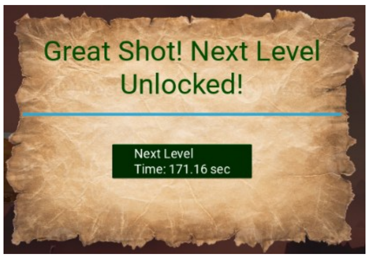
Each level (Level1Game, Level2Game, Level3Game, Level4Game) is managed as an independent screen, loading only the necessary resources at that moment.

Popups and Visual Feedback

We implemented a popup system to provide information and feedback to the player. For example, popups are displayed to signal level completion or errors (e.g., collision with an indestructible obstacle, or using the wrong weapon to hit the obstacle).


You can't destroy Perpetuo!


Game Over! You ran out of shots!

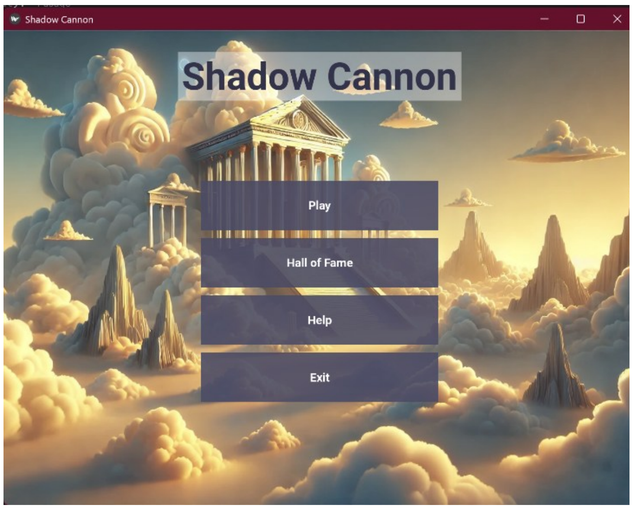

Great Shot! Next Level Unlocked!

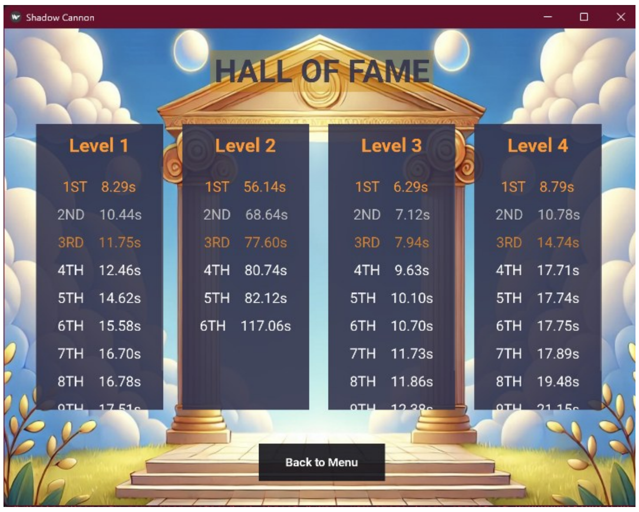
Background Music

The background music was carefully chosen to enhance the player's immersion and make each section unique. We selected one for the menu and its sections, one for all four levels, and another for when the game concludes.

Record Display

The Hall of Fame interface displays records in a time-ordered table, divided by each level.





Level 1		Level 2		Level 3		Level 4	
1ST	8.29s	1ST	56.14s	1ST	6.29s	1ST	8.79s
2ND	10.44s	2ND	68.64s	2ND	7.12s	2ND	10.78s
3RD	11.75s	3RD	77.60s	3RD	7.94s	3RD	14.74s
4TH	12.46s	4TH	80.74s	4TH	9.63s	4TH	17.71s
5TH	14.62s	5TH	82.12s	5TH	10.10s	5TH	17.74s
6TH	15.58s	6TH	117.06s	6TH	10.70s	6TH	17.75s
7TH	16.70s			7TH	11.73s	7TH	17.89s
8TH	16.78s			8TH	11.86s	8TH	19.48s
9TH	17.51s			9TH	12.38s	9TH	21.15s

5. Level Features

Level 1: Hell

- **Objects**: rocks, indestructible perpetuities, and ghost (target).
- **Projectiles**: bomb (basic projectile).
- **Difficulty**: low, introduction to the basic mechanics of the game.
- **Number of shots available**: 15

Description

The first level of Shadow Cannon is set in hell, a dark and menacing place. The player must use a cannon to hit a target positioned at the opposite end of the screen. The main obstacles are destructible rocks and indestructible perpetuities, which complicate the projectile's path. The projectile used in this level is the bullet, which follows a parabolic trajectory influenced by gravity. This level serves as an introduction to the basic mechanics of the game, allowing the player to familiarize themselves with cannon control and obstacle management.

This level was fundamental for familiarizing with the basics of the game and projectile dynamics. The main challenge was implementing the projectile physics and managing collisions with obstacles. We decided to keep the first level relatively simple.

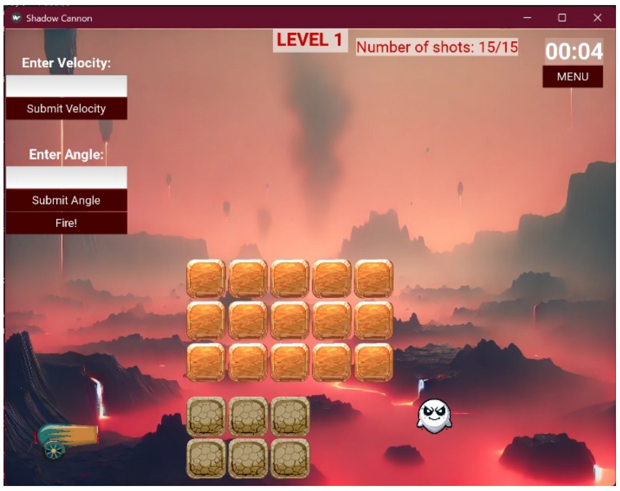
Level 2: Abyss

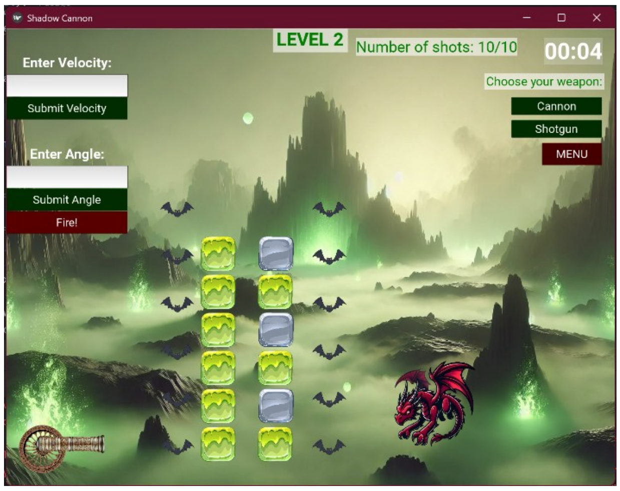
- **Objects**: bats, rocks, indestructible perpetuities, and dragon (target).
- **Projectiles**: bomb and shotgun (projectile that can hit bats).
- **Difficulty**: increased complexity with the introduction of new obstacles and projectiles.
- **Number of shots available**: 10

Description

The second level is set in darkness, an even darker and more unsettling environment. In addition to the obstacles present in the first level, new elements such as bats are introduced, which can only be hit with the Shotgun projectile. This level requires the player to use both the bullet and the shotgun to overcome obstacles and hit the target. The complexity increases as the player must manage different types of projectiles and obstacles, making the game more interesting and challenging.

This level made the game more interesting and challenging by introducing new elements and obstacles. Managing the bats and implementing the Shotgun projectile were the main challenges.





Level 3: Cursed Lands

- **Objects**: bats, rocks, indestructible perpetuities, mirror, and witch (target).
- **Objectives**: hit the target using the laser and reflecting it off the mirrors.
- **Projectiles**: Bomb, Shotgun, and Laser (projectile that can reflect off mirrors).
- **Number of shots available**: 10

Description

The third level is set in the cursed lands, a place full of dangers and obstacles. In this level, the laser is introduced, a new type of projectile that can reflect off mirrors. The player must use the laser to hit the target, taking advantage of reflections off mirrors to overcome obstacles. This level adds a new dimension to the game, requiring the player to carefully plan their shots and use reflections to their advantage.

This level added a new weapon to the game, making it more strategic and interesting. Managing the laser reflections and implementing collisions with mirrors were the main challenges.

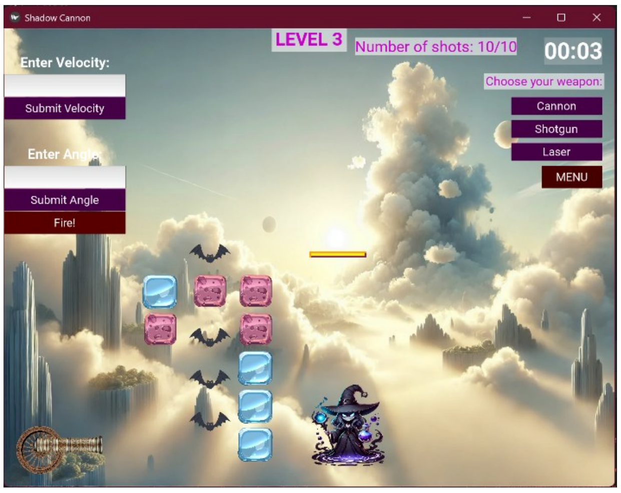
Level 4: Space

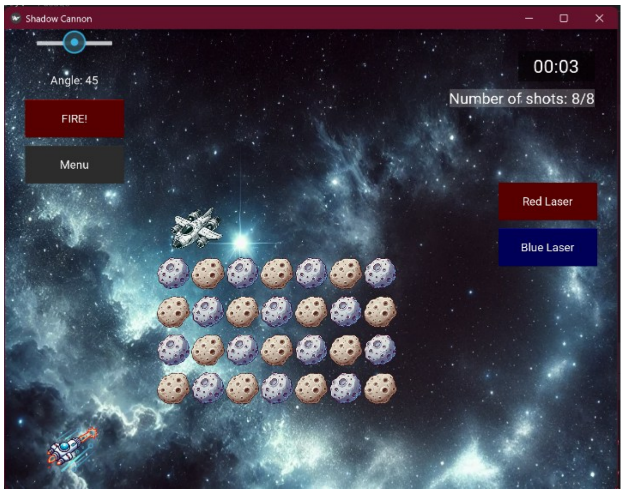
- **Objects**: asteroids and spaceship (target).
- **Projectiles**: red laser (to hit the spaceship) and blue laser (to hit the asteroids).
- **Difficulty**: maximum complexity with the introduction of two types of lasers and specific obstacles.
- **Number of shots available**: 8

Description

The fourth and final level is set in space, a vast and mysterious environment. In this level, the player must use two types of lasers: the red laser to hit the spaceship and the blue laser to hit the asteroids. Each type of laser has unique characteristics and can only hit certain obstacles. This level represents the maximum complexity of the game and requires the player to strategically use the different types of lasers to overcome obstacles and hit the target.

Managing the two types of lasers and implementing specific collisions for each type of obstacle were the main challenges. We decided to make this level as complex and stimulating as possible to offer a final challenge to the players.





6. Testing and Results

We conducted tests on a laptop with Windows 10, Intel i7 processor, and 32GB of RAM, using Python 3.12.0 and Kivy 2.3.1.

When a bug occurred, we inserted print statements before and after the action that was supposed to happen but didn't. This approach was effective in verifying whether the code executed that particular section or if it was skipped, ignored, or incorrect. Our main issues were with collisions, laser functionality, and the overlap of menu music with level music.

This project has taught us many valuable skills and lessons:

- We learned the importance of having a well-organized and modular code structure to manage increasing complexity.
- We experienced the benefits of an incremental approach, allowing us to test and refine each component before moving on to the next.
- We gained insights into level design principles, balancing difficulty, and creating an engaging user experience.
- We applied practical concepts of physics, such as parabolic motion, collisions, and reflections.

7. Possible Improvements and Future Developments

- **More realistic physics**: A possible evolution would be the implementation of more advanced physics that consider factors such as air resistance, friction, and deformations.
- **Multiplayer mode**: Adding a two-player mode, where turns are alternated or players compete in real-time, could significantly increase the fun.
- **Mobile and OS version**: Adapting for touch devices and also for OS devices.
- **More efficient level definition**: Finding a way to define levels without having to copy and paste similar parts multiple times.
- **New worlds**: Adding new worlds with different themes to increase the variety of the game.
- **Collision Management**: Adopt more complex algorithms to improve accuracy in partial collisions, instead of using bounding check.

8. Conclusions

The development of Shadow Cannon has been a challenging but extremely rewarding journey. We chose to customize our project a lot to express our creativity, especially at the graphical level. Each level adds new elements and obstacles, allowing us to continuously improve the game. Working in pairs allowed us to make the most of our complementary skills and stimulate our creativity. We are particularly satisfied with the final result and the positive feedback received from our families. Shadow Cannon represents not only a successful university project but also a formative experience that allowed us to put into practice and consolidate the knowledge acquired during the course, facing real game development challenges and discovering creative solutions to the problems encountered. The project demonstrates how programming can be used not only as a technical tool but also as a means of creative expression, combining logic, mathematics, design, and storytelling in an engaging interactive experience.

9. Tools, Resources, and References

- **Python** (3.12.0): The programming language used to develop the game logic.
- **Kivy** (2.3.1): The library used to create the game's graphical interface.
- **PyCharm** (2024.3.4): The IDE used for code development, chosen for its ease of use and advanced features.
- **ChatGPT 4**: Used to get suggestions and improvements during the project development, particularly to understand the use of Kivy and its concepts.
- **DALL-E**: Used to create backgrounds and objects.
- **Suno AI**: Used to create the game's soundtracks.

10. References

We drew inspiration and materials from various sources, including:

- Artillery game - Wikipedia
- YouTube Python Kivy Game Tutorial
- https://www.geeksforgeeks.org/kivy-tutorial/
- https://github.com/kivy/kivy/tree/master/examples
- https://github.com/GrumpyOldSkeleton/cannon-game