Nunki - Tower Defense Game
Final Report

CS 467 Online Capstone Projects
June 9, 2019

Christopher Frenchi
Jonathan Ruby
Ruben Torres

Introduction

Team Nunki has developed an HTML5 web-based Tower Defense game using a 3D Javascript library, three.js, to give the traditional 2D tower defense board some pizzazz. We modeled, rigged, and animated our world elements using Blender and exported to three.js using GLTFLoader.js.

Our team began this project with research into 3D world modeling using WebGL, and have rendered a lego-inspired world for our users to experience.

Description

This program is a tower defense game that the user can play and try to beat. The premise of the game is that dinosaurs are stealing the pirate's treasure. The user, or the lead pirate, must build defensive towers in order to stop the dinosaurs from stealing all of the treasure. The towers attack the dinosaurs and prevent them from reaching the end of the path. Each level has ten waves of dinosaurs that the user must defeat. Each wave gets a little harder. Each defeated dinosaur gives the user a little treasure that can be used to build more towers. The user has 10 lives per level and each dinosaur that makes it to the end of the path subtracts a life from the user. If the user survives all three levels, they win.

The user starts a new game which creates the first level. Each level is a square grid with areas that the user can build towers on. The buildable areas are the spaces without palm trees and that are not part of the path. The user can build towers as long as they have enough coins. The user can build towers at any time. There is a countdown timer that shows how long until the next wave of dinosaurs starts. The game saves after each wave of dinosaurs. If the user loses all of their lives, they have the option to restart or to go back to the previous save. If the user exits the game, a cookie is saved and they will have the option to load the game at the beginning of the last wave started. If the user beats all three levels, a credits screen will appear.

Usage and Gameplay

Starting the game locally

Unzip the attached file and using the terminal inside of nodetest, run:

```
npm install
npm start
```

This will start a version of the game that is accessible by the browser if you go to localhost:3000.

Starting the game through the hosted website

Follow the link provided (http://3.91.212.9:3000/) which will take you to the Menu Screen.

Objective

The objective of the game is to build various pirate towers that shoot projectiles to stop dinosaurs from getting to the end of their path.

Controls

To make a selection using mouse/pad/touchscreen:

- Click to press buttons (All Screens)
- Double Click to select a location on the map (Play Screen Only)

To change your view using mouse/pad/touchscreen:

- Click, hold, and drag to change perspective (Play Screen Only)
- Scroll/Pinch to zoom in and out (Play Screen Only)

Menu Screen

On the menu screen, you have two options available:

1. Start a New Game Button
2. Load a Game Button

Clicking **Start Game** will start a new instance of the game at level 1. *Clicking* **Load Game** will load your score (coins, lives, wave, level, towers) from the last successfully completed wave. Load Game will start a new game if you have not successfully completed a wave before. Clicking either option will direct you to the Play Screen, where the game is played.

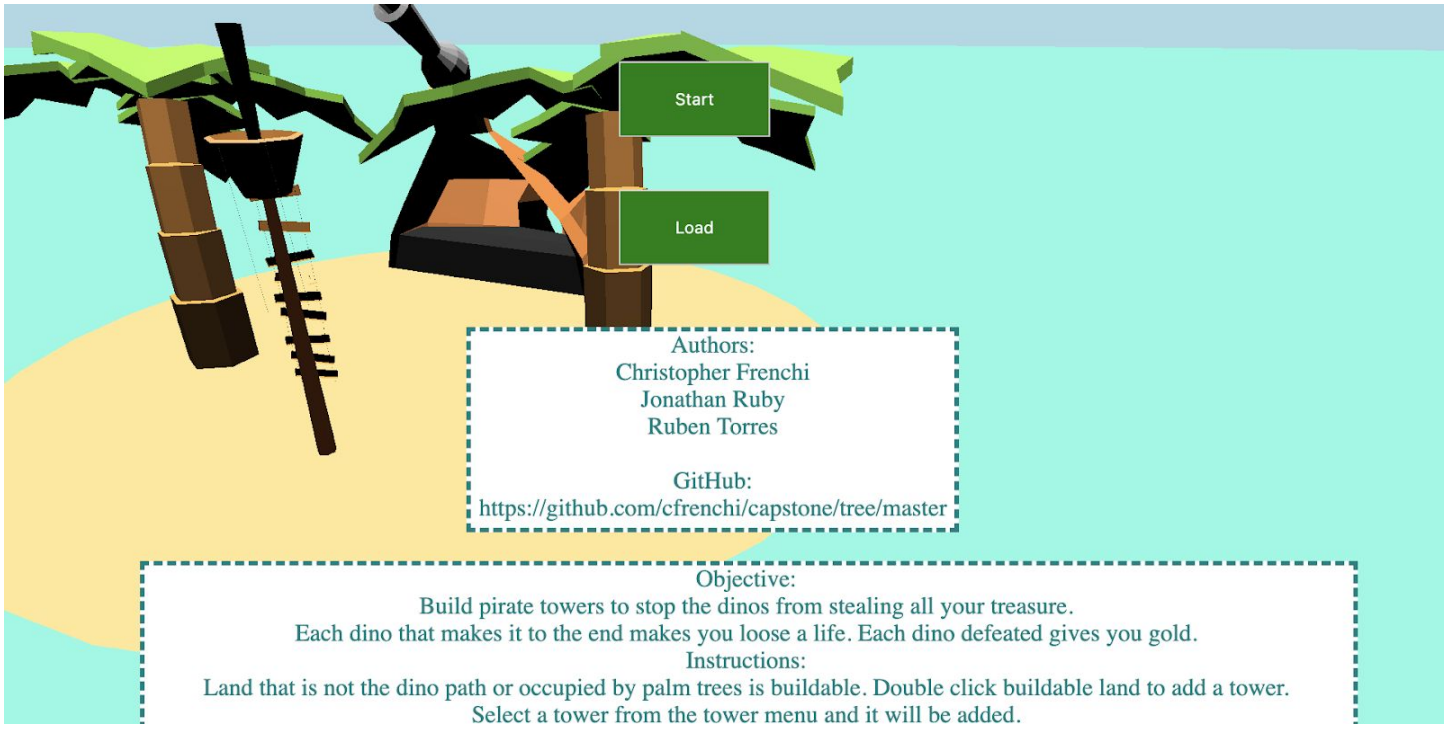


Image: Menu Screen

Play Screen

There are several key components of the Play Screen:

1. Status Bar
    a. Coins
    b. Lives

    c. Waves
    d. Level
    e. Sound Button
2. Wave Countdown Timer
3. Map Grid
    a. Non-Buildable Land
    b. Buildable Land
4. Build Tower Bar
    a. Musket Tower Button
    b. Cannon Tower Button
    c. Cabin Tower Button
    d. Exit Button
5. Towers
    a. Musket Tower
    b. Cannon Tower
    c. Cabin Tower
    d. Radius
    e. Projectiles
6. Dinosaurs
    a. Oviraptor
    b. Velociraptor
    c. T-Rex
    d. Health Bars
7. Waves

The **Status Bar** is located at the top right of the Play Screen. It is always visible during play. It will show you how many coins and lives that you currently have, as well as your current level and wave. Additionally, within the Status Bar is a button that allows you to toggle the background music between on and off.

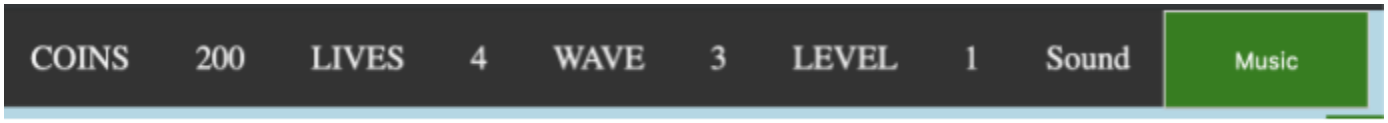


Image: Status Bar

Just below the Status Bar toward the middle of the screen is the **Wave Countdown Timer**. The Wave Countdown Timer displays when the next incoming wave is about to start. The Wave Countdown Timer will also announce the start of each wave by displaying “Wave <value> Incoming!” At the start of each level, or if you are returning to the game through Load Game, you are given additional time to build towers (approximately ten seconds). The Wave Countdown Timer will display “Get Ready” before beginning to countdown to the next upcoming wave during this this ten seconds. If you happen to make it to the last wave of any level, the Wave Countdown Timer will display “Stay Alive!” - let’s hope you make it that far!

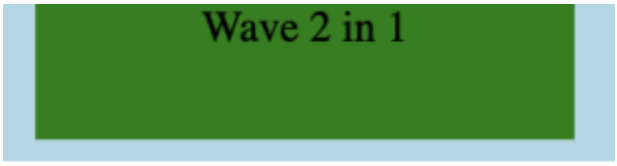


Image: Wave Countdown Timer

The **Map Grid** is the primary playing space of the game. The map will change between levels as you progress through the game. There are two categories of spaces on the Map Grid, **Non-Buildable Land** and **Buildable Land**. Non-Buildable Land are spaces on the board that are occupied by the environment (trees, caves, sand, water), towers, or the path taken by Dinosaurs. You will not be able to build new towers on Non-Buildable Land. Buildable Land are green spaces not already occupied by the environment, towers, or path. By *double clicking* Buildable Land you will see the Build Tower Bar appear at the bottom of the screen. Initial perspective of the Play Screen is set to nearly the center of the Map Grid. Zoom out and change perspective to see the vibrant, lego-reminiscent textures and colors of the environment. You should notice the palm trees swaying in the wind.

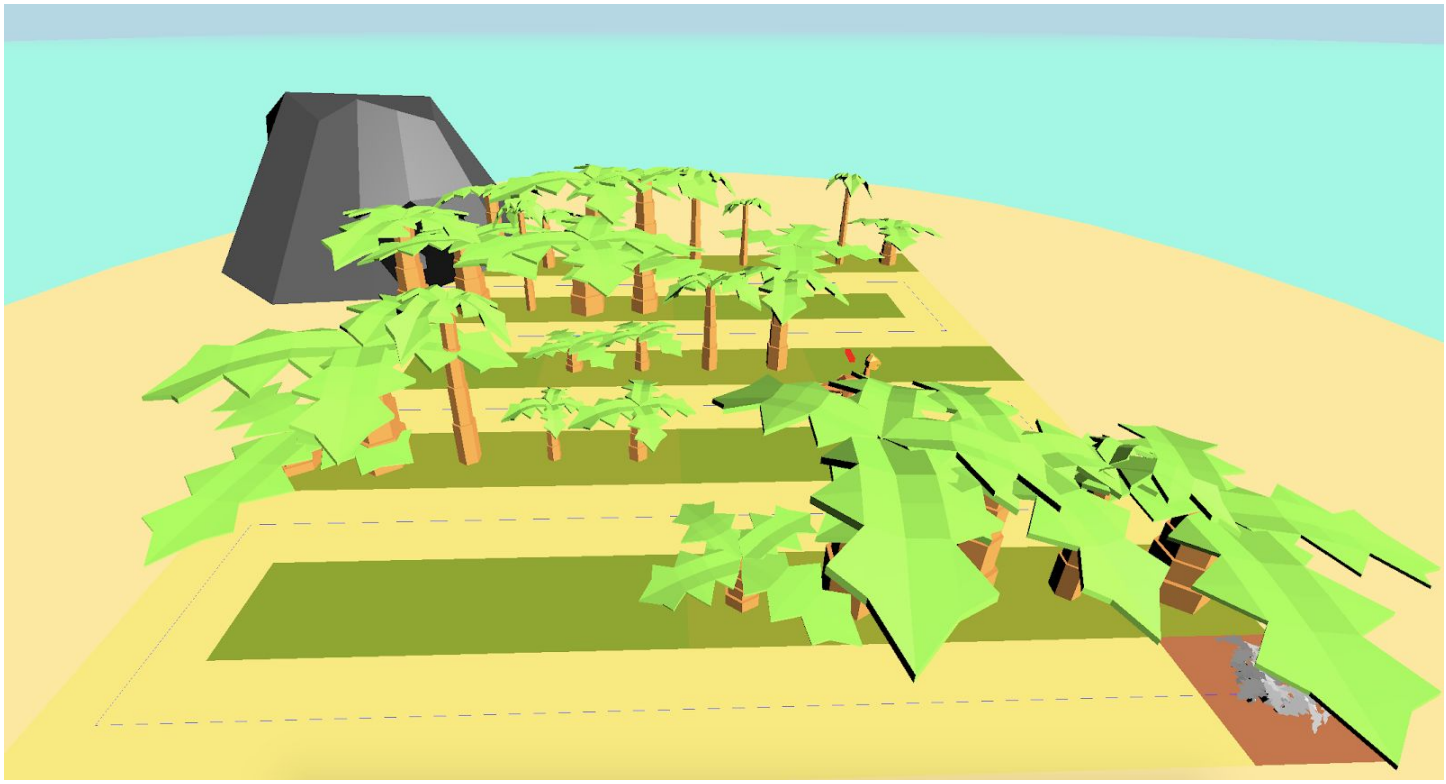


Image: Map Grid Level 1

The **Build Tower Bar** will allow you to choose which towers to build at a given space and only appears when you *double click* Buildable Land. The Build Tower Bar is composed of four buttons; **Musket Tower Button**, **Cannon Tower Button**, **Cabin Tower Button**, and the **Exit Button**. Tower buttons allow you to build the given tower in the space that was double clicked. If you do not have enough coins to build a given tower, that particular button will not be available. If you do not have enough coins to build any tower, you will only be able to click the Cancel Button. Buttons will become available if you gain enough coins while the Build Tower Bar is open. By *clicking* a tower button within the Build Tower Bar, the given tower will be built at the specified space on the Map Grid, the space will become Non-Buildable, and your coins will decrease accordingly. If you do not wish to build at the specified location, you will need to *click* the Exit Button.

Image: Build Tower Bar; Cabin Tower not Buildable

There are three types of towers: **Musket Tower**, **Cannon Tower,** and **Cabin Tower.** Each cost different amounts of coins to build, have different attack power, and different radius size. The Musket Tower has the largest radius, cost the least to build, and weakest projectile. The Cannon Tower has a medium radius, cost more than the Musket Tower to build, and has a projectile strength between the Musket Tower and Cabin Tower. The Cabin Tower has the smallest radius, cost the most to build, and has the strongest projectile.

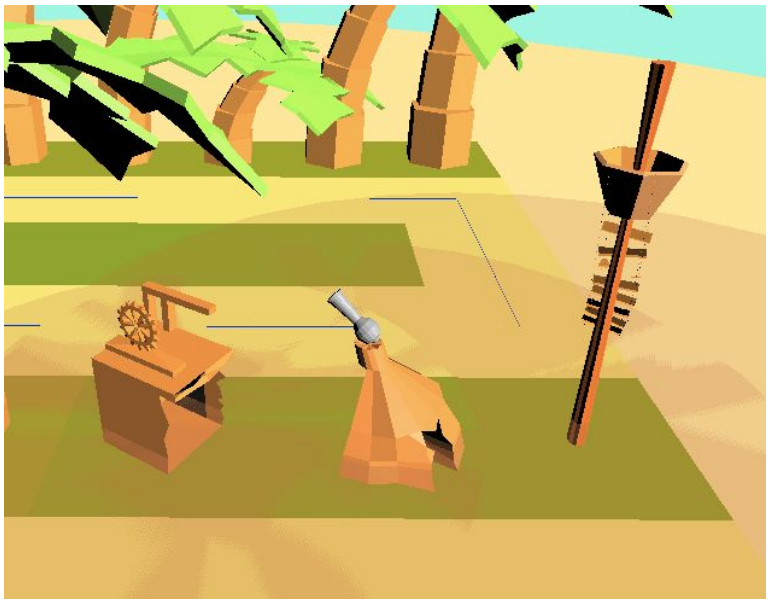
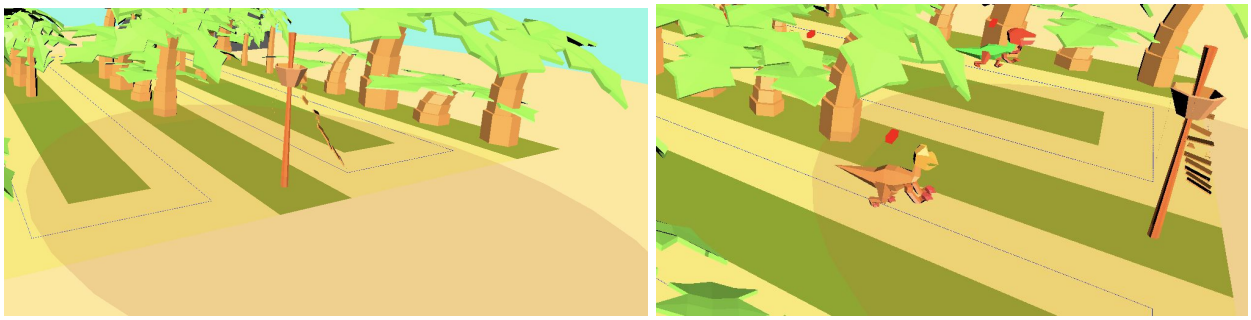


Image: Cabin Tower (left), Cannon Tower (middle), Musket Tower (right)

Each tower's **Radius** is visible on the Map Grid. When a dinosaur is within the radius, the tower will shoot its **Projectile** toward it. Projectiles decrease a dinosaur's health bar.



Images: Tower with radius (left), Dino with health bar (right)

Dinosaurs move along the path on the Map Grid toward your treasure during each of the waves. They will spawn at the entrance to their cave based on the wave. There are three dinosaurs; **Oviraptor**, **Velociraptor**, and **T-Rex**. Each type has unique stats, including movement animation. The Oviraptor has the least amount of life, but moves the fastest. The Velociraptor is much sturdier than the Oviraptor, but moves a bit slower along the path. In fact, you might even see Oviraptor run right past a Velociraptor. The T-Rex will appear much later in each level, but as the strongest of the group, it is a formidable opponent.

Images: Ovi (left), Velociraptor (middle), T-Rex (right)

Another key element for each dinosaur is its **Health Bar**. This is a red bar above its head to indicate its remaining health. As its health decreases, the health bar will shorten. Once its health bar reaches zero, the dinosaur is eliminated from the path and you will gain additional coins to continue to build. As a dinosaur reaches the end of the path your Lives will decrement by 1.

As mentioned, Dinosaurs will appear in **Waves**. You'll know a new Wave is approaching based on the Countdown Timer. If you manage to live through the wave, your stats and towers will be saved so you may return to the start of the proceeding Wave via the Lose Screen or Menu Screen if you were to lose all your lives or navigate away from the page in your browser before the next wave finishes.

If you successfully reach the end of the final wave of Level 1 and 2 without losing all your lives, you will be taken to the next level, where a new map will be loaded into the Map Grid. During the transition you will see a blue box momentarily spin as the next level loads. Once loaded, your stats within the Status Bar will update to the new level's settings. You begin each new level without towers, full lives, and a certain amount of coins.

If you successfully reach the end of the final wave of Level 3, the entire Map Grid and Status Bar will be removed and you will be taken to the Credits Screen.

If your lives reach zero, you will be taken to the Lose Screen.

Credits Screen

Once you reach the **Credits Scene**, the credits will roll across the screen over approximately 35 seconds, at which point you will return to the Menu Screen. The credits are similar to the Star Wars intro, so the text will start from the bottom and move away from you at an angle. Our blue friend in the middle of the screen provides additional depth and spins as the credits roll.

Image: Credits Screen

Lose Screen

If your lives reach zero, the camera will pan in to provide you with two options:

1. Load Game
2. Return to Menu

By *clicking* **Load Game**, the game will return to the end of the last successfully completed wave before you ran out of lives. By *clicking* **Menu**, you return to the Menu Screen. While on the Lose Screen, you will continue to see (and hear) Dinosaurs move along the path until you click Load or Menu.

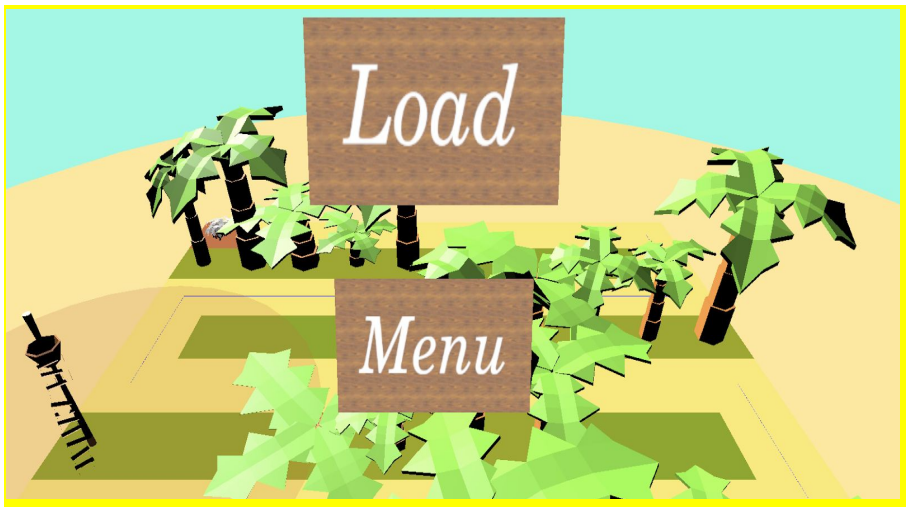


Image: Lose Screen

Software and Systems

Nodejs, express, home.handlebars, mapX.js, rules, towersX.js, dinosX.js, buildUI, save, credits, assetloader, path, gameui

At its heart, this is a node.js application that is run as a server using Express and handlebars.

The game itself is started in the home.handlebars script tag. It is here where we load in all of our supporting libraries, javascript files, and initialize our three.js scenes.

This program has a lot of moving parts. The files have been separated out to try and create a logical flow of how the program is working and where certain functions reside. It can be broken out into three.js aspects and our game engine.

Three.js overview- *init(), animate(), render(), OrbitControls, setting camera, scene, renderer, assetLoader()*

Learning three.js was a lot different from what we have experienced in our CS classes. Taking CS 475 in conjunction with this class had some overlap with OpenGL that was covered at the end of the quarter as three.js is a library for WebGL. They share a common workflow, using three.js made this a lot easier. The way that we learned three.js was through the documentation and examples provided on the three.js website. This was a great boon for us as it showed us from a mix of examples that would become prototypes to make sure that this was possible. Through much trial and error, we have evolved from no exposure to graphics programming to hosting a graphical web based game on our own AWS Lightsail instance.

To describe how the scene is rendered, I will quote the three.js documentation:

"To actually be able to display anything with three.js, we need three things: ***scene, camera and renderer,*** *so that we can* ***render*** *the* ***scene*** *with a* ***camera****."* -https://threejs.org/docs/index.html#manual/en/introduction/Creating-a-scene

This was our initial understanding of creating our game using three.js and it's functions. We first initialize our scene in our init function which creates the scene, camera, and renderer. We create an animation loop that continually renders the scene with the camera. This allows us to add elements to the scene and update them continuously.

We used the OrbitControls.js file to allow for user interaction and added in double touch controls so that the game can be used on a mobile device.

Game Flow- *menu(), play()*

All of this initial rendering setup is wrapped around our game flow detailed in our "High Level Game Flowchart" seen below.

Menu.js - After initialization and setting up our loadingManager, we call the file menu.js. This creates the scene the user sees when they load our project's website or when it is run locally. If the user clicks play, the play function is called from within home.handlebars. If the user clicks load, the loadGame function is called from load.js.

Play(loaded) - This function creates the user's board and sets the variables for dino path and triggers the waves outlined in rules.js

When the play function is called, the function will see if play is being triggered from a cookie or from a new game. This distinction dictates what level is loaded and what game variables are set. Once this happens, the appropriate mapWave will be called and the player will be set up to start playing! The high level logic of the menu flow is seen in the image below.

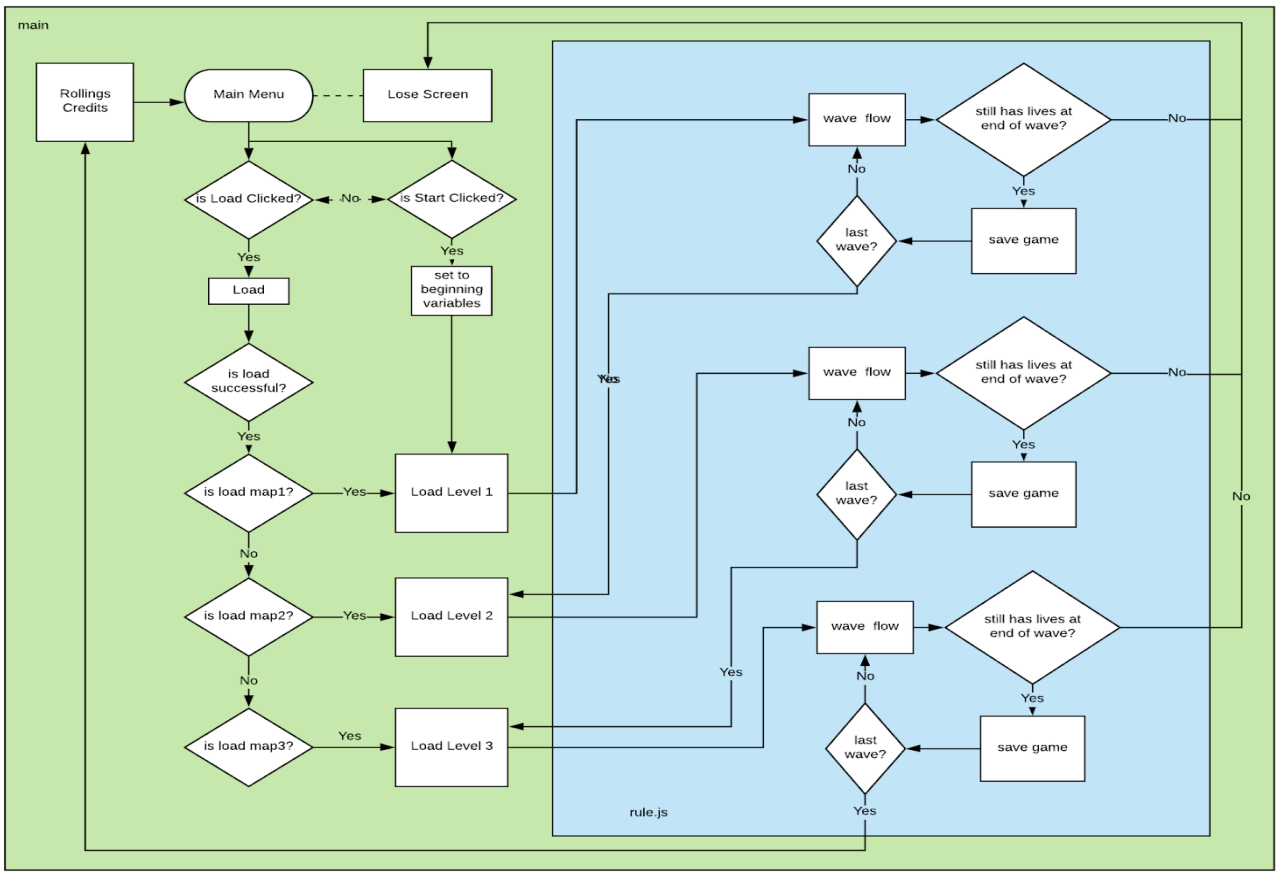


Image: High Level Game Flow

Building a tower- *play(), viewBuildMenu(), assetLoader(), buildUI*

The logic of building a tower has been explained previously when discussing the placement of towers on buildable land. In order for the player to build a tower, they must double click on "Buildable" Land. The logic for this comes in the play function and in the buildui.js. By checking the user variables and the current location of the mouse clicks, the build menu can be opened or updated to reflect if the user has enough coins to buy new Pirate towers!

Raycasters were used to determine where on the map the user double clicked. This was translated to our map grid array to get back proper placement for where the towers would be placed within the three.js scene. When the user has clicked on buildable land, clicks on a tower choice, and has enough coins to build that tower, the chosen tower is added to the three.js scene. By using our assetLoader specifically for our towers, we can add to our towers array that will be saved in our cookie and prevent towers from building on top of each other.

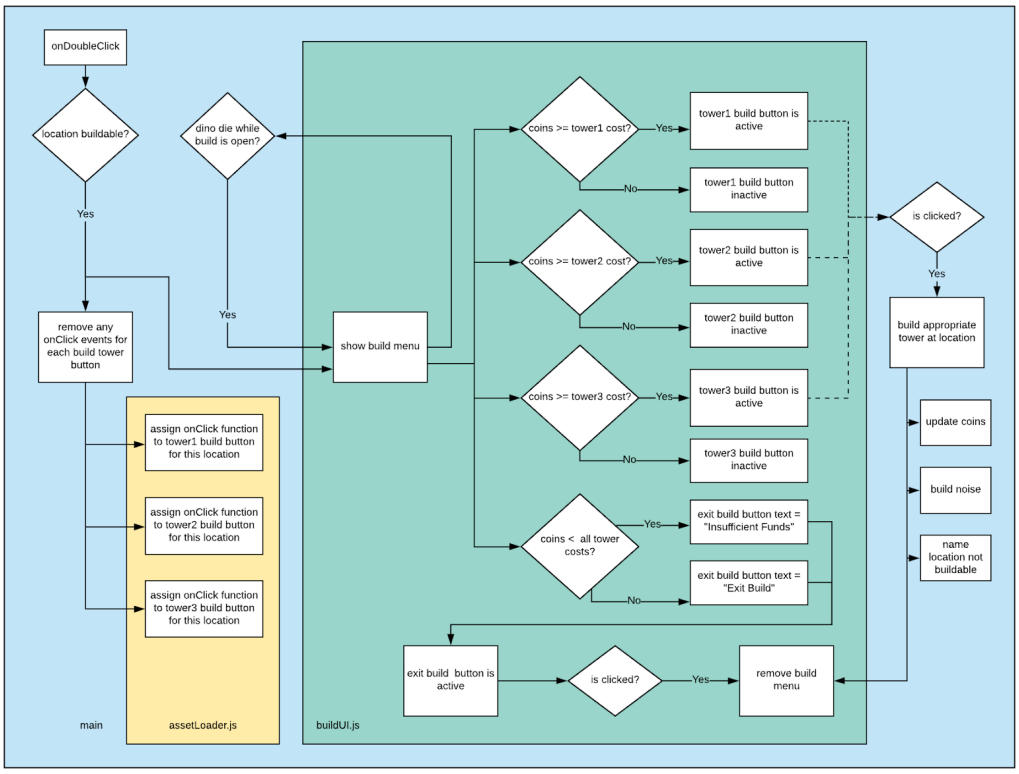


Image: Build-A-Tower Process

Game Rules- *mapWaves(), assetLoader(), path.js*

Setting up the game rules can be found in rules.js when the function mapWaves is called from the play function. This controls the logic that is used in the Countdown Timer that the user sees and when the waves are released and what dinos are spawned. Using javascript's setTimeout function added in the need for additional checks as the waves are created and need a means to be deleted prior to the next wave.

Once the play function is called and the tiles are added, the rules will check the current wave and see if it is less than the total waves. This will check if the level has been been won or if you've lost. If you're not on the last wave and you have 0 or less lives, you lose and the Menu overlay is displayed. If lives are above 0 the next wave will begin. If the wave is greater than or equal to the total waves then you win and the next level will be loaded. The setTimeout functions are used in a recursive loop with a delay and this delay is increased every wave to trigger the next wave.

Different dinos are added every wave depending on the wave. This loads the dino assets through the GLTFLoader.js file and adds them to the scene and sets the dinos path.

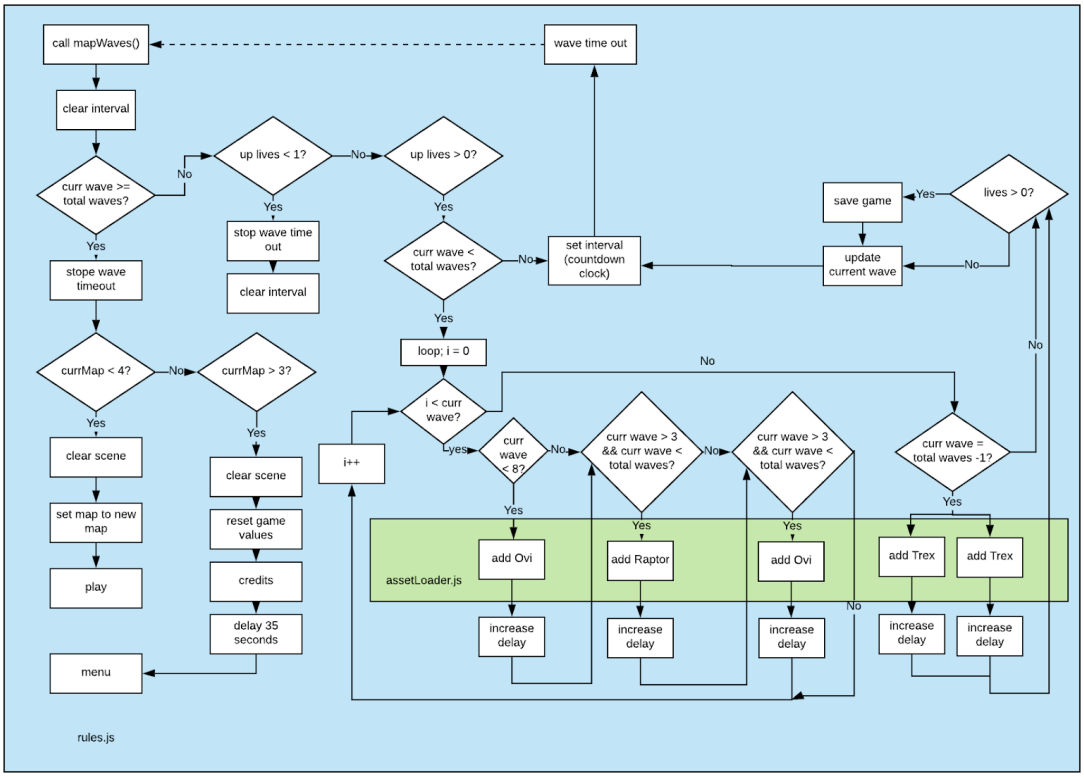


Image: Wave Rules

Tools and Resources

- Languages
  - html/css
  - Javascript
- Software libraries:
  - GLTFLoader.js
  - OrbitControls.js
  - Three.js
  - Tween.js
  - WebGL.js
- Development tools:
  - Atom text editor
  - VS Code text editor
  - Chrome DevTools
  - Sublime Text
- Servers:
  - Flip
  - Amazon Lightsail
  - Node.js
- Other:
  - Blender
  - Git/GitHub

    - Google Hangouts
- Resources:
    - https://threejs.org/docs/index.html#manual/en/introduction/Useful-links
    - https://www.august.com.au/blog/animating-scenes-with-webgl-three-js/
    - https://threejs.org/docs/#manual/en/introduction/Creating-a-scene
    - https://github.com/mrdoob/three.js/blob/master/examples/webgl_geometry_text_shapes.html
    - https://github.com/mrdoob/three.js/blob/master/examples/webgl_interactive_lines.html
    - https://docs.blender.org/manual/ja/dev/addons/io_gltf2.html
    - https://github.com/tweenjs/tween.js/
    - https://stackoverflow.com/questions/24723471/three-js-scale-model-with-scale-set-or-increase-model-size
    - https://stackoverflow.com/questions/30359830/how-do-i-clear-three-js-scene
    - https://github.com/mrdoob/three.js/blob/master/examples/webgl_animation_multiple.html
    - https://github.com/mrdoob/three.js/blob/master/examples/webgl_animation_keyframes.html
    - https://css-tricks.com/snippets/css/star-wars-crawl-text/
    - https://stackoverflow.com/questions/2980143/i-want-to-store-javascript-array-as-a-cookie
    - https://github.com/mrdoob/three.js
    - Sound Effects:
        - soundbible.com
        - www.bensound.com

Creating Models and Animations with Blender

To create models and animations, Blender 2.80 beta was used. This version of Blender was used because of the Khronos Group's built in GLTF exporter. By having this available, we could create a model, add an Armature to the model, and use the Blender Dope Sheet and Action Editor to set keyframes for the animations to follow. By setting it up this way we could create the model and and animations and export in a .glb/.gltf file format that three.js easily accepts.

We quickly moved away from .obj files not only because of the file overhead, but also because of the lack of adding animations. A glb file of the same model AND an animation was roughly a quarter of the size of just an obj file with no material and no animations. This will help with server cost as it keeps the AWS Xfer allotment amount low.

There is a learning curve in modelling/animations and using the new Blender UI. It really does take practice and trial and error to see what works and what doesn't. Modelling and Animating is very much a merge of art and science.

Git/Github

Version control was an important portion of our project and Git and Github were instrumental in organizing how we approached collaborating and making updates to our code. We each would work on individual tasks in branches from our master branch and then about twice a week our team would come together to pull and merge changes to the master branch. Admittedly, version control (and git) can be very tricky, but GIT aided our progress more than it scared us. We had to communicate extensively about changes to branches, and were able to troubleshoot and debug collectively through Github GUI. It was very beneficial to see what changes occurred through commit history, and when changes were made, so we could revert back to successful branches during testing and debugging.

Team Contributions

Between the three of us, we created a game none of us could have imagined at the beginning of the term. We truly tested ourselves and made something we hope you find enjoyable

Christopher Frenchi

Christopher worked primarily on the game engine and the rules. He also created models, animations, and rendering in Blender and exported into the three.js scenes. He created various functions in the assetLoader to apply the correct logic to the models when loaded graphically into the scenes. Worked on setting tower radius and created the projectiles system. Worked on refactoring and ensuring code worked between merges. Worked on the loading of maps and variable refreshes on level changes. He setup the AWS Lightsail instance and kept the project up to date on the website. Added double touch input to the controls so that the game can be played on mobile devices. He worked to find and squash bugs.

Jonathan Ruby

Jonathan worked on creating the initial map, tower and dinosaur files containing their attributes. He also worked on allowing the initial models to be loaded into the game with obj and mtl loaders for testing. Jonathan also researched and started to use the tween.js library for movement of the dinosaurs down the path. He created the first map and path for the dinosaurs to follow, and made the function that could convert the path array to an array that matched the game board. He created the first test waves of dinosaurs. He also figured out how to the dinosaurs to turn and face the correct direction when going down the path. Jonathan created the health bar above the dinosaurs. Jonathan researched open source and royalty free music and sounds, and added the game music and sounds to the program. Jonathan also did a lot of manual testing of the game and worked on fixing bugs that were found. He also worked on balancing the attributes of the dinosaurs and towers so that the game would be difficult enough, but still winnable and fun to to play.

Ruben Torres

Ruben worked on user interface components such as the Build Tower Bar (build-ui.js) and Status Bar (game-ui.js) so that dynamically created html elements could be shown to the user. Worked to ensure that user stats were updated to indicate coins, lives, waves, and current level. To provide a richer experience, he also worked on the css files associated with build-ui and game-ui. He worked on the interface between the main script and build-ui to allow the user to build a tower on buildable space; this allowed the build bar to appear when needed, and only when appropriate. He also worked on saving (save.js) and loading (load.js) game instance materials through cookie setting and getting. This required collaborating on the menu() and play() functions within the main script to set all the appropriate variables and to only load as appropriate based on the user selections. Additionally, Ruben worked to create a credits screen (credits.js) to be shown as the user completes the entire game. Ruben also worked on extensive unit and regression testing.

Development Changes

We primarily intended to use three.js to create and manage our user interface (such as the Status Bar and Build Tower Bar), but since most of these text elements needed to change regularly, it was more effective

to create and update them using using html/css. It cut down on the extensive code required to create a text object using the three.js library.

Another minimal change from our initial plan was the reduction in the contrasting environments between each map. Originally we expected to have different themes; such as a volcano, the beach, etc, however, given how time consuming it was to render objects using Blender, it was more important for us to focus on the mechanics and flow of the game. Our game now is primarily based on the dinosaur infested island where the environment assets rearrange to form the various levels.

Conclusion

We went into this project expecting it to push us out of our comfort zone; it did not disappoint! It pushed us to gain a greater understanding of javascript, not only for gaming and 3D programming, but overall. Between the single-threaded nature of javascript, and the joy that is asynchronous programming, what's not to love? Besides cementing our love-hate relationship with javascript (and admittedly with git/github as well), this project provided us an opportunity to come together and build a fun, quirky, little game that demonstrates the principles and techniques we've studied at OSU.

We've had a great experience this term and we hope you enjoy our game.