Università degli Studi di Pavia

Bachelor's in Artificial Intelligence

The Cannon Game

Final Exam Project

A.Y. 2023/24

Alessandro Di Piano

Pelinsu Topaloglu

Computer programming, Algorithms and Data Structures, Module 1

TABLE OF CONTENTS

Summary

This project is a Kivy-based interactive game designed to engage players with a simple artillary game. The game involves controlling a cannon to shoot projectiles at targets while destroying obstacles like rocks. Key gameplay elements include firing bullets, bombshells, and lasers, each with unique behaviors and effects. The game is developed using Python, and the library called Kivy.

The Goals of the Project

The primary goal of this project is to create an engaging and interactive experience where players navigate the game to achieve a high score. The player must control a cannon, destroying rocks on the path, while simultaneously hitting the target. Key gameplay elements include shooting projectiles and deploying powerful weapons like bombshells.

The game includes various screens such as the menu screen, game screen, and game over screen to provide a seamless user interface. Additionally, the game incorporates features like name input for high scores, dynamic backgrounds, and explosions for visual appeal.

Constraints in the project include maintaining a smooth and responsive user experience within the limits of computational resources, such as ensuring efficient collision detection and rendering even with numerous objects on the screen. The game must also handle different screen resolutions and input methods, providing accessibility and playability across various devices.

Another constraint is balancing the difficulty to keep players engaged without causing frustration. This involves fine-tuning the speed and spawn rates of obstacles, the effectiveness of weapons, and the resilience of targets. Furthermore, the game should provide clear instructions and help screens to guide new players, while offering depth and complexity to keep experienced players challenged.

Overall, the project aims to deliver a fun game that can captivate players, encouraging sharing among friends.

Assumptions

- The project assumes that players have a basic familiarity with standard gaming concepts, such as navigation, and shooting. It is designed so that players will use common input devices like keyboards.
- The physics of the projectile follows the real world case, in addition to some offset. This is explained in more detail in Implementation Section.
- Rocks are not intended to block the cannon movement as the game has a different purpose for them. This concept is elaborated on in Implementation Section.

Methods

The game employs an object-oriented programming approach, encapsulating game elements within classes to manage their behavior and interactions effectively. Efficient collision detection algorithms are implemented to manage interactions between game objects like projectiles and obstacles. For rendering graphics, the game utilizes images to draw game objects, backgrounds, and effects on the screen, ensuring smooth and visually appealing graphics.

Procedures

The game initialization process involves loading game assets, such as images, from predefined paths and setting up game settings, including screen dimensions, frame rate, and initial game state. During gameplay, the main game loop continuously handles user inputs, updates game states, checks for collisions, and renders the updated game state on the screen. Screen management procedures ensure smooth transitions between different screens, such as the menu, gameplay, and game over screens, maintaining a consistent UI and navigation flow. Scoring and progression mechanisms track player scores, providing feedback through visual cues such as the number of bullet used. The power bar is displayed in the normal game screen, moreover, in hard mode the remaning time is also shown. Game over and restart procedures detect game-over conditions, display the game-over screen with options to restart or return to the main menu, and reset the game state for new sessions while maintaining high scores or

player progress as appropriate. A help screen guides new players through basic mechanics and controls.

These assumptions, methods, and procedures collectively contribute to the development of a this engaging and user-friendly game that performs well across different devices and provides an enjoyable experience for players.

Motivations Behind Design Choices

The design choices in this game project are motivated by the desire to create an engaging, intuitive, and performance-optimized gaming experience. The desing is intended to be simple, the background color helps to not lose track of the small target and the cannon is intended to be an old-fashioned type.

In the menu screen, the leaderboard button is displayed in the shape of a medal and the help button as a question mark. This is done in order to visually appeal the user as well as make navigating easier for children. Scoring and progression mechanisms, along with game over and restart procedures, are implemented to provide players with clear feedback.

The name input option allows players to tailor the game, it takes after the same as the classical arcade, namely, in case a name is entered twice the diversion with the most excellent score is kept and the old best score is discarded. This reflects a commitment to user experience, allowing players to tailor the game.

Overall, these design choices are driven by the goal of creating a well-rounded, engaging, and accessible game that delivers a high-quality experience to players across different platforms.

Logical Structure Of The Solution

The logical structure of this game is built around a clear and modular organization, which ensures that the game's components are well-defined and interact seamlessly. At the core of the solution is the main application class (`MainApp`), which serves as the entry point and overall manager for the game. This class initializes the game, sets up the main loop, and manages transitions between different game states such as the main menu, gameplay, and

game over screens. By centralizing these functions, the `MainApp` class provides a cohesive framework that ties all other components together.

Each game screen, such as `MenuScreen`, `GameScreen`, `GameOverScreen`, and `LoserScreen`, is encapsulated within its own class. These classes are responsible for rendering their respective interfaces, handling user inputs, and transitioning to other screens as needed. This separation of concerns ensures that the logic for each screen is self-contained, making it easier to manage and modify. For example, the `MenuScreen` class handles navigation options, while the `GameScreen` class focuses on the main gameplay loop, updating game objects, and rendering the game world.

Game objects, such as the player character, enemies, projectiles, and obstacles, are represented by specific classes like `Rock`, `Laser`, `Bullet`, `Bombshell`, `Mirror`, `Cannon`, and `Target`. Each of these classes encapsulates the properties and behaviors of the corresponding game elements, including movement, collision detection, and interactions with other objects. This object-oriented approach allows for easy addition of new game elements or modification of existing ones without impacting other parts of the codebase. For instance, adding a new type of projectile would involve creating a new class with its own behavior, which can then be integrated into the existing game loop.

Collision detection and resolution are crucial components of the logical structure. The game employs built-in collusion detection systems as well as a custom method to manage collisions between multiple objects. This ensures that even as the number of game objects increases, the performance remains stable.

Input handling is another vital aspect of the logical structure. The `Keyboard` class manages user inputs, translating them into game actions such as movement, shooting, and menu navigation. This event-driven approach ensures that the game remains responsive to player actions, providing a smooth and intuitive user experience.

The rendering process is handled by a dedicated part of the game loop, which draws all game objects, backgrounds, and UI elements on the screen. The `Background` class manages the visual environment, ensuring that the game world remains visually appealing and cohesive.

Special effects, such as explosions managed by the `Explosion` class, add to the visual feedback and enhance the gameplay experience.

Overall, the logical structure of the solution is designed to be modular, scalable, and maintainable. By organizing the game into clearly defined classes and separating concerns, the project ensures that each component can be developed, tested, and debugged independently. This modularity also facilitates collaboration among developers, as different team members can work on distinct parts of the game without causing conflicts.

Implementation of the Game

The implementation of the game involves the development of various components that work together to create a cohesive and interactive gaming experience.

Main Application (`MainApp`)

The `MainApp` class serves as the entry point for the game. It initializes the game window, sets up the main loop, and manages transitions between different screens (e.g., menu, gameplay, game over). Upon starting, `MainApp` initializes essential game settings such as screen dimensions, frame rate (FPS), and game states. It continuously updates the game state based on user inputs and internal game logic and renders the appropriate screen.

Screens

- MenuScreen

The `MenuScreen` class is responsible for displaying the main menu, which includes options like starting the game, accessing settings, and exiting. It captures user inputs to navigate between options and transitions to the `GameScreen` when the player chooses to start the game.

- GameScreen

The `GameScreen` class is where the main gameplay takes place. It handles the game loop, which includes updating game objects, detecting collisions, and rendering the game world. This class manages the game state during active play, including player actions and interactions between objects. It also provides differences based on the mode the user chooses. Even though

the structure is the same for both hard mode and easy mode, the horizontal movement of the cannon is restricted in the hard mode. Moreover, a time limit to win the game is implemented.

- GameOverScreen and LoserScreen

The `GameOverScreen` and `LoserScreen` classes display the end-of-game messages and options. When the game ends, these screens provide the score of the user, and offer options to restart or return to the main menu.

- Help Screen

A help screen guides new players through the game's basic mechanics and controls. This is implemented as an additional screen that can be accessed from the main menu.

Game Objects

- Projectiles:

Projectiles, namely bullets, lasers, and bombshells are represented by their own classes such as `Bullet`, `Laser` and `Bombshell`. In the calculation of the initial position of the projectiles, the vertical and horizontal coordinates of the cannon object are used. The projectiles that are affected by gravity (bullets and bombshells) are intended to follow the real world case. This is ensured by the formula of parabolic motion: $x_{initial} + vt - \frac{1}{2}gt^2$. This formula is adjusted according to the angle the muzzle of the cannon has. The laser, on the other hand, follows a straight line, in which case the gravity constant in the equation mentioned earlier is set to 0.

- Obstacles:

Obstacles include rocks and a mirror. Each of these obstacles are created within their own classes through kivy. These objects have properties like position and movement patterns. As mentioned in the assumptions section, rocks are not intended to stop player movement, they are rather a side quest that needs to be completed before unlocking the laser option.

- Collision Detection:

Collision detection is a critical part of the game, ensuring that interactions between game objects are accurately processed. For the bullet and laser objects, this is done through the "collide_widget" function provided by kivy. On the other hand, the bomb object utilizes a

method to check the radius to detect collusions. Each game object's update method includes collision checks, and upon a detection, the object who collide are removed.

- Input Handling:

Input handling is managed by the class `Keyboard`. This class captures user inputs (e.g., key presses) and translate them into game actions. The main game loop processes these inputs to update the game state accordingly. The following keys are used for the following functions:

- “a”: move the cannon to the left
- “d”: move the cannon to the right
- “w”: turn the muzzle of the cannon up
- “s”: turn the muzzle of the cannon down
- “b”: fire the bombshell
- “l”: fire the laser (this functionality is enabled only after the rocks have been cleared)
- spacebar: fire the bullet
- “m”: increase the firing power
- “n”: decrease the firing power

Used Tools And Resources

Tools

- Kivy Library (Version 2.3.0): This is the primary framework used to build the graphical user interface and handle interactions. In this project, the following functionalities provided by kivy are mainly utilized:
    - Config: Used to set the game to fullscreen mode.
    - App: The base class for creating the application.
    - Widget: The base class for all widgets used in the game.
    - Properties: Used to manage and bind state within widgets.
    - Image: Used to display images. (The cannon, the background)
    - Clock: Used for scheduling and timing events.

- Animation: Used to create animations. (The rotation of the cannon and the movement of the target)
- Label: Used to display text.
- Window: Used to access the window properties and handles keyboard inputs.
- ScreenManager: Manages different screens in the application.
- BoxLayout: Arranges widgets in a box layout.
- Button: Used to create interactive buttons. (Start game, reset game)
- TextInput: Allows player input. (Entering a nickname)

- Python Standard Libraries
  - Random: Generates random numbers for gameplay elements like enemy positions.
  - Time: Manages timing of the hard mode.
  - Os: Handles file operations and paths, checking for file existence and manipulating paths.
  - Json: Reads and writes JSON data for saving and loading game states and leaderboards.
  - Math: Provides mathematical functions for calculations involving angles, distances, and projectile motion.
  - Cannon_constants: Stores configuration values and constants (projectile speeds etc.) used throughout the game.

Resources

- Image Files: `rock.png`, `un_rock.png`, `cloud.png`, `explosion.png`, `target.png`.
- Text Files: `leaderboard_text.txt`, `help_text.txt`
- JSON Files: `saved_game.json`

All these tools and resources collectively enable the development of an interactive, and maintainable game, ensuring a cohesive user experience and efficient game management.

Testing

Unit Testing: Testing individual components or functions to ensure they work correctly in isolation, such as the fire methods.

Integration Testing: Ensuring that different parts of the application work together as expected. An example of this is the transitions between screens like NameInputScreen, MenuScreen, GameScreen, GameOverScreen, and LoserScreen.

Functional Testing: Testing the game's functionality to ensure it meets the specified requirements. For instance, the firing mechanics of the cannon, movement of projectiles, collisions with rocks and targets, and updating of scores.

User Interface Testing: Ensuring that the user interface elements are displayed and function correctly. These include the buttons such as start game, reset game, and save the game.

Results

After performing these tests, it was observed that individual functions for file operations and leaderboard management work correctly and the screen transitions happen smoothly while the game states are managed correctly. Additionally, the gameplay mechanics work efficiently and the UI elements are visible, functional, and correctly capture user inputs.

Possible Improvements for the Project

A possible improvement area for the game is to enhance the collision detection with more accurate methods such as pixel-perfect collision. This could greatly improve the user experience. Additionally, improving the graphics used in the game, as well as adding sound effects could also serve this purpose. Another way to improve the game experience could be to introduce a level progression system within the game instead of 2 difficultiy settings. Furthermore dynamic menus could be implemented. By implementing these improvements, the game can provide a richer, more engaging, and polished experience, catering to a wider audience and ensuring long-term player engagement.

Bibliography

Freepik. "A cannon isolated." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/cannon-isolated_4607679.htm#fromView=search&page=1&position=2&uuid=948372c9-e6b4-45f9-bb09-491561fc4a1e>.

Freepik. "Cartoon nature landscape dirt road go along field." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/cartoon-nature-landscape-dirt-road-go-along-field_25864527.htm#fromView=search&page=1&position=1&uuid=92f6f82a-9e6b-4713-8d08-9c9281b5f8ae>.

Freepik. "Hand drawn download buttons label collection." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/hand-drawn-download-buttons-label-collection_34643585.htm#fromView=search&page=1&position=12&uuid=d4359fc2-12e5-4389-8ee3-1b4844feb0b7>.

Freepik. "Web button set in flat style." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/web-button-set-flat-style_3211594.htm#fromView=search&page=1&position=12&uuid=4232193b-f96f-4855-905e-823b2cbdd7a1>.

Hogan, Amanda. "Kivy Basics." 07 05 2017. *Youtube*. 04 2024. <https://www.youtube.com/watch?v=3GBNMBhm6UU>.

Kivy. *Kivy*. n.d. 05 2024. <https://kivy.org/about.html>.

MacroVector. "Award medal with red ribbon." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/award-medal-with-red-ribbon_10603705.htm#fromView=search&page=1&position=9&uuid=c8efde8f-50a1-4013-aff2-496c3bf46fd1>.

MacroVector. "Holes and Bullets Collection." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/holes-bullets-collection_6415521.htm#fromView=search&page=1&position=11&uuid=4183322a-280c-416c-8614-9c65ecc46c8d>.

MacroVector. "Rock garden wild nature stones with sparse vegetation green growth moss 4 realistic landscape elements illustration." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/rock-garden-wild-nature-stones-with-sparse-vegetation-green-growth-moss-4-realistic-landscape-elements-illustration_21253088.htm#fromView=search&page=1&position=13&uuid=ca95cbc5-b5d3-46ae-ae11-18b71a26d47b>.

Pch.Vector. "Heaps of rock stones set." n.d. *Freepik*. 03 2024. <https://www.freepik.com/free-vector/heaps-rock-stones-set_9175312.htm#fromView=search&page=1&position=3&uuid=36e7f13b-8f6e-4063-a381-2fbee36fffaf>.

Pixabay. "Question Mark Button Red." n.d. *Pixabay*. 03 2024. <https://pixabay.com/vectors/question-mark-button-red-round-31190/>.

Sandberg, Eric. "Python Game Development with Kivy - Flappy Bird Tutorial." 27 11 2019. *Youtube*. 03 2024. <https://www.youtube.com/watch?v=2dn_ohAqkus>.

Serway, Raymond and John Jewett. *Physics for Scientists and Engineers with Modern Physics*. Boston: Cengage, 2017.

Wikipedia. 14 January 2024. 06 2024. <https://en.wikipedia.org/wiki/Artillery_game>.