**Portfolio 1**

**ComS 319 – Fall 2016**

**Evan Kammermeier and Nick Lewis**

Table of Contents

[Overview 3](#_Toc462783240)

[New and Complex Section 4](#_Toc462783241)

[**Part 1: Jquery** 4](#_Toc462783242)

[**Part 2: Objects** 5](#_Toc462783243)

[Bloom’s Taxonomy 5](#_Toc462783244)

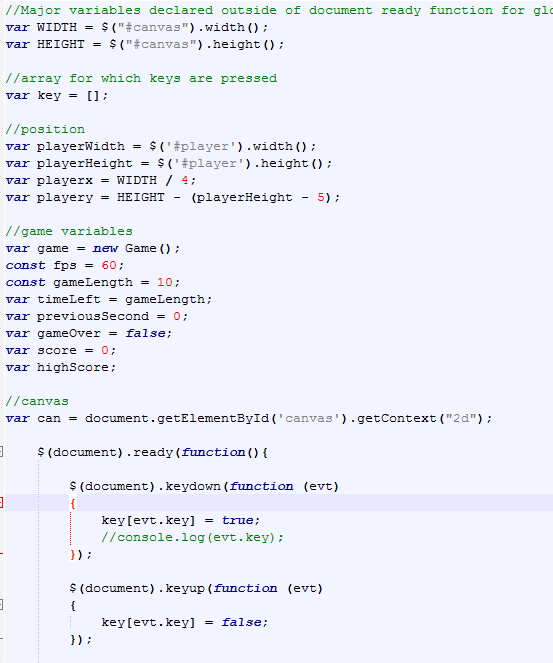
# Overview

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 math. We have made a single player tank shooter game where the player controls a tank and shoots with varying power at moving targets for points. This game taught us a lot about manipulating images within a canvas and responding to user input.

# 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**

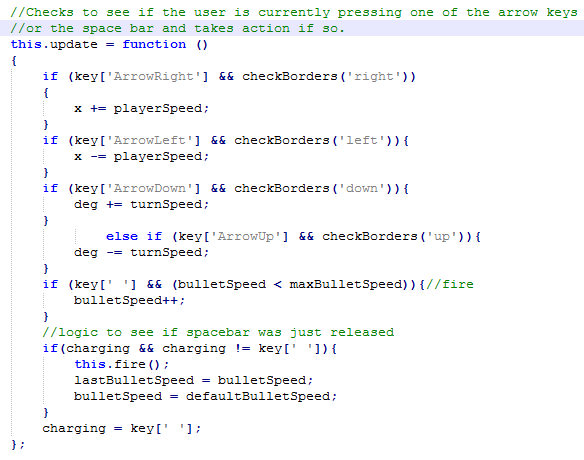
Manipulating many separate objects and taking input at the same time seemed like it would be the hardest part of the game. To combat this, we decided to put all of the objects into one array with a similar function structure. In our ‘loop’, function, we iterate over every object currently in the ‘game’. 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**

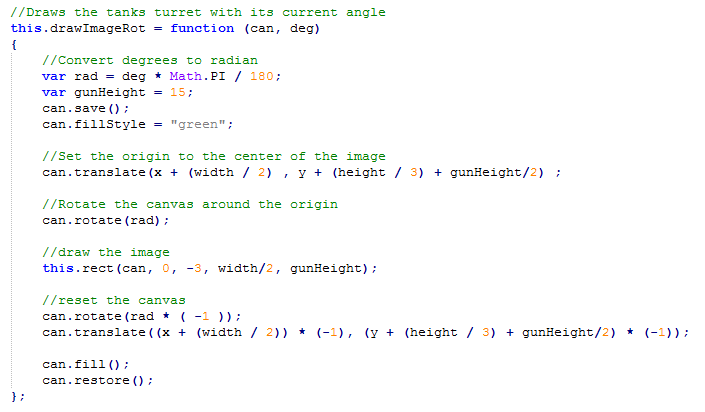
With the line crawler we made in one of our labs, the potential for games became clear. Instead of drawing a line, we investigated the draw image function, and made a bunch of simple images with Microsoft Paint. Once the images were on the screen, we made a ‘update’ function manipulate the x and y coordinates 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:



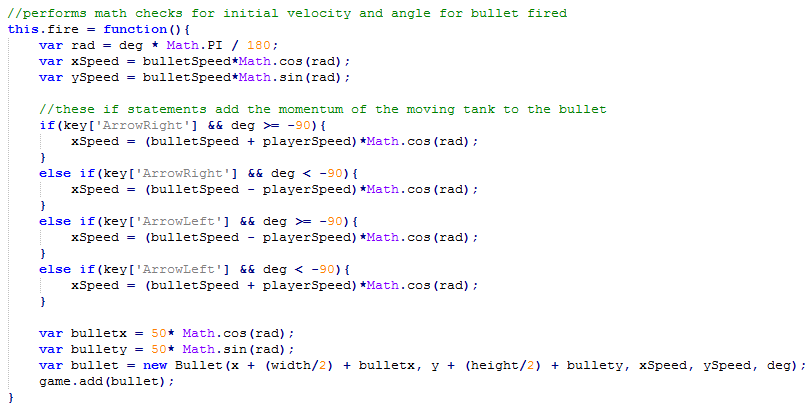
The most complicated part of canvas manipulation was drawing images at an angle. This was necessary to have a rotateable cannon, as well as bullets that follow a curvature path similar to that caused by gravity. The way we learned to do this was by translating the canvas context to the origin of what 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:

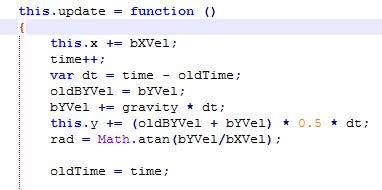


The hardest part of this project was to perfect the curvature of the bullets as their speed vectors changed. After a lot of googling for math help, we came up with a few formulas that calculate initial speed in x and y vectors according to the angle of the turret, and calculate the change in those vectors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

Logic for initial bullet velocity vectors:



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



# Bloom’s Taxonomy