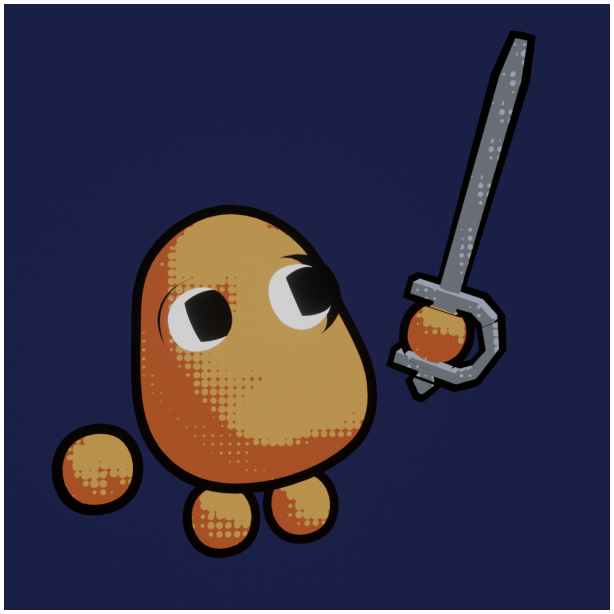
# BattleFields
## Return of the Potato



Julie Fiadino  Paul Dufour
Auguste Charpentier  David Goncalves

Prepa S2

# Contents

# *Introduction*

This project is a two players PC VR game named:
***"Battle Fields: Return of the Potatoes"***.

The game is a Tower Defense: potatoes will spawn in different places and will try to reach the player's base. The players must prevent this and hold as long as possible.

Two players are required for this game :

- The **VR player**, with a headset and controllers. He has a bow, on top of a tower, and has to aim and shoot the incoming enemies.
- The **PC player**, with a mouse and a keyboard. He has a view from the top of the map and can purchase upgrades.

Both players have to communicate together to survive through the endless waves of potatoes, wanting to steal all of our resources.

# Summary of first presentation

### 1.1.1 What was done

While the game didn't launch for the first presentation, a lot was done before that date:

- Most models and animations were ready. Graphics are using URP and some custom shaders were made. The map remains untouched.
- Base work for the enemies (potatoes) was done, they have walking animations and can carry weapons.
- `git` was properly configured for working together.
- We played with Unity and SteamVR to see what could be done. Input system, teleportation and assets from Valve were imported successfully and are being used actively.
- The longbow was implemented, with working arrows, different torches, and shootable potatoes.
- IA started being made, first thinking about the different options we could take.
- PC View and menus were being designed, but not yet implemented.
- The website had its first tools setup and the base work was done, but without much content.

### 1.1.2 What needed to be done

We mostly focused on the VR part, therefore the main focus afterwards was the gameplay for the PC player.

- Have a proper interface for the PC player
- Implement the potatoes AI
- Add variations in the gameplay
- Tinker the VR view
- Develop content for the website

# *PC View*

For this second presentation, we mainly focused on the PC-view.

## Tower placement

*Paul Dufour Julie Fiadino*

### 2.1.1 First attempts

At first, we tried to set up a script to simply place towers on click.
However, problems were encountered:

1. The player could overlay the towers which is obviously not supposed to happen.
2. The towers could be placed on the slopes. The foot of the tower went out while half of the tower was buried under the mountain.
3. Map bounds were not respected when placing new towers.

### 2.1.2 Our current solution

We decided to make a grid that covers the entire map up to our game borders.
Each possible tower slot will therefore be a tile on that grid.

After we started creating our script, we realised that, without seeing the outline of the tiles, the view was not very clear.
To fix this, we made line renders to outline them, and added numbers corresponding to the tiles.

But this solution was not ideal: the numbers were displayed on the wrong axis and the rendering was not very clean.

Julie had the idea of making a texture to show the different tiles:

1. In green the ones available
2. In orange, the one unavailable

3. In grey the borders of the boxes

Here is the result of the texture once applied to the map.

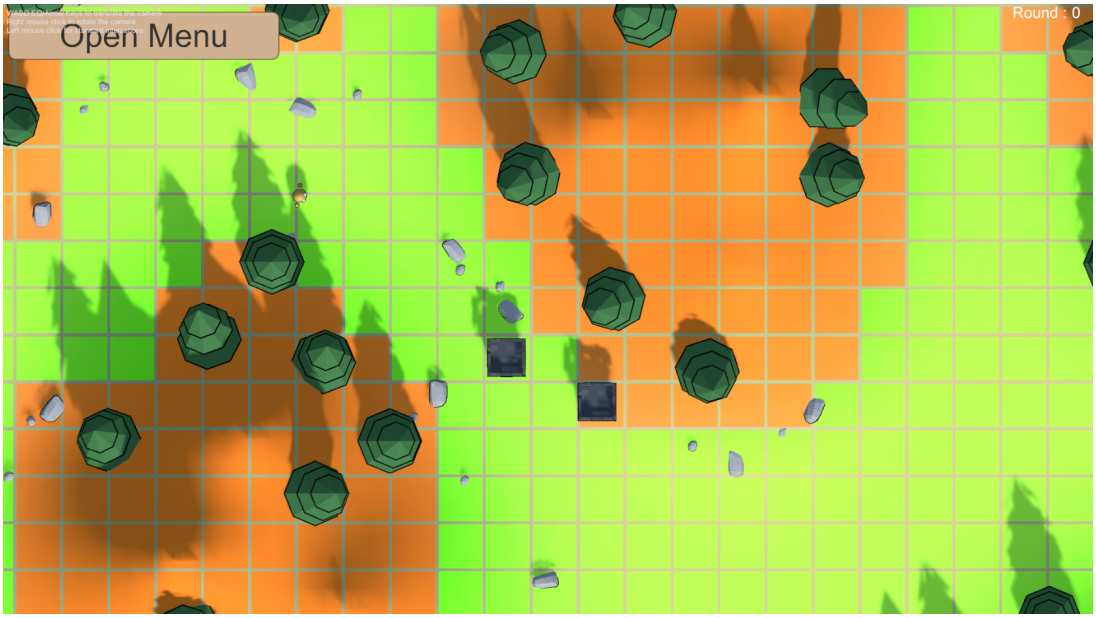



Figure 2.1: Grid for tower placement

All that remained was making sure that the towers were placed correctly on the map according to the grid.

In practice, it was a little more complicated. We needed to calculate the position of the tower on the grid according to the mouse position.
We then had to define the tiles with mountains or rocks, to prohibit any placement.

Finally, when the player adds a tower, we update the corresponding value to the grid and the job is done!

# Torch placement

*Paul Dufour*

As mentioned in the first report, we have torches that can ignite arrows for extra effects. The next step was making them available to the PC player, so we added a button for that.

The used script has two references: the list of towers, and the prefab of all torches.
When one torch must be lit, the script looks for the corresponding one in the prefab reference, then enables them for all towers.

There is currently one issue that may become a feature: new towers will not have the previously bought torches.
This is actually a good game design question: when an additional tower is bought, should you have to buy new upgrades or keep it synchronized with other towers?

We are considering the first option and will work on this afterwards.

# The menu

*Paul Dufour Julie Fiadino*

### 2.3.1 First concepts

Our first idea was this simple schematic. The goal was to create tabs to allow the player to choose between adding towers and torches. It also needed to blend in the game with animations.

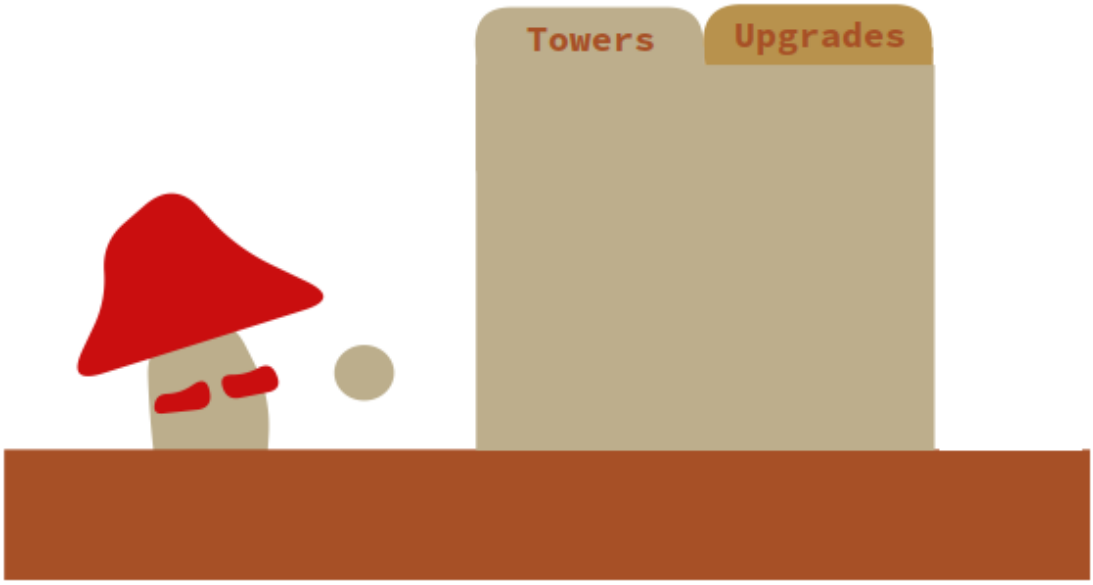



Figure 2.2: First idea for the menu

### 2.3.2 Animator

We then tried to implement the animations allowing the different elements to appear on the scene.

For this, we used the "Animator" tool in unity:

There are two stages:

1. Panels out of the player's view

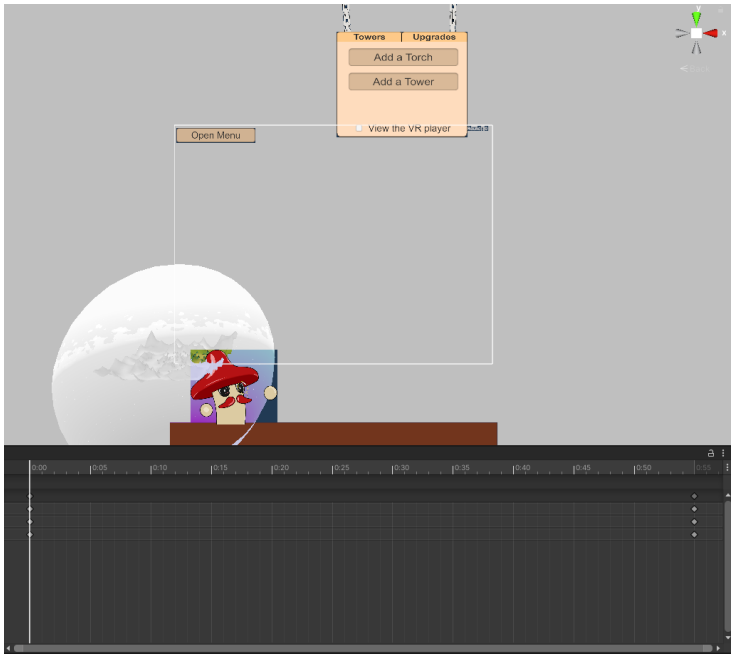



Figure 2.3: First animation state

2. Everything is displayed with a moving transition (when the shop is active) :

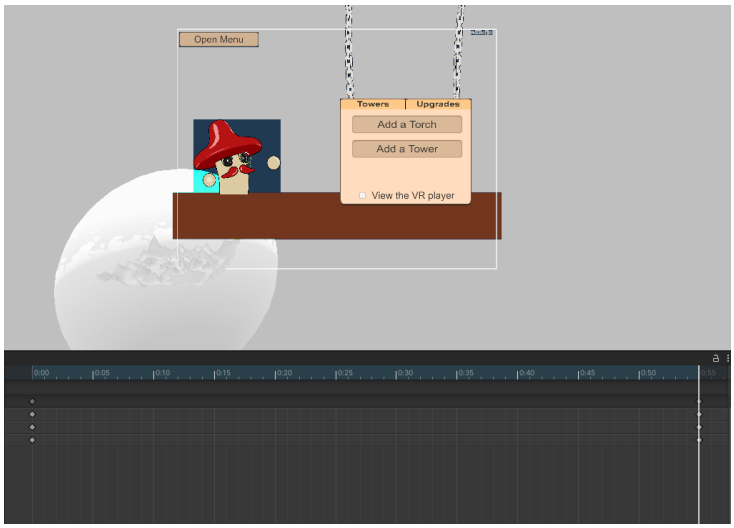



Figure 2.4: Second animation state

Then we had to manage the animator, with a boolean variable to tell if the menu is open or closed and the trick is done.

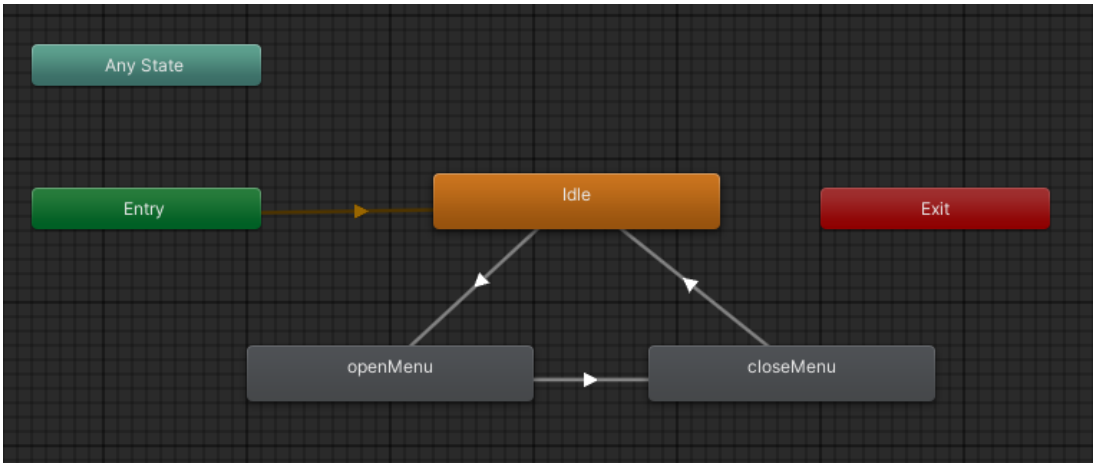



Figure 2.5: The animator

### 2.3.3 Result

In the end, with a few sprites and changes in the buttons, we reached this result :


Figure 2.6: The actual menu

# *Technical aspect*

Unity is used as the game engine, and setting up the game engine for VR took an important portion of the time since handing out the book of specifications.

## Collaborating with Unity

*Auguste Charpentier*

We explained in the first report how we used `git` for collaborating together with Unity.

A lot of time was spent tinkering different settings to ensure the workflow would be efficient, and this ended up quite well.

Branches were used accordingly for different features: everyone works on their side, rebasing changes from others from time to time, and then we work on merging all the features together.
You can see in figure 3.1 the graph of branches from git, this looks a bit like spaghetti, we're still learning how to have a proper history, but this has proved to work well already, thanks to the properly configured settings.

However, Unity doesn't play very well with this kind of setup, and we had to find more solutions.

### 3.1.1 Merging scenes and prefabs

Working on this project means constantly playing with the Unity objects, and this isn't supported well by `git`.

Merge conflicts with scenes and prefabs were very hard to deal with. The structure of the file is very confusing, and Unity assigns unique identifiers to all objects. If IDs aren't properly set back, the scene is broken, and we had to deal with these situations a lot!

At some point, we had a complex merge conflict that was very hard to resolve, and I began looking for new tools. The following were tested:

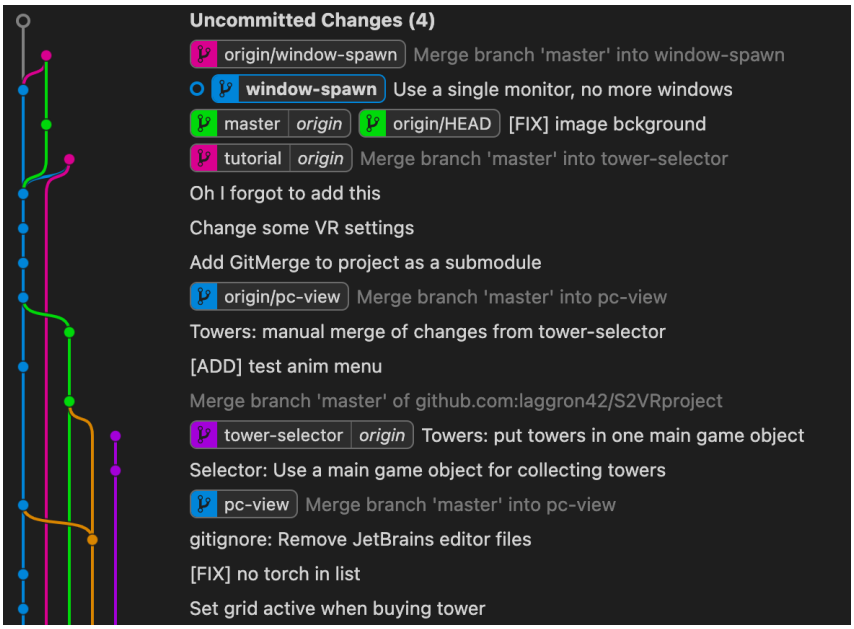



Figure 3.1: Graph of the git repository

- Unity Smart Merge, this was mentioned in the first report, and is the tool used by a lot of other programs under the hood. However, I never really managed to make this work with git itself, the documentation is outdated and this was taking to much time to try and understand how this works.
- Github for Unity, an integration made by Github themselves. Not updated for 3 years, this is completely broken on the version we're using. However, we used the underlying `.gitattributes` file from the repo.

I ended up finding the GitMerge for Unity plugin, something that I never heard of and was not looking very promising: 8 years old, paused for 4 years because of a Unity bug, and started becoming active in the last month.

I gave it a try, and was amazed by how good it worked! The plugin works inside Unity and displays a 3-ways merge for the conflicting items, see figure 3.2.
Not only is the conflict very clear, it is also very easy to resolve. Click any arrow to accept the solution from x or y, or even write your own solution. All objects are clear, a double click on anything will focus the editor accordingly, and the conflict is now resolved! I'm now excited to see a merge conflict, given how easy and satisfying it is to use this tool.

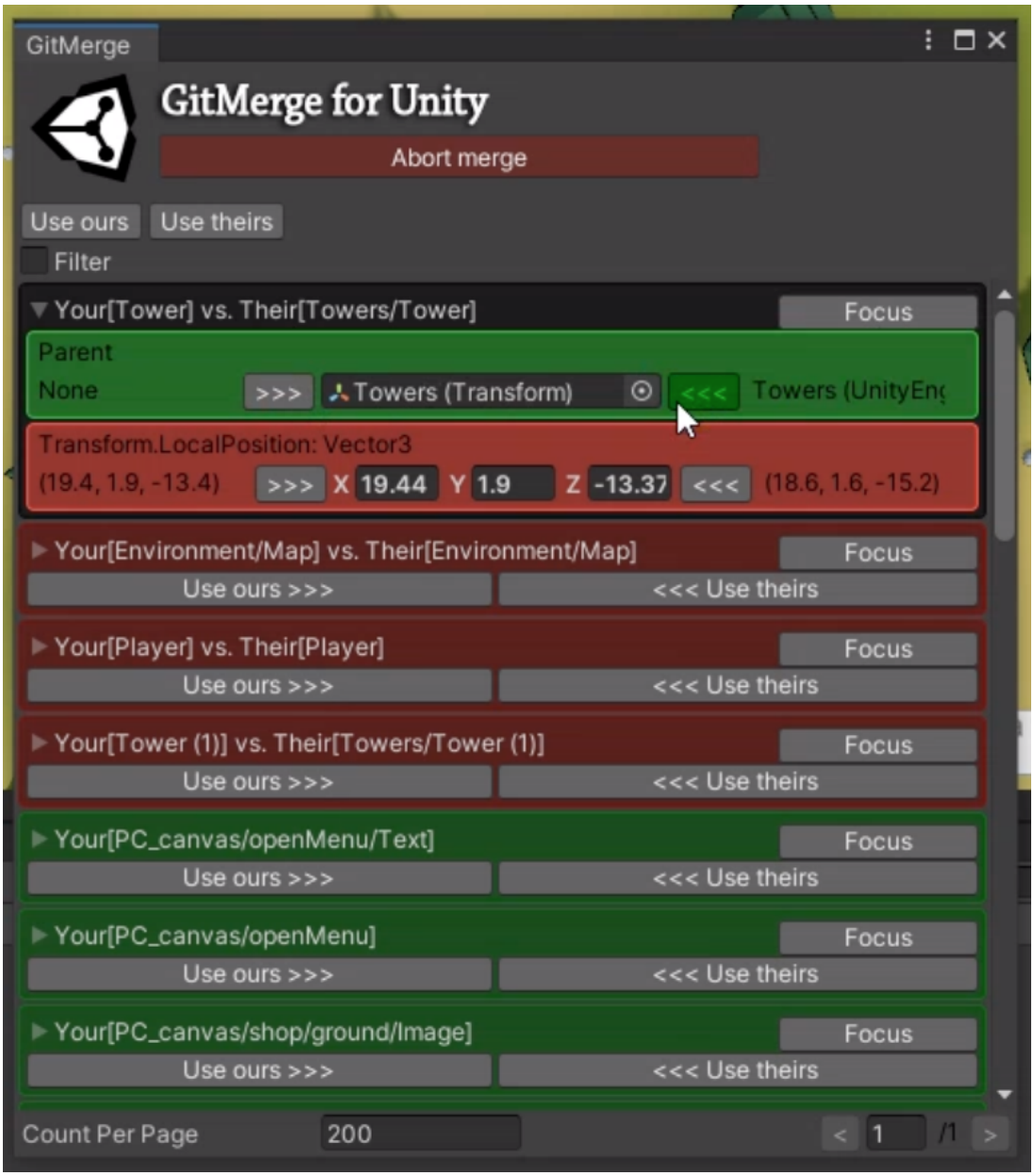



Figure 3.2: GitMerge for Unity

# Asymmetric gameplay

*Auguste Charpentier*

Battle Fields features an innovative asymmetric gameplay:

- The VR player is placed on top of a tower, with a bow, and has to shoot at incoming enemies
- The PC player has a global view and will manage upgrades, money, bonus, and communicate with the other player about the waves

Therefore, we have to manage two cameras in our game, which in itself is not hard to do, but displaying them once the game is built is less straightforward.

Unity offers 8 displays, each camera can be setup for one specific display, and it is easy to display all of them in the Unity editor.

Our game has to deal with 3 views:

- The VR player: this counts as two cameras, one for each eye, but they do not use any display, and are managed by SteamVR.
- The VR view: It's nice to see what the VR player is doing, so by default, a VR device will use Display 1 to show the view of the VR player.
- The PC player: Interface for the player, with buttons and such.

Then we need two displays. However, a display for Unity represents a physical display, so anything set to "Display 2" will only display if the user has a dual screen setup.
This is a serious limitation, so I tried to make Unity spawn a new application window, and after a day of searching, I confirmed this is not supported at all.

One questionable unofficial package claims to do so, but it modifies the engine internally in `C++` to do so, and the windows doesn't even support GUI events such as clicks (while costing $25).

In the end, we managed to find a way to do everything with a single display: trick the VR driver to make it render into a texture, which is then part of the canvas.
We do not know if we're going to keep this for the final release, as the goal of the game is to communicate information with the other player. In the meantime, we will keep this for debugging.

SteamVR has a standalone "VR view" button, so there is still a solution for displaying the VR view in a separate window.

# Virtual Reality

*Auguste Charpentier and Julie Fiadino*

VR is great, but it comes to the cost of an increased difficulty in general, but I didn't expect *that many* problems.

### 3.3.1 Problems

Our first presentation was a failure, in front of the teacher and 5 ACDCs who came for us. The game refused to launch, and it is time to explain what happened.

We have decided not to produce a build of the game and instead run it directly from the Unity editor, because of an issue with multiple windows that was not yet fixed.
However, there is a random bug where the game refuses to launch in VR, claiming SteamVR has not been initialised (not installed or no headset plugged in). This is what happened during the presentation.

This is not the first time I have encountered this bug, and I remembered one thing: it was fixed randomly.
When looking on the Internet, there are plenty of different advises: change BIOS options, reinstall SteamVR, the plugin, or even Unity, reboot Windows, change seemingly unrelated options...
In the end, I found one solution that always work: disable and enable SteamVR in the project settings. Then the issue is resolved until the editor is closed.

I got used to very random issues like that, but this one was quite frustrating given the context. VR in Unity has a lot of issues, a limited community and rarely updated tools, up to now I managed to find solutions but this is very time consuming.

I do not regret choosing to develop a VR game, I'm having a lot of fun anyway, I only hope we will not have more blocking issues that will seriously affect the final render of our game.

### 3.3.2 Future tasks

#### Tutorial

VR is basically working and playable for me, but the game is still not playable if you do not have someone behind you to explain how things work.
I therefore started working on a tutorial scene, walking you through the different controls.

This scene should be a short sequence introducing you to controls gradually, assuming the player never played in VR before, so we need to have this in mind.

For now, the plan is the following: There will be multiple platforms the player has to teleport to, one platform for each control introduced:

1. Teleport to the next platform
2. Use the joystick to turn around
3. Pick the longbow
4. Pick an arrow and shoot on a target
5. Ignite an arrow and shoot on a target
6. Grab and use a throwable weapon*
7. Open the menu and exit the tutorial

** This has not been implemented yet*

For that, we will use SteamVR's hints features, highlighting a button on the controller with a little text window.
In addition to this, Julie Fiadino made animations of the interactions, to better show how the user should use the controller.

For example, to aim the bow, the player needs to press the trigger and place his thumb on the trackpad, as if he was pinching the string. This is not very intuitive at first.

Another example is throwable objects: the Valve Index's controllers are made so you can completely open and close your hand, and first-time players are scared to drop the controller while it's properly attached.
The player should be familiar with grabbing and throwing objects, which means literally throwing your controller.

Those examples and animations should help for a better understanding of the VR features, making the player ready to hop in the game in no time!

Figure 3.3: NavMesh pathfinding

## Artificial intelligence

*David Goncalves*

For the AI, Unity's NavMesh is used. It allows us to easily define the space our enemies will be in and move in.

Figure 3.4: NavMesh Zone in blue

The enemies will spawn randomly on defined spawn points, then will choose a target between the one that are available depending on which one is closer.
Then Unity will find the shortest path, taking into account the obstacles and height of the ground (see figure 3.3).

Once set, the potatoes will find the fastest way to access the target. If a building appear and it is closer than the one they were aiming for, the potatoes will change targets. It as been made sure that potatoes don't overlap with each other or with constructions.
When they are close enough, the potatoes will attack the target.

There is three type of potatoes that have different stats and weapons (no weapons, sword, hammer). Each wave will spawn after a certain amount of time; the higher the number of rounds is, the higher the count and power of the enemies is.
In terms of stats, the towers contain only health while the potatoes have attack damage, health and attack rate.

Though everything seems fine at first, once a tower is destroyed, the platform allowing the player to teleport is destroyed as well. And this cause problems because it becomes a null value in a list where it shouldn't be.

To fix this, instead of finding a way to remove it from the list, we've come with the idea to keep the towers in the game but in a "destroyed" state and repairable. This allow us to keep the teleport platform but make it inaccessible to the player. To avoid blocking all paths possible by creating indestructible walls of towers, a high price will be put on the construction of a tower (probably a limit on the number of them as well).

## Music and Sound design *Paul Dufour and Julie Fiadino*

For the game to be complete, it needs a good amount of sound effects and music.

We started by recording battle noises for the potatoes, as well as footsteps. Then a set of weapon noises for the potatoes' attacks. We also recorded a little horn tune to signal the start of a wave.

For the music, we wanted it to give a mediaeval feel to the game. We went for several royalty free tunes to match the different scenes of the game.

Now the only part missing is the game integration. The potato have a state function which takes care of the different animations transitions and the waves are managed by script so building those sounds into the game should not be very difficult.

# Website

The website was already well advanced at the first presentation. But that was no reason to stop there.

## Credit page

*Paul Dufour*

We've done a credit page with all the different tools used for this project.

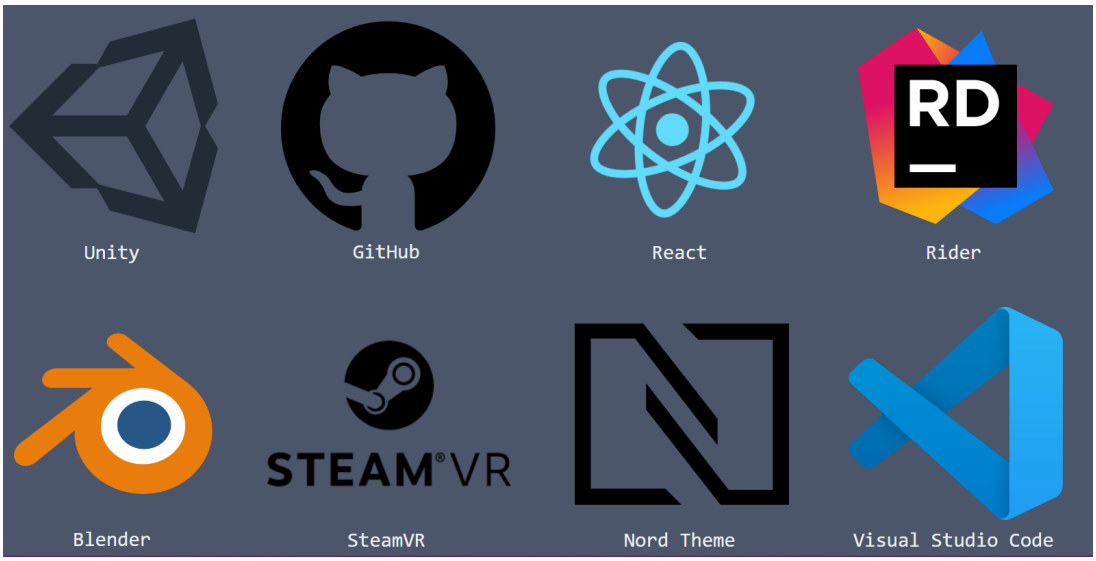



Figure 4.1: Credit page

So this page is fully responsive. If the page width is less than 600px then there will be only one tool per line.
Between 600px and 1600px there will be two tools per line, and above the code behaves as shown here.

# Scroll to top

*Paul Dufour*

We have added a script to go back to the top of the page with each change.
Indeed, before the user remained at the bottom, it was necessary to change that.

```
export default function ScrollToTop() {
    const { pathname } = useLocation();

    //redirect with a scroll to top
    useEffect(() => {
        window.scrollTo(0, 0);
    }, [pathname]);

    return null;
}
```

After some time, reflection, and several tests, just this small code made it possible to solve the problem.

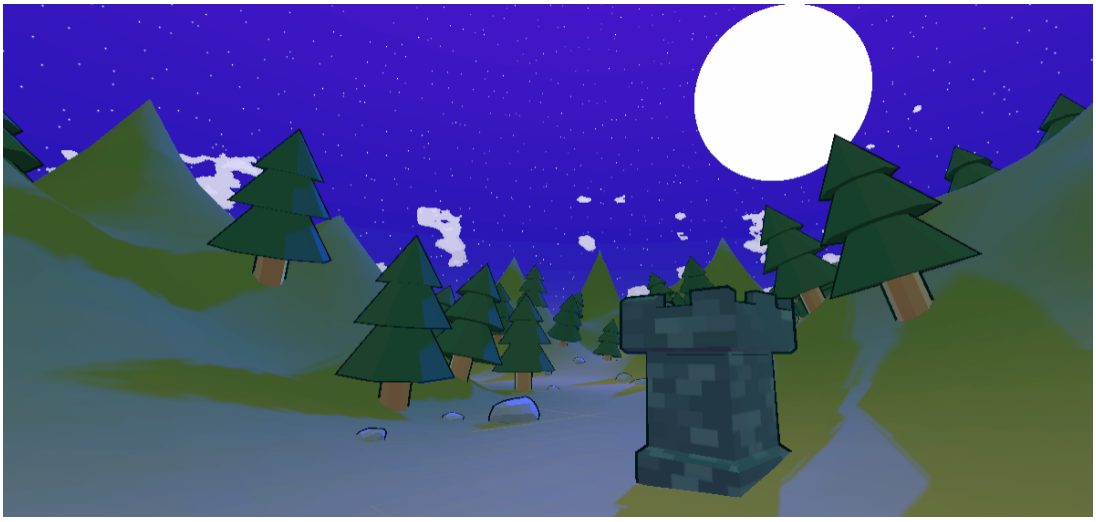


Figure 4.2: Our game at nighttime

# *Conclusion*

The game is starting to look good now!

Provided our game doesn't crash once again during the presentation, we are very satisfied with the progress we have done.
The game has now reached a functional state, it is mostly working with the flagship features we want.

Of course, there is still something big remaining: fun. The most important part of creating a game is the final one, details and little things that will make the game fun to play.
Interface is very basic, the stream of waves is continuous, there are almost no upgrades possible, overall the game is bland.

The goal from now on is to find fun things to add. Upgrades, ammunition and money management, tinkering the UI, game and level design adjustments...

Publishing the game to Steam as a free title is still considered! Last in our priorities, but an achievement we would be proud of.

**More than expected** **Less than expected** **Less than 30%** **As expected**

| Progress forecast (in %) | | | | |
|---|---|---|---|---|
| Categories | Presentation 1 | Expected Progress | Current Progress | Final |
| Local Coop<br>Paul/David/Auguste | 20 | 80 | 80 | 100 |
| VR<br>Auguste | 80 | 90 | 95 | 100 |
| PC<br>Paul/Julie/David | 20 | 80 | 70 | 100 |
| Menus<br>David | 20 | 40 | 40 | 100 |
| Map<br>Julie | 90 | 100 | 100 | 100 |
| Gameplay<br>Paul/David/Auguste/Julie | 40 | 75 | 80 | 100 |
| Enemies<br>David | 20 | 60 | 70 | 100 |
| Longbow<br>Auguste | 100 | 100 | 100 | 100 |
| 3D modeling<br>Julie | 60 | 80 | 90 | 100 |
| Sound Design<br>Paul/Julie | 0 | 60 | 70 | 100 |
| Music<br>Paul/Julie | 0 | 60 | 60 | 100 |
| Website<br>Paul | 50 | 70 | 85 | 100 |

Figure 5.1: Current advancement and expected progress