Journey to the West

Yiwei Yao
yiwei.yao001@umb.edu
University of Massachusetts Boston

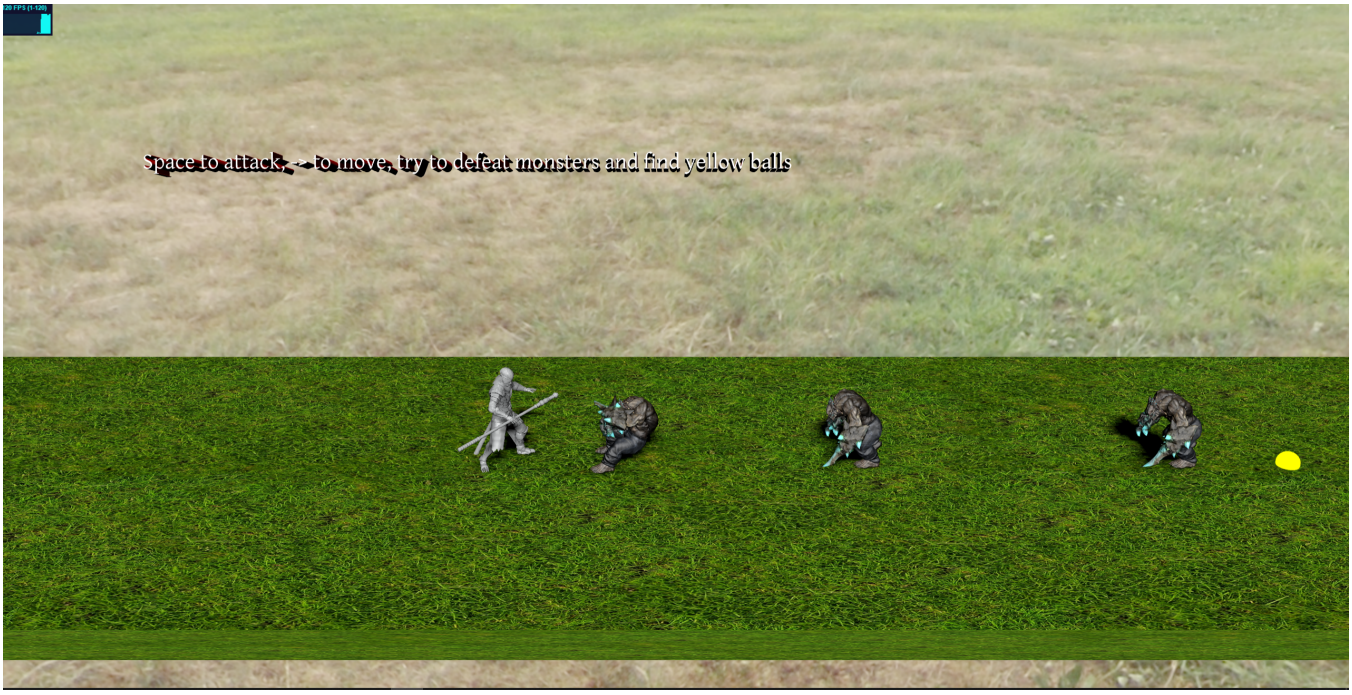

Figure 1: Defeating the monsters in the game.

ABSTRACT

This project is a horizontal board game based on Based on a story of Journey to the West that is a Chinese mythology fiction. The project uses ThreeJs, and it has three implemented stages which are opening stage, game stage, and ending stage.

KEYWORDS

WebGL, Visualization, ThreeJs, Mixamo, Sketchfab

ACM Reference Format:
Yiwei Yao. 2020. Journey to the West. In *CS460: Computer Graphics at UMass Boston, Fall 2020*. Boston, MA, USA, 3 pages. https://CS460.org

1 INTRODUCTION

The project tells a short Chinese mythology story in a way of game that a monkey finds foods for his master by defeating monsters.

CS460, Fall 2020, Boston, MA

ACM ISBN 1337.
https://CS460.org

Through WebGL, players can access a nice image quality game easily, it can help to grow the interests of discovering different culture.

2 RELATED WORK

Three.js [2]
Wukong 3D model from Sketchfab [3]
Mixamo [1]

3 METHOD

The project contains three scenes:

- Loading Scene.
- Main Scene.
- Failure Scene.

Loading Scene renders as soon as players access the web page, it shows a simple line of text that tells players that models and asserts are being loaded. Once everything is loaded up, renderer will jump to Main Scene.
Failure Scene that contains a simple line of text that tells players that game is over, it will be rendered when the character that is played by players is failed in getting foods in the game.
The Main Scene contains five main parts:

- Background & Lights & Camera
- Text
- Plane & Characters
- Animations
- Collision check & Attack check & Remove useless models

Background is a sky box that provides the 3D background for the whole scene. There is only one directional light in the project that positioned at $(1, 68, 90)$. Camera is set at position $(0, 800, 2000)$ that makes players to view the whole scene from the side.
Text is the key to introduce the story and guide players how to play, it will change base on the position of the character that players are played.
Characters are positioned on the plane in the scene. Plane is used to hold the characters and casts shadow of the characters.
Animations reflects the interactions between the players and the scenes, for example, when players press down → button, the character that players are played will run, when players press down Space button, the character that players are played will attack.
Collision check will switch the scene to failure scene when objects collision. it runs each rendering.
When players press down Space button, attack check feature will check if other models are in attack range. If other models are in attack range, then set other models to dead, and switch model's animation to dead animation.
Remove dead objects feature will remove every object that is dead. it runs each rendering.

3.1 Implementation

There are two main kinds of implementation in the code:

- import models and their animations by using fbx loader
- render the animations base on position, time, and flags.

The models and their animations are imported into the scene by loading a T-Pose object model and loading its corresponding animations into this object.

```
......
const objectloader = new FBXLoader();
const animloader = new FBXLoader();
......
// load object
objectloader.load('Wukong.fbx', (fbx) => {
    ......
    this.body = fbx;
    ......
    // load animation
    animloader.load('run_forward.fbx', (anim) => {
    const m = new THREE.AnimationMixer(fbx);
    ......
    this.run = m.clipAction(anim.animations[0]);
    ......
    });
    ......
});
```

In the above code, 'Wukong.fbx' is a T-Pose fbx object which means it has no animation, after load the T-Pose object, it uses another fbx loader to load the animation file 'run_forward.fbx', and mix the loaded animation with the T-Pose object.
All loaded animations will be stored into a mixers array for rendering purpose.

```
......
animloader.load('run_forward.fbx', (anim) => {
    ......
    mixers.push(m);
    ......
});
......
```

The stored animations are being rendered by updating each frame in animate() function.

```
......
function animate() {
    requestAnimationFrame(animate);
    const delta = clock.getDelta();
    if (mixers) {
        ......
        mixers.map(m => m.update(delta));
    }
}
```

However, if a object has multiply animations associated with, we want to choose a particular animation to run. we have to fade out all other animations and fade in the particular animation.

```
......
Wukong.prototype.move = function() {
    ......
    if(this.attack.isRunning()){
        this.run.fadeOut();
        this.attack.fadeOut();
        this.run.fadeIn();
    }
};
......
```

the above code shows that we only want to run idle only.
In the case of switch other animations to running, we can call idle_state() in animate() function base on position, time and flag.

```
......
function animate() {
......
if (mixers) {
    ......
    else if (wukong.body &&
    wukong.body.position.x >= 150 && flag == 7) {
        wukong.move();
    }
    ......
    mixers.map(m => m.update(delta));
}
......
```

The above code only runs wukong's run animation when wukong's x position is bigger than 150, and flag is 7.

3.2 Milestones

3.2.1 Milestone 1. Created the background, lights, camera. Imported models and their animations.

3.2.2 Milestone 2. Implemented animations that reflects the interaction between players and Main scene.

3.2.3 Milestone 3. Implemented Loading scene, and failure Scene.

3.3 Challenges

- Challenge 1: In the animations switch function, it can not only depend on fadeOut and fadeIn because fadeIn a animation does not play the animation from the most first frame, it may play the animation from the middle frame. In order to solve this problem, I stop the animation then restart the animation that will play the animation from the most first frame.
- Challenge 2: Lagging when move camera x-axis position beyond a particular point, it may be caused by a bug in THreeJs.
- Challenge 3: It is hard to create a 3D models and its animation by myself, therefore, I find a free wukong model on Sketchfab and make animations by using Mixamo.

4 RESULTS

The result of my project is a 3D horizontal board game, it looks like image 2 at start, players can use → key to move and collecting yellow balls image 3, and Space key to defeat the monsters like image 4, and during the game it has some automatic cut-scenes base on rendering time and character position like the ending cut-scenes image 5
Github page: https://yylhyyw.github.io/cs460student/final/

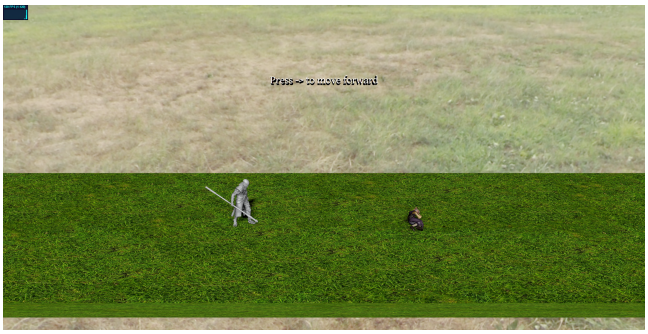

Figure 2: At start

5 CONCLUSIONS

In this project, I have learned a lot in gaming development. In gaming development, developers use hundreds or thousands of 3D models and their corresponding animations, and match the animations frame by frame in the game. I also learned the power of OOP programming, it allows developers to reuse the code when creating a new object with different functionality.
At last, it teaches me that developing a game is a huge task, it have to deal with running engine, 3D object models, and game-play. Even a little game that I do not have to care about rendering engine and object models, I have to revise my code, logic, and algorithm for many times.

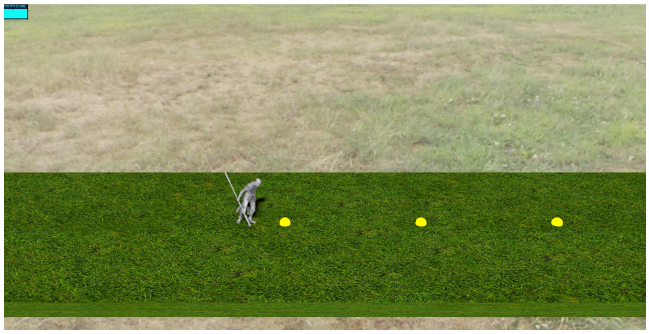

Figure 3: collecting

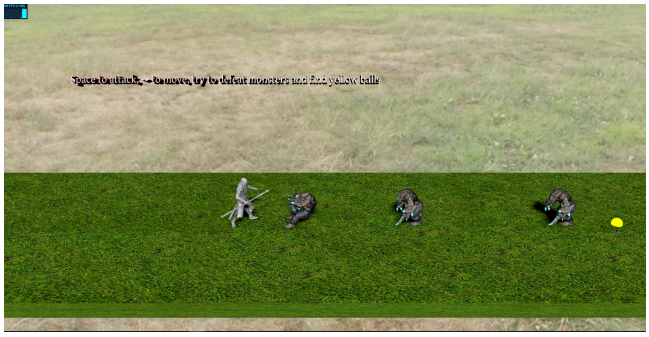

Figure 4: Defeating the monsters.

Figure 5: victory

REFERENCES

[1] Adobe. 2020. Mixamo. *URL:https://www.mixamo.com//* (2020).
[2] Ricardo Cabello et al. 2010. Three.js. *URL: https://github. com/mrdoob/three.js* (2010).
[3] Mask. 2020. Wukong 3D model. *URL: https://sketchfab.com/3d-models/wukong-49078b5b61b845bb9de6313ede07dba1* (2020).