Mind's Magical Arsenal - Report

[COMP 8047 – Major Project]

[Kiwon Song - A00967329]
04/20/2023

Table of Contents

1. Introduction

1.1. Student Background

My name is Kiwon Song, a 3rd term student in BCIT's Bachelor of Technology program. I enjoy gaming and would like to create a unique game of my own in the future. The main reason that I chose to pursue programming is that I like the aspect that challenges the mind to think of logical solutions to various problems.

1.1.1. Education

BCIT Computer Systems Technology – Diploma 2021

BCIT Bachelor of Technology program (In Progress 2022)

1.1.2. Skills

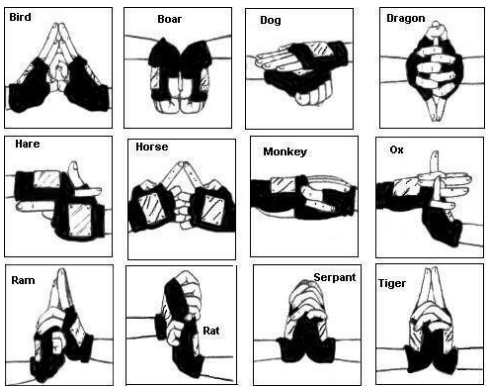
- Programming languages (C, C#, C++, Python, Java)
- Unity Game Engine
- Editors (Visual Studio, Visual Studio Code)
- Game Project Development

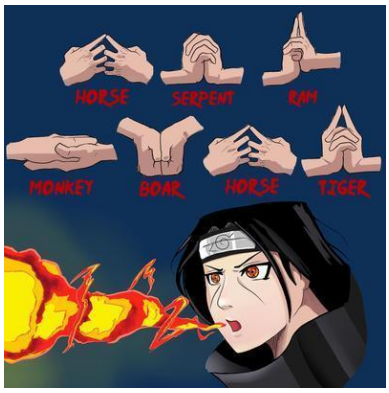
1.2. Acknowledgements

- Cortex API Guide
- Cortex Sample Project

1.3. Project Description

The project is a game that is planned to be developed around the usage of the EMOTIV EEG EPOC+ Headset that uses EEG to detect electrical activity in the brain. Brain-Computer Interface(BCI) uses a machine learning function in the Cortex API to translate the data into a specific external activity that can be registered by the computer as a specific command. Utilizing the technology, the project is planned to allow users to trigger abilities by various combinations of mental commands such as shooting fire or freezing enemies. The main idea is inspired by the manga/anime "Naruto" with their usage of hand seals:





In the anime, the characters use a specific combination of hand symbols to activate what they call 'Ninjutsu', which is the magical abilities of ninjas that produce various effects like healing or spitting out fireballs from their mouths.

Using a similar concept, the plan is to change the hand symbol to specific mental commands of the user's choice to assign them to each of their unique signs. When players utilize certain combinations of such symbols, the player will be able to enact similar actions. They can then move around the game and activate abilities with only their mental commands detected by the headsets and navigate the environment given to them.

An example would be assigning each command to input values of A, B, C, D, and E. The combination command for a fire attack could be C + D + A commands in the specific order in which the player would have command mentally in order for the in-game character to activate the fire attack.

The game is planned to be a simulation game that does not have a story. In this game, the player can explore in peace and interact with the environment as they please with their magical abilities like telekinesis to move objects or shoot fireballs to destroy wooden crates.

1.3.1. Essential Problems

The problems are mainly the two categories of connection and application of casting.

The problem with the connection would be to connect the headset and connect the data received from the headset to the Cortex API in order to train the data with AI.

While the problem with the application of casting is with design as players are to only use their minds to cast with no physical medium such as keyboards and mouse to enter the combination for casting the spell. This brings many issues such as the following examples.

- How do you differentiate spell-casting mode from normal movements in the game?
- How do you cast quickly enough with monsters coming towards you? How do you concentrate?
- How to visualize the process of casting?

Without the proper medium to cast spells with just the user's mind, this game would be considered a failure.

1.3.2. Goals and Objectives

The game's main objective is to have an interesting interactive game that can stimulate interest in Neural Headsets. These are the following goals and objectives are the minimum requirements for this game to be considered complete.

- Connection to the headset
- Connection to the Cortex API
- Proper data connection
- AI headset training
- Magic casting setup
- Basic 3D world
- Interactable objects - Preferably interactable with magic

2. Body

2.1. Background

The reason that Neural Headsets was chosen to be used for the project started from a simple research presentation. The topic could have been anything, but what was coincidently chosen at that time was the Neural Headsets. As research was done on this topic, it was found that the technology was more developed than it was initially assumed and a question popped up. Why was noit as well known as such a well-developed technology should be?

It was later discovered that it was because there was a lack of people using it outside of pure research purposes. There were people who did try to use it, but it was mostly to use the headset just as another form of a controller to be used to move characters in games. They lacked the individuality of using the headset as the main component of the entire game in itself. During the research, there wasn't much interest in the headset itself until a certain trailer was discovered that created the inspiration for this entire project.

The trailer portrayed a convention where a Neural Headset simulation game was demoed where people could move around objects with just their minds as though they were psychics. However, even though the game was advanced with great graphics, it felt lacking.

After some thought there was a sudden recollection of the anime called 'Naruto' and then it finally hit what was lacking. Versatility.

Instead of just moving objects, players should be able to have more freedom in their actions and creativity. This thought expanded into magic and spell casting where players would be able to cast spells using just the control over their mind to formulate the spell they wish to cast, including telekinesis from the trailer.

The desired result of this project is to give a proof-of-concept game that can demonstrate how Neutral Headsets can be utilized to formulate various combinations to cast abilities. To give people a jumpstart for more various and concrete ideas.

2.2. Project Statement

This project is a demonstration of a possible way to utilize Neural Headset features in games. In the process of making this game, the Cortex API will be applied using the Emotiv unity plugin in order to connect to the Emotiv Cloud server. With the connection, the players can utilize the headset to activate magic within the game with just their minds. This will allow the players to experience the feeling of formulating magic with just the effort of their minds to affect the physical reality of the game in the 3D world.

2.3. Possible Alternative Solutions

In this project, there were a few possible alternatives that could have been assigned to the project which were dimensionality, hardware, and software.

Dimensionality to change the 3D concept into a 2D one would allow the process of magic formulation to be easier to implement. This would essentially scale down the complexity of the project so that the effort put into the game and magic aspects of the projects could have been decreased a lot.

Hardware-wise, other than the Epoc+ headset there was another headset that had a lower specification and price. It had plenty of benefits to using it as equipping the headset was less tedious and had fewer requirements for players' concentration on giving commands as it was not as detailed. If this headset was used, the project would have less need for handling the cumbersome data that the EPOC+ headset would require.

Software in this alternative solution is talking about both the game engine and the imported library. For the game engine, there are various other engines other than Unity such as Unreal Engine or Godot that can be used. While for the imported library, the library is the same as the one chosen in the original solution, the difference is that there is a class with plenty of premade functionalities that are required for the project so it would be convenient to use it.

2.4. Chosen Solution

The chosen solution is still the original solution in the categories of dimensionality, Hardware, and software.

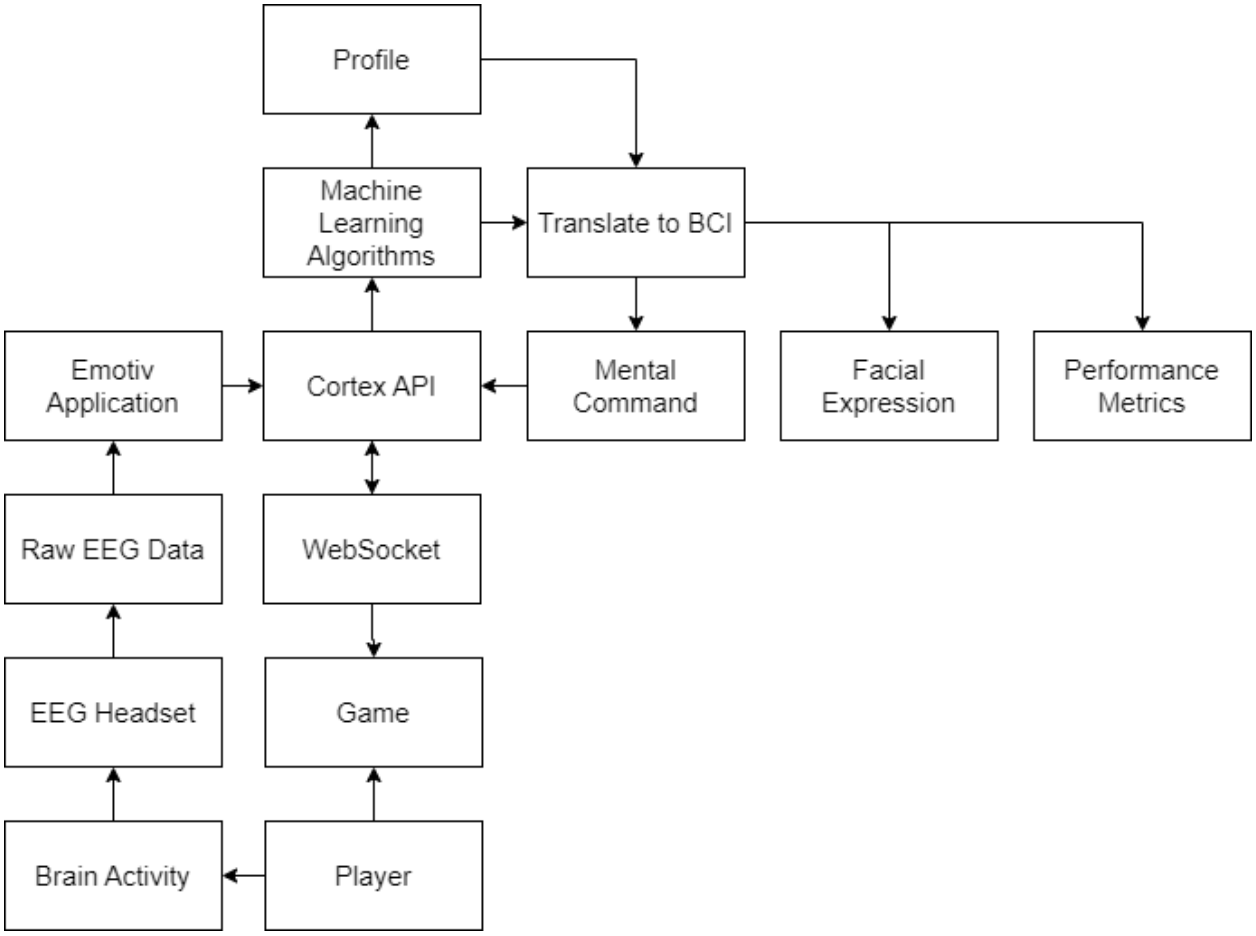
Dimensionality was kept as 3D as 2D was determined to be lacking in portraying the initial intent to give players the experience of being a proper magician properly.

EPOC+ Headset was still used for the hardware as even though it is more cumbersome the other headset doesn't use EEG which takes in the intent of the player better. For this project, utilizing the intent of the player is important to portray the magic of the game to the player.

The softwares that are used are still the Unity engine and the full imported library. The reason that Unity is still used is that it is the most familiar game engine utilized so far in the education program and the Emotiv headset only has Unity for plugins regarding game engines to utilize. Whereas the reason the simplified class is not utilized is that the class is too rigid and a lot of the functionality is slightly off the functions that are required for the project.

2.5. Details of Design and Development

2.5.1. How the headset works



The Neural Headset or EEG EPOC+ Headset works by putting sensors on the various parts of the player's head and detecting the Brain Activity using faint electronic signals that are emitted on the surface of the head. The data obtained is called EEG (Electroencephalogram).

EEG (Electroencephalogram) is a non-invasive method to record an electrogram of the spontaneous electrical activity of the brain. The image below is an example of raw EEG-monitored brain data.[1]

This Raw EEG Data at that moment is the accumulation of all the Brain Activities that have not been differentiated from one another making no sense at the moment. There when the headset continuously sends the EEG data to the Cortex API server through the Emotiv Application. The Cortex API then feeds the data into various Machine-Learning Algorithms. These machine-learning algorithms analyze and understand the various aspects of the raw data and they then organize the data into their proper categories.

Once the analysis is complete it can now be translated into Brain-Computer Interface (BCI) data which is the simplified form of the analyzed data that can easily be used directly by other programs. There are two options for this step: direct translation or profile translation.

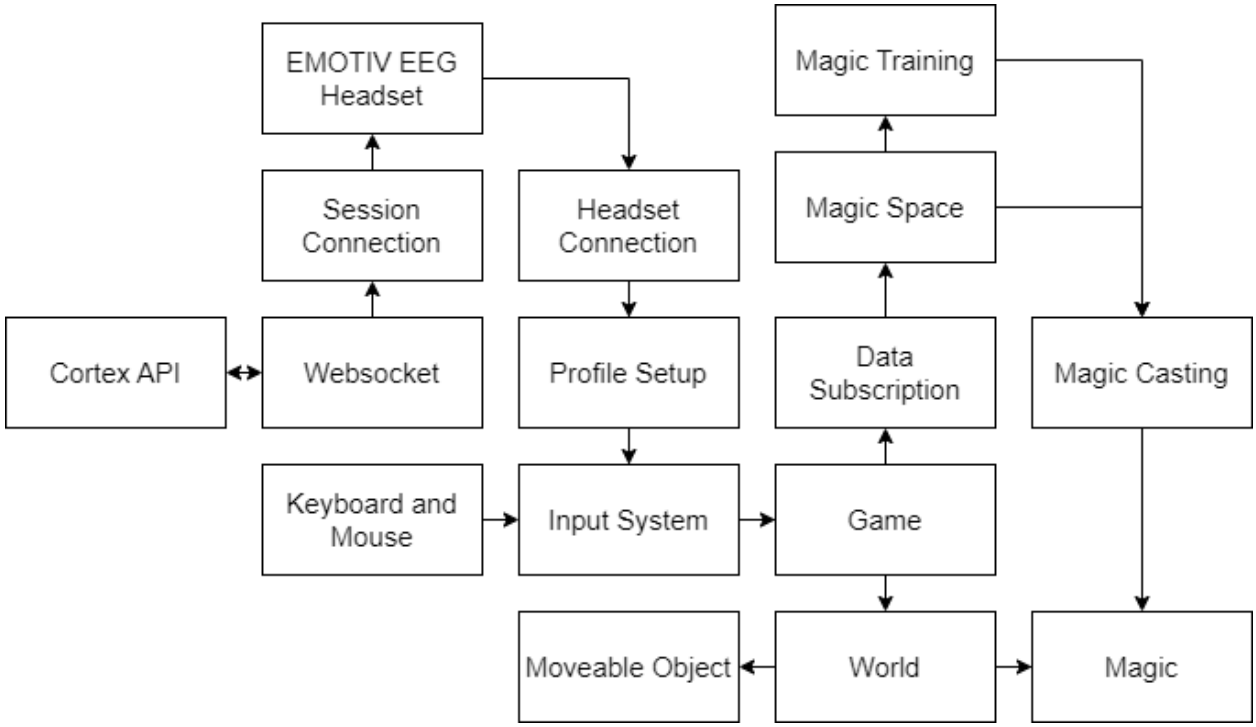
A direct translation would directly translate the output of the analyzed data into BCI.

Meanwhile, a profile translation would use profiles, which are user-customized machine-learning algorithms that redistribute the analyzed data based on how it is trained. Once the profile redistributes the data, it translates the data into BCI data.

Once the BCI data is generated, it sends the data through the Cortex API and back to the WebSocket. Where it then can feed the new BCI data into the game so that it can be used to perform the required actions.

The WebSocket is a network connection between the application and the Cortex API.

2.5.2. System/Software Architecture



Previously in the original project proposal, the diagram was more oriented toward game details due to the underestimation of the difficulty of implementing Neural Headsets into Unity. Now the project is oriented toward the Neural Headset while the game is only built to minimum standards to demonstrate the utility of the Neural Headset.

The process begins with the input setup as the game needs them before it is executed. The keyboard and mouse are simple and direct and are immediately added to the input system. Whereas the Neural Headset has to first have a session created with the user's Client ID and Client Secret. These two credentials are required to connect the application to the Cortex API through the WebSocket. Using Cortex API, the headset can be detected and connected. The inner working of the headset's data processing is described above.

Profiles are customized machine-learning algorithms for players that contain all their training progress, without them the player's unique brain signals cannot be interpreted correctly to the right command. This is why this was one of the most important parts of the project. To set up the profile, the project needed to do the following features: create, delete, load, and save profiles.

With the input system complete, the game is activated and the world is generated with the necessary details

The world mainly has two components: objects that include the player and magic.

The stated objects are basically in-game items that can be physically affected by the player's interaction including the player themselves.

Magic is the activation of special abilities by the player. To use magic, the player first has to enter the magical space that can be thought of as the mind of the wizard.

To make the magical space work, the data from the headset needs to be subscribed from the Cortex API. This translates the headset data into a much easier and more understandable format. This allows the players to train and store their unique training data that can be utilized to make the magic performance to be performed smoothly and quickly.

Once the player casts a spell and saves it, they can then use it in the actual game world environment.

2.5.3. Development

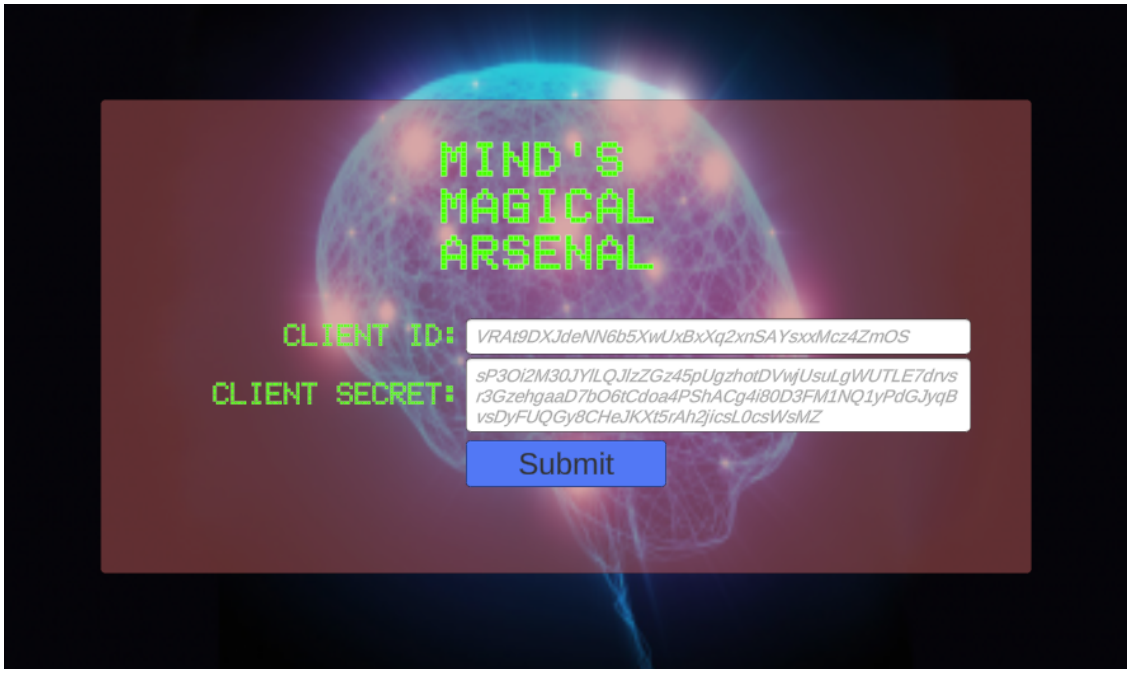
The development of the project was separated into the headset development stage, the magic development stage, and the game development stage.

2.5.3.1. Headset Development Stage

The headset development stage is where the headset connection and related functionalities are implemented such as login, connection, profile creation/loading, and training.

2.5.3.1.1. Cortex API connection

In this step the only thing that needs to be done is to connect to the Cortex API with the Client ID and Client Secret. The process to obtain the Client ID and Client Secret is explained in the requirements part of the Implication of Implementation section.



2.5.3.1.2. Cortex API log-in and Emotiv App status confirmation.

Here the Zenject Library was utilized to inject dependencies of script components of different interfaces to share important information that is acquired in each step of the process. These interfaces handle the various user interactions and explanations that the players would require in order to connect to their Neural Headsets.

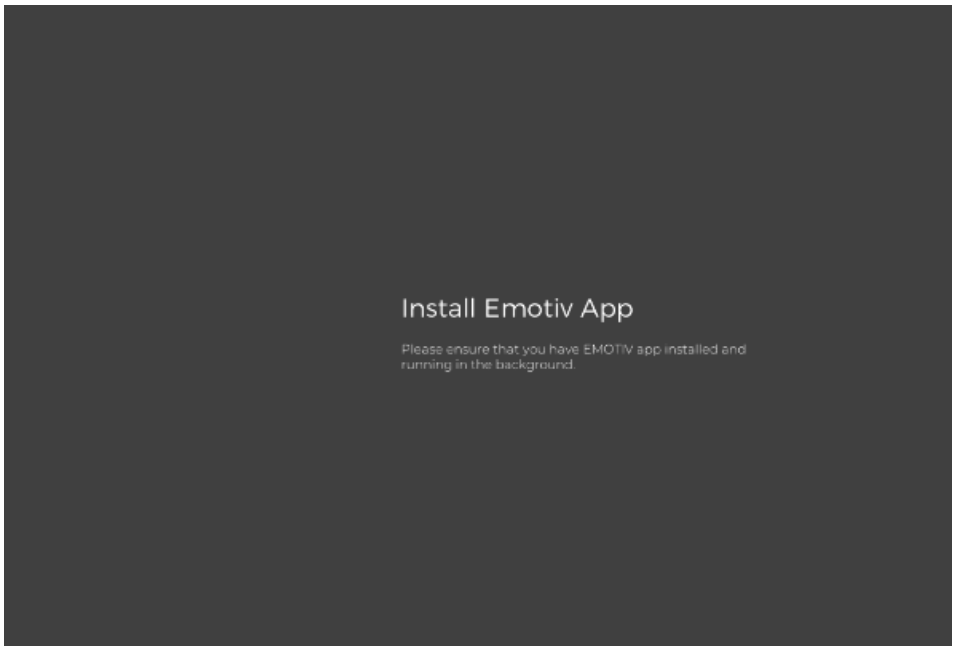
It has the following assisting factors before the next step.

Connect to Cortex API

Connecting to the Cortex did not require any user interfaces; all it needed was the Client credentials.

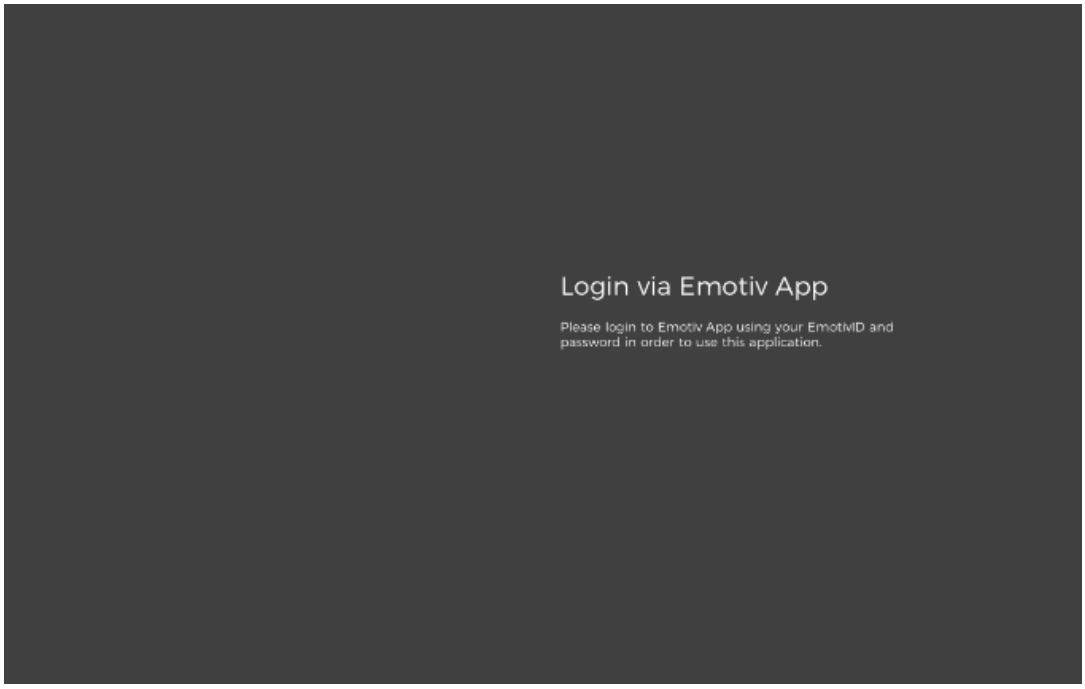
Emotiv App installation reminder

If the player does not have the Emotiv App the following page will show up as the headset cannot be connected otherwise.



Log in through the Emotiv App

If the player is not logged into the Emotiv app the following page will show up as it is required in order to detect and connect to the Neural Headsets.

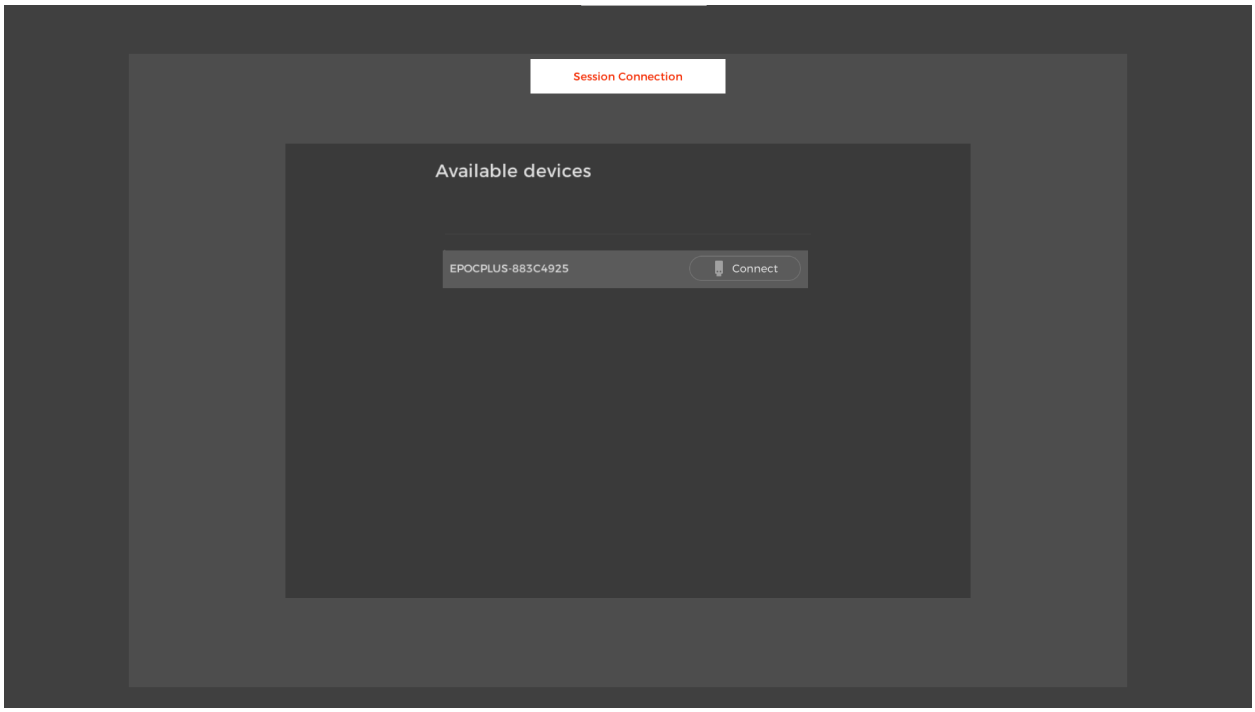

Login via Emotiv App
Please login to Emotiv App using your EmotiID and password in order to use this application.

2.5.3.1.3. Headset search and session forming

Here the headsets are searched for within the Emotiv App by requesting information from the Cortex API. The headset information is requested every few seconds and would detect the headset and display the list of headsets that are connected to the Emotiv launcher.

Once the headsets are detected and displayed the player can select the headset they wish to connect to and continue.

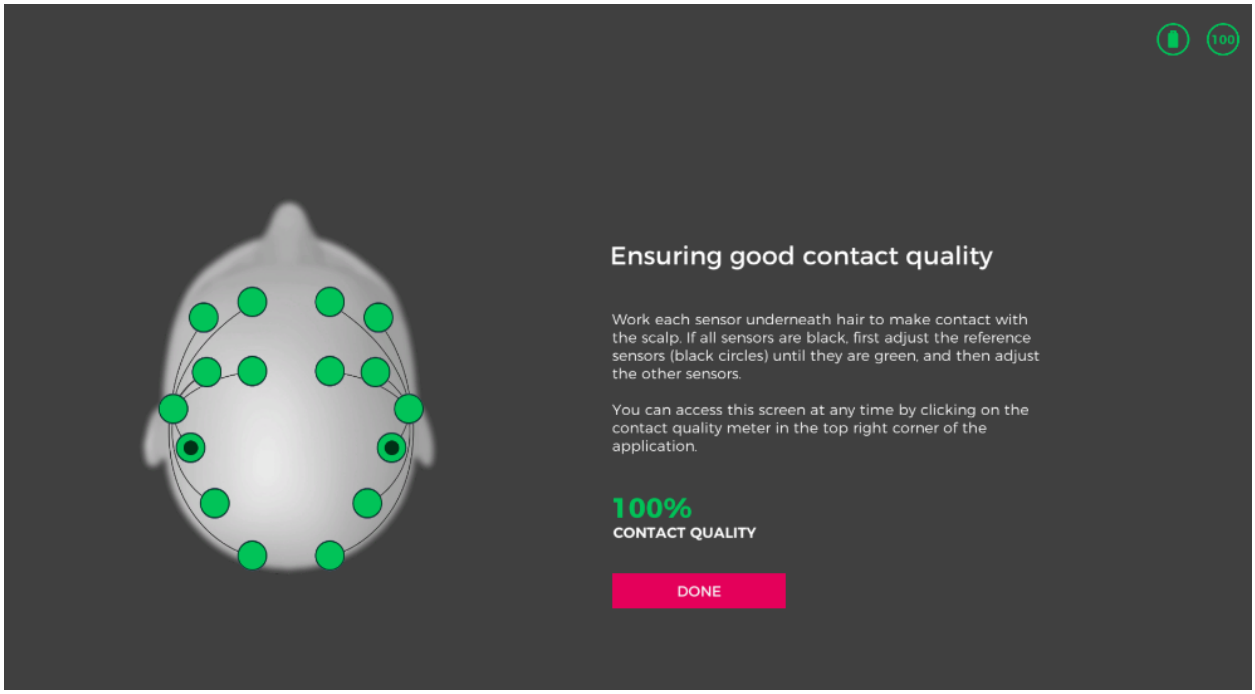
By selecting the headset, the Cortex API can then create a 'Session' which is how the connection between the headset and the Cortex API is defined. This process synchronizes the data between the headset and the Cortex API allowing it to then transmit the processed data back to the application once requested.



2.5.3.1.4. Contact Quality

Then the following interface displays the player's current contact quality with the headset. This is very important as without good contact quality, the accuracy of the commands decreases severely. This information was retrieved by using the headset information synchronized in the Cortex API.

There is also an EEG quality for how well the sensors detect the brainwaves from the contact. It was debated if this feature should be added to this project, but it was discovered that as long as the contact quality was decent, the EEG quality usually also follows it to the same level. Therefore it was deemed unnecessary for this project.

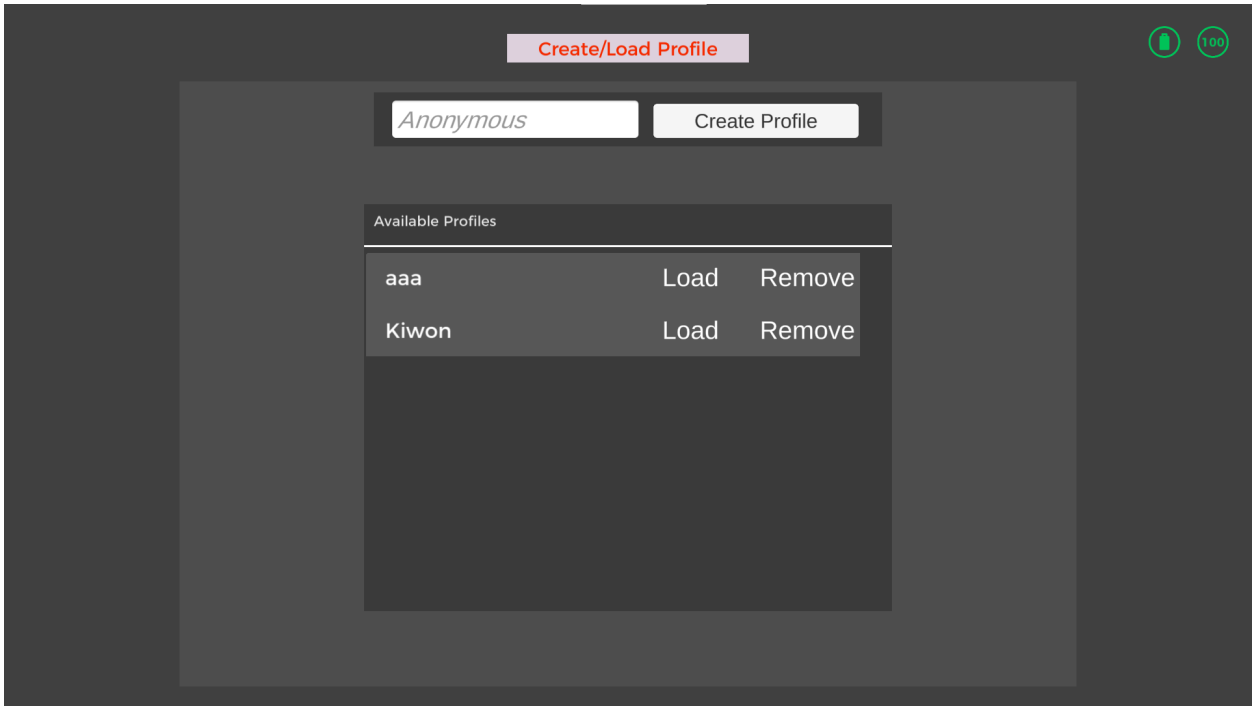


2.5.3.1.5. Profile setup

Profiles are customized machine-learning algorithms for players that contain all their training progress. Without them, the player's unique brain signals would not be interpreted correctly for the Cortex API to provide the correct information back to the player. This is why this was one of the most important parts of the project. To set up the profile, the project needed to do the following features: create, delete, load, and save profiles.

This setup was a lot more complicated in comparison to the previous steps as they had some examples to refer to. In comparison, the profile setup was done solely by looking through the files in the plugin in order to understand and utilize their functionalities.

The development process started by referring to the headset connection setup as a reference for the user interface organization and spawning of headset elements. This was referred to as the template to spawn profile elements when discovered.



As for the features of create, delete, load, and save, these were implemented by utilizing multiple scripts with the controller script at the core. The progress took a while but was smooth until it came to a halt when the plugin library began to show problems. It turned out that the development of the plugin was incomplete and that certain functionalities clashed with each other or did not exist in the first place.

To solve this, custom functions were created to make them better fit the project. However, there were certain functions that were automatically called once the Cortex API gave a response that caused the project to get caught on unexpected error calls that should never have happened because of the plugin. Therefore the plugin was slightly modified which could possibly cause problems for people who wish to use the project later on.

```
2 references
public int CreateProfile(string profileID)
{
    if (_profileList.ContainsKey(profileID)) {
        Debug.Log("Profile with the name " + profileID + " exists, delete existing profile before creating a new profile");
        return -1;
    }
    Debug.Log("Profile: " + profileID + ". created");
    //BCITraining.Instance.WantedProfileName = profileID;
    _bciTraining.CreateProfile(profileID);
    // if (_deletedProfileList.ContainsKey(profileID))
    //     _deletedProfileList.Remove(profileID);
    Process();
    return 0;
}
```

```
    }
    1 reference
    private void OnCreateProfileOK(object sender, string profileName)
    {
        UnityEngine.Debug.Log("BCITraining: OnCreateProfileOK profilename " + profileName);
        if (profileName == _wantedProfileName)
        {
            // auto load profile
            _trainingHandler.LoadProfile(_wantedProfileName, _workingHeadsetId);
        }
        else
        {
            //UnityEngine.Debug.LogError("BCITraining: OnCreateProfileOK: mismatch profilename ");
        }


    }
```

```
1 reference
public void DeleteProfile(string profileName)
{
    lock (_object) {
        string cortexToken = Authorizer.Instance.CortexToken;
        CortexClient.Instance.SetupProfile(cortexToken, profileName, "delete");
        //_deletedProfileList[profileName] = new dirox.emotiv.controller.Profile(profileName);
    }
}
```

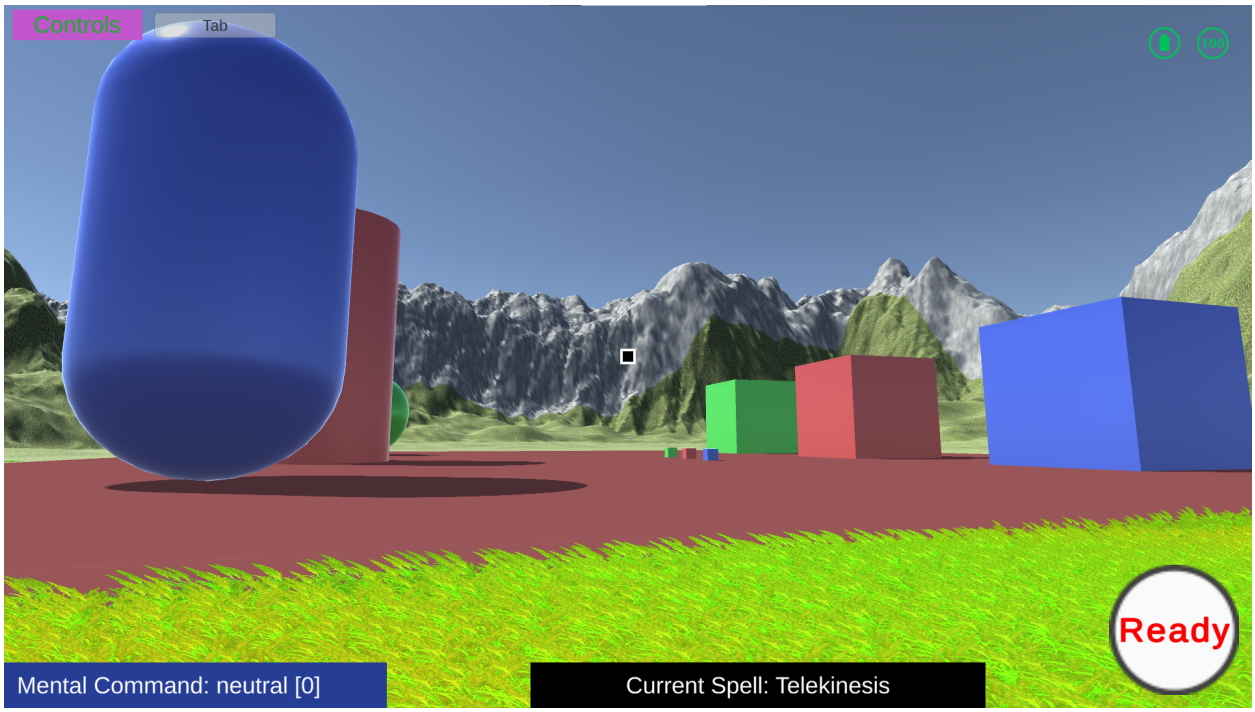
After fixing the issue, the create function worked perfectly, but once a delete function or load function was implemented, the contact quality function would stop working. It was initially not a concern at the time and the project moved on, but later it had a huge effect and it was soon found out that the connection to the Cortex API would freeze causing all responses from the API to freeze causing the entire WebSocket to stall.

After many hours of searching through this issue, it was later discovered that the plugin would call a non-existing function causing the Cortex response thread to get stuck. This issue was fixed once discovered, completing the profile setup. The code below was one of the causing reasons that called an event handler that had no functions attached to it.

```
else if (action == "delete")
{
    DeleteProfileOK(this, profileName);
}
```

2.5.3.1.6. Game World

The game world is a simple 3D environment with flat terrain land where players can try and test the various spells. Objects are to be spawned and utilized to portray the utility of the spells. This is the main scene where all the actions happen in the game.



2.5.3.2. Magic Development Stage

2.5.3.2.1. Data Subscription

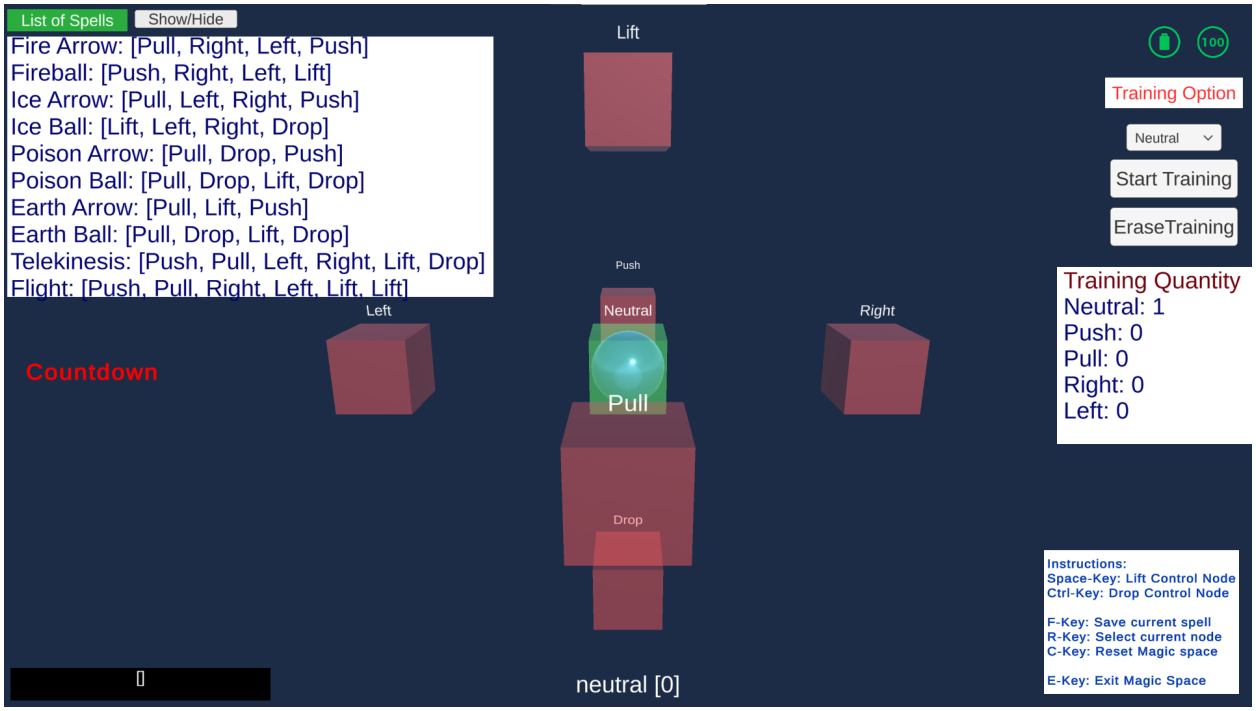
Before moving on to the magic side of the project, the data has to be subscribed first. Subscribing the data means that the game will be able to obtain real-time simplified data of the mental commands of the player. The data has the following attributes: action, power, and time. The action is what kind of mental command it is, power is the intensity of the mental command, and the time is for when this data was taken. Using these data, we are able to send signals to the various aspects of the magic space.

```
1 reference
public void Subscribe()
{
    Debug.Log("Data Subscription request sent");
    _dsManager.SubscribeMoreData(GetStreamsList());
}

1 reference
public void UnSubscribe()
{
    Debug.Log("Data Unsubscription request sent");
    _dsManager.UnSubscribeData(GetStreamsList());
}
```

2.5.3.2.2. Magic Space

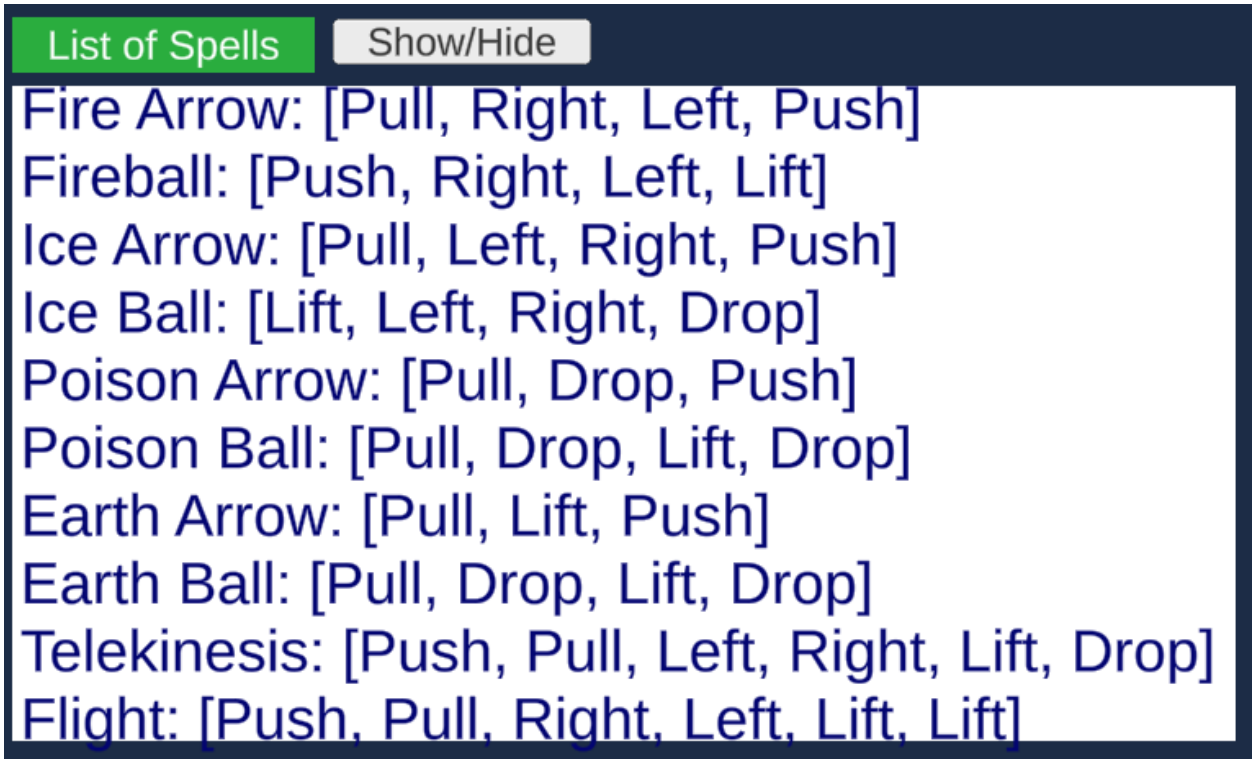
The magic space is a game object that also exists deactivated in the game world as it would be a waste to switch between scenes to simply use magic.

Before explaining this, scene needs some explanation.



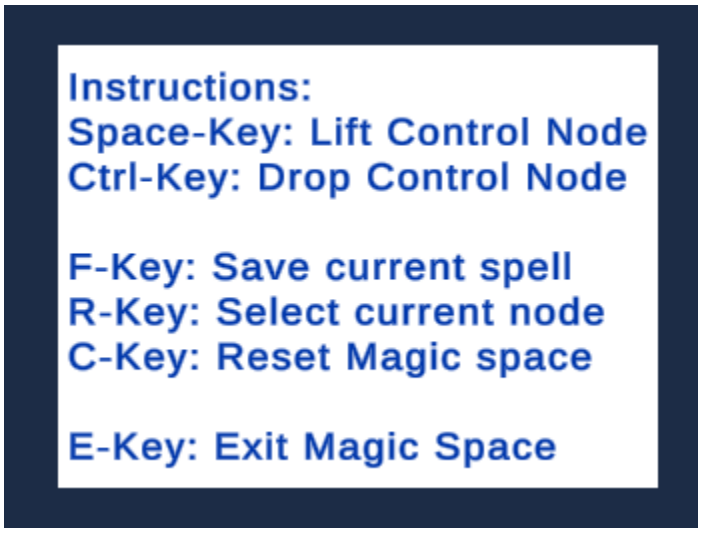
The First thing that needs to be described are the control nodes and the direction(symbol) nodes. The control node is the sphere in the center within the green cube, while the symbol nodes are the red cubes that are used to formulate the spell to be used by the player.

On the top left-hand side of the screen is a formula sheet for casting spells. They show the list of combinations that are required to perform the spell. This sheet can be hidden away with a click of a button.

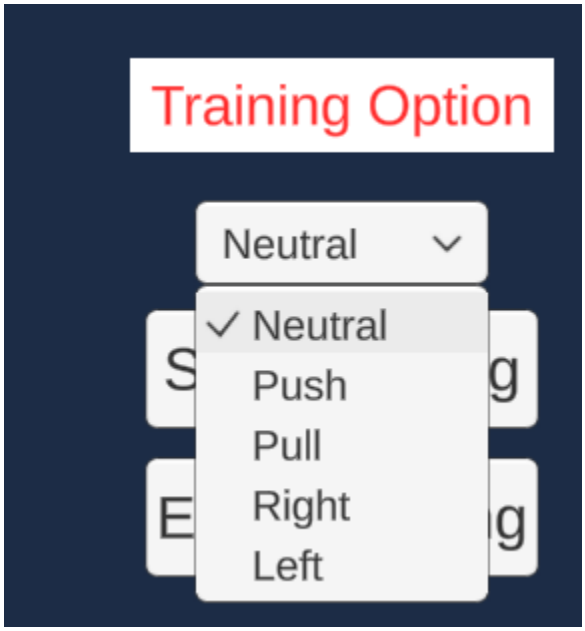



```
public void ShowHide()
{
    listDisplayed = !listDisplayed;
    content.SetActive(listDisplayed);
}
```

On the bottom right side of the screen are the control's instructions for the player to understand how to control the magic space.



On the top right section, there are a few training options where the player can select a specific command from one of the 5 options to train in or delete.



2.5.3.2.3. Magic Training

Once the desired command is selected, the player can either start or erase the training progress of the command specified. Once the training is started the countdown begins.

Here is the code for starting the training and has all the necessary parameters restarted to prevent any previous actions from affecting it, and cancel any ongoing training in progress.

As for the delete action, it sends a request right away to create all training data for the specific mental command.

```
0 references
public void BeginTraining()
{
    if (_isTraining)
    {
        _subscribeTrain.ResetTraining(curTrainingAction);
    }
    _isTraining = true;
    countTrained = 0;
    powerSum = 0.0;
    trainingStartTime = currentTime;
    curTrainingAction = trainingChoice.captionText.text.ToLower();
    _subscribeTrain.StartTrain(curTrainingAction);
    trainingTarget = RecognizeSymbol(curTrainingAction);
    _controlNode.Training(trainingTarget);
}

0 references
public void DeleteAction()
{
    _subscribeTrain.DeleteTrain(trainingChoice.captionText.text.ToLower());
    _numTraining.ChangeValues(trainingChoice.captionText.text.ToLower(), 0);
}
```

Right below the formula sheet is the countdown section where the countdown for the remaining time in training the command is displayed. Once the countdown is complete, the training is stopped and the next steps are made


```
1 reference
private void Countdown()
{
    if(currentTime - trainingStartTime >= trainingTimeLimit)
    {
      StopTraining();
      _countdown.text = 0.ToString();
    }
    else
    {
      _countdown.text = ((int)(trainingTimeLimit-(currentTime-trainingStartTime))).ToString();
    }
}
```

The stopping of the training results in the score of how well the training was done and two options. One to accept the training and the other to reject it.

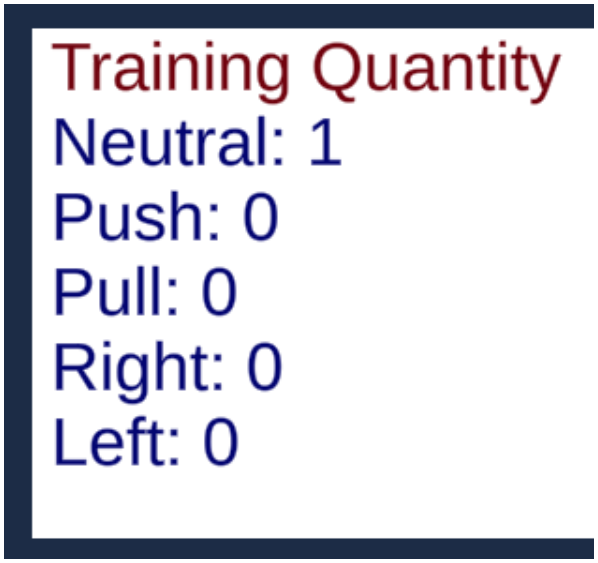
Accepting the training would add it to the profile while the reject does the opposite.


```
public void StopTraining()
{
    Debug.Log("Training Complete");
    double score = powerSum / countTrained * 100;
    _trainingResult.DisplayScore(score);
    _trainingResultUI.SetActive(true);
    _isTraining = false;
    _controlNode.StopTraining();
}

0 references
public void AcceptTraining()
{
    _subscribeTrain.StopTrain(true);
    ReturnToDefault(false, true);
}
0 references
public void RejectTraining()
{
    _subscribeTrain.StopTrain(false);
    ReturnToDefault(false, true);
}
```

When the choice is made, the trained profile is saved and the number of training done for each mental command is requested. This happens for every training that is done.

```
if (_trainingResultUI.activeSelf && butPressed)
{
    _trainingProcessing.SaveCurProfile(_trainingProcessing.StaticHeadset.HeadsetID);
    _trainingResultUI.SetActive(false);
    _subscribeTrain.GetTrainedActions();
}
```

These are the displayed list of the number of times a mental command was trained in. This program required a completely custom function made from scratch to send the request to and from the Cortex API as the Emotiv/unity-plugin did not have any related function.



Request

```
3 references
public void GetTrainedActions()
{
    if (!TrainingProcessing.Instance.IsProfileConnected())
        return;
    Debug.Log("Training Data request sent");
    _bciTraining.GetTrainedSignatureActions("mentalCommand");
}
```

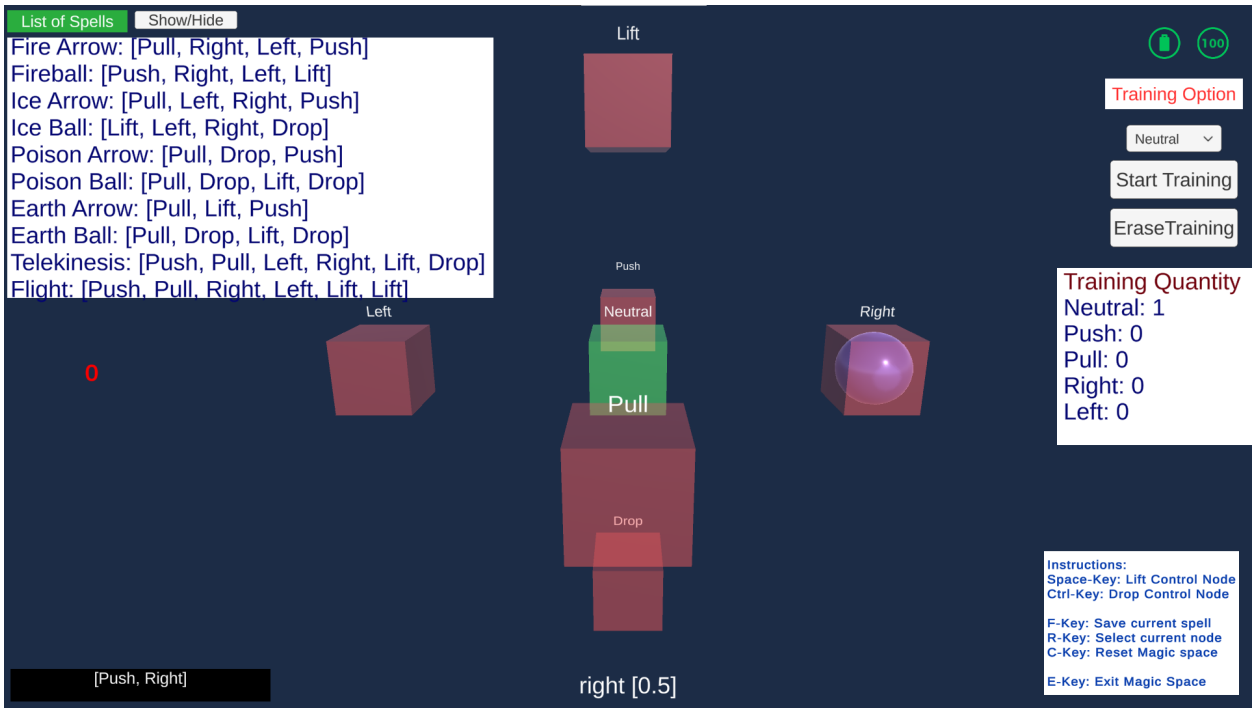
Receive

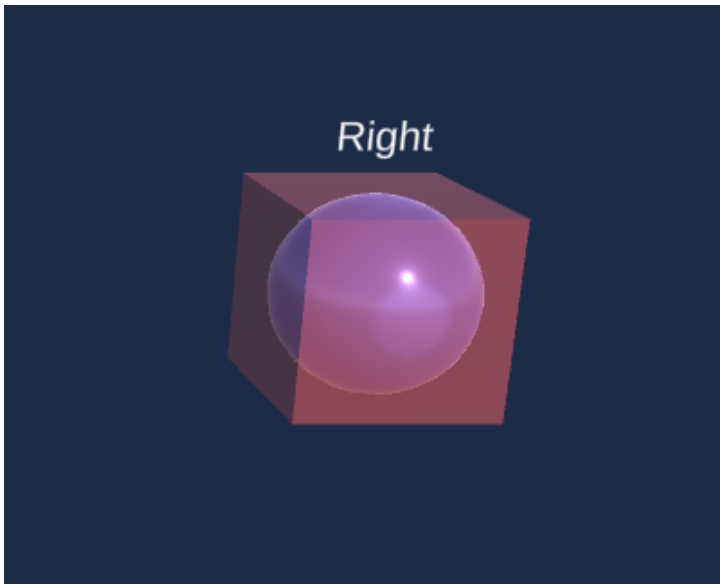
```
1 reference
public void LoadTrainingData(JObject data)
{
    foreach (JToken d in data["trainedActions"])
    {
        tmp[d["action"].ToString()] = (int) d["times"];
        ChangeValues(d["action"].ToString(), (int) d["times"]);
    }
}
```

2.5.3.2.4. Magic Casting

In this space, the original intention was that the player can control the control node through any of the 6 mental commands of their choosing, push, pull, right, left, lift, and drop. However, due to the unforeseen discovery that only 4 mental commands can be stored in a profile at a time, the Lift and drop commands were removed from the Cortex API request. Instead, they were converted into manual keybinds so that there can still be 3-Dimensional manipulation of objects including the control node.

The objective to cast the spell is to move the control into the symbol nodes





To get this to work, a function that can be called based on received data and manual input was made so that it can accurately call the correct function in the control node which is translated into its movement data.

```
1 reference
public void Move(Spells.SpellSymbols sym, double multiplier)
{
  switch(sym)
  {
    case Spells.SpellSymbols.Neutral:
      Neutral();
      break;
    case Spells.SpellSymbols.Push:
      Push(((float)multiplier));
      break;
    case Spells.SpellSymbols.Pull:
      Pull(((float)multiplier));
      break;
    case Spells.SpellSymbols.Right:
      Right(((float)multiplier));
      break;
    case Spells.SpellSymbols.Left:
      Left(((float)multiplier));
      break;
    case Spells.SpellSymbols.Lift:
      Lift(((float)multiplier));
      break;
    case Spells.SpellSymbols.Drop:
      Drop(((float)multiplier));
      break;
    default:
      Neutral();
      break;
  }
}
```

There were also other considerations to be made such as, what if a player accidentally enters a different command? What if it was on purpose?

This code is the push command that is similar to the other 5 commands that consider most scenarios of how the player can manipulate the control node so that the node does not miss the desired node as long as the correct mental command is maintained.

```
1 reference
public void Push(float multiplier = 0.5f)
{
  lock(_object)
  {

    if (push && pushTrain && (!z_lock || _isTraining))
    {
      _timerCounter_move_back = _timerCounter_reset = 0;
      if(!inNeutral)
      {
        gameObject.transform.localPosition = new Vector3(_position.x, _position.y, gameObject.transform.localPosition.z);
        x_lock = y_lock = true;
      }
      _rigid.velocity = new Vector3(0, 0, multiplier*max_speed);
    }
    else if (!push)
    {
      _timerCounter_move_back = _timerCounter_reset = 0;
      if(_rigid.velocity.z > 0)
      {
        _rigid.velocity = new Vector3(_rigid.velocity.x, _rigid.velocity.y, 0);
      }
    }
  }
}
```

This function detects what node the control node is inside of.

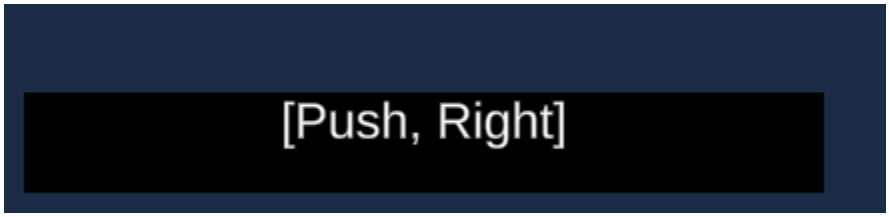
```
0 references
private void OnTriggerEnter(Collider other)
{
    Spells.SpellSymbols sym = TranslateNode(other.gameObject.name);
    if (sym == Spells.SpellSymbols.Neutral)
    {
        CurrentNode = Spells.SpellSymbols.Neutral;
        inNeutral = true;
        _rigid.velocity = Vector3.zero;
        ResetPos();
        return;
    }
        CurrentNode = sym;
}
0 references
private void OnTriggerExit(Collider other)
{
    CurrentNode = Spells.SpellSymbols.None;
    Spells.SpellSymbols sym = TranslateNode(other.gameObject.name);
    if (sym == Spells.SpellSymbols.Neutral)
    {
        inNeutral = false;
    }
}
```

The node is then translated into the correct symbol terms.

```
2 references
private Spells.SpellSymbols TranslateNode(string node)
{
    switch(node)
    {
        case "NeutralNode":
            return Spells.SpellSymbols.Neutral;
        case "PushNode":
            return Spells.SpellSymbols.Push;
        case "PullNode":
            return Spells.SpellSymbols.Pull;
        case "RightNode":
            return Spells.SpellSymbols.Right;
        case "LeftNode":
            return Spells.SpellSymbols.Left;
        case "LiftNode":
            return Spells.SpellSymbols.Lift;
        case "DropNode":
            return Spells.SpellSymbols.Drop;
        default:
            return Spells.SpellSymbols.None;
    }
}
```

In the magic casting space, the player can move the control nodes around and add select nodes to the formula. To add the nodes, the player must move the node into the desired node and press the R key in order to add it to the formula.

Once the control node is correctly placed inside the desired symbol node, the player can lock the node into the spell pattern and display it.



When the R-Key is pressed, the node will be locked in as long as the node is none or neutral.

```
1 reference
public void LockInNode()
{
    lock(_object)
    {
        if(_controlNode.CurrentNode == Spells.SpellSymbols.None)
        {
            _controlNode.ResetPos();
            return;
        }
        else if(_controlNode.CurrentNode == Spells.SpellSymbols.Neutral)
        {
            _controlNode.ResetPos();
            return;
        }
        else
        {
            if(!AddSymbol(_controlNode.CurrentNode))
            {
                ResetFormula();
            }
            _controlNode.ResetPos();
        }
    }
}
```

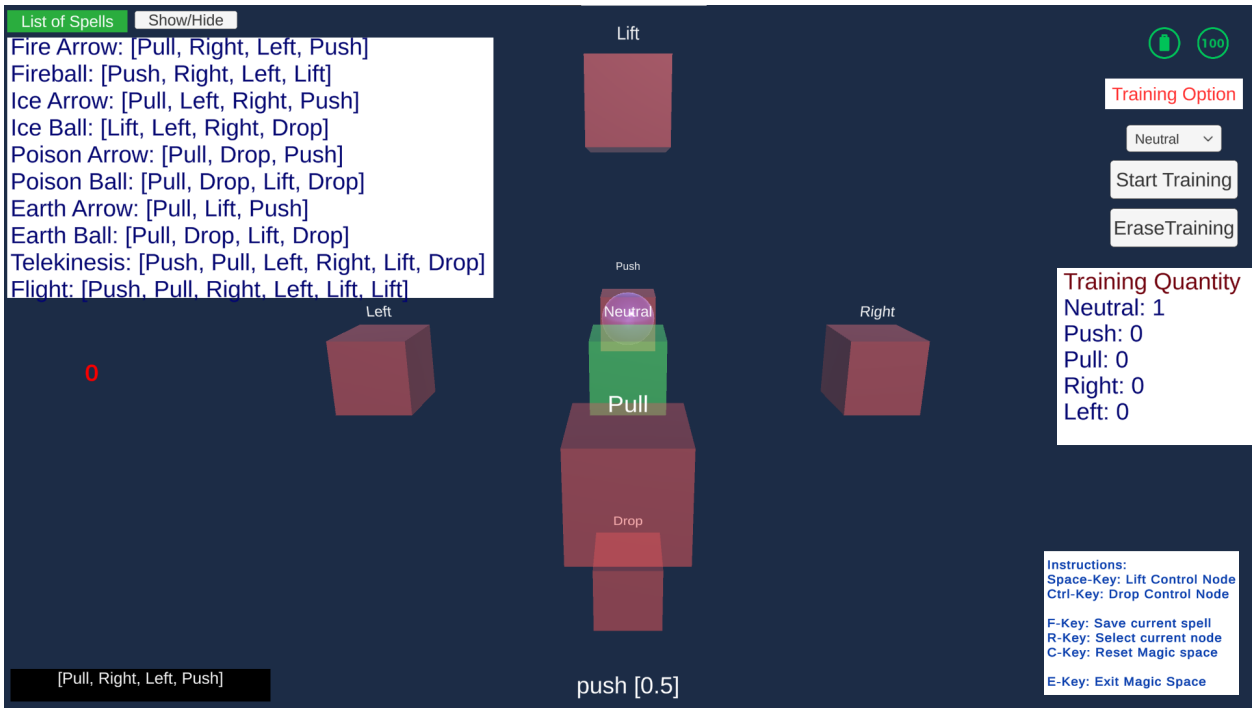
This function determines whether the symbol should be added as only the correct pattern will be allowed and any incorrect pattern will be erased and the player would have to restart from scratch, making sure that the player stays careful when casting the spell.

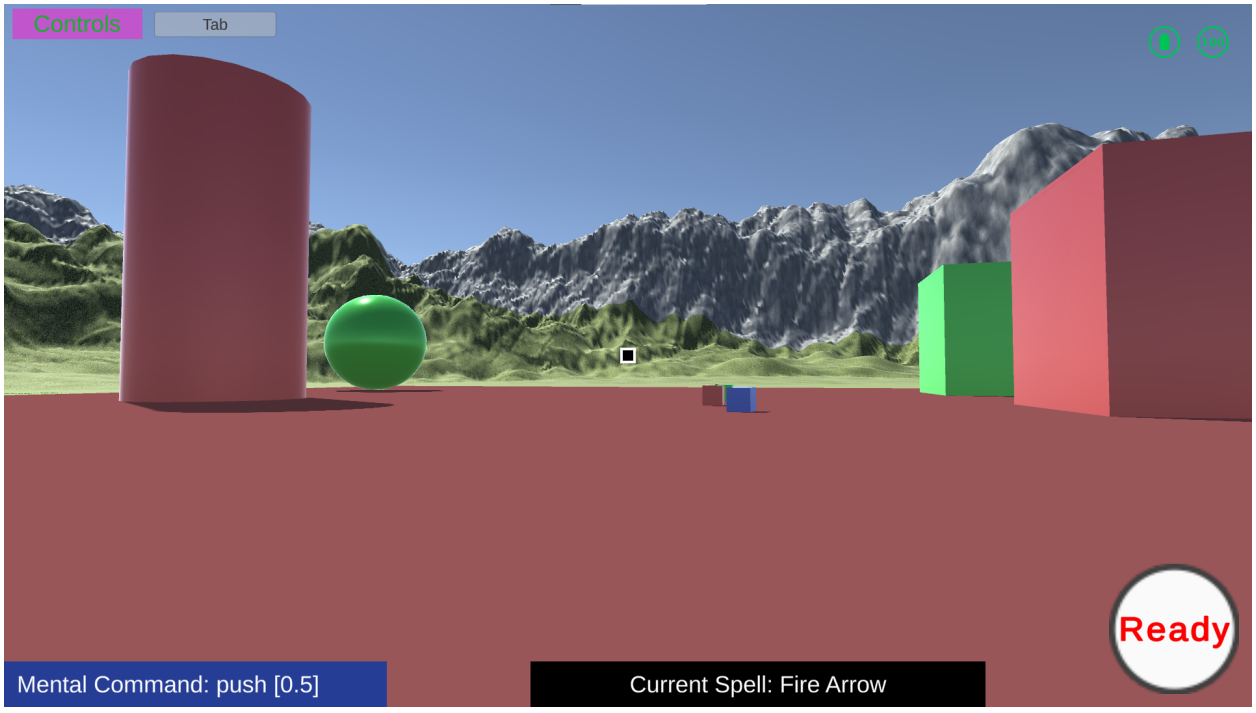
```
1 reference
private bool AddSymbol(Spells.SpellSymbols symbol)
{
  if (formula == null)
  {
    currentMatching = new Dictionary<string, List<Spells.SpellSymbols>>(Spells.Formula);
    formula = new List<Spells.SpellSymbols>();
  }
  bool matchFound = false;
  List<string> removeItems = new List<string>();

  //Check if recently added symbol fits the pattern
  foreach (KeyValuePair<string, List<Spells.SpellSymbols>> entry in currentMatching)
  {
    //Debug.Log(entry);
    int i = formula.Count;
    if(i >= entry.Value.Count)
    {
      continue;
    }
    else
    {
      if (entry.Value[i] == symbol)
      {
        matchFound = true;
      }
      else
      {
        removeItems.Add(entry.Key);

      }
    }
  }
  foreach(string k in removeItems)
  {
    currentMatching.Remove(k);
  }
```

```
  if(matchFound)
  {
    formula.Add(symbol);
    return true;
  }
  return false;
}
```

Once the formula is the same as the pattern, the player can then press the F key in order to send it to the game world to be stored and utilized. Only one spell can be stored at a time.

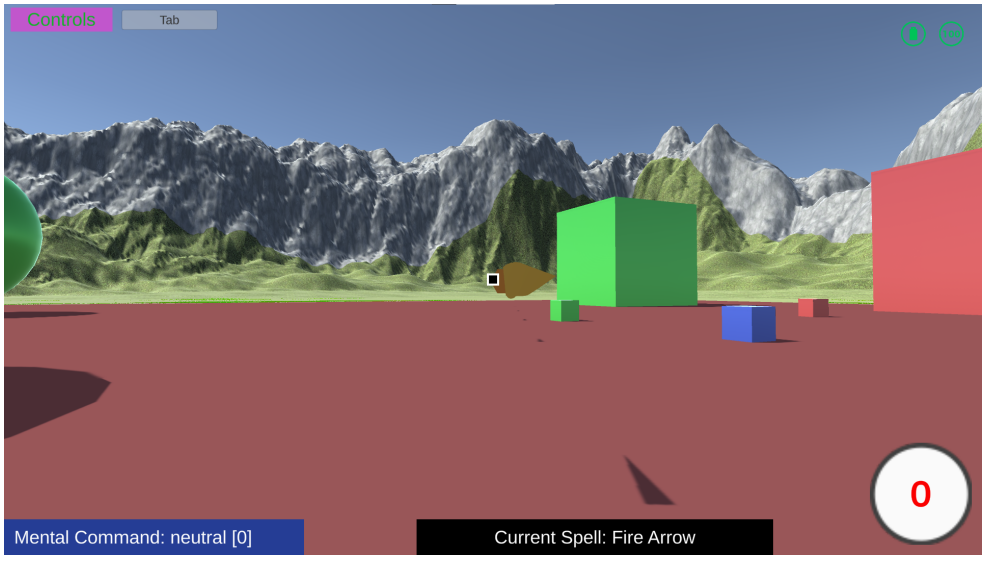
For example, this is the Fire Arrow spell pattern



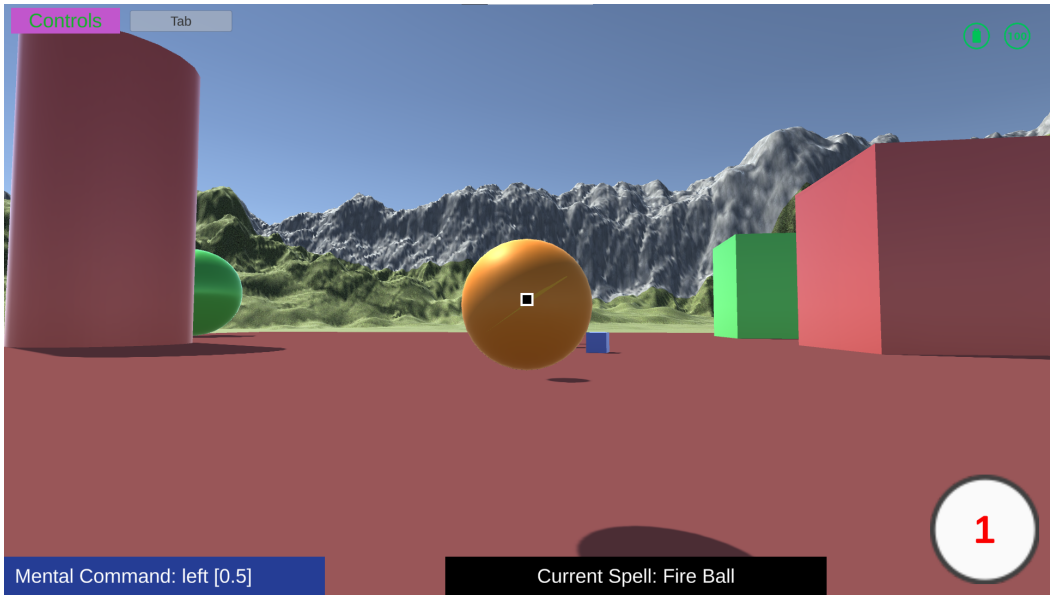


2.5.4. Game Development Stage

Once the spell is stored, the game world is reactivated with the magic space hidden once more. This time the spell that was stored is displayed at the bottom of the screen. Then the spell can be activated with the left mouse button. On the bottom right it is different from the previous screen where the text “ready” changed to 0 to indicate the cooldown time for this spell. As the spell used here is the “Fire Arrow” spell, it has a 1-second cooldown making it reach 0 very quickly causing this scenario, otherwise, the text would have been “ready” instead.



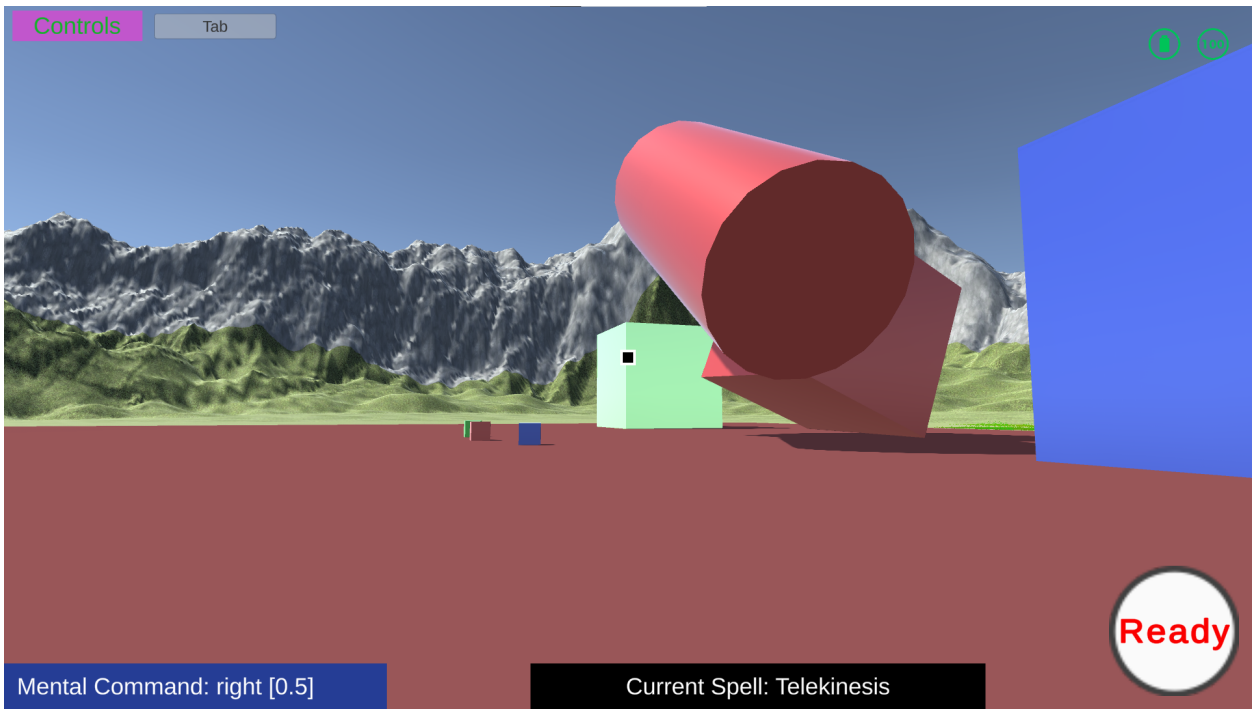
This is the “Fire Ball” spell that works identically with the “Fire Arrow”, but longer cooldown and slower speed.



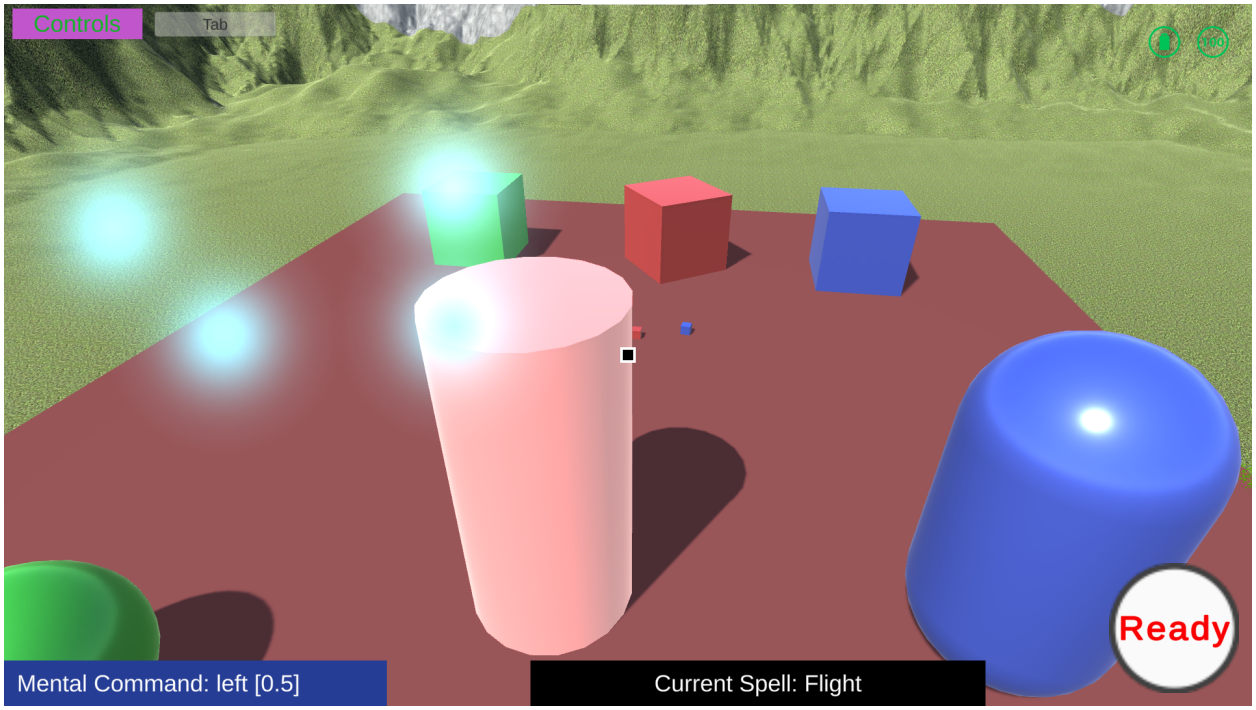
The other Arrows and Ball spells are basically replicas of the “Fire Arrow” and “Fire Ball” spells but in different colours.

Meanwhile, the main aspect of the spell-casting capabilities lies in the spells “Telekinesis” and “Flight”. To indicate their uniqueness, a little particle effect was added to portray the magical aspects of these abilities.

Telekinesis is the ability to move the object that the player is currently looking at and then activate the spell. Once they activate the ability to jump is locked as the lift and drop commands are the manual keys of space and ctrl. The ability to move them in the horizontal axis lies purely on the player’s mental command.

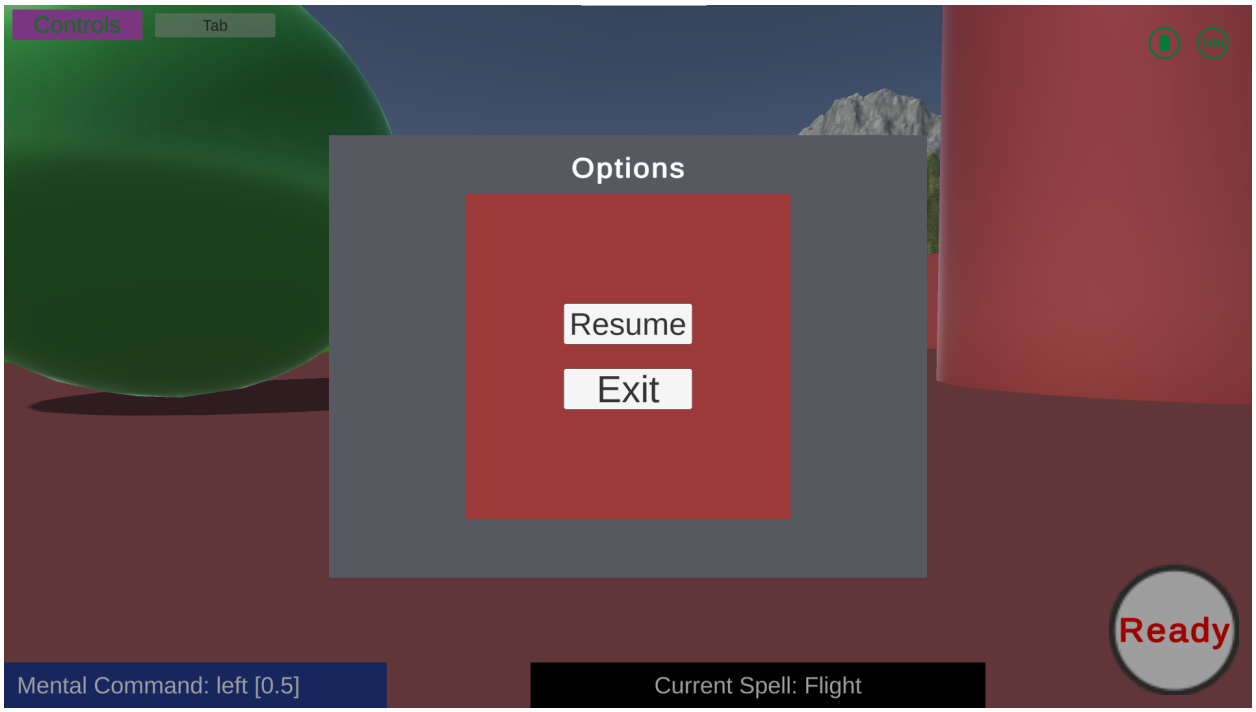


Flight is pretty much the same ability as Telekinesis except the player is moving their own body with it.



2.5.5. End

Then finally when the player finishes, they can enter the options menu to exit the game.



2.6. Testing Details and Results

2.6.1. User Testing

The user testing was difficult as nobody had a Neural Headset to use in their own houses so the test was done using the simulated device option provided in the Emotiv App. The users were all gamers so their computer hardware specifications were of higher quality than average. The testing is separated into 2 parts for version 1 and version 2 as a lot of bugs were fixed between the versions.

2.6.1.1. Game Version 1

2.6.1.1.1. John Lim

User Case
Name: John Lim
Age: 22
Sex: Male
Hobby: Soccer, Games, Videos
Player actions:

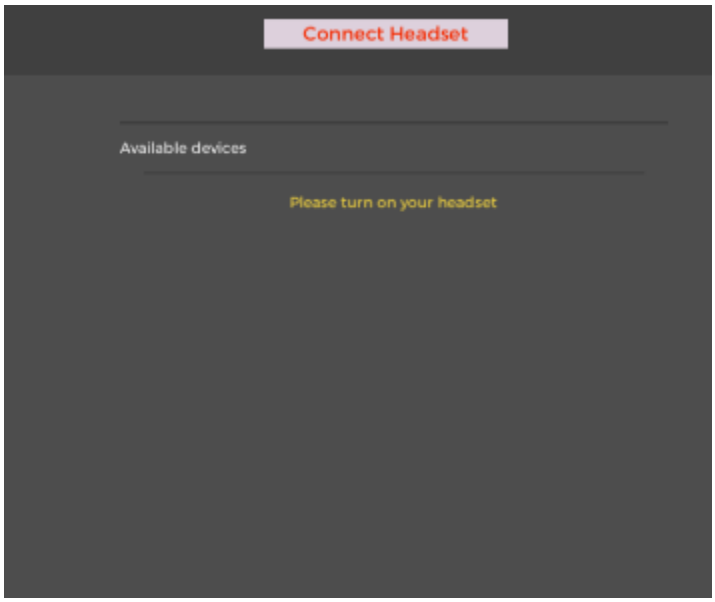
- Turned on the game
- Couldn't continue because he didn't have the client ID and client Secret for login credentials.
- Reluctantly made an account as he didn't like making new accounts
- Had to instruct him on how to get the client id and client secret
- Accidently forgot to copy the client secret so he had to register another application
- went back to the game
- Couldn't continue as he didn't have the Emotiv app installed
- Installed the app
- No headset detected because he didn't add the virtual headset
- Headset appeared after adding the virtual headset
- On the profile, he made a profile and deleted it immediately
    - Found bug where the last profile element cannot be visually removed
- after doing it a few times, he loaded a new profile and entered the game world
- he was confused what to do as he didn't know the controls
- had to tell him the controls
- he entered the magic space
- had to instruct him how to use the virtual device to move the control node
- This one didn't assistance as he figured it out on his own
- He tried things out
- decided to run the longest spell formula
- made a mistake in the middle and got annoyed
- finished the spell
- stored the spell

Test:

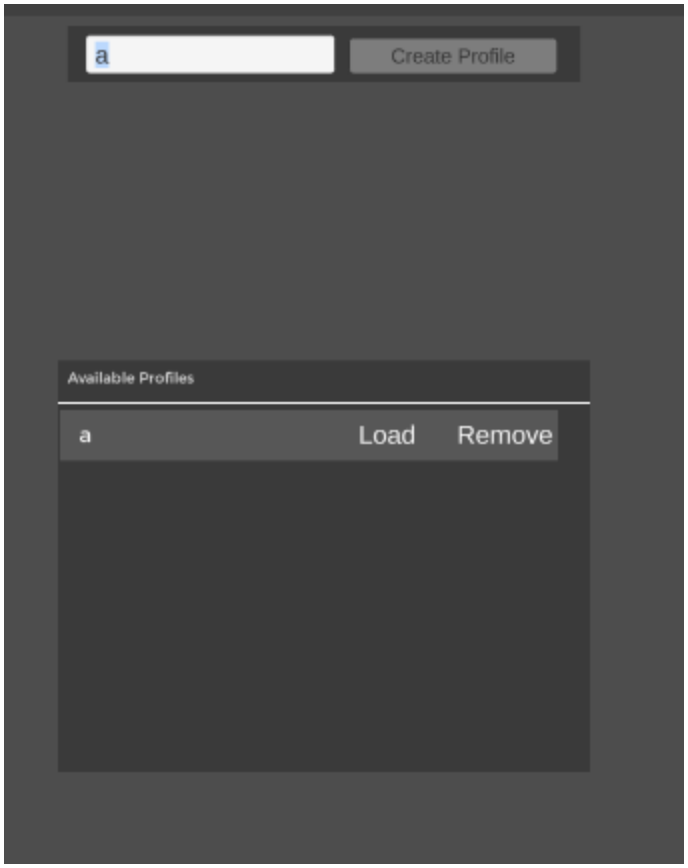
- Run through the game without guidance
- Run through the game with a list of tasks

    - Don’t have any headsets devices
    - Create, delete, load profiles
    - Test all magical abilities
    - Purposefully incorrect magical ability command
- Test questions
    - Did the game feel smooth?
        - Little visual glitches
        - smooth movements
    - How did the usage of headsets feel?
        - Nothing since I used the virtual headsets
    - Was there difficulty in enacting desired actions?
        - No
    - How difficult was it to activate a spell?
        - Feels kind of annoying to do, but interesting at the same time
- Personal feedback
    - Work on the user interface a bit more, what was with the headset and profile page?
    - Why is the game world so empty?
    - Maybe use actual symbols for the spells instead of push and pull
    - I want to test how it feels against real enemies
    - The game lacks depth, where’s the story?

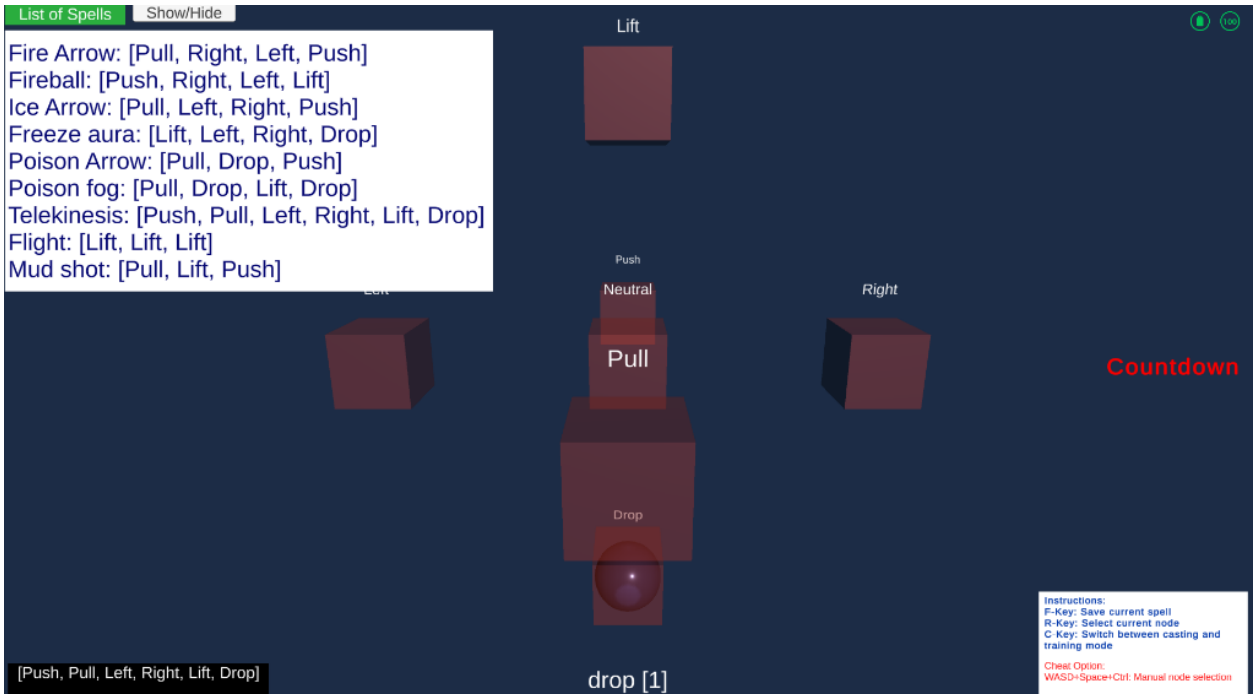
No headset

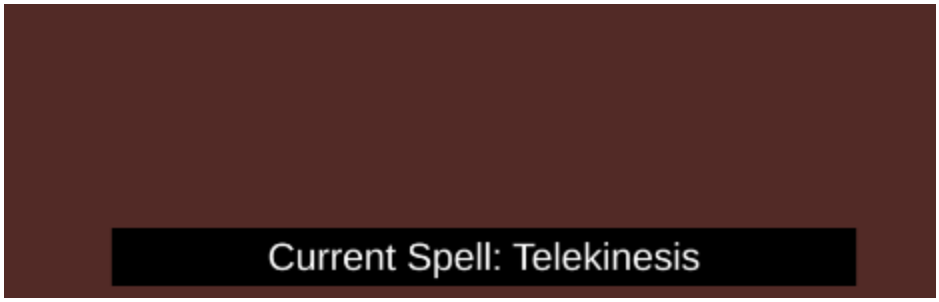


Profile creation and unable to delete the last profile element from the page



Activating the longest spell



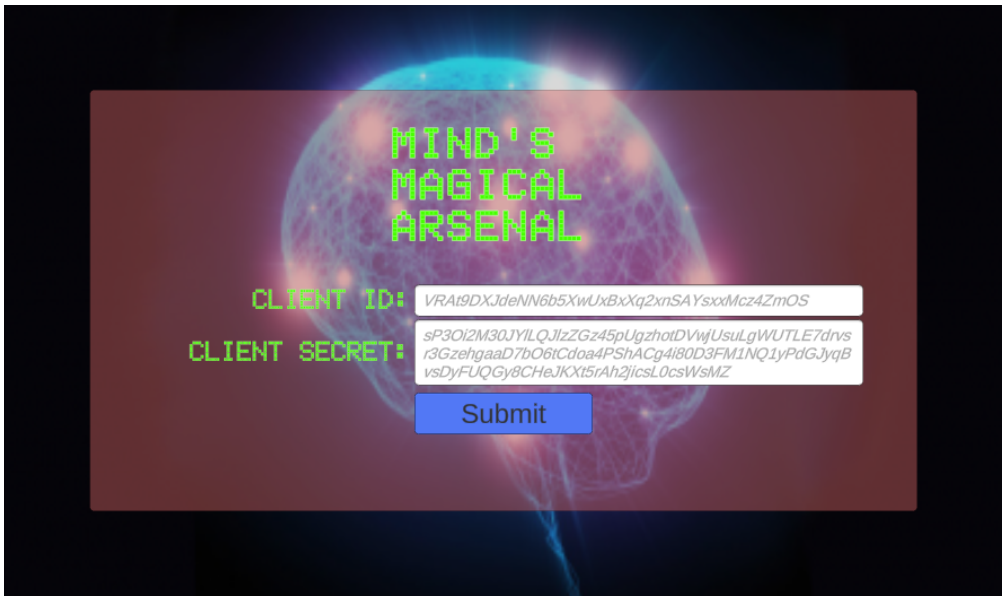


2.6.1.2. Game Version 2

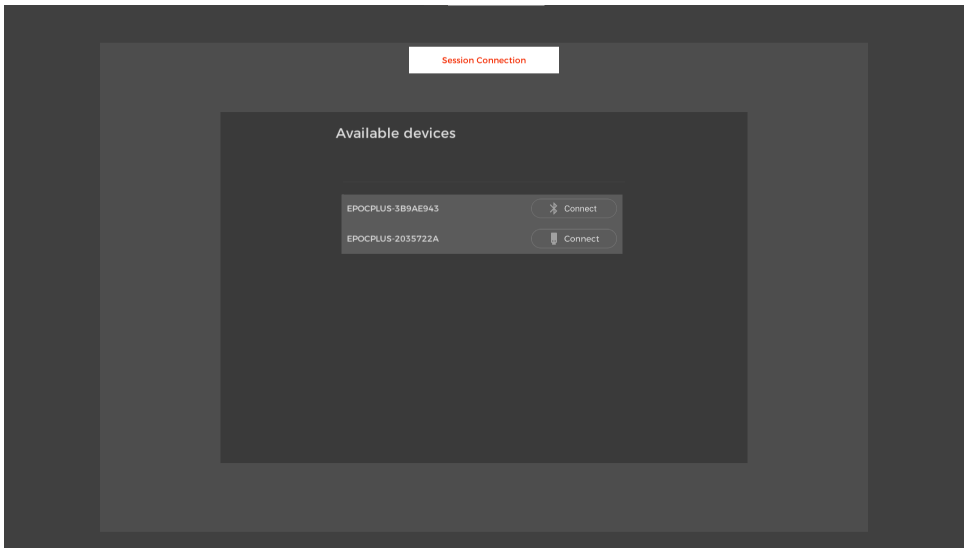
2.6.1.2.1. John Lim

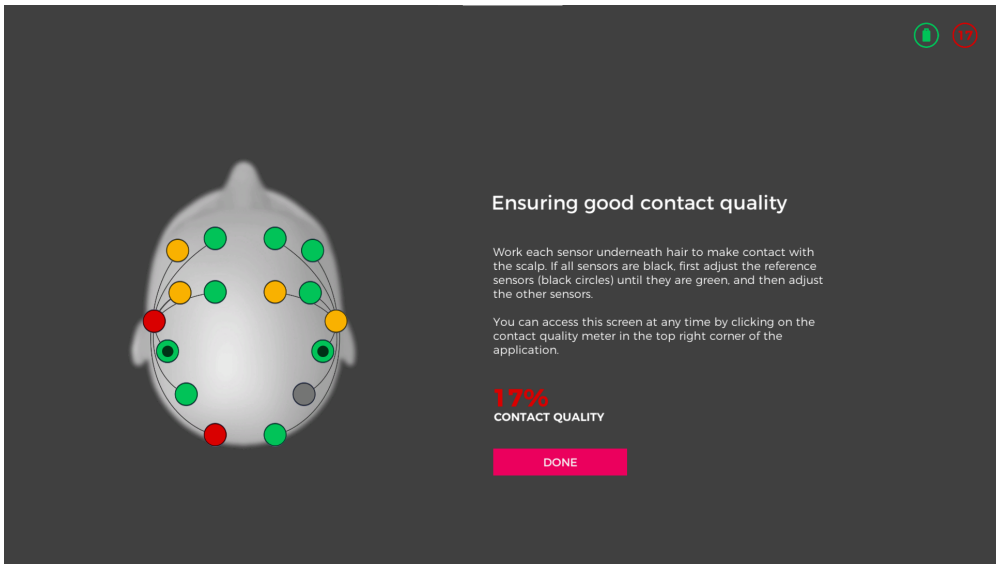
User Case

Name: John Lim
Age: 22
Sex: Male
Hobby: Soccer, Games, Videos
Player actions:

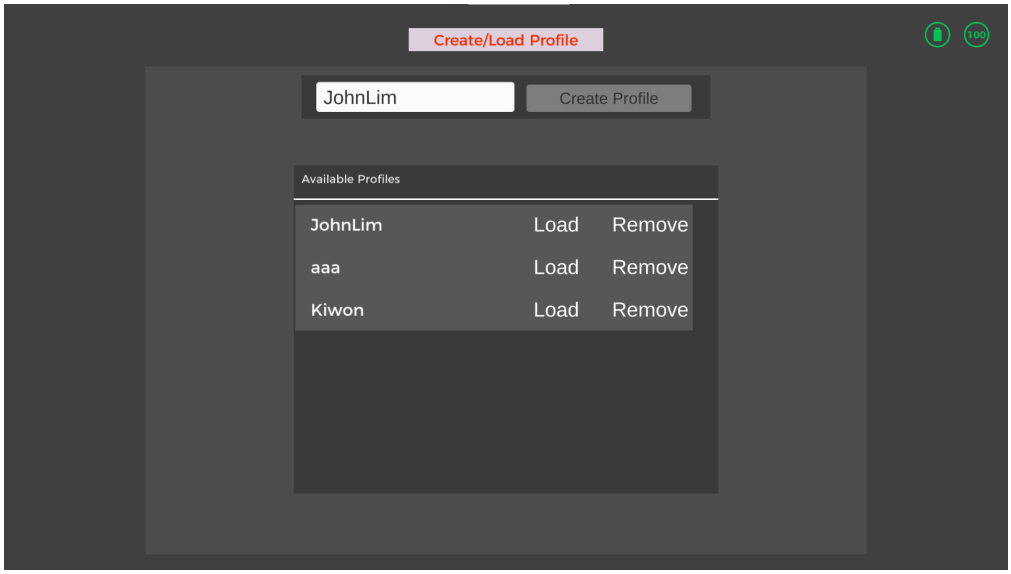
- Seeing how he had to put on the headset with the wet sensors he was reluctant but agreed to
- Wore the headset
- Found it uncomfortable
- Turned on the game
- New most of the steps through the previous test case
- Logged in (Default credentials were used as the tester was present)
  - 

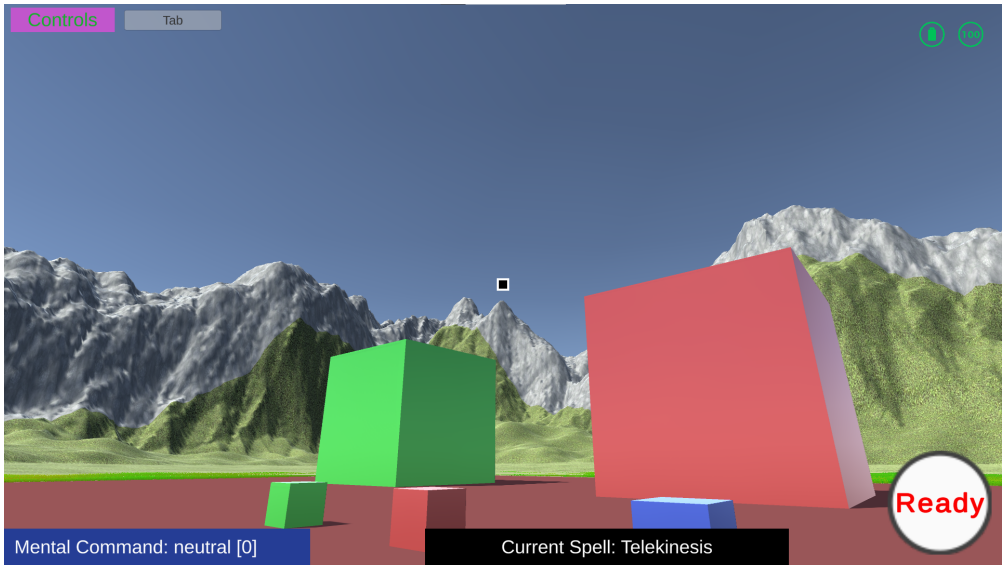
- selected headset

- 

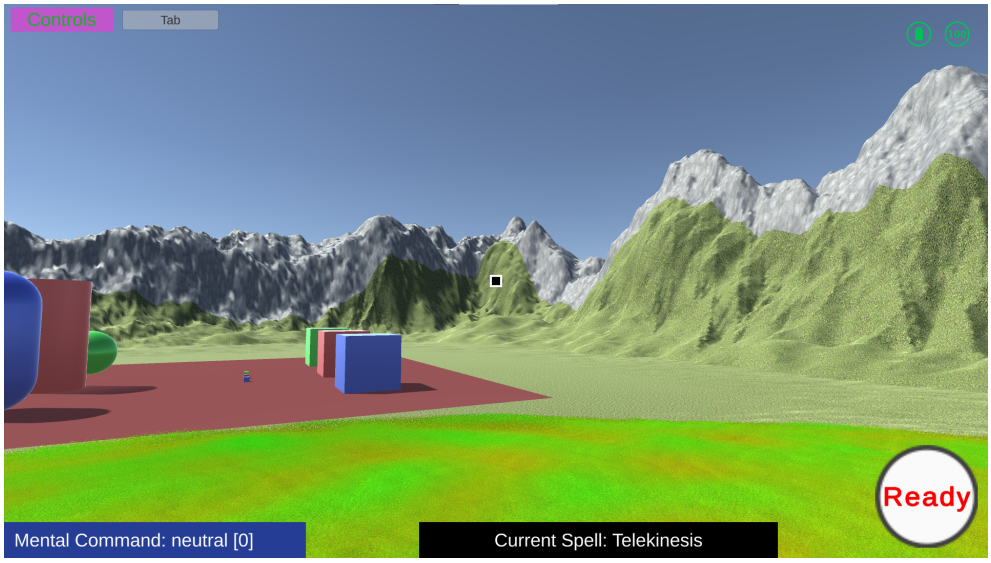
- Checked Contact quality and moved the headset around to make sure they fit
  - 

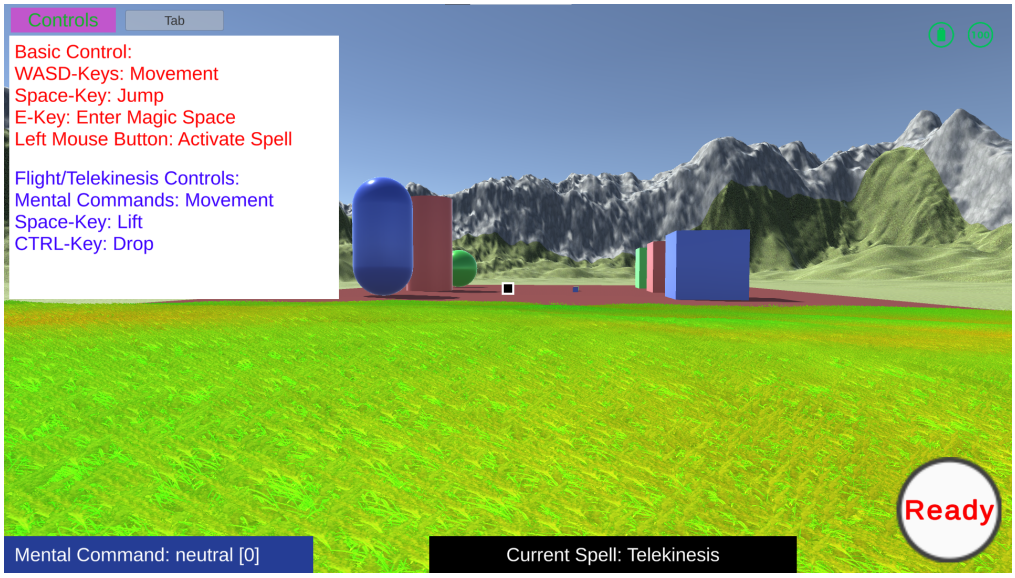
- profile created and selected

- 

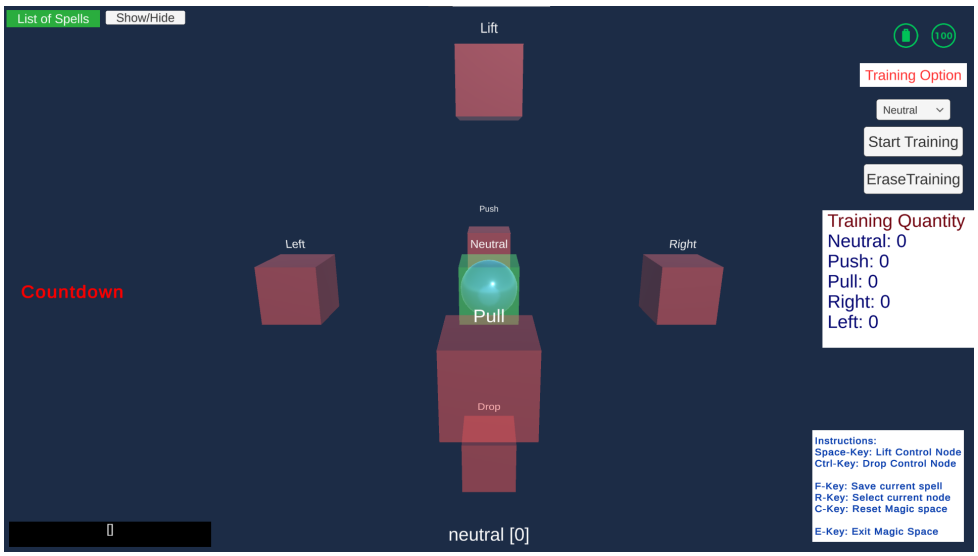
- Surprized by the new visual additions
  - 

- Moved around

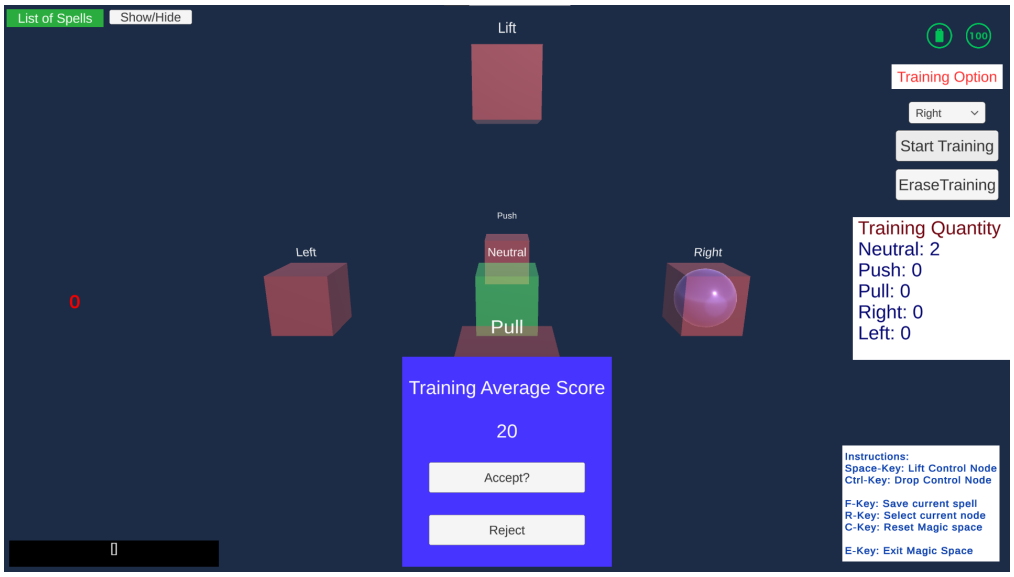
- 

- He was amazed at how much things changed from the last test
- Realized the controls tab
  - 

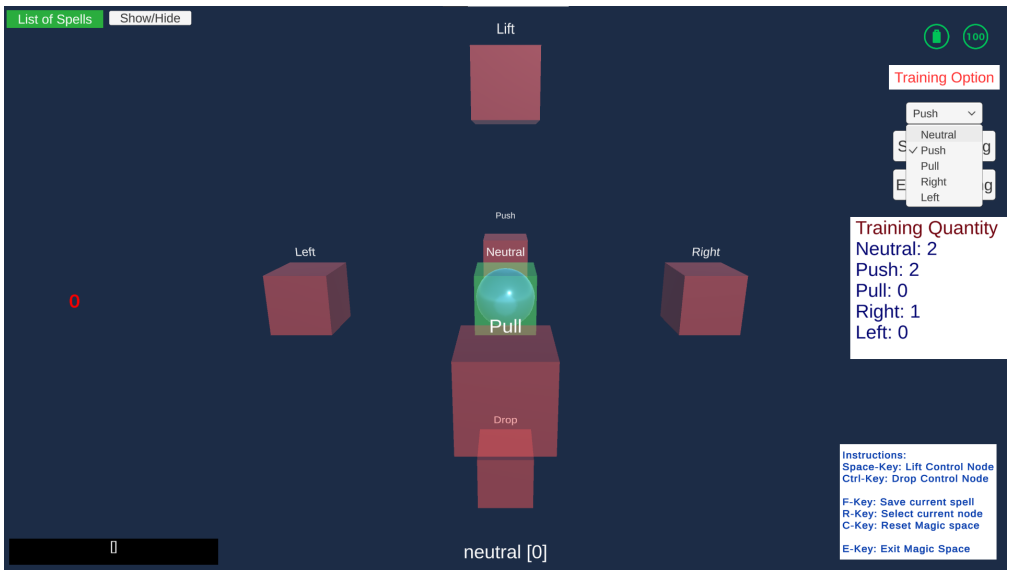
- commented that there were more guiding factors now.
- Entered magic space

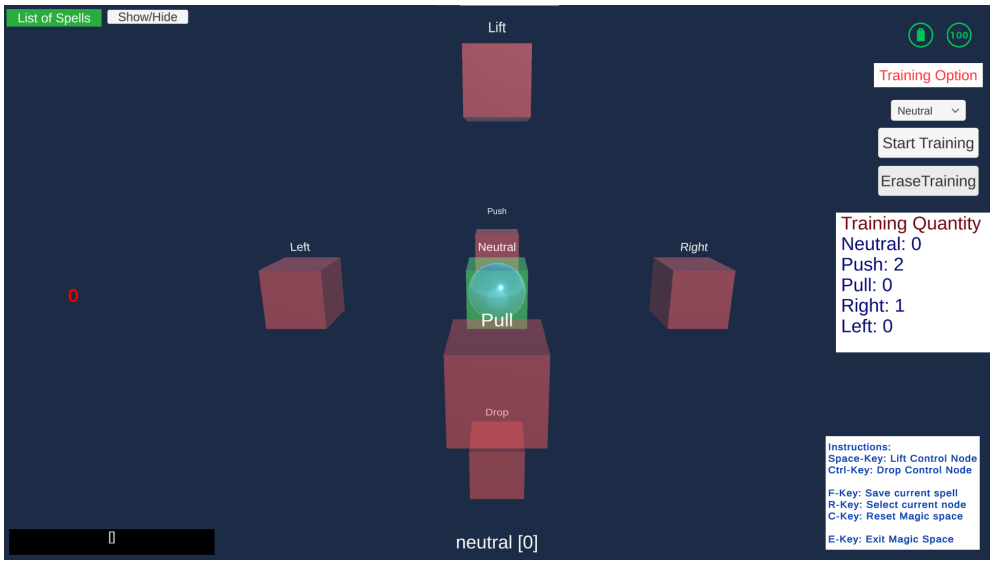
-


- Trained profile

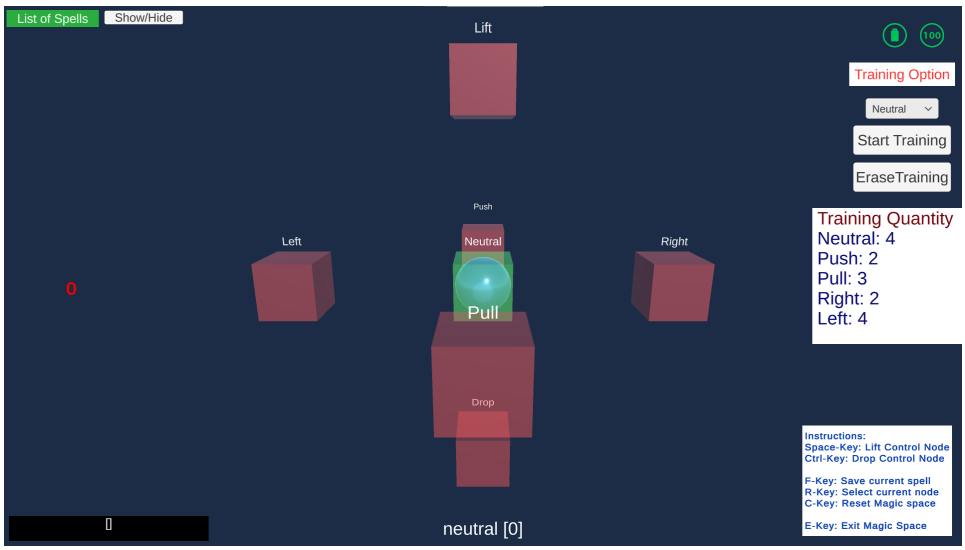


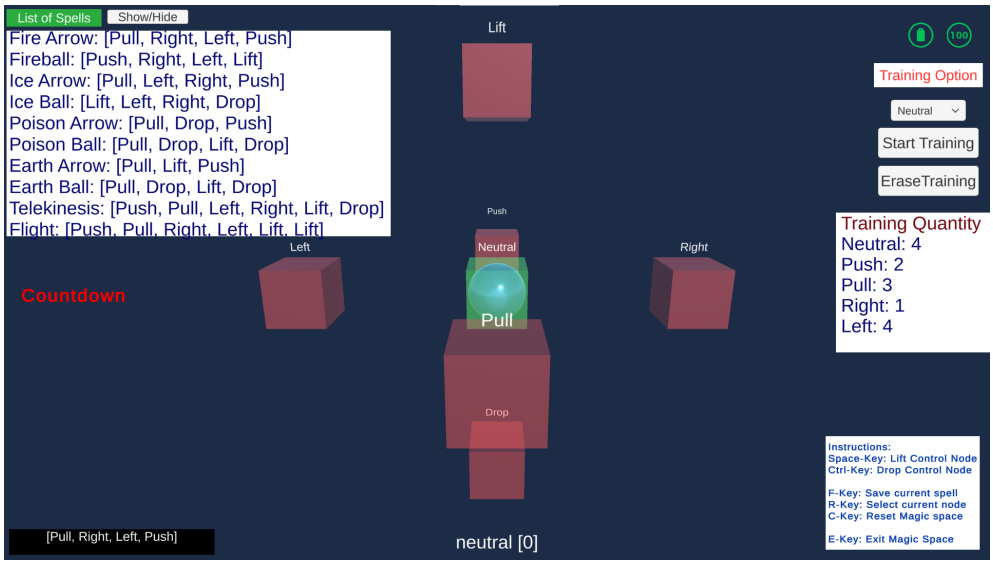
- Where are the lift and drop training options?

- 

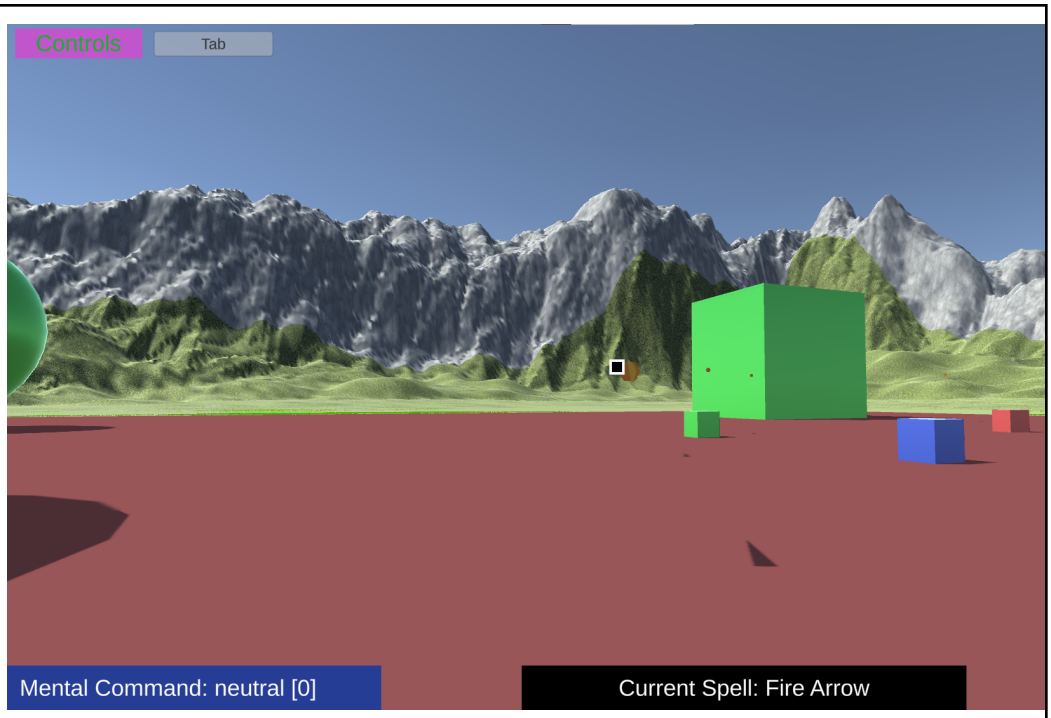
- Why did it change this way?
- Tried the erase training option on neutral
- 

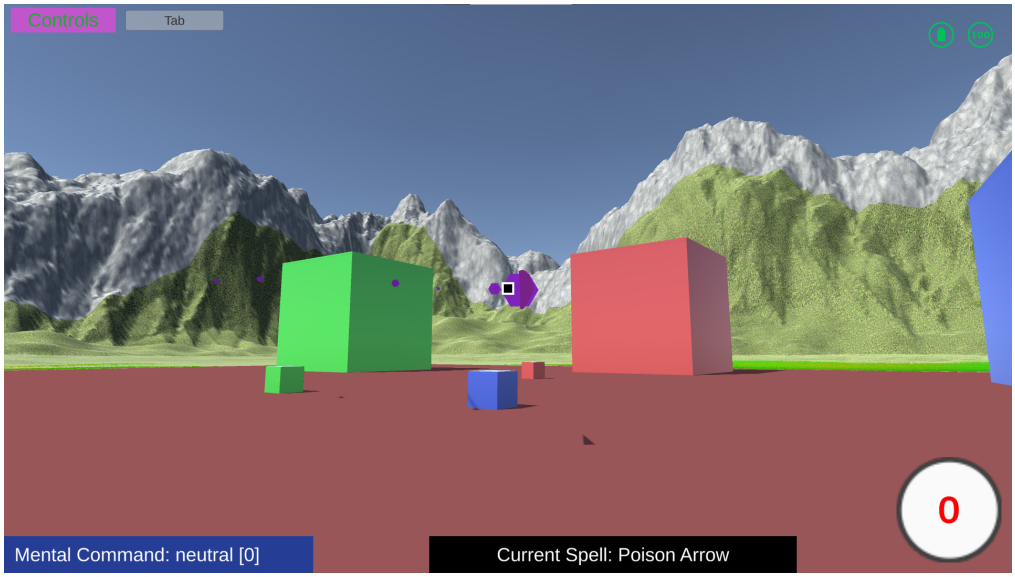
- Had to retrain it
- Trained result

- 

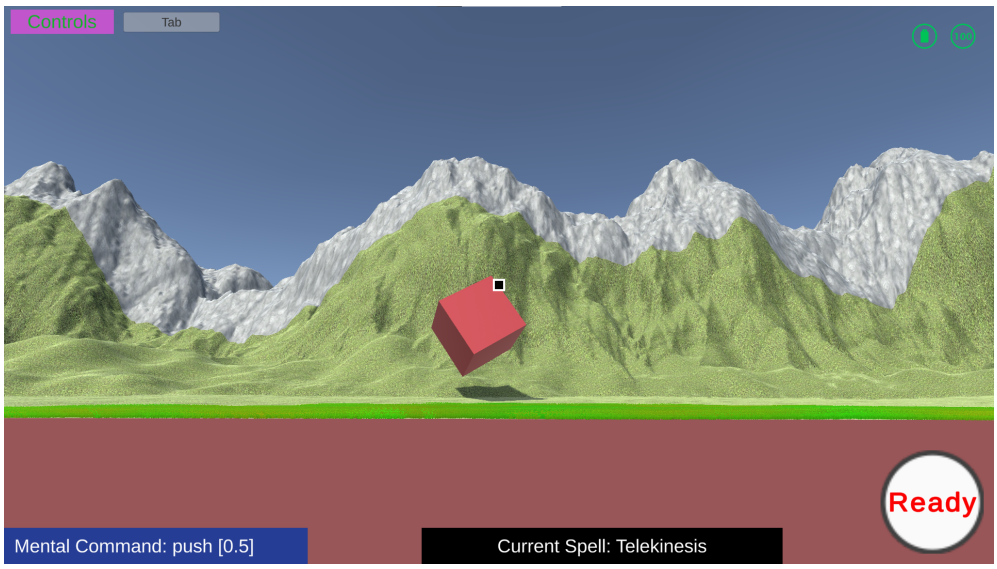
- Trying out pattern
- 

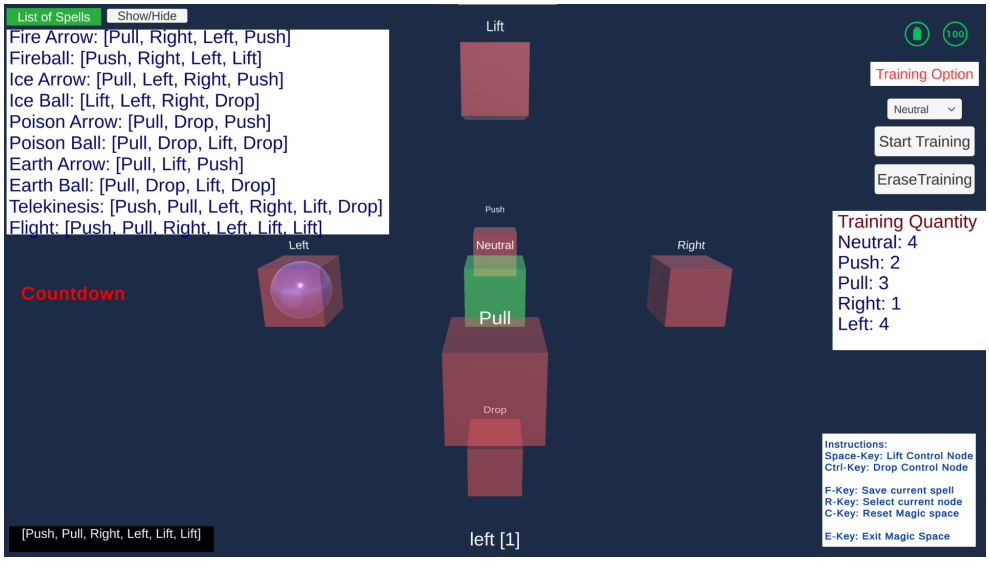
- Trying Fire Arrow

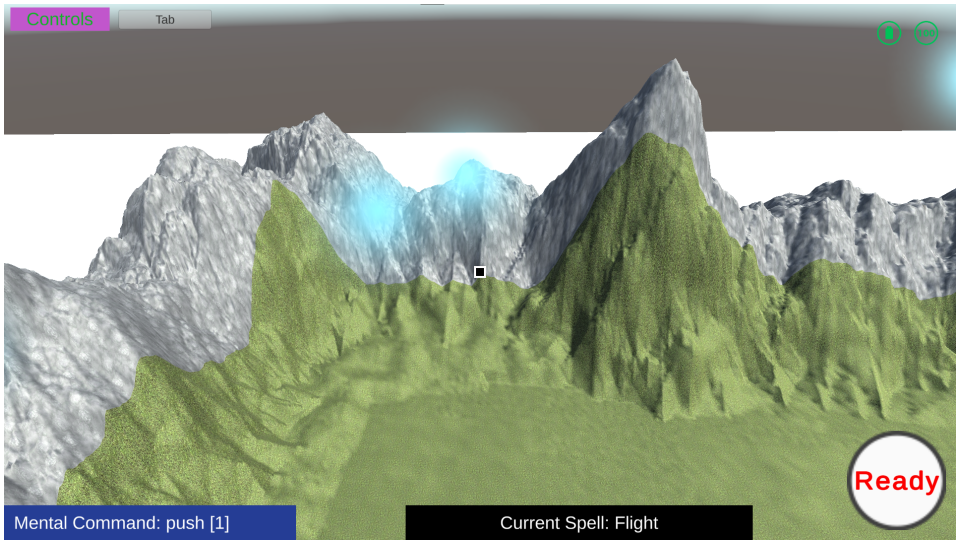
- 

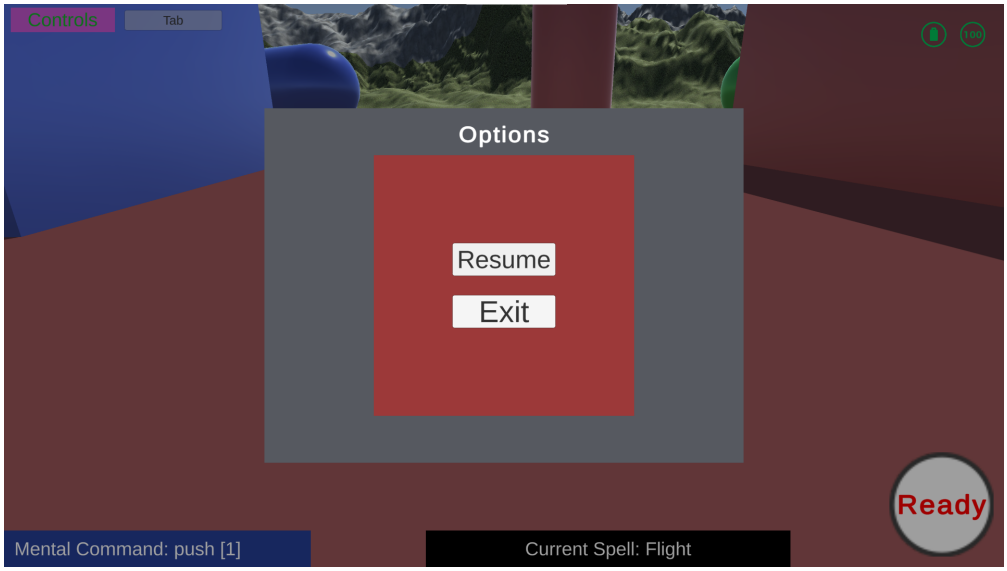
- Why are the arrows so slow? (Arrows were too fast to catch properly
- Trying out poison arrow
  - 

- Trying out Telekinesis
  - Doesn't budge when the command is low powered.
- Stronger attempt

- 

- Trying out Flight
  - 

- 

- Exit Game
  - 

Test:

- Run through the game without guidance
- Run through the game with a list of tasks
  - Create, delete, load profiles
  - Test magical abilities
  - Purposefully incorrect magical ability command
- Test questions
  - Did the game feel smooth?
    - smooth movements
  - How did the usage of headsets feel?
    - Uncomfortable especially the weird sensation as though part of the hair touched seawater
  - Was there difficulty in enacting desired actions?

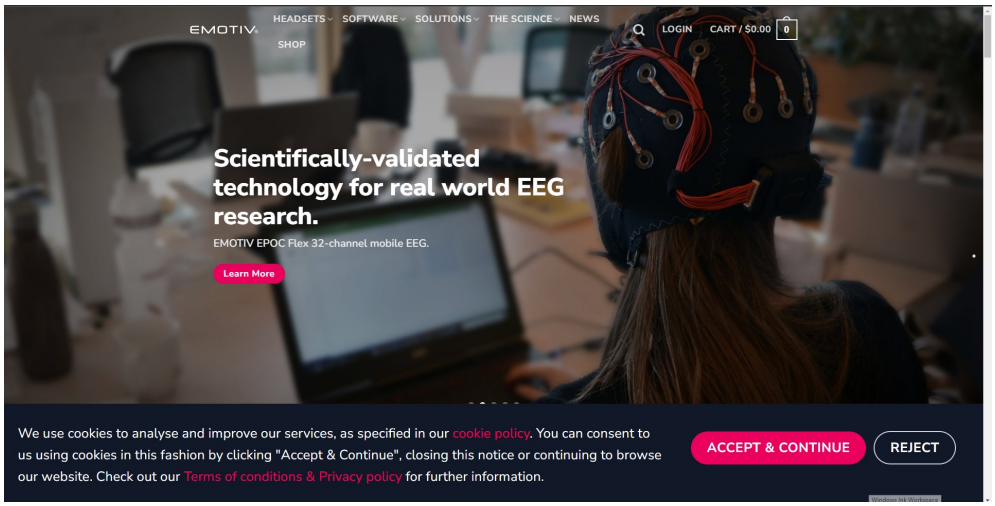
        - No really
    - How difficult was it to activate a spell?
        - Feels kind of annoying to do, but interesting at the same time
- Personal feedback
    - Don’t really want to put that headset on again, but the game was interesting enough for me to put it on
    - I guess work on assigning symbols instead of push and pull like mentioned last time?
    - I want to test how it feels against real enemies
    - The game lacks depth, where’s the story?

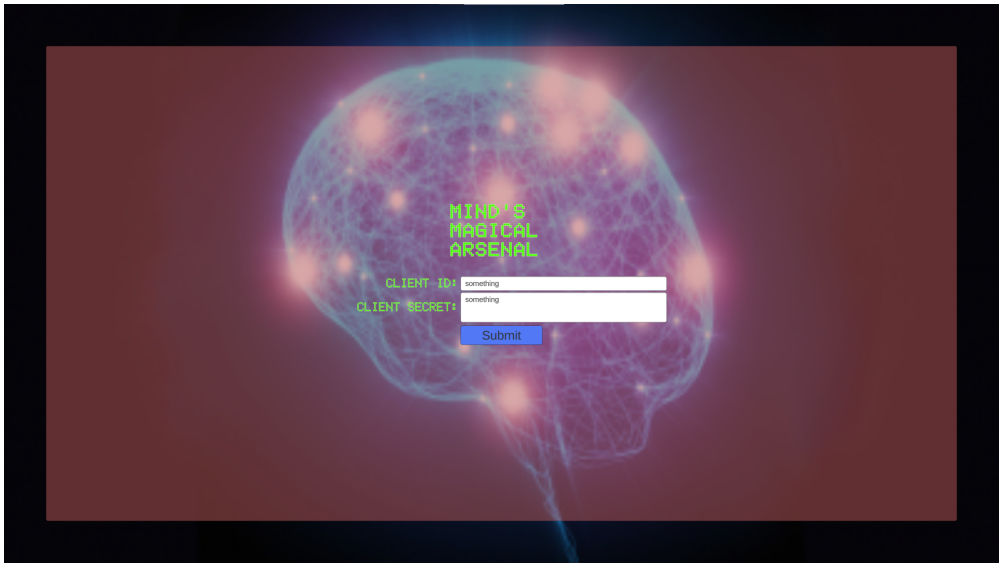
2.6.1.2.2. Charles Huang

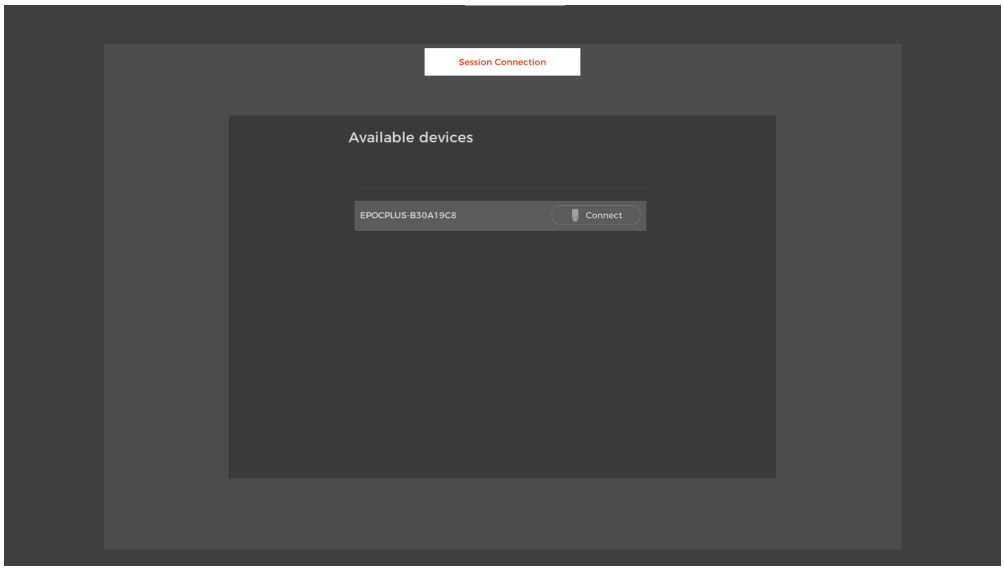
3.

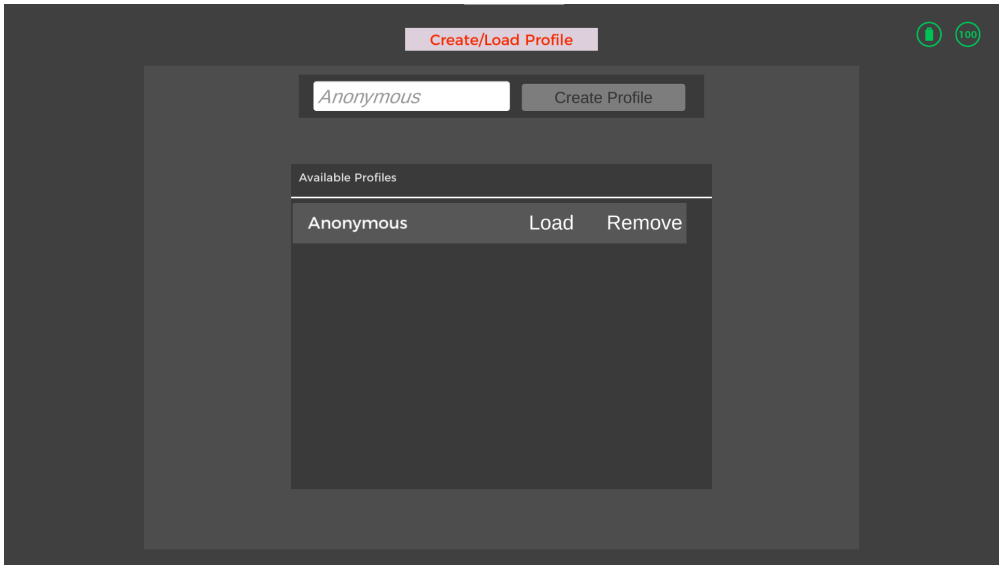
User Case

Name: Charles Huang
Age: 25
Sex: Male
Hobby: Games, Watching video clips
Player actions:

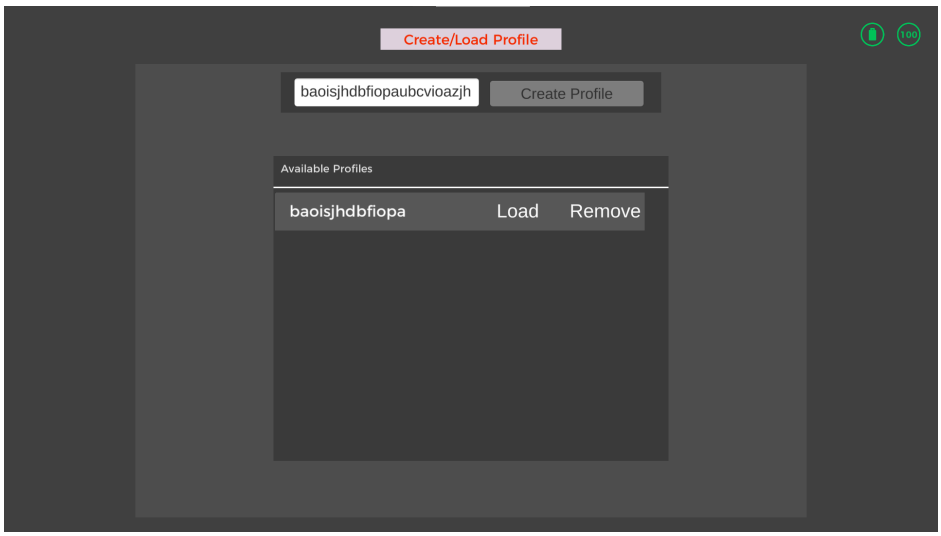
- Turned on the game
- Couldn't continue because he didn't have the client ID and client Secret for login credentials.
- Made a new account and registered the application as instructed
  - 

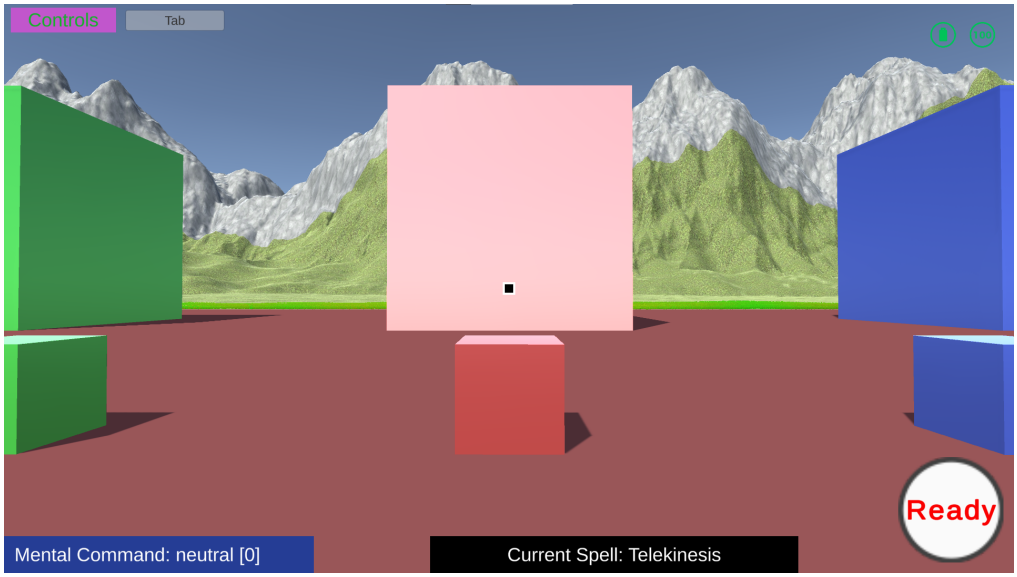
- Had to install Emotiv launcher
- Logged in (Client ID and Client Secret were hidden for privacy)
  - 

- Created Virtual device for headset simulation
- Connected headset as that's the only option
  - 

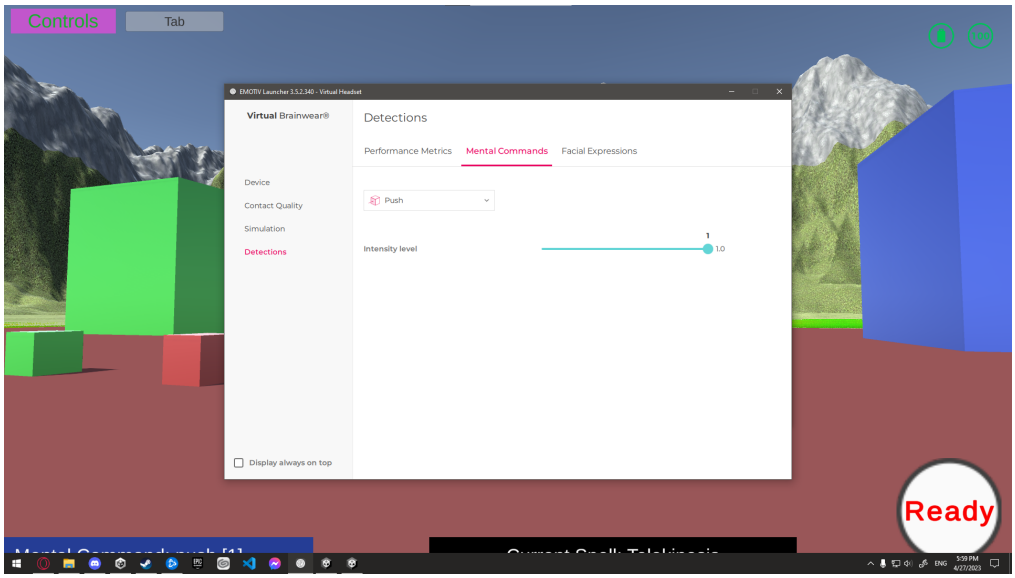
- Created default anonymous profile
  - 

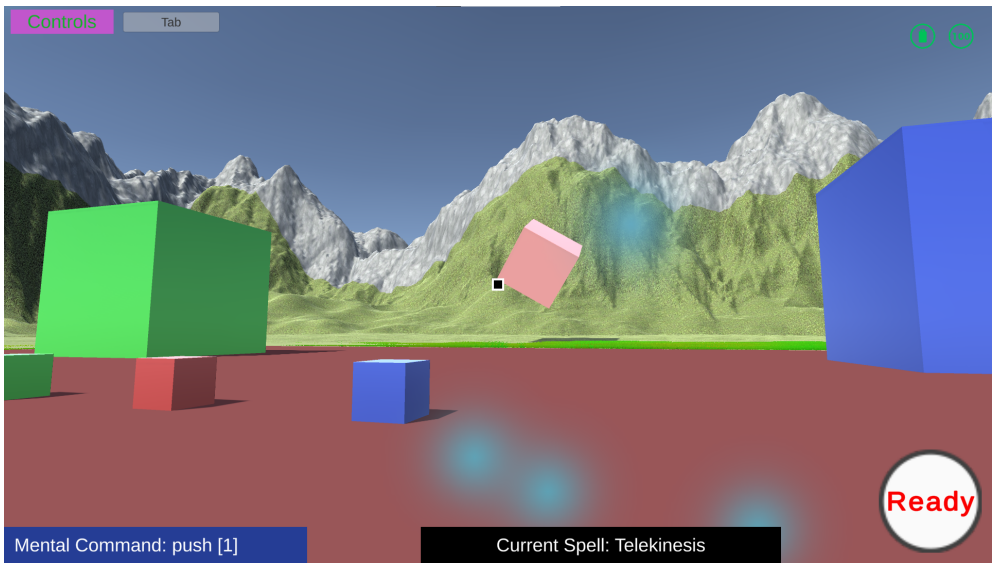
- Deleted it and created keyboard smash name

- 

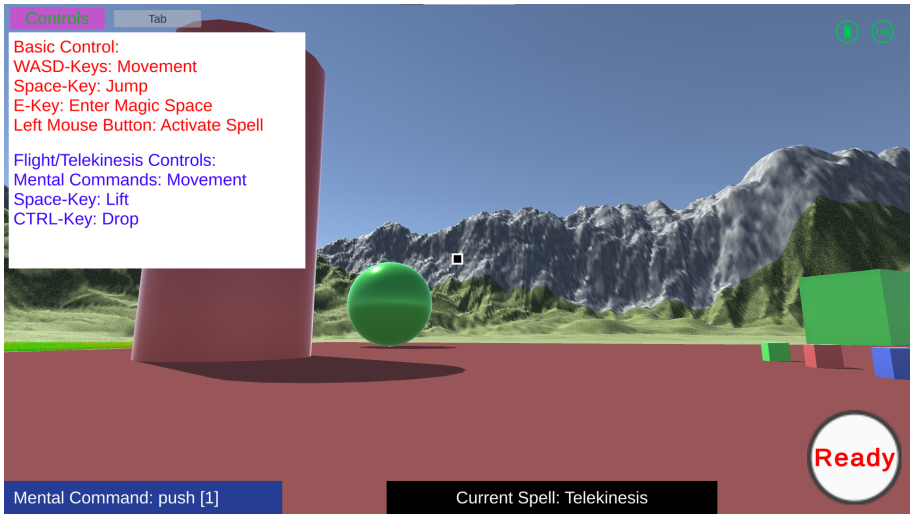
  - The profile element couldn't hold the full keyboard smash name so only displayed it partially
- Loaded profile
  - 

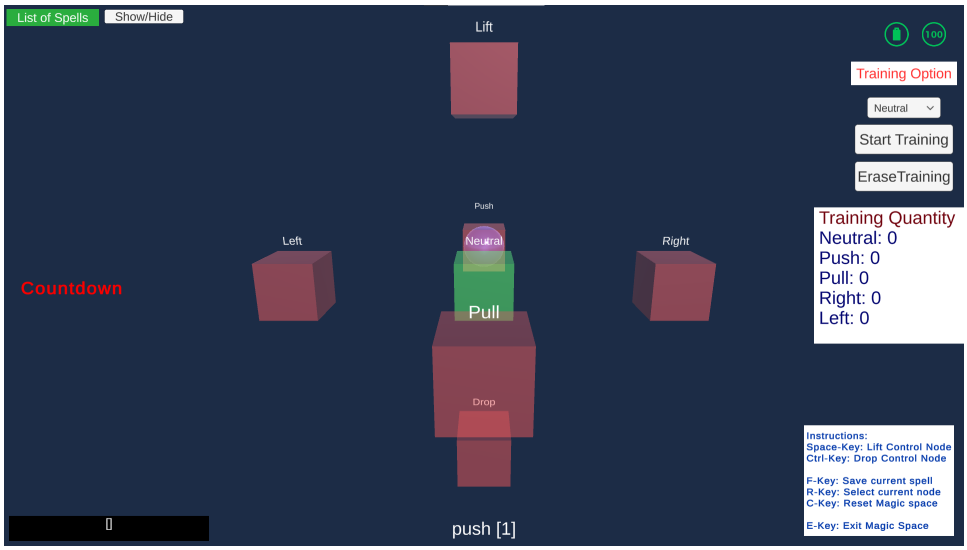
- Notices default spell as telekinesis
- Changes the command input from virtual device settings

- 

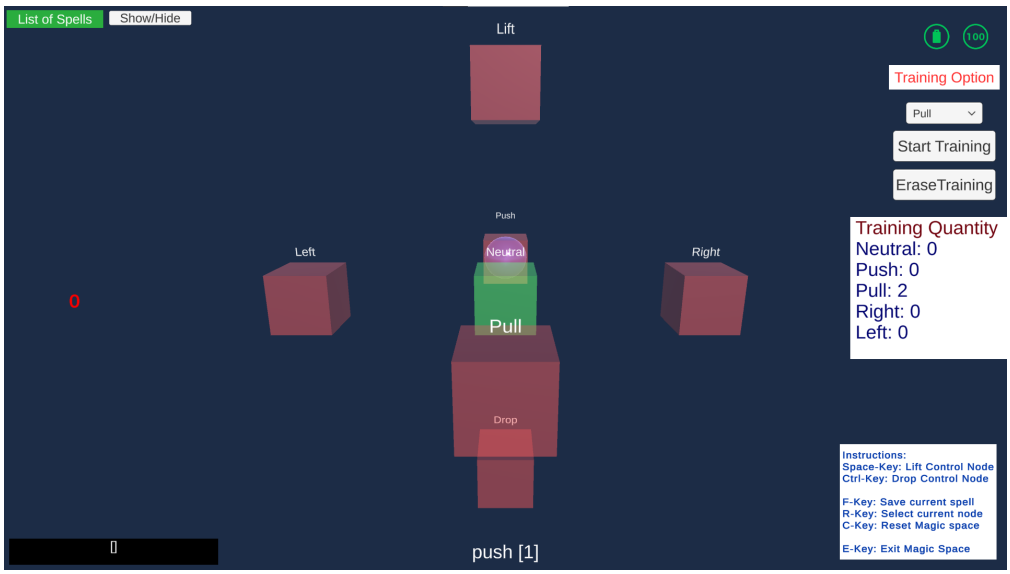
- Activates spell
  - 

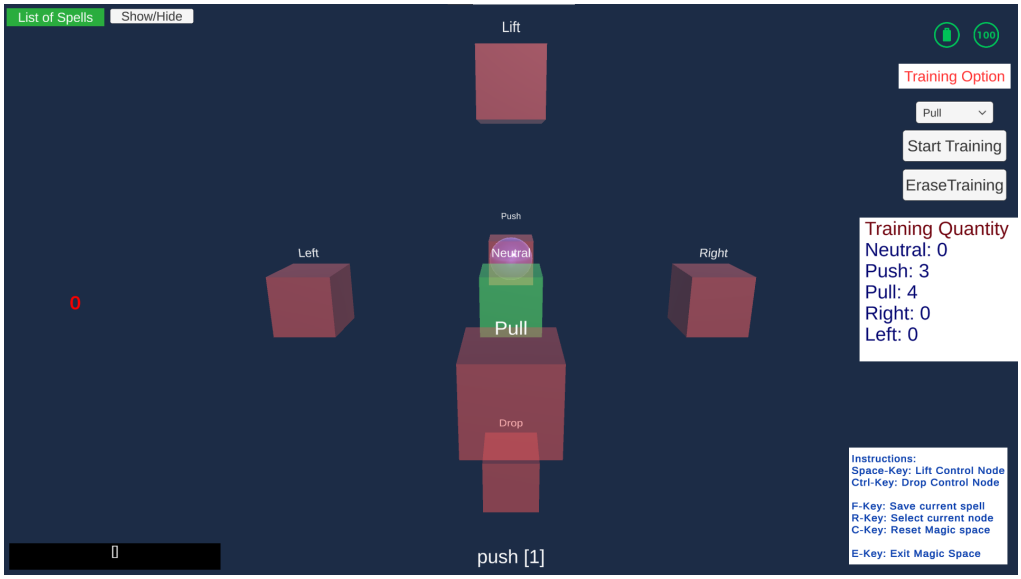
  - Shouldn't be possible on a real device with no training, but simulation device sends the perfect data so it ignores training aspects
- Notices the control options

- 

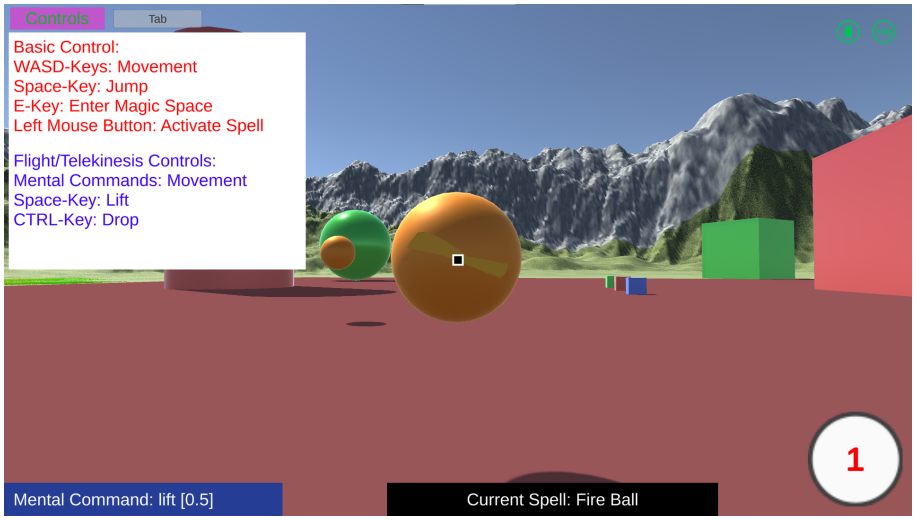
- Goes to magic space
  - 

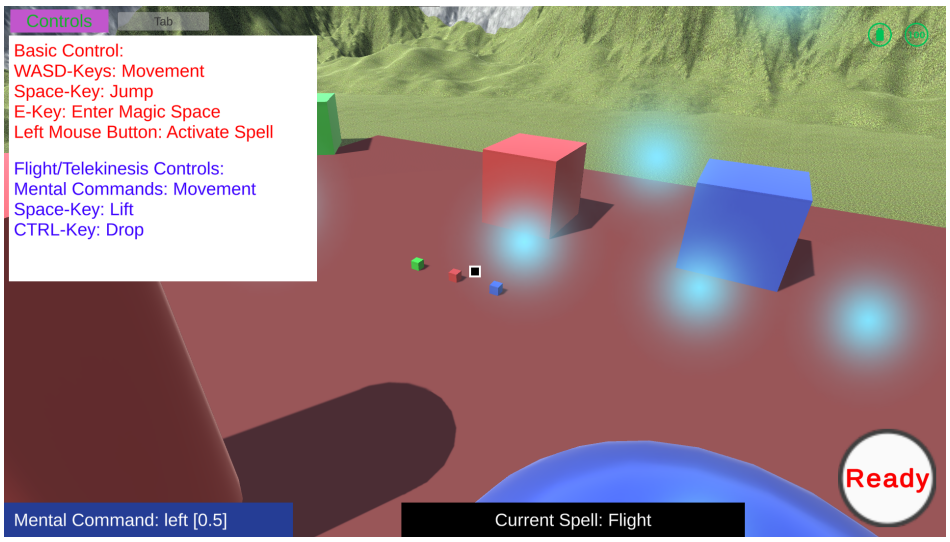
- Sees training
  - Had basic understanding because of listening in during the project development process
- Tries training in opposite command

- 

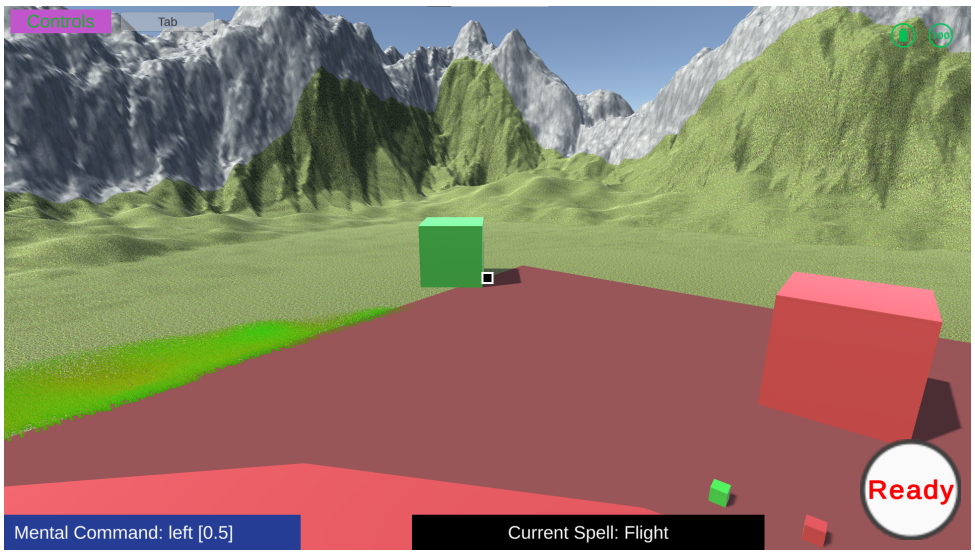
- Sees no progress due to the simulated device data bypassing profile customizations
  - 

- Loses interest and moves to the spells
- Fire Ball

- 

- Telekinesis was already tried
- Tries Flight
  - 

- On the cylinder object

- 

Test:

- Run through the game without guidance
- Run through the game with a list of tasks
  - Don't have any headsets devices
  - Create, delete, load profiles
  - Test all magical abilities
  - Purposefully incorrect magical ability command
- Test questions
  - Did the game feel smooth?
    - smooth movements and animation
    - cool spells
  - How did the usage of headsets feel?
    - Nothing since I used the virtual headsets
  - Was there difficulty in enacting desired actions?
    - No as I used the virtual headset
  - How difficult was it to activate a spell?
    - A bit tedious, maybe because I had to control the commands manually?
- Personal feedback
  - Very novel and interesting
  - Push and pull is kind of boring for spell symbols
  - Why are lift and drop mental commands?
  - I want to test how damaging the spells are
  - There could be more things to do?

3.1.1. Unit testing:

3.1.1.1. Headset Testing

Due to the complexity and differences of inputs among different people, it will be difficult to implement the aspects related to the headset. Therefore, there will be one main tester that will be used as the baseline for testing the accuracy of commands and while a few other testers will be used to test if the headset is able to recognize a select few commands from these participants as well as it did with the main tester.

How the score was calculated was:

sumCorrect = sum of the power levels of the commands that were received
numCommands = number of commands received
score = sumCorrect / numCommands x 100

3.1.1.1.1. Testing Scores

John Lim (Real Device)

Command	Performance Score (0-100)
Push	10.8
Left	20.6
Pull	12.7
Right	25.4

Charles Huang (Virtual Device)

Command	Performance Score (0-100)
Push	100
Left	100
Pull	100
Right	100

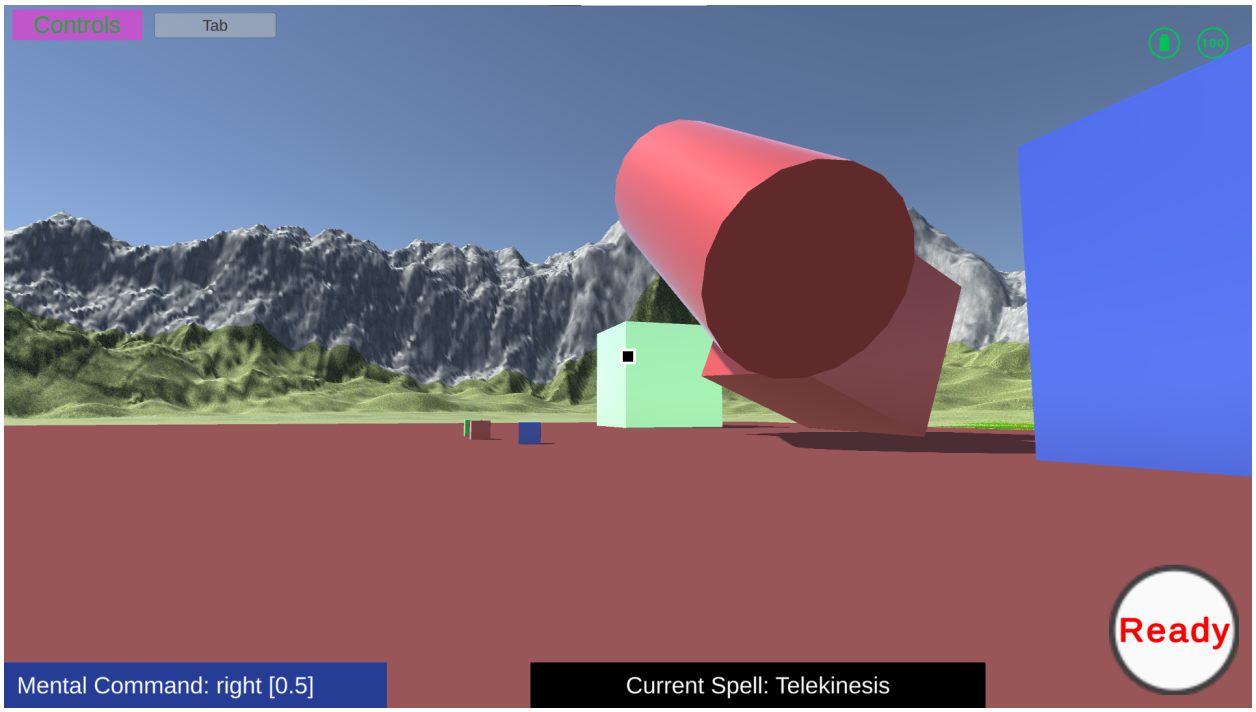
The Analysis is that the virtual device greatly overperforms due to perfectly conforming to data preference of the Cortex API machine learning algorithms to obtain perfect scores.

Meanwhile when using the actual device the performance drops. Right and left commands were the easiest commands for the player to grasp meanwhile the other commands were not as easy as it was not that simple to grasp the movements within their minds, but after training, the scores exceeded the ones listed.

3.1.1.2. Interactable Testing

Interactactable objects will be tested if they obeyed the physics law given in the game itself and if they successfully abide by their functionalities.

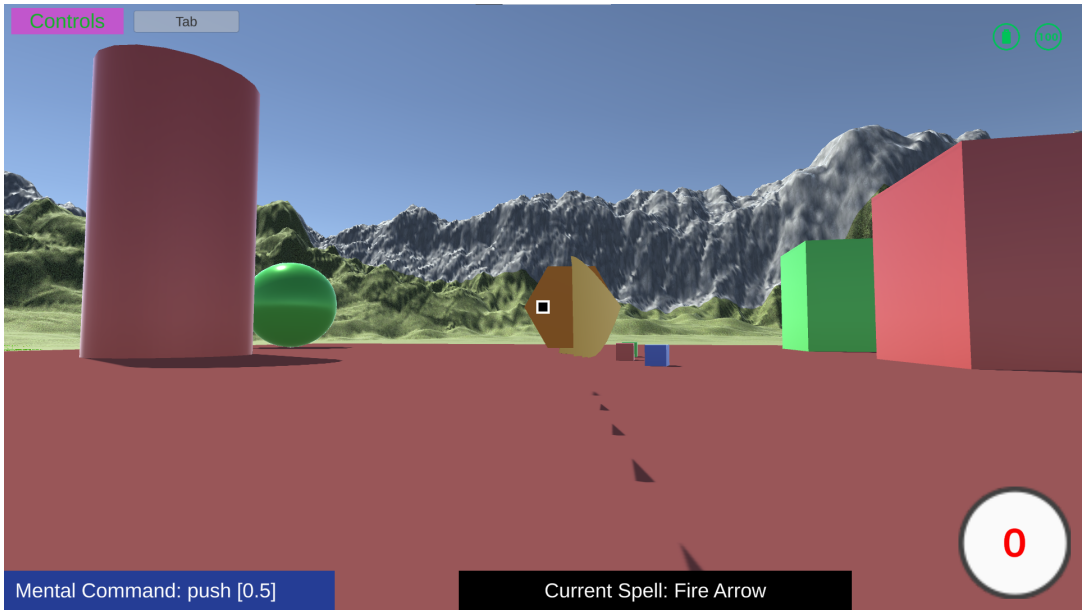
3.1.1.2.1. Results (Everything works as it should)

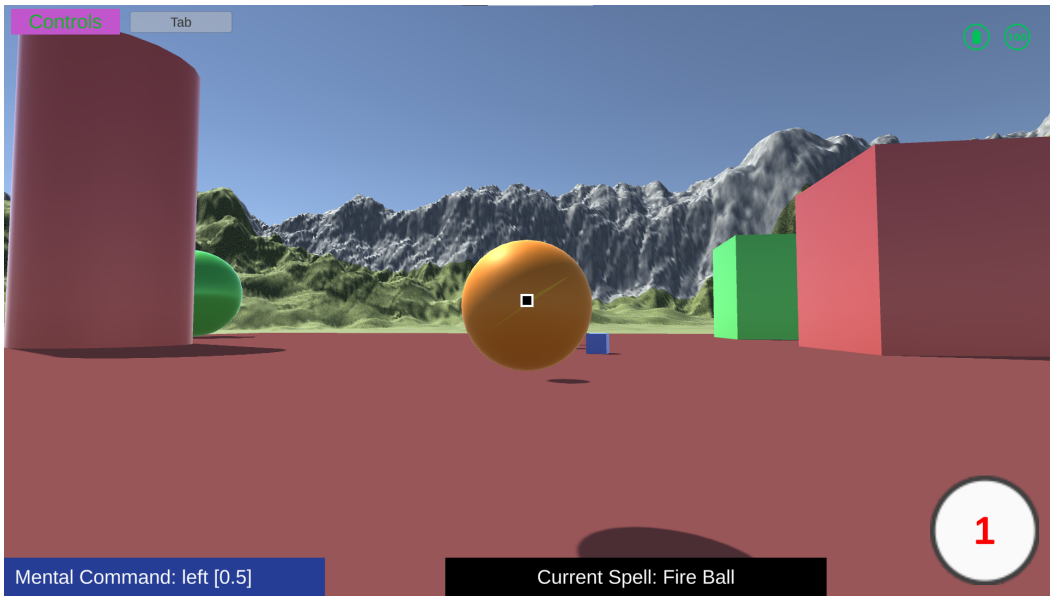


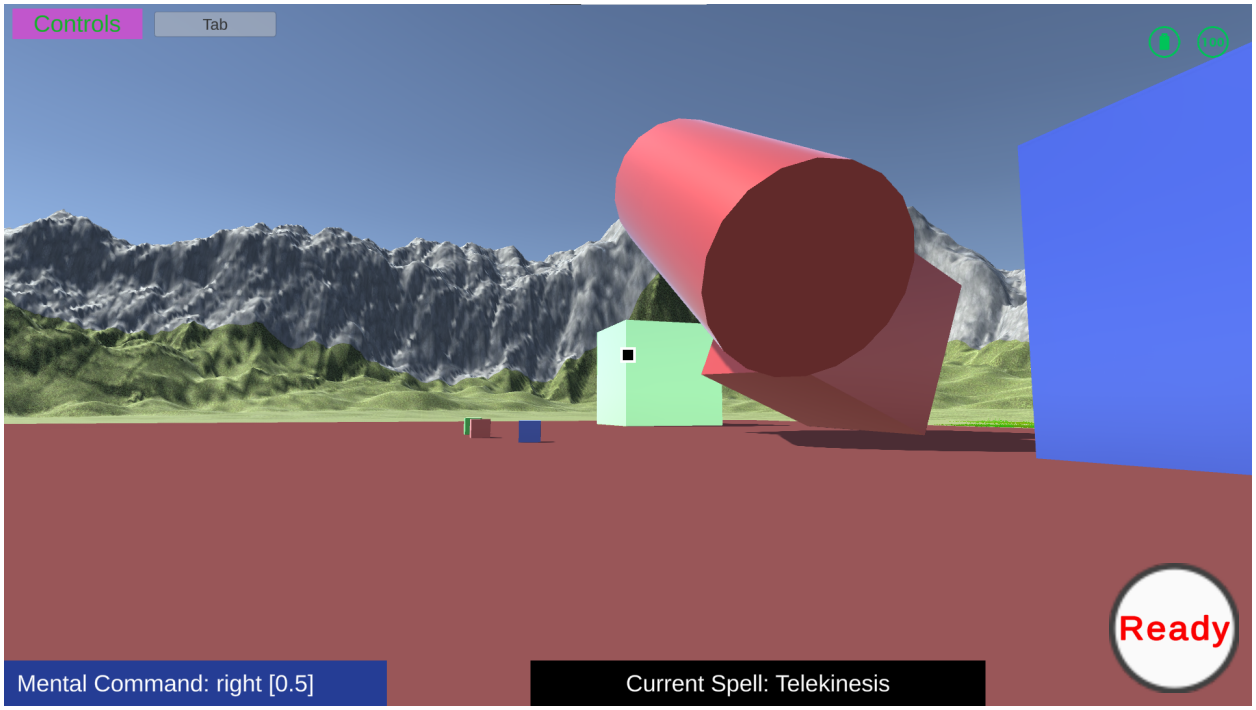
3.1.1.3. Ability Testing

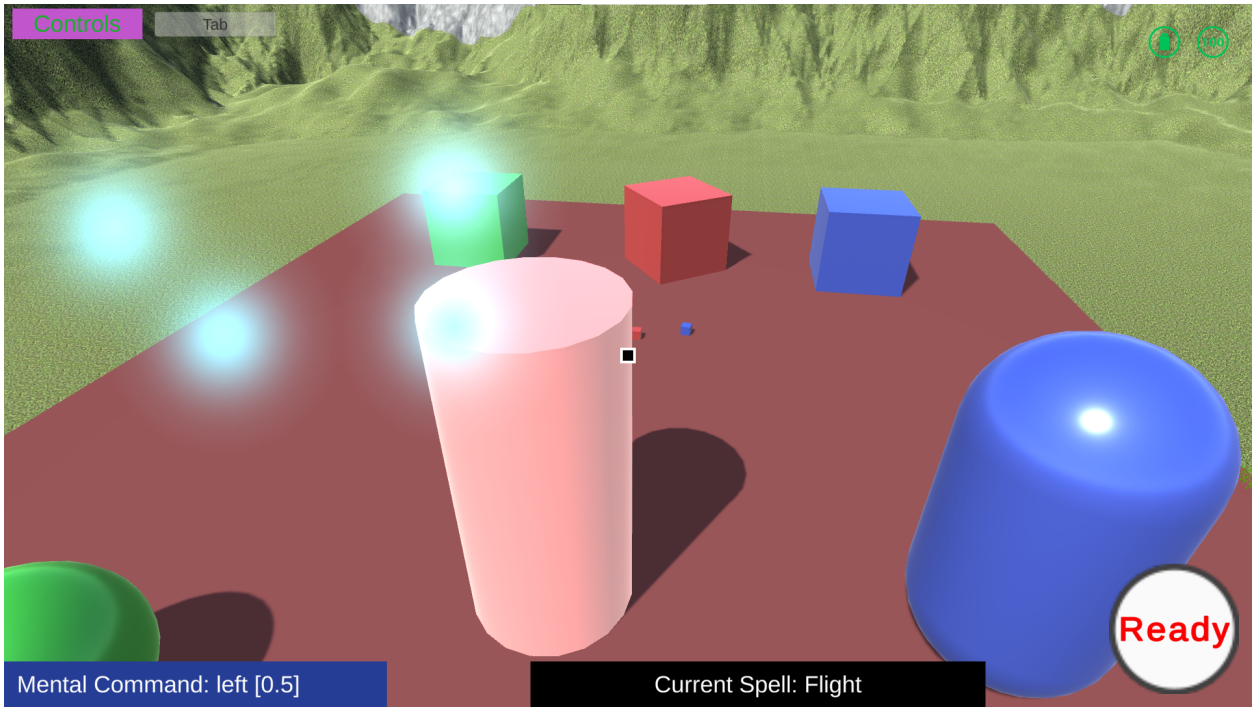
Abilities will be tested on their functionality and desired effects on other objects. Visual aspects will be on lower priority and only the bare minimum will be accounted for in its testing such as correction of direction, colour, and duration.

3.1.1.3.1. Results (Everything works as it should)


Controls
Tab
0
Mental Command: push [0.5]
Current Spell: Fire Arrow


Controls
Tab
1
Mental Command: left [0.5]
Current Spell: Fire Ball





3.2. Implications of Implementation

3.2.1. Summary

The project works fine and there are only minor bugs that did not affect the overall project.

The most important problem to note beforehand is that when the game is launched for the first time since the device started without the Emotiv launcher opened, it fails to authenticate for the

first time. It is only when the application is reset or if the Emotiv launcher is open beforehand that it can progress. There was no actual solution to this other than opening the Emotiv launcher before starting the game.

The project lacks a bit on the game side as the Neural Headset side of the development took too much time. The development of the Neural Headset side of the project was referenced from the cortex-v2-example git repository project. It demonstrated the usage of the Zenject library and how to implement it, which allowed it to be used as a template for the profile UI. However, the UI of the profile page is lacking in appearance due to the page being different in what it needs in comparison to the example.

The problem with the profile page is that the last profile cannot be removed visually until restarted even though it is removed from the Cortex API. This can easily be resolved by creating another profile and deleting that profile so this issue was not considered, due to lack of time.

When a player erases training data of certain commands too frequently, it eventually corrupts the machine learning parameters to the point that no commands would be recognized. This needs further observation as it has happened only once and might not be a bug.

The game is also no longer compatible with the original Emotiv plugin due to key changes in their scripts. The faulty code that calls on a null function is also a problem in the provided plugin making the update of the Emotiv plugin not possible.

In the implementation of magic, the movements of the control nodes are good and functional, but the fact that the node moves even when players do not intend to as mental commands are not always voluntary needs some consideration.

On the game side, for unknown reasons, the ram usage of the game increases when the objects are moved outside of the spawn platform. Then also there are issues where the game world's lighting is extremely dark on one run time and bright as intended on another.

The biggest issue with this game is that first, the player must own an EPOC+ headset from Emotiv and second the fact that saline solution needs to be soaked into the sensors of the headset to make them work causing the hair to feel discomfort. There is also the fundamental discomfort of the headset itself which might even be painful after prolonged use.

3.2.2. Requirements

3.2.2.1. Emotiv EPOC headset

The headset model has to be of the EPOC model line as the project only supports this specific model as of yet.

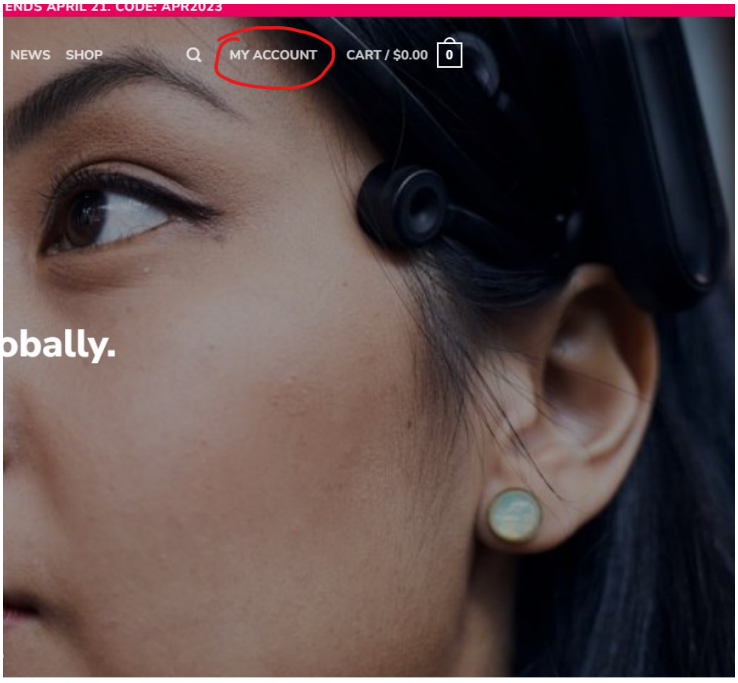
3.2.2.2. Emotiv Account

An account on https://www.emotiv.com/ is required in order to play the game

3.2.2.3. Client ID and Client Secret

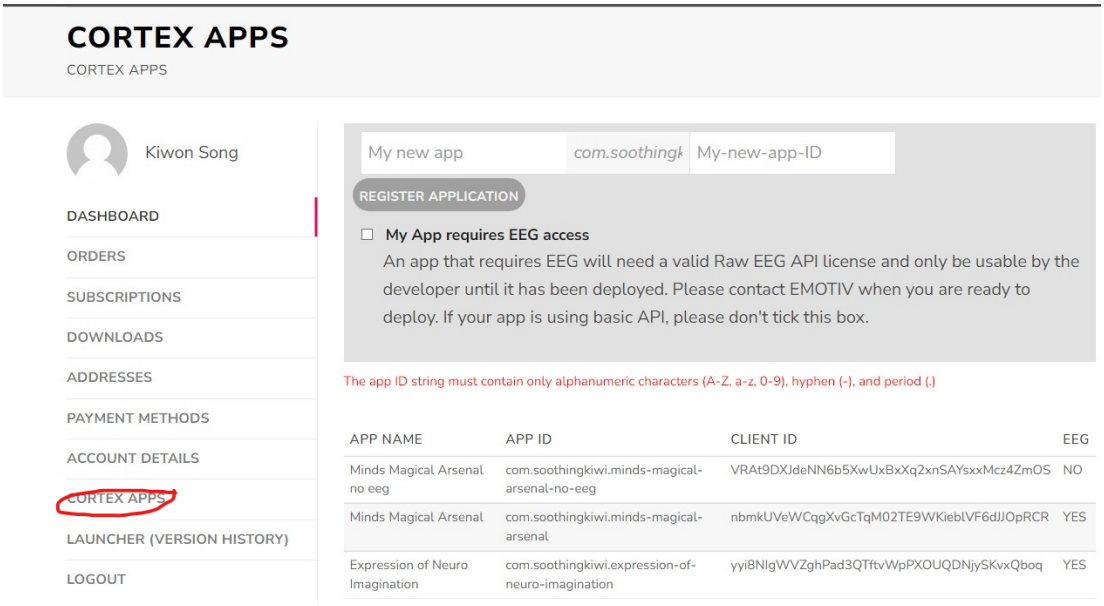
In order to log in to the game the user must have a 'Client ID' and 'Client Secret', to do so players must follow the following steps

The player first has to log in to the Emotiv website, if they do not have an account, they are to do so before going to the next step. Go to the user dashboard by clicking ‘My Account’ on the top right of the Emotiv homepage after logging in

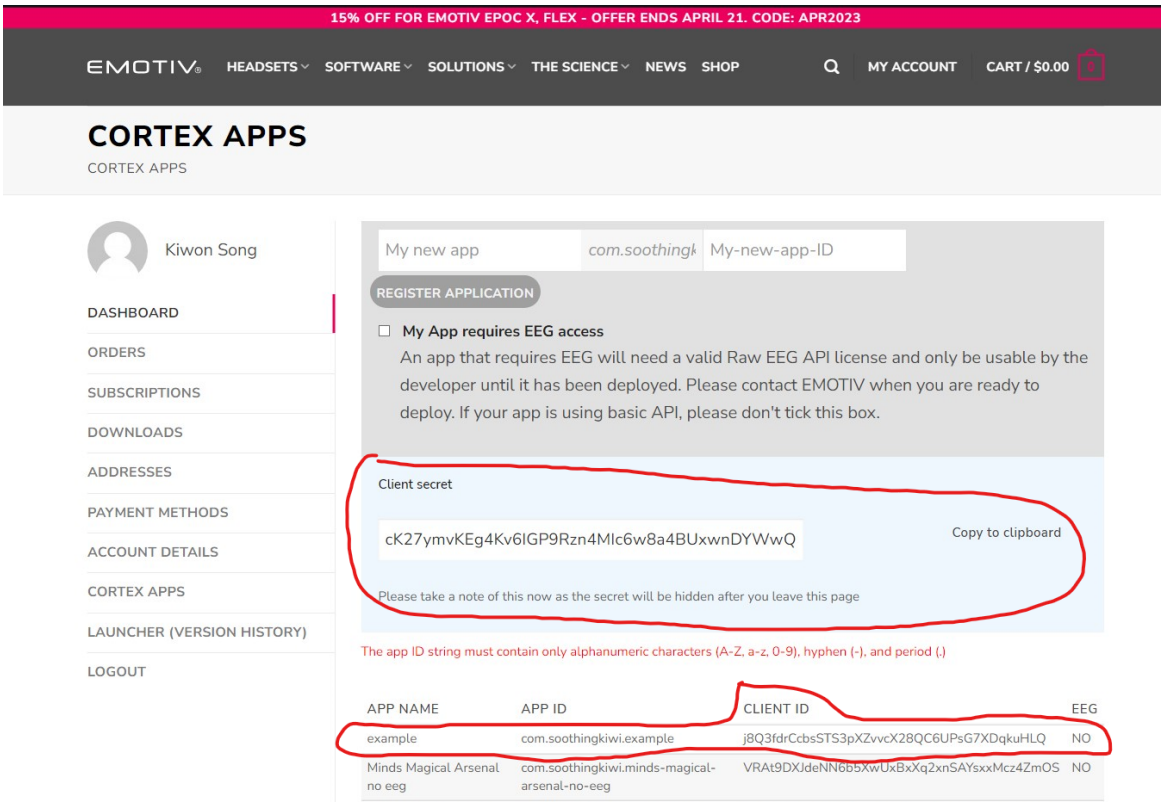


Then make sure that the “My App requires EEG access” is **Unchecked** and enter in the name for the app, just the left input box is enough as the right input box would automatically generate the same words based on the input on the left. Once complete enter the ‘Register Application’ button.

The check box being unchecked is very important as EEG requires a subscription to make it work otherwise the game would continuously request the player to apply for a subscription and would not work. This caused some great delays due to not realizing that EEG data is membership-only content.



Once the application is created, the Client ID and Client Secret are made and shown, but the Client Secret is only shown once, which causes applications to be created multiple times due to this issue.



Now the Cortex API can be entered by passing the Client values through the Websocket. The WebSocket is an inbuilt Unity feature that developers can override to connect to

web services, including APIs. The WebSocket requests the Cortex API for access to the Emotiv App where the game can then access the headsets to connect.

3.3. Innovation

The central concept of this project is based on the desire, “I want to feel like I have superpowers”. VR and AR games have a similar concept, but it is activated by the tap of a button or specific physical movements. One example is Natural Magic VR which portrays a lot of the magical aspect but lacks the novel fantasy sensation of activating spells with a simple thought.

The technology used in this project is the EMOTIV EEG headset that uses sensors to capture the faint electrical activities of the human brain to detect unique intents of the mind. An innovative technology that does not have much software or games outside of research purposes. This includes games, social media, and publicity, making it so that this project will be one of the first games made solely for this technology.

The only similar technology for Neural Headsets would be VR which allows players to move physically in comparison to Neural Headsets that actually make physical movements a demerit as it takes in brain signals instead of making physical movements unnecessary and would worsen the quality of the signals it receives.

So far the only use that Neural Headsets have been for research purposes only except for a demo where people made a demo game using a Neural Headset which allowed people to move objects in the game around like a psychic. That game was only for that specific demonstration and the code was not released, so the idea was to make a game using Neural Headsets as well.

The idea was to try using this technology to demonstrate the possibility of gaming using this technology that counters VR as its opposite. VR is active gaming technology while Neural Headsets are non-active gaming technology. It is a way for people to experience a different kind of ‘fun’ than moving their bodies and instead move their minds.

3.4. Complexity

The complexities of this project stemmed from the following reasons: the lack of community and the incomplete plugin library.

The lack of community from what can be observed is mainly caused by the lack of interest in Neural Headsets other than their uses in research in aspects like science or health and the fact that the

technology is still young. When any questions pop up about how to use the related technology and library, there are little to no answers to those questions. This caused plenty of issues in using the Cortex API and Emotiv/unity-plugin as there was nothing other than a sample project that barely implemented basic functionalities.

Then a bigger problem cropped up as it was later found out that the Emotiv/unity-plugin was incomplete and had its own issues. This problem caused the progress to grind to a halt for multiple days due to a single issue until finding out the cause was simply a null function being called automatically from a WebSocket response.

This was also an issue when specific training data was required from the Cortex API and it had the functionality for it, but the plugin did not have the function to do so. To fix this, the plugin itself had to be modified where certain parts had to be erased and new variables and functions were created out of necessity.

With these two issues combined, the time it took to solve each issue involving the Cortex API and the Emotiv/unity-plugin greatly increased.

3.5. Research in New Technologies

The technologies that are used in this project that are new to me and had big impacts were the following: Neural Headsets, Cortex API, Zenject, and Emotiv/unity-plugin.

Neural Headsets are headsets that pick up signals from the brain by attaching sensors to the scalp on the head. Even so, the headsets do not pick up brain signals that well and mostly only pick up surface-level intents of the mind.

Cortex API is a database provided by Emotiv to allow programmers to access data from their headsets and store them online or offline. The API mostly acts as a proxy between the headset and the program that receives the data from the headset. It uses AI to allow people to train a unique model for themselves that is tuned for their brains. It can also translate the received data into a simpler form that can easily be understood and utilized in for example this project. If the person has a membership with Emotiv, they can also obtain raw EEG data to be used for research purposes or to be fed into their own AI.

Zenject is a library that allows programmers to use dependency injections. This allows various script objects to pull another script object that exists in the injected space and use them as though that object was a static object and could be accessed without purposely assigning them to each other.

Emotiv/unity-plugin is a library provided by Emotiv developers for Unity developers to access the Cortex API and its functions. The problem is that this plugin has some problems as it is still quite young, only made in 2020, and does not have many developers working on it.

3.6. Future Enhancements

Visuals can be upgraded in the future however this is completely optional as it does not affect the performance of the game. The visuals included are the world generation, the appearance of spells, and the object.

More spells are best to be considered as it is the easiest and direct area for improvements. Examples would entail meteor showers, teleportation, and area-of-effect abilities like poison fog.

Hostile NPCs are best to be added to increase the entertainment factor of the game. Stationary objects with health bars to portray the damage of the spells are also an option for players to test out the effects of the spells.

3.7. Timeline and Milestones

Milestone	Task	Estimated Time of Completion	Time of Completion
Milestone 1	Create basic 3D world	1	3
Milestone 1	Headset connection research	10	20
Milestone 1	Headset connection	40	50
Milestone 1	Profile setup	30	90
Milestone 1	Setting up Neural Connection UI	10	20
Milestone 1	Player setup	20	10
Milestone 1	Created dictionary for possible magic abilities	5	2
Milestone 2	Cortex API functionality setup	10	20
Milestone 2	Magic UI setup	10	5
Milestone 2	Magic Neural setup	30	40
Milestone 2	Magic setup	50	30
Milestone 2	Training setup	50	40
Milestone 2	Interactable Object setup	30	20
Milestone 2	Hostile Mobs setup	20	None
Milestone 2	Health and Attack setup	10	None
Milestone 2	Hostile mob pathfinding	30	None
Milestone 3	Game Terrain Decoration	10	5

Milestone 3	Options Menu	5	5
Milestone 3	Documentation	30	35
Milestone 1 Total		116	195
Milestone 2 Total		240	165
Milestone 3 Total		45	40
Total		401	390

4. Conclusion

This project widened my view of what programming can really do. Requesting data from APIs and receiving constant responses allowed me to understand web interactions better. It told me that problems do not always appear on a silver platter for me to analyze. Instead, I have to discover it on my own initiative or it will come back later as a bigger problem. It taught me that you can experience many unknown aspects of things that you believed to have known about already. As for the Neural Headset, it was interesting to use, but I did not really learn much about the Neural Headset itself but I did find that it had quite a bit of untapped potential and I hope that I can have a career relating to it in the future.

4.1. Lessons Learned

What was learned through the course of this project was mostly about the various technical aspects of unity and good code of conduct in programming, however most important thing learned in the project was time management.

Time management was the biggest issue of the project if time had been managed well early on in the development, the progress of this project is likely to have gone even further than it is now with more leeway in the schedule for more development in less important aspects of the project such as graphics.

As for the technical aspects of unity, the main part of the learning experience was about Zenject. Zenject is a dependency inject library that allows game objects to connect to each other without needing to find the object or dragging them to each other.

For the code of conduct in programming, two things were learned. First, plugins and libraries are not perfect, they don't always work so always double-check before continuing that they work before continuing. Second, problems do not always show up as errors and one must be careful for any sign of such problems as they are very hard to fix later along the line.

4.2. Closing Remarks

It was a fun yet intense project where I could explore the usages of Neural Headsets in close proximity while realizing the potential and difficulty of utilizing them without having any constraints on what I can and cannot do.

5. Appendix

5.1. Approved Proposal

A00967329_KiwonSong_Major_Proposal_Minds_Magical_Arsenal.docx

5.2. Project Demonstration Video

Demo(Final).mkv

5.3. Project Supervisor Approvals

Dear Gustavo,

Kiwon Song has successfully demonstrated his project to me, and I am satisfied he has completed the essential requirements as set out in his approved proposal. I have also reviewed his final report, and feel it is ready to be submitted to the Major Projects Committee for its review.

Thanks,
Borna

Borna Noureddin, PhD
Faculty, Bachelor of Technology Program
British Columbia Institute of Technology | Computer Systems Technology
3700 Willingdon Ave | Burnaby, BC V5G 3H2
borna_noureddin @ bcit.ca | +1 604 453 4007
http://commons.bcit.ca/bnoureddin/

6. References

Wikimedia Foundation. (2023, February 20). *Electrogram*. Wikipedia. Retrieved April 28, 2023, from https://en.wikipedia.org/wiki/Electrogram

Getting started. Getting Started - Cortex API. (n.d.). Retrieved April 14, 2023, from https://emotiv.gitbook.io/cortex-api/

Emotiv. (n.d.). Emotiv/Cortex-V2-example: Example with cortex V2/V3 API. GitHub. Retrieved April 14, 2023, from https://github.com/Emotiv/cortex-v2-example

YouTube. (2020). *[Tutorial] How to setup Emotiv Epoc headset*. *YouTube*. Retrieved April 22, 2023, from https://www.youtube.com/watch?v=Z81Lb3EIEz4

Waltz of the Wizard: Natural Magic on Steam. (n.d.). Retrieved November 1, 2022, from https://store.steampowered.com/app/1094390/Waltz_of_the_Wizard_Natural_Magic/

7. Changelogs

- Accepted all grammar change requests
- Removed the section where it says the game has both fighting and peaceful mode as it only has peaceful mode. (1.3)
- Added a small paragraph for EEG quoted from Wikipedia (2.5.1)
- Changed the sentence structure for the EEG to BCI translation paragraphs (2.5.1)
- Updated contents of innovation (3.3)
- Expanded contents for 4.2
- Added 5.2 for the demo video (5.2)