Introduction

For this practical I was tasked to create a variant of the artillery game in processing.

My variant is called “Sawmill Showdown” and is a lumberjack themed variant of the artillery game. The tanks are replaced with lumberjack sprites and the shell is replaced with an axe. The terrain blocks are replaced with a log texture. Otherwise, the gameplay is the same as the traditional artillery game.

My game features a local 2 player mode where both players take turn controlling their character on the keyboard, as well as a 1 player versus AI game mode when a solo player can take on the AI I created that performs player 2’s moves automatically.

Player’s Guide

The specification asked for a players guide to the game, but I believe the following compilation instructions, screenshots, as well as the controls and rules given in game are more than enough to explain how to play.

Compilation

The folder containing all .pde source files as well as the textures folder is needed to run the game.

To run my game open the “SawmillShowdown.pde” source file in Processing 4 and click run in the top left.

Screenshots

These screenshots that show my game in operation and illustrate its features.

Figure 1: Main Menu

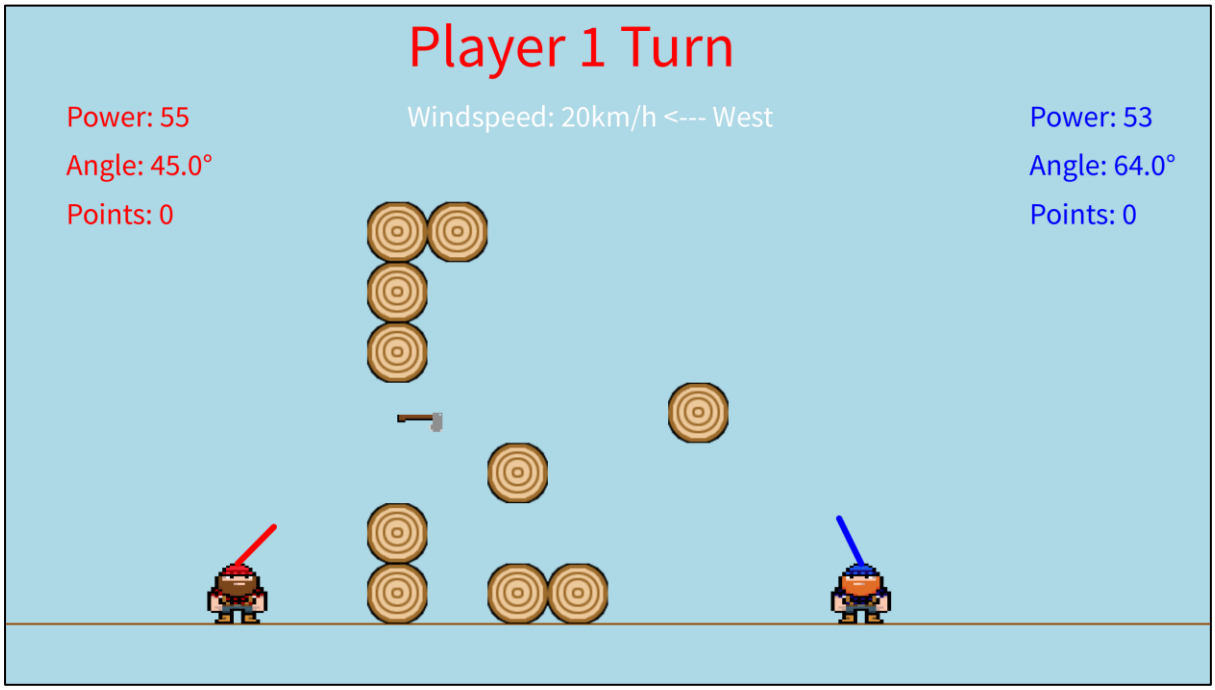

Figure 2: Round 1 of game

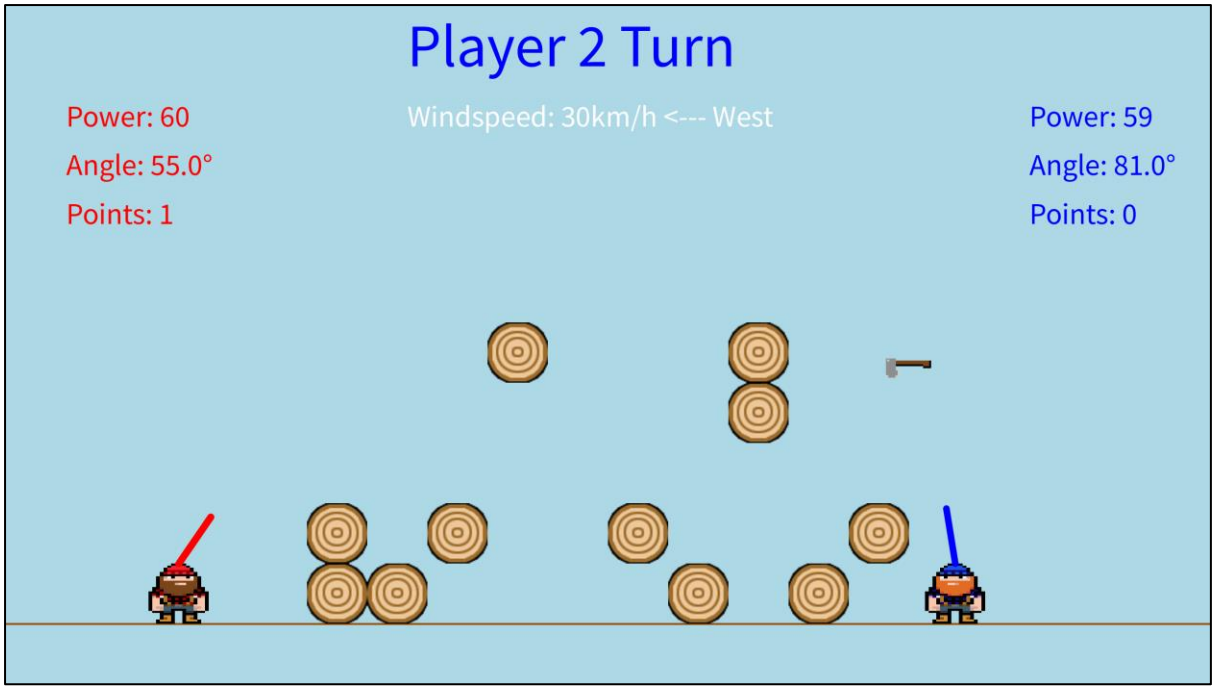

Figure 3: Round 2 of game

This screenshot shows how the terrain generation and windspeed can change between rounds.

Figure 4: End Screen showing winning player

Design

Physics

For my forces I have a force registry class which holds a map of force generators and the particles registered to them. In this game the only particle is the axe.

My force generators are gravity and wind. When a new axe is created, it is registered against these force generators, and then when the axe's position is updated, I iterate over the map of generators and apply each one to the particle in turn. Then when the axe is destroyed, I can deregister it from the force generators.

Gravity is simple as it is just a vertical force. I found a Y value of 200 to feel the most realistic.

The wind is a horizontal force, and I found a range between -50 and +50 to work well as it gave enough variation between rounds without it becoming too strong. I chose to represent these values on screen as a positive integer of km/h as well as the direction. (East for positive and West for negative). I found 2 to be a good dampening factor as well.

I chose to randomise the strength and direction of wind each round, a new round being every time a player is hit with an axe and the other scores a point. This kept it interesting without changing it too frequently to be annoying, this way allows the player to get used to adjusting to one wind speed for subsequent throws.

Dimensions

Using a player width proportion and a player increment proportion as well as the dimensions of the screen size currently being played on, I can determine relative sizes for the height, width and movement increment of the players. This is important as the width of the players is mainly used to size and scale all other text and objects on the screen.

Doing this means my game isn't restricted to one screen size and can instead dynamically adjust sizes of all its elements to keep its relative proportions and run full screen on a wide range of screen sizes and shapes.

This screenshot of my game with annotations in green demonstrates how using only the displayWidth and calculated playersWidth, all other elements can be structured.

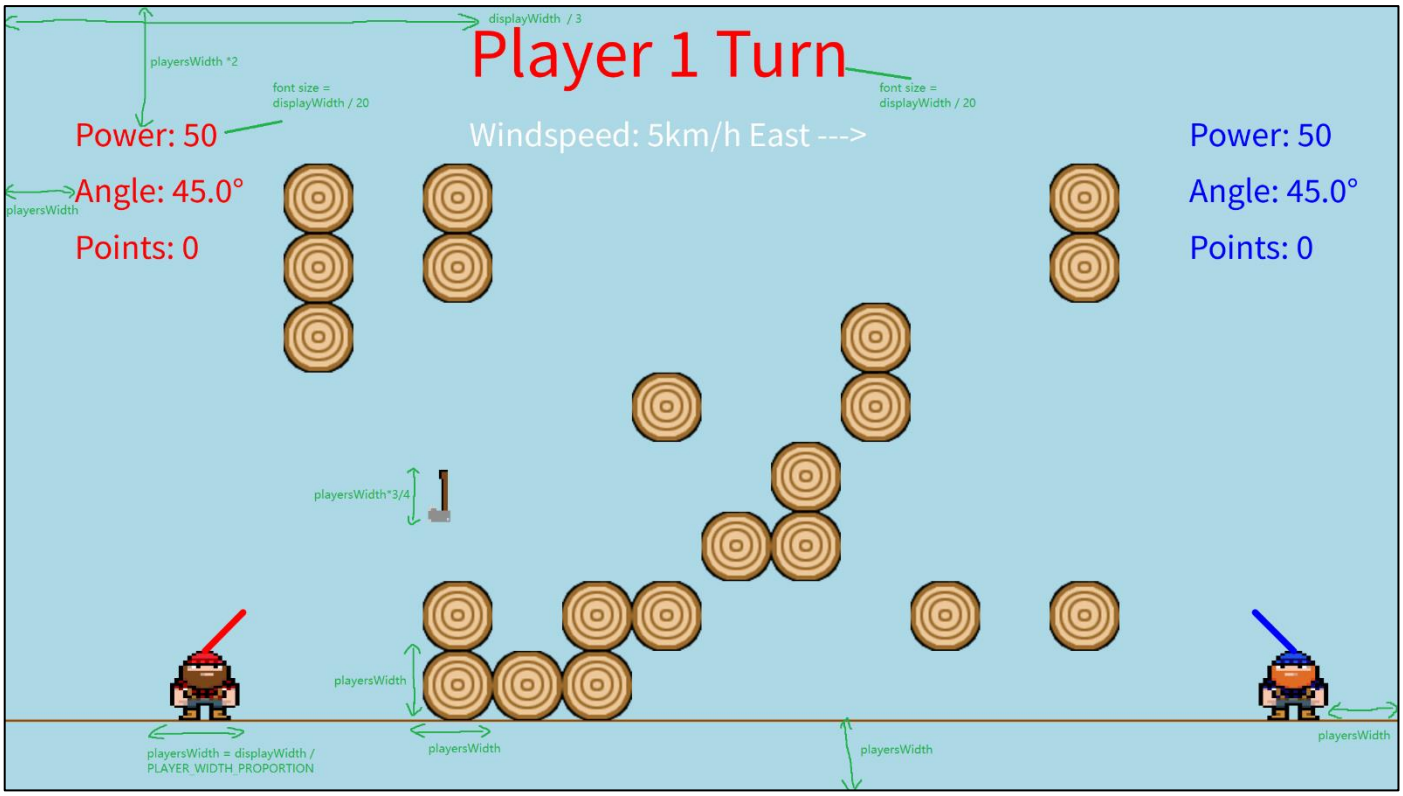

Figure 5: Annotated Sketch

Game Logic

My game starts with a menu screen that displays the controls the player needs to know and then the rules of how a game works. Then there are two buttons for the two different playing modes the game offers.

I set the score limit to 3 to win a game.

The current players turn and both players current points are clearly displayed. I used the red and blue colours to help with natural association.

Once a player reaches 3 points, the game returns to the main menu and displays the winner, allowing the user to play more games if they wish, and to change mode.

Controls

For my controls I chose to use the arrow keys as the keys to modify the angle and power of the throw, as it felt natural and convenient having these two controls in the same area.

That leaves the left hand for the other controls, so I chose A and D to move the character and the spacebar to throw an axe, as these are traditional key bindings within other PC games.

When changing values for power and angle I made the increment be 5 as 1 was too slow, this also gives a nice angle “snapping” feel to the rotation of the throw indicator.

Collisions

When detecting collisions between either the axe and player, axe and terrain or player and terrain, I use Rectangle2D boxes that can be drawn to the dimensions of either the player or axe, and then the .interesects() method can be called to see if the rectangle is intersecting another player or a piece of the terrain.

I have playerCollision and terrainCollision methods to distinguish between the two and handle the games logic accordingly. For the terrain collision, I have to iterate through the array list of all blocks currently in the terrain to check if they are intersecting my player or axe. I think this is quite resource intensive and a more efficient method is worth looking into.

I also have a bounds checking method for the axe, to check if it goes too high or too low and then the method will despawn it accordingly. I set the upper limit to be the height of the displaySize again upon leaving the top, and the lower limit to be the ground the players stand on.

Terrain

My terrain class holds an array list of all currently drawn blocks on the screen. The blocks are drawn as the log texture. When initialising the terrain, the width and height of the blocks is the same as the players, which allows for 12 columns of potential blocks. For each of these columns the total potential height in each column is randomly decided between 0 and 8. At this point every block in the potential 12x8 maximum matrix has a 1/3 chance of spawning and being added to the array list.

Generating my terrain in this random manner makes for new and unique terrain generation every round which is not only visually appealing, but also alters the gameplay as players can move under and hide behind blocks.

Theme/Textures

I chose to go for a lumberjack theme as I thought it would be more visually appealing and interesting to play than blank rectangles. I find the rotating axe to be especially engaging. For the rotation I had to save 8 versions of the axe, 4 for each rotation for each player. Then in the draw method I switch between these images every 5 passes or frames to create the appearance of the axe being animated as it flies.

For the main menu background I acquired the image copyright free from https://moonempire.itch.io/free-pixel-art-nature-platformer-tileset

All other textures were hand-drawn in Photoshop CS6.

AI

My AI class has access to both current players as well as the current wind information. It works by first moving left and then right a random number of times, but that randomness is more heavily waited on the left side as the AI is the player on the right. This means that the AI sometimes moves right but most of the time moves left towards the middle of the game, which is my desired effect.

Once it has moved it gets the human player's values for power that they just used and either adds a random amount of power up to 20 if the wind is blowing against the AI, or removes a random amount of power up to 20 if the wind is blowing with the AI.

Finally the angle is again copied from the human player's last turn and then up to 30 degrees of angle is randomly either added on or removed.

This AI is fairly simple but isn't completely random either, and it does take into account the current wind direction in an appropriate way. It isn't super competitive, but it can definitely score some points against a human player. If I had more time, I would flesh my AI out to conduct simulations of potential axe throws to determine the best one to take for that turn.

Specification Design Question

The specification states:

"For at least one element of your game, compare how different design choices affect the game for the players. Justify the choice you have made for the submitted version of your game."

I believe the most important design choice relating to this question is the generation of the terrain. How the terrain is generated and behaves makes a big outcome in the experience of the game for the player. For my design choices I chose not to make the blocks follow gravity as this way players could travel underneath them and use them more effectively as cover. I also chose to randomise where blocks would spawn in quite a tall and wide area as I wanted them to actually serve a purpose in the game instead of the players just shooting over them.

Another element of design choice that affects the game for the players is the appearance of the coloured line indicating the rotation and power of a shot. As the line gets longer and thicker as the power increases, as well as rotating towards the angle of throw, it gives an easy to visualise and remember representation of the current power and angle. This is preferrable to having to read and remember values between rounds, but I still display these values to offer more fine-tuned control.

Conclusion

I found my variant of the artillery game in processing to be successful and not only meet the basic requirements but also expand upon them, with new textures and animations.

However there are improvements I could make given more time. If I did have more time, I would mainly improve the AI to make it more competitive and accurate, an idea I had to do this was to simulate it taking multiple random shots and choosing / find tuning the closest to the other player.

I also noticed that the more logs there are spawned in a round the slower the game runs, this could be looked into to try optimise the collision detection methods which is what I believe is causing this slowdown.

Finally, I feel like more work could be done on the menu and winning screen to improve the interface and looks but this would be purely for visual purposes.