May 19, 2020

Bradley De Ath

DEA18001355

SCDT45 – Assignment 2

Table of Contents

[**Solution Evaluation** 1](#_Toc40695362)

[Meeting the Requirements 1](#_Toc40695363)

[*Interactivity 1*](#_Toc40695364)

[*Navigate the Information 2*](#_Toc40695365)

[*Chemistry: Simple Compounds 2*](#_Toc40695366)

[*Chemistry & Physics Information Separation 2*](#_Toc40695367)

[Development Tools 3](#_Toc40695368)

[*3DS Max 3*](#_Toc40695369)

[*Photopea 3*](#_Toc40695370)

[Programming Methodology 4](#_Toc40695371)

[**End-User Documentation 4**](#_Toc40695372)

[The Main Menu 4](#_Toc40695373)

[The Biology Branch 5](#_Toc40695374)

[The Chemistry Branch 5](#_Toc40695375)

[The Physics Branch 6](#_Toc40695376)

[**Testing 6**](#_Toc40695377)

[**Testing Appendix 7**](#_Toc40695378)

[**Bibliography 10**](#_Toc40695379)

[**Source Code Appendix 10**](#_Toc40695380)

[SceneChange.cs 10](#_Toc40695381)

[returnButton.cs 12](#_Toc40695382)

[H202Interaction.cs 12](#_Toc40695383)

[CH4Interaction.cs 13](#_Toc40695384)

[textChange.cs 14](#_Toc40695385)

[textChange2.cs 15](#_Toc40695386)

[seriesManage.cs 16](#_Toc40695387)

[manageWires.cs 19](#_Toc40695388)

[cellButtons.cs 22](#_Toc40695389)

# Solution Evaluation

## Meeting the Requirements

For the development of the application, there are several features and pieces of functionality that need to be met for it to meet the requirements of the client. This section of the documentation will identify critical requirements for the software along with evaluating if they are achieved in this version of the application.

### Interactivity

Arguably the most essential requirement for the science experience is for there to be some form of interactivity. With the concept behind the application heavily relying on students being able to have a hands-on experience with 3D representations of essential science entities that may be key to their learning. It has been reported that the more a user interacts with an application, the better received it will be **Kim et al., 2014)**. Without the use of interaction, the system would likely be no different than reading a science textbook, with information simply displayed to users in an unengaging and straightforward design**.**

For the current solution, this requirement has been met, thanks to the use of several interaction methods throughout the application. While still being easy to understand and follow, due to the User Interface containing important instructions to prevent users getting confused or lost. These can be seen below:

For the physics section, keyboard controls can be used to modify and control the layout and power settings for the different circuit options available. With these modification details and the associated keys clearly displayed to the user next to the circuit models themselves, the control information is never hidden away and can be easily identifiable for new users.

For Biology, the interaction relies on users clicking on key identifiers for the plant cell model with their mouse, adding another form of interaction that is common practice for many computer users and so easy to understand. With the instructive text: “Click on a Location Marker to get started” clearly visible on screen accompanied by an example image of what this marker looks like, users are immediately given all the information needed to interact with this wing of the application.

Finally, for the chemistry section of the application, users can interact with the chemical compound models in two distinct forms. The first is through using the arrow keys on their keyboard to rotate the model. This allows for the students to get a better 360-degree view of the compounds, again allowing for a more unique experience that using a typical textbook. The second form of interaction is individually interacting with the different elements that make up each compound. Through clicking on the spheres representing the chemicals, additional information is displayed to the user. These instructions are clearly presented to the user, so once again there is no need to seek out directions for using the system, as they can always be seen upon initially loading the scene.

### Navigate the Information

Another important requirement established by the client is for the students to be able to navigate through the full range of content provided in the application. Without the ability to navigate through the system, students may be unable to experience all of the information provided for the science branches. One of the fundamentals for designing a user interface is for users not to be overwhelmed by the options presented, this is stated in Hick’s Law, where the more choices a user is presented with, the harder it can become for them to reach their intended decision **(Proctor and Schneider, 2018)**. This is another reason why allowing users to navigate through the application is essential, as were all the content available to be displayed on a single screen at once, there would likely be overwhelming an so prevent students from being able to able to effectively learn from the information provided.

Through the use of menus clearly separating the content by the science branch they are related to, users are never presented with an abundant amount of content at any given time, while still being able to access everything that the experience has to offer. With buttons, this allows for users to return to the main menu from any of the science branches, it is simple for students to swap between the content they are viewing at will, without having a large amount of options at once.

### Chemistry: Simple Compounds

With the content of the chemistry branch surrounding chemical compounds and their structures, it is important for the structures to be the correct form for what the client is looking for in the experience. It is stated in the brief that the chemical compounds used are to be simple, with a limited number of elements in the makeup. It is important for the students to not feel overwhelmed when using the system, and so limiting the context to those that are simple is an important requirement to meet.

While both compounds that are used in this application are vastly different in design and properties, both remain simple with only two different elements making up the structure. The first, Hydrogen Peroxide, contains a total of 4 elements in its design, with the chemicals being oxygen and hydrogen. The second compound, Methane, contains 5 instances of the chemical: carbon and hydrogen. These compounds meet the requirements due to their structure being easy to follow and having a limited number of items to interact with while still providing all the necessary information.

### Chemistry & Physics Information Separation

The final of the main requirements further expands on the need for the content of the application to be appropriately separated and divided between different sections. Due to both the chemistry and physics branches having two separate interactive features for their content, having them divided and separated helps to prevent confusion for users. Continuing to build off the principles of Hicks Law, ensuring that users are not presented with too many options for each of the science branches helps to increase usability for the users. With interfaces that are too overwhelming or confusing for their users, tasks that were intended to be completed in the software may take longer or be unachievable all together **(Galitz, 2007)**, with users becoming frustrated or lost attempting to navigate through the content.

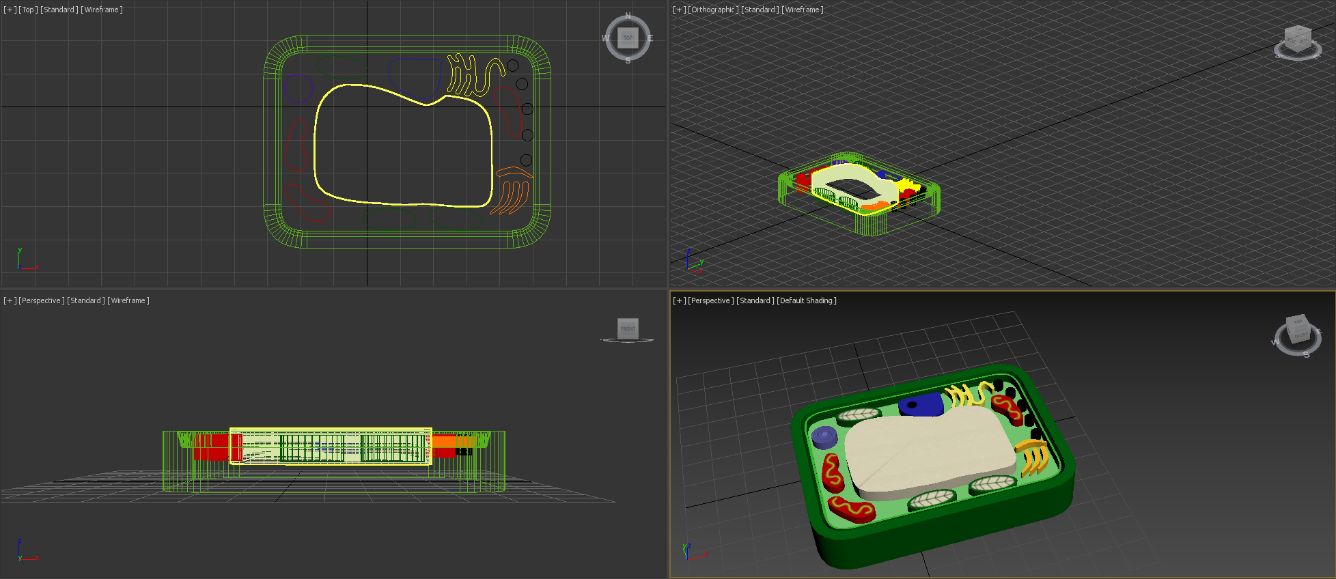
## Development Tools

### 3DS Max

During the development process, several tools were used for developing the 3D models that the users can interact with and are presented with. The primary tool used for this stage of development was 3DS Max. The 3DS Max software contains the tools and features needed to create 3D rendered objects that can be interacted with from a 360-degree angle. With the purpose of The Science Experience relating to students achieving a more interactable education than a typical textbook, the use of a creation software such as 3DS is a suitable fit for providing the necessary content. Through using the tools in this software, detailed models can be created in a range of shapes, which is especially useful for the more intricate designs needed in the experience, such as the biology plant cell. With the high number of components that make up the overall cell’s design, 3DS Max allows for several objects to be created in the same space, so that once they are imported into Unity: can be used as a single object while still having each component be interactable on their own.

### Photopea

Alongside the geometry, the materials are another component of the 3D models that are needed to create a high-quality experience for the students using the application. This can be achieved through using a UV map to assign materials across the 3D surfaces of an object, these ‘maps’ allow for a visual representation of where materials will be applied to base on the different faces on the objects. The Photopea software is a creation application, that allows for art-based tools to be used to modify a canvas, such as a paintbrush or pre-determined shapes. I used Photopea for the materials as it can alter the files created by 3Ds Max of the model’s net, and the tools available can add the details needed to provide the models with more detail.

This can be seen with the components of the biology plant cell, as while the majority of the models in the other branches contain mostly solid colours, the makeup of the plant cell contains specific details, with the screenshot below showing an example:

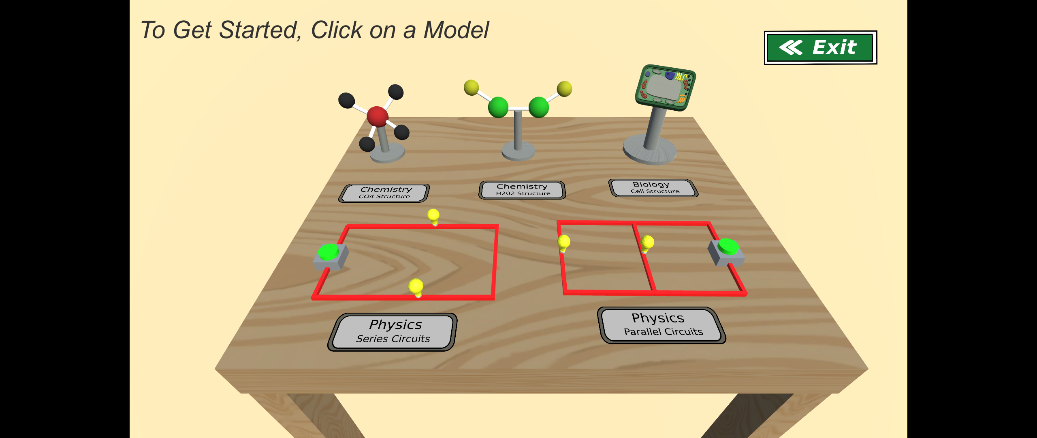
However, this application was not the best choice for modifying these maps, as the Adobe software ‘Photoshop’ contains many of the same tools but to a higher overall quality than the Photopea application that I used **(Blatner and Fraser, 2001)**. This is to be expected due to Photoshop being a more expensive option, with Photopea having no access cost and being available to anyone. Had I used photoshop for the material creation, the overall quality could have been increased but due to the design of the models being a more simplistic approach to keep in line with the textbook-style, I feel the use of Photopea is an acceptable quality.

## Programming Methodology

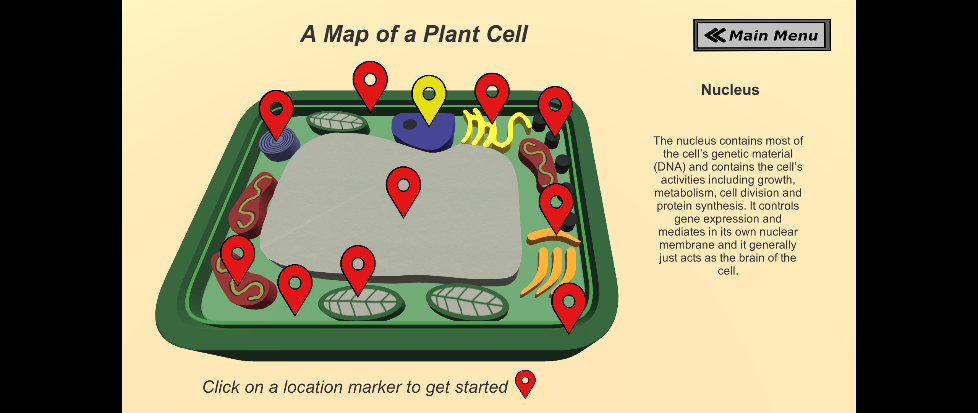
Due to the programming for the application relying on the modification and properties of 3D objects, the used methodology is relatively simple. With each object in a scene being called through the use of ‘Gameobjects’ in the C# program, objects are not needed to be created listing the individual properties such as size, shape and positioning: as these features are already built into the development environment for easy use. This eliminates the need for OOP (Object-Oriented Programming) where each object would have to be instantiated and assigned properties each time they are needed. The items are simply stored in lists to make it easier to organise the Gameobjects by purpose, along with implementing widespread changes to all variables contained within these lists.

# End-User Documentation

## The Main Menu

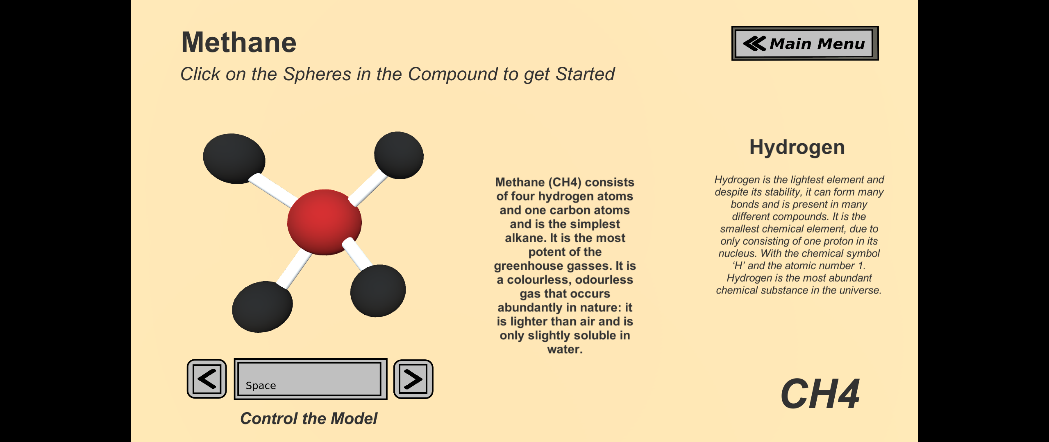
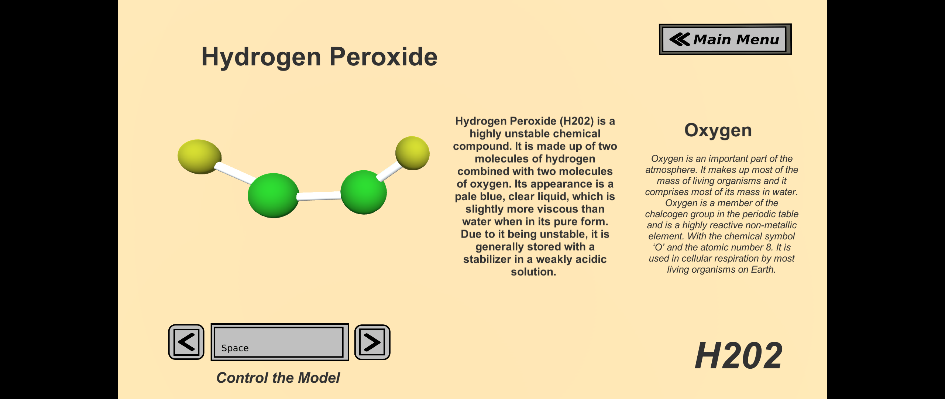
**Upon first loading into the application, you are presented with the main menu for the experience, with links to each of the different branches and scenes available. The table in the centre of the screen contains five models, which are each scale representations of the models that the branches use to provide interactive experiences for the content within. Each model is accompanied with a label that specifies which branch and component it provides a link to (such as the H202 model, which identifies the chemistry branch and the Hydrogen Peroxide compound. All the return buttons on the different branches will bring you back to this single main menu, to allow for easy navigation.

## The Biology Branch

The design of the biology branch revolves around a single 3D object that can be interacted with. The Plant Cell that is presented on screen is covered in markers, like those found on interactive maps, to identify the separate components that make up the structure quickly. With each icon corresponding to an identifiable part of the cell, once clicked on the relevant information for that component is displayed to the right of the model in a size that is easy to read. Once clicked on, that icon will change colour to easily identify which part of the cell is being viewed, this will update every time a new marker is selected.

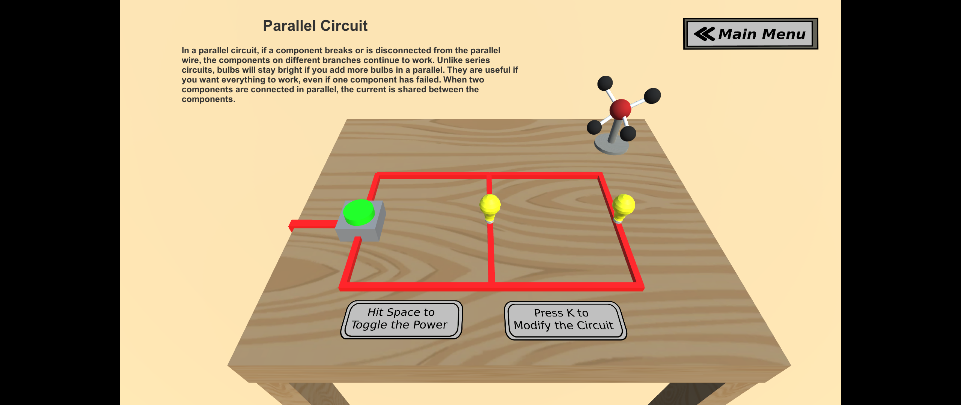
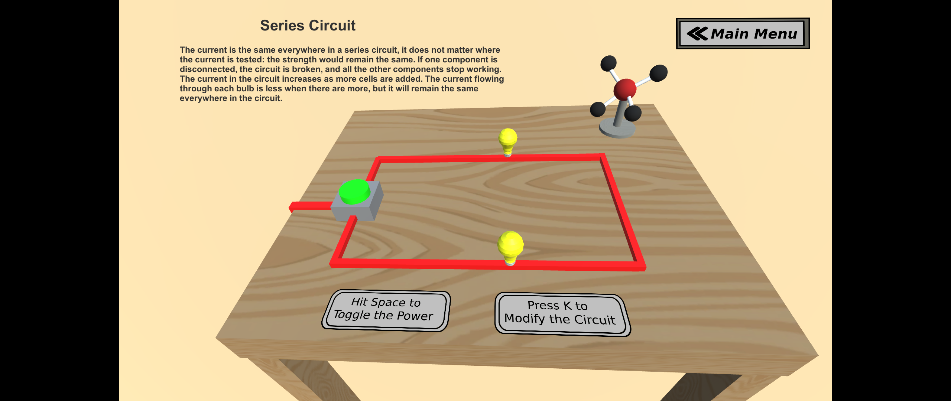
## The Chemistry Branch

While the chemistry branch is split across two scenes, both ae used in the same and contain variations of the same type of content. Once loaded, you will be shown a 3D representation of the selected chemical compound along with a description of the over structure. There are two fundamental forms of interaction for this model: Rotation and Clicking. Through using the arrow keys and space bar as shown in the interface, the model can be rotated horizontally, allowing a 360 degree of the structure of the chemical compound and how the components are connected.

When one of the chemicals that make up the compound are selected, the corresponding information is displayed in a large font on the right of the screen, with the data updating each time a new chemical component is selected.

## The Physics Branch

Similarly, to the chemistry branch, the Physics content of the application is split across two separate scenes; both contain variations of the circuitry information that physics is centred around. The circuits that are displayed can be interacted with by using two different key presses that each impact the display and state of the model. Upon pressing space, the power is toggled for the switch and so powers any connected wires and bulbs provided they are in a usable state.

The second key ‘K’ is used to modify the connections/placement of certain components. In the series circuit, a single wire is displaced, breaking the connection. This is done to provide a visual representation of how this circuit type deals with a break, as the circuit is unable to function, preventing the light from toggling. Similarly, in the Parallel Circuit an extension of the setup is disconnected from the main setup. This allows the bulb connected to switch to toggle while the other is left unpowered. This ‘K’ function can be used to connect and disconnect these components at will.

# Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test No.** | **Test** | **Description** | **Expected Result** | **Actual Result** | **Evidence** |
| **1** | Content Separation | The Content of the Experience is correctly divided within the application | Success | Success | *Testing Appendix Figures 1.1 & 1.2* |
| **2** | Chemistry Separation | The Content of the Chemistry Branch is separated | Success | Success | *Testing Appendix Figures 1.1 & 1.2* |
| **3** | Physics Separation | The Content of the Physics Branch is separated | Success | Success | *Testing Appendix Figures 1.1 & 1.2* |
| **4** | Biology Interaction | The User can interact with the Biology Model | Success | Success | *Testing Appendix Figures 2.1 & 2.2* |
| **5** | Chemistry Interaction | The User can interact with the Chemistry Models | Success | Success | *Testing Appendix Figures 3.1 & 3.2* |
| **6** | Chemistry Rotation | The User can rotate the Chemistry Models | Success | Success | *Testing Appendix Figures 4.1 & 4.2* |
| **7** | Physics Interaction | The User can interact with the Physics Models | Success | Success | *Testing Appendix Figures 5.1 & 5.2 & 5.3* |
| **8** | Displayed Instructions | Instructions for how to use the application are clearly visible | Success | Success | *Testing Appendix Figures 1.1 & 1.2* |
| **9** | Biology Performance | The Loading Time for the Biology Interaction | ~0.3 Seconds | 00.200 Seconds | N/A |
| **10** | Physics Performance | The Loading Time for the Physics Interaction | ~0.3 Seconds | 00.13 Seconds | N/A |
| **11** | Chemistry Performance | The Loading Time for the Chemistry Interaction | ~0.3 Seconds | 00.13 Seconds | N/A |
| **12** | Menu Performance | The Loading Time for using the Main Menu to navigate | ~1 Second | 01.016 Seconds | N/A |

# Testing Appendix

|  |  |
| --- | --- |
|  |  |
| *Figure 1.1 : The Main Menu providing access to each branch of the application.* | *Figure 1.2 : Source Code for how user selections on the Main Menu are handled* |
|  |  |
| *Figure 2.1 : User view of the interaction with the Biology Branch* | *Figure 2.2 : Source Code Extract for handling user interaction in the Biology Branch* |
|  |  |
| *Figure 3.1 : User view of the interaction with the Chemistry Branch* | *Figure 3.2 : Source Code Extract for handling user interaction with the Compound Elements* |
|  |  |
| *Figure 4.1 : User view of the Chemistry Branch after rotation* | *Figure 4.2 : Source Code for allowing the user to rotate the Compound Model at will* |
|  |  |
| *Figure 5.1 : Source Code Extract for handling users’ ‘Space’ interaction in Physics Branch* | *Figure 5.2 : Source Code Extract for handling users’ ‘K’ interaction in Physics Branch* |
|  | |
| *Figure 5.3 : User view of the interaction with the Physics Branch* | |

# Bibliography

Galitz, W.O., 2007. The essential guide to user interface design: an introduction to GUI design principles and techniques. John Wiley & Sons.

Kim, S.Y.S., Prestopnik, N. and Biocca, F.A., 2014. Body in the interactive game: How interface embodiment affects physical activity and health behavior change. Computers in Human Behavior, 36, pp.376-384.

Proctor, R. and Schneider, D. (2018). Hick’s law for choice reaction time: A review. Quarterly Journal of Experimental Psychology, 71(6), pp.1281-1299.

Blatner, D. and Fraser, B., 2001. Real World Photoshop 6. Peachpit Press.

# Source Code Appendix

## SceneChange.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**using** **UnityEngine.SceneManagement**;

**public** **class** **SceneChange** : MonoBehaviour

{

**public** GameObject object1;

**public** GameObject object2;

**public** GameObject object3;

**public** GameObject object4;

**public** GameObject object5;

**public** GameObject object6;

**public** GameObject object7;

**public** GameObject object8;

**public** GameObject object9;

**public** GameObject object10;

**public** GameObject object11;

**public** GameObject object12;

**public** GameObject object13;

**public** GameObject object14;

**public** GameObject object15;

**public** GameObject object16;

**public** GameObject object17;

**public** List<GameObject> wires = **new** List<GameObject>();

**public** List<GameObject> boxes = **new** List<GameObject>();

**public** List<GameObject> lights = **new** List<GameObject>();

**public** GameObject Biology;

**public** GameObject Chemistry1;

**public** GameObject Chemistry2;

**public** GameObject Physics1;

**public** GameObject Physics2;

**public** **void** **LoadBiology**()

{

SceneManager.LoadScene("Biology");

}

**public** **void** **LoadChemistry1**()

{

SceneManager.LoadScene("Chemistry1");

}

**public** **void** **LoadChemistry2**()

{

SceneManager.LoadScene("Chemistry2");

}

**public** **void** **LoadPhysics1**()

{

SceneManager.LoadScene("Physics1");

}

**public** **void** **LoadPhysics2**()

{

SceneManager.LoadScene("Physics2");

}

**public** **void** **ExitSoftware**()

{

Application.Quit();

}

**public** **void** **Start**()

{

wires = **new** List<GameObject>() { object1, object2, object3, object4, object5, object6, object7, object8, object9, object14, object15, object16, object17 };

boxes = **new** List<GameObject>() { object12, object13 };

lights = **new** List<GameObject>() { object10, object11 };

**foreach** (**var** w **in** wires)

{

w.GetComponent<Renderer>().material.color = Color.red;

}

**foreach** (**var** b **in** boxes)

{

b.GetComponent<Renderer>().material.color = Color.grey;

}

**foreach** (**var** l **in** lights)

{

l.GetComponent<Renderer>().material.color = Color.green;

}

}

**public** **void** **Update**()

{

**if** (Input.GetMouseButtonDown(**0**))

{

RaycastHit hit;

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

**if** (Physics.Raycast(ray, **out** hit, **40000.0f**))

{

**if**(hit.transform.name == Biology.name)

{

LoadBiology();

}

**if** (hit.transform.name == Chemistry1.name)

{

LoadChemistry1();

}

**if** (hit.transform.name == Chemistry2.name)

{

LoadChemistry2();

}

**if** (hit.transform.name == Physics1.name)

{

LoadPhysics1();

}

**if** (hit.transform.name == Physics2.name)

{

LoadPhysics2();

}

}

}

}

}

## returnButton.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**using** **UnityEngine.UI**;

**using** **UnityEngine.SceneManagement**;

**public** **class** **returnButton** : MonoBehaviour

{

**public** **void** **returnMenu**()

{

SceneManager.LoadScene(**0**);

}

}

## H202Interaction.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**public** **class** **H202Interaction** : MonoBehaviour

{

**public** GameObject model;

**bool** isSpinning = **false**;

**bool** isBackwards = **false**;

**void** **Start**()

{

}

**void** **Update**()

{

**if** (Input.GetKeyDown(KeyCode.RightArrow))

{

isBackwards = **true**;

isSpinning = **false**;

}

**if** (Input.GetKeyDown(KeyCode.LeftArrow))

{

isBackwards = **false**;

isSpinning = **true**;

}

**if** (Input.GetKeyDown(KeyCode.Space))

{

isBackwards = **false**;

isSpinning = **false**;

}

**if** (isSpinning)

{

model.transform.Rotate(**0f**, **1f**, **0f**);

}

**if** (isBackwards)

{

model.transform.Rotate(**0f**, -**1f**, **0f**);

}

}

}

## CH4Interaction.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class CH4Interaction : MonoBehaviour

{

public GameObject model;

bool isSpinning = false;

bool isBackwards = false;

void Start()

{

}

void Update()

{

if (Input.GetKeyDown(KeyCode.RightArrow))

{

isBackwards = true;

isSpinning = false;

}

if (Input.GetKeyDown(KeyCode.LeftArrow))

{

isBackwards = false;

isSpinning = true;

}

if (Input.GetKeyDown(KeyCode.Space))

{

isBackwards = false;

isSpinning = false;

}

if (isSpinning)

{

model.transform.Rotate(0f, 1f, 0f);

}

if (isBackwards)

{

model.transform.Rotate(0f, -1f, 0f);

}

}

}

## textChange.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**using** **UnityEngine.UI**;

**public** **class** **textChange** : MonoBehaviour

{

**public** Text elementName;

**public** Text elementInfo;

**public** GameObject Hydrogen1;

**public** GameObject Hydrogen2;

**public** GameObject Hydrogen3;

**public** GameObject Hydrogen4;

**public** GameObject Carbon;

**public** GameObject model;

List<GameObject> Hydrogen = **new** List<GameObject>();

List<GameObject> Chemicals = **new** List<GameObject>();

// Start is called before the first frame update

**void** **Start**()

{

Hydrogen = **new** List<GameObject>() { Hydrogen1, Hydrogen2, Hydrogen3, Hydrogen4 };

Chemicals = **new** List<GameObject>() { Hydrogen1, Hydrogen2, Hydrogen3, Hydrogen4, Carbon };

}

// Update is called once per frame

**void** **Update**()

{

**if** (Input.GetMouseButtonDown(**0**))

{

RaycastHit hit;

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

**if** (Physics.Raycast(ray, **out** hit, **40000.0f**))

{

**if**(hit.transform.name == Carbon.name)

{

elementInfo.text = "Despite Carbon’s ability to make 4 bonds and its presence in many compounds, it is highly unreactive in normal conditions. With the chemical symbol ‘C’ and the atomic number 6. Carbon has different allotropes (different forms in which it can exist), which include graphite and diamond, both with vastly different properties.";

elementName.text = "Carbon";

}

**foreach** (**var** h **in** Hydrogen)

{

**if**(hit.transform.name == h.name)

{

elementInfo.text = "Hydrogen is the lightest element and despite its stability, it can form many bonds and is present in many different compounds. It is the smallest chemical element, due to only consisting of one proton in its nucleus. With the chemical symbol ‘H’ and the atomic number 1. Hydrogen is the most abundant chemical substance in the universe.";

elementName.text = "Hydrogen";

}

}

}

}

}

}

## textChange2.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**using** **UnityEngine.UI**;

**public** **class** **textChange2** : MonoBehaviour

{

**public** Text elementName;

**public** Text elementInfo;

**public** GameObject Hydrogen1;

**public** GameObject Hydrogen2;

**public** GameObject Oxygen1;

**public** GameObject Oxygen2;

List<GameObject> Oxygen = **new** List<GameObject>();

List<GameObject> Hydrogen = **new** List<GameObject>();

**void** **Start**()

{

Hydrogen = **new** List<GameObject>() { Hydrogen1, Hydrogen2 };

Oxygen = **new** List<GameObject>() { Oxygen1, Oxygen2 };

}

**void** **Update**()

{

**if** (Input.GetMouseButtonDown(**0**))

{

RaycastHit hit;

Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);

**if** (Physics.Raycast(ray, **out** hit, **40000.0f**))

{

**foreach** (**var** o **in** Oxygen)

{

**if** (hit.transform.name == o.name)

{

elementInfo.text = "Oxygen is an important part of the atmosphere. It makes up most of the mass of living organisms and it comprises most of its mass in water. Oxygen is a member of the chalcogen group in the periodic table and is a highly reactive non-metallic element. With the chemical symbol ‘O’ and the atomic number 8. It is used in cellular respiration by most living organisms on Earth.";

elementName.text = "Oxygen";

}

}

**foreach** (**var** h **in** Hydrogen)

{

**if** (hit.transform.name == h.name)

{

elementInfo.text = "Hydrogen is the lightest element and despite its stability, it can form many bonds and is present in many different compounds. It is the smallest chemical element, due to only consisting of one proton in its nucleus. With the chemical symbol ‘H’ and the atomic number 1. Hydrogen is the most abundant chemical substance in the universe.";

elementName.text = "Hydrogen";

}

}

}

}

}

}

## seriesManage.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**public** **class** **seriesManage** : MonoBehaviour

{

**public** GameObject wire1;

**public** GameObject wire2;

**public** GameObject wire3;

**public** GameObject wire4;

**public** GameObject wire5;

**public** GameObject moveWire;

**public** GameObject powerWire1;

**public** GameObject powerWire2;

**public** List<GameObject> wires = **new** List<GameObject>();

**public** GameObject bulb1;

**public** GameObject bulb2;

**public** GameObject lightObject;

**public** Light toggleLight;

**public** GameObject switchBox;

**public** GameObject switchLight;

**public** Color state;

**public** **bool** flow;

**public** **bool** power;

**public** Material on;

**public** Material off;

**void** **Start**()

{

wires.Add(wire1);

wires.Add(wire2);

wires.Add(wire3);

wires.Add(wire4);

wires.Add(wire5);

wires.Add(moveWire);

switchBox.GetComponent<Renderer>().material.color = Color.gray;

power = **false**;

flow = **true**;

**foreach** (**var** w **in** wires){ w.GetComponent<Renderer>().material.color = Color.gray;}

moveWire.GetComponent<Renderer>().material.color = Color.gray;

toggleLight = lightObject.GetComponent<Light>();

toggleLight.enabled = **false**;

powerWire1.GetComponent<Renderer>().material.color = Color.red;

powerWire2.GetComponent<Renderer>().material.color = Color.red;

}

**void** **Update**()

{

**if** (power)

{

state = Color.red;

switchLight.GetComponent<Renderer>().material.color = Color.green;

bulb1.GetComponent<Renderer>().material = on;

bulb2.GetComponent<Renderer>().material = on;

}

**if** (!power)

{

state = Color.gray;

switchLight.GetComponent<Renderer>().material.color = Color.red;

bulb1.GetComponent<Renderer>().material = off;

bulb2.GetComponent<Renderer>().material = off;

}

**if** (flow)

{

**if** (Input.GetKeyDown(KeyCode.Space))

{

power = !power;

**foreach** (**var** w **in** wires)

{

w.GetComponent<Renderer>().material.color = state;

}

}

**foreach** (**var** w **in** wires)

{

w.GetComponent<Renderer>().material.color = state;

}

}

**if** (!flow)

{

power = **false**;

**foreach** (**var** w **in** wires)

{

w.GetComponent<Renderer>().material.color = state;

}

}

**if** (Input.GetKeyDown(KeyCode.K))

{

**if** (flow)

{

moveWire.transform.Translate(**0**, **0**, **300**);

}

**if** (!flow)

{

moveWire.transform.Translate(**0**, **0**, -**300**);

}

flow = !flow;

}

**if** (power)

{

toggleLight.enabled = **true**;

}

**if** (!power)

{

toggleLight.enabled = **false**;

}

}

}

## manageWires.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**public** **class** **manageWires** : MonoBehaviour

{

**public** GameObject staticWire1;

**public** GameObject staticWire2;

**public** GameObject staticWire3;

**public** GameObject staticWire4;

**public** GameObject movingWire1;

**public** GameObject movingWire2;

**public** GameObject movingWire3;

**public** GameObject movingBulb;

**public** GameObject staticBulb;

**public** GameObject switchBox;

**public** GameObject switchLight;

**public** GameObject powerWire1;

**public** GameObject powerWire2;

**public** GameObject lightObject;

**public** Light toggleLight;

**public** Material active;

**public** Material off;

**public** **bool** on;

**public** Color wire;

**public** **bool** connected;

**public** List<GameObject> staticWires = **new** List<GameObject>();

**public** List<GameObject> movingObjects = **new** List<GameObject>();

**void** **Start**()

{

on = **false**;

connected = **true**;

staticWires.Add(staticWire1);

staticWires.Add(staticWire2);

staticWires.Add(staticWire3);

staticWires.Add(staticWire4);

movingObjects.Add(movingWire1);

movingObjects.Add(movingWire2);

movingObjects.Add(movingWire3);

switchBox.GetComponent<Renderer>().material.color = Color.gray;

powerWire1.GetComponent<Renderer>().material.color = Color.red;

powerWire2.GetComponent<Renderer>().material.color = Color.red;

**foreach** (**var** m **in** movingObjects)

{

m.GetComponent<Renderer>().material.color = Color.gray;

}

**foreach** (**var** s **in** staticWires)

{

s.GetComponent<Renderer>().material.color = Color.gray;

}

toggleLight = lightObject.GetComponent<Light>();

toggleLight.enabled = **false**;

movingBulb.GetComponent<Renderer>().material = off;

}

**void** **Update**()

{

**if** (on)

{

wire = Color.gray;

switchLight.GetComponent<Renderer>().material.color = Color.green;

staticBulb.GetComponent<Renderer>().material = active;

}

**if** (!on)

{

wire = Color.red;

switchLight.GetComponent<Renderer>().material.color = Color.red;

staticBulb.GetComponent<Renderer>().material = off;

}

**if** (Input.GetKeyDown(KeyCode.Space))

{

on = !on;

**foreach**(**var** s **in** staticWires)

{

s.GetComponent<Renderer>().material.color = wire;

}

**if** (connected)

{

**foreach**(**var** m **in** movingObjects)

{

m.GetComponent<Renderer>().material.color = wire;

}

}

**if** (!connected)

{

toggleLight.enabled = **false**;

}

**else** **if** (connected)

{

**if** (!on)

{

toggleLight.enabled = **false**;

movingBulb.GetComponent<Renderer>().material = off;

}

**if** (on)

{

toggleLight.enabled = **true**;

movingBulb.GetComponent<Renderer>().material = active;

}

}

}

**if** (Input.GetKeyDown(KeyCode.K))

{

**if** (connected)

{

movingWire1.transform.Translate(**200**, **0**, **0**);

movingWire2.transform.Translate(**200**, **0**, **0**);

movingWire3.transform.Translate(**0**, **0**, **200**);

movingBulb.transform.Translate(**200**, **0**, **0**);

**foreach** (**var** m **in** movingObjects)

{

m.GetComponent<Renderer>().material.color = Color.gray;

}

toggleLight.enabled = **false**;

movingBulb.GetComponent<Renderer>().material = off;

}

**if** (!connected)

{

movingWire1.transform.Translate(-**200**, **0**, **0**);

movingWire2.transform.Translate(-**200**, **0**, **0**);

movingWire3.transform.Translate(**0**, **0**, -**200**);

movingBulb.transform.Translate(-**200**, **0**, **0**);

**if**(staticWire1.GetComponent<Renderer>().material.color == Color.red)

{

**foreach** (**var** m **in** movingObjects)

{

m.GetComponent<Renderer>().material.color = Color.red;

toggleLight.enabled = **true**;

movingBulb.GetComponent<Renderer>().material = active;

}

}

}

connected = !connected;

}

}

}

## cellButtons.cs

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**using** **UnityEngine.UI**;

**public** **class** **cellButtons** : MonoBehaviour

{

**public** Text componentName;

**public** Text componentInfo;

**public** Button amyloplast;

**public** Button vacuole;

**public** Button membrane;

**public** Button mitochondria;

**public** Button chloroplast;

**public** Button golgi;

**public** Button wall;

**public** Button ribosome;

**public** Button er;

**public** Button nucleus;

**public** Button cytoplasm;

**public** **static** List<Button> buttons = **new** List<Button>();

**public** Sprite unclicked;

**public** Sprite clicked;

**static** **string** n = "";

**static** **string** i = "";

**static** **string** v = "";

**public** **void** **Start**()

{

buttons = **new** List<Button>() { amyloplast, vacuole, membrane, mitochondria, chloroplast, golgi, wall, ribosome, er, nucleus, cytoplasm };

}

**public** **void** **Update**()

{

componentName.text = n;

componentInfo.text = i;

**foreach** (**var** b **in** buttons)

{

**if**(v == b.name)

b.GetComponent<Image>().sprite = clicked;

**else**

{

b.GetComponent<Image>().sprite = unclicked;

}

}

}

**public** **void** **Amyloplast**()

{

n = "Amyloplast";

i = "Amyloplasts are leucoplasts that function mainly in starch storage. They are colourless and are found in plant tissues that do not undergo photosynthesis (like the roots and seeds). They synthesize temporary starch that is not permanently stored in the chloroplasts. They also help to orient root growth downward, towards the directions of gravity.";

v = amyloplast.name;

}

**public** **void** **Vacuole**()

{

n = "Vacuole";

i = "Vacuoles are much larger in plant cells than animal cells. They can store food or any or any variety of nutrients a cell might need to survive. They will even store waste products so that the rest of the cell is protected from contamination. Eventually those waste products will be removed from the cell. The vacuole can sometimes take up more than half the cells’ volume. The contents inside the vacuole are more commonly known as ‘Sap’.";

v = vacuole.name;

}

**public** **void** **Membrane**()

{

n = "Cell Membrane";

i = "All cells are contained by a cell membrane. These are found between the cytoplasm and cell wall of a plant cell. It is semi-permeable, meaning it will allow specific substances to pass through it while preventing the passage of others. It can also be another layer of protection for the contents of the cell.";

v = membrane.name;

}

**public** **void** **Mitochondria**()

{

n = "Mitochondria";

i = "They are known as the powerhouses of the cell. They act like a digestive system in a cell, they take in nutrients, break them down and create energy rich molecules for the cell. They are more commonly recognised for Aerobic Respiration (requires oxygen and glucose to produce carbon dioxide, water and energy), as this respiration’s reactions occur in the mitochondria. They are working organelles that maintain a strong amount of energy for the cell.";

v = mitochondria.name;

}

**public** **void** **Chloroplast**()

{

n = "Chloroplast";

i = "These are not found in animal cells and are key in a plant cell. Photosynthesis takes place in the chloroplast, hence why they are known as the food producers. They work to convert light energy of the sun into sugars that can be used by cells ( this is the photosynthesis process). Chloroplasts all contain chlorophyll, which is what the photosynthesis depends on to occur, the chlorophyll sit on the surface of each chloroplast and capture light energy from the sun. The chloroplast requires carbon dioxide and water to produce glucose and oxygen.";

v = chloroplast.name;

}

**public** **void** **Golgi**()

{

n = "Golgi Body";

i = "The Golgi Body is responsible for transporting, modifying and packaging the proteins and lipids into vesicles that can then be transported to where they are needed within the plant.";

v = golgi.name;

}

**public** **void** **Wall**()

{

n = "Cell Wall";

i = "Cell walls are made of cellulose, and only surround plant cells and a few other organisms. The cellulose inside is a specialised sugar that is classified as a structural carbohydrate. As well as protecting the cell, they also allow plants to grow to great heights. The wall can be slightly elastic for smaller plants, leaves and thin branches. ";

v = wall.name;

}

**public** **void** **Ribosome**()

{

n = "Ribosomes";

i = "Ribosomes are used when a cell is in need to make proteins. They are the protein builders/photosynthesizers of the cell. They can be found floating in the cytoplasm or in the endoplasmic reticulum (ER). The main role is to manufacture and assemble all the proteins in the cell.";

v = ribosome.name;

}

**public** **void** **ER**()

{

n = "ER";

i = "Endoplasmic Reticulum has two major regions: the smooth ER and the rough ER. Rough ER contains ribosomes whilst smooth ER does not. Rough ER manufacturers membranes and synthesises proteins, but smooth ER serves as a transitional area for transport vesicles. The ER in general plays a role in the productions, processing and transport of proteins and liquids.";

v = er.name;

}

**public** **void** **Nucleus**()

{

n = "Nucleus";

i = "The nucleus contains most of the cell’s genetic material (DNA) and contains the cell’s activities including growth, metabolism, cell division and protein synthesis. It controls gene expression and mediates in its own nuclear membrane and it generally just acts as the brain of the cell.";

v = nucleus.name;

}

**public** **void** **Cytoplasm**()

{

n = "Cytoplasm";

i = "Cytoplasm is the liquid that fills a cell, like the blood in other organisms. ‘Cyto’ means cell and ‘plasm’ means blood. The cytoplasm is responsible for giving the cell its shape. They contain molecules such as enzymes which are responsible in the cell for breaking down waste and aid in metabolic activity.";

v = cytoplasm.name;

}

}