# Portfolio 3

## Boxxy McShooterFace

2-D Browser game that doesn’t require server connection. Jump the white box to the black zone while avoiding the evil boxes and their projectiles.

**COM S 319**

MICHAEL WEEMS & ZACH WILD

## Contents

## Overview

In this portfolio, we wanted to create a 2-D 'platforming' game that did not require you to connect to a server to play. In this game, users try to move a white square the end of a preset 'level' (black square) using keyboard controls. Obstacles will present themselves to the player, such as red pitfalls that teleport you back to the beginning of the level. To make the game 'competetive', we included a timer to track how quickly you finish each level.

Our game was created with scalability and new features in mind. This game includes two complete levels, showing that we can scale our game into many more levels easily. By utilizing Node.js, we can easily create and include more modules into the game to allow for more features.

Our application uses the core concepts we learned in class with the addition of a few new libraries. Beefy helped tremendously with the development of this game. By bundling the modules we created into one bundle.js file on the fly, we reduced the testing and running time of our application significantly. CRTRDG was the basis of our entire project. Using the CRTRDG modules in tandem with our modules, we were able to create a fully functioning 2-D 'platforming' game.

Focusing on the higher levels of Bloom's Taxonomy, we determined exactly how we wanted to create this portfolio. We created the idea of a 2-D game 'platforming' game with obstacles and utilized CRTRDG to run a lot of the control and event handling. We evaluated making our game multi-level, whereas we lost in pushing our knowledge of CRTRDG to the fullest, we gained in making our game much simpler to scale up. Our analysis of the game breaks down the various components of it and how they interact. We utilize the CRTRDG modules to handle the entity events such as the game loop and physics. The modules we created create all of the physical elements that interact with the handlers.

After spending a ton of time on this portfolio, we learned a lot about how to utilize new libraries into Node.JS. These include optimizing workflow using Beefy and exploring open-source libraries to find new possibilities.

## New Things

### Beefy

While exploring the idea of creating a game with Javascript, we stumbled across Beefy. Beefy is an open-source plugin for Node.js that allows you to create one controller javascript file to 'bundle' all of your modules together. By simply running this command in the terminal:

```
beefy game.js:bundle.js --live
```

beefy will look at the game.js and auto-generate the file 'bundle.js' in your browser. You can now view your application on port 9966. Any updates you make to game.js while this command is running in the terminal will automatically compile and update in your browser. This allows for instantaneous testing and running of the application, significantly speeding up development time.

### CRTRDG

After investigating including new libraries for web apps, it became clear that CRTRDG was the library of choice for us. Utilizing CRTRDG, we were able to create our 2-D ‘platforming’ game to match our goals. These goals included scalability, modular features, and a physics engine to handle player movement. CRTRDG supplied a simple to use game-loop that would use Node.js to update the game canvas, while also allowing us to attach our own nodes to the game loop. Attaching our own nodes to the game loop allowed the game-loop to recognize the many modules interacting together, which enabled our physics engine and collision detection mechanisms to fire appropriately. CRTRDG really helped us to create the project to our vision.

## Complex Issues

### Multiple Levels

Initially we thought that adding multiple levels to our game would be easy. However, after digging into CRTRDG, it became clear that each game-canvas is it’s own entity, and CRTRDG does not provide a way to remove a game-canvas or game-loop from the page once it is placed. This made switching out one loop for another very challenging. We looked into creating various states of the game-loop, but ultimately decided completely destroying the game-loop in place to make way for the new game-loop. This new game-loop would include a new level. Making the game loop null did not destroy it, however, and stumbling around blindly, we discovered that we could in a way pause the game when we ‘destroyed’ it. This would allow the new game-loop to be added, while the other loops wait for the newer loop to be paused.

### Collision Detection

Counter intuitive to what one may think the most effective collision detection would be handled, instead of ensuring an entity always stayed outside of another entity, we checked to see if an entity overlapped with another entity in 2-D space. If this happened, we would set its position to be adjacent to the other entity relative to the position of the other.

### Projectiles

Utilizing techniques in collision detection, we were able to make entities disappear if they collided with the projectiles we fired from another entity. We ran into issues when we had too many projectiles on the screen, and discovered that the application could not process the sheer volume of projectiles all at once. We reduced the amount of projectiles that could be on screen at once, and this solved our problems.

## Configurable Levels

To speed up future development time, we created an object containing the configurations for each level in the game that could be easily modified. Using this object in conjecture with the CRTRDG modules was trickier than we thought it would be, as CRTRDG entities like to be hardcoded and not dynamically created. However, we were able to dynamically create new levels using this configuration by looping through each object in the module, checking to see if the module existed in the object, and making it as modular as possible.

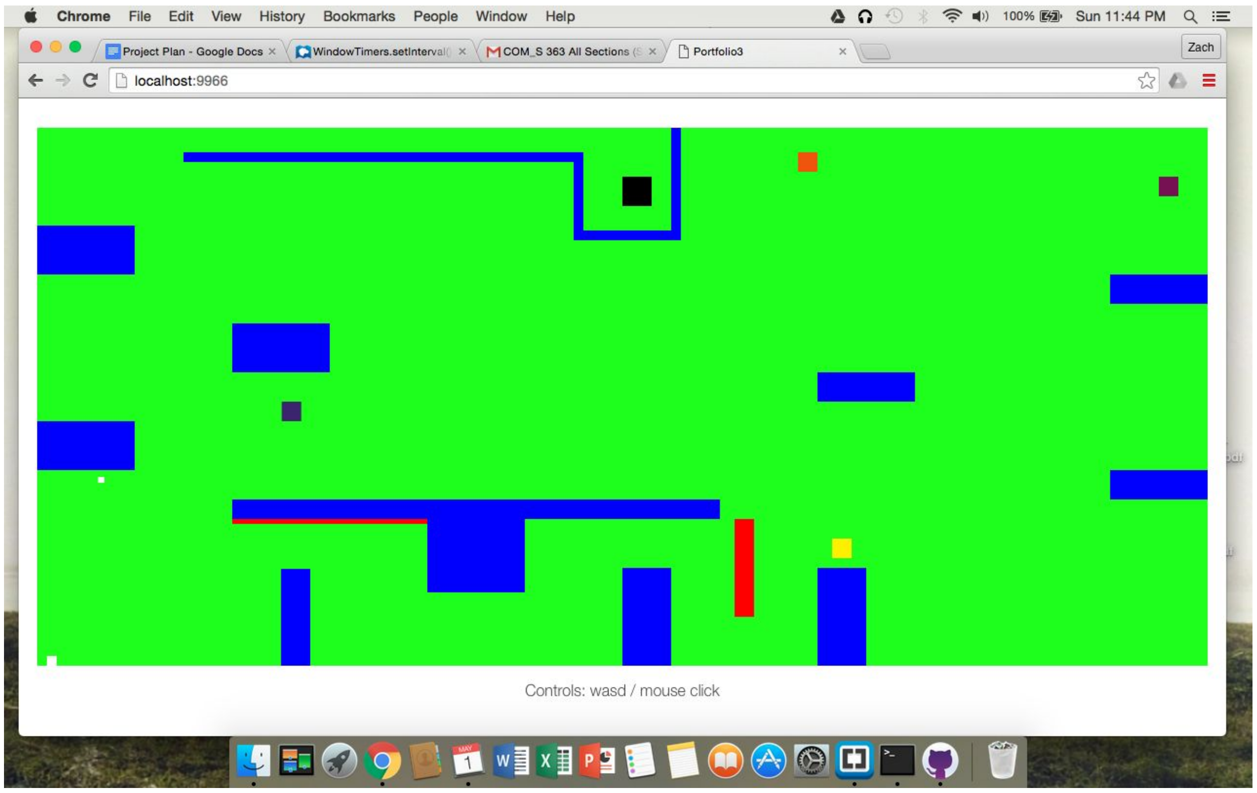
## Level Design

To account for the many potential bugs that could come from creating our own physics engine, we designed one of the levels to heavily restrict the player from making decisions that would break the ‘rules’ established by the game. Things like clipping through walls happen occasionally, so adding pitfalls in trouble areas as a kind of band-aid allowed us to better mask the collision detection shortcomings.

## Functionality

In order to run this application you can simply open the index.html file in the root directory. If you wish to install the dependencies yourself, simply run ‘npm install’ in the root directory. As Beefy has handled bundling the modules we have created already, the only file you should need along with ‘index.html’ is ‘bundle.js’.

### Level 1


Chrome File Edit View History Bookmarks People Window Help
Sun 11:44 PM
Project Plan - Google Docs
WindowTimers.setInterval
COM_S 363 All Sections
Portfolio3
Zach
localhost:9966
Controls: wasd / mouse click

Upon starting the game, you will be presented with Level 1. To complete the level, you must travel your small white square to the black square. The controls include:

A -> move left

D -> move right

W -> jump

Left click on mouse -> fire projectile (only after picking up maroon powerup)

1 -> switch to level 1

2 -> switch to level 2

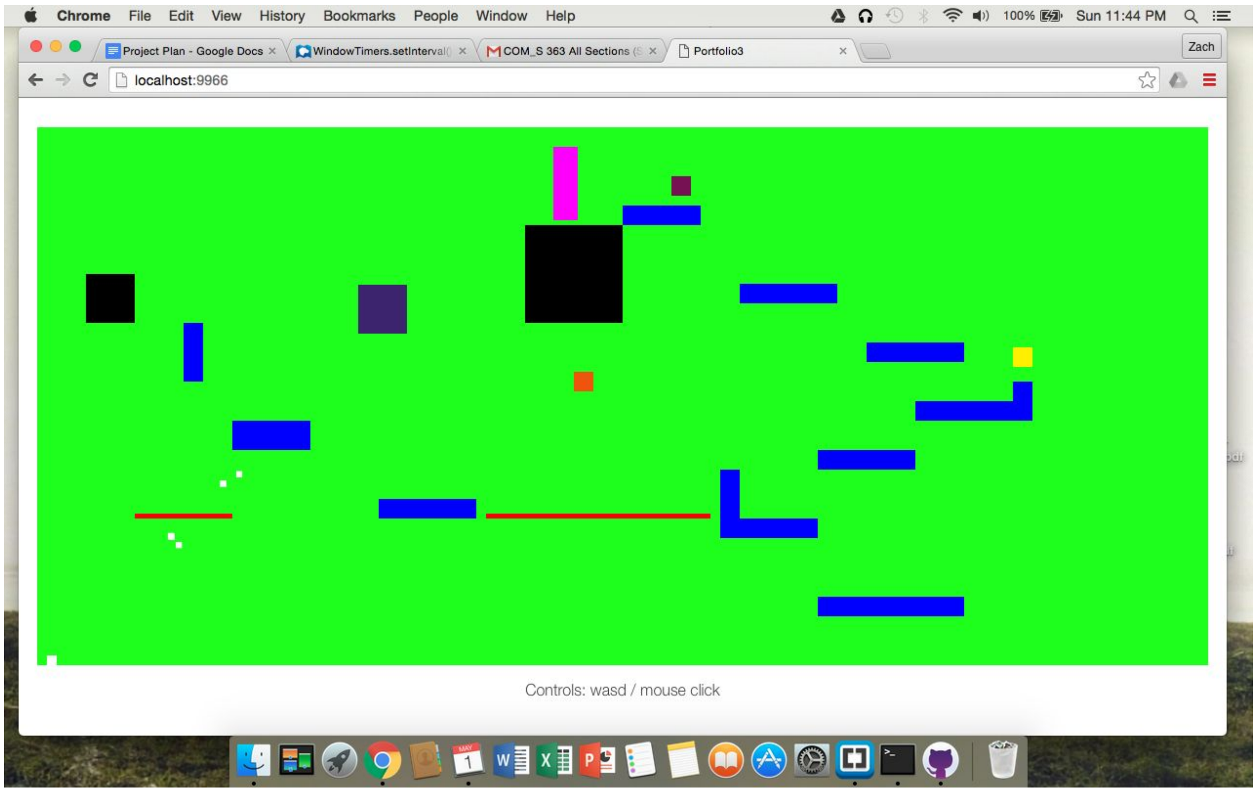
You may utilize the blue platforms to maneuver the level, as they offer full collision detection allowing you to jump on top of them. Red areas will teleport your character back to the last checkpoint, so avoid them if you wish to continue further. Floating boxes will fire projectiles in your direction which will teleport you back to the last checkpoint if they hit you. Also, colliding with these floating boxes will teleport similarly.

There are two collectible power-ups located in level 1 and level 2. The Yellow power-up serves as a checkpoint, allowing you to teleport back to the place it was picked up if your white square is teleported. The Maroon power-up allows your white square to fire projectiles. These projectiles can be used to eliminate enemies from the field, allowing for easier passage through the level. Left clicking with your mouse on the game canvas will fire a projectile in a line from your white square to the clicked point.

While you are progressing through level 1, various messages will display under the game canvas to alert you of various events, such as you being teleported, or picking up the projectile power-up. Along with this, there is a time counter to show you how long you have spent completing this level.

Once you have reached the end of level 1, the game will freeze. Above the game canvas, a victory message will be displayed, and below the canvas the final time it took you to complete the level is displayed. Once you are done here, continue to level 2 by pressing the number 2 on your keyboard.

## Level 2



Level 2 functions similarly to level 1, however it introduces a 'Boss' mechanic. The pink boss floats sporadically, making it difficult to predict where it will be. Also, the boss will not be eliminated by one projectile. It will take 30 projectiles to take it's 'health' down to 0, at which point it will disappear from the level. Be careful as you progress through the level, as it may interfere with your path, requiring you to either wait for it to move somewhere else or to shoot it.

## Bloom’s Taxonomy

### Analysis

Bloom’s Taxonomy dictates that the skill of analysis has the creator split up the project into component parts and analyze the interfaces and relationships between them. This project uses components of JavaScript, HTML, JSON, CSS, and Node JS.  The HTML page shows the contents of the created game, which has the JavaScript handle the mechanics of the game.  We imported CRTRDG using Node JS and npm install.  CSS styles the html, and the JSON is used for the npm install to create the dependencies for crtrdg.

### Evaluation

As we evaluated the decisions we made for this project, we determined exactly why we built certain aspects of the project as we did. Utilizing the CRTRDG Game-Loop to bind event handlers to dynamic game entities allowed us to build the game in any way we wished. This led us to creating many separate modules to be controlled and handled by the game loop. The entities module of CRTRDG gave us the ability to recognize the many animated parts of our game and handle any events based on collisions and physics. The keyboard and mouse modules allowed us to create the movement, level switching, and projectile firing controls for the game. The modules we created heavily relied upon the CRTRDG Entity module to handle checking statuses and updating animations.

### Creation

In creating this portfolio, we had to review what we already knew from the labs leading up to it. The knowledge gained from these labs includes javascript, json, and html, and Node.js,  all interacting with each other. Creating something new from this knowledge led us to explore new libraries to add, and explore deep into new methods of interaction between these tools. As we wanted to bring a Facebook style experience to a user with our application, we had to see how we could apply what we have learned already. In creating something new, we had to expand our knowledge into new territory with Materialize, Vexflow, and various new jquery interactions. Materialize allowed us to animate much of our web pages, and allowed for page elements to transition in shape after integrating with javascript, jquery, and css. The addition of Vexflow allowed us to explore canvas elements in javascript, and how we can set up easy to use forms for customization options. Utilizing timers to make a looped call to ajax was a new idea we came up with to allow for dynamic music changing in the sight reading app.

While creating this portfolio, we used our prior knowledge gained from the previous labs by using Node JS to run socket.io. Here, we used Node to install crtrdg and ‘beefy’, which we used to create this game. CRTRDG drew a canvas and update on an interval for us, while also redrawing the progressions of entities in the game. This also allowed us to implement interactions using the keyboard and mouse creating a fully functioning game that runs like a pc game would. This also required us to go the extra mile to discover the functionality of CRTRDG since we have not worked with this module before, and in return put our own spin on it to make a unique game that is fun and enjoyable experience for the player.  Beefy, on the other hand, encouraged us to create the game, as it made development significantly easier. By running a single command in the terminal, we were left to simply write our code and see immediate results in the browser. With the knowledge we gained from the labs leading up to this portfolio, we felt comfortable exploring new modules in Node.js, and felt confident in pursuing the creation of a video game.

## Conclusion

In conclusion, the game that we created for this portfolio proved to be challenge as well as a success. In creating a controllable player that could interact with the environment and the AI created to act as an obstacle, we were able to create a truly fun (at least to us!) experience for the player. This project also demonstrates our knowledge of installing node modules using Node JS to make our own creations.