3-Rex Final Project Report
Anthony Ng, Kai Obinata, Kevin Phan, Young Ha Yoo
December 16, 2022

Abstract

Our team developed 3-Rex, a 3D version of a famous Google Chrome browser game *Dino Run*. The original game scrolled the landscape and obstacles towards the player to which the player would react by jumping. Our 3D version of this game added a significant change to this single-key dynamic by allowing side-to-side motion. In this report, we include a detailed brief of all the interconnected components that made up our finished game – including mesh resources, collision detection, user interactions, performance, audio, multiple views, visual decorations, and HUD. With the help of some peer review testing, we've concluded that our game was fairly successful in accomplishing what it set out to do: to engage the player. Although we have our own ideas as to what more we could have polished, the flair we were able to add was compelling for immersion and repeated playability.

Introduction

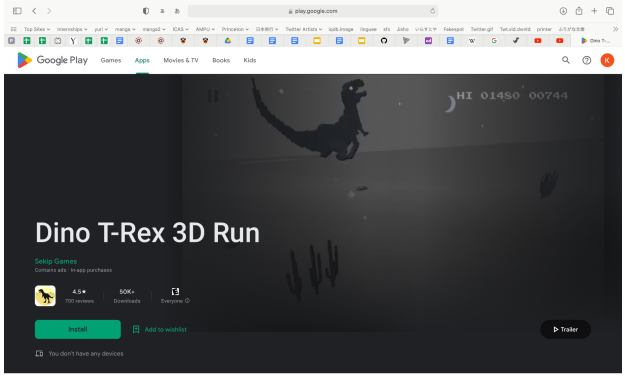
In 2014 a team on Google Chrome published a game that would automatically run on the browser if a user has no internet connection. This game, which came to be known as the *T-Rex Game* or *Dino Run*, is one in which a player controlling a running dinosaur carefully times jumps in order to avoid incoming obstacles, such as cacti and birds, as they are procedurally generated across a 2D playing field. Our game 3-Rex is a 3D version of this Dino Game.

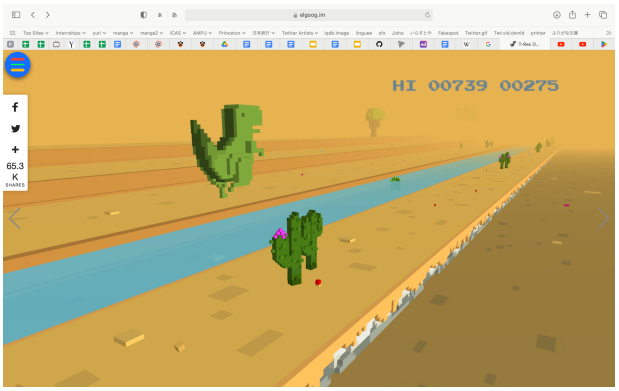
Goal

Our goal was to create a 3D version of the Google Chrome Dino Game, with additional flair that gives greater engagement and immersion for potential players.

Previous Work

There are other related works that other people have done in regards to a 3D version of the Dino Game. Attached below are examples of these previous versions. However, compared to these, our implementation is unique in that we offer sideways motion which we believe makes the fullest use of 3D space. The extra degree of freedom of tilting either left or right to avoid obstacles in a sideways motion is bound to increase immersion akin to the feeling of running in an open world.





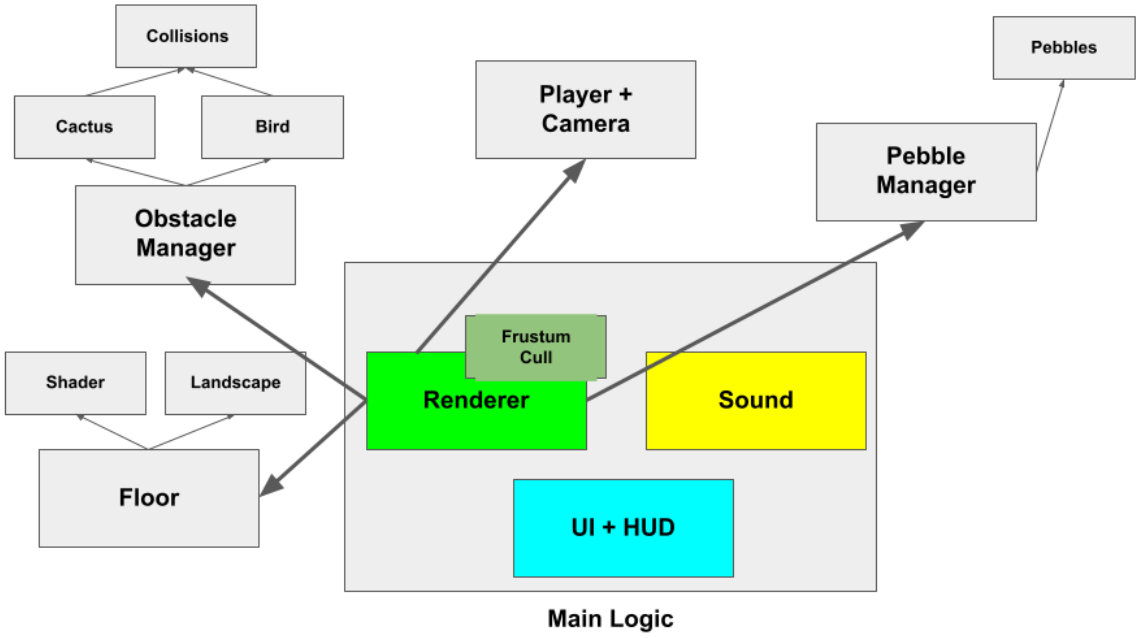
https://www.youtube.com/watch?v=6QGAfU-ChpE, https://www.youtube.com/watch?app=desktop&v=ThjURZITKT0, https://play.google.com/store/apps/details?id=com.sekip.dinoTrex3Drun&hl=en_US&gl=US, https://elgoog.im/t-rex/3d/

Approach

Our general approach towards the mechanics of the game is to prioritize a simple, minimalistic and yet visually appealing and smooth running game. For instance, while more high definition meshes for each of the given objects – cacti, bird, and the runner – were available, we chose to maintain a simpler aesthetic in line with the original 2D Dino Run game. And in addition, we would implement frustum culling to ensure that the game would feel smooth in playtime.

Methodology

3-Rex is composed of several major, interconnected components. The below diagram succinctly summarizes the various components that made up our system. Critical components are delved into in deeper detail in ensuing paragraphs.



Player Character and Player Character Movement

Our player character consisted of an animated mesh that we retrieved from Mixamo. There were several different possible implementations, such as creating our own player mesh, modeling the player as a simple geometry (e.g. sphere, box), retrieving an animated character from Mixamo, and importing a static GLTF model. Although each option had its pros and cons that were mostly based on a tradeoff between simplicity and complexity, we decided to go with the animated character. The primary reason for this was that our game was fundamentally a runner-based game. Having a character whose legs were not animated seemed to harshly detract from the game experience. The major disadvantage of this approach was that although our game was called '3-Rex', we had to use a model that was not a dinosaur. The ideal way to have gotten an animated character that was a dinosaur would have been to create our own skinned mesh. However, due to the contracted timeline of this project, this option was simply not viable. Therefore, we strived for an ideal balance between simplicity and complexity.

Aside from player animation, we also enabled the player to be translated and rotated. For the translating and rotation, we simply used the Three JS position and rotation fields. We also added the ability for the player to jump by incorporating the Cannon-es physics engine.

Obstacles

For obstacles, there were three decisions that we ended up having to make in order to execute our approach. The first choice was regarding if we were going to move the obstacles towards the player character or if we were going to move the player and camera towards the obstacles.

We decided to go with the second option, because moving everything towards the player character would mean that we would have to render more changes, such as the obstacles, the floor, and the pebbles. Therefore, for a potential performance upside, we decided to keep the obstacles stationary. The second choice was regarding how many obstacles we were going to use. Again, for performance reasons, we decided to spawn only a few obstacles and to continuously reposition the obstacles rather than creating a new Three JS Object for each obstacle that the player encountered. We created an Obstacle Manager class that abstracted away the complexities of how we would clean up obstacles and then reposition them as if the obstacles had been generated. The third choice was similar to the player character. For the obstacles, we had the choice of using external GLTF files or simply using simple geometric meshes. The pro of the GLTF files was more realistic graphic quality, but a con was that there was more pressure on the renderer. The pro of the simple geometric meshes was renderer performance but a huge con was that the objects would not reflect any real game idea. As a result, for this, we ended up retrieving GLTF files for the cacti and the birds.

Collisions

Collisions were an area where there were multiple options. We could have gone with collisions of any types of geometries (spheres, boxes, etc), or we could even have modeled a particle system like we did for the simulator assignment (COS 426 assignment 5). Of course, modeling as a particle system would have created the most realistic collisions. Likewise, assigning each object a specific geometry and implementing collisions between all types of geometries would also have created very accurate collisions. However, for the purpose of this assignment, we observed three things. First, the basis of the game was not grounded in reality. As a result, a particle system did not seem to add much to our envisioned goal. Secondly, after experimenting with only bounding box collisions, we found that using only boxes was more than sufficient in the level of accuracy that was suitable for our game. Thirdly, we realized that if we decided to go with bounding boxes, we could use the Three JS object Box3. All three observations compelled us to take the bounding box route for handling collisions. The code block below shows the way in which the box intersection code was simple and, thus, inexpensive to call very frequently throughout our program.

```
export function boxesIntersect(box1, box2) {
    const min1 = box1.min;
    const max1 = box1.max;
    const min2 = box2.min;
    const max2 = box2.max;

    const xIntersects = intersectHelper(min1.x, max1.x, min2.x, max2.x);
    const yIntersects = intersectHelper(min1.y, max1.y, min2.y, max2.y);
    const zIntersects = intersectHelper(min1.z, max1.z, min2.z, max2.z);

    return xIntersects && yIntersects && zIntersects;
}

function intersectHelper(min1, max1, min2, max2) {
    if (min1 < min2) return max1 > min2;
    else {
        return max2 > min1;
    }
}
```

User Interaction

The user interaction piece was more intuitive to implement. We ultimately enabled the user to jump via the up arrow key and to rotate left and right with the left and right arrow keys respectively. Pressing the ‘v’ key enabled the user to toggle between two different camera views (one behind the player and one in front of the player). Pressing the ‘s’ key enabled the user to enter a special mode, where the floor was colored with the fragment shader in a grid-like pattern according to world coordinates.

Another piece to the user interaction was a HUD, which included a start menu, a game over menu, a pause button, and a score button.

Performance

Performance was also another aspect that we had to take into consideration while building our project. The largest portion of this performance feature was implementing frustum culling, so that objects not in the camera’s frustum were not rendered (shown in the code block below).

```
for (let i = 0; i < verts.length; i++) {
    const v = verts[i];
    const worldV = obj.localToWorld(v);
    const worldV4 = new Vector4(worldV.x, worldV.y, worldV.z, 1);
    const cameraV4 = worldV4.applyMatrix4(camera.matrixWorldInverse);

    const projV4 = cameraV4.applyMatrix4(matrix);
    const x = projV4.x;
    const y = projV4.y;
    const z = projV4.z;
    const w = projV4.w;

    if (x >= -w && x <= w && y >= -w && y <= w && z >= 0 && z <= w)
        return true;
}

return false;
```

The second portion of the performance was the realization that the game's speed differed greatly based on the fps capability of the computer running the game. Therefore, we included code to cap the framerate at 30 in the render loop.

```
// Render loop
const onAnimationFrameHandler = (timeStamp) => {
    if (timeStamp - prevTimeStamp > fps) {
        renderer.render(scene, currCam);
        scene.update && scene.update(timeStamp);
```

Visual, Decorations, Other Features

A major component of our project was making sure that the project looked good. This involved many pieces. **First**, we added pebbles to further enforce the idea of a desert background. Due to slight rendering lag when loading in pebble GLTF files, we instead opted for a few different types of polygonal geometries and then texture mapped the pebble texture on top of them. This enabled us to reach the ideal balance between visual impact and performance. **Second**, we used the JavaScript random function in order to change aspects of the floor landscape. This let us create a mountainous region in the background. **Third**, we added sound for the player movement. There were no drawbacks to adding sound, and we realized that sound further enhanced user engagement in the game. **Fourth**, we added a toggle feature that would apply a fragment and vertex shader to the floor to create a cool, grid-like pattern for the floor. **Fifth,** while we did not use Perlin noise, we subdivided the floor into triangles and used the JavaScript random function in order to create basic mountainous landscapes in the distance.

Results

At the conclusion of the creation of our game, we were able to have a good grasp on how well the final product was able to accomplish what we had originally envisioned. Since our goal was to create a 3D version of the Google Chrome Dino Game, where the player would have the extra degree of freedom of tilting either left or right to avoid obstacles in a sideways motion, we think that for the most part we were successful in creating the feel of a runner game like Dino Run, while also giving more of an immersive experience in the 3D nature of the game.

Besides allowing for a 3D extension of Dino Game, we also measured our success by how smoothly the game ran, how accurately collisions with the obstacles occurred, and how visually and aurally pleasing our game was. We gathered these results by doing a lot of play testing on our game, running it on different computers to make sure the frame-rate would be consistently stable across different devices, intentionally running into and grazing past obstacles to make sure the collision detection operated in a way that was intuitive and accurate, and asking people outside our group to play test our game and give their opinions on how well it looked and sounded.

Overall, from these experiments as well, we concluded that our game was fairly successful in accomplishing what it had set out to do. The game ran fairly smoothly on all the devices we observed, mostly because of the optimizations we achieved in the implementation of the rendering of meshes and calculation of collision detection. Although we optimized collision detection by opting to use simple boxes to represent the hit boxes of the obstacles, we observed that most of the time this made little to no difference in the experience of the game, and the implementation of collisions in this way differed little from the players' intuitive understanding of where the areas of collision lied, so in this regard we also view the results of our game as successful. Lastly, we received positive feedback from the visual and audio elements of our game, especially the sounds that we implemented to play throughout the game as the player ran, jumped, and collided with meshes, as well as the random generation we put into the terrain that allowed for a more interesting experience for the player.

One downside is that we were not able to implement a dinosaur mesh as our player character, which would have aided in providing a better experience in giving the same atmosphere as the Dino-Run game as we would have intended, but we think that overall this did not detract significantly from the playing experience or the immersion of the premise of the game as a runner-type game where you are meant move forward and avoid obstacles, so we still view the final state of the game as largely successful.

Discussion

Throughout the project, we made sure to divy up the roles among our group members so that each person had an important role in the project and was able to focus on a concrete

component of the game. This turned out to be very effective since it allowed each of us to develop familiarity with a specific aspect of the game and collaborate seamlessly with each other when we needed those aspects to interact.

As for our general approach towards the actual mechanics of the game itself, we felt that our approach that prioritized a simplistic, minimalistic but visually appealing and smooth running game was a good compromise, especially since we did not have a considerable amount of time to work on the project.

If we had more time, we could have taken a more ambitious approach in making more complicated textures, allowing for more complex procedural generation for the floor and background for more unique and interesting visuals, and possibly adding different meshes to make the game feel more fleshed out. This more ambitious approach would have been better in that it probably would have allowed us to create a better feeling and more immersive experience, which could definitely be something that could be pursued further in the future.

Through our experience in developing this game, we were able to learn a lot about the creation of a web-based game, especially in regards to implementing libraries to load in and texture meshes, as well as implementing simulated dynamics through a physics engine library, movement through event handlers, and collision handling. More specifically, for this game, we gained experience in working with the THREE-JS and cannon-es libraries, as well as creating HUDs and implementing sounds through HTML. Finally, we gained a lot of insight into rendering objects, how to optimize that rendering procedure so we wouldn't have to continuously render objects in situations where they could be reused, as well as frustum culling to avoid situations in which rendering would be unnecessarily entirely.

Conclusion

As for what was achieved, we completed all of the goals that we deemed required. For one, we completed all of the primary tasks that we set out for ourselves while maintaining good pace through close coordination with each other. We were in a state to add embellishments that we did not originally intend or plan to do, such as the rudimentary terrain generation. While we did not complete our stretch goals of a level editor or implementing our own linear skin blending, we were still able to include animations that made the game more visually appealing.

To further improve the game, many elements could be implemented or added to increase the polish of the game. For example, the next steps would probably involve a visual overhaul of the game's aesthetics. Due to the limited time and knowledge, we were unable to fully stylize our game to the fullest. Finding/creating more attractive meshes may be a nice next step to improve the game, such as actually finding a dinosaur for the player mesh. In addition, finding suitable post-processing effects may help with the overall visual appeal of the game. The HUD was a fairly simple HTML overlay that could either be stylized further with additional CSS properties or replaced with a whole new overlay method altogether (eg. using THREE components to create it).

Certain gameplay details could also be finetuned. We had a lot of discussions about how to handle varying device capabilities, as the FPS of our computers varied wildly. This led to differing philosophies about what the player's initial speed should be, and the rate at which it increased as the game progressed. Perhaps allowing the user to customize these settings could be a good compromise, but also keeping in mind differing device specifications would allow for a better gaming experience. We could also implement other user actions (sliding, powerups, etc).

One potential detail that may need to be reviewed is how we capped the framerate. We capped it at 60 fps using timestamps, but there may be a more elegant way or a better control method than capping the framerate. We did this to help standardize the gameplay experience, but perhaps some other solution would work as well.

Contributions

Kevin Phan

We divided our work up into tasks that we saw were important at the time, so I ended up touching quite a few parts of the codebase. I implemented the HUD, which we used to manage the state of the game and handled transitions between those states (start menu, gameplay, pausing, game over). I also handled the sounds of the game, finding the sound effects and integrating them with the state transitions I mentioned above. I experimented with the landscape generation and textures to dictate the general appearance of the game, and also added the animations that are seen in the game. In addition, I worked out the collision detection logic, ensuring that the meshes had their bounding boxes set correctly and using simple box intersection logic.

Young Ha Yoo

Throughout the project, I took on the role of merging our branches. Towards the beginning, I worked on the obstacle manager logic (generation, garbage collection) while modeling the obstacles as box meshes and also worked on player rotation. After the MVP was complete, I implemented frustum culling and added a simple grid-pattern shader for our floor. Then, I added the sand texture for the floor, which we ended up actually removing at the end of the project. Towards the end, I added the pebbles to the floor, implemented logic to make the floor endless, implemented a function to check if two box objects were intersecting, did a little bit of CSS styling for small parts of the HUD, and debugged our code to make deployment seamless.

Kai Obinata

I assisted a bit in implementing the camera, positioning it a certain distance relative to the player at all times. I also took on the role of searching for and dealing with mesh resources, and implemented the bird mesh into the game.

Anthony Ng

I took on the role of creating and implementing the floor and boundary walls of the game, as well as implementing simulated dynamics in the game, which I did through integration of the cannon-es library. This allowed me to implement the jumping feature of the game and also

allowed the player to fall down to the floor after jumping. Furthermore, to ensure that the player did not fall through the floor or pass through the restriction walls, I also added added restrictions to moving the movement of the physics body of the player by preventing movement when the player is at the boundary of the walls and adding the floors as another body to the physics engine so that the player would land on top of the floor instead of falling through it. I also implemented the feature of changing between the first-person and third-person views of the player through the press of a button, which I did through the addition of an event handler that would change the position of the camera relative to the coordinates of the player model within the scene.

Attributions

We solely built off of the starter code given to us.

Art Resources

- Cactus Object: https://opengameart.org/content/low-poly-cacti
- Cactus Texture: https://opengameart.org/content/freebies-mundo-commissions
- Bird Object/Texture: https://sketchfab.com/3d-models/low-poly-bird-651a09f1866447688ab88b06090b054d
- Player Object/Animations: https://www.mixamo.com/#/
- Pebble Texture: https://sketchfab.com/3d-models/small-ground-stone-pebble-adbc195285d7469e9d51bd088003659c
- Sand Texture (Not used but commented out in code): https://img.rawpixel.com/private/static/images/website/2022-05/px1266865-image-kwvw6ica.jpg?w=1200&h=1200&dpr=1&fit=clip&crop=default&fm=jpg&q=75&vib=3&con=3&usm=15&cs=srgb&bg=F4F4F3&ixlib=js-2.2.1&s=4fc5dcaf6cc9d91b706218e00c1f7600

Audio

- Running Sound: "Running, Snow, A.wav" by InspectorJ (www.jshaw.co.uk) of Freesound.org
- Jumping Sound, Death Sound: **Sound from Zapsplat.com**

List of documentation and snippets that we examined: https://docs.google.com/document/d/1zGHW2hl6xZwIlcQK7Wggd_zEoD41QLwnwi6yIJ_iMp8/edit?usp=sharing