Forest Card Wars - A WebGL VR Multiplayer Card Game

Names: Shachak Gil, Sagi Taizi.

Supervisors: Boaz Sterenfeld, Yaron Honen.

Table Of Contents

Introduction

Our game is a multiplayer VR game built for direct browser play, with no downloads or installs required. We developed an action packed, interactive and competitive two player card game that can be played with any VR headset, by simply opening a browser and typing in the address.

The game draws inspiration from popular card games like Hearthstone, where the main goal is to damage your opponent and deplete his life points before he depletes yours, but with a fun VR twist that involves physical actions the player needs to do.

The player can throw cards, activate different kinds of spells with varying effects, shoot arrows and attack with a sword. Strategizing is key if you want to win!

Technologies and Platforms

Our game was developed using Unity WebGL, built as a web application and scripted in Visual Studio in C#. We have used De-Panther's WebXR Export for the browser VR tracking, and Photon PUN 2 for the multiplayer and networking.

Unity

Unity is a cross-platform game engine developed by Unity Technologies. Unity gives users the ability to create games in both 2D and 3D, as well as different build target options. in our case, we have created a 3D game with a WebGL build target.

Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Our scripting was done in C# using Visual Studio.

De-Panther's WebXR Export

De-Panther's WebXR Export is a Unity export to the WebXR JavaScript API. It supports both Augmented Reality and Virtual Reality WebXR API immersive session, and integrates the API to Unity WebGL and lets you develop experiences in the familiar editor using C#. We have used this export to allow cross-headset VR movement and action tracking in our WebGL build.

Photon PUN 2

Photon Unity Networking (PUN) is a Unity package for multiplayer games. It re-implements and enhances the features of Unity's built-in networking. Flexible matchmaking gets your players into rooms where objects can be synced over the network. RPCs, Custom Properties or "low level" Photon events are just some of the features. PUN exports to basically all platforms supported by Unity.

Equipment

- Oculus Rift Headset, Controllers and Trackers.

- Oculus Quest Headset + Controllers.

Scenes Overview

Main Menu:
A main menu scene for the game in which a ui is displayed and a room (1,2 or 3) can be chosen. After choosing a room the player can join it. Once a player joins a room the second scene is loaded.

Forest Card Wars:
The scene where the game is played. In this scene there are at most 2 players. Each scene is a depiction of a room (1,2 or 3) and once a second player joins the room, the game starts for both players. The players play until one player wins and after that happens both players are removed from the room and the Main Menu scene gets loaded again.

Gameplay

When the game starts both players get 3 initial cards in their hand and a target with full health points. Each player, in their turn, must use a card in their hand to try and damage the opponent's target. At the end of their turn, after using a card, the player needs to draw another card from their deck. There are 10 types of cards in the game which can be drawn from the deck, each type with different effects.

Sword Card: The most basic card. If thrown and the target is hit, deals a fixed amount of damage.

WarHammer Card: Activatable. On activation the card grows, making it easier to hit the target. If thrown and the target is hit, it deals less damage than the sword.

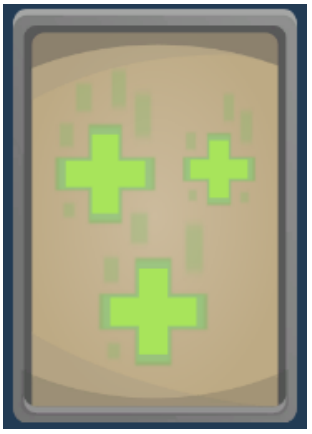
Shuriken Card: If thrown and the target is hit, deals damage based on throw speed. The faster the throw is, the more damage you deal.

Dagger Card: Activatable. The only card that guarantees damaging the target every time. This card can be activated and upon activation deals a small amount of damage to the target.

Crossbow Card: Activatable. On activation the player can shoot 3 small arrows from their hand, with decreasing velocity and damage.

Cursed Potion Card: Activatable. On activation it lowers the target's defenses, making it receive 25% more damage in the next 4 turns.

Heal Card: Activatable. Upon activation heals the player's target.



Teleport Card: Activatable. Upon activation teleports the player to the opponent's target and spawns a large sword. The player can then hit the target with the sword for a limited amount of time, before teleporting back.

Fireball Card: Activatable. After this spell is activated, the player can shoot a fireball from the palm of the hand. The fireball explodes on collision and deals some damage to the target if it was hit. It also burns the target for 3 turns and in each turn damages it more (if target was hit).

Bow Card: Activatable. After activation, allows the player to load an arrow and when the player releases, an arrow is shot with speed that scales with the time the player loaded the arrow. Deals damage if it hits the target, and the damage scales with the time the arrow was in the air.

Development Process

The main challenge we have faced in our development is with the WebXR Export we have used and the browser gameplay support. At the beginning, we searched for different approaches that will satisfy our needs for both web support and a smooth multiplayer experience.

After many trials and errors with the packages we used for enabling web VR tracking, we decided to go for De-Panther's WebXR Export as it worked best for us and we understood how we could expand his work and change it according to our needs.

However, De-Panther's WebXR Export was still far from perfect.

- The way Controller interaction was handled was a bit messy, and required us to dive deep into his code in order to understand how we can modify it.
- The GameObjects and scripts used in the asset are "hardcoded" and can only be modified to a limited degree, and some of their properties are a bit off and have caused trouble for us, for example the model's angles.
- Button pressing and the GameObject's representation to interaction and implementation were all "hardcoded" as well, forcing us to always look for creative workarounds to problems we came across. For example - At some point, we wanted to be able to pick up cards using the controllers, then destroy them while they are attached to the hand. We discovered that because of the way De-Panther implemented his controllers, we can not do that without receiving a runtime error. So we thought of a different solution, marking cards used as "to be destroyed" and disabling all other interaction with them as soon as they are used, then destroy them when we leave them.
- While Unity offers official ways to implement UI for VR applications, we could not use those ways as we were using the export and not the official Unity VR plugin. We had to implement UI interaction ourselves using custom canvases created and destroyed at runtime, button colliders and raycasting.

Another main challenge we faced was the WebGL build itself. Many times during development we added new features that worked just fine during debugging within unity, but when building the game to WebGL we found out that the features don't work. Sometimes it was simply assets that we used that weren't supported in WebGL, and sometimes it was simply unexpected behaviour that only occurred in WebGL and finding exactly what the problem was was difficult.

As we were developing a multiplayer game, another challenge we had was synchronizing game objects and events in the game for both players in the room. Photon PUN2 is a great tool for this but there are multiple ways to synchronize events using PUN2, and understanding exactly which kind of events should be synchronized in which way took us some time as there is a lot to consider.

Moreover, during development we were very cautious when adding more assets and features to the game, since keeping its size low is a top priority for us. The reason for this is that the game is supposed to be loaded in the browser, meaning every time a player wants to play it, the browser downloads all the files required to run the game from a server, and we don’t want this action to take a very long time (otherwise simply downloading the game once and storing it would be a better choice).

But even though we faced a lot of tough problems, we truly had a great time developing the game and adding new features, and we are very happy with the result!