Shooting & Trajectory Mechanic:
FileNames: Blakes prototypes
Blake Leahy
Time to complete: a couple hours per prototype

When prototyping the shooting mechanic and arc/trajectory mechanic, I wanted to mainly get an idea of the different approaches for the trajectory. As I found the Unity physics engine to be very useful and function well in my initial testing of shooting prior to testing the arc, I am using the physics engines built in components and functions to shoot over changing the cannonball's position in a parabola. Therefore, trying different visual trajectory methods (LineRendered vs procedurally adding 'point' prefabs dotted) was my main prototype goal.

My most basic first prototype is designed to test the visual trajectory line using a LineRenderer component. It is using the player sprite Sen created with cannonballs (downloaded online for now as placeholder) just firing straight out of the ship. With this implementation, I liked how I have the trajectory only come on-screen on player mouse click - a feature I think will go well in our alpha version. The flexibility of testing the parabola length through changing the 'Trajectory Time Step' and 'Trajectory Step Count' public variables set to my Launch script is another useful feature considered to add to future implementations. The main negative to this prototype approach using a LineRenderer was the trajectory visually being a coloured line straight through when our vision for Rogue Waves was a dotted trajectory. A second prototype arose from this (Blakes 2nd Prototype shooting Scene) , where I procedurally generated 'point' prefabs into the trajectory line on each game update() after initially spawning them in on start() before the mouse is moved to aim the arc. I found this approach much more visually appealing for the player so it will likely be the style of showing trajectory we implement in our alpha version (minus arc spawning on game start). In the second prototype, I also messed around with adding a cannon object (sprite from online as a placeholder) on top of the ship which moves with the mouse trajectory aim - another alpha release implementation.

Therefore, from this prototype, we have been able to decide on the dotted line arc mechanic procedurally spawned in on update(), and physics based shooting mechanics. This prototyping raised the following questions for future implementations:
With player ship movement and crew in the player ship, how will the cannon sit on the ship and follow the arc and visually show the cannonball shooting out?
When do we want and when don't we want the trajectory showing? Trajectory opacity increases each turn?
Could the trajectory showing just be a powerup to eventually obtain from a crew member? (maybe a cannon barrel powerup for accuracy)
Could the force of the players shot be calculated as mouse is dragged back rather than a public variable for just the developers to access? Or a bar for how much force you want in your shot?

Turn Based Prototype:

File Name: Jamie Prototype
Jamie Rule
Time to complete: hour of research and couple hours to implement

For the turn based combat system prototype I based it off a pokemon type fighting system. Where one player attacks and their turn ends and their opponent then attacks with their turn ending. This continues until one player defeats their opponent on their turn. In the prototype the player has a button to fire their shot where the enemy fires automatically. To show the combat system working I gave each sprite an attack of 1 and HP of 2. This means the player will shoot twice before winning the battle.

I created an enum called the BattleState which has the different states during the game. Starting in the start state for the start of the game and then changing into the player state. The player attack button can only be activated during this state. During testing the player could spam the button fire more than one shot a turn, so I created an idle state for when switching between states. The state will switch between the player and enemy turn after each time they attack. But each time an attack is fired it checks if the opponent is at 0 health. If this is true the game will end and will show a win/lose screen depending on who won.

This worked to show how the turn based system should work but is very basic. The states are good for limiting what can be done on the screen for each player's turn. But for this version the shots are not objects and always hit. When creating the game the shots fired will have to travel through the air and have the chance to miss. This is something I have thought about and might have to do differently for the game. As this will require knowing when the shot has missed or landed to know when to change the state. But this creates a good starting spot for the combat system that can be merged with the other game machanics.

AI:

File Name: EnemyAI
Sen Macmaster
Time to complete : AI movement= one hour, AI Shooting= at least half a dozen hours

The enemy movement mechanic took less than an hour. Enemy movement is used to make the game harder and avoid Players from copying the same angle constantly. If the Enemy moves after a hit, the players must think about how much extra or less power or angle they need to hit. After a turn ends (space key), the Enemy ship moves using random chance. A number between -x and x is added to the initial position of the Enemy which causes it to randomly choose where to go. The value has to be more than a 1/6x so that it doesn’t make a pointless move.
This was done simply through an if in the Update() method where it would call the coroutine which, generated a random value, checked if it was more than 1/6x, added it to the initial

position incrementally using a loop to create a smoother animation. WaitForSeconds() was added to format the phase in a way that lets the Player understand when the move finishes.

The shooting mechanic is a script which functions similarly where it knows the perfect shot but uses randomness to vary the accuracy in a fair way. The arc is updated constantly to check new values of the Player(target) and Enemy(start) positions. The shooting required a parabola with two x-axis points and a vertex. The vertex would be the difference between starting and target position and a placeholder height. The script loops a number of points which connect into a line. Each x point on the line calculates the value of the y point using a quadratic formula y = a(x-h)^2 = k (where the vertex is denoted as (h, k)) put into a Vector3 array and the positions of those point are set on the line renderer. The array is used to guide the projectile along the line when the Shoot() method is called, which creates an instance of a cannonball prefab and destroys it after it finishes its trajectory. Randomness is added during the drawing of the arc where a random value between 3f and 10f is created for the height of the vertex, and a random value between -x and x displaces the value of the target position. This causes the AI to overshoot and undershoot more consistently. The value of x will be called as a public variable and will change as difficulty and turn number increases.
A couple of issues with the AI reside from the shooting mechanic being clunky. The bullet from the Enemy only shoots along the points and therefore has no gravity or physics attached to it, unlike the Player shooting from Blake's code.

UI -

File name: MainMenu

Shamen Kumar

I completed the UI lab before creating a main menu page (2 hours) and then spent time trying to create a Main Menu based off the UI Lab (2 hours)

For the prototype I was more focused on getting the parts of the UI created instead of having it functioning at this stage. I wanted to have the buttons in place and roughly have the main menu created with fonts/art roughly looking like what we want the end product to be like. At this stage the art and background of the main menu is just a rough idea of what we want the game to look like and will likely be changed for alpha/beta release. The HUD still needs to be created to display the current player health and enemy health, this should be simple to add however we need to discuss the impact that player/enemy shots will have on health. Once we incorporate prototyped scenes we just need to map the buttons to their respective functions. As a group we need to discuss whether we want the player to be able to use the mouse to click the buttons or whether we want to navigate the menu through keyboard input.

Player Movement and Wave Feature -

File name: TrigonometryWaveTest

Sen Macmaster, Shamen Kumar

We wanted to get basic functionality of a wave which we can increase the intensity of. We chose to use the player movement function form the Space invaders lab as we are moving the ships left to right on the horizontal axis. Initially we created a wave feature (white line shows outline of wave which will be changed so that there are wave art/animations) that causes the players to move up and down on the vertical axis, inspiration was taken from the procedural lab, After this the player movement script to the player sprite. At this stage the wave functions well, however the intensity of the wave for each level needs to be finalised to add difficulty through the levels. Also the player movement needs to be restricted to a certain section of the screen as currently, the player can move all across the screen and collide with the enemy ship. This is not the desired functionality of player movement and we need to decide upon the speed of movement and the limit to which the player can move the ship across the screen.