Jeff-Boy and Pearl-Girl

Jeffrey Cheng, Chau Truong

December 13, 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` [1]. 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 [2], 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` [3] 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 higher level.

The walking and jumping animations for the characters were sourced from Mixamo [4].

2.3 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.

Deciding the friction coefficients of materials was an important process that significantly affected the movement mechanics of our characters. We initially decided that all materials were going to have friction, but this had unintended consequences on our character models. Since our characters were physically represented by a bounding box, moving horizontally would enact a friction force on the bottom face of the bounding box, applying a torque to the character and having it rotate and bounce around. If our character was rotated and a jumping force was applied, the speed of rotation became unnaturally fast and an unintended behavior. As a result, we removed the coefficient of friction from the character box and the ground and manually stopped the character's velocity upon key release instead of through friction. The stepping boxes, however, do have friction, so they do not move horizontally infinitely when pushed.

2.4 Game Features

2.4.1 Ice Creams

Our game features collectible ice creams, whose 3D graphic comes from Vectary [5]. Each ice cream is assigned a "region of collection", defined by two coordinates $\{x_{min},\ y_{min},\ z_{min}\}$ and $\{x_{max},\ y_{max},\ z_{max}\}$ 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 a position 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.

2.4.2 Stepping Boxes

A stepping box is a `Box` from `cannon.js` that the characters can push around and jump on top of to reach higher levels. 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.

2.4.3 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, whose 3D graphic also comes from

Vectary[5]. If one character stands inside of the rocket ship, 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 2.4.1. 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.

2.4.4 Lava Pit

The lava pit is a `Three.js` box loaded with a lava texture using a simple lava image file. If a player enters its region of collection, the user is alerted of their death and forced to restart the level.

2.4.5 Doors

In order to beat the level, the players must enter the region of collection of their respective door. Upon successfully doing so, the user is alerted of their win and the level is restarted.

3 Results

Our main tests of correctness were visual confirmations that our characters and objects obeyed conventional laws of physics, except for specific mechanics we designed. For example, characters are able to move horizontally during free fall, although this is not possible in the real world because an object cannot move without a net force acting upon it. In one of our experiments, we listed every possible movement combination and direction, and simulated it with keypresses to ensure that the model was moving and oriented appropriately. For example, we have a walk-left, jump-left, jump-walk-left, jump-straight, and the list continues. For the most part, our models obey normal physics and collision detection.

4 Discussion and Conclusion

4.1 Ethical Concerns

For the most part, the graphics of our game are not terribly controversial. However, there were a few things we did have to pay attention to in order to minimize the ethical concerns of our game. With a male and female character in the game, we were sure to implement equal abilities for both the characters, as well as equal roles within each level. Since both characters must make it to the end, they must both complete similar movement sequences and help each other to completion. Our title, however, does fail to include non-binary genders

for our characters. Thus, upon expansion of our game, we would like to introduce a third non-binary character, or the ability to switch models to a non-binary characters.

Continuing with the ability to switch models, we would hope to add in many more selectable character models as future work. We would also like to include different races to choose from to promote inclusivity, while avoiding stereotypes for such character models. As a final stretch goal, we had talked about imported yourself into the game by taking 360° photos of yourself, converting the scan into a texture, and loading onto models. However, we would have to write some strict privacy rules if we did so. People's personal scans should not be able to spread into other people's games in order to protect user privacy. Furthermore, the user who uploaded the scan should be solely in control of when that model is usable or not.

Another concern that our game may bring is the violence of lava pits, in which characters can die in magma. In the original game, the characters are 2D, cartoon characters that also perish in fire or water pools. However, since our game features realistic 3D models, our characters falling into lava pits is more graphic and could require some age restriction.

4.2 Follow-Up Work

For immediate follow-up work, we would definitely like to expand the number of features and levels in our game. More components that require cooperation make the game more complex and enjoyable. This would lead to the development of many more windows in order to select a level, navigate between levels, and display level completions or failures. By extension, to make level creation more fluid, we would want to write a script that features a GUI, in which users could place game elements in a 2D canvas by clicking and dragging. Then, upon clicking a button, the script would output the coordinates and dimensions of all the game elements that were specified in the GUI. This would speed up level creation immensely.

As mentioned before, we would also like to diversify the available character models that users can play with. Ultimately, this exists with the ability to import custom models and even models of yourself. We would aim to find technology that could convert 360° photos of oneself to a texture file that is compatible with `Three.js`'s texture loader.

There are also some minor visual effects we would like to improve upon. The models take on a different orientation when their movement animations are played, and correcting these orientation changes introduces occasional visual bugs. For example, if the orientation correction occurs slightly before the animation starts playing, the character with flash to an unnatural orientation for a few frames. Additionally, in the original *Fireboy and Watergirl* game, the characters can pass in front of and behind each other, without having to jump over each other. This would involve disabling the interactions between the two characters' physical bounding boxes, but preserving them for collision detection for the rest of the world.

One last feature we dream of implementing would be a full extension of the game into 3D. Currently, we feature 3D graphics, but a 2D playing plane because characters can only move horizontally and vertically. We could easily expand the world to a 3D space, and then we could model the characters' point of view in side by side windows. This would make our game the ultimate 3D game of cooperation.

4.3 What We Learned!

Throughout this project, we learned a lot about the commonly-used tools for 3D graphics and physical simulation so developers do not have to start from scratch. For graphics, `Three.js` allows for visual geometry creation and the importing of custom meshes and animations through GLTF and FBX files, so we can mix and match already-existing objects and animations. However, we did also learn that these custom meshes and animations are not always compatible, because the FBX animation files may animate bones that the GLTF mesh simply does not contain.

For physics, `cannon.js` allows for the existence of physical bodies and handles forces and collisions for you automatically. Despite these helpful libraries, we did learn that world creation still takes a lot of manual work, creating objects with the correct position, orientation, and size. Furthermore, even though these libraries do a lot of the physics calculations and simulations for you, implementing physical movement is still a non-trivial task. Deciding where to put friction, how characters' physical bodies are modeled, and many other decisions will make or break your game. We had to go through several implementations before settling, as mentioned in Section 2.3.

Overall, we learned a great deal about real world graphics development, since neither of us had written a project like this before. We had a great time and are happy with the experience.

References

[1] Three.js.

[2] *Fireboy and Watergirl*.

[3] cannon.js.

[4] Mixamo.

[5] Vectary.