Dino Dash: Now in 3D

Anika Agarwal, Diya Dalia, Juliana Li

Abstract

Dino Dash is an infinite third-person game inspired by the dinosaur game on the Chrome browser that appears when there is no internet. Our game is a three dimensional recreation of the original game where users control a dinosaur by moving left, right, and up. Players are expected to either dodge the obstacles (cacti, birds) in their path or jump over them, and the game ends upon a collision with an obstacle. This game combines childhood nostalgia with exciting new visuals to put a twist on a timeless classic.

Introduction

Goal & Previous Work

In a third-person infinite runner game, the player's goal is to stay alive for as long as possible. We were inspired by Google's 2D dinosaur game, which provides Chrome users with some solace during times of no internet. In the Chrome game, users have a side view of the dinosaur and are expected to jump over the cacti that appear along the dinosaur's path. Dino Dash is an infinite runner game built using Three.js to create a 3D version of this iconic game. Our game allows players to move the dinosaur left, right, and up using the arrow keys or the WASD keys. The player's goal is to avoid the obstacles that arise along their path and survive for as long as they can. Google's current game is extremely popular, but we wanted to improve upon this by changing it to be three dimensional, making it more visually appealing, and providing players with more customization options.

Approach

The game begins on the homepage, which provides instructions for unfamiliar users. The homepage is also where the player has the ability to render their dinosaur in three different styles: original, cartoon, and realistic. Once the user selects the desired look of their dinosaur, the game begins with the dinosaur running on a path with a score counter on the top-left of the screen. The longer the player stays alive, the higher their score will be. The scoreboard also displays the highest score that the player has been able to achieve thus far.

When the game is initially loaded, the camera will be looking from behind the dinosaur at a slight angle. One distinctive feature of our game is that users have a fun way to customize the game to their desired settings. Users can rotate between the 5 following camera angles by clicking the 1-5 number keys on their keyboard: (1) default view, (2) view from behind the dinosaur, (3) view from above the dinosaur, (4) bird's eye view of the dinosaur, and (5) wide side angle view of the dinosaur. Playing from these different angles adds a level of complexity to our game.

Another addition to our game is a pause feature. Users can click the ESC key at any time while playing to pause the game. A popup will appear instructing the user to either click ESC again to resume the game or click a button to be redirected back to the home screen. This feature gives users more flexibility by allowing them to pause the game instead of sacrificing a great run

because of a possible inconvenience. This feature also gives the users an option to return back to the homescreen if they want to restart or switch the style of their dinosaur.

As the player moves forward, cacti begin appearing along the path for the users to avoid. In addition to being spawned alone, cacti are also spawned in clusters to pose an additional challenge for the user by forcing them to move across the entire path or to jump over them. Once the user's score is greater than 50, birds are also spawned as another obstacle for users to avoid. Additionally, every time the user's score increases by 50, obstacles are spawned at a faster and faster rate to increase the difficulty of the game the longer the user plays.

When users meet their untimely demise, a "Game Over" popup appears with a button to allow the users to play again. Pressing the button will redirect the user back to the home page, where they can once again choose their character.

Overall, this game is designed for individuals who enjoy the dinosaur game on Google Chrome but want a new challenge. These people are already familiar with the concept of a third-person infinite runner game and would likely appreciate the additional functionality our game provides.

Methodology

Starter Code and Approach

To build our game, we used the starter code provided by the COS426 course staff. Low-poly models were then found online and loaded into our game to generate the characters and scene objects. Occasionally, we used Blender to modify the color or shapes of the models that we found online to better fit the desired aesthetics of our game.

User Controls, Player GLB Meshes, and Animations

To move the dinosaur left and right, players can use either the A / D keys or the left and right arrow keys. When any of these four keys are pressed, an internal flag is raised in the `js/EventHandlers` file using a javascript event listener for keydown events. While the flag is raised, the dinosaur's position along the x-axis is incremented (decremented) to replicate moving left (right). Once the key is released, the internal flag is unraised in the `js/EventHandlers` file using an event listener for keyup events to stop the movement of the dinosaur along the x-axis.

To make the dinosaur jump, players can use either the W key, spacebar, or the up arrow. Similar to the left and right control movements, when one of the jump keys is pressed, an internal flag is raised in the `js/EventHandlers` file using a javascript event listener. When the internal flag is raised, the character's position along the y-axis is incremented by 3 if the y-position of the character is equal to the y-position of the floor (the character can only be allowed to jump if it is on the floor). To make this movement smooth, `TweenJs` is used with a quadratic-out easing function to replicate the action of jumping up. Immediately after the character is at the desired y-position, `TweenJs` is used with a quadratic-in easing function to bring the character back down to the floor. Using `TweenJs` on the movement up and down ultimately allows us to replicate a realistic smooth jump.

To simulate a dinosaur running on an infinite path, we decided to use an approach where the dinosaur is kept at a fixed position along the z-axis, and the position along the z-axis of every other object in the scene is adjusted. This ultimately allows us to create the illusion of the dinosaur running on an infinite path without having to adjust the camera position and view dynamically as the dinosaur moves. To make this illusion more realistic, we used `Three.js' AnimationMixer` to run the walking animations defined within the glb files of each of the dinosaur styles.

Dinosaur Styles

There are currently three styles that users can pick from: original, cartoon, and realistic (see Figure 1). Low-poly models of each of these styles were found on Sketchfab and loaded into our game with `ThreeJs' GLTFLoader` function. Within each of the models seen in Figure 1, the models were already pre-programmed with either walking or running animations and are called using `ThreeJs' AnimationMixer`.

Figure 1: Original, Cartoon, and Realistic Dinosaur Styles (from left to right)

Scene and Obstacle Generation

Our scene consists of 3 main parts: `scene_left`, `scene_right`, and `floor` (see Figure 2). To generate our scene, the initial `land` asset provided in the source code was stretched along the z-axis to give the illusion of an infinite path. Two of these land assets were used to generate `scene_left` and `scene_right`. The color of the `land` asset was also changed within Blender and stretched along the z-axis to create the `floor`.

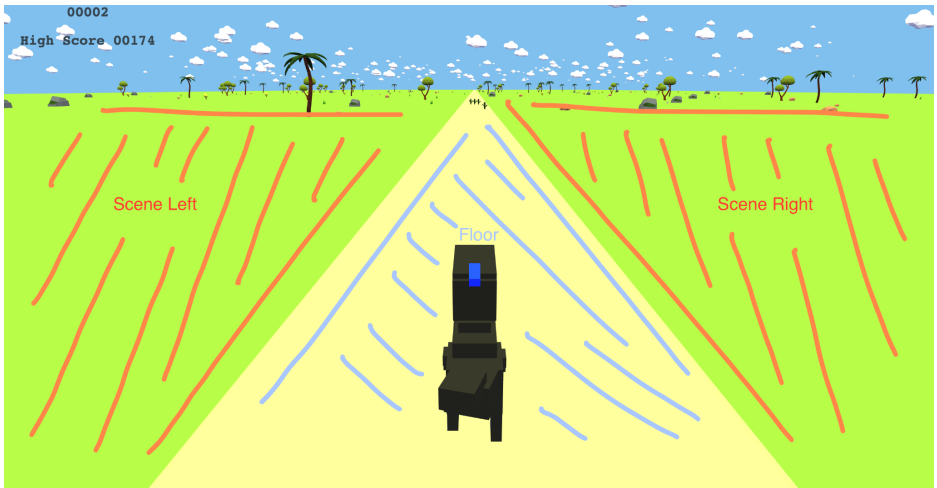

Figure 2: Diagram of scene_left, floor, and scene_right

To generate the obstacles in the scene, 100 random obstacles (tree, palm tree, bush, flower, grass, and 2 types of rock) are loaded at a random position on each side of the scene (`scene_right` and `scene_left`).

The objects are then added to the scene's `obstacles` array. Every single time the scene is updated, the position of every obstacle in the scene's `obstacles` array is shifted along the z-axis towards the player. If an obstacle moves behind the player past a certain z value, the obstacle is removed from the scene and another obstacle is generated in front of the player.

To generate the clouds in the sky, 250 clouds were generated of varying sizes at random positions. Each cloud was added to the scene's `clouds` array. During the scene's update function, the position of every cloud in the scene's `clouds` array is shifted along the z-axis towards the player. If a cloud moves behind the player past a certain z value, the cloud is removed from the scene and another cloud is generated in front of the player.

Scoreboard and High score

To display the scoreboard, the `createScoreboard` function used javascript to add HTML to the game. The score that is shown is calculated based on the `timestamp` parameter that is passed into the scene's update function. Because the score is determined by the `timestamp` parameter, an adjustment is made when the game is paused to ensure that the score does not increase while the game is on the pause screen.

If a user earns a high score, the new high score is stored in the window's local storage available through the window interface. When the user starts playing a new game, the high score is retrieved from local storage and displayed as part of the scoreboard.

Camera Views

To implement several preset camera angles, we used javascript event listeners for keydown events. Each of the keys from 1-5 are binded to a specific preset `camera.position` and `camera.lookAt` position to create the desired camera view. When any of the keys associated with a preset camera view is pressed, the camera is adjusted to the desired view.

Increasing Game Difficulty

In general, one important feature of games is the increasing difficulty as the game progresses. To increase the difficulty of the game, birds (with either the original, cartoon, or realistic design based on the dinosaur's style) are spawned when the user's score is over 50. Similarly, every time the user's score increases by 50, the spawn rate of the objects increases, resulting in more obstacles to avoid as the game progresses. We also programmed the score function to increase at a slower rate as the player gets further into the game.

Collision Detection

An accurate collision detection function was one of the biggest challenges that we faced as a group. When we initially implemented the collision detection function, we calculated the

bounding boxes of the dinosaur mesh and the bounding boxes of each of the obstacles by using the `Box3().setFromObject` function. We then implemented our `detectCollision` function to return true if there was an intersection between the bounding box of the player and the bounding box of any of the obstacles. However, we realized that the bounding boxes of the player and the obstacles were not very accurate. The bounding boxes were often bigger than the actual mesh object which resulted in a lot of false collisions.

To improve our collision detection, we tried to prevent false collisions by using the bounding box of the dinosaur character and the position of the center of the obstacles to determine if a collision occurred. Our function would return true if the position of the object's center was inside of the bounding box. After initial testing, we realized that this method was still detecting collisions inaccurately. The dinosaur player's bounding box was still a lot bigger than the actual mesh object which would often invoke a collision when a collision never happened. Similarly, using the position of the center of an obstacle would often result in either a delayed collision (when a collision actually occurs, our function will detect a collision with a bit of a delay) or an undetected collision (a collision occurs visually—ex: the dinosaur clips the edge of an obstacle—but our function does not detect a collision).

To improve our collision detection, we experimentally determined an offset for each of the dinosaur styles to create an accurate bounding box. Similarly, we determined an offset for each of the obstacles through extensive testing. Using the offsets and the bounding boxes, we detected collisions if there was an intersection of the bounding boxes. Using this offset method, we were able to arrive at a fairly accurate collision detection function.

When the scene is updated, the position of each obstacle is moved along the z-axis to be closer to the player. After shifting each obstacle, the scene's `obstacle` array is looped through to check if there was a collision between the player and an obstacle. If a collision occurred, the `showGameOver` function is invoked to end the game and show the end scene.

Home, Pause, and End Scene

When the app first loads, the app calls the `startScreen` function to add in HTML to create and display the home screen. After the user chooses one of the desired styles, the style is passed in as a parameter to the `SeedScene` constructor. The scene with the desired style is then generated, and the start screen is then hidden to show the game.

Figure 3: Home screen with buttons for users to select dinosaur style

To implement the pause feature, we used javascript event listeners for keydown events. Whenever the ESC key is pressed and the game is currently not paused, the `togglePause` function is called which adds HTML to the game to display a pause popup. Within the scene's update function, a `break` function is hit whenever the game is paused to prevent the scene assets from moving or the score from updating. If the ESC key is pressed and the game is paused, the `togglePause` function is called again but the HTML popup is hidden this time, and the scene and all of the scene objects are updated. Within the pause popup, we have a button that redirects the player to the home screen when clicked.

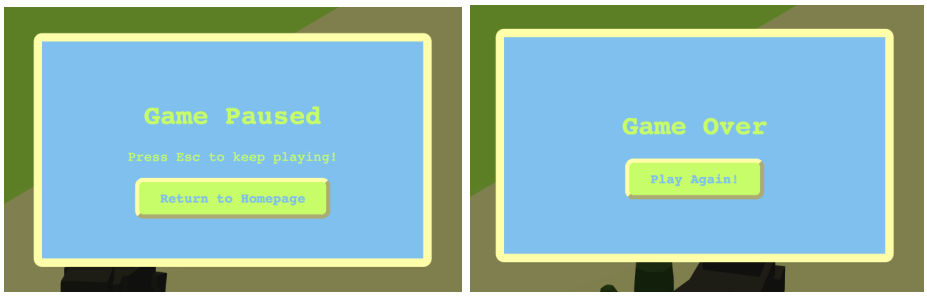

Figure 4: Pause screen and Game Over screen.

After detecting a collision, the `showGameOver` function is called which adds HTML to the game to display an end game popup. Within this function, the rest of the screen is dimmed gray and the score and game objects are paused (the scene update function stops moving the assets' positions). Similar to the pause popup, the end game popup also has a button that redirects the player to the home screen when clicked.

Results

As we built out our game, we measured our success by our excitement to play our own game and our desire to show it to others. Each improvement to our animations, aesthetics, and

collision detection made us more confident in our abilities as developers and more eager to share the game with friends.

While testing our game, we primarily focused on fixing bugs in our code and ensuring that our game was playable. Some of the challenges we encountered were ongoing movement during a paused game, inaccurate collision detection, and buggy animations of the glb models. To remedy these issues, we performed extensive testing on our event handlers, tried to crash into different objects intentionally, and changed our camera angles to observe collisions more clearly. This extensive testing helped us remedy existing bugs and identify unexpected ones.

After a substantial amount of testing, we believe that our game is playable and users should not encounter bugs while playing. Once we had completed internal testing, we sent our game out to friends who served as beta testers and gave us feedback that we incorporated into our game. This included modifying some of our color schemes to make the game more aesthetically pleasing and further improving our collision detection. We found that modifying the bounding boxes of the objects helped us make the collisions more realistic. When we asked our friends to replay the game, our improvements were well received by users as many of them were impressed by the game's smoothness and customizability.

Discussion & Conclusion

The approach that we took for developing this game was very promising because we were able to complete all of the necessary features for our MVP. Initially, our goal was to make a 3D version of the Google Chrome dinosaur game, which required creating an infinite moving scene. To accomplish this, we programmed our game so that the dinosaur was kept in place along the z-axis while all of the other objects on the screen were moved towards the player. This, along with the walking animation of the glb models, simulated a "running" dinosaur. Once we had completed our MVP, we decided to expand upon our initial idea by adding extra functionality. Instead of just having one dinosaur, we gave the user the option to choose from 3 different dinosaur styles. We also built out a homepage where the user could select their dinosaur. Another feature we added was a pause screen, which does not exist in the current version of the Chrome game. Lastly, we added in various camera angles. We feel that our development approach really streamlined the process of designing a fun, user-friendly game.

Some future work on this game would be to modify the surroundings based on the type of dinosaur the user selects. Currently, we maintain the same blocky objects for all three dinosaurs, even though this is best-suited for the aesthetic of the original dinosaur. User feedback was that the realistic dinosaur should have a realistic background, similar to that of our homescreen; the same applies for the cartoon dinosaur. Another aesthetic feature we could have implemented would have been the inclusion of sounds such as footsteps, roars, or birds cawing to make the game more realistic. To make the game more engaging, we could add additional obstacles of varying sizes and include power-ups for the dinosaur if it catches food that falls from the sky.

One feature that is currently in the Chrome version of the game that is not in our current game is the ability for the dinosaur to duck when the down arrow is pressed. Pressing the down arrow would not only change the dinosaur animation to mimic a dinosaur ducking its neck down but it would also make the dinosaur fall down to the ground faster if it is in the middle of a jump. We tried to implement this functionality but struggled in changing between the standing dinosaur glb model and the ducking dinosaur glb model every time the down arrow was pressed. Due to the short implementation time frame, we decided not to pursue this feature.

By doing this project, we learned a great deal about how to interact with meshes using `Three.js' GLTFLoader` and how to animate movement through `Three.js' Tween` and `AnimationMixer`. We figured out how to acquire the various assets from Sketchfab needed to make a game and also how to modify their properties in Blender to best fit our needs. We gained a better understanding about the complexities of collision detection using 3D objects. We also improved our knowledge of HTML and CSS formatting as we overcame a plethora of challenges while designing our homepage.

Contributions

Juliana worked mainly on the character animations and movements, scene building, and obstacle generation. Anika worked primarily on improving the collision detection, camera angles, and the scoreboard. Diya worked on the home screen, pause screen, end screen, and improving the cohesiveness of the game between the home, pause, end scene, and the game itself. We all worked together on the written report and video demo.

Work cited

Tutorials / Explaination pages

- Discover threejs book: https://discoverthreejs.com/

Assets

Dinosaur glb assets:

- Original: https://sketchfab.com/3d-models/chrome-dino-3d-animated-d1e8d87cae4e4e5da7880b695a54d678
- Cartoon: https://sketchfab.com/3d-models/dinosaur-walking-9b3dc24306ee4d85834a84a34a247586
- Realistic: https://sketchfab.com/3d-models/animated-tyrannosaurus-rex-dinosaur-running-loop-38007d947ae74dea83988cb0b08ee053

Bird glb assets:

- Original: https://sketchfab.com/3d-models/chrome-pteranodon-6e3e48f29bbb4f61836d4b73c63e67fb
- Cartoon: Parrot asset from THREE.js
- Realistic: https://sketchfab.com/3d-models/pterosaur-quetzalcoatlus-5ef112b3208d4444a51e480beca56277

Scenery and obstacle assets:

- Cloud: https://sketchfab.com/3d-models/cloud-852ae741a60b44ec886fbf34bbd058f8
- Rocks and Trees: https://quaternius.com/packs/simplenature.html
- Palm Tree: https://quaternius.com/packs/ultimatecrops.html
- Rock and cactus: https://quaternius.com/packs/ultimatenature.html

Land and Flower assets

- Land: from source code (originally from the Google Poly object)
- Flower: from source code (originally from the Google Poly object)

Start Screen Background Image

- Dreamworks: AI image generator used to generate the background image of our start screen