Arenathon Post Mortem

Luc Kadletz

Gonzalo Rojo

[Project Goals](#_xrk4xwxw4fso)

[Enemy System](#_6pz67iv2rysb)

[Wave Spawner](#_49kqvqe9a33j)

[Movement AI](#_agtz8sli5grg)

[Attack System](#_ppmx0qbgx6bv)

[Ability System](#_nsvlcb4lr8o9)

[Dash](#_uufb1c5uzur)

[Blink](#_bkug5qn6bthr)

[Punch](#_2bhwcmj05akh)

[Explode](#_nrbx02vp20nd)

[Slash](#_fvukefo6gj4f)

[Shoot](#_c5i58cmuvb17)

[Possible Improvements](#_8s97kxko85z2)

# Project Goals

Initially, our team’s goal was to make a game that showcased our technical ability. However, we both expected to busy this semester, so we knew that our overall scope had to be very low. To achieve this, we aimed for an arcade style game with quick combat. In Arenathon, one or two players use a variety of abilities to defeat waves of enemies.

Since the goal of this project is to talk about it in future interviews, we will focus on the technical features we implemented, and how we would improve them.

# Enemy System

Since we wanted to demonstrate AI technical ability, we wanted to focus on creating a framework​ that would enable​ rapid iteration and data driven design.

Since the Unity editor has an extensive archetype & serialization system, this meant implementing enemy abilities as child objects of the enemy, so that attacks could be easily swapped out or instantiated at run time.

## Wave Spawner

Since we decided to go with an unending game, we implemented a wave spawner that scaled based on difficulty from wave to wave, spawning more enemies per round while always having a maximum enemies at a time counter. This last helped keep the game balanced and at a not to high difficulty.

## Movement AI

Our movement ai was a simple system that switched allowed for walking towards the player, running away from the player, and zigzagging around the player. After testing, we found that moving towards the player was the best fit for most enemies. We also unified the movement system for the player and enemies, with a component sending either player inputs or AI outputs to a shared movement component.

## Attack System

Since we wanted the enemies to use the same or similar attacks as the players, the enemies offload their logic to the abilities. By writing the ability logic in the script, it reduced the time it took to add enemy behaviors. As a nice side effect, we found that it also enables simple compounding of enemy behaviors. The base AI behavior of an ability was a simple distance & cooldown check, which covered most cases relatively well. Also, since the behaviors were implemented as game objects, we could leverage the Unity editor system to quickly adjust and specialize the enemies’ attacks.

# Ability System

Abilities were implemented as two logical components: a *meter*, which handled cooldowns and charging, and the ability, which handled the spawning of any effects the attack required. By keeping these abilities as data driven as possible, we could use a small number of scripts for several different abilities.

### Dash

The dash ability was implemented first, as a proof of concept. It simply increases your movement speed while you hold down the button, with a cooldown period before you can use it again. It was kept for its simplicity, but it wasn’t a particularly fun mechanic in this style of game.

### Blink

The blink ability allows the player to teleport a short distance with a long cooldown. However, it is known to cause bugs with the character controller. As a workaround, we simply allow the player to teleport into the level geometry, since they can’t attack the enemies from inside, and it is guaranteed not to break the game or kill them unfairly. To improve it, I would simply rework the ability to detect invalid teleports, and refund their cooldown.

### Punch

The punch ability was the first to use Unity’s legacy animation system, which uses key framing on game objects. This allowed us to make functional animations without any art talent.

### Explode

The explode ability was created exclusively for enemies. We found that our fast enemies would often get in several hits before players could react. By modifying the punch ability to create an area of effect that also harms the attacker. This effect ended up being one of the most fun enemy types, and was relatively simple to create.

### Slash

The slash ability is simply a modified punch ability. However, by inverting the cooldown on the enemy slash attack, the cooldown meter warns players of danger allowing them to move out of the way or counterattack. In the future I would add the inverted cooldown functionality as a specialized meter component, so it would be more intuitive and versatile.

### Shoot

This ability creates a projectile. It required some coupling with the targeting system in movement for AI characters, which would have been better moved to an isolated component. Since enemies have perfect aim, they needed a firing delay so that players would have time to react. This was accomplished by slowing down the projectile speed, and giving both enemies and players an aiming indicator.

Still, players have difficulty aiming well with the projectile attacks. This is fine because it's a secondary attack, but we would have liked to as a new homing ranged attack so that players who had trouble aiming could find a ranged attack that suits them.

# Possible Improvements

There wasn’t much else we could have done due to time constraints but if we’ve had the time, one of the main achievement would have been a larger variety of abilities and weapons, as well as a larger variety of enemies. We could have made some kind of menu or equipment UI where you could select your abilities or weapons and with that trying to get the strongest equipment in order to advance as much as possible and get the highest score.

Another main objective would have been playtesting. We would have loved to play test our game way more, testing all the abilities, the enemies, the difficulty and the controls.

I think having tackled these two issues we would have end up with an even better game that what we already have.