Jeff-Boy and Pearl-Girl

Jeffrey Cheng, Chau Truong

December 14, 2024

Abstract

Jeff-Boy and Pearl-Girl is a cooperative puzzle-platform game inspired by the popular 2D game *Fireboy and Watergirl*. Built using `Three.js`, the game challenges players to navigate two characters through maps, collaborating to operate button mechanisms, collect ice creams, and reach their respective doors. By incorporating elements like character-specific controls and strategic gameplay, the project emphasizes coordination, teamwork, and problem-solving.

1 Introduction

Inspired by *Fireboy and Watergirl*, *Jeff-Boy and Pearl-Girl* is a cooperative puzzle-platform game built using `Three.js`. The two titular characters are moved by using the arrow keys (Jeff-Boy) and the WASD keys (Pearl-Girl). Ice creams are collected by both characters as they progress through the level, and button mechanisms must be operated strategically for them to win. A level is won once Jeff-Boy and Pearl-Girl reach their respective doors. Our goal was to create a fun and engaging puzzle game that emphasizes teamwork and problem-solving.

1.1 Previous Work

Fireboy and Watergirl [1], a 2D cooperative puzzle-platform game, served as a major inspiration for this project. In the game, Fireboy is moved using the arrow keys and Watergirl is moved using the WASD keys. There are puddles of fire/lava and water; Fireboy can only travel through the fire puddles, while Watergirl can only travel through the water puddles. If either travel through the puddle of the opposite element, they will die and the level must be restarted. There are also puddles of green acid that can kill both characters. Red and blue diamonds must be collected throughout the game by Fireboy and Watergirl, respectively, and button and lever mechanisms are operated to aid them progress through the level and reach their respective doors. As of today, there are six games in the series, with the sixth, *Fireboy and Watergirl: Fairy Tales*, released in November 2021.

2 Methodology

2.1 User Controls and Character Movements

Both characters in the game can move left, right, up, and down. To implement the characters' movements, we utilized Javascript event handlers for keypress and keyup events. When a keypress event occurs, we determine whether the key pressed is an arrow key or a WASD key. If an arrow key is pressed, Jeff-Boy moves in the appropriate direction; similarly, if a WASD key is pressed, Pearl-Girl moves in the corresponding direction ('w' = up, 'a' = left, 'd' = right). We did not implement a reaction for the 's' or ArrowDown keys since gravity in the game world causes the characters to fall naturally to a lower platform when no ground supports them. This simplifies the movement mechanics while maintaining realism within the game's mechanics.

Jeff-Boy/Pearl-Girl continues to move horizontally in the specified direction until the ArrowLeft/'a' or ArrowRight/'d' keys are released, at which point the character stops. The upward movement is caused by an upward force being applied to the character, which we will discuss in the following subsection. As a result, vertical movements are constrained by a maximum height. Once the character reaches the maximum height, they fall naturally due to gravity. Continuously pressing of the ArrowUp or 'w' key does not enable the character to "fly"; they must land on the ground before moving upward again.

If a character is in the middle of a diagonal upward movement (e.g., when the ArrowUp/'w' and a corresponding horizontal key are pressed simultaneously) and the keys are released, the character will fall immediately, following a downward trajectory influenced by gravity.

2.2 Character Movement Animations

Before implementing movement animations, we used Quaternions from `Three.js` and `cannon.js` to orient each character upright and facing the camera as a default. This ensures consistent positioning whenever a key is not pressed.

Upon a corresponding horizontal key press (ArrowLeft/ArrowRight for Jeff-Boy and 'a'/'d' for Pearl-Girl), the character's Quaternion is adjusted so that they face the desired direction, and a walking animation is triggered. Horizonal movement is achieved by setting a velocity for the character in the direction they are facing, enabling smooth traversal across the ground.

When the ArrowUp or 'w' key is pressed, a jumping animation is triggered and a force is applied to the character that moves them upward. To ensure a smooth transition between animation and physics, the upward force is applied 10% of the way into the jumping animation. This timing creates a more seamless and visually coherent movement. The upward force is constrained to ensure that the character can only ascend to a maximum height before gravity causes them to fall.

If both a vertical (ArrowUp or 'w') and a horizontal key are pressed simultaneously, the appropriate character's movement combines these actions. The character will face in the desired direction specified by Quaternions, a velocity is set for horizontal movement, and an upward force is applied. This allows the character to move diagonally upward. This capability is essential for gameplay elements like jumping onto a box or onto a platform.

The walking and jumping animations for the characters were sourced from Mixamo [2].

2.3 Game Features

2.3.1 Ice Creams

Our game features collectible ice creams, which are placed around the level and assigned a "region of collection", defined by two coordinates $\{x_{min},\ y_{min},\ z_{min}\}, \{x_{max},\ y_{max},\ z_{max}\}$ that represents an axis-aligned box. If either character's position coordinate lies inside the region of collection, the ice cream is "collected" and removed from the screen. An alternative implementation would be to assign the ice cream one coordinate and define the proximity needed to collect an ice cream, but we opted for consistency and used a region of collection for all of our game features. We did not implement an ice cream collection counter, but would implement one given more time.

2.3.2 Stepping Boxes

Stepping boxes are just boxes that the characters can push around and jump on top of to reach higher platforms. Their physical bodies are defined to be a simple box that is short enough for the players to jump on and tall enough to reach levels of the stage.

2.3.3 Buttons and Movable Platforms

Buttons and platforms are the main feature that necessitates cooperation between the two characters. Our buttons are displayed as rocket-ships, and if one character stands inside of it, it raises a platform elsewhere to elevate the other character to a higher level. However, since both characters must make it to the top and one person must stand on the button to raise the platform, the character that took advantage of the platform must find an alternate way to bring their partner up with them.

Buttons were implemented using the region of collection from section 2.3.1. If either character stands within the region of collection of a button, the target position of a platform is elevated according to a height parameter. Using the platform's update function, which is repeatedly called, if the platform is not at its target position, it gradually adjusts its height accordingly.

2.3.4 Lava Pit

2.3.5 Doors

2.4 Collision Handling

To handle collisions in the game, we used physical boxes to represent the boundaries of the characters, the ground, the movable platforms, and the stepping boxes. Each of these features in the game was assigned a bounding box created using the `Box` class from `cannon.js`.

cannon.js handles all collision detections and responses. When a collision occurs—for example, when a character pushes a stepping box—cannon.js resolves the interaction by calculating the positional adjustments in real time.

somewhere in here, mention that there are front, back, left, and right boxes that prevents the characters from sliding off the platforms

and also friction (characters and platforms do not have friction, but the stepping boxes do)

3 Results

character movements, quaternions were correct. aligns with common conceptions of physics except for the parts that we changed (example, can change velocity direction while in the air with key presses)

minimizing irregular interaction between objects (cannot go through objects- collision handling)

different combinations of key presses and made sure that the animations/quaternions matched (jump-left, jump-right, etc)

compensate for animations by using quaternions, so sometimes it looks like the characters are glitching

making sure the platforms moved up based on characters on top of buttons

characters dying in the lava pit

made sure the doors worked

tried giving friction to the characters and ground, but characters were bobbing/glitching when they walked (add free body diagram)

4 Discussion and Conclusion

4.1 Ethical Concerns

300 words!

what models we decide to pick (ethnicity?)

violence of the fire pits

gender norms???

basing the models off ourselves (safety issue?)

4.2 Follow-Up Work

many more features and levels

level management system

character models

character collisions

currently implemented as a 2D game, just with 3D graphics, we could make it a fully 3D game by showing POVs of the characters

writing scripts (jeff)

4.3 What We Learned!

a lot of manual coding

physics is hard

importing custom meshes and animations, and sometimes they are not compatible

References

[1] *Fireboy and Watergirl*.

[2] `Mixamo`.