**Introduction**

The Brain Computer Interfaces have started gathering the attention of more people in the last couple of years but it still is at its inception. This project started from the wish to work with a new technology and do something fun at the same time.

The idea of a 3D car driving game came from an observation that most mind controlled games are very basic:

* Cortex Arcade



* Emotipong



* Darts

They have basic graphics (usually 2D) and they also don’t attempt to fully control the game using your mind or face actions exclusively.

For this project, the Emotiv EPOC headset was chosen as it is the only portable headset which consists of 16 sensors (14 electrodes which record EEG data and 2 which are for reference).

**What does Emotiv do?**

In the Cognitiv suite the user can train 13 actions: left, right, push, pull, lift, drop, disappear, rotate clockwise, rotate anticlockwise, rotate left, rotate right, rotate forward and rotate backward. The downside is at any one time it can only recognise up to 4 actions. This means you have to carefully decide on which ones you choose. In this project, the push, pull, left, right actions are going to be trained by the player to control the car movement.

In the Affective suite, it can detect 4 emotions: excitement, engagement, meditation and frustration. These will be used to activate special game environment actions (e.g. increased levels of frustration will determine clouds darkening, rain and thunderstorms).

The Expressiv suite detects a range of facial expressions: left/right smirk, smile, laugh, furrow brow, raised brow, look left, look right, blink, left wink, right wink, teeth clenching. Some of these facial expressions will be used to control other events in the game. For example, teeth clenching will be used for braking while raised brow will be used for changing the camera view.

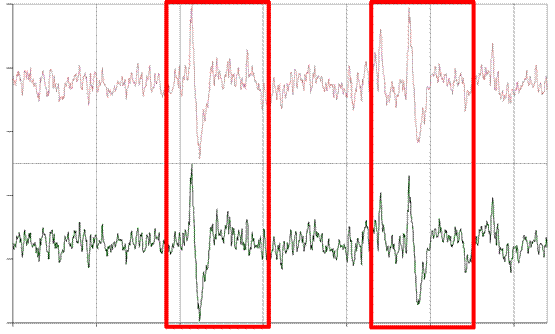
The final feature of the headset consists of the gyro detection. This will be used to control the car movement not using thoughts, but using head movement instead.

One thing to note is that during this project, we don’t have direct access to the raw EEG data, only its interpretations.

**How do we train the Emotiv?**

Training the first two types of actions is relatively easy. You just have to keep you thoughts constant and concentrate. I have found out that small head skin movements might have an effect on the training. Most people suggest visualising the actual movement should work.

A thing to be kept in mind is that any type of movement, blinking for example, has an effect on our brainwaves. The way the headset learns when we think of left or right, is by trying to find a brain wave pattern in our EEG data.



**Eye blink response on the AF3 and AF4 channels (according to An experimentation and mind-reading application for the EMotiv EPOC by Michael Adelson, Princeton University)**

Building the game

For the creation of the game, the Unity game engine has been chosen as it allows easier implementation of some of the features, in comparison to its competitor, UnReal. Unity allows you to import files simply by drag-and-drop and it also auto updates the objects if you change the source files using another program (Paint, Photoshop). The good points of UnReal would be cheaper price, better graphics (mainly lighting), and it can be used to fracture meshes. At the same time, Unity allows development in JavaScript, C# and Boo. UDK only allows UnrealScript. For BrainSpeeder, C# has been the scripting language choice.

We will now do a step by step run through what the user will experience during the game and how it is implemented.

In Unity, the biggest unit of measure is a scene. The first scene is represented by the main menu.

The user will firstly be prompted to create a new player, if one doesn’t exist, or select a player if data is already registered. There can be a maximum of 10 profiles registered at any one time. This decision has been taken so that the storage space won’t be affected negatively by storing too many accounts but also because Unity doesn’t have a drop-down menu option nor does it allow for window overflow.

The way the profiles are stored is done by using PlayerPrefs. This is a good thing since Unity ensures the data stored in this way will be easily accessible from Windows(HKCU\Software\[company name]\[product name]), Mac(~/Library/Preferences) or Linux(~/.config/unity3d/[CompanyName]/[ProductName]).

|  |  |  |
| --- | --- | --- |
| StringKey | Value | Current number of profiles (int key) |
| PlayerName0 | Albert | 1 |
| PlayerName1 | Barbara | 2 |
| PlayerName2 | Charlie | 3 |
| PlayerName3 | David | 4 |
| PlayerName4 | Elena | 5 |

PrefixKey + current number of profiles - 1

Next, the user will be expected to train the headset

Training one state takes 8 seconds and the main idea would be to visualise how that action would be accomplished. The developers suite also gives a percentage of the skill achieved in training for a specific action. Once you get 100% or close to it, you are ready to train the next action. It is also worthy to note that some people are unable to use any BCIs and this inability is called BCI illiteracy/ According to a study done by Minkyu Ahn, Hohyun Cho, Sangtae Ahn, Sung Chan Jun, these people seem to have noticeable high theta and low alpha waves which are preserved across different mental states.

The SDK is not very complicated to use once you get the hang of it. Connection to the headset is done through EE\_EngineConnect. Next, you have to permanently query for event. Then you have to obtain the event type through EE\_EmoEngineEventGetType. Examples of event type can be EE\_EmoStateUpdated, EE\_CognitivEvent, EE\_ExpressivEvent. The facial expressions will normally be returned as Booleans apart from smile, clench and raise brow which can also be returned as floats but those functions are obsolete. At the end of all the operations, EE\_Engine\_Disconnect should be called.

By accessing the Statistics option, the user will be able to see the high scores of everyone playing the game on that computer. These will be stored using PlayerPrefs in a similar way as that of creating the user profiles.

The Options tab allows the user to select the music and sound effects volume and whether the game will be played full screen.

The Help tab will display information about how to play the game and train the headset.

The Exit button will display a simple window to ask whether you are sure you want to quit. On exit, all user data will be automatically saved.

The GUI is done using mainly GUILayout. Compared to the GUI option, in GUILayout positioning is done automatically so you don’t have to specify the position of every element in the window. Position is represented by a class Rect(float left, float top, float width, float height) .

All the buttons in the scene contain a BoxCollider to detect whether the mouse is over a specific button.

Race button was left for the last because this is where most of the action happens in terms of car control.

The car model contains 2 box colliders for the lower and upper parts of the car and 4 wheel colliders, one for each wheel. The car is also subject to gravitation and all the colliders had to be in place so that the car wouldn’t fall through the ground or walk through other objects.

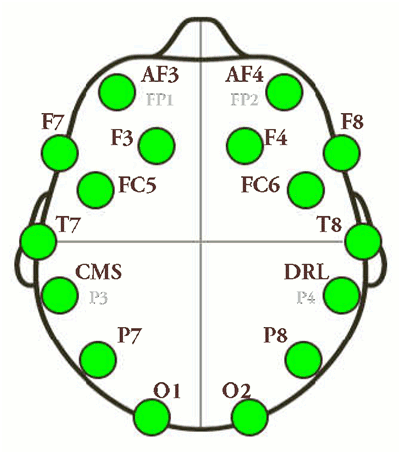
A main camera is following the car from behind. This is similar to the SmoothFollow script written in JavaScript which comes with Unity but has been rewritten in C#. The way the following is accomplished is by observing the car rotation around the Y axis. When the car rotates, the camera does it as well. This is one of the difficulties I came upon as I couldn’t find out why my camera was always following from a side of the car. Later I realised my car had an initial rotation around the Y axis so that it could face the Z direction. This simply meant I had to find the initial rotation of the car and subtract it from the wanted camera angle. This camera is also used to go inside the car when the user presses ‘C’ to change to driver view. The camera also moves to the front of the car when the user is driving in reverse.

The other parts contributing to the car movement are:

* Wheel rotation and steering angle
* Steering wheel rotation. This is where I had come upon another difficulty as a normal car is steering to the default position once the driver doesn’t exercise any force upon it. I managed to make it rotate to the original position by remembering the original steering wheel rotation and assigning those values back to it. At the moment this happens instantly which is not realistic. All of my trials to interpolate didn’t come with great success though.
* motorTorque and brakeTorque were used to make the car accelerate or brake.
* The car decelerates and stops after it reaches the finish line by using above method.
* The car has a realistic handbrake action where it slips to the left or to the right based on the speed by setting the forwardFriction and sidewaysFriction.
* The skidding script belongs to FlatTutorials and it works by finding the place where the wheel and the ground touch and seeing whether the current force is greater than the current friction.

During the game, the GUI displays the current speed, the elapsed time and a minimap where an above camera is used to track the car on the ground. In the future the GUI will also display the sensor contact quality and the current cognitive action performed and its power.

A third scene will be added where the user will be able to choose a car and colour from those available.



Difficulties:

Headset losing wireless signal – not much I can do about it

GUILayout.TextField not updating default

These are some other photos I intend to add but I thought I shouldn’t clutter the text above and have more photos then text so I’m putting them at the end. I’ll also have printscreens of the emotiv suites so I can explain on them. I am trying to find a description of what each sensor does. I saw it somewhere at some point but cannot find it anymore ☹

