Università degli Studi di Pavia

Bachelor degree in Artificial Intelligence

Cannon — Final exam project — A.Y. 2023/24

Stefano Ferrari

Computer programming, Algorithms and Data str., Mod. 1

Contents

date: January 10, 2024
version: 0.9

The final exam project consists in the design and implementation of a videogame called "Cannon", freely inspired to the artillery retrogames [1].

The Cannon game is constituted of a single cannon, which can be controlled by the player, a target, and optional obstacles. In the game, the player have to hit the target by firing projectiles that are subject to the physics of the world and can interact with obstacles under the rules explained in the next sections.

The game has to be realized using the framework `kivy` [2] (see Sect. 5.1). In order to allow some flexibility in the design, the specifications below described will refer to the simulation parameters through suitable constants, summarized in Sect. 4.

Although the Kivy framework allows to obtain apps that can be deployed (after some processing) to different operating systems, the final exam do not require that an actual app for a specific OS is realized. The source code is expected to run on any system with Python 3 interpreter and Kivy framework.

1. The Cannon game

Cannon is single-player 2D side-view artillery game [1]. The player aims at a target by controlling the elevation angle of the cannon, and the muzzle velocity and the type of the projectile.

The cannon is positioned at the left end of the screen, while the target is at the opposite side. The relative height of the target wrt. the cannon can have any value, both positive and negative.

Each round terminates when a projectile hits the target. The number of shots per round has to be specified at the beginning of the round and is possibly unlimited.

The game can be composed an optional number of rounds.

The goal of the player is to terminate the game in the least number of shots.

The projectile type states the allowed range of the muzzle velocity, and its interaction with the game physics and obstacles.

The scenario of each round can be fixed in advance or generated automatically. The designers can choose the first option for inventing schemes requiring skills to be solved. On the other hand, a suitable rule for increasing the number of obstacles, and hence the number of shots required to reach the target, can be devised.

1.1. Specifications

In the following, some game properties are described using constants, instead of specifying numerical values, to grant more flexibility in the design choices to the developers.

Lengths in the game can be measured in pixels. The size of game field is `SCREEN_WIDTH` $\times$ `SCREEN_HEIGHT` pixels.

Time can be measured in seconds. The frame rate, `FPS`, can be an optional value chosen as tradeoff of the performances.

As a consequence, the velocity can be specified in pixels/s.

1.2. Projectiles

At the beginning of each shot, the player can choose the type of projectile to be used.
The projectiles can be of the following types:

Bullet :

- Has a mass, `BULLET_MASS`, and is affected by the gravity.
- Follows a parabolic trajectory [3].
- Affects the obstacle in the impact point up to a given radius, `BULLET_RADIUS`.

Bombshell :

- Has a mass, `BOMB_MASS`, and is affected by the gravity.
- Follows a parabolic trajectory [3].
- Penetrates the obstacle for `BOMB_DRILL` from the impact point and affects the obstacle up to a given radius, `BOMB_RADIUS`.

Laser :

- Has no mass and is not affected by the gravity.
- Its muzzle velocity, `LASER_VEL`, is always constant.
- Follows a linear trajectory.
- Has a duration, which can be rapresented by the length of the impulse, `LASER_IMPULSE`.
- Penetrates the obstacles for `LASER_IMPULSE` from the impact point and affects the obstacles up to a given distance, `LASER_DIST`, of the trajectory.

Bullet dynamics

The physics of the game elements can be simplified. For instance, a parabolic trajectory can be considered as a starting point. However, the interaction with the other elements of the game can deviate the trajectory from this law (providing some approximation of the realistic effects and, hopefully, more fun). The projectile can be modelled with an instant velocity property, which at each frame provides a displacement in its position. At the same time, the velocity can be modified in its vertical component to simulate the gravity.

1.3. Obstacles

The trajectory of the projectiles can be hindered by game elements called obstacles.
The obstacles can be of the following types:

Rock :

- It is the main component of the enviroment.
- It can be constituted by atomic elements (e.g., small squares) that can be overlayed.
  - These elements are intended to be destroyed when hit by a projectile.

Bulletproof Mirror :

- Reflects the laser impulses.

- Cannot be destroyed by the projectiles.
- It can be represented by a segment (also curved).

Perpetio :

- As Rock, but cannot be destroyed by any projectile.

Optional types of obtacles:

Wormhole :

- Two-element indestructible obstacle.
- When a projectile hit one of the two elements, it continues its trajectory out of the other element.

Gravitonio :

- Locally changes the gravitational field. It can be realized as an element that affects both the vertical and horizontal components of the massive projectiles, radially with respect to the obstacle.
- Can be attractive or repulsive.

Elastonio :

- Reflects the massive projectiles.

2. Game functionalities

The game should allow the user to operate the basic operations:

- Save/Load
- Hall of Fame
- Help

3. Visualization

The suggested visualization is single-screen. It means that the whole field of the game is visualized in a single fixed window. Alternatively, game field can be extended to a portion of space larger than a window and the player can be allowed to explore the field by scrolling, or the scrolling can be allowed only after the shot, following the trajectory of the bullet.

4. Constants

In this section, some parameters that characterize the game are reported as constants. Their values are a design choice to customize the dynamics of the game. Referring to them in the code allows to modify easily the overall dynamics.

`SCREEN_WIDTH`, `SCREEN_HEIGHT`: The size of the field of the game, in pixels.

`FPS`: The frame rate of the game, in frame/s.

`BULLET_MASS`, `BOMB_MASS`: Bullet and Bombshell projectiles parameter that affects the muzzle velocity range.

`BULLET_RADIUS`, `BOMB_RADIUS`: Bullet and Bombshell projectiles parameter that affects the (spherical) range of the damage.

`LASER_DIST`: Laser projectiles parameter that affects the (cylindrical) range of the damage.

`BOMB_DRILL`, `LASER_IMPULSE`: Parameter of the Bombshell and Laser projectiles that represents the space travelled inside the obstacles.

`LASER_VEL`: Laser muzzle velocity (cannot be controlled by the player).

5. Resources

5.1. The `kivy` framework

`kivy` is a free and open-source Cross-platform Python Framework for GUI apps Development [2].

Besides the Kivy project site [2], source and information can be found at the Kivy GitHub [4].

A simple but enlightening videotutorial is “Kivy Basics”, by Amanda Hogan [5].

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.