School of Computer Science & Electronic Engineering,

Subject:

Main Programming Assignment: A physics-based game

(A completed bike jumping game in 2D)

Submitted as a part of:

CE812-7-AU : Physics-Based Games

Author:

Sayed Maqbool Ahmed Inamdar

(**Registration: 2204389**)

Module Supervisors:

Dr Michael Fairbank

Finished on April 14, 2023.

Contents

Synopsis:

This report presents the development of a 2D bike adventure game using the Box2D physics engine in Java, inspired by hill mountain bike racing games. The game focuses on providing a fun and challenging experience for players, as well as showcasing the capabilities of both Java and Box2D. The development process involved setting up the environment, designing levels, implementing game logic, integrating Box2D, and testing and refining the game mechanics.

Key methods in the game include env(), torquerotation(), win(), gameover(), and resetGame(), which handle various aspects such as environment setup, bike rotation, win and game over conditions, and game reset. And the modified BasicView Class that handles the timer and camera translation for the game. The bike model consists of two particles representing wheels and a single polygon representing the body. The camera system keeps the bike's body at the centre of the screen, providing an engaging visual experience. A background gradient and timer display enhance the game's presentation.

Challenges faced during development included handling complex physics involved in bike movement and balancing, as well as optimizing performance. Despite these difficulties, the final game provides an enjoyable experience for players and serves as a testament to the power and versatility of Java and Box2D.

Introduction:

In the modern era of gaming, physics engines play a crucial role in providing realistic and immersive experiences for players. One such physics engine is Box2D, a popular open source 2D physics engine for games. Box2D is written in C++ and has been ported to various programming languages, including Java. In this report, we discuss the development of a 2D bike adventure game using Box2D in Java. The game is inspired by hill mountain bike racing games, featuring a bike navigating through various terrains while overcoming obstacles and maintaining balance.

The game's primary objective is to provide a fun and challenging experience for the player, while also showcasing the capabilities of the Box2D physics engine. The game features a bike controlled by the player, which must traverse through a series of levels with varying difficulty. The player's performance is evaluated based on their ability to complete the levels and the time taken to do so.

The development of the game was done using the Java programming language and the Box2D physics engine. The choice of Java as the programming language was due to its widespread usage, ease of learning, and rich ecosystem of libraries and tools. Box2D was chosen as the physics engine due to its extensive features, performance, and ease of integration with Java.

The game's development process involved the following steps:

1. Setting up the development environment, including the required libraries and tools.

2. Designing the game's levels and creating the necessary assets, such as colours and sounds.

3. Implementing the game's logic, including the physics simulation, user input handling, and game state management.

4. Integrating the Box2D physics engine into the game to handle collision detection, response, and other physics-related aspects.

5. Testing and refining the game's mechanics and features to ensure a smooth and enjoyable experience for the player.

Throughout the development process, various challenges were encountered, such as handling the complex physics involved in bike movement and balancing, optimizing the game's performance. However, with the help of the Box2D physics engine and the Java programming language, these challenges were overcome, resulting in a fun and engaging game that showcases the power and versatility of both the programming language and the physics engine.

In this report, I provide an in-depth analysis of the game's mechanics and features, focusing on the implementation of the Box2D physics engine in Java. We discuss the various methods used in the game, such as environment setup, handling win and game over conditions, game reset, and torque rotation. We also explore the physics concepts employed in the game, such as gravity, linear drag force, and the use of joints to connect different parts of the bike.

Important Methods:

env():

The env() method is responsible for setting up the initial environment of the game. It defines the particles and their respective positions, velocities, colours, and drag forces. In this bike adventure game, the particles are the wheels of the bike.

The method starts by creating two BasicParticle objects, p01 and p02, with their respective positions, velocities, radii, colours, densities, and linear drag forces. These particles represent the wheels of the bike. After creating the particles, the method adds them to the 'particles' list.

Next, the env() method defines the joints between the particles (wheels) and the bike's body (base) represented by the polygon (Rectangle) . This is done by creating a RevoluteJointDef object, which is used to create two joints: one between p1 (first wheel) and the base, and another between p2 (second wheel) and the base. These joints ensure that the wheels rotate around the bike's body.

torquerotation():

The torquerotation() method handles the rotation and torque applied to the bike in response to user input. This method is responsible for the bike's movement and interaction with the environment. It reads the user's input through the BasicKeyListener class and applies the necessary torque to the bike's particles (wheels) and polygons (bike frame).

The method first checks whether the left or right rotation key is pressed. Depending on the key pressed, it applies a positive or negative torque to the particles (wheels) and checks for additional key inputs. For example, if the shift key is pressed, the torque applied is increased, providing a boost in speed. Similarly, if the space key is pressed, the angular velocity is set to zero, essentially applying a brake to the bike.

Furthermore, the torquerotation() method also handles wheelies by checking if the up or down key is pressed along with the right key. Depending on the key combination, the bike frame's torque is adjusted to perform a front or rear wheelie.

The torquerotation() method is an essential component of the entire code as it governs the rotation of the bike's wheels and body, as well as the user's ability to perform various actions, such as wheelies. This method is designed to handle user input and apply the appropriate torque to the bike's wheels and body.

Below is a breakdown of the method's key components:

Variable Initialization: At the beginning of the method, we retrieve the two particles representing the wheels (pp and qq) and initialize the left (ltorque) and right (rtorque) torque values.

```
BasicParticle pp = particles.get(0);
BasicParticle qq = particles.get(1);

float ltorque = 3f;
float rtorque = -3f;
```

Handling Rotation Input: The method checks for specific key presses using the BasicKeyListener class. If the user presses the left key, the left torque value is applied to both wheels. Similarly, if the user presses the right key, the right torque value is applied to both wheels.

The Shift key doubles the torque value applied, making the rotation more aggressive.

The Space key stops the rotation by setting the angular velocity of the wheels to zero.

Performing Wheelies: The method also allows the user to perform wheelies by applying torque to the bike body (represented by the polygon). When the user presses the key to move down or thrust, a torque value (wheelieTorque) is applied to the bike body, making it perform a wheelie. The positive value represents a front wheelie, while the negative value represents a rear wheelie.

```
if (BasicKeyListener.isDownKeyPressed() == true)
{
    float wheelieTorque = -10;
    for (BasicPolygon pp1 : polygons)
    {
        pp1.body.applyTorque(wheelieTorque);
    }
}

if (BasicKeyListener.isThrustKeyPressed() == true)
{
    float wheelieTorque = 10;
    for (BasicPolygon pp1 : polygons)
    {
        pp1.body.applyTorque(wheelieTorque);
    }
}
```

win():

The win() method is responsible for detecting when the player has won the game and providing the option to retry for a better score or close the game. The player wins the game when the bike's body (represented by the polygons) reaches a specified position.

The method first checks if the player has won by comparing the x-position of polygon that is attached to the bike with the specified position threshold. If that polygon is beyond the threshold, the player has won, and a sound effect is played.

Next, the method displays a JOptionPane showing the player's score (based on elapsed time) and asking if they want to try again for a better score. If the player chooses to retry, the 'resetGame()' method is called to reset the game environment. If the player chooses not to retry, another JOptionPane asks if they want to close the window. If they choose to close the window, the program exits; otherwise, the game remains in its current state and player can explore the physics and environment.

gameover():

The gameover() method is responsible for detecting when the game is over and providing the player with the option to restart or close the game. The game is considered over when the bike falls below a certain height.

The method first checks if the game is over by comparing the position of the bike with the specified height threshold. If the bike is below the threshold, the game is considered over, and a sound effect is played ”Ohh Nooo”.

Next, the method displays a JOptionPane asking the player if they want to restart the game. If the player chooses to restart, the player plays the game again. If the player chooses not to restart, another question is asked, if they want to close the window. If they choose to close the window, the program exits; otherwise, the game timer is stopped, and the game remains in its current state.

resetGame():

The resetGame() method is responsible for restarting the game when the user chooses to do so. It reinitializes the game world with gravity, clears the lists of particles, polygons, barriers, and connectors, and creates a new instance of the BasicPhysicsEngineUsingBox2D class. The method then resets the elapsed time, printt flag, and timerStopped flag to their initial values.

The resetGame() method also reassigns the particles, polygons, barriers, and connectors from the new instance to the current instance of the BasicPhysicsEngineUsingBox2D class. After this, it calls the env(), gameover(), and other methods to set up the game environment and handle game states.

Tuning The Values:

The torque values used in the torquerotation() method can be tuned to control the responsiveness of the bike's wheels and body. The variables ltorque and rtorque represent the left and right torque values, respectively. By increasing or decreasing these values, you can adjust the rate at which the bike rotates in response to user input. In the given code, ltorque is set to 3 and rtorque is set to -3. You can experiment with different values to achieve the desired bike behaviour. I did select those value because for me I thought those value are more natural to the physics.

The wheelieTorque value is responsible for the torque applied when performing a wheelie. The positive and negative values represent the torque applied to the front and rear wheelies, respectively. Adjusting these values will modify the wheelie's intensity, making it easier or harder to perform. In my code, wheelieTorque is set to 10 for front wheelies and -10 for rear wheelies.

Design and Aesthetics:

Bike Model

The bike model consists of the following components:

- Two particles representing the front and rear wheels of the bike. These particles have physical properties such as mass, position, and velocity, which are updated during the simulation. Forces and torques are applied to these particles to control the rotation of the wheels and simulate the effects of acceleration, deceleration, and braking.
- A single polygon representing the bike's body. This simplifies the model and reduces computational complexity. The polygon has physical properties like mass, position, and velocity, and its interaction with the particles representing the wheels helps to simulate the bike's movement and stability.

Camera System

The camera system in the simulation is designed to keep the bike's body at the centre of the screen. This is achieved by translating the camera's position based on the position of the bike's body polygon. The camera follows the bike's movement throughout the simulation, ensuring that it remains the central focus. This is accomplished using the following steps:

- Obtain the position of the bike's body polygon from the game.polygons list.
- Calculate the required camera offset values (offsetX and offsetY) by subtracting the polygon's screen coordinates from the desired screen centre coordinates.
- Translate the Graphics2D object by the calculated offsets (offsetX, offsetY) before rendering the simulation objects.
- After rendering the objects, translate the Graphics2D object back by the negative offsets (-offsetX, -offsetY) to restore its original position.

Background Gradient

A visually appealing background gradient is created using LinearGradientPaint, which blends multiple colours across the screen. The gradient colours are defined in an array (gradientColors), and their corresponding fractions (gradientFractions) determine their positions within the gradient. The gradient is applied to the Graphics2D object before rendering the simulation objects.

Timer and Score Display

A timer is implemented to display the elapsed time as the simulation runs. The timer is drawn on the screen using the drawTimer method, which sets the font, colour, and position of the text. The elapsed time is formatted as a string and displayed as the score.

Bug Noticed

I noticed a bug in the code where sometimes the bike automatically accelerates when the game is restarted by pressing “Yes” option after the game over, and continues to accelerate until the right key which is used to accelerate the bike is pressed or any relevant key which is used to control the vehicle is pressed.

Reflection

As I sit back and reflect on my achievements in developing the 2D bike adventure, I feel a sense of happiness and accomplishment. This assignment was a significant undertaking, and I am pleased with how it turned out.

Overall, the assignment went smoothly, with few major setbacks. The initial setup of the development environment and integration of the Box2D physics engine into the game were relatively easy, thanks to the comprehensive documentation and resources available.

However, I encountered some challenges when it came to implementing the game's logic and mechanics, such as handling the complex physics involved in bike movement and balancing. It required a lot of experimentation and tweaking to get the bike's movement and stability right. Additionally, optimizing the game's performance while maintaining its realism and visual appeal was a challenge.

Despite these difficulties, I am pleased with the final submission of the game. It provides a fun and challenging experience for players, with smooth and realistic bike movement and a visually appealing environment. I am particularly proud of the camera system, which keeps the bike's body at the centre of the screen, providing a user-friendly experience.

In conclusion, the development of this 2D bike adventure game using Box2D was a rewarding experience that showcased the potential of combining a powerful programming language with a versatile physics engine. Despite some challenges along the way, I am happy of the final product and look forward to further refining and improving it in the future.

Conclusion

This bike physics simulation provides an effective representation of the bike's movement and stability using a simplified model with two particles for the wheels and a single polygon for the body. By implementing a camera system that keeps the bike's body at the center of the screen, the simulation offers a user-friendly and visually engaging experience. The addition of a background gradient and a timer display enhances the overall presentation of the simulation.

References:

[1] M. Fairbank, "BasicPhysicsEngineUsingBox2D," 2016. [Online]. Available: https://moodle.essex.ac.uk/mod/folder/view.php?id=284209 . [Accessed: 14- Feb- 2023].

[2] J. E. Box2D, "Box2D: A 2D Physics Engine for Games," Box2D. [Online]. Available: https://box2d.org .[Accessed: 14-Feb-2023].

Appendix

1. This appendix provides information about the BasicRecangle class that was created by the author of this document, with the help of module supervisor Dr Michael Fairbank.

The BasicRecangle class is a part of the package pbgLecture5lab_wrapperForJBox2D. It extends the BasicPolygon class and defines a rectangular object using the JBox2D library. The class includes properties such as size, position, velocity, height, width, colour, mass, and rolling friction.

The purpose of the BasicRecangle class is to create a rectangular object in a simulation environment, enabling the user to define its properties. The class leverages the JBox2D library to manage the physics and rendering of the objects within the simulation environment.

2. Modifications in the BasicView class

- Background colour changed to gradient using multiple colours.
- In the paintComponent methos the offset values for the x and y axis is calculated based on the position of the bike body frame in this case the position of the polygon. And after that applied a translation to the Graphics2D object using the calculated offset values i.e., offset and offset. This will move the camera always positioning our bike at the centre.
- DrawTImer methos is added that takes a Graphics2D object as a parameter. This method sets the the properties of the timer such as colour, font and size, and draw a string displaying the score at the top left corner of the screen.