Super Mario Bros

Daniel Goncalves
daniel.goncalves001@umb.edu
University of Massachusetts Boston

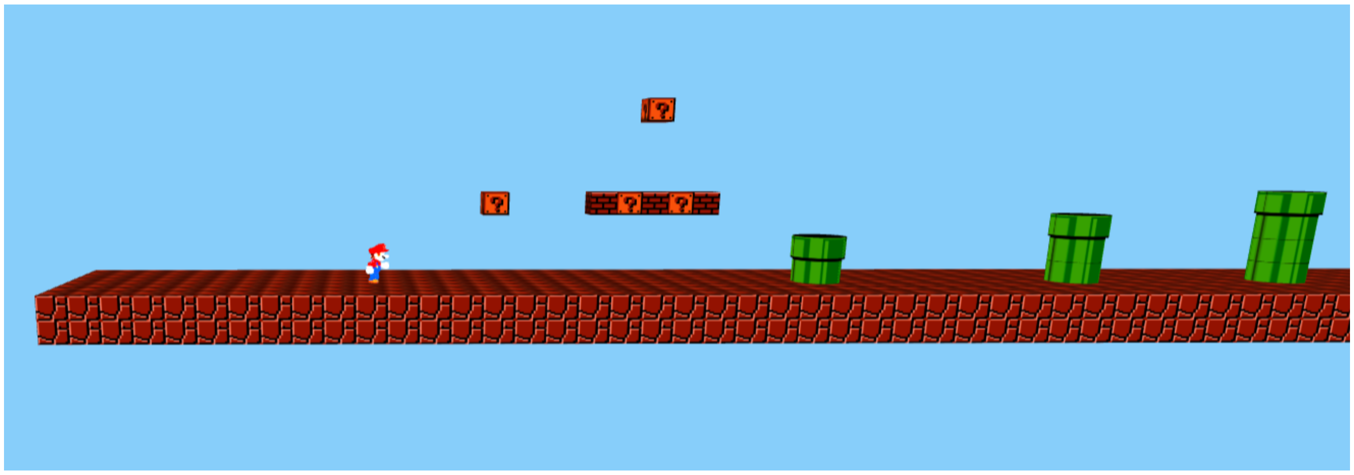

Figure 1: Screenshot of the game at the beginning

ABSTRACT

For the final project, I decided to make the Super Mario Bros game. I loved playing it as a kid growing up and I thought I would have a lot of fun making the game. I used Three.js, which is a WebGL framework, and Tween.js to help with the jumping movement. I used 3D models of Mario and world downloaded from sketchfab.

KEYWORDS

WebGL, Three.js, Tween.js, Game

ACM Reference Format:
Daniel Goncalves. 2021. Super Mario Bros. In *CS460: Computer Graphics at UMass Boston, Fall 2021.* Boston, MA, USA, 2 pages. https://CS460.org

1 INTRODUCTION

In this project I used what I've learned throughout the semester. I aimed to recreate the infamous Super Mario Bros game but in 3D. This project was harder than I originally anticipated. I had to overcome a lot of obstacles and I'm fairly happy with the progress I made.

2 RELATED WORK

Tween.js [2] and Three.js [1].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
CS460, Fall 2021, Boston, MA
© 2021 Copyright held by the owner/author(s).
ACM ISBN 1337.
https://CS460.org

3 METHOD

I used GLTFLoader and FBXLoader to load the 3D models into the scene. I save the reference to Mario model to change it position whenever user clicks on WASD/Space Bar keys. After all the models have successfully been imported and loaded, then I had to setup some like of movement for the character. In the game W is jump, A is move back, D is move forward, and Space Bar is also to jump. Moving forward and back was easy to implement as all I had to do is modify the x position of the character. But to jump I had to use a animation/tweening library called Tween.js[2]. Their documentation was really helpful. The Mario model I download also came running and jumping animations. This saved me the hassle of coding the animations using quaternions/slerping. I saved all animations and switched the current animation based on the movement of the character using AnimationMixer and AnimationAction from Three.js [1]. It still not perfect but gets the job done. The hardest part of this project was collision detect, eventually I decided to use the bounding boxes of the THREE.Mesh to detect a collision. If a collision was detected then it restricts the movement of the character. When you fall, a game over text will appear on the screen. I have spend a lot of time on this project and have learned a lot as well.

3.1 Implementation

This is the snippet of code where I use FBXLoader to load in the Mario model and save the running and jumping animations so I can switch between them later on.

```
async getMarioModel() {
  const loader = new FBXLoader();
  loader.setPath("./resources/mario/");
```

```
    const anim = new FBXLoader();
    anim.setPath("./resources/mario/animations/");
    var [idle, run, jump] = await Promise.all([
      loader.loadAsync("Idle.fbx"),
      anim.loadAsync("Run.fbx"),
      anim.loadAsync("Jump.fbx"),
    ]);
    this.animations.push(
      idle.animations[0],
      run.animations[0],
      jump.animations[0]
    );
   this.currentMixer = new THREE.AnimationMixer(idle);
    this.animationAction.play();

    return idle;
  }
```

3.2 Milestones

3.2.1 Milestone 1. Find downloaded models on sketchfab of Mario and the world.

3.2.2 Milestone 2. Using Three.js [1], setup the scene, camera, lights, and load in the models.

3.2.3 Milestone 3. Add controls for Mario,such as moving forward, backward, and jumping

3.2.4 Milestone 4. Add invisible rectangular mesh for each pipe, and death fall areas of the world.

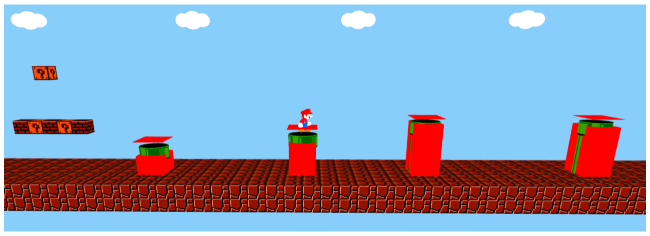

Figure 2: The "invisible" meshes I added on each pipe. These would be invisible in-game.

3.2.5 Milestone 5. Add collision detection to the game using the invisible meshes, and restrict movement/ end game based on the collisions.

3.3 Challenges

- Challenge 1: I spent a day trying to debug an error I was getting whenever I deployed on GitHub Pages. I was getting an 404 error because it could locate the .fbx files for Mario model. But with the help of Professor Haehn, the file sizes was too large so I removed unessary files and it worked.
- Challenge 2: For collision detection, I wanted to use a physic engine because it would save me a lot of time coding it myself. The engines I had in mind was Cannon.js, Ammo.js, and Physi.js. Unfortunately with Ammo.js and Physi.js I couldn't get the library to import. With Cannon.js, the Meshes were not updating on the scene at every frame.
- Challenge 3:I used a 3D model for the world, which came with object such as pipes, bricks, coins, etc. The problem was that it did not give reference to all the objects, only each different object. For example, there is a coin object, and have reference to it but that reference points to all the coins and not each individual coin.

4 RESULTS

Although I did not complete everything I hoped for, I made a lot of progress the past week. The final product is somewhat of a Super Mario Game. There are still some part of the world where you can go through objects.

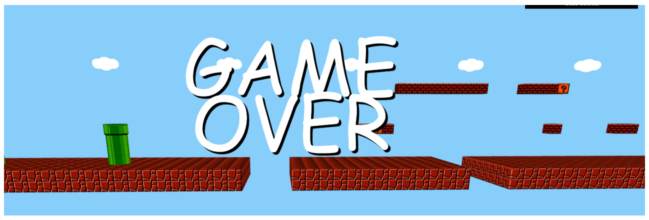

Figure 3: An example image.

5 CONCLUSIONS

This project was harder to make than I thought. I spend a lot of time researching and trying to integrated a physics engine onto the game but ultimately had difficulties importing it. I still had a lot of fun and learned a lot. I plan to continue this project even after the course, maybe look more into cannon.js because I've had the most success with that.

REFERENCES

[1] Ricardo Cabello et al. 2010. Three.js. *URL: https://github. com/mrdoob/three.js* (2010).
[2] Grant Skinner et al. 2010. Tween.js. *URL: https://github.com/CreateJS/TweenJS* (2010).