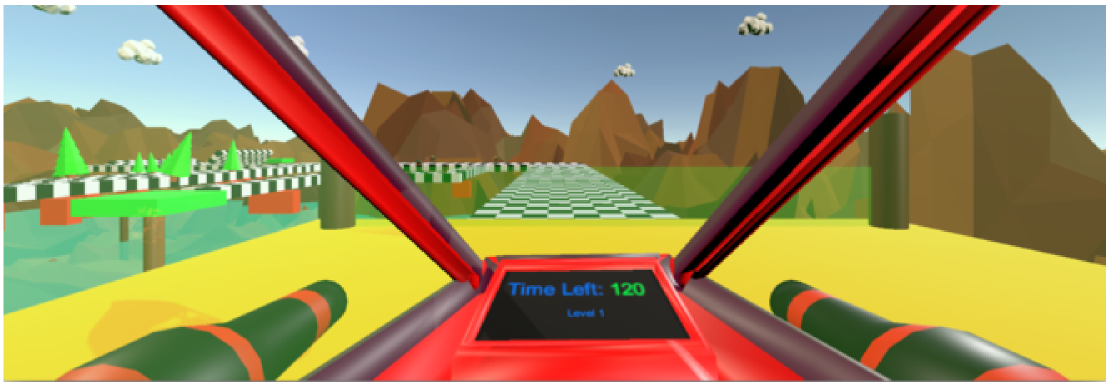
CockpitVR - Group 6

Adan Häfliger, Jonathan Link & Xavier Willemin
Supervised by Prof. Ronan Boulic & Thibault Porssut

26 mai 2017



Contents

CockpitVR is a game initially inspired from Gundam. You are immersed in the cockpit of a robot equipped with one canon in each hand and move in the level. The goal is to reach one of the green platforms in a given amount of time. Of course, some obstacles prevent you from progressing too fast : walls, turrets, moving floors.

Since gaming is better when experienced in multiplayer, we also implemented an asymmetric gameplay that can be scaled to any number of players. There roles is to activate the obstacles slowing the progression of the Vive-equipped player by triggering obstacles at distance through a tablet.

1 Game design & rules

Since no blood is allowed in this project, we opted for a time-oriented approach. This means that you cannot die in our game. You can just run out of time. If the user falls of the pathway, the robot is teleported at the beginning of the level. Of course, the time continues to elapse and is not reset. So be quick, but be careful not to fall to avoid losing too much time by starting over !

You can control the robot the following way :

- Pitch one or both controllers forward to move forward and backward to move backwards
- Yaw one or both controllers to move sideways
- Roll one or both controllers to rotate on yourself

You can of course combine any of these three basic controls to move more freely.

When both controllers are used to move the robot, the average of both inputs is computed. This has the benefit to increase the precision of the movement of the robot with respect to your hands. To disengage a controller, hold its trigger button (below your index). You will then move the canon of the corresponding hand.

At the moment, the following features and interactions are available in the first level :

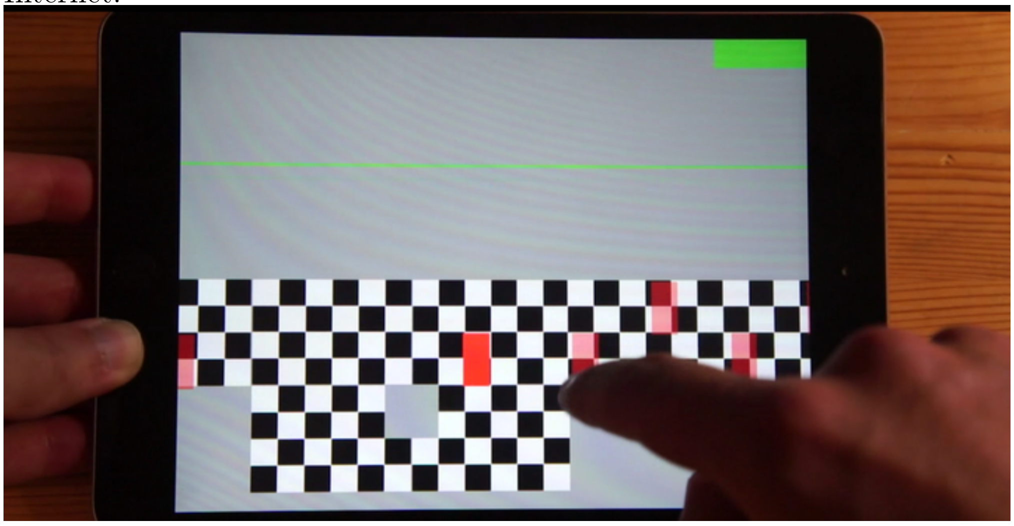
- Shrinking pathway along the course to force the mastering of the movement metaphor.
- Rotating blocs of pathway, requiring the user to carefully time its movements to avoid falling.

- Walls that appear for predetermined duration that the user can avoid. Be careful, some of these walls might surprise and/or frighten you, as they are tagged with images (Trump faces, for instance).
- Destroyable walls (distinguished from the other walls by a painted target-sign). The user needs to shoot the wall multiple times with his canons to destroy it.
- Turrets projecting flames. The flames will not hurt you, but rather strongly unbalance your robot. You should then avoid them if you do not want to fall.

These obstacles can be triggered by other players connected to the host (the computer to which the Vive is connected) with their device (tablet, smartphone or computer). They will see a scrollable 2D view of the level from above, called the minimap (see figure below). This map is refreshed every second to display the position of the robot. A latency in the refresh rate has been added to increase the difficulty.

The minimap also shows the position of the obstacles that can be triggered by the network players. Clicking on them activates the obstacle.

Figure 1: The players on the client application can trigger the obstacles over the Internet.



2 Running the project

2.1 Technical requirements

As for any virtual reality project, both the hardware and software requirements can't be neglected. We can guarantee that the project is running in the following configuration, but might as well in different environments :

- HTC Vive in a calibrated room-scale (2 lighthouses, 1 HMD and 2 controllers)
- A powerful computer with a gaming graphic card (tested with an Nvidia GTX Titan). Check www.vive.com/us/ready for more informations.
- Unity 5.5+ (tested on 5.5.2)
- SteamVR with a Steam account
- Any tablet, smartphone or computer (for the players without the Vive in multiplayer)

2.2 Setup

1. Turn on the lighthouses and wait that they sync each other.
2. Turn on the HMD and the two controllers and check that they are tracked.
3. Place a chair in the tracking area and sit on it once you wear the HMD and have the controllers in your hands.
4. Launch the .exe file (or ask somebody to do it for you). Note that the game will be oriented with respect to the orientation of the HMD at the moment of the launch.
5. Since you will use the 3D orientation of the controllers to move the robot and its arms, you now have to calibrate the neutral orientation of the controllers. So relax and once you are in a comfortable position, stay more or less steady with your hands and click on one of the grip buttons (the rounded-rectangles on the handle under your palm) of one of the controllers. This step will also reliably compute which controller is in your left hand and which one is in your right one.

6. You are now ready to play ! You can take your time at this point in the yellow zone, but be careful ; once you are on the checkered-flag pathway the time starts ticking...

Hint : if you do not want to move the robot but still need to move your hands, you can hold the trigger of both controllers. This will indeed disengage the controllers as a way to move the robot and move the canons instead.

3 Implementation

3.1 HTC Vive Interaction

Since this is a VR project, we worked a lot to build a convincing experience for the user with the help of the VR hardware, namely the HTC Vive. We also wanted to be innovative and bring to the field new means of interaction between the player and the virtual game. That is why we developed an algorithm to be able to use the Vive controllers as a joystick, which is to us the most user-friendly way of driving without any additional (and costly) hardware. Here are the details of its implementation.

We first captured the orientation of both controllers by reading the rotation angle around the right (to move forward/backward), the up (to move sideways) and the forward (to rotate on itself) axis. This is quite each when the robot is still. But we placed the cameraRig in the cockpit, such that the player is always looking in the same direction as the robot. This means that we have to be careful with the movements of the robot (especially the rotations) while doing our projections.

We also had to fine tune some thresholds to reduce as much as possible improper movements such as moving forward even if the intended movement was a single roll of the controllers which should have only induced a rotation on itself.

All this has also been replicated for the movements of the canons. The initial position of the arm of the player is captured in the first frame of the triggered button down. We get the position of the hand with the built-in tracker and we extrapolate the position of the shoulder to compute an "arm" direction vector, locally to the cameraRig.

We do this at each frame and get the rotation quaternion between the current and the initial "arm" vector, always locally to the cameraRig. We then apply this rotation (after a transformation of coordinates system) to the initial "arm" vector of the robot in the world. The "arm" vector of the robot is known, since we created it in Unity.

3.2 Networking

We built two independent applications ; one for the Vive-player and one for the network-player. The two applications communicate through the internet.

The first application, the level, is running on the host, the computer to which the Vive is connected to. It handles the movements of the robot and displays the 3D world containing the game level. The second application, the minimap, is the client.

Here is the list of all the possible messages that can be exchanged between the host and the client.

- GetBlocks
- Block
- GetWallObstacles
- WallObstacles
- GetWallObstaclesImage
- WallObstaclesImage
- GetStartPlateforms
- StartPlateform
- GetPlateforms
- Plateform
- GetRobotPosition
- RobotPosition
- Start
- TriggerWallObstacle
- WallObstacleHasFinished
- WallObstacleHasFinished
- GameOver

- Finish

Here is an example of the structure of one message. This is the one used to send the blocks to the mini map in order to reconstruct the level.

```
public class BlockMessage : MessageBase {
        public Vector3 position;
        public Vector3 size;
        public string name;
        public string materialName;
}
```

3.3 Graphics

To make the game more dynamic and pleasant to play, we added some little animations. The clouds are moving in the sky, the water is animated as well and the finish green platforms moves to add a little difficulty. There is also some distance fog to increase the depth perception. We also took some time to make sure the environment stays coherent in terms of colors and aesthetic when we downloaded the assets or created ours. Here is the list of the downloaded assets :

- The terrain : www.assetstore.unity3d.com/en/#!/content/74333
- The trees : free3d.com/3d-model/low-poly-tree-73217.html
- The clouds : www.cgtrader.com/items/644366/download-page
- The turrets : www.assetstore.unity3d.com/en/#!/content/9872

Our own components like the cockpit and the canons have been designed with Blender.

4 Unity Specifics

Here we want to mention some key unity implementation that are worth mentioning.

We used the OpenVR plugin to integrate the HMD into our game by dragging the SteamVR prefab and the Camera rig into the scene. Since our game is a seated experience we changed the "Tracking Space" parameter of the SteamVR prefab to "Tracking Universe Seated" and we use the following code to reset the initial position of the camera:

```
Valve.VR.OpenVR.System.ResetSeatedZeroPose ();
Valve.VR.OpenVR.Compositor.SetTrackingSpace (
     Valve.VR.ETrackingUniverseOrigin.TrackingUniverseSeated);
```

Also the ResetCam.cs script makes the CameraRig follow the robot since it is not recommended to make the CameraRig a child of the vehicle directly.

To be able to develop the game without access to the HMD we implemented keyboard controls and we also found out how to disable the OpenVR plugin to remove errors. To do this, you simply add an empty Virtual Reality SDK "None" above OpenVR in Edit->Project Settings->Player.

We used rigidbody physic to move the robot. To achieve this we changed the max angular velocity of the robot rigidbody as well as good drag and angular drag values such that the robot doesn't have too much inertia but still feels like a vehicle. Then to move the robot we add relative forces and torque.

We also made a set of prefabs to ease the development of levels. That way we can instantly set up a new level simply by importing the "LevelSetup" prefab and a vehicle and all the VR code is directly in the scene. The game could easily be expanded with more levels in that manner.

5 Discussion

We didn't have time to recrute subject and conduct a proper trial. But we tested our game on several types of persons including men and women, skilled in computer science or not and gamer or not.

None of them reported unpleasant feelings nor motion-sickness. However, non-gamers subjects struggled to get the intuition of the controls with the Vive controllers. The notion of pitch, yaw and roll is indeed only known to gamers, game developers and in the aviation industry.

6 Conclusion

We learnt a lot by practicing virtual reality with this project. We are now especially skilled in projections, quaternions and spatial representation.

We did not expect to spend so much time on what we thought were basic features, but the learning curve was quite steep. We then had to reduce the number of features.

7 Ressources

The video providing an overview of our game is available on YouTube at this address : www.youtube.com/watch?v=8nC1i0AayuY. Feel free to share it broadly and use it as you want.

The codebase is entirely available on GitHub at this address : www.github.com/Ahaeflig/VR6. We encourage you to submit issues and/or pull requests on the project if you want to collaborate and participate to the development of the game.

8 Images

Figure 2: view from the cockpit designed by ourself

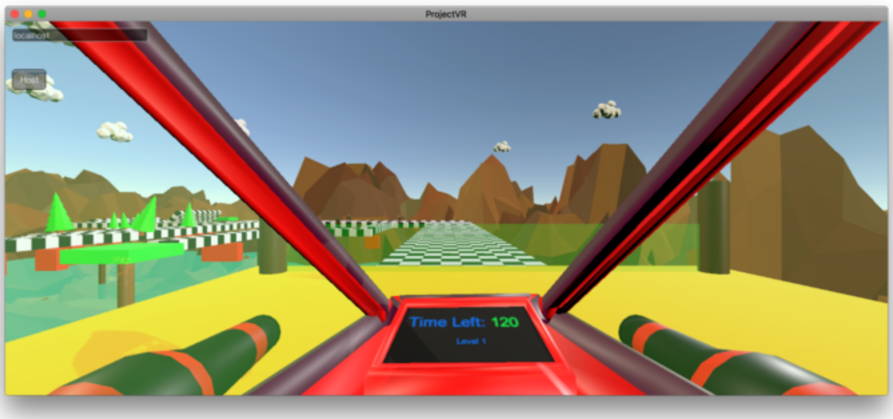


Figure 3: level 1 - low poly style

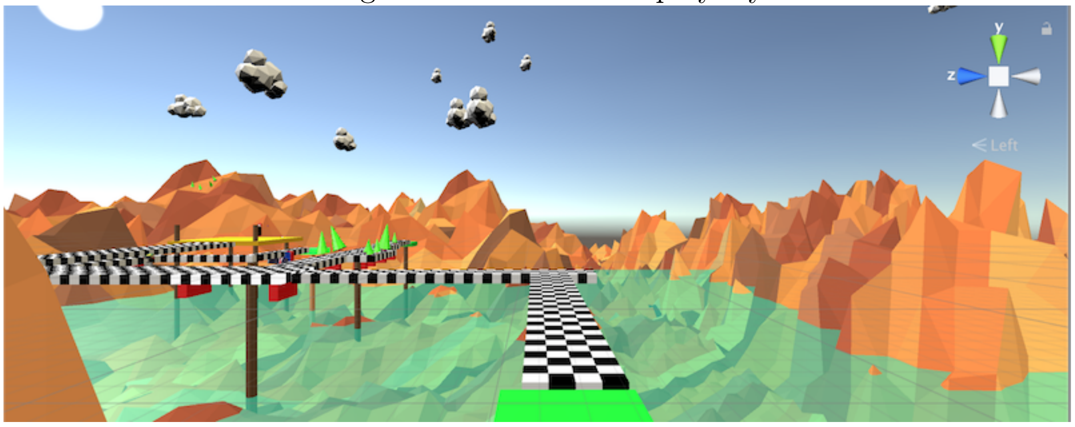


Figure 4: outside view of the ship piloted by player 1 with the htc vive

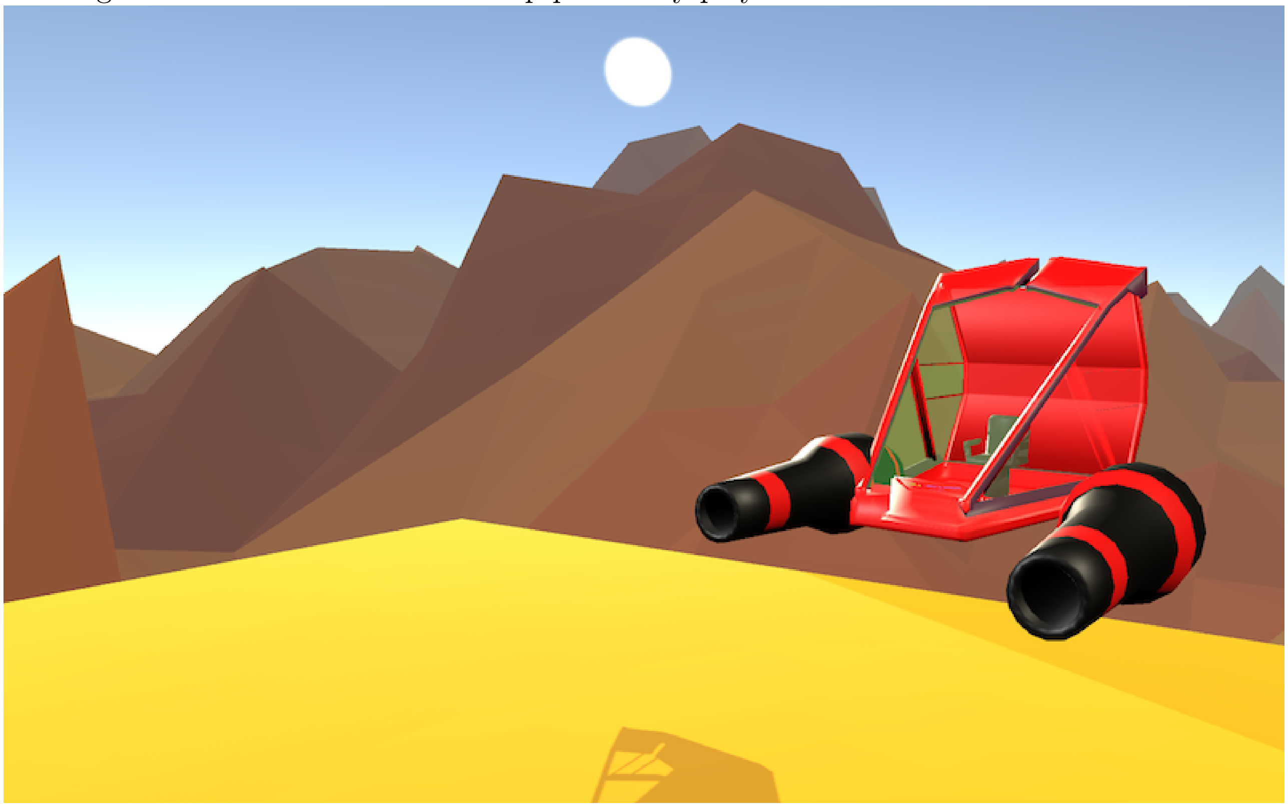


Figure 5: top view of level 1 - the orange platform is the start and the green ones are different finishes

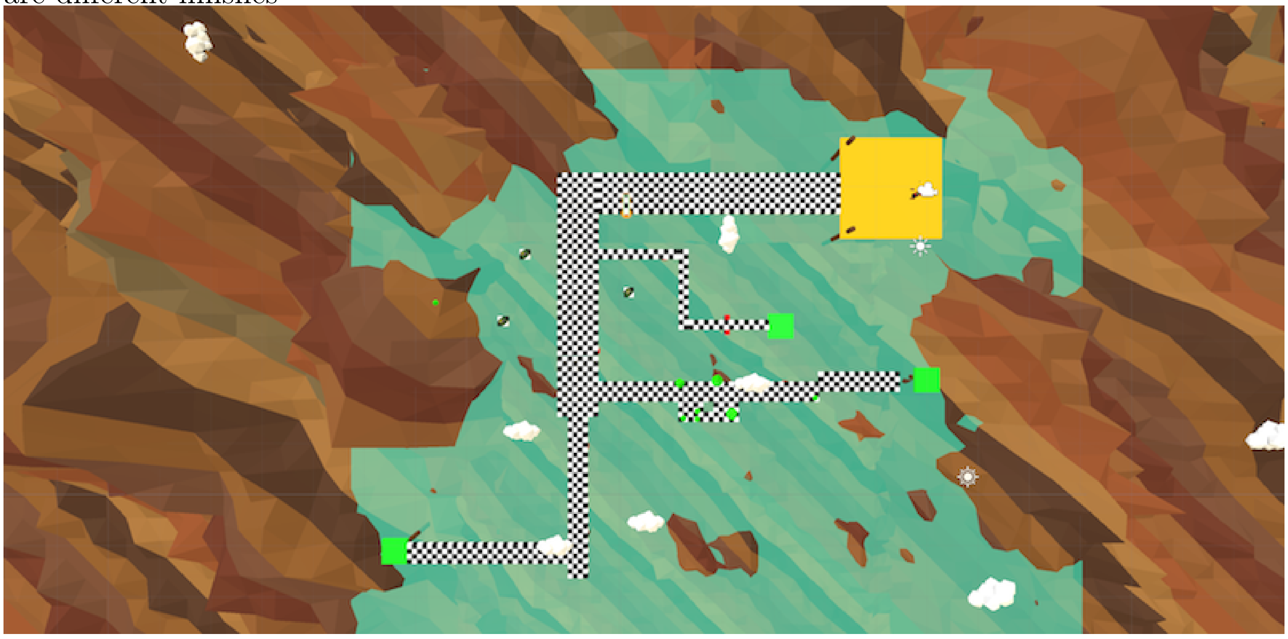


Figure 6: a wall triggered by player 2 with the tablet