The Cannon Game: Around The World - Final Exam Project

* **Course Name:** Computer Programming, Algorithms and Data Structures, Mod. 1
* **Instructor:** Stefano Ferrari
* **Academic Year:** 2023/24
* **Authors:** Chiara Pierini, Marta Paniconi MATRICOLEEE
* **University:** Università degli Studi di Pavia
* **Date:** February 25, 2025



## **1. Introduction**

### **Project Overview and Objectives**

The **Cannon Game: Around the World** is a **2D artillery-based physics game** developed using **Python and Kivy**. Our goal was to create an **interactive and accessible game** that combines **simple mechanics with strategic depth**, while offering an engaging retro-style experience. Players must **destroy all rocks within a limit of 10 shots per level** to progress through **five increasingly difficult levels**, each representing a different continent. The final objective is to **conquer all continents** by successfully completing all levels.

In developing this project, we aimed to **apply and refine our understanding of programming concepts**, particularly **object-oriented programming and game physics**. This project was our first **full Python implementation**, and working with **Kivy for UI and game development** was a major learning experience. Initially, we faced challenges in understanding Kivy’s structure, but as we progressed, we adapted and improved our approach, refining gameplay mechanics and optimizing UI responsiveness.

The key objectives of this project include:

* Implementing a **realistic physics engine** that simulates **gravity, projectile motion, and collisions**, allowing players to experiment with angles and velocities.
* Designing a **strategic gameplay experience** that requires players to **adapt their approach** based on different projectile types and obstacles.
* Creating a **visually engaging UI**, ensuring clear score tracking, shot management, and immersive feedback to enhance gameplay.

Throughout development, our original ideas evolved as we tested and improved different aspects of the game. Balancing **difficulty progression** was an important factor—we wanted the game to become more challenging over time while still allowing players to **learn from their mistakes and achieve a high score**.

Additionally, we considered **basic usability principles**, assuming that players would have **familiarity with simple computer games** and that the game would be played on devices with **adequate screen resolution and processing speed**.

This project not only strengthened our **technical skills** but also taught us valuable lessons in **game design, problem-solving, and iterative development**, making it a rewarding experience from both a learning and creative standpoint.

## **2. Game Specifications**

### **2.1 General Gameplay**

The game consists of the following **core elements**:

* **A Cannon** controlled by the player, capable of adjusting firing angle and velocity.



* **A Target** consisting of **destructible rocks** that must be cleared in **10 shots or fewer**.
* **Obstacles** that **block, reflect, or teleport projectiles**, altering their trajectories.
* **Projectile physics** incorporating **gravity, reflection, and penetration mechanics**.

### **2.2 Projectile Types**

Each projectile behaves uniquely, requiring **different strategies** for effective use.

#### **Bullet**

* **Trajectory:** Parabolic (affected by gravity).
* **Effect:** Explodes on impact, dealing damage within a small radius.
* **Best used for:** Straightforward shots with precise aiming.

#### 

#### **Bombshell**

* **Trajectory:** Parabolic (affected by gravity).
* **Effect:** **Penetrates obstacles** before detonating, causing **wider damage**.
* **Best used for:** **Breaking through obstacles** or hitting targets behind cover.

#### 

#### **Laser**

* **Trajectory:** Moves in a **straight line** (unaffected by gravity).
* **Effect:** **Reflects off mirrors**, penetrates obstacles **without explosion**.
* **Best used for:** **Ricochet shots** and navigating around obstacles.

The player must **strategically select** the right projectile for each scenario.

### **2.3 Obstacles**

Obstacles add variety and **increase difficulty** by altering projectile paths.

#### 

#### **Rock**

* **Main objective**: Must be destroyed to complete the level.
* **Effect**: Absorbs projectile damage and breaks apart.

#### 

#### 

#### 

#### **Bulletproof Mirror**

* **Effect**: **Reflects lasers** at an angle based on the impact direction.
* **Strategy**: Used to create **indirect shots** with lasers.

#### 

#### 

#### 

#### **Perpetio**

* **Effect**: Stops all projectiles **except lasers**.
* **Strategy**: Requires aiming around or using **penetrating projectiles**.

#### 

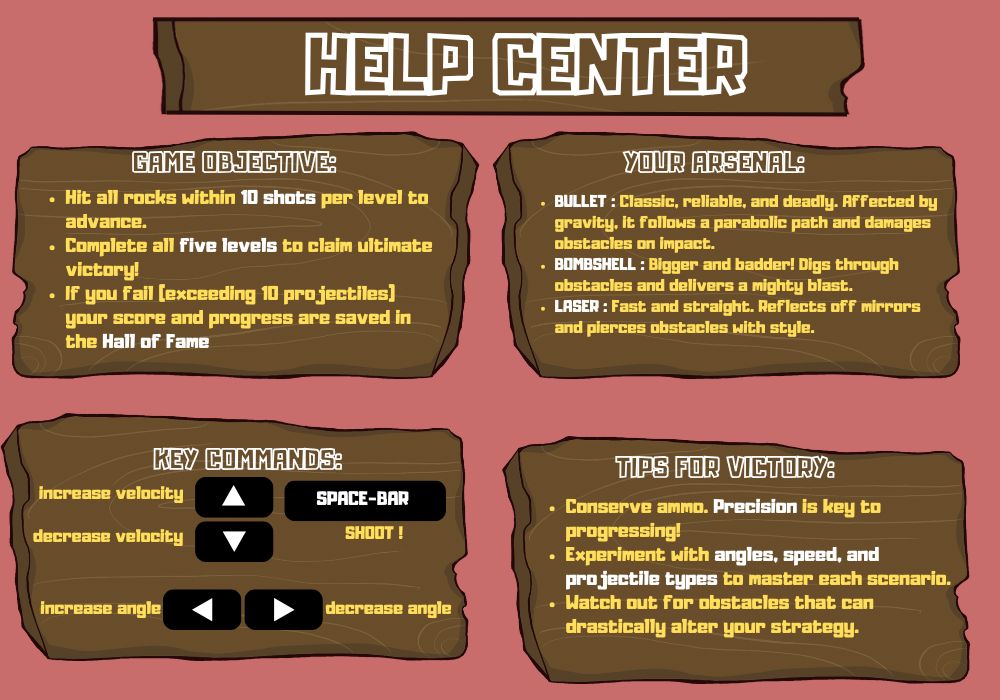
#### 

#### **Wormhole**

* **Effect**: **Transports projectiles** from one portal to another, maintaining their speed and angle.
* **Strategy**: Allows creative shots by **bypassing obstacles**.

### **2.4 Game Functionalities**

The game includes key functionalities that enhance the player’s experience:

* **Hall of Fame**: Stores the best scores based on **accuracy and completion time**.
* **Help menu**: Provides **game instructions, projectile descriptions, and obstacle mechanics**.

### 

### 

### 

### **2.5 Visualization**

* **Fixed-screen mode**: The **entire game field** is visible at all times.
* **Dynamic background changes**: Different levels have distinct backgrounds.

The game maintains **clear UI elements** for displaying:

* **Current shots remaining**
* **Score**
* **Level progression**

## **3. Implementation Details**

### **3.1 Code Structure**

The project is structured into **three main files** for modularity:

1. **cannon\_constants.py**
   * Stores constant values for **screen size, projectile physics, and obstacle behavior**.
2. **game\_interface.py**
   * Manages the **UI elements**, including labels, buttons, and backgrounds.
3. **main.py**
   * Implements **game logic**, including **physics calculations, obstacle interactions, and level progression**.

## **4. Code Explanation**

### **4.1 Constants (cannon\_constants.py)**

Defines all game parameters, such as:

* **Screen dimensions** (SCREEN\_WIDTH, SCREEN\_HEIGHT).
* **Frame rate** (FPS) for smooth animations.
* **Projectile physics** (mass, velocity ranges, impact behavior).
* **Obstacle properties** (reflectivity, destructibility, gravitational effects).

This ensures **easy adjustments** without modifying the game logic.

### **4.2 User Interface (game\_interface.py)**

Uses **Kivy widgets** to create an interactive interface:

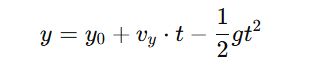
* **Labels**: Display score, shots left, and level progress.
* **Buttons**: Provide interactions for starting the game, viewing the Hall of Fame, and accessing help.
* **Background management**: Updates scenery dynamically for each level.

Kivy’s **flexibility** allows responsive **UI scaling** across different screen sizes.

### **4.3.1 Projectile Class**

The **Projectile class** is responsible for simulating the **physics and interactions** of the different projectile types. It manages their **movement, collisions, and special behaviors**.

* **Launch mechanics:** When a projectile is fired, its **initial velocity is computed** based on the **launch angle and firing power**, converting the user’s input into a trajectory.
* **Gravity simulation:** For **bullets and bombshells**, gravity is applied as a downward acceleration per frame, causing them to follow a **parabolic path**. The vertical displacement is updated using the equation:



* **Collision detection:** The projectile continuously checks for collisions with **obstacles, targets, or the ground**.
* **Reflection and teleportation:** If a **laser** hits a **mirror**, its trajectory is **reflected** according to the **law of reflection**. If a projectile enters a **wormhole**, it is instantly teleported to another location while preserving its **velocity and direction**.

Each projectile type (bullet, bombshell, laser) has **specific interactions** with obstacles, requiring different strategies to clear each level.

### **4.3.2 Obstacles Class**

The **Obstacles class** defines and updates how obstacles interact with projectiles, ensuring they **modify projectile behavior appropriately**.

* **Collision handling:** Determines **how each projectile interacts** with different obstacles, whether it is stopped, reflected, teleported, or destroyed.
* **Behavior control:** Manages unique effects such as **mirrors reflecting lasers, gravitational fields altering trajectories, and wormholes teleporting projectiles**.
* **Destruction system:** Tracks **damage applied to destructible objects** (such as rocks) and removes them when they receive enough hits.

The system ensures that obstacles react **consistently and predictably**, requiring players to carefully plan their shots.

### **4.3.3 Cannon Class**

The **Cannon class** controls all aspects of aiming and firing, ensuring the player has full control over their shots.

* **Adjustable angle and velocity:** The cannon’s angle and firing power can be modified using the **arrow keys**, allowing the player to aim precisely.
* **Shooting mechanics:** Pressing the **spacebar** launches a projectile based on the selected angle and velocity. The cannon then switches to a **cooldown state** before allowing the next shot.
* **Ammo management:** The cannon tracks the number of **remaining shots**, preventing firing if the player has run out.

This class ensures that shooting mechanics feel **responsive and intuitive**, while still requiring skill to master.

### **4.3.4 Game Flow**

The **game flow** ensures a smooth and structured experience, managing transitions between levels, user input, and scorekeeping.

* **Game loop:** Runs continuously at a fixed **frame rate (FPS)**, updating **projectiles, obstacles, and UI elements**.
* **Keyboard inputs:** Processes **player actions** such as adjusting the **cannon’s angle, velocity, and firing projectiles**.
* **Level transitions:** Checks if all **rocks have been destroyed**. If so, the player advances to the **next continent**. If the player **runs out of shots**, the game ends.
* **Score tracking and Hall of Fame:** After game completion, the **final score is saved** in the **Hall of Fame**, allowing players to compare performance.

This structured approach ensures that **every element of the game is updated efficiently**, providing an engaging and well-paced experience.

## **5. Challenges and Solutions**

During development, we encountered several challenges that required **refinement of physics, game logic, and UI management**.

* **Physics Implementation:** Achieving accurate **gravity simulation and collision detection** proved difficult initially, leading to inconsistencies in projectile motion. We refined **velocity calculations** and implemented a **distance-based detection system** to improve accuracy.
* **Obstacle Interactions:** Managing **wormhole teleportation, laser reflections, and rock destruction** required careful handling of physics. We addressed **infinite teleport loops**, ensured **correct reflection angles**, and implemented **hit counters** to track destructible obstacles.
* **Kivy UI Management:** Keeping **UI elements updated in real-time** was critical for smooth gameplay. We used **scheduled updates** for dynamic content and structured **responsive layouts** to maintain alignment across different screen sizes.

## **6. Conclusion**

### **6.1 Summary of the Development Process**

Developing **The Cannon Game** required **integrating physics-based mechanics, obstacle interactions, and dynamic UI updates**. The biggest challenges involved implementing **realistic projectile motion** and ensuring **precise obstacle behavior**. Transitioning to a **distance-based collision system** improved accuracy, while refining **wormhole and mirror interactions** ensured logical consistency.

Handling **real-time UI updates** was essential for **score tracking, shot counters, and level progression**. Using **Kivy’s scheduling functions**, we ensured **dynamic updates and proper screen scaling**. **Saving player scores** in the **Hall of Fame** required structured **file handling and error management** to prevent data loss.

### **6.2 Reflection on Key Learnings**

This project reinforced key technical skills, including:

* **Physics Simulation:** Implementing **gravity effects, velocity updates, and impact detection**.
* **Game Logic and Structure:** Managing **projectile behaviors and obstacle interactions** in a modular way.
* **Kivy UI Development:** Ensuring **real-time updates, responsiveness, and event handling**.
* **File Management:** Handling **persistent data storage for player scores** effectively.

Beyond technical aspects, we gained valuable **problem-solving and debugging experience**, particularly in refining **collision mechanics, UI synchronization, and projectile behavior**. The iterative development process strengthened our ability to **identify issues, refine implementations, and optimize performance**.

### **6.3 Future Improvements**

Several enhancements could make the game more engaging:

* **Audio Feedback** – Adding **sound effects and background music** to enhance immersion.
* **Multiplayer Mode** – Implementing **turn-based or online multiplayer** for competitive gameplay.
* **Expanded Gameplay Mechanics**
* **Enhanced Graphics** – Improving **animations, particle effects, and visual details**.

These additions would refine the **gameplay experience**, making it more dynamic and accessible.

## **7. References**

[1] “Artillery game — wikipedia.” <https://en.wikipedia.org/wiki/Artillery_game>.

[2] “Kivy: Cross-platform Python framework for GUI apps development.” <https://kivy.org/>.

[3] “Projectile motion — wikipedia.” <https://en.wikipedia.org/wiki/Projectile_motion>.

[4] “Kivy GitHub.” <https://github.com/kivy/kivy>.

[5] A. Hogan, “Kivy basics.” <https://www.youtube.com/watch?v=3GBNMBhm6UU>.

[6] 2023\_24\_cannon.pdf

* testing and results
* possible improvements

It should not be a log of the project development, although major changes from the initial design can be documented (along with the motivation to the change).  
The report should describe the algorithms used in the solution and their implementation.  
The main blocks of code should be described in order to ease the work of coders for future updates and improvements.

The discussion of the project will be carried out jointly by the team, which will have approximately 15 minutes for the presentation.

##### Aim of the project

The goal of the project is to apply the principles of the design of a system and its implementation studied during the course.  
The project will be based on an existing project or library, with the aim of extending the functionalities of the existing software.

##### Specifications release

The project specifications will be released during the last lesson of the course.

##### Teams

The project can be developed by a team of one or two students.  
Larger groups will be discouraged and will be required to carry on additional features.  
The project complexity will be sized based on a single-person team, for a person who is skilled in programming.  
The aim of the two-person team is to learn to work together and support each other.

##### Deliverables

The project deliverables consist in the code and the report.  
Few days before the presentation, the deliverables of the project (code and report) have to be submitted at the page:

<https://elearning.unipv.it/mod/assign/view.php?id=134075>

The material can be packaged in a single zip file (especially if the code consists in more that one file), possibly named with the matriculation number and the name of the team members.

The submission can be made by only one of the teams members (no need of submitting the same material for each member).

The submission page allows multiple submissions, so the deliverables can be updated (i.e., it can be used as a repository).

##### Deadlines

There are no strict deadlines to submit the project code and report, which have to be submitted at least two days before the discussion of the project.  
The date of the discussion can be set by appointment with the teacher, although for the January and February exam sessions a calendar will be proposed based on the exam enrollment lists.  
In any case, the possibility of changing the exam date will be provided, compatibly with the teacher availability.

##### Grading

Final exam projects will be graded on a maximum scale of 30 points, based equally on:

* The overall quality of the documentation.
* The understanding and appropriate use of Python features.
* The quality of the submitted code both in terms of performance and comments.
* Extra credit may be granted for solutions that are particularly creative.

##### Structure of the report

The project report is a (formal) technical report aimed to explain the design choices and their implementation.  
It should include:

* description of the goal(s) of the project (problems to be solved, constraints)
* assumptions, methods, procedures
  + motivations of the design choices
  + logical structure of the solution
  + implementation
* tools and resources
  + libraries used (with version)
  + references (where you took inspiration and materials)
* testing and results
* possible improvements

It should not be a log of the project development, although major changes from the initial design can be documented (along with the motivation to the change).  
The report should describe the algorithms used in the solution and their implementation.  
The main blocks of code should be described in order to ease the work of coders for future updates and improvements.

+++AGGIUNGI COMMENTI PIU DETTAGLIATI AL CODICE