Software Major Project: Check 3

By: Tongyu Hu (436016921)
Teacher: Mr Shirlaw
Class: Software Design Year 12
Due date: Week 7 Term 2

User Interface

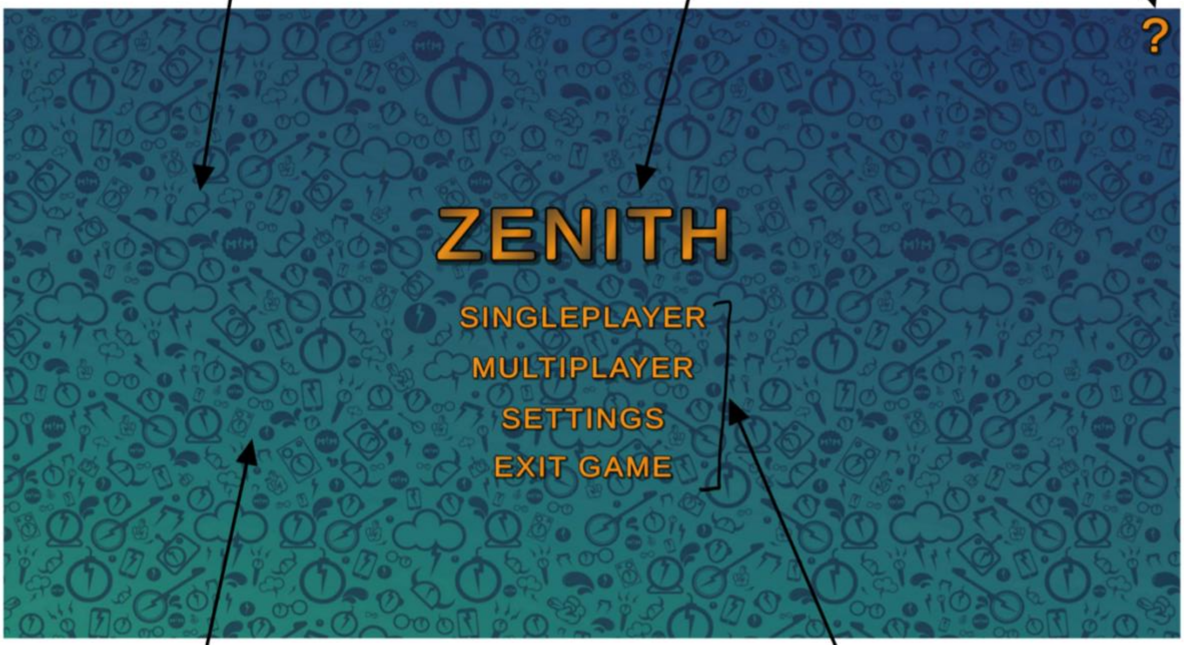
The user interface that has been designed is in accordance with ergonomic considerations that contribute to its overall effectiveness. In consideration to the intended audience, primarily individuals who frequent in playing video games; the interface is highly consistent throughout, making it easy to use and efficient for navigating through the game and improving its overall reception by users. In recognition to inclusivity, a help menu has been built in, along with a settings menu that controls sound, graphics and allows for the user to toggle on fullscreen or windowed mode. This will benefit newer users and with the addition of the font size and colour being bright and visible, will create an intuitive application to use. The use of white space then simplifies the overall feel of the main menu, not overwhelming users but providing them with enough information to easily navigate through the program.

As seen in the example below, there is a clear header of the game logo "Zenith", along with selections for the user to navigate to differing locations within the game itself. These are placed within the centre to improve its visibility with a salient and visible bright orange font along with a contrasting aqua blue background.

Salient aqua blue background with effective use of white space

contrasting Logo in easy to read bold font

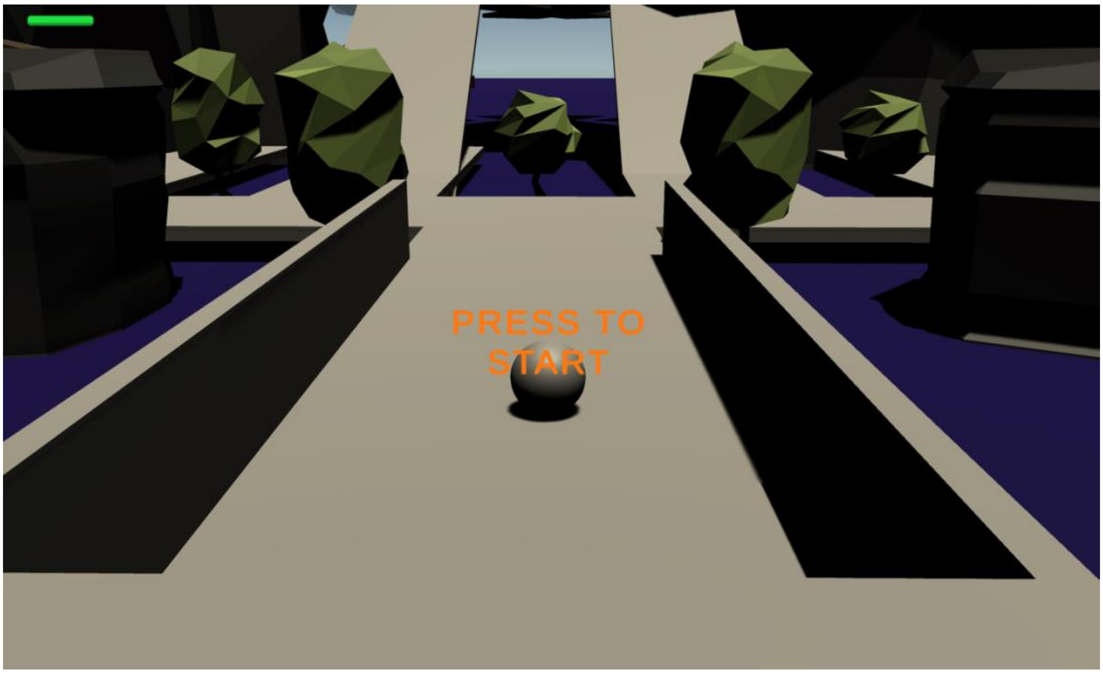
Help Menu (easy to interpret icon)

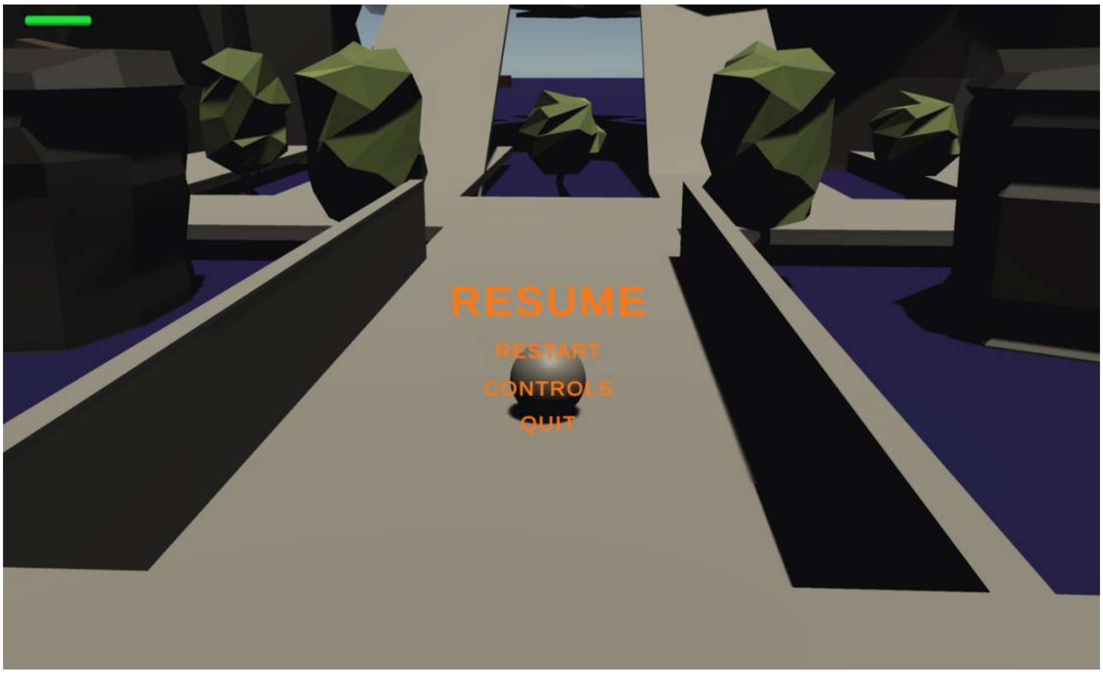


Background with icons to promote user friendliness

Grouping of buttons with Sans serif font (also consistency throughout)

As the user above presses the Singleplayer button, they will be loaded into the game and immediately the initial player view will be displayed with a grey overlay that requires the player to click the button to start. The label on the button is in a bold font named Liberation Sans SDF, making it clear and easy for the user to understand the actions that they need to take. At the top right corner, there exists a health bar, which changes gradients of colour from green to red depending upon the health that the player has, thereby providing visual indication of their health during gameplay. When the escape key is pressed the game pauses, then providing a menu for the user to exit, restart or refer to the controls list, improving ergonomics for the user and a simple way to navigate throughout the whole game. The buttons also darken when the mouse hovers over it, providing further visual indication that it has been pressed.





EBNF and Railroad Diagram

The following EBNF and railroad diagram depicts3D vector in the C sharp language. It provides the basis of many functions that are located within Zenith itself. A valid vector would be Vector3Angle(x: 12.4, y: 8.6, z: 1) and a non-valid vector would be Vector1Dot(x:12.2h, y: 8,9, z: 10).

EBNF of vector:

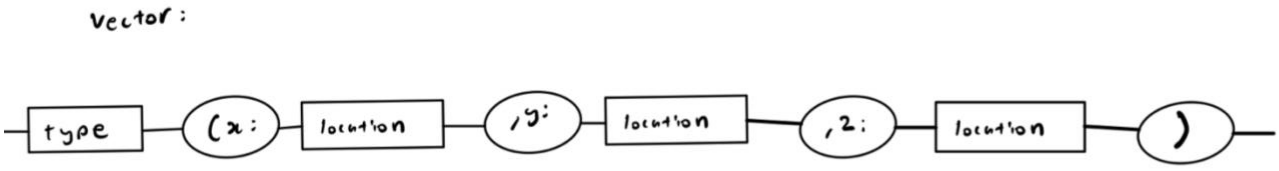
vector = <type> (x: <location>, y: <location>, z:<location>)

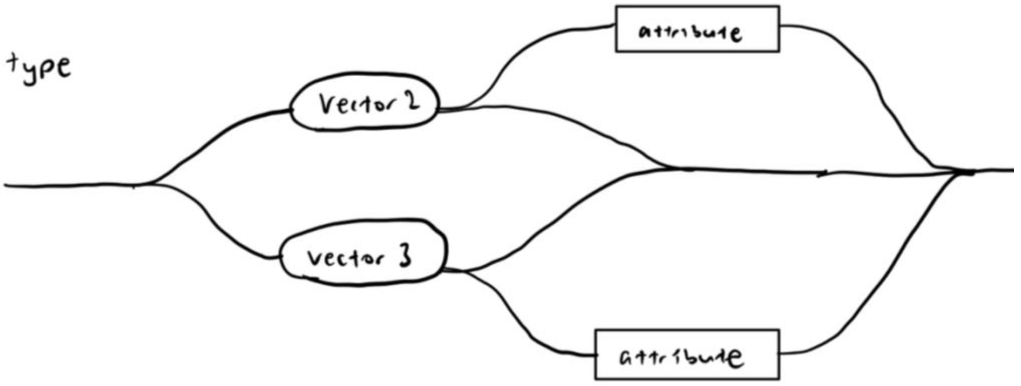
type = Vector2[<attribute>] | Vector3[<attribute>]

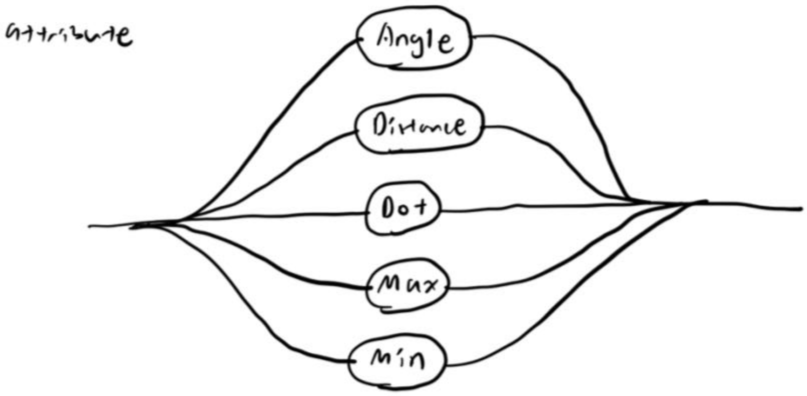
attribute = Angle | Distance | Dot | Max | Min

location = <number> {<number>} [.] {<number>}

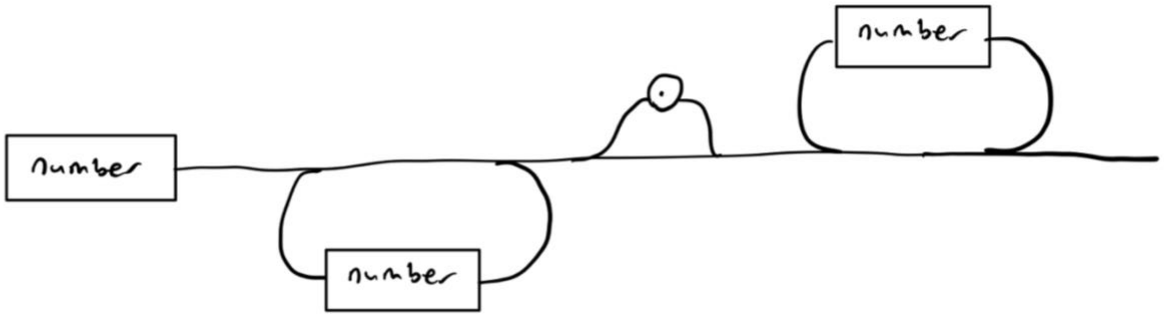
number = 0|1|2|3|4|…|9



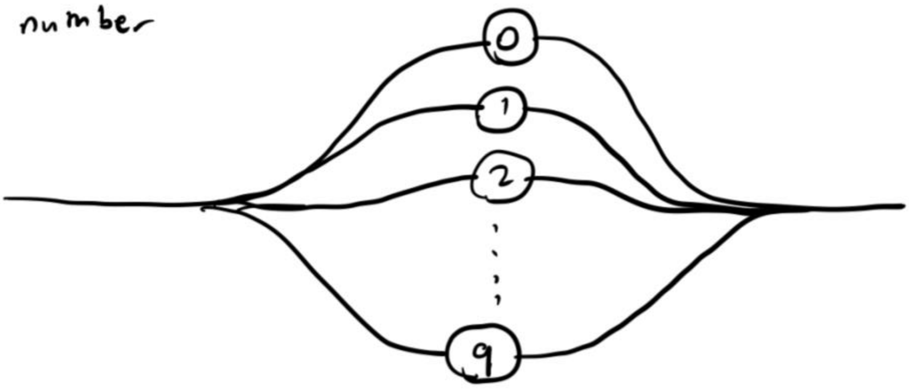




location



number



Test Report

Item being tested: Singleplayer controls module
Date: 19/05/2021
Tester: Tongyu Hu
Type of test: Black box test within alpha phase

Appendix 1: Test data used and the expected versus actual results obtained

Interpretation of test

- Typical and non-typical inputs were all tested, as the module was subjected to thorough testing to ensure no runtime or logic errors are present.
- The issues that were identified are discussed below

1. The control of the 360-player camera for its upward and downward motion seemed to be inverted to what was expected (downwards mouse movement was not equal to downwards pan of camera)
2. To control the player, only the W, A, S, D keys functioned, but not the keyboard arrows, which was unexpected
3. The press of the escape button paused the game, though the navigation menu to resume, check controls or quit the game did not show, preventing the user from controlling the game further

Appendix 1: Table of expected output versus actual output

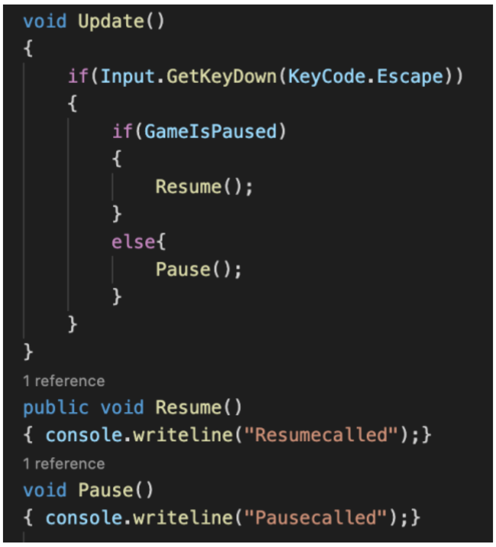
Input	Expected Output	Actual Output	Accurate?
Keyboard press of 'W'	Player moves forward	Moves forward	Yes
Keyboard press of 'S'	Player moves backward	Moves backward	Yes
Keyboard press of 'A'	Player moves left	Moves left	Yes
Keyboard press of 'D'	Player moves right	Moves right	Yes
Keyboard press of 'up arrow'	Player moves forward	Does not move	No
Keyboard press of 'down arrow'	Player moves backward	Does not move	No
Keyboard press of 'left arrow'	Player moves left	Does not move	No
Keyboard press of 'right arrow'	Player moves right	Does not move	No
Keyboard press of 'up'	Player jumps up	Player jumped	Yes

Keyboard press of ‘esc’	Pauses game and brings up menu of options	Paused game though menu of options did not come up	No
Keyboard press of ‘B’	Nothing happens	Nothing happened on screen	Yes
Keyboard press of ‘$’	Nothing happens	Nothing happened on screen	Yes
Mouse moves to right	Pans player camera to the left	Pans player camera to the left	Yes
Mouse moves to left	Pans player camera to the right	Pans player camera to the right	Yes
Mouse moves up	Pans player camera down	Pans player camera up	No
Mouse moves down	Pans player camera up	Pans player camera down	No

Error Checking

1. Stubs

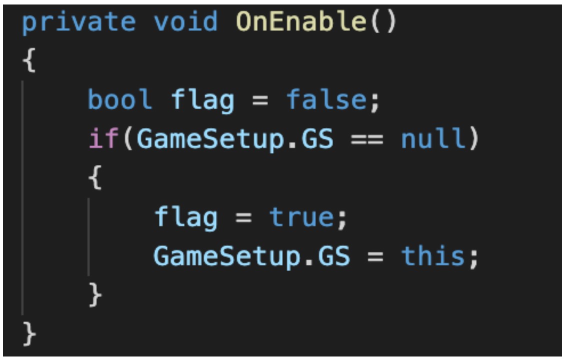
A stub is a small subroutine that is used in place of a yet to be coded subroutine. This allows for higher level subroutines to be tested within a top-down design, thus checking their execution so that they can be sure that it performs its required task. Stubs are usually an empty function that do not perform any real processing and aim to simulate processing by producing a debugging output statement to confirm that it has been called. In the example on the right, the subroutines of Resume() and Pause() acts as stubs and are yet to be coded. The Update () subroutine calls these empty functions which will print a statement of “Resumecalled” or “Pausecalled” to confirm to the developer that the lower-level subroutine is called.



```
void Update()
{
    if(Input.GetKeyDown(KeyCode.Escape))
    {
        if(GameIsPaused)
        {
            Resume();
        }
        else{
            Pause();
        }
    }
}
1 reference
public void Resume()
{ console.writeline("Resumecalled");}
1 reference
void Pause()
{ console.writeline("Pausecalled");}
```

2. Flags

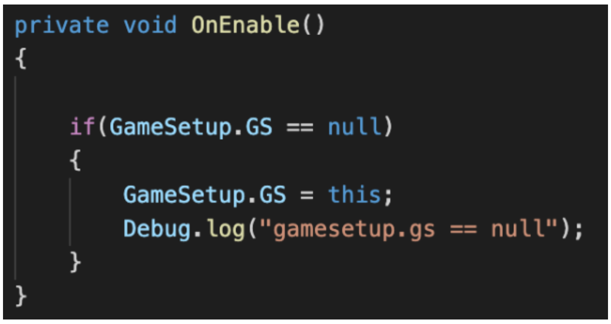
Flags are used to indicate a certain condition has been met and acts as a Boolean variable that is set to either true or false. It allows for developers to check that certain sections of code have been executed or conditions have been met, thereby making it easier to find the source of bugs and issues. To the right, the OnEnable function initially sets the flag to false, indicating that the GameSetup.GS is not null though if it is, the flag will become true, informing that the if statement has been run.



```
private void OnEnable()
{
    bool flag = false;
    if(GameSetup.GS == null)
    {
        flag = true;
        GameSetup.GS = this;
    }
}
```

3. Debugging output statements

Debugging output statements are strategically placed temporary output statements that can be made by developers to isolate sources of error within a piece of code. It can help to determine if a subroutine has been called and allow the flow of execution to be observed. On the right, the Debug.log statement will display the orange message in the console if the if statement is true.



```
private void OnEnable()
{

    if(GameSetup.GS == null)
    {
        GameSetup.GS = this;
        Debug.log("gamesetup.gs == null");
    }
}
```

Desk Check of Module

The desk check below is of the screen resolutions, where it is initially obtained from the laptop itself, and then appended into a list, while also checking for the current resolution of the screen.
Let available screen resolutions be: [[2440,2320], [1800,1300], [1600,900], [1200,800]] for desk check.
Let current screen resolution be [1800,1300]
Note: [x, y] = [width, height]

res	i	curResIn	options	reslen	res[i].wid	res[i].hei	scrreswid	scrreshei
[[2440,2320],	0	0	[]	4			1800	1300
[1800,1300],			[[2440,2320]]		2440	2320		
[1600,900],	1	1	[[2440,2320],[1800,1300]]		1800	1300		
[1200,800]]	2		[[2440,2320],[1800,1300], 1600,900]]		1600	900		
	3		[[2440,2320], [1800,1300], [1600,900], [1200,800]]		1200	800		

Evidence of Breakpoint, Traces and Single Line Stepping

1. Breakpoint

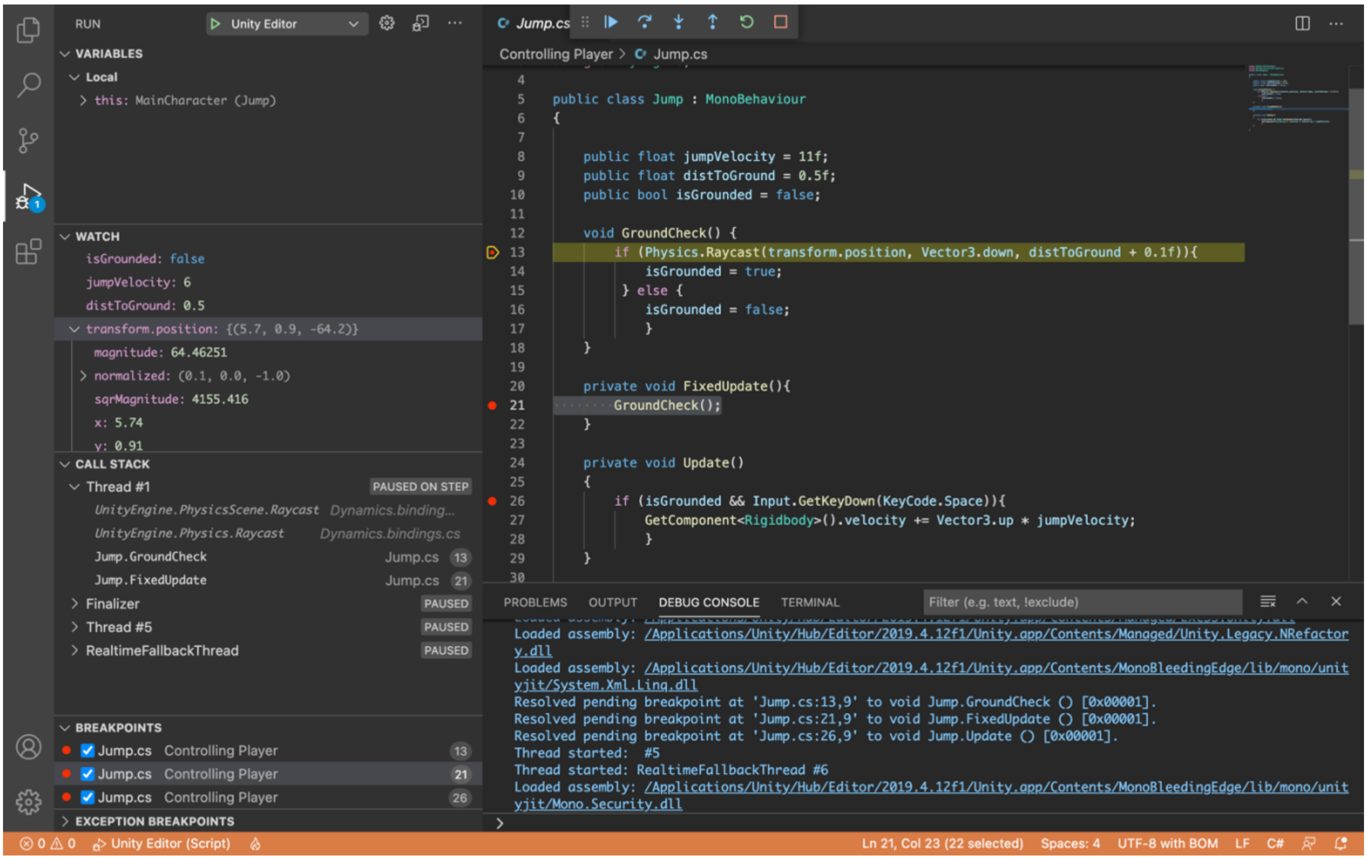
Breakpoints are used to temporarily halt the execution of code to examine the current values of variables at that time. This allows for sources of errors to be found and diagnosed, which can help to find logic or runtime errors, so that they can be fixed. Within most development environments, such as visual studio code, the addition of breakpoints to the program is simple, by clicking on the line number displayed on the left to add a red dot that signifies to the program to halt execution once the line of code is reached. On the image below, the red dots are all placed as breakpoints so that execution can be halted, and the contents of the variable can be analysed.

2. Traces

The use of traces within software debugging allows for the following or tracking of variables and the event sequence that have been processed. In addition to this, some programming environments also allow the developer to analyse the call stack which gives a list of subroutine names that have been called. This allows for the flow of execution to be observed and software to be debugged. From the image below, we can see the listed call stacks and the calls that it has made to the subroutine GroundCheck().

3. Single Line Stepping

Single line stepping is the process of halting execution after each statement is executed and is usually highlighted at each statement. Different environments allow for variations in single line stepping though in visual studio code, it is fairly synonymous to traditional means of stepping through the code line by line. From the image below, the top middle menu allows for single line stepping or 'stepping over' through the lines of code after the breakpoint has been added.



```
VARIABLES
  Local
    this: MainCharacter (Jump)

WATCH
  isGrounded: false
  jumpVelocity: 6
  distToGround: 0.5
  transform.position: {(5.7, 0.9, -64.2)}
    magnitude: 64.46251
    normalized: (0.1, 0.0, -1.0)
    sqrMagnitude: 4155.416
    x: 5.74
    y: 0.91

CALL STACK
  Thread #1                                          PAUSED ON STEP
    UnityEngine.PhysicsScene.Raycast   Dynamics.binding...
    UnityEngine.Physics.Raycast        Dynamics.bindings.cs
    Jump.GroundCheck                   Jump.cs   13
    Jump.FixedUpdate                   Jump.cs   21
  Finalizer                                          PAUSED
  Thread #5                                          PAUSED
  RealtimeFallbackThread                             PAUSED

BREAKPOINTS
  ☑ Jump.cs  Controlling Player   13
  ☑ Jump.cs  Controlling Player   21
  ☑ Jump.cs  Controlling Player   26
EXCEPTION BREAKPOINTS
```

```

public class Jump : MonoBehaviour
{

    public float jumpVelocity = 11f;
    public float distToGround = 0.5f;
    public bool isGrounded = false;

    void GroundCheck() {
        if (Physics.Raycast(transform.position, Vector3.down, distToGround + 0.1f)){
            isGrounded = true;
        } else {
            isGrounded = false;
            }
    }

    private void FixedUpdate(){
        GroundCheck();
    }

    private void Update()
    {
        if (isGrounded && Input.GetKeyDown(KeyCode.Space)){
            GetComponent<Rigidbody>().velocity += Vector3.up * jumpVelocity;
            }
    }

```

```
Loaded assembly: /Applications/Unity/Hub/Editor/2019.4.12f1/Unity.app/Contents/Managed/Unity.Legacy.NRefactor
y.dll
Loaded assembly: /Applications/Unity/Hub/Editor/2019.4.12f1/Unity.app/Contents/MonoBleedingEdge/lib/mono/unit
yjit/System.Xml.Linq.dll
Resolved pending breakpoint at 'Jump.cs:13,9' to void Jump.GroundCheck () [0x00001].
Resolved pending breakpoint at 'Jump.cs:21,9' to void Jump.FixedUpdate () [0x00001].
Resolved pending breakpoint at 'Jump.cs:26,9' to void Jump.Update () [0x00001].
Thread started:  #5
Thread started: RealtimeFallbackThread #6
Loaded assembly: /Applications/Unity/Hub/Editor/2019.4.12f1/Unity.app/Contents/MonoBleedingEdge/lib/mono/unit
yjit/Mono.Security.dll
```

```
⊗ 0 ⚠ 0    Ln 21, Col 23 (22 selected)   Spaces: 4   UTF-8 with BOM   LF   C#
```

Readability of Code

1. **Code Comments**

Code comments are used to explain the function of a section of code in cases where the purpose of specific subroutines may be unclear to other developers. Good comments should explain what a section of code does rather than explain its logic since it should be visible from the code itself.

2. **White Space**

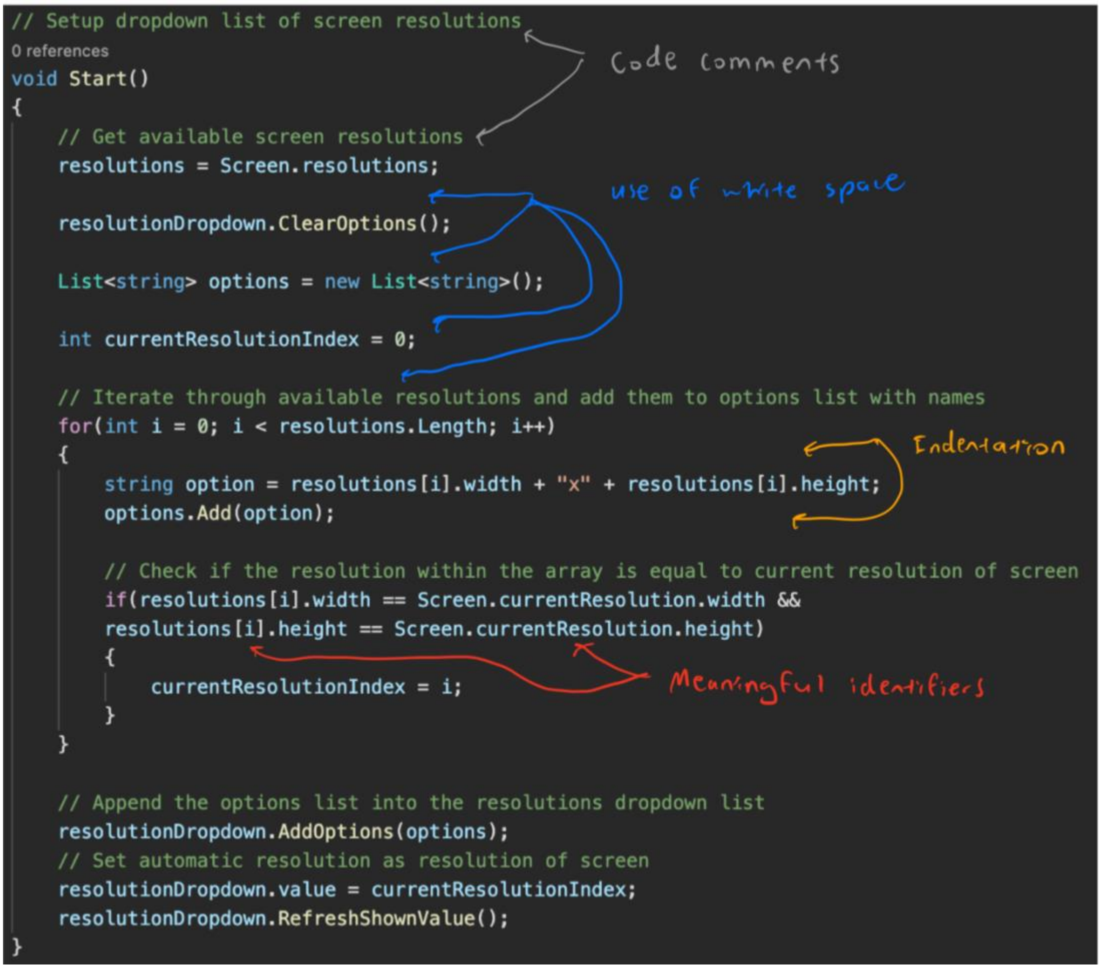
The use of white space allows for code to be viewed in a more appealing and easier to understand way. The empty black lines provide space between groups of code so that functions and statements can be separated.

3. **Indentation**

The use of indentation allows for code to be easily read and understood in terms of their scope and how they fit in with other control structures such as loops. This enhances the understanding of logistics within code for other developers to view.

4. **Use of meaningful identifiers**

Meaningful identifiers supplement the logic of the code and ultimately makes the code much easier to understand in a more human-like manner. This means that variables are assigned a name that is attributed to their purpose, enhancing the readability of code.



```
// Setup dropdown list of screen resolutions
0 references
void Start()
{
    // Get available screen resolutions
    resolutions = Screen.resolutions;

    resolutionDropdown.ClearOptions();

    List<string> options = new List<string>();

    int currentResolutionIndex = 0;

    // Iterate through available resolutions and add them to options list with names
    for(int i = 0; i < resolutions.Length; i++)
    {
        string option = resolutions[i].width + "x" + resolutions[i].height;
        options.Add(option);

        // Check if the resolution within the array is equal to current resolution of screen
        if(resolutions[i].width == Screen.currentResolution.width &&
        resolutions[i].height == Screen.currentResolution.height)
        {
            currentResolutionIndex = i;
        }
    }

    // Append the options list into the resolutions dropdown list
    resolutionDropdown.AddOptions(options);
    // Set automatic resolution as resolution of screen
    resolutionDropdown.value = currentResolutionIndex;
    resolutionDropdown.RefreshShownValue();
}
```

Logic, Runtime and Syntax Errors

1. Syntax errors

Syntax errors occur within the translation process of source code into machine code. It refers to errors that are present within source code statements due to faults that do not adhere to the rules of the programming language (i.e., EBNF and Railroad diagrams). Because of this, the source code will be unable to be transformed to machine code and must be fixed. IDE's usually will provide indications upon these errors as they undergo lexical and syntactical analysis before code generation.

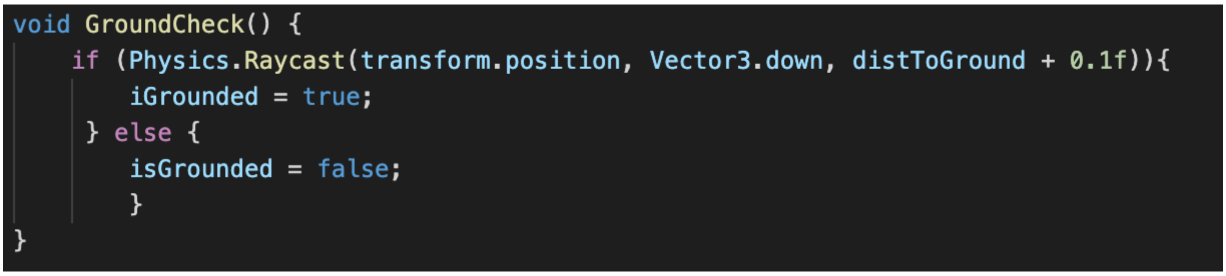
The initial process of translating source code is lexical analysis, where each word is examined to ensure that it is a valid part of the language when compared with EBNF. It ensures that the identifiers, constants, reserved words, and operators are legitimate through a table of symbols or tokens. This is done when a lexeme matches an element in the symbol table, it will be replaced by an appropriate token. If some words are not valid, then the entire source code will be unable to be tokenised, thus producing an error. If all words are valid, and it is tokenised, it will be sent to syntactical analysis.

Syntactical analysis then undergoes the checking of syntax/grammar of sentences and phrases to analyse their adherence to EBNF rules. This will be done through parse trees which checks each statement against precise descriptions of syntax. If a particular component of source code is unable to be parsed, this means that there is an incorrect arrangement of tokens and thus, an error message is generated.

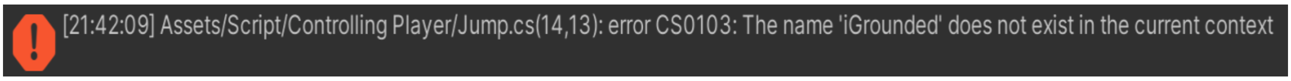
From the above description, syntax errors are fairly common in modern day coding, as slight mistakes by humans will introduce an error. Although this is the case, the IDE's that we code in such as visual studio code introduces error messages that locate the source of error, making it usually easy to correct.

An example of a syntax error is shown below from the code of Zenith:

- The iGrounded variable does not exist and was a typo meant to be spelt isGrounded



```
void GroundCheck() {
    if (Physics.Raycast(transform.position, Vector3.down, distToGround + 0.1f)){
        iGrounded = true;
     } else {
        isGrounded = false;
        }
}
```



```
[21:42:09] Assets/Script/Controlling Player/Jump.cs(14,13): error CS0103: The name 'iGrounded' does not exist in the current context
```

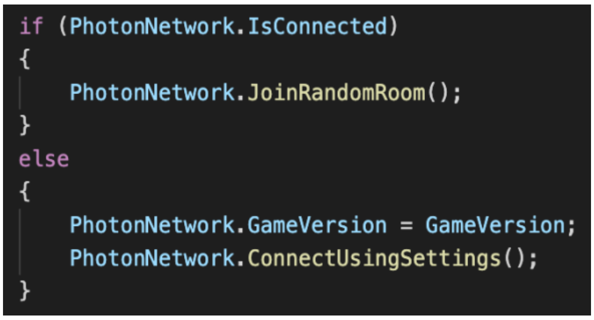
2. Logic errors

Logic errors occur on the basis that the code is syntactically correct, though the expected output does not correspond to the actual output. Logic errors could cause the program to halt or even continue to run incorrectly, thus making them hard to solve as IDE's are unable to debug them as they cannot be aware of the intentions of the programmer itself. Logic errors can occur when converting written algorithms into a particular programming language, which can cause errors that are unexpected.

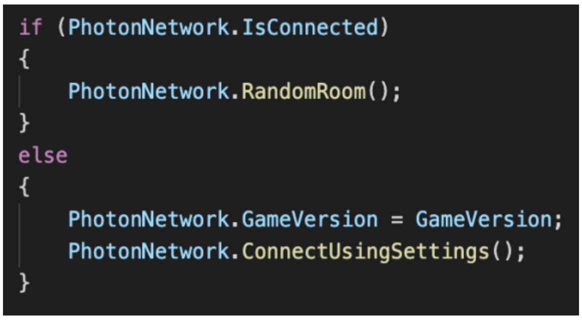
To minimise these errors, thorough testing must be undertaken to minimise its effects, or it will be a large contributor to the time taken to fix and update the game. In addition to this, good programming practices such as use of meaningful identifiers, intrinsic documentation, parameter passing all contribute to an easier time to debug a logic error, so that a well thought out and quality software solution can be produced.

Below shows a logic error within the multiplayer connection of two players

- Upon establishing connection with the server (i.e. PhotonNetwork.IsConnected) the program should join a lobby/room so the player can wait within it
- The image on the left shows the correct logical function to call while the image on the right shows an incorrect logic error which calls the function RandomRoom that does not perform the expected task



```
if (PhotonNetwork.IsConnected)
{
    PhotonNetwork.JoinRandomRoom();
}
else
{
    PhotonNetwork.GameVersion = GameVersion;
    PhotonNetwork.ConnectUsingSettings();
}
```

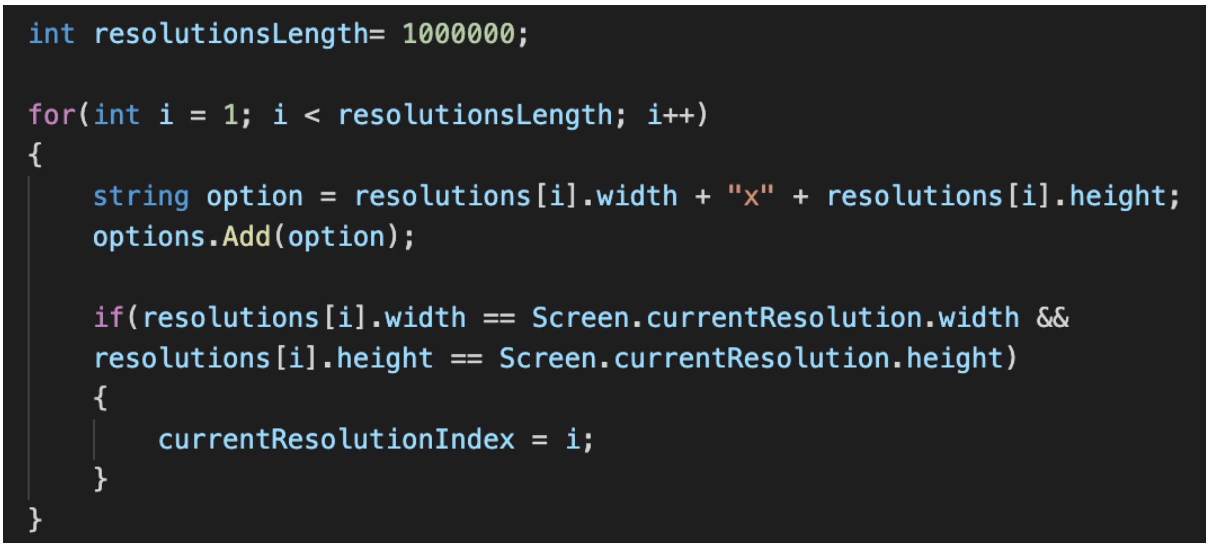


```
if (PhotonNetwork.IsConnected)
{
    PhotonNetwork.RandomRoom();
}
else
{
    PhotonNetwork.GameVersion = GameVersion;
    PhotonNetwork.ConnectUsingSettings();
}
```

3. Runtime errors

Runtime errors occur while the program is executed, and usually will cause the program to stop running due to a fatal error that prevents it from continuing. This means that the program may become frozen or crash and usually is unexpected to the developer. This error could be hardware and software issues such as with the OS (operating system), BIOS (basic input, output system) or hardware drivers. In addition to this, within applications, runtime issues can occur with arithmetic overflows, division by zero, accessing inappropriate memory locations or accessing an index that is out of an array. Because of the above issues, programming environments allow for breakpoints or other forms of error detection to help developers solve runtime errors.

Below is an example of a runtime error, where the program will try to access an index out of range within the array of resolutions, thus causing the program to halt and stop its execution.



```
int resolutionsLength= 1000000;

for(int i = 1; i < resolutionsLength; i++)
{
    string option = resolutions[i].width + "x" + resolutions[i].height;
    options.Add(option);

    if(resolutions[i].width == Screen.currentResolution.width &&
    resolutions[i].height == Screen.currentResolution.height)
    {
        currentResolutionIndex = i;
    }
}
```

User Documentation

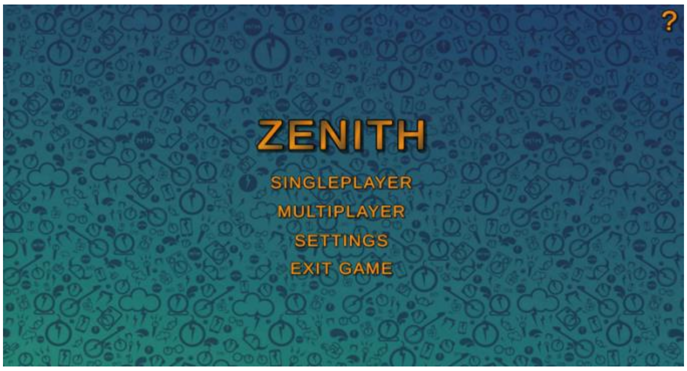
1. User Manual

Welcome to Zenith!

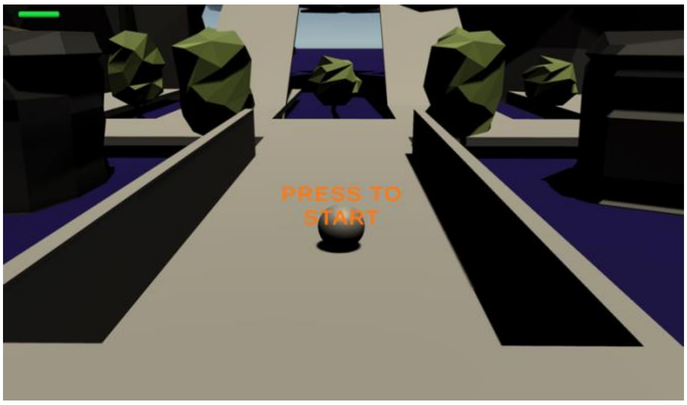
This online multiplayer game provides an effortless distraction from your worries and dilemmas of modern life. With a fun and intuitive UI, you will be lost within the game itself, where you will control a ball going through obstacles and levels until either you or your friend reaches the end first. Not up for multiplayer? Then go and experience the singleplayer version; a much more relaxed version of the game, allowing you to train your skills before you get into the multiplayer world just so you can achieve victory over your friends!

Upon the opening of the application, you will be presented with the main menu, consisting of five options:

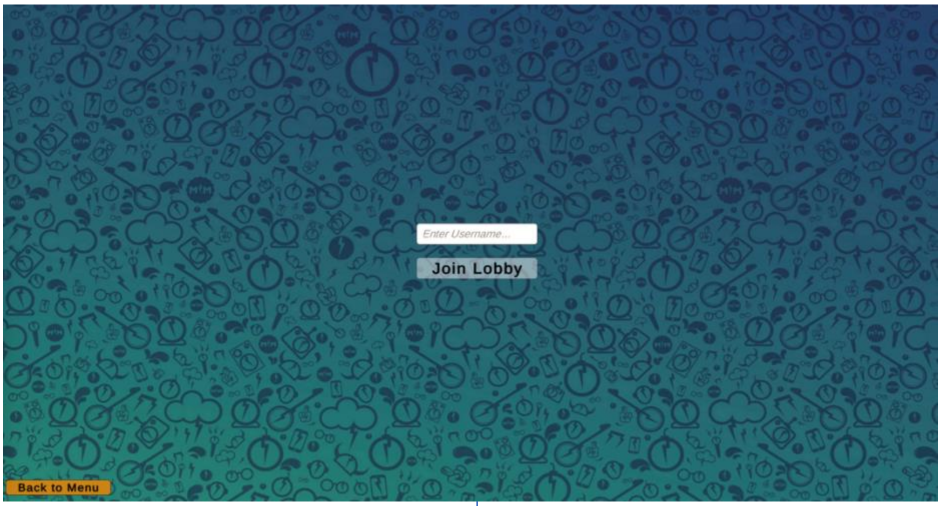
- Singleplayer
- Multiplayer
- Help Menu (Top right)
- Settings
- Exit Game

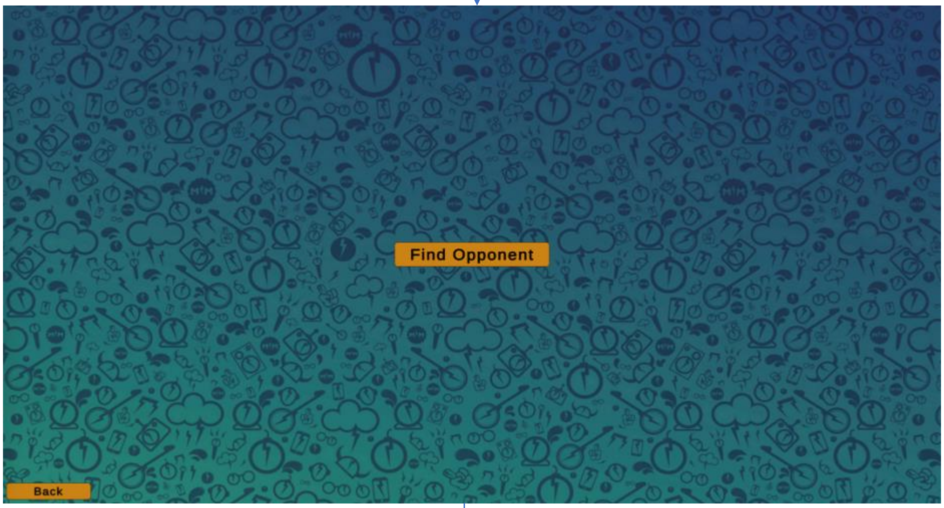


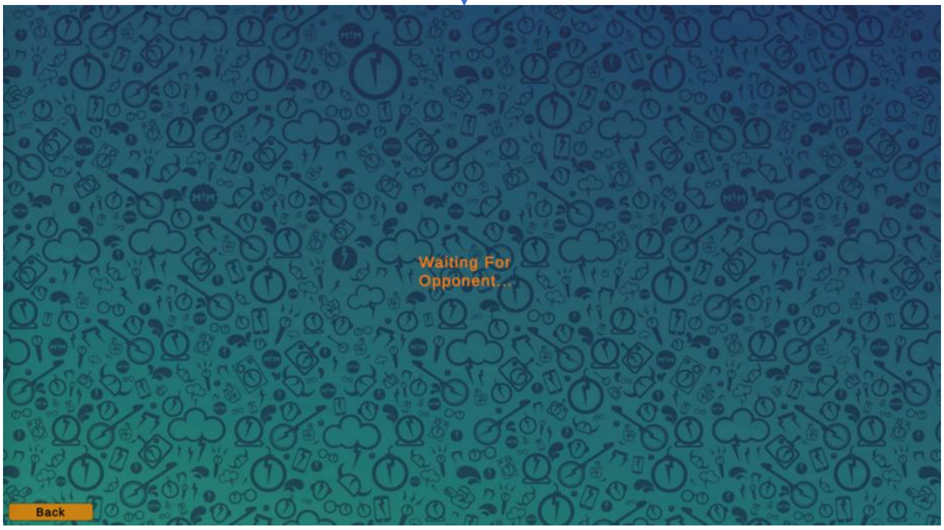
The first button (singleplayer) will immediately bring you to the singleplayer mode of the game, allowing you to begin your experience!



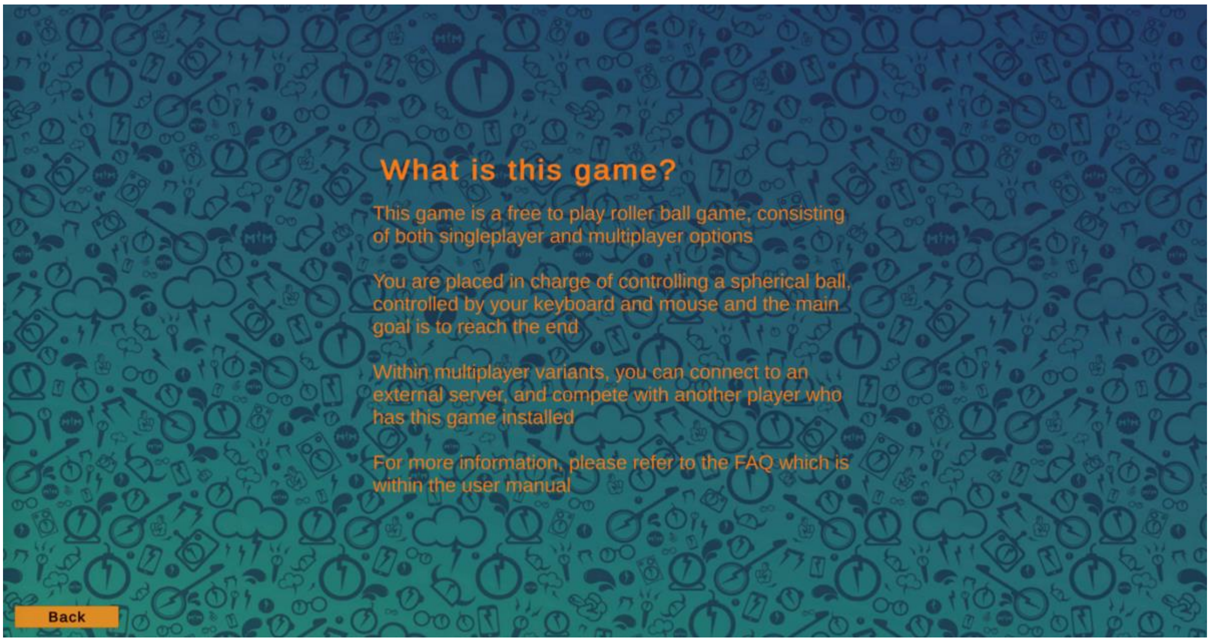
The second button (multiplayer) will take you to a screen where there is a prompt for you to enter a username so you can join a lobby (Don’t worry, your username will be saved the next time you run the application). After doing so, you will be waiting for another player, and once you establish connection with them, the game will automatically load.






Waiting For Opponent...

The third button (Help Menu) is located on the top right and embedded within the ? icon. This will bring you to a page that gives a description of the game, and informs newer players on the basics of the game itself.



What is this game?

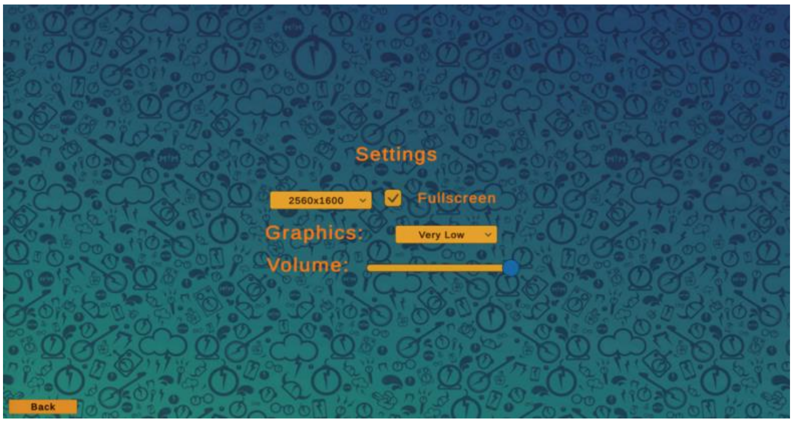
This game is a free to play roller ball game, consisting of both singleplayer and multiplayer options

You are placed in charge of controlling a spherical ball, controlled by your keyboard and mouse and the main goal is to reach the end

Within multiplayer variants, you can connect to an external server, and compete with another player who has this game installed

For more information, please refer to the FAQ which is within the user manual

The fourth button (Settings) will bring you to the settings menu, where you will be able to adjust volume, toggle fullscreen or windowed mode, adjust quality and adjust the resolution. This allows for you to tune the game to your preferred settings, providing a more immersive experience!



Settings

2560x1600 ☑ Fullscreen

Graphics: Very Low

Volume:

The fifth button will (sadly) exit the game, closing the application automatically.

For more information, please refer to the FAQ online and the installation guide provided.

2. Online help

<u>Below are some sample questions that would be found on the online FAQ</u>

1. *What platforms can I play this game on?*

Zenith has been currently developed for Mac OS, Windows and Linux platforms, the majority of which modern computers use on a daily basis. To simply run the game once it has been installed (refer to installation guide), locate it within your applications and click on it, and the game will subsequently run.

2. *Why is my game lagging when I am playing?*

As Zenith is still under current development, it has not been fully optimised for all devices and as such, you may experience drops in framerate or connection issues during gameplay. If so, please report these and as development continues, these issues can be fixed through software updates. Also, consider the hardware you are running it on and how powerful it is.

3. *I am unable to connect to the server and with others, what do I do?*

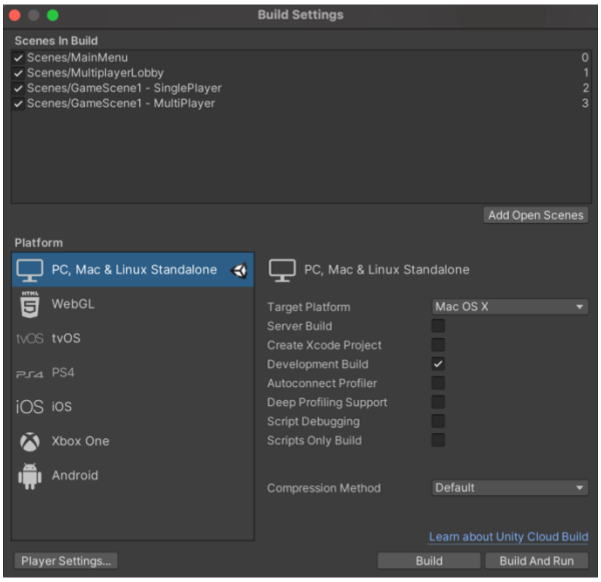
When utilising the multiplayer aspect of the application, ensure that you have a stable and sufficient internet connection of at least 10Mbps, so that you can run the game smoothly. In addition, check to see if your device is connected to the internet and ensure that you are on the same network as your friend so that a connection can be established between your two devices across the server.

4. *What are the controls for the game?*

To refer to the controls list during gameplay, please press the escape key and there you will see a button named ‘controls’. Click on the button and you will be brought to a page that lists out all the controls.

3. Installation

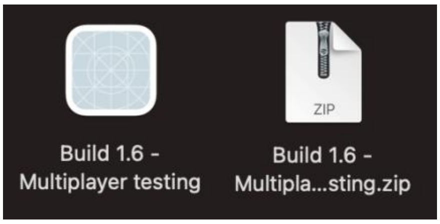
At the current point in development, the game is not yet able to be distributed to a large audience, but rather only shared between beta testers for feedback on the game. The game will initially be built from the developer's device by using the Unity IDE. This will create an application file that can be zipped. Once this is zipped, it can be sent via email or shared via cloud services such as google drive to other people. To download and run the game, users will follow the steps below.



✓ Scenes/MainMenu 0
✓ Scenes/MultiplayerLobby 1
✓ Scenes/GameScene1 - SinglePlayer 2
✓ Scenes/GameScene1 - MultiPlayer 3

Target Platform	Mac OS X
Server Build	□
Create Xcode Project	□
Development Build	☑
Autoconnect Profiler	□
Deep Profiling Support	□
Script Debugging	□
Scripts Only Build	□
Compression Method	Default

1. Upon receiving the zip file of the application (i.e., Zenith game), unzip the file using an appropriate extension or application that is built into the device
2. This will result in the application being automatically located within your files
3. Upon completion, click on the application and allow the game to run


ZIP
Build 1.6 - Multiplayer testing
Build 1.6 - Multipla...sting.zip

In the future, the game could be distributed on larger platforms such as steam, which will allow the game to be viewed by a much larger audience from around the world. This means that they will be able to download the game via the steam app, which will store the game automatically within their steam library and can be accessed through steam application.

MOLEK-SYNTEZ A$ 14.50
You already own this game[1]

Estimated total[2] A$ 14.50

1 Your account already owns some of the above items, so you can't purchase them again. You can purchase these items as a gift or adjust the contents of your cart to continue.

2 All prices include VAT where applicable