**Portfolio 1**

**ComS 319 – Fall 2016**

**Evan Kammermeier and Nick Lewis**

Table of Contents

[Overview 3](#_Toc462796951)

[New and Complex Section 4](#_Toc462796952)

[**Part 1: JQuery** 4](#_Toc462796953)

[**Part 2: Objects** 5](#_Toc462796954)

[**Part 3: Canvas** 5](#_Toc462796955)

[Bloom’s Taxonomy 8](#_Toc462796956)

[Analysis 8](#_Toc462796957)

[Evaluation 8](#_Toc462796958)

[Creation 8](#_Toc462796959)

# Overview

In this project, we aimed to learn the basics of JQuery, object-oriented javascript, as well as familiarize ourselves with html canvas. Unfortunately, it also involved some amount of mathematics. These new elements, when combined with the material that was discussed in lectures and the lab activities, were used to produce a complete game for the first portfolio. We have made a single player tank shooter game with JavaScript and HTML that is played in a web browser. The player controls a tank and shoots with varying power at moving targets to try to score as many points in the time limit as possible. The player is able to move the tank left and right across the screen using the left and right arrow keys and is able to rotate the tank’s turret counter-clockwise or clockwise using the up and down arrow keys. The player controls firing with the spacebar. The longer the spacebar is held down, the more powerful the shot will be when the spacebar is released.

This game taught us a lot about manipulating images within a canvas and responding to user input. The primary animation of targets sliding across the screen, the tank moving left and right across the screen, and the projectiles flying through the air was accomplished by manipulating an HTML canvas with JavaScript. Much attention was paid to the smoothness and responsiveness of the animations. Because of the nature of this game, animations that appear choppy (not smoothly moving from one point to another in the world) or laggy (not reacting to user inputs on the keyboard sufficiently quickly enough) are not simply aesthetically displeasing. More importantly, sufficiently laggy or choppy animations have the potential to inhibit the player’s ability to control the tank and to line up precise shots. Given this risk, the group took great care to ensure that animations were acceptably smooth and responsive. This included refactoring the way that some of the objects were handling inputs when it was determined that the initial animations were not performing acceptably.

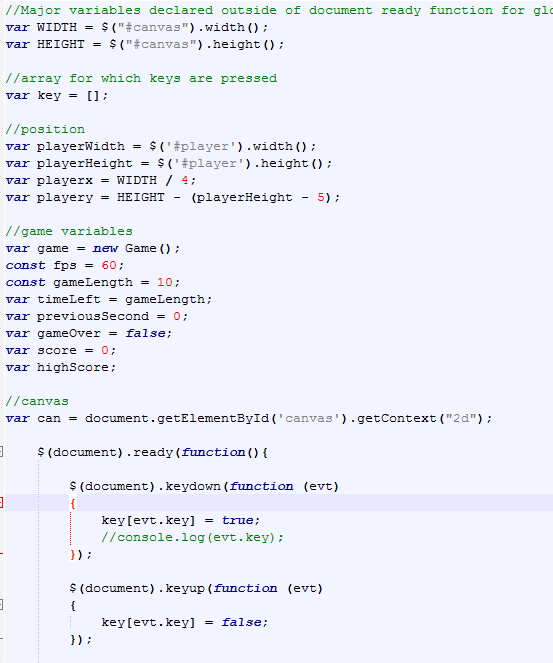
Needing to be able to parse user input, have that input impact one of several different conditions in the game, and modifying the visual output given to the player several tens of times per second required the group to consider the best way to handle keyboard input from the user. For the same reasons listed above in reference to animations, the responsiveness of keyboard input handling is also crucial to the game’s playability. It was with these concerns in mind that the group implemented the ability to parse multiple key inputs at once. Players are able to move the tank side to side while rotating the turret and charging a shot.

In addition to working with new tools and concepts like JQuery and object-oriented JavaScript, the group also had to make deliberate design choices regarding the design and implementation of the game. The group took care to optimize the drawing of animations and the handling of user input so that the game runs as smoothly and responsively as possible, resulting in an enhanced player experience.

# New and Complex Section

## **Part 1: JQuery**

So far, the class has been studying the many features of javascript. My partner and I decided to look into how JQuery could enhance our programming. JQuery proved itself very useful in selecting attributes from the canvas in shorthand. A few examples of where we took advantage of this shorthand would be grabbing the attributes of our images and canvas for global variables, initiating the game using $(document).ready(), and the calls to the keyup() and keydown() functions.



## **Part 2: Objects**

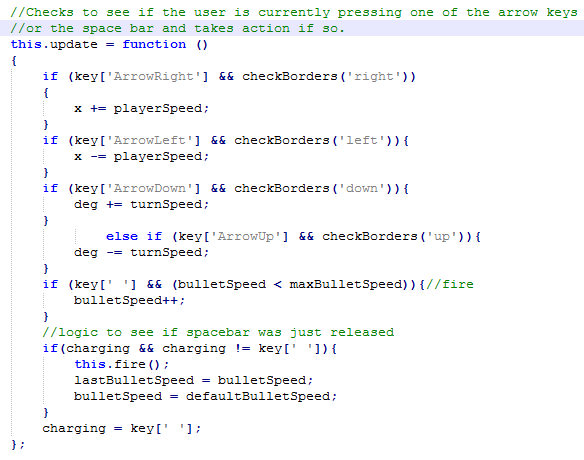
Manipulating many separate objects and taking input at the same time seemed like it would be the hardest part of the game. To combat this, we decided to put all of the objects into one array with a similar function structure. In our ‘loop’, function, we iterate over every object currently in the ‘game’. Every type of object in the game has a draw function to put them on the screen, and an update function to calculate the changes in position or action. Using these methods, the player, every target, and every bullet, can coexist as separate objects without the need for bulky turn orders.

##pic of loop function when finished

## **Part 3: Canvas**

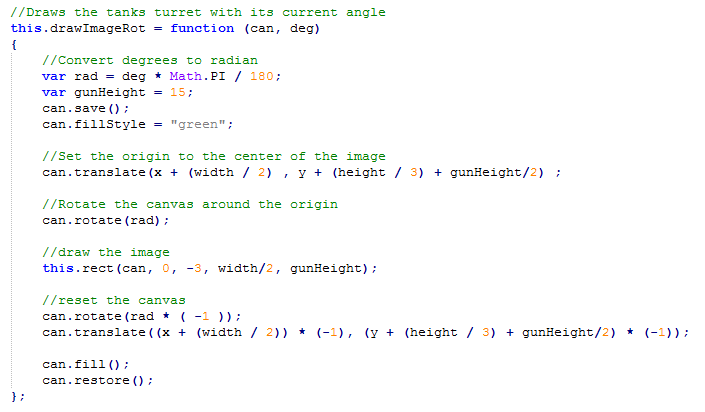
With the line crawler we made in one of our labs, the potential for games became clear. Instead of drawing a line, we investigated the draw image function, and made a bunch of simple images with Microsoft Paint. Once the images were on the screen, we made a ‘update’ function manipulate the x and y coordinates of the image as well as handle the more complicated maneuver of rotating the image to simulate gravity. Every iteration of the loop, the screen is cleared and the objects are redrawn with their new coordinates.

Code to manipulate the player object and its attributes:



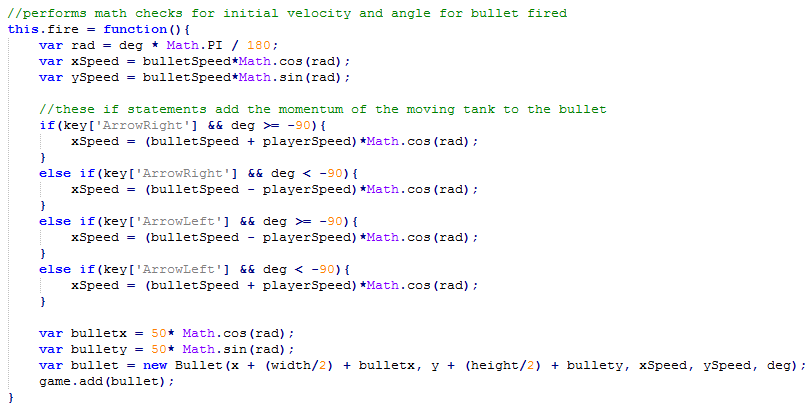
The most complicated part of canvas manipulation was drawing images at an angle. This was necessary to have a rotatable cannon, as well as bullets that follow a curvature path similar to that caused by gravity. The way we learned to do this was by translating the canvas context to the origin of what we planned on manipulating, then rotating the canvas. Once the canvas is rotated the image can be drawn normally. However, since the image was drawn on a rotated canvas it appears to be at an angle on the regular canvas. The save() and restore() methods were very useful in grabbing a snapshot of the original canvas state and restoring it. This is how we made sure that only the objects that needed to be rotated did so.

Code to handle drawing the turret at a specific angle:

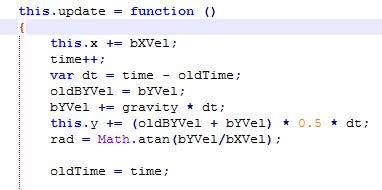


The hardest part of this project was to perfect the curvature of the bullets as their speed vectors changed. After a lot of googling for math help, we came up with a few formulas that calculate initial speed in x and y vectors according to the angle of the turret, and calculate the change in those vectors due to ‘gravity’.

Logic for initial bullet velocity vectors:



Logic for bullet velocity vectors being affected by ‘gravity’:



# Bloom’s Taxonomy

## Analysis

Before beginning the project, we analyzed how it would be possible to efficiently manage many different objects in the same game loop. It became clear that if each object had its own method for updating its view, it would become very challenging to make them interact. After comparing several ideas, we decided to build the objects with a similar display and update architecture that allowed for a simple game loop. Implementing this structure made design easy because while every object behaved differently, their design is overall the same.

## Evaluation

Deciding how to handle user input was a large part of our evaluation. Initially, we argued over whether or not to use a switch statement with cases for each input. A switch statement was very powerful in terms of code cleanliness and simplicity. However, it proved challenging to access multiple inputs at the same time. To combat this, we considered the idea of using if-else statements, but the ability to access variables was the same as the switch statement. We finally settled on all if-statements as this allowed for the loop to run in real time and have multiple fields be accessed on each iteration.

## Creation

* Made pictures from scratch to avoid copyright issues
* Implemented an automatically updating key input array for user input
* Structured user input formulas to allow multiple key interactions at the same time
* Created a unified object display structure for simpler looping
* Performed vector calculations to determine angle and vector speeds of bullets
* Utilized JQuery to simplify code usage
* Manipulated HTML Canvas to display multiple objects and rotate them if necessary
* Used localStorage to allow for persistent high scores
* Calculate object collision based on x, y, and size variables
* Store all game objects in a single array for better access, creation, and removal