# CS 3308 Tanks - Post Mortem

Rosetta Roberts

December 15, 2018

So this project was fairly fun, though I usually don't particularly enjoy doing graphical stuff. Some of the things I'm particularly proud of for this project include the random rock generation, collision detection, and the missiles I added to the game.

## 1 Features Omitted

Originally, the AI system for this game was planned to be far more extensive and complicated. However, I ended up writing very little AI code. The reason for this is because a simple AI that drives straight towards the player is actually a fairly decent AI. To make up for the lack of different tanks having different difficulties, I made it so that as time progresses, tanks spawn more and more quickly. Eventually, the game becomes very challenging even with the overpowered ability of the player to shoot missiles.

Another feature omitted was a leaderboard. Instead, the main menu displays the best score since starting the program and the score of the last game. The reason for doing this is because I figured my project was good enough at this point.

Another feature omitted from the original specification was the ability to control each track separately. The reason for this change is because I have never played a game with that mechanic. I instead replaced it with using the WASD keys to move and rotate the tank. This is more intuitive because most people would be more used to these controls.

## 2 Unique Features

Perhaps the feature I'm most proud of in this project is the random rock generation. The rocks vary in smoothness and size. They also include concave rocks. Unsurprisingly, this also made collision detection more difficult. To create a rock, first the smoothness and size are randomly generated. The size parameter is then used to create a circle. 20 points are selected on the circumference of this circle and randomly moved. How far they are randomly moved depends on the smoothness. Finally, each pair of points is averaged. This has the effect of

smoothing sharp corners and makes it look much more rock-like. For collision detection, objects check for a collision with every edge of the rock.

Another unique feature is the ability of the player to fire missiles. These missiles will instantly kill enemy tanks. They are fired by the player when they click on the screen and are fired at the enemy closest to the player's mouse, which is indicated by a circle and cross-hairs around the enemy. The missiles automatically aim for the enemy. This might seem overpowered, but the game quickly becomes quite difficult with the way the number of enemies increases over time. Also, the code to aim the missiles as perfectly as possible was particularly difficult to make but should be optimal.

Another nice feature I added was little life bars beneath each tank.

# 3 Difficulties

Perhaps the biggest difficulty I had in this project was actually getting keyboard input to work correctly. Originally, I used the arrow keys for controlling the tank. C# uses a bitset to denote keys on the keyboard. Unfortunately, the definitions for the arrow keys overlap somewhat. So using a switch statement on key types fails to work. Another problem is that if both the left and up arrow keys are pressed, Windows Forms will ignore the spacebar being pressed. This problem eventually caused me to change from using the arrow keys to the WASD keys. Finally, at one point I refactored the Game display into it's own Windows Forms control. After I did this, C# stopped calling my event handlers for arrow keys. Eventually I found that it was because Windows Forms only passes common keys to event handler of custom controls by default. It took me a long time to figure that out. Of course, if I had switched to WASD beforehand I would not have had that problem.

Another difficult was getting my missiles to aim perfectly. For the longest time, my missiles would overshoot the tanks they were aiming at, start slowing down before reaching their target, or make unexpected changes in direction. As part of how the missiles work, they solve for the smallest positive root of a quartic polynomial. I initially figured that my code must accidentally not get the smallest positive root but rather a different root. So I switched to a math library for finding the root for me. However, the missile kept doing stuff. Eventually, I redid my equations a couple of times and each time it worked better and better. But the missiles kept overshooting and my equation was supposed to handle it perfectly. I found a hack that fixed the problem most of the time. But the math for the hack made no sense at all. Eventually, I found a problem in a completely different part of the code. Fixing this problem and reverting my hack made the missiles aim perfectly as expected.

In the end, this project was fairly fun to make. I wish I had the opportunities to use the data structures and algorithms presented in this class though.