--COS 426 Final Project May 2021--

Tome Runner

Bates Brodie, Beatrice Caruntu

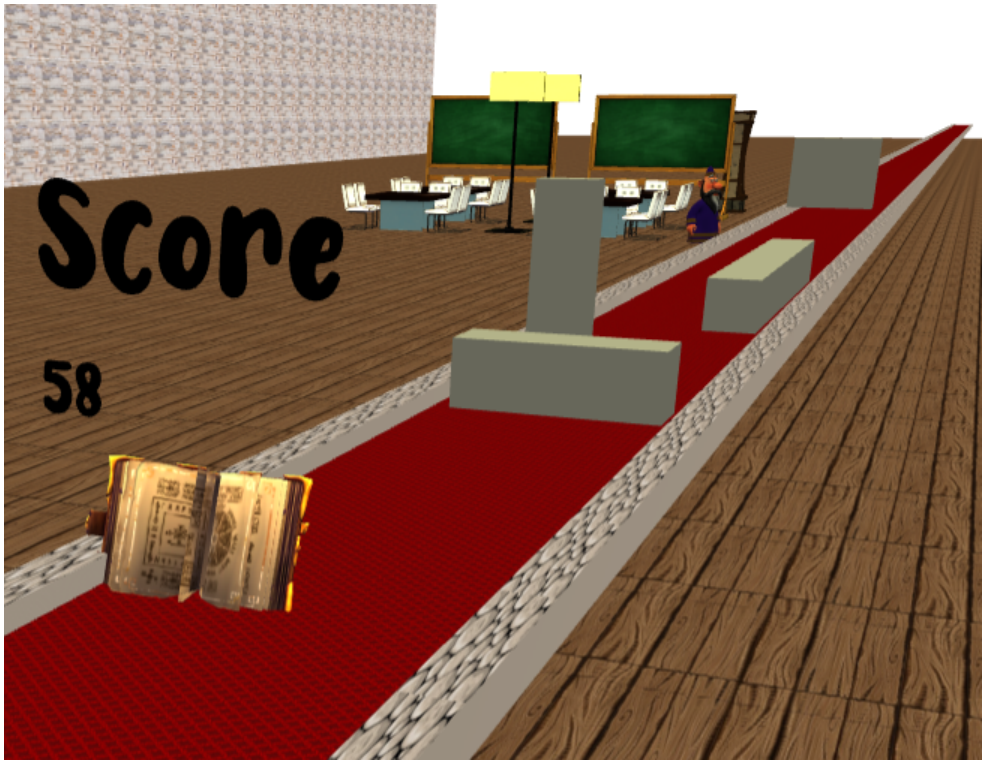
Abstract

This project entails the creation of a Javascript-based endless runner game. Inspired by the well known work of Temple Run [1], we implemented a similar game called Tome Runner. Using the arrow keys, the player controls the movements of a flying book in a magical library that is about to get burned. While trying to escape from the evil fire, our valuable book is also trying to earn rewards and avoid various obstacles. The goal of the game: get as many rewards as possible without getting burned by the fire.

1. Introduction

Tome runner is an infinite-runner with obstacles and rewards game. The game setup is quite simple: a book if flying in the air and players have to control the book to switch among the left, middle and right pathways, as well as fly upwards or downwards, to avoid our generated obstacles and collect as many rewards as possible. Some obstacles have movements of their own, making them more difficult to avoid. Our book's flying speed is increasing as the game goes on, with the maximum final speed being twice the initial one. Player's scores are computed based on the elapsed time and the number of rewards collected so far and is shown and updated in the left top corner of the screen.

To make the game interface more entertaining, we also generate background items such as library shelves, study desks, chairs, lamps, boards. Some of these items have movements of their own (small random noise movements) to create a more interesting scenery.



2. Methodology

Book:

We added a book into our scene with the help of a glb file (Figure 1) obtained from Sketchfab[3], which was then converted to gltf using the GLTFLoader() function from THREE.js. The glb files was already having a “flying book” animation included and we used AnimationMixer to loop through that animation clip. We used the difference in timestamps in the update function to calculate the duration of each frame.

As a backup for the book's default animation, we added random movements on the y-axis, as if it was flying.

We then decided to bound the movement of the book to keys. We added keydown and keyup event listeners to register the key presses. The key-movement pairs used are:

1. RightArrow: move the book to the right
   If in bounds, move the book's position and tilt the book right.
2. LeftArrow: move the book to the left
   If in bounds, move the book’s position and tilt the book left.
3. UpArrow: fly higher
   If not already jumping, give the book an initial upwards velocity which gradually decreased to simulate gravity.
4. DownArrow: fly lower

If in bounds, move the book's position to a lower y-value, shortly wait, then move the book's position back to the baseline.

Figure 1: Book 3D object

The book itself is not moving on the z-axis. Every object in the scene, including the floor and walls, moves with respect to the book. All objects except the book are moving towards the camera on the z-axis with a specified speed. The speed is increasing its value as the time passes (the objects are accelerating in order to make the game progressively harder and more entertaining).

Collisions:

Collisions can occur between the book and any other object on the trail, or between any object on the trail (including the book) and the falling weights objects.

At each frame update, we for those collisions. To model them, we assigned each object a bounding box. At each frame update, we check for intersections among those bounding boxes with Three.jsBox3. If two bounding boxes intersect, their respective objects have collided. If a collision occurs, we call the collision function for that object's class.

If there is a collision between the book and any other object, the collision function updates the remaining lives (remaining allowed collisions). If there are still lives remaining for the player, their number decreases by 1 and the book continues flying. Otherwise, the game ends, and the player can see his end score.

If there is a collision between a falling weight object and any other object in the scene (except the book), the weight object rebounces from the trail.

Scenery:

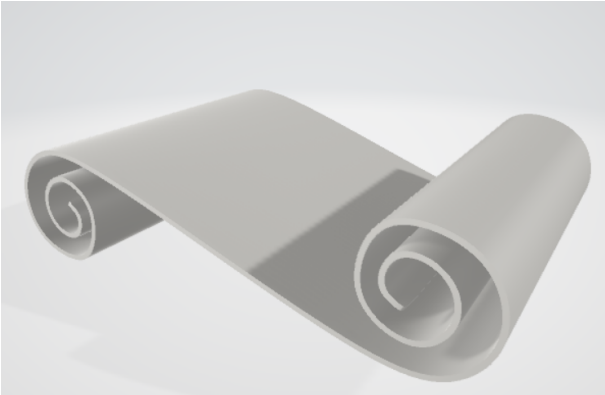
The scene consists of a path with stone borders in the middle of a library. For the path, floor, and walls we used the TextureLoader() function from THREE.js to load jpg texture pictures and use them for a better general aesthetic. Also, by using textures, the effect of the book flying on the z-axis is more prominent, as not only the obstacles move towards it, but also the entire scene.

Furthermore, we added ash particles falling in the background to support the idea of a burning library. (The ash is similar to snow particle implementations in THREE.js)

Background Items:

The library is populated with different objects moving towards the camera on the z-axis with a specified speed as the game is running. Once an object becomes invisible to the camera it moves back into the scene at the furthest away position as possible (recycling its position).

All these objects have been imported from glb files and uploaded with the use of the GLTFLoader() function from THREE.js. They were then resized, rotated and placed on specific positions in the scene. Some of these items include: bookshelves, chairs, desks, lamps, stairs, documents, boards..



Some of those objects have a small randomized noise movement in order to make the scene more alive.

Obstacles:

The pathway is populated with various obstacles that move towards the camera on the z axis at an increasing speed as the game progresses. Eventually the speed reaches a maximum at roughly two times the initial velocity. Once the obstacle passes the camera it resets its position in both the z and x directions, moving it in front of the player and randomly assigning a lane for it to occupy. There are many different types of obstacles, each with unique bounding boxes and positionings which the player must avoid.

Rewards:

The pathway is populated by various pickups that move towards the camera using the same methodology as the obstacles. They similarly increase their movement speed towards the camera over time. In addition to resetting their position if they pass the camera, these pickups also reset their position upon collision with the player, in addition to yielding other effects. Currently the only type of pickup is the score pickup, which directly increases the player's score by 10 each time the player collides with it.

Fire:

The fire object that chases our book is a collection of random sized 3D models positioned below the camera. They are all moving with random small steps to make the fire appear live.

Score panel:

The score panel is made up of two objects located on the top left of the default camera positioning. As the game progresses the score counter increases the longer the player stays alive. Furthermore, picking up score pickups immediately raises this counter by 10.

3. Results

Playability:

In terms of gameplay, we measured success via how natural it feels to move the character around, both in effectiveness and methodology. For instance, rather than instantly teleport the player when an arrow key is pressed, the book moves (relatively) more slowly left or

right. Furthermore, we implemented our movement where the player must actively hold the arrow key to continue moving in a cardinal direction, which we feel more effectively engages the player in the game. Ultimately, we feel that our controls emphasize a very reactive playstyle that actively keeps the player focused on the game.

Graphics:

For graphics and design, we decided to collect feedback from our friends. After playing around with several 3D models and designs for each object (books, desks, textures for floors and walls), we finally arrived at a scene that all of us agreed on. Overall, the objects fit well into the scene and add a live effect to it.

4. Conclusion

Overall, we believe that our approach worked out well and that we managed to create a fun and entertaining game. We implemented all of our target goals and a lot of our stretch goals, including some that were added along the way.

We learned a lot about the THREE.js library while working on this project, as well as game development.

Given more time, we would like to add more features to this code. The path could take random turns instead of being a straight line. We could add more interactions with the existing obstacles or rewards, such as a weak magnetic force that attracts the rewards.

5. Contributions

Bates:

- Obstacle Functionality
- Collision Detection
- Player Movement
- Score Counter
- Score Pickup Functionality
- Starting / Playing Screen Split

Beatrice:

- Graphics design: glb files and textures
- Library Items placement and movement
- Base scenery: floor/walls design
- Glb Animation extraction
- Fire movements
- Ash/Smoke particles implementation

6. References

1. Temple Run: https://en.wikipedia.org/wiki/Temple_Run_(series)
2. No Virus Font: https://www.dafont.com/no-virus.font
3. Sketchfab (source for 3D model glb files): https://sketchfab.com/feed
4. Microsoft 3D Viewer (source for 3D model glb files: https://www.microsoft.com/en-us/p/3d-viewer/9nblggh42ths