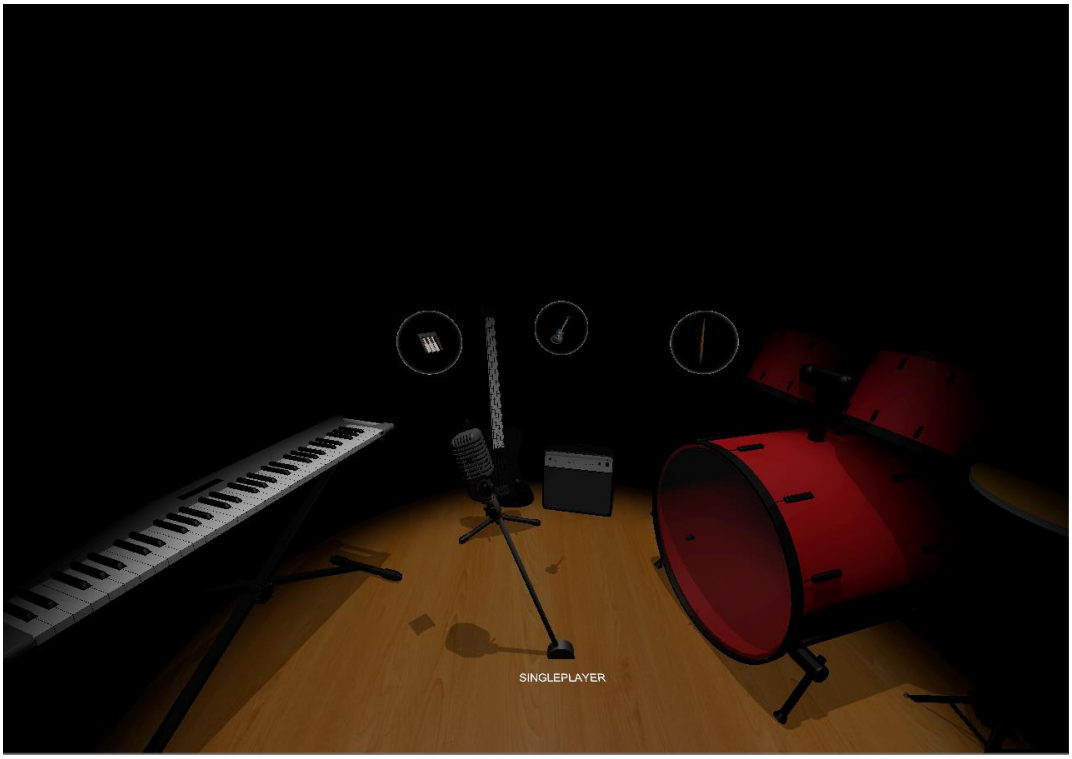
Oculus Band

SIMON NARDUZZI, ROBIN WEISKOPF (GROUP 10)



Supervisors

Ronan BOULIC

Thibault PORSSUT

Virtual Reality (CS-444)

1. Project Description

In this project, we re-created the popular Guitar Hero (© Activision) and/or Rock Band (© Electronic Arts) experience in virtual reality. There are 3 playable instruments: a guitar, a set of drums and a piano. The piano can be played with Leap Motion with or without a VR headset. The guitar and drums are Oculus Rift with Touch controllers only. The game can be played in single-player and multi-player LAN.

Our approach on the guitar allows the user to play without the need of a separate controller, unlike the newly released game Rock Band VR (© Activision 2017), which requires a separately sold controller to be able to play.

This project had been started during a hackathon in St-Gallen. We had only implemented a rudimentary drum set without haptic feedback and a buggy game system. We've since implemented new instruments, Leap Motion interaction and improved the game logic.

2. User experience

2.1 Main Menu

When the user starts up the game, he is transported inside a darkened room with a single spotlight shining on a set of instruments. He is presented with three bubbles and a mic lever. You can select the instrument to play by pointing and touching a bubble. Keep in mind that the piano only works with Leap Motion and the guitar and drums only with Oculus Touch. The lever can be grabbed to switch between single- and multi-player. When in multi-player mode, the game will wait for another player to join.

2.2 Concert

Once an instrument has been selected, the player is set for his performance on stage. When ready, he can lift his hand up high above his head for 3 seconds to start the song with a cheering crowd (when in desktop mode, press the spacebar).

On each instrument, there is a scrolling note timeline that indicates when you must hit a note. There is a visual feedback if you hit the note (green) or if you've missed it (red).

2.2.1 Drums

Playing the drums is like real life: just hit the corresponding drum at the right timing and the note is played. You get a haptic feedback when hitting a drum. Each channel of the timeline represents a drum. From left to right: Left High-hat, Left Snare, Right Snare, Right High-hat, Tom.

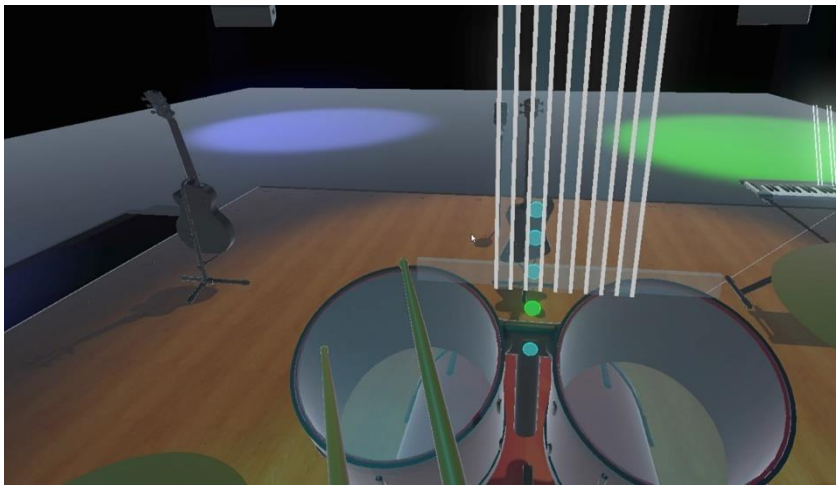


Figure 1 - Drums gameplay

2.2.2 Guitar

The guitar is played like an air guitar. You need both Oculus Touch controllers to be able to play it. The left hand holds the neck of the guitar. To hold a fret, you need to press the left-hand trigger, you won't be able to play a note unless you hold it. When holding a note, you can strum it by holding the index trigger and B at the same time (on the right Touch controller) as if you were holding a plectrum.

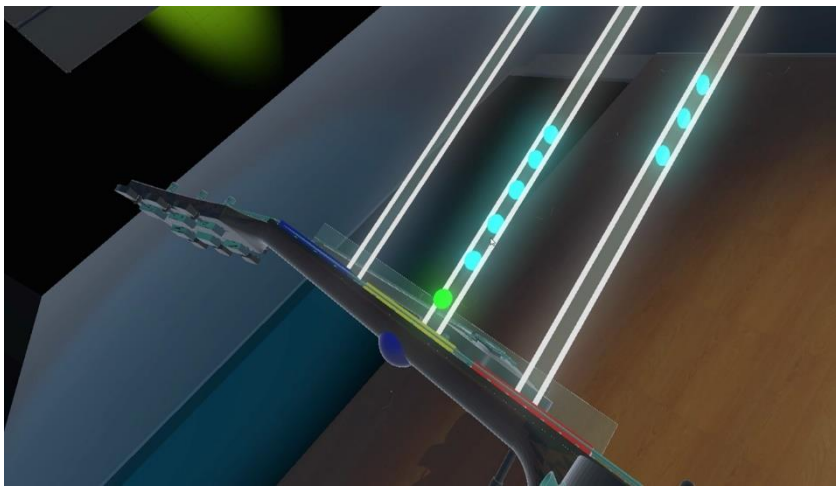


Figure 2 - Guitar gameplay

2.2.3 Piano

When playing the piano, you have 5 notes that you must hit at the right time with any of your virtual fingers from the Leap Motion.

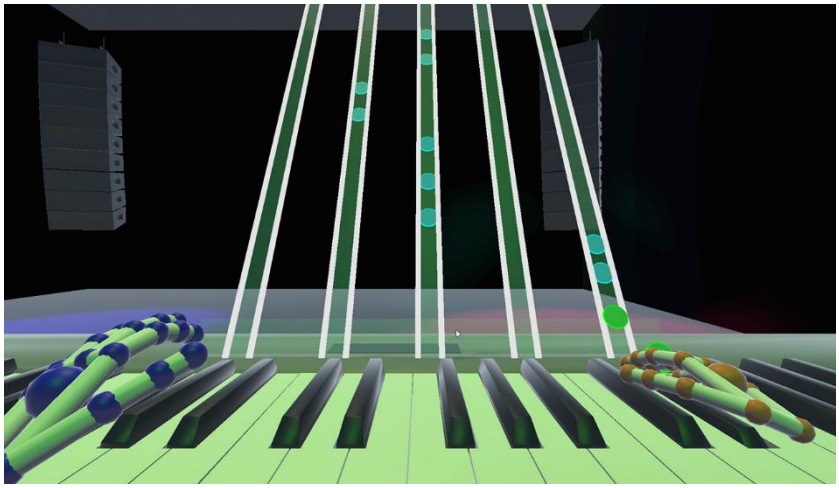


Figure 3 - Piano in Desktop Mode

3. Libraries, Resources and Algorithms

Unity3D is the game engine we chose to use for it's simple integration for VR. Additionally, we used the OVR libraries from Oculus and the Leap Motion SDK.

3.1 Credits

We used a limited set of personal use models and would like to credit the authors:

- Instrument models: Concert Stage by jonatangm20
- Player model: Futuristic Humanoid Cyborg by raptorface

We used a licensed song for the game:

- Lean On – Major Lazer& DJ Snake (feat. MØ) . The different instruments were extracted following the FLP project (Fruity Loops) made by Jan Warin, by activating the corresponding VSTs and recording each instrument separately one by one.

4. Running the code

4.1 Requirements

Material requirements:

- Oculus Rift CV1 with Oculus Touch controllers
- Leap Motion

System requirements (Oculus Rift minimum system requirements):

- Processor: Intel Core i5-4590 or equivalent
- Graphics card: NVIDIA GTX 970 or AMD Radeon R9 290
- RAM: 8GB
- Video output: HDMI 1.3
- USB: Three USB 3.0 and two USB 2.0
- OS: Windows 7 or newer

4.2 Opening the Unity3D Project

To open the Unity3D project, you need Unity 5.5.0f3. Any newer version should also work though. The binary does not have any additional requirements.

When running the project, make sure to execute the scene “Scenes/MainMenu.unity” as “Scenes/Concert.untiy” requires objects from the previous scene to be passed over to function.

5. Code comments

This section describes some of the main scripts in our application.

5.1 Main menu

NetworkDiscovery.cs

This script allows the broadcasting and discovery of a game in the local network. If the router of the LAN does not allow broadcasting, this sadly does not work. You can still enter the local IP manually on the desktop version of the game.

PointUse.cs / PointUsable.cs

In the main menu, we need to be able to select the instrument we want to play by pointing or grabbing it. Therefore, we implemented those scripts that are based on Unity collisions to be triggered. They have a list of callback events that we can define to be triggered when the object is used.

LeverManipulator.cs / LeverSelector.cs

These two scripts allow to grab the lever and to be able to move it from left to right, or the other way around. It has a callback event attached to it when the lever changes states. This script works for the grabbing for Oculus Touch and Leap Motion.

Network Parameters.cs

This script is attached to the “Params” gameobject. It does not destroy on the load of the Concert Scene. This script contains all informations about the devices connected, the mode (Singleplayer/Multiplayer) and the instrument selected. The instrument is set on click on corresponding bubble.

5.2 Concert

Setup

The Setup gameobject contains the scripts used to initialize the game. It checks the parameters passed by the Network Parameter and places the player accordingly.

GameLogic

The GameLogic is responsible to link the input given by the InstrumentManager and check if the notes are correctly played. One GameLogic is present for each instrument in the Scene. The logic is based on “Steps” gameobject (representing a note) that are displayed on different channels. Each channel has a collider (OKCollider) at his bottom. The steps collides with the OKCollider. If the corresponding key on the instrument is pressed at the same time, the GameManager register the note as “correctly played” and increment the score. Else, it desactivate the audio track corresponding to the instrument.

GameManager.cs

The GameManager contains the song asset (notes) and references to the InstrumentManager and AudioManager. It is responsible to stops the track of the instrument in the AudioManager if the ChannelManager detects an error from the InstrumentManager.

ChannelManager.cs

The ChannelManager is responsible to create the number of channels (timelines) for the instrument. It creates a clone of the original Channel gameobject. It also detect when a note collides with the OKCollider, and transmit to the GameManager if the note was correctly played or not.

StepReader.cs

Reads the partition and transmit the steps (notes) to the corresponding channel. The StepReader take the corresponding Song Asset (same as the GameManager one), and reads the lines of the XML files. It creates a list of notes by channel, with their apparition time.

NoteEvaluator.cs

Located in each “OKCollider” gameobject, the NoteEvaluator checks if an input occurs when the note is inside the collider, and return the result to the ChannelManager.

AudioManager.cs

This script stores and manages the different audio tracks of the game. It is networked to be able to send updates to other players when missing a note so that our track gets stopped for them as well. It is located in the “MusicManager” gameobject.

InstrumentManager.cs

This is the default interface between the game manager and the instrument. It stores which note is currently pressed and allows the game manager to read it.

Song.cs

Represents the song’s notes as a list of events. It uses .NET’s XmlSerializer to implicitly serialize objects to XML and inversely.

GuitarTracker.cs

This script manages the guitar interaction. It is based on a position anchor and a directional anchor to position and direct the guitar. We fixate the guitar corpus below the current viewpoint (VR headset) and use the left hand as a direction for the guitar to point towards. The distance between the left hand and the guitar center is used to calculate the current note. The strumming is based on a collider that checks the hand speed on a collision (StrumListener.cs).

DrumCollider.cs

This script manages the collision of the drumsticks with the drum. It is simple and just tells the game’s instrument manager when a collision with a drumstick has happened.

Piano

Each (active) key of the piano has a KeyPressedAnimator.cs script, which is responsible to rotate the piano key and transmit an event (KeyPressed) to the InstrumentManager of the Piano.

HandsUpDetector.cs

The HandsUpDetector detects when a hand (from Touch or Leap Motion) is placed above the head. When it is the case (and a sufficient number of players are present), it sends a signal to the SyncManager.

SyncManager.cs

The SyncManager contains a SyncVar that is shared by all AudioManagers and GameManagers of the game (singleplayer or multiplayer). It is responsible to start the game.
All managers have a link to the SyncManager. They only activate themselves when the syncvar “Play” equals true.

To record the partitions, I had to use one GameLogic, and attached the “StepRecorder” script to it. This script is responsible to detect and register input from the KeyBoard in an XML file (song asset). First, we create a mapping of keyboard keys to the channels. When a key is pressed, a step (note) is pushed in the corresponding channel and his apparition time is registered in the XML file. When we read the partition, each notes are displayed in order of apparition.

6. Conclusion

In the matter of 4 weeks we were able to significantly improve our existing project to include many instruments and new input methods. We did not focus much on visuals, but our main menu shows what the graphics of the final game could look like if we invested enough time into it.