# Final project Report

# Virtual Reality Electronics Laboratory for an Educational Environment

Dropbox Link:
https://www.dropbox.com/sh/jj8zfv3qnvyriiz/AABrZX5WyshwQkt13-Tda6Fwa?dl=0

Zack Bloundele

Student Number: K1549996

Zack Bloundele
K1549996

VR Electronics Laboratory

**Abstract**

This project looks at how to improve the educational environment by the addition of Virtual Reality; with the goal of improving student's attentiveness. This report will explain the research and development process that went into producing a Virtual Reality Electronic Laboratory Prototype Educational Game for Mobile VR.

Zack Bloundele
K1549996

VR Electronics Laboratory

## Introduction

In the current education system, educators are faced with the problem of how to capture the attention of a new generation of learners [1]. This is important, as explained in a report by the UK Department for Education, 'The Impact of Pupil Behaviour and Wellbeing on Educational Outcomes' (2012) [2], which found that those with fewer attention problems achieved better scores in Key Stage tests. Taking into consideration how much funding goes into education – the UK having just spent £54 billion in 2015 [3], and the US looking to spend $210 billion in 2017 [4] – it is crucial to make sure that money spent is improving student engagement within the educational setting.

Today, instead of employing traditional educational tools, it may be worth considering a new approach. A study by Laseninde, Mpofu and Campbell (2015) [5], showed that the use of technology in an educational environment helped increase the retention of taught concepts, and provided an enjoyable experience for the students. This is backed up by Barata, Filho and Nunes (2015) [6], who found that incorporating Virtual Reality (VR) and Virtual Environments (VE) into the classroom, for the purpose of visualisation and virtual training, made learning more interesting, motivating and clear for those that participated. As such, those in education can benefit from the technology of VR games [7].

VR has come a long way since the concept first put forward in Ivan Sutherland's paper, 'The Ultimate Display' (1965) [8]; he introduced the concepts of immersion in a simulated world, which used a complete sensory input and output. This has since become the basis of VR research. In the current technology market, VR Headsets are now affordable for the everyday consumer [9] and available on a variety of hardware – ranging from games consoles (Playstation VR), to computers (Oculus Rift) and even mobile phones (Google Cardboard) – it is now easier than ever to access VR. This consumer access has lead to reports indicating VR hardware to be worth an estimated $25 billion, and VR software around $15 billion, by the year 2020 [10] [11].

With the advent of mobile VR, a user can turn their smart phone into a VR headset, all with the addition of the Google Cardboard. This allows the user to share the VR experience in a cost effective way – Google Cardboard retails for £15 [12] – compared to those on other devices. These cheap VR options are a way to incorporate VR into the education setting, and can be used to help boost attentiveness as explained by Barat, Filho and Nunes (2015) [6]. They are however not the only ones who have investigated VR's potential in the education landscape. Borst, Ritter and Chambers (2016) [13], created a VE of a Solar Technology Applied Research and Testing Laboratory: a solar thermal power plant. This was then used to allow students to better understand and explore the laboratory, from the safety of the classroom, without needing a physical tour. Which could have proved difficult for the students, due to geographical or scheduling issues.

Virtual Environments also provide other benefits. VR is able to provide a user a safe environment in which they can better tackle a difficult problem. By being able to attempt different solutions and observe the after-effects, the user can better understand what they have observed, which promotes out the box style thinking [14]. This is something that is much harder to attempt within an traditional educational setting. As well as problem solving, VR can also be used as a cost efficient training system, as proved by Marthur (2015) [15], who created an interactive and immersive demo, in which students got to interact with a three-dimensional model of the human body. With the addition of hand interactivity – using motion controllers – users were able to learn by doing. This is extremely important in certain fields, such as medical learning or auto-mobile repair. As such, VR provides a cheap way for people to gain experience in a VE, while being at home or in the classroom.

There are a few concerns about using VR in the education environment, including educators' lack of experience and health and safety issues. Teachers have a limited amount of time they can spend preparing and conducting their lesson plan, as well as covering required topics in a semester. This could lead to VR and other technology being limited to those teachers with outside experience using the VR devices [16]; which could lead to educators with poor technological experience, negatively affecting the learning experience of student, when attempting to incorporate VR into the lesson setting.

Current health and safety issues encountered while using a VR device have been drastically reduced, compared to those experienced ten or twenty years ago [17]. This is due, in part, to advancements in technology, and also better design and development practises being used [18]. The main two issues that users currently encounter are cyber sickness, which is similar to motion sickness – usually caused by accelerating the player – and side effects from using a VR device for long periods of time. These side effects can range from eye strain to mild migraines [19]. However, a study carried out by Arns and Cerney (2005) [20], found that those issues are unlikely to appear in most users, and symptoms more likely to occur in older users.

## Project Aim

For this project a Virtual Reality Mobile Electronic Laboratory Education Game prototype, for use in an educational environment, was developed. It is designed to provide an example of how VR can be used to enrich a user's learning experience. The main aim of the game is to teach the user the basics of circuit construction; this is achieved through a series of lessons. Each lesson requires the user to complete a series of tasks and is also introduced to a new component, which they will use in constructing the lessons circuit.

## Methodology

With creating a VR game or app, the first thing to decide is what type of VR experience it's going to be. This is important, as it helps inform the choice of platform to develop for and affects the design choices made, based on the pros and cons of the device.

For this project, the prototype is intended for use in an educational environment, as such ease of accessibility and cost were big factors in the eventual choice of hardware; which was Mobile VR. Mobile VR requires the minimal amount of hardware requirements of all VR headsets, with most current generation of phones being able to run a variety of VR experiences. This is perfect for an educational environment, due to VR ready mobile devices being much cheaper than £1000+ VR ready computers [20]; thus requiring less of an education institutes budget to obtain.

Having selected the platform in which the app was developed for, the next step was choosing which VR headset to use. As with mobile VR devices, the headset viewer component is a fraction of the cost of its console or computer counterparts [12] [21]. The choice of which VR headset to use came down to the Google Cardboard and Samsung Gear VR; the Cardboard being the eventual winner, due to being the cheaper view but also not exclusive to one make of phone. Which would greatly reduce the usability of the Electronic Laboratory educational game. Another benefit that Cardboard provided, was the VR Camera toolkit that is available for developers; it is easily be imported into the game engine of choice, making it very simple to create VR apps and games.

The choice of which engine to use was fairly simple; Unity Game Engine provides a great environment for developing a variety of projects for different platforms. There are many available VR camera toolkits, that can be imported into a Unity project, providing an instantly accessible VR

camera within the scene, while also being easily customisable for the project's needs. Lastly, Unity makes it very easy to build and deploy a Unity project to an app store of the developer's choice.

**Game Navigation**

When it comes to designing the game levels, it is extremely important to consider how the user navigates from start to finish. Figure 1 is the game navigation for the Electronics Laboratory. The game starts off in the main game scene, from here they can play around in the level, choose to start a lesson of their choice or access the game's menu. If the user chooses to begin a lesson, then they stay in the lesson state until they either complete the lesson or choose to leave it, in either case they return back to the main scene. The user can choose to also exit the VR Lab scene, which is done in the main game scene by accessing the menu and choosing to exit.

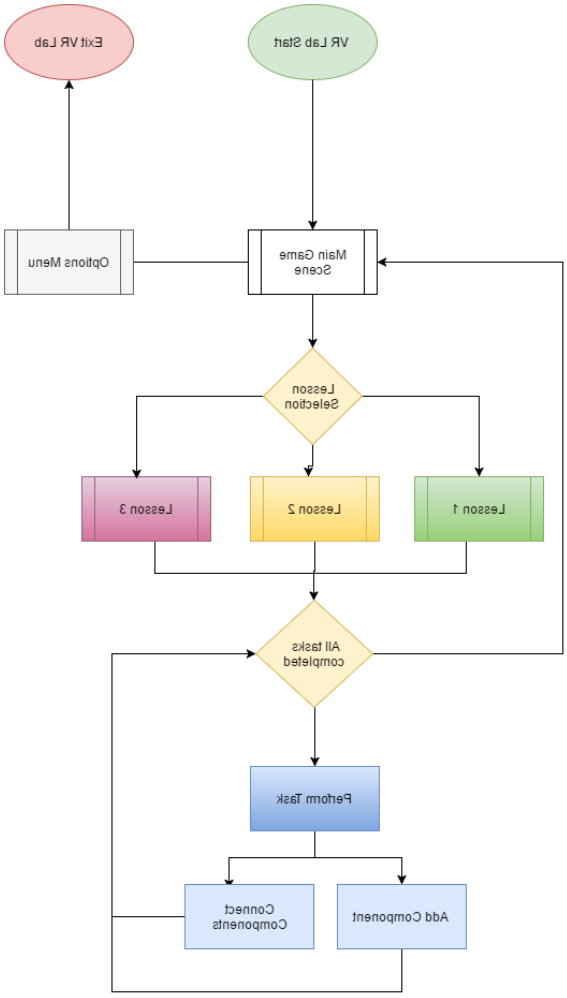


*Figure 1. Game Navigation*

Development – User Interaction

User interaction is the most important thing in any game or app, it's how the user access the content that has been created. VR has added new challenges with how the user interacts with the VE displayed to them. Headsets such as Oculus Rift (Reference) and HTC Vive (Reference) use a combination of motion tracking, of the user and headset, and motion controllers to achieve an immersive experience. However, a gamepad could also be used for user input but is a less

immersive. For mobile VR, it is not possible to use motion tracking, and while a gamepad can be connected and provide input, it requires user to own a USB OTG (on the go). Most mobile VR apps and games, use a reticule or counter based system, which allows the user to look at an object or action and after a period of time it's selected. This is a fairly simple system, but means the user requires no external controller to interact with the game. It does have the issue of increasing game time, as unlike with a controller, the user is not seeing an immediate change. As such the choice of using a timer based system could lead to a negative experience for the user.

For the VR Electronics Laboratory, the timer based system was used for user interaction. This choice was made as it provided the most accessibility to the user. While controller input is much more efficient, it's more of a hassle to setup and could detract from people using it.

**Evolution of the Interaction Implementation – Initial Implementation**

The first method of user interaction implemented was the timer/counter system. Essentially the user would look at an object and a circle game object would increase in size, until it reached the max size. Once this happened the object or interaction would be selected and appropriate code would be run. Figures 2, 3, 4 show how this process looked in the game scene.

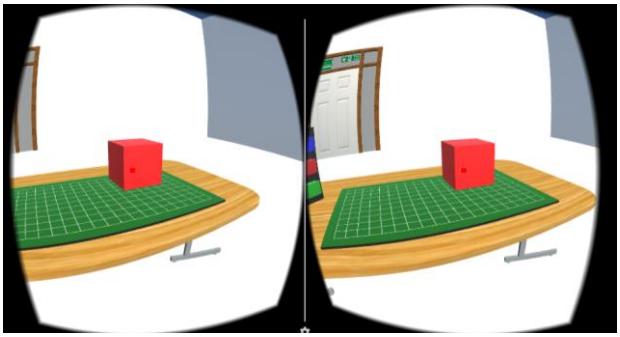

*Figure 2. Object Unselected*

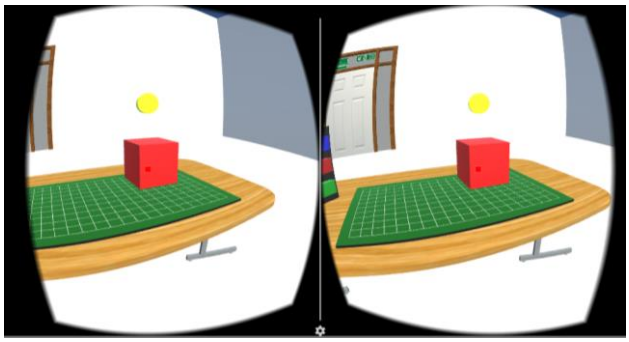

*Figure 3. Object Selecting*

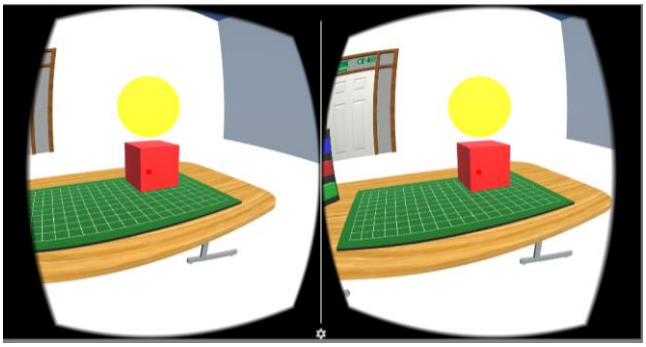

*Figure 4. Object Selected*

Figure 5 is from the old ZB_ReticuleLook script, the section being part of the main function ObjectInteraction() and is called on update. What the reticule does is raycast from the camera straight ahead and returns if it hits an object. Next the object layer is checked to make sure it's interact able, any object or interaction would have their layer set to this and means no static objects are being affected. The next step is to work out if the object his is one that was looked at in the previous frame or is a new one; this determines how to proceed. If the object is a new object being looked at, then it sends a message to the old object, which causes it to stop its counter grow, updates the current object variables with the new object and sends a message to start its counter grow. If the looked at object is still the same one as the previous frame, then keep the counter grow function going. Lastly, if there was no object being looked at in the previous scene, update the current object to the looked at object and start the counter grow.

```
#region Methods
//Check if user is looking at a object
void ObjectInteraction()
{
    Debug.DrawRay(transform.position, transform.forward * 10, Color.red, 0);
    Debug.DrawRay(transform.position, transform.TransformDirection(Vector3.forward) * 10, Color.blue, 0);
    RaycastHit hit;

    //If controllerless
    if (SM_Script.VRInteractionStates == ZB_SettingManager.VRInteractionState.HandsFree)
    {
        if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit))
        {
            if (hit.collider.gameObject.layer == LayerMask.NameToLayer("Interactable"))
            {
                Debug.Log(hit.collider.gameObject.name);
                if (go_current_object != null)
                {
                    //If the hit object is a new object than update the current object
                    if (go_current_object.name != hit.collider.gameObject.name)
                    {
                        //Send a message to the current object, which will stop any interaction
                        go_current_object.SendMessage("StopSelection", SendMessageOptions.DontRequireReceiver);

                        go_current_object = hit.collider.gameObject;

                        //Send a message to the new current object
                        go_current_object.SendMessage("StartSelection", SendMessageOptions.DontRequireReceiver);
                    }
                    else if (go_current_object.name == hit.collider.gameObject.name)
                    {
                        //If the current object name is the same as the hit object
                        //Send a message to the new current object
                        go_current_object.SendMessage("StartSelection", SendMessageOptions.DontRequireReceiver);
                    }
                }
                if (go_current_object == null)
                {
                    go_current_object = hit.collider.gameObject;

                    //Send a message to the new current object
                    go_current_object.SendMessage("StartSelection", SendMessageOptions.DontRequireReceiver);
                }
            }
```

*Figure 5. Old ZB_ReticuleLook Script - ObjectInteraction*

Figure 6 are from the ZB_ActionsCounter script, which is attached to interactable objects. If an object was being selected, it would first check what type of action was being selected. Next the counter size is increased until the action counter is greater than the action cooldown, this being the

time it takes for an action to be selected. Once selected it sends a message to the appropriate scripts, so the correct action can be performed – use the object, pick it up or deselect it.

```
void CounterGrow ()
{
    //Increase selection counter and size of the counter gameobject
    //If the counter is not equal to cooldown total
    if (fl_action_counter != fl_action_cooldown_total)
    {
        //Increase counter time
        fl_action_counter += Time.deltaTime;

        //Increase counter gameobject size
        //go_object_selection_counter.transform.localScale += new Vector3(Time.deltaTime * 0.25f, 0, Time.deltaTime * 0.25f);
        switch (st_action_type)
        {
            case "Action - PickUp":
                go_action_pick_up_counter.transform.localScale = new Vector3 (Time.deltaTime * 0.25f, 0, Time.deltaTime * 0.25f);
                break;
            case "Action - Use":
                go_action_use_counter.transform.localScale = new Vector3 (Time.deltaTime * 0.25f, 0, Time.deltaTime * 0.25f);
                break;
            case "Action - DeSelect":
                go_action_deselect_counter.transform.localScale = new Vector3 (Time.deltaTime * 0.25f, 0, Time.deltaTime * 0.25f);
                break;
        }
    }

    //If the counter is greater than the cooldown total
    if (fl_action_counter > fl_action_cooldown_total)
    {
        //Set the counter equal to the cooldown total
        fl_action_counter = fl_action_cooldown_total;
    }

        }

        //If the counter is equal to the cooldown total
        if (fl_action_counter == fl_action_cooldown_total)
        {
            //Update the component manager so the object is selected
            if (SM_Script.VRInteractionStates == ZB_SettingManager.VRInteractionState.HandsFree)
            {
                switch (st_action_type)
                {
                    case "Action - PickUp":
                        ResetValues();
                        CPU_Script.SendMessage("PickUpGameObject");
                        CM_Script.SendMessage("PickedUp");
                        break;
                    case "Action - Use":
                        CAM_Script.SendMessage("ActionOn");
                        break;
                    case "Action - DeSelect":
                        ResetValues();
                        CM_Script.SendMessage("UnSelected");
                        break;
                }
            }

        }
    }
```

*Figure 6.Old ZB_ActionsCounter Script – CounterGrow*

Now this method had some issues, it wasn't very efficient code wise and required a counter gameobject to be attached to every interaction and interact able object. It also didn't look very appeasing, however at the time it was implemented it was functional, so development moved onto another section and it planned to be reworked down the line.

**Evolution of the Interaction Implementation – Interaction Updated**

Having finished the implementation of the mechanics for the VR Electronics Laboratory, there was time to rework the user interaction in the game, and make it look and work better. However, during

this process, it became clear they may be a better way to do the interaction; instead of having a counter appear above the object, a reticule that filled up when looking at an object or interaction and selected/initiated it when filled. This would be much cleaner, as a interaction state script could be made that worked with objects and interaction all the same, and there would be no need for having a counter object on everything. Figures 7, 8, 9 shows off how the new reticule interaction system works.

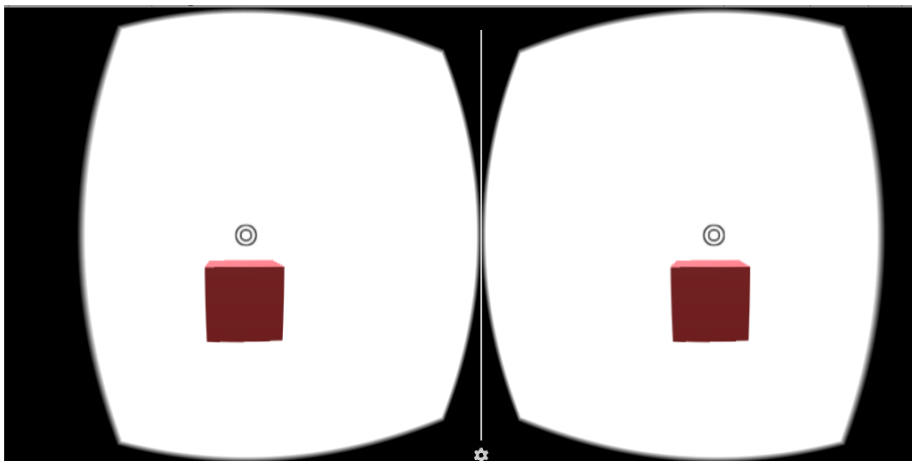

*Figure 7. Object Unselected*

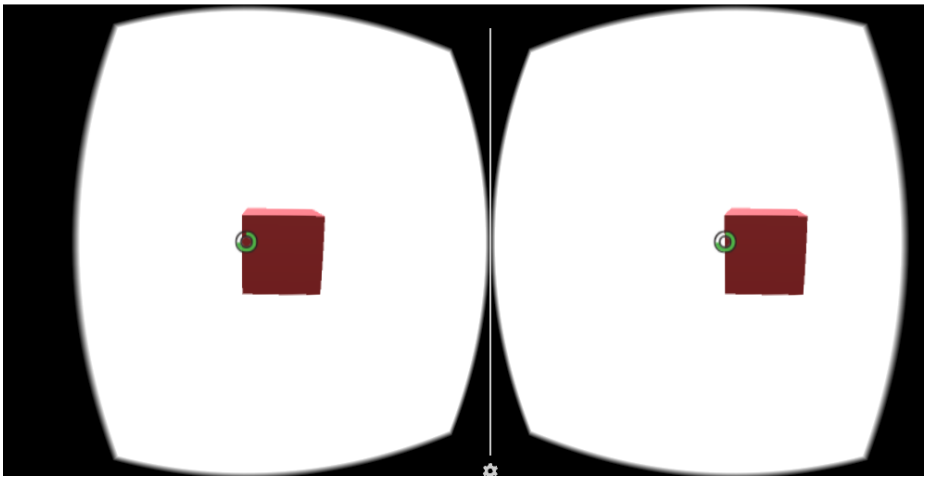

*Figure 8. Selecting Object - Reticule Filling*

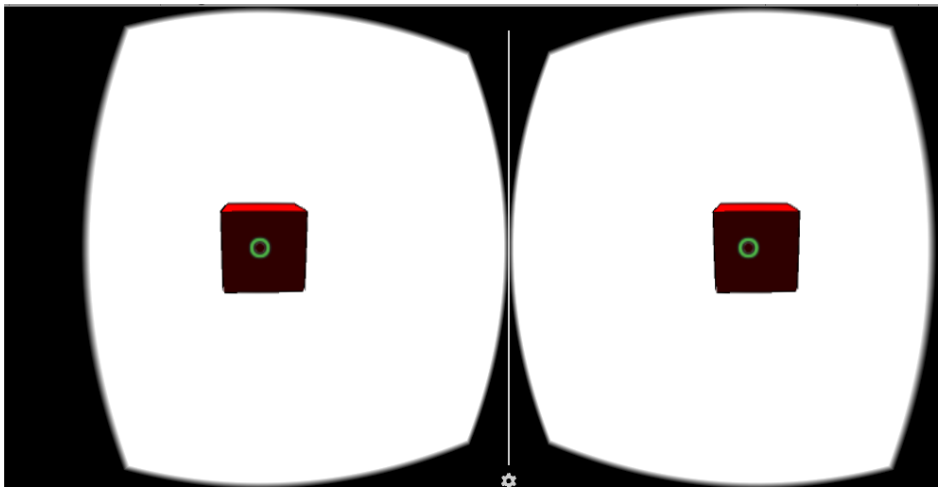

*Figure 9. Selecting Object - Reticule Filled*

This is achieved with three main scripts; ZB_InteractionState, ZB_ReitculeLook and ZB_ReticuleFill. The InteractionState script is attached to any object that can be interacted with, while the two reticule scripts attached to the reticule gameobect, which is part of the main camera.

```
// Update is called once per frame
void Update ()
{
    switch (componentState)
    {
    case ComponentState.UnSelected:
        if (this.gameObject.GetComponent<Renderer> () != null)
        {
            this.gameObject.GetComponent<Renderer> ().material.shader = Shader.Find ("Standard");
        }
        break;
    case ComponentState.Selected:
        if (this.gameObject.GetComponent<Renderer> () != null)
        {
            this.gameObject.GetComponent<Renderer> ().material.shader = sh_highlighted;
        }
        break;
    }
}
```

*Figure 10. ZB_InteractionState Script - Update Function*

Figure 10 is from the update function of the ZB_InteractionState and is relatively simple, but key to everything working. Essentially it keeps track of the attached objects state, which can either be selected or unselected. This information is then used when trying to select the attached object in ZB_ReticuleLook.

```
//Check if user is looking at a object
void ObjectInteraction()
{
    //Debug.DrawRay(transform.position, transform.forward * 10, Color.red, 0);
    Debug.DrawRay (transform.position, transform.TransformDirection (Vector3.forward) * 10, Color.blue, 0);
    RaycastHit hit;

    //If controllerless
    if (SM_Script.VRInteractionStates == ZB_SettingManager.VRInteractionState.HandsFree)
    {
        if (Physics.Raycast (transform.position, transform.TransformDirection (Vector3.forward), out hit))
        {
            if (hit.collider.gameObject.layer == LayerMask.NameToLayer ("Interactable"))
            {

                Debug.Log (hit.collider.gameObject.name);

                if (bl_reticule_filled == true)
                {
                    hit.collider.gameObject.GetComponent<ZB_InteractionState> ().ComponentSelected ();

                    if (hit.collider.gameObject.GetComponent<ZB_InteractionState> ().InteractionTypes == ZB_InteractionState.InteractionType.Component)
                    {
                        bl_selected_component = true;
                    }
                    else if (hit.collider.gameObject.GetComponent<ZB_InteractionState> ().InteractionTypes == ZB_InteractionState.InteractionType.Storage)
                    {
                        bl_selected_storage = true;
                    }

                    bl_reticule_filled = false;
                }

                in_check_object_value = GazeObjectCheck (hit.collider.gameObject); //Get the check value of the object

                switch (in_check_object_value)
                {
                case 0:
                    //Not Viable, display red reticule
                    RF_Script.StateInvalid ();
                    break;
                case 1:
                    //Viable, increase the reticule
                    RF_Script.StateActive ();
                    break;
                case 2:
                    //Interaction Selected
                    RF_Script.StateSelected ();
                    break;
                }
            }
```

*Figure 11. ZB_ReticuleLook Script - Updated ObjectInteraction Function*

Figure 11 is the main function in reticule look and is similar in structure to the old reticule look script. Like before it checks to make sure the object is on the interaction layer but the next part is new. First the script checks if the reticule has been filled, if it has then action has been selected and it updates all the appropriate states. However, if it has not yet been filled, it next checks the looked at objects value in the GazeObjectCheck function. This function returns an integer based on which if statement it passes or fails, such as if the object state is selected then return 2, while if a different object has been selected or the object is not viable to be selected return 0. If it doesn't trigger any of those then return 1; the integer returned then put through a switch statement, and the ZB_ReticuleFill state is updated accordingly. Figure 12 is a table showing what each integer means.

| Returned Integer Value | Reticule Effect |
|---|---|
| 0 | Not Viable: Display red fill reticule |
| 1 | Viable: Increase green reticule fill |
| 2 | Object Selected: Display full green fill |

*Figure 12. GazeObjectCheck Function Table*

```
// Update is called once per frame
void Update ()
{
    switch (reticuleState)
    {
    case ReticuleState.Inactive:
        im_reticule_red_fill.enabled = false; //Set red ring image to false
        im_reticule_fill.fillAmount = 0;
            break;

    case ReticuleState.Active:
        im_reticule_red_fill.enabled = false;

        if (im_reticule_fill.fillAmount != 1)
        {
            im_reticule_fill.fillAmount += Time.deltaTime * fl_fill_time;
        }
        if (im_reticule_fill.fillAmount == 1)
        {
            Debug.Log ("Reticule Filled");
            //Send message to the raycasting script
            RL_Script.InteractionSelected();
        }
            break;
    case ReticuleState.Selected:
        im_reticule_fill.fillAmount = 1;
        im_reticule_red_fill.enabled = false;
        break;

    case ReticuleState.Transistion:
        im_reticule_fill.fillAmount = 0;
        im_reticule_red_fill.enabled = false;
            break;

    case ReticuleState.Invalid:
        //Turn on red reticule fill
        im_reticule_fill.fillAmount = 0;
        im_reticule_red_fill.enabled = true;
        break;
    }
```

*Figure 13. ZB_ReticuleFill Script - Update Function*

Figure 13 shows how the ZB_ReticuleFill controls the reticule fill behaviour. In each update, the script checks the current state and increases or changes the fill amount/colour accordingly.

## Development – Lessons

For this game, other than the user interaction, only one other mechanic was core to the implementation and that was the Lessons. Essentially how the game tries to educate the user in how to construct circuits. Initially the design was to have it so a user could create a lesson in notepad, have it then read by Unity and then constructed so the user could complete it. However, due to time constraints this wasn't possible for the deadline, so a simpler method was used (the file reading method has been added to future implementation).

Figures 14, 15, 16, 17, 18, 19 show the lesson chalkboard, from lesson selection to the different tasks for lesson 1 and the lesson complete scene.

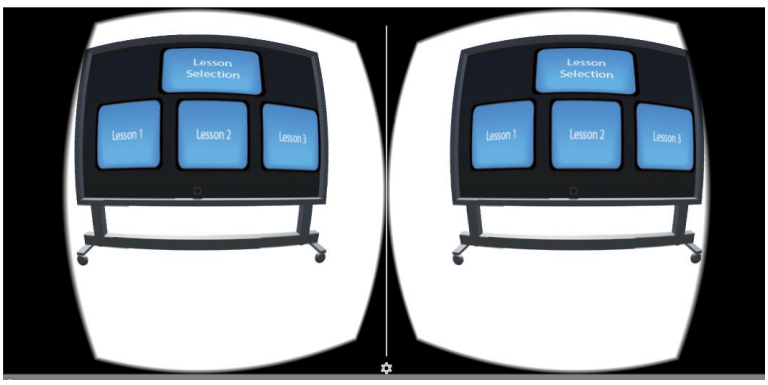


*Figure 14. Lesson Selection*

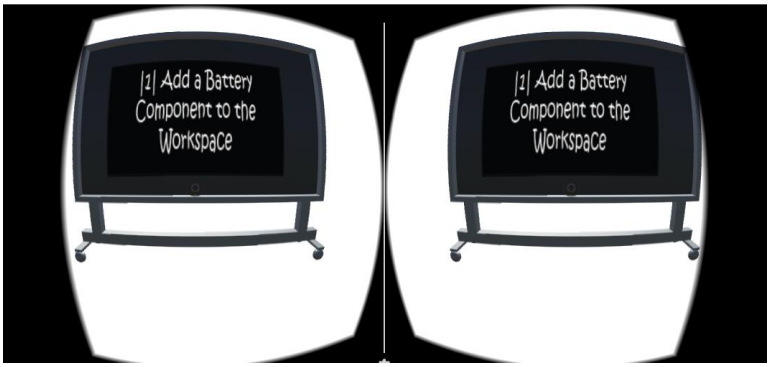


*Figure 15. Lesson 1 - Task 1*

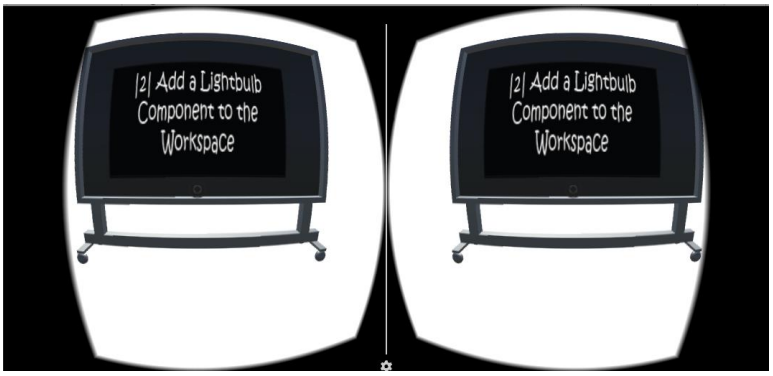


*Figure 16. Lesson 1 - Task 2*

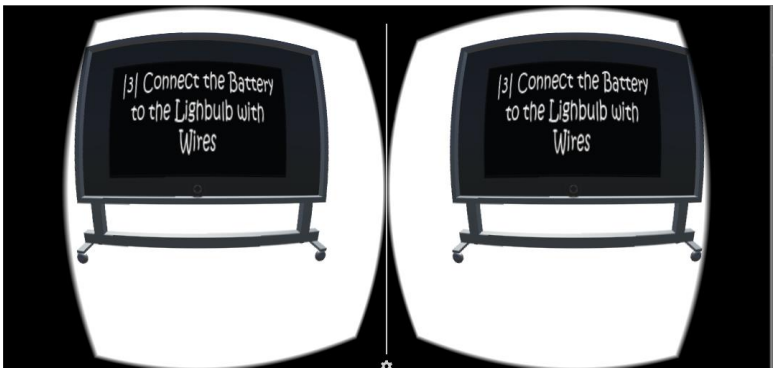


*Figure 17. Lesson 1 - Task 3*

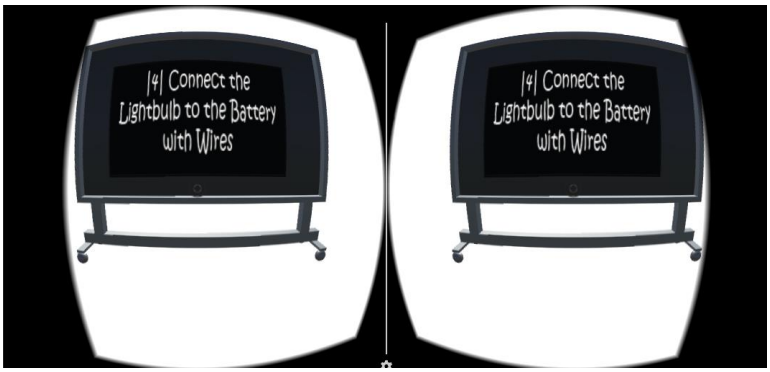


*Figure 18. Lesson 1 - Task 4*

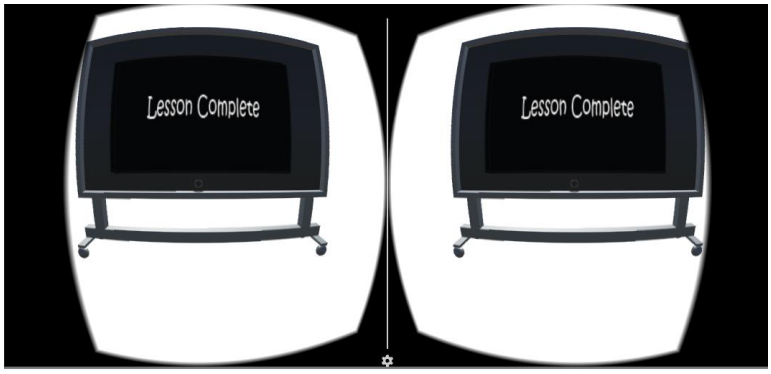


*Figure 19. Lesson 1 - Lesson Complete*

As mentioned before a simple lesson mechanic was implemented, basically for each lesson a set script with each task was made. The lesson manager would then check which lesson had been selected, and update the chalkboard based on which tasks the user had left to complete of the current lesson. The tasks themselves either require the user to add a component to the workspace, or connect two components in the workspace together. Figure 20 is from the ZB_Lesson_1 script and shows the two check functions that are called depending on the current task type, and check to see if the user has completed them.

```
public bool AddComp(string _st_component_add)
{
    //Pass the component to the workspace manager, return true if found else return false
    return WC_Script.SearchList(_st_component_add);
}

public void ConnectComp(string _st_start_connect, string _st_end_connect)
{
    //Check to see if wire start and end points = the lightbulb and battery (either direction)
    if (_st_start_connect == st_component_1_battery && _st_end_connect == st_component_2_lightBulb
        || _st_start_connect == st_component_2_lightBulb && _st_end_connect == st_component_1_battery)
    {
        in_current_step += 1;

        if (in_current_step == 5)
        {
            LeM_Script.LessonComplete ();
        }
    }

}
```

*Figure 20. ZB_Lesson_1 Script - Check Functions*

AddComp Is used when the user needs to add a component to the workspace, checks the list of components in the WorkspaceManager, and returns true if it finds the gameobject. The ConnectComp function is called when the user creates a wire between two objects, it checks if either of those connecter points match the tasks two components, that need to be connected. If they do it updates the current task number.

## Discussion

### Issues Encountered

While much of the development phase of this project ran smoothly, some issues were encountered throughout the project. One big one was an early issue with the reticule, where looking at an object close up would cause two reticules to appear while using the VR headset. This is partially due to how our eyes work in VR but also due to the reticule not adjusting its distance from looked at object. This was solved by updating the reticules depth to be just a big in front of the object that the user is

looking at, while also scaling it so it stays the exact same size; this method was found from a Youtube Tutorial by eVRydayVR [22].

**Project Plan**

Looking back at the proposed project plan – or see Figure 21 below – more time should be allocated to the implementation phases and less time to the prototyping. Many of the deadlines for mechanics to be completed by were missed, and this lead to not all the intended features to be implemented; instead being moved to future implementation. The reasoning for this was due to the inefficient implementation of the mechanics to begin with. They would often be more complex and less effective than they should be, which lead to them being redone down the line; as such it took even longer to fully finish them. A good example of this is the interaction implementation; the refined version took way less time to develop, and works ten times better than the old version (it's even easier to setup). This has really shown how better planning out of mechanics mean less time spent is improving and adjusting them in the future.

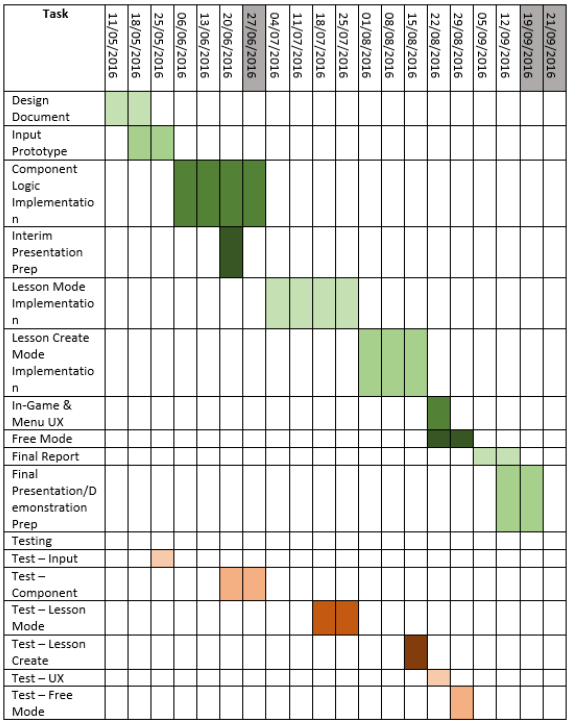


*Figure 21. Project Proposal - Project Plan*

**Project Management**

For this project Dropbox was used to manage all the files related to the project, from the different unity builds to all the developer notes made (at the end of each day working on this project, a note was made of the current aim, what was done that day and any useful links used). Dropbox made it also super easy to work on the project on multiple machines and always have access to the current build that was being worked on.

**Future Implementation**

Unfortunately, not all the planned functions were implemented however work will continue on this game once this project is over. Below is a list explaining what features will be implemented in the near future:

- *More Lessons and Lesson Creation*: Only the first lesson was managed to be implemented due to time constraints, and used a very simple system. In the future the lessons will be stored to a file, which a user can edit and create their own lessons. Next using XML read, can have the lessons be implemented in the game.
- *Better model quality*: All models in the game where made in 3DS Max but not to a high standard. Better models would improve how the game looks.
- *Controller support*: This was partially implemented early on but dropped to focus solely on the counter/timer method. It wouldn't be hard to add it back in, though adding the UI parts would take time.
- *Upload to app store*: Ideal end goal, once all the features have been implemented, is to upload to either the Google Play or Android store, so that other people may try out the game.

**Conclusion**

Overall this project was an excellent opportunity to gain a better understanding of VR and it's uses within an educational environment. The prototype produced demonstrates that educational VR games are possible and can foster a more engaging environment in which to learn. This project has also helped better understand the challenges with designing and developing for VR and how to better tackle them in the future.

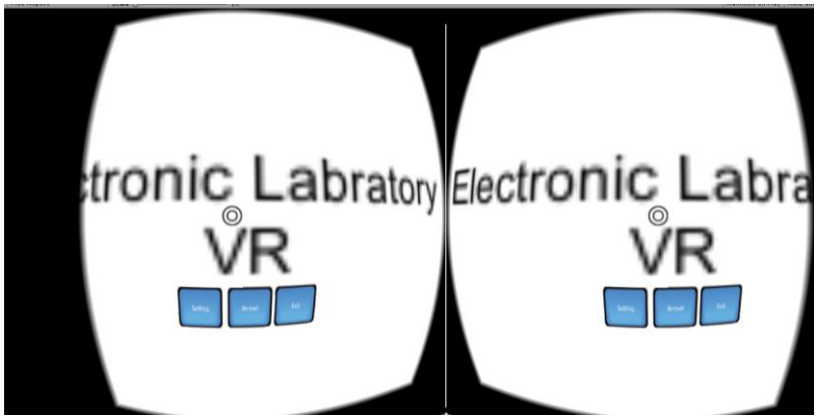


*Figure 22. Menu*

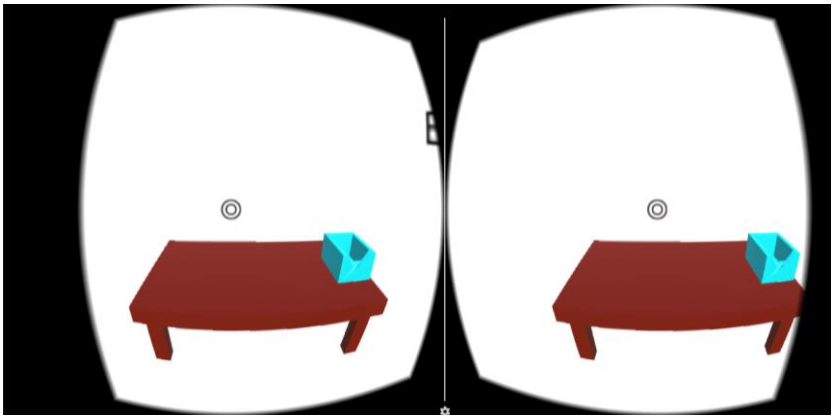

*Figure 23. Workbench*

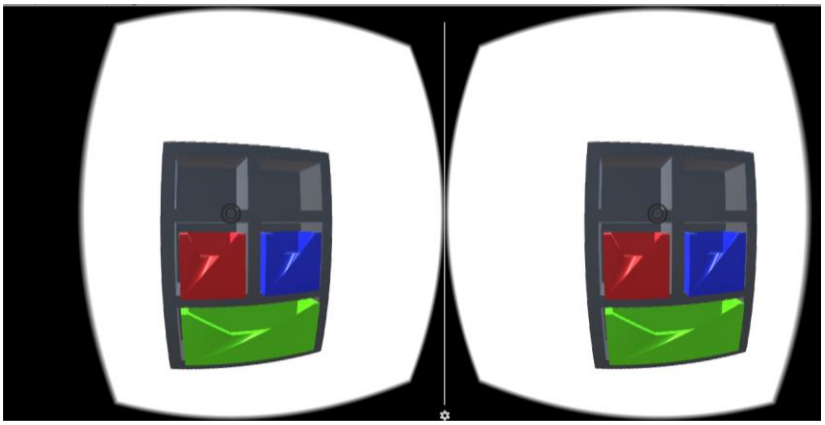

*Figure 24. Component Storage*

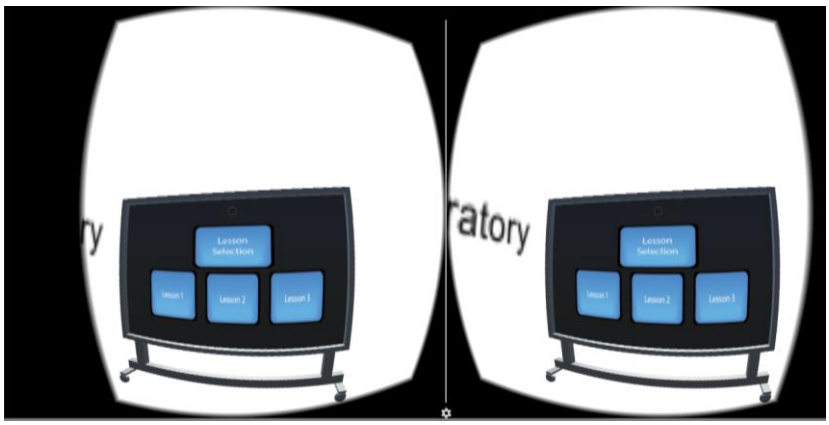


*Figure 25. Lesson Chalkboard*

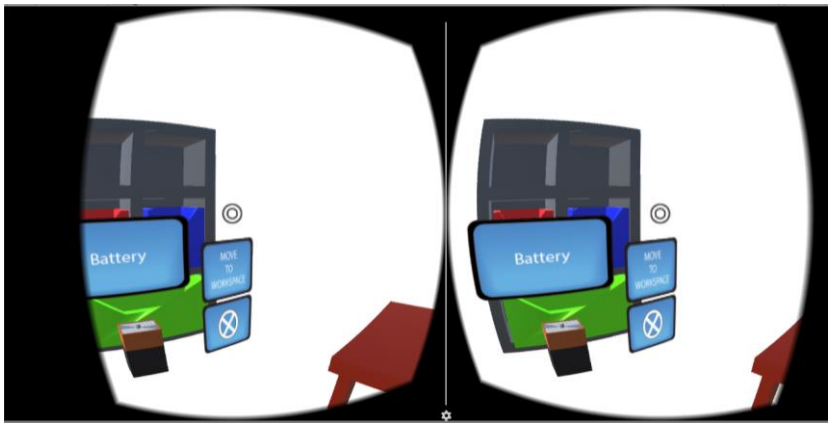


*Figure 26. Battery Draw Selected*

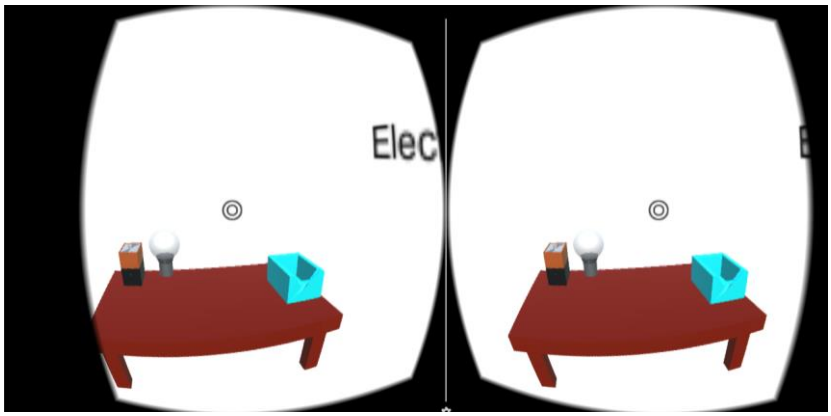


*Figure 27. Battery and Lightbulb in workspace*

Dropbox Link: https://www.dropbox.com/sh/jj8zfv3qnvyriiz/AABrZX5WyshwQkt13-Tda6Fwa?dl=0

In the folder: VR Electronic Lab APK, Unity Package, Final Report, Developer Notes, Supervisor Meeting Notes, Old VR Electronic Lab Unity Builds

**References**

1. Gutl, C. Cheong, C. Cheong, F. Chang, V. Nau, Zaung. Pirker, J. 2015. *Expectations of the generation NeXt in higher education: Learning engagement approaches in information sciences subjects*. Interactive Collaborative Learning (ICL), 2015 International Conference on [Online] Available at: http://ieeexplore.ieee.org/document/7318027/ [Accessed: 10/09/2016]

2. UK Department for Education. 2012. *The Imapact of Pupil Behaviour and Wellbeing on Education Outcomes*. [pdf] London: Department of Education. Available at: https://www.gov.uk/government/publications/the-impact-of-pupil-behaviour-and-wellbeing-on-educational-outcomes [Accessed: 09/09/2016]

3. Education Funding Agencies UK. 2016. *A Annual Report and Accounts for the Year Ended 31 March 2016*. [pdf] Available at: https://www.gov.uk/government/publications/efa-annual-report-and-accounts-for-the-year-ended-31-march-2015 [Accessed: 09/09/2016]

4. U.S. Department of Education. 2016. *US Education Budget Fact Sheet*. [pdf Available at: http://www2.ed.gov/about/overview/budget/budget17/budget-factsheet.pdf [Accessed: 09/09/2016]

5. O.T. Laseinde. S.B. Adejuyigbe. K. Mpofu. H.M/ Campbell. 2015. *Educating tomorrows engineers: Reinforcing engineering concepts through virtual relaity (VR) teaching aid*. Industrial Engineering and Engineering Management (IEEM), 2015 IEEE International Conference on [Online] Available at: http://ieeexplore.ieee.org/document/7385894/ [Accessed: 10/09/2016]

6. Barata, P, Filho, M. Nunes, M. 2015. *Consolidating Learning in Power Systems: Virtal Reality applied to the study of the operation of Electric Power Transformers*. IEEE Transactions on Education ( Volume: 58, Issue: 4, Nov. 2015 ) [Online] Available at: http://ieeexplore.ieee.org/document/7027870/ [Accessed: 12/09/2016]

7. Wang, G. Wu, H. 2008. *Research of Education Game Based on Virtual Reality*. Computational Intelligence and Industrial Application, 2008. PACIIA '08. Pacific-Asia Workshop on [Online] Available at: http://ieeexplore.ieee.org/document/4756910/ [Accessed: 08/09/2016]

8. Sutherland, E. *The ultimate display*. In Proceedings of IFIPS Congress (New York City, NY, May 1965), vol. 2, pp. 506–508. [pdf] Available at: http://worrydream.com/refs/Sutherland%20-%20The%20Ultimate%20Display.pdf [Accessed: 10/09/2016]

9. Abulrub, A. Attridge, A. Williams, M. 2011. *Virtual Reality in engineering education: The future of creative learning*. Global Engineering Education Conference (EDUCON), 2011 IEEE [Online] Available at: http://ieeexplore.ieee.org/document/5773223/ [Accessed: 08/09/2016]

10. The Farm 51. 2015. *VR Market Report*. [pdf] Avaiable at: http://thefarm51.com/ripress/VR_market_report_2015_The_Farm51.pdf [Accessed: 08/09/2016]

11. Superdata Research. 2016. *VR Industry Report 2016*. [pdf] Available at: https://www.superdataresearch.com/market-data/virtual-reality-industry-report/ [Accessed: 08/09/2016]

12. Google. 2016. *Google Store*. [Online] Available at: https://store.google.com/product/google_cardboard [Accessed: 10/09/2016]

13. Borst, C. Ritter, K. Chambers, T. 2016. *Virtual Energy Center for Teaching Alternative Energy Technologies*. Virtual Reality (VR), 2016 IEEE [Online] Available at: http://ieeexplore.ieee.org/document/7504701/ [Accessed: 13/09/2016]

14. Griffiths, M. 2002. *The Educational Benefits of Videogames*. Education and Health Journal (Vol.20 No.3) [pdf] Available at: http://sheu.org.uk/sites/sheu.org.uk/files/imagepicker/1/eh203mg.pdf [Accessed: 05/09/2016]

15. Marthur, A. 2015. *Low cost virtual reality for medical training*. Virtual Reality (VR), 2015 IEEE [Online] Available at: http://ieeexplore.ieee.org/document/7223437/ [Accessed: 07/09/2016]

16. Marklund, B. Backlund, P. Engstrom, H. 2014. *The Practicalities of Educational Games: Challenges of Taking Games into Formal Educational Settings*. Games and Virtual Worlds for Serious Applications (VS-GAMES), 2014 6th International Conference on [Online] Available at: http://ieeexplore.ieee.org/document/7012170/ [Accessed: 10/09/2016]

17. Cobb, S. Nichols, S. Ramser, A. Wilson, J. 1999. *Virtual Reality Induced Symptoms and Effects*. Presence ( Volume: 8, Issue: 2, April 1999 ) [Online] Available at: http://ieeexplore.ieee.org/document/6788197/ [Accessed: 11/09/2016]

18. Kim, Y. Less, G. Jo, D. Yang, U. Kim, G. Park, J. 2011. *Analysis on virtual interaction-induced fatigue and difficulty in manipulation for interactive 3D gaming console*. Consumer Electronics (ICCE), 2011 IEEE International Conference on [Online] Available at: http://ieeexplore.ieee.org/document/5722577/ [Accessed: 11/09/2016]

19. Arns, L. Cerney, M. 2005. *The Relationship between age and incidence of cybersickness among immersive enviroment users.* Virtual Reality, 2005. Proceedings. VR 2005. IEEE [Online] Available at: http://ieeexplore.ieee.org/document/1492788/ [Accessed: 11/09/2016]

20. Gortari, A. 2015. *What can game transfer phenomena tell us about the impact of highly immersive gaming technologies?*. Interactive Technologies and Games (iTAG), 2015 International Conference on [Online] Available at: http://ieeexplore.ieee.org/document/7399494/ [Accessed: 11/09/2016]

21. Oculus. 2016. *Oculus Shop*. [Online] Available at: https://shop.oculus.com/ [Accessed: 10/09/2016]

22. eVRydayVR. 2014. *Oculus Rift DK2 – Unity Tutorial: Reticle*. [Video Online] Available at: https://www.youtube.com/watch?v=LLKYbwNnKDg [Accessed: 07/07/2016]