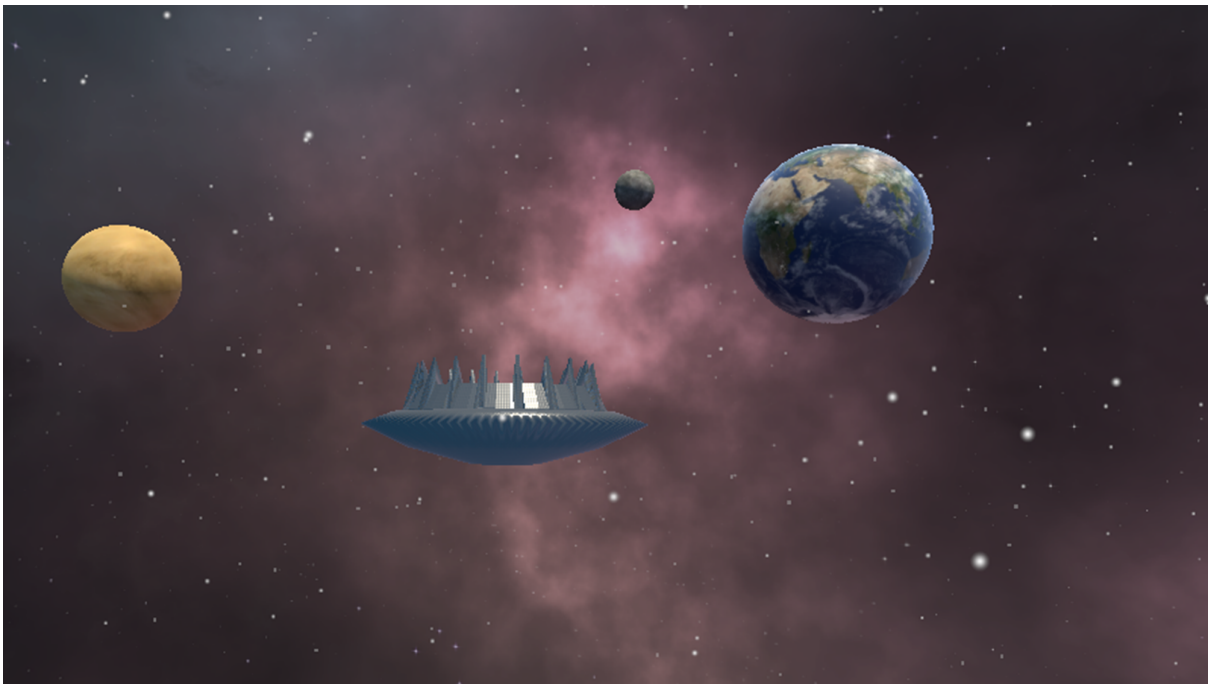


SPACEHULK_VR USER MANUAL

Group 13

Sidharth Rao

Logan Armstrong

Qilin Wang

Yibo Chen

Contents

1 Overview **3**

2 Non-Technical Explanation **3**

2.1 About 3

2.2 Age Rating 3

2.3 How to play 3

2.4 How to setup the headset 4

2.5 Controls 5

2.6 Tutorial 5

3 Technical Explanation **8**

3.1 Outline 8

3.2 System Descriptions 9

3.2.1 Main Menu 9

3.2.2 Exit Game 10

3.2.3 VR locomotion 11

3.2.4 Joystick Movement 11

3.2.5 View Character's Action Points 12

3.2.6 Pick up Characters 12

3.2.7 Place Characters on Available Space 13

3.2.8 Switch Active Team 18

4 Maintenance **20**

4.1 Safety warning 20

4.1.1 Seizures 21

4.1.2 Children 21

4.1.3 For General Use 21

4.2 Adaptive Maintenance 22

4.3 Perfective Maintenance 23

4.4 Corrective/Preventive Maintenance 23

4.4.1 Corrective 23

4.4.2 Preventative Maintenance. 23

5 Future System and Function developments **24**

5.1 Access code from Github 24

5.2 Future Functional development 24

5.2.1 Dice rolling system 25

5.2.2 Pause Menu 25

5.2.3 Advanced Combat System 26

5.2.4 Alien Spawning System 28

5.2.5 Win Condition 28

5.3 Future Non-Functional development 29

5.3.1 Sound effects 29

5.3.2 Character Animations 30

1 Overview

The following document details all essential information for the user to make full use of SPACEHULK_VR, and possibly continue to extend its functionality beyond its current state. For non-technical users who only seek guidance on how to play the game, section 2 is the only necessary reading. For technical users who wish to extend the game or simply understand the source code, it is recommended to read sections 4 and 5 -referring to section 3 if you are struggling to understand a certain section of existing code. This user manual will continue to be extended should future game updates be released.

Current Version - 1.0.0

2 Non-Technical Explanation

2.1 About

Like the classic Space Hulk, this game takes place on an abandoned space ship where there are a bunch of surviving crewmates stuck aboard this interstellar vessel. There are a bunch of aliens which keep spawning in to hunt these crewmates and kill them. 4 Space Marines however dock in, with the hopes of finding and extracting all of the crewmates. These 4 space marines tasked with the perilous mission of saving these helpless crewmates must now navigate their way through a complex board filled with spawning aliens of different types. They must locate the survivors and lead them to the escape pods without dying. The goal of the Aliens is to prevent this from happening at all cost and devour each space marine. Welcome to SPACEHULK_VR where you will either be the Alien Commander or the General of the Space Marine Corps, the survivors fate... lies in your hands.

2.2 Age Rating

The age rating of this game has not been disclosed however, our team would rank the game as E for everyone under the 2021 ESRB guidelines.

2.3 How to play

SPACEHULK_VR is a 2+ person Virtual Reality Board game. Here are the rules:

1. There are 2 teams to select from: The Aliens and The Space Marines.

2. If you choose Aliens, you must win the game by defeating all the marines
3. If you choose Marines, you must win the game by rescuing the spaceship's survivors
4. The game is played by rolling the dice to determine where to move on the board and for each space you move on the board you sacrifice an action point.

   What is an action point:

5. An action point allows each individual player to engage in combat. In the beginning, each side has 4 action points. The more they move around the board, the more action points they must sacrifice. In order to be able to engage in combat, the player must have a sufficient amount of action points to do so. 4 action points spawn in every round

   What is a round?

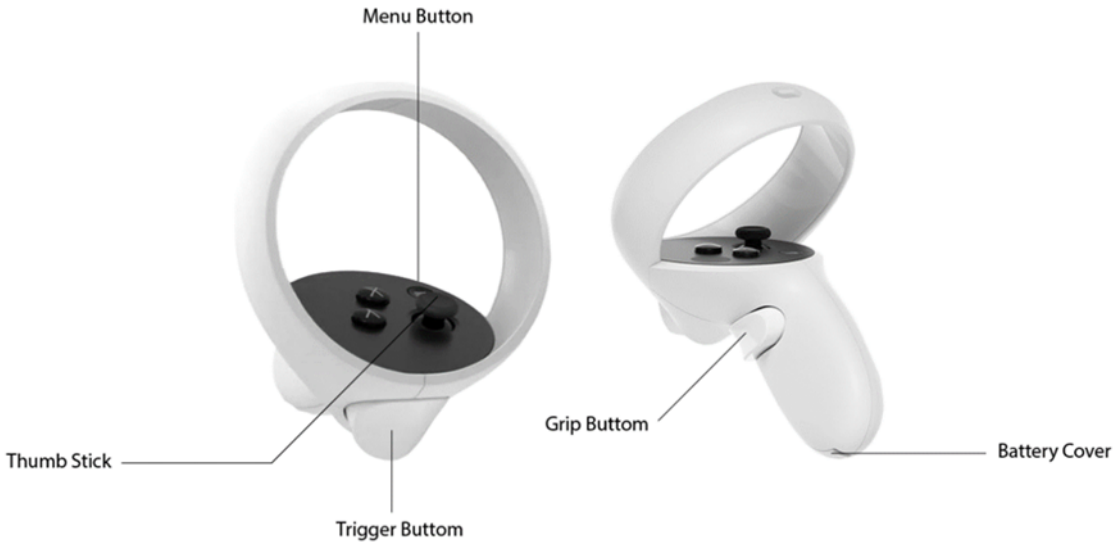
6. A round is determined by 1 roll of the dice by both ends. Once player 1 has rolled the dice, moved his piece and optionally acted, they then pass the headset to player two who must do the exact same before passing the headset back to player one.
7. Once the win condition is met, the game returns back to the start menu.
8. There is no pause menu, if you would like to take a break from the game, simply remove the headset and the game will pause automatically.

Please refer to the safety guide found at the beginning of the Maintenance section of this user manual, Section 4.1.

2.4 How to setup the headset

This has been covered more extensively in the Maintenance section of the User Manual. Please refer to https://support.oculus.com/articles/headsets-and-accessories/oculus-link/connect-link-with-quest-2/?locale=en_GB for how to set up the quest 2 on PC.

2.5 Controls



Thumb Sticks. Right thumb stick for moving front, left, right and back only when stationary.

Trigger button: to pick up an object or select an option

X, Y, A, B: switch teams

Menu button: Go into Oculus menu

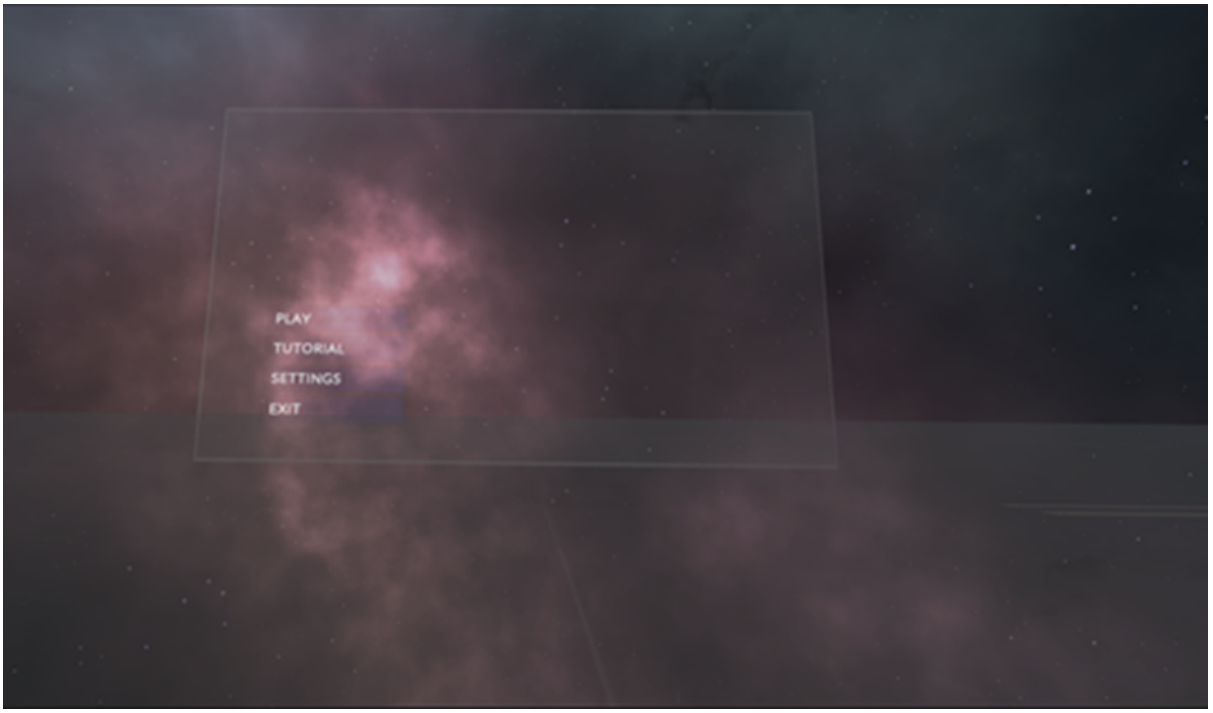
Grip Button: grabbing objects (dice, pieces),, clenching fists

That was how to use the headset. Now here is the tutorial:

Before running the game, we must first download it. This will be explained in further detail in the future development guide of the game. However, here is how to run the game if you are a non-technical user:

2.6 Tutorial

1. Download the game from the GitHub linked here: https://github.com/armypele1/SPACEHULK_VR
2. Launch the SPACEHULK_VR.exe file
3. Once in the game, put on the VR Headset and look towards the start menu



4. Here you will see a Play, Tutorials, Settings and Exit option.

5. We haven't yet implemented the Settings and Tutorial, however, the play button goes to the demo of the game which can also be considered as the tutorial.

6. The game can be played both sitting down and standing up so make sure that once you have set up the headset, you have defined a stationary or dynamic movement area.

   If you are playing on a stationary movement area:

7. Use the joysticks to manoeuvre around the map and make sure you have space to move your arms around to interact with the map and pieces.

   If you are playing in a set space:

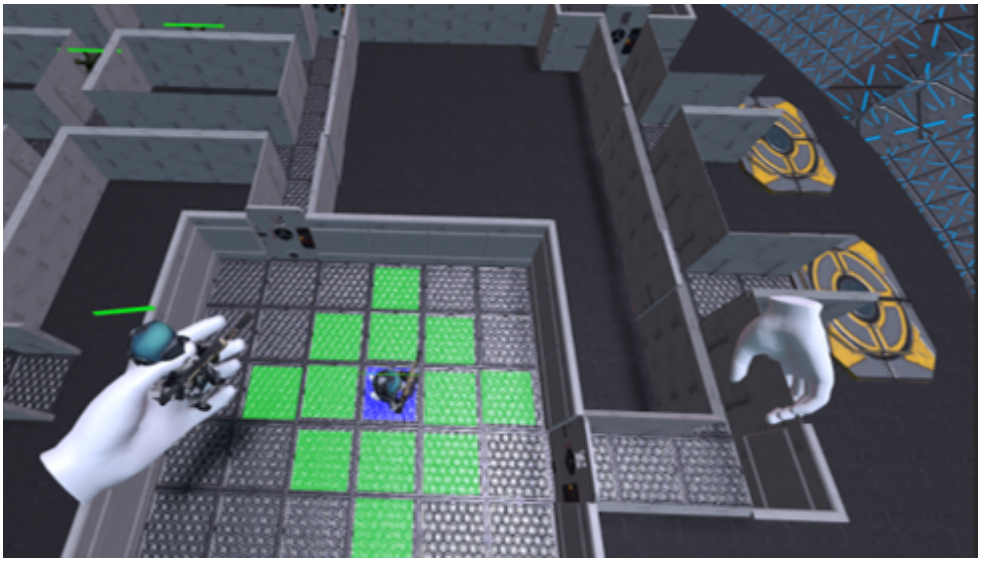
8. Set an appropriate play area within the Oculus HUD

9. Move around the map to get a better perspective, i.e. make good use of the play area.

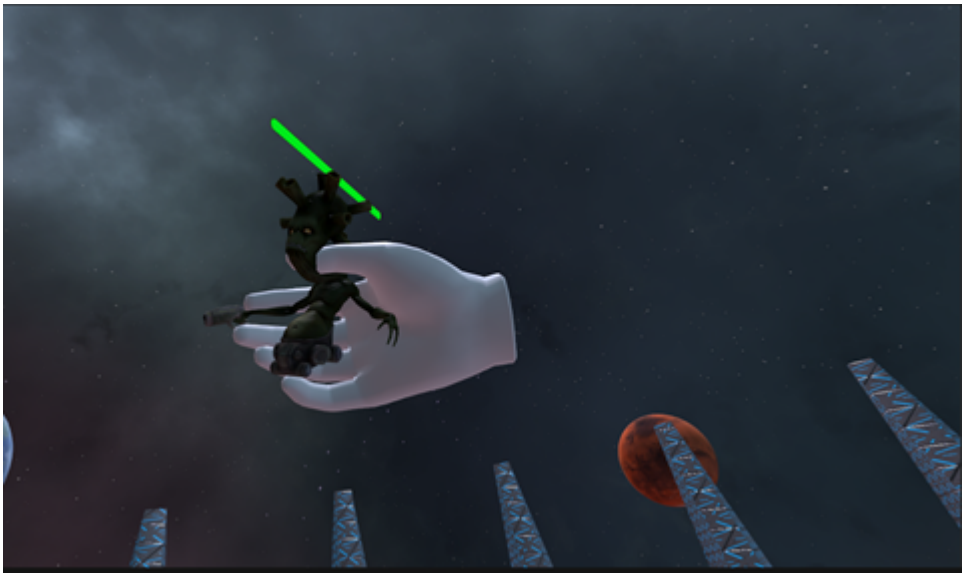
Once in the game, here are the controls for interacting with the map/dice/characters.

Movement:

1. Once the dice is rolled, a number would appear on top of the piece's head. This refers to how many moves that particular piece has got.
2. Pick up the piece using the grip button on the controller.
3. Once picked, you should see a hologram of the character you picked up which would be outlined in your hand. There should also be a radius of green squares indicating where you can drop the character next. The blue spaces indicate an allied piece and the red squares indicate an enemy piece.



4. Use the hologram as a pointer to where you want to drop the actual piece.



3 Technical Explanation

3.1 Outline

This section covers the technical implementation of key areas of SPACEHULK_VR in order to aid interested developers in extending the current functionality of the game. Each described system is referenced using the same conventions as the non-technical descriptions covered earlier, so one can directly refer between the two.

SPACEHULK_VR is built on Unity Engine version 2021.2.5f, and is supplemented with a number of external packages to facilitate the various systems of the game. The reasoning for using each of these packages is detailed below.

Package Name (A-Z)	Purpose
Animation Rigging	Used to animate the alien and marine characters in the game.
Input System	A new input system which can be used as a more extensible and customizable alternative to Unity's classic input system in UnityEngine.Input. Used to handle all inputs of the game.
Oculus XR plugin	Provide display and input support specifically for Oculus devices.
OpenXR plugin	Allows developers to easily target a wide range of AR/VR devices.
ProBuilder	Build, edit, and texture custom geometry in Unity. Used to create various sections of the map.
ProGrids	Advanced grid and object snapping for the scene view, making ProBuilder easier to use.
Shader Graph	The Shader Graph package adds a visual Shader editing tool to Unity. This is used to create the star system particle effects.
Test Framework	Test framework for running Edit mode and Play mode tests in Unity.
TextMeshPro	TextMeshPro uses Advanced Text Rendering techniques along with a set of custom shaders. It is used to display action point counts above each character in the main game and any text in the menu

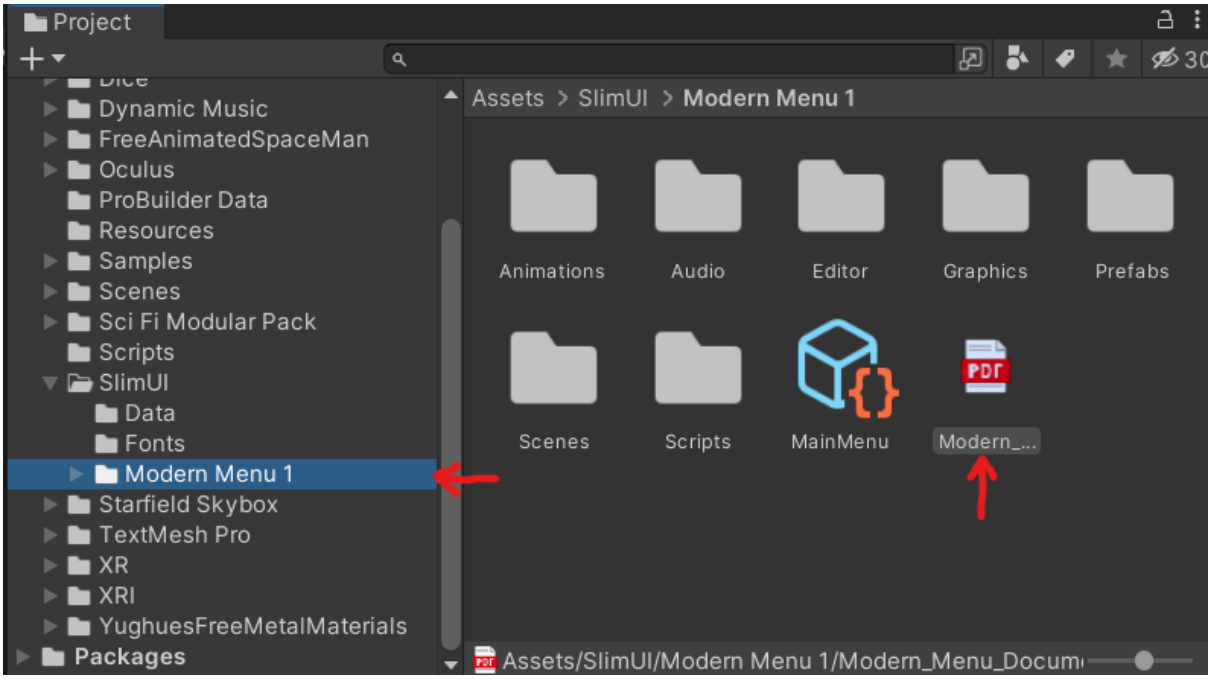
	system.
Timeline	The Timeline package can be used to create cinematic content, game-play sequences, audio sequences, and complex particle effects. It is used to create the star system particle effects.
Unity UI	Unity UI is a set of tools for developing user interfaces for games and applications. It has been used to create the menu system of the game.
Version Control	Automatically creates a gitignore file for the Github repo.
Visual Studio Code Editor	Used to edit all scripts associated with the game.
XR Interaction Toolkit	A high-level, component-based, interaction system for creating VR and AR experiences. It provides a framework that makes 3D and UI interactions available from Unity input events. The core of this system is a set of base Interactor and Interactable components, and an Interaction Manager that ties these two types of components together. It also contains helper components that you can use to extend functionality for drawing visuals and hooking in your own interaction events. This is used as the basis for all interactions in the game.
XR Plugin Management	Package that provides simple management of XR plug-ins. This helps with managing the various XR packages found within this project.

3.2 System Descriptions

All code is written in C# for this project, following standard Unity conventions.

3.2.1 Main Menu

The main menu allows the user to start a new game, edit settings associated with the main game, and exit the game entirely. It uses an asset package called SlimUI, imported from the unity asset store. Details on how to use the various scripts and models within this package can be found in the SlimUI folder.



Along with the components found in the figure above, an additional script was added to allow a new game to be started. This code can be used as a basis to extend the main menu's functionality by connecting other buttons to additional scenes.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class MenuController : MonoBehaviour
{
    public void StartBtn(){
        SceneManager.LoadScene("Project");
    }
```

3.2.2 Exit Game

The exit game button works using the following method, found in the MenuController script.

```
public class MenuController : MonoBehaviour
{
    public void ExitBtn()
    {
            Application.Quit();
```

```
    }
}
```

3.2.3 VR locomotion

VR locomotion is handled by the XR Interaction Toolkit. To learn more about how to manipulate this package, information can be found here:

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.9/manual/index.html

3.2.4 Joystick Movement

All inputs are handled by the new unity input system, which is utilised in the script "MoveBoard". Rather than directly moving the player, the playable board is moved to simulate the player moving instead. The following function is called every frame to achieve this:

```
void FixedUpdate()
    {
        mainCamera = Camera.main;
        var forward = mainCamera.transform.forward;
        var right = mainCamera.transform.right;

        horizontalMoveInput = horizontalMove.action.ReadValue<Vector2>();

        float horizontalAxis = horizontalMoveInput.x;
        float verticalAxis = horizontalMoveInput.y;

        //project forward and right vectors on the horizontal plane (y = 0)
        forward.y = 0f;
        right.y = 0f;
        forward.Normalize();
        right.Normalize();

        //this is the direction in the world space we want to move:
        var desiredMoveDirection = forward * verticalAxis + right *
horizontalAxis;

        //apply vertical movement with right stick
        verticalMoveInput = verticalMove.action.ReadValue<Vector2>();
        desiredMoveDirection.y = verticalMoveInput.y;

        //rbody.velocity = desiredMoveDirection * speed;
        transform.Translate(desiredMoveDirection * speed * Time.deltaTime);
    }
```

3.2.5 View Character's Action Points

The character's action points are displayed using the TextMeshPro package, which allows for easy text manipulation. This is displayed on a quad which is a child of the character whose action points they are associated with. This quad always points directly towards the player, and is only active when in contact with the controllers. The scripts used to achieve this functionality are below:

In pointCamera.cs:

```
// Update is called once per frame
    void Update()
    {
        if(target != null)
       {
            transform.rotation = Quaternion.LookRotation(transform.position
- target.position);

       }

    }
```

In updateText.cs:

```
// Update is called once per frame
    void Update()
    {
        actionPointCounter = GetComponent<TextMeshPro>();
        if (actionPointCounter != null) {
            currentPoints = thisCharacter.actionPoints;
            actionPointCounter.text = currentPoints.ToString();
        }

    }
```

3.2.6 Pick up Characters

All grabbable objects in the game inherit from a class called 'GamePiece'. The XR interaction toolkit package is used to recognise objects of this class and allow the user to pick up these objects when their colliders are active. To learn more about the XR interaction toolkit, documentation can be found here:

https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@0.9/manual/index.html

3.2.7 Place Characters on Available Space

A number of classes are used to handle the movement system of the game. These are the GamePiece, GridSpace, Manager and Board Grid classes.

The GamePiece class is used by all grabbable objects in the game. It is used in tandem with the XR interaction toolkit. Upon an object being 'selected' (picked up) using the XR interaction toolkit, the following method is activated:

```
public void onStartDrag()
    {
        //save previous pos before movement begins
        previousPos =
grid.transform.InverseTransformPoint(transform.position);

        //disable colliders of all other grabbables while holding this
object
        manager.toggleGrabbables(false);

        //get dummy object to pose in original pos
        dummyObject = GameObject.Instantiate(gameObject);
        dummyObject.GetComponent<Collider>().enabled = false;

dummyObject.GetComponent<Renderer>().material.SetColor("_Color",oldColor);
        dummyObject.transform.SetParent(grid.transform, true);
        //set to ghost mode
        activateGhost(gameObject.GetComponent<Renderer>().material); // Set
to ghost version of original

        List<GameObject> availableSpaces =
grid.getAvailableSpaces(previousGridSpace, actionPoints);


        foreach (var space in availableSpaces){
            var rend = space.gameObject.GetComponent<Renderer>();
            rend.enabled = true;
        }

    }
```

In this method, the grid attribute refers to the BoardGrid class associated with the game board. The manager attribute refers to the manager class instantiated with an empty game object.

The following function is called in manager.cs to disable colliders of all other grabbable objects while currently holding a character.

```
public void toggleGrabbables(bool toggle){
        var activeTeam = currentTeam == Team.marines ? "marine" : "alien";

        gamePieces = GameObject.FindGameObjectsWithTag(activeTeam);

        if (toggle == true){
            foreach (GameObject gamePiece in gamePieces){
                gamePiece.GetComponent<Collider>().enabled = true;
            }
        }
        else{
            foreach (GameObject gamePiece in gamePieces){
                gamePiece.GetComponent<Collider>().enabled = false;
            }
        }
    }
```

After this, the spaces available for the character to move are calculated using the getAvailableSpaces method in the BoardGrid script. It uses raycasting to efficiently find all squares that the character is able to move according to their remaining action points.

```
public List<GameObject> getAvailableSpaces(GameObject originSpace, int
actionPoints){
        List<GameObject> availableSpaces = new List<GameObject>();
        //first add the origin space to the available spaces list so that it
is visible
        originSpace.GetComponent<GridSpace>().cost = 0;
        availableSpaces.Add(originSpace);
        int originalActionPoints = actionPoints;
        return getAvailableSpacesHelper(availableSpaces, originSpace,
actionPoints, originalActionPoints);

    }
    private List<GameObject> getAvailableSpacesHelper(List<GameObject>
availableSpaces, GameObject originSpace, int actionPoints, int
originalActionPoints){
        int layer_mask = LayerMask.GetMask("Grid");
        if (actionPoints > 0){
            actionPoints -= 1;
            RaycastHit hit;
            if (Physics.Raycast(originSpace.transform.position,
Vector3.forward, out hit, rayRange, layer_mask)){
                var availableSpace = hit.collider.gameObject;
                if (!(availableSpaces.Contains(availableSpace)))
                {
                    availableSpaces.Add(availableSpace);
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
```

```
                }
                else if (availableSpace.GetComponent<GridSpace>().cost >
originalActionPoints - actionPoints){
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
                }
                if (actionPoints > 0 && availableSpace.tag == "free"){
                        availableSpaces =
getAvailableSpacesHelper(availableSpaces, availableSpace, actionPoints,
originalActionPoints);
                }

            }
            if (Physics.Raycast(originSpace.transform.position,
Vector3.right, out hit, rayRange, layer_mask)){
                var availableSpace = hit.collider.gameObject;
                if (!(availableSpaces.Contains(availableSpace)))
                {
                    availableSpaces.Add(availableSpace);
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
                }
                else if (availableSpace.GetComponent<GridSpace>().cost >
originalActionPoints - actionPoints){
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
                }
                if (actionPoints > 0 && availableSpace.tag == "free"){
                        availableSpaces =
getAvailableSpacesHelper(availableSpaces, availableSpace, actionPoints,
originalActionPoints);
                }
            }
            if (Physics.Raycast(originSpace.transform.position,
Vector3.back, out hit, rayRange, layer_mask)){
                var availableSpace = hit.collider.gameObject;
                if (!(availableSpaces.Contains(availableSpace)))
                {
                    availableSpaces.Add(availableSpace);
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
                }
                else if (availableSpace.GetComponent<GridSpace>().cost >
originalActionPoints - actionPoints){
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
                }
                if (actionPoints > 0 && availableSpace.tag == "free"){
                        availableSpaces =
getAvailableSpacesHelper(availableSpaces, availableSpace, actionPoints,
```

```
originalActionPoints);
                    }
            }
            if (Physics.Raycast(originSpace.transform.position,
Vector3.left, out hit, rayRange, layer_mask)){
                var availableSpace = hit.collider.gameObject;
                if (!(availableSpaces.Contains(availableSpace)))
                {
                    availableSpaces.Add(availableSpace);
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
                }
                else if (availableSpace.GetComponent<GridSpace>().cost >
originalActionPoints - actionPoints){
                    availableSpace.GetComponent<GridSpace>().cost =
originalActionPoints - actionPoints;
                }
                if (actionPoints > 0 && availableSpace.tag == "free"){
                    availableSpaces =
getAvailableSpacesHelper(availableSpaces, availableSpace, actionPoints,
originalActionPoints);
                }
            }
        }
        return availableSpaces;
    }
```

All available spaces' renderers are enabled, until the character is unselected. Once this occurs, the following method is triggered in GamePiece:

```
public virtual void onEndDrag()
    {
        //disable ghost mode

gameObject.GetComponent<Renderer>().material.SetColor("_Color",oldColor);
        Material tempMaterial =
dummyObject.GetComponent<Renderer>().material;

        //destroy dummy object
        if (dummyObject != null){
            Destroy(dummyObject);
            Destroy(tempMaterial);
        }


        //snap back to grid and update grid accordingly
        SnapToGrid();
        var aliens = GameObject.FindGameObjectsWithTag("alien");
        foreach (var alien in aliens){
```

```
            if (alien.GetComponent<GamePiece>().previousPos ==
this.previousPos){
                Destroy(alien);
            }
        }

        //re-enable grabbable colliders now that movement has ended
        manager.toggleGrabbables(true);

        //make grid invisible
        grid.setVisibility(false);
    }
```

The SnapToGrid method is called here, which sets the character back onto the grid and calls the appropriate grid spaces (using their class GridSpace) to update their colour.

The SnapToGrid method:

```
protected void SnapToGrid(){
        Transform nearestSpace = grid.getNearestSpace(transform);
        if (nearestSpace != null){

            if (nearestSpace != previousGridSpace.transform){
                actionPoints = actionPoints -
nearestSpace.GetComponent<GridSpace>().cost;
            }


            Vector3 snappedPosition =
nearestSpace.parent.parent.InverseTransformPoint(nearestSpace.position);
            Debug.Log(nearestSpace.position);
            snappedPosition.y = nearestSpace.lossyScale.y + 1;
            transform.rotation = Quaternion.identity;
            transform.localPosition = snappedPosition;

            Debug.Log(transform.position);

            grid.updateGrid(previousGridSpace.GetComponent<GridSpace>(),
nearestSpace.gameObject.GetComponent<GridSpace>());

            //update previousGridSpace ready for next movement
            previousGridSpace = nearestSpace.gameObject;
        }
        else{
            transform.localPosition = getPreviousPos();
            transform.rotation = Quaternion.identity;
        }

    }
```

The method used to update grid space colours in the GridSpace class:

```
public void UpdateColor(){
        if (gameObject.tag == "free"){

gameObject.GetComponent<Renderer>().material.SetColor("_Color",freeColor);
        }
        else if (gameObject.tag == "ally"){

gameObject.GetComponent<Renderer>().material.SetColor("_Color",allyColor);
        }
        else if (gameObject.tag == "enemy"){

gameObject.GetComponent<Renderer>().material.SetColor("_Color",enemyColor);
        }
    }
```

3.2.8 Switch Active Team

The currently active team is switched using any of the 4 face buttons thanks to the new Input System package referenced earlier. The script that utilises this is manager.cs, within the function below:

```
private void switchTeams(){

        if (currentTeam == Team.marines){
            currentTeam = Team.aliens;

            var oldTeam = GameObject.FindGameObjectsWithTag("marine");

            foreach (GameObject oldTeamMember in oldTeam){
                oldTeamMember.GetComponent<Collider>().enabled = false;
                oldTeamMember.GetComponent<GamePiece>().actionPoints = 4;

oldTeamMember.GetComponent<GamePiece>().previousGridSpace.tag = "enemy";

oldTeamMember.GetComponent<GamePiece>().previousGridSpace.GetComponent<GridS
pace>().UpdateColor();

            }

            var newTeam = GameObject.FindGameObjectsWithTag("alien");

            foreach (GameObject newTeamMember in newTeam){
```

```
newTeamMember.GetComponent<GamePiece>().previousGridSpace.tag = "ally";

newTeamMember.GetComponent<GamePiece>().previousGridSpace.GetComponent<GridS
pace>().UpdateColor();
            }
        }
        else{
            currentTeam = Team.marines;

            var oldTeam = GameObject.FindGameObjectsWithTag("alien");

            foreach (GameObject oldTeamMember in oldTeam){
                oldTeamMember.GetComponent<Collider>().enabled = false;
                oldTeamMember.GetComponent<GamePiece>().actionPoints = 6;

oldTeamMember.GetComponent<GamePiece>().previousGridSpace.tag = "enemy";

oldTeamMember.GetComponent<GamePiece>().previousGridSpace.GetComponent<GridS
pace>().UpdateColor();
            }

            var newTeam = GameObject.FindGameObjectsWithTag("marine");

            foreach (GameObject newTeamMember in newTeam){

newTeamMember.GetComponent<GamePiece>().previousGridSpace.tag = "ally";

newTeamMember.GetComponent<GamePiece>().previousGridSpace.GetComponent<GridS
pace>().UpdateColor();
            }
        }

        toggleGrabbables(true);
    }
```

4 Maintenance

4.1 Safety warning

WARNING: HEALTH & SAFETY WARNINGS:

PLEASE ENSURE THAT ALL USERS OF THE Oculus Quest 2 HEADSET CAREFULLY READ THE WARNINGS BELOW BEFORE USING THE HEADSET.

PLEASE CHECK THE www.oculus.com/warnings FOR THE LATEST WARNINGS AND SAFETY IN USING AN OCULUS VR HEADSET.

- To limit the risk of discomfort, make sure the side and top straps are properly adjusted, and that the facial interface is in a comfortable position and that you can see a single, clear image; this will help with the headset's weight balancing and distribution. To avoid any unintended modifications to any adjustments, double-check the settings before resuming use after a break.

- Virtual reality is a powerfully immersive experience. Your body can react as if the content is real if it is frightening, violent, or anxiety causing. If you have a history of discomfort or physical symptoms when confronted with certain scenarios, be cautious about the information you choose. Some content on Oculus has comfort ratings, which you should look over before using it. (For more details on comfort ratings and how they can assist in providing a comfortable experience, go to https://support.oculus.com/help/oculus/918058048293446/). If you are new to virtual reality, start with content rated Comfortable, before trying Moderate, Intense or Unrated content.

- Having a good feeling of motion and balance is essential for a pleasant virtual reality experience. When you're tired, need to sleep, under the influence of alcohol or drugs, hungover, have digestive issues, under emotional stress or worry, or suffering from a cold, flu, headaches, migraines, or earaches, don't wear the headset because it can make you more susceptible to negative symptoms.

- Consult with your physician before using the headset if you are pregnant, elderly, have pre-existing binocular vision abnormalities or psychiatric disorders, or suffer from a heart condition or other serious medical condition.

4.1.1 Seizures

Even if they have never had a seizure or blackout before or have no history of seizures or epilepsy, some people (about 1 in 4000) may experience severe dizziness, seizures, eye or muscle twitching, or blackouts triggered by light flashes or patterns while watching TV, playing video games, or experiencing virtual reality. Seizures like these are more common in youngsters and teenagers. Anyone experiencing any of these symptoms should stop using the headset and seek medical attention. Before using the headset, visit a doctor if you've ever had a seizure, lost consciousness, or any other symptom associated with an epileptic condition.

4.1.2 Children

Because the headset is not intended for children, wrong fitting might cause discomfort or harmful health effects, and younger children are in a key stage in their visual development, this product is not a toy and should not be used by children under the age of 13. Adults should ensure that children (ages 13 and up) use the headset in compliance with these health and safety warnings, including ensuring that the headset is used as indicated in the Before Using the Headset and Safe Environment sections. Adults should keep an eye on children (ages 13 and up) who are using or have used the headset for any of the symptoms listed in these health and safety warnings (including those listed under the Discomfort and Repetitive Stress Injury sections), and limit the amount of time they use it and make sure they take breaks. Long-term use is discouraged since it can impair hand-eye coordination, balance, and multitasking abilities. Adults should keep a close eye on children during and after they use the headset to see if their abilities have deteriorated.

4.1.3 For General Use

- Allow your body to acclimate to the headset by using it for only a few minutes at a time initially, and gradually increase the amount of time you spend using it as

you become more accustomed to virtual reality. When you initially enter virtual reality, look around to assist you acclimate to any slight variations between your real-world motions and the virtual reality experience that results.

- When in a moving vehicle, such as a car, bus, or train, do not wear the headset because it may enhance your sensitivity to negative sensations.
- Even if you don't think you need it, take a 10 to 15-minute break every 30 minutes. Because everyone is different, if you're uncomfortable, take more frequent and longer pauses. It's up to you to figure out what works best for you.
- When utilising headphones, listening to music at extreme volumes might cause irreversible hearing impairment. Background noise and long-term exposure to high volume levels can make sounds appear quieter than they are. Because of the immersive nature of the virtual reality experience, do not wear the headset with the sound turned up loudly in order to maintain awareness of your surroundings and avoid hearing damage.
- If you suffer any of the following symptoms, stop using the headset or your laptop right away: Dizziness; disorientation; impaired balance; impaired hand-eye coordination; excessive sweating; increased salivation; nausea; light-headedness; discomfort or pain in the head or eyes; drowsiness; fatigue; or any symptoms similar to motion sickness.

4.2 Adaptive Maintenance

We are making this game an open source project and therefore this means that if some other developers are editing the GitHub, there are going to be certain implementations that allow for change to occur. The best way to solve this would be to track the version control on github and identify which user made changes. At the top of the GitHub, in future implementations, there should be separate sub folders for mods or other future implementations which can be run independently or with the original game code.

IBM's requirements are likely to evolve in the future, necessitating modifications. The system will be required, and subsequent implementations will be necessary, such as if IBM wants to expand the scope of the project by adding new rooms or animations.

4.3 Perfective Maintenance

As technology advances and requirements change, further adjustments to the system may be required to increase performance and simplicity of maintenance. This may entail migrating the project to a more recent version of Unity Engine.

4.4 Corrective/Preventive Maintenance

4.4.1 Corrective

While the project is now functioning as intended, if other maintenance is performed, new problems or issues may occur. As a result, corrective or adaptive maintenance may be necessary to resolve these difficulties.

4.4.2 Preventative Maintenance.

Although the program is currently working as intended, implementing new features always comes with the risk of introducing bugs to the system. To minimise this, it is recommended that section 3 is comprehensively understood before attempting to add more features. Furthermore, introducing new packages to Unity can also result in compatibility issues, so this should always be checked before adding a package to the project. In the case that bugs do arise, corrective or adaptive maintenance may be required to fix these issues.

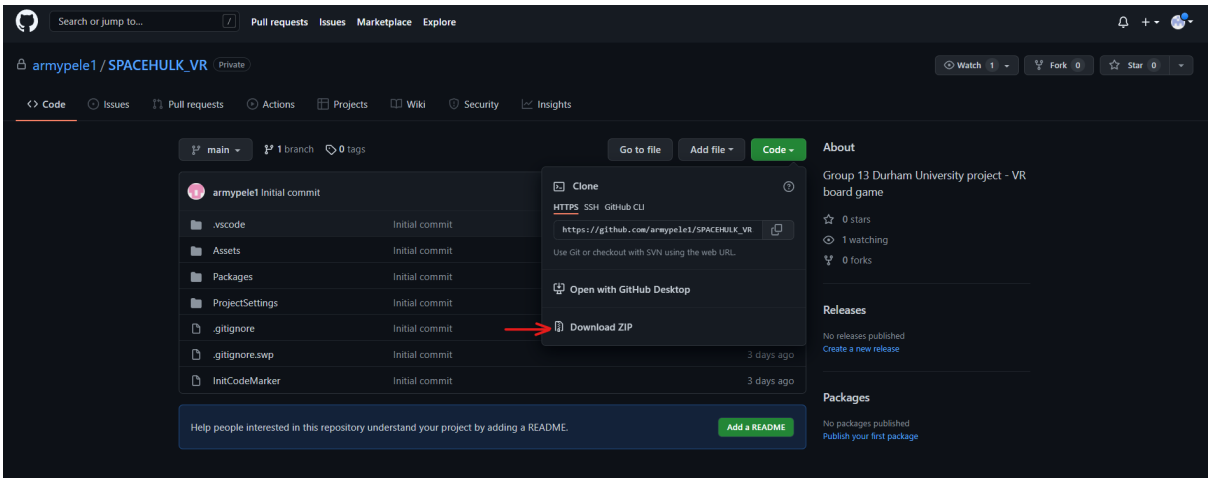
5 Future System and Function developments

This section helps developers to update the game source code to add more functionality in the future. This section shows you how to download the game's source code from Github and open it in Unity. We will also suggest some functional / non-functional requirements and modifications that may need to be added in future updates.

5.1 Access code from Github

Developers can access the source site via the Github link below, which requires a Github user account.

Here is the Link: https://github.com/armypele1/SPACEHULK_VR



Developed the ability to download a ZIP file of the game to the computer desktop by using the Code button shown in the picture. You can then unzip the ZIP file using any decompression software. The developer can then open the newly unzipped file from Unity.

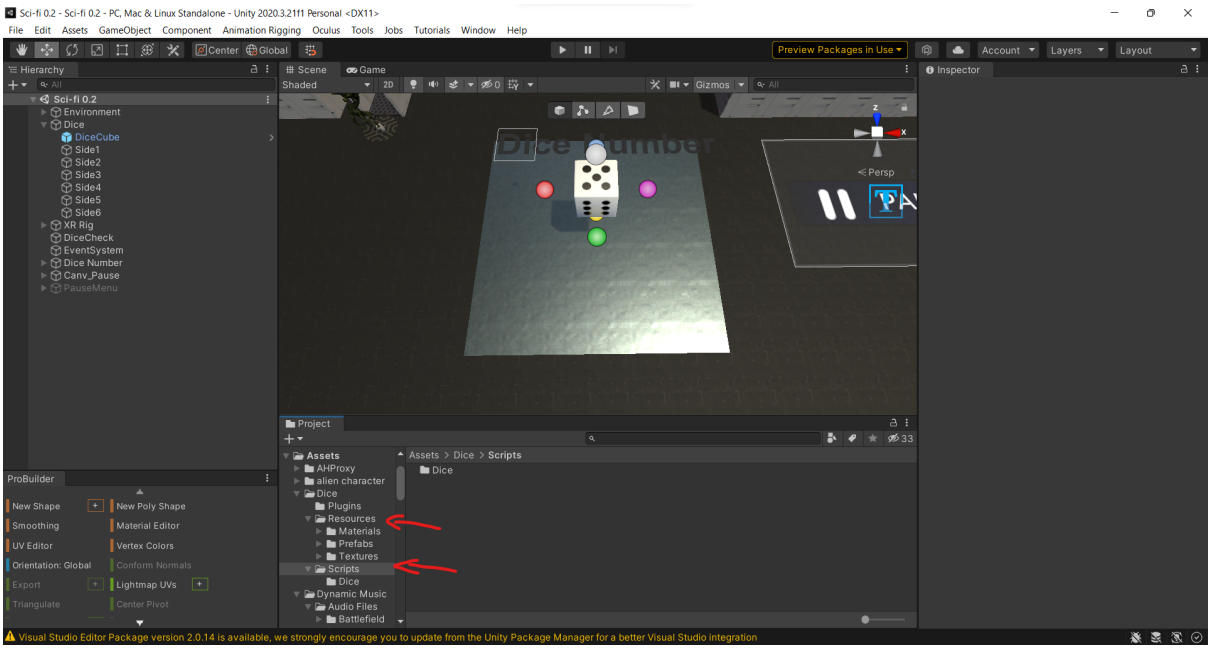
For developers, you can share your code to Github via Unity for future updates and code changes. We strongly recommend that subsequent updates to the game be synchronised to Github to make it easier for the team to experience and modify.

5.2 Future Functional development

Below we will give some of our suggestions for functional requirements in future game updates.

5.2.1 Dice rolling system

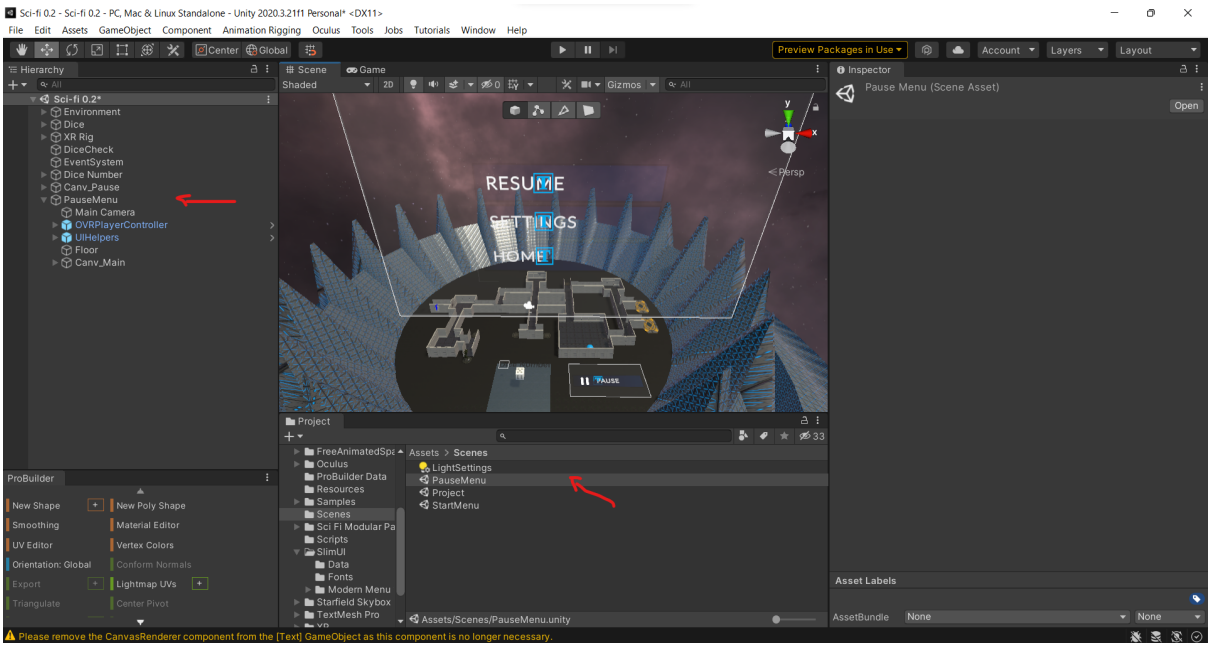
In the current version, the dice rolling system is not implemented into the main executable game. So, we're going to give you a rough idea of what this part is. In the following diagram, we will show our early design for this section. There will be a rough idea of the die's design and the layout of some scenes, as well as the location of material files and script files.



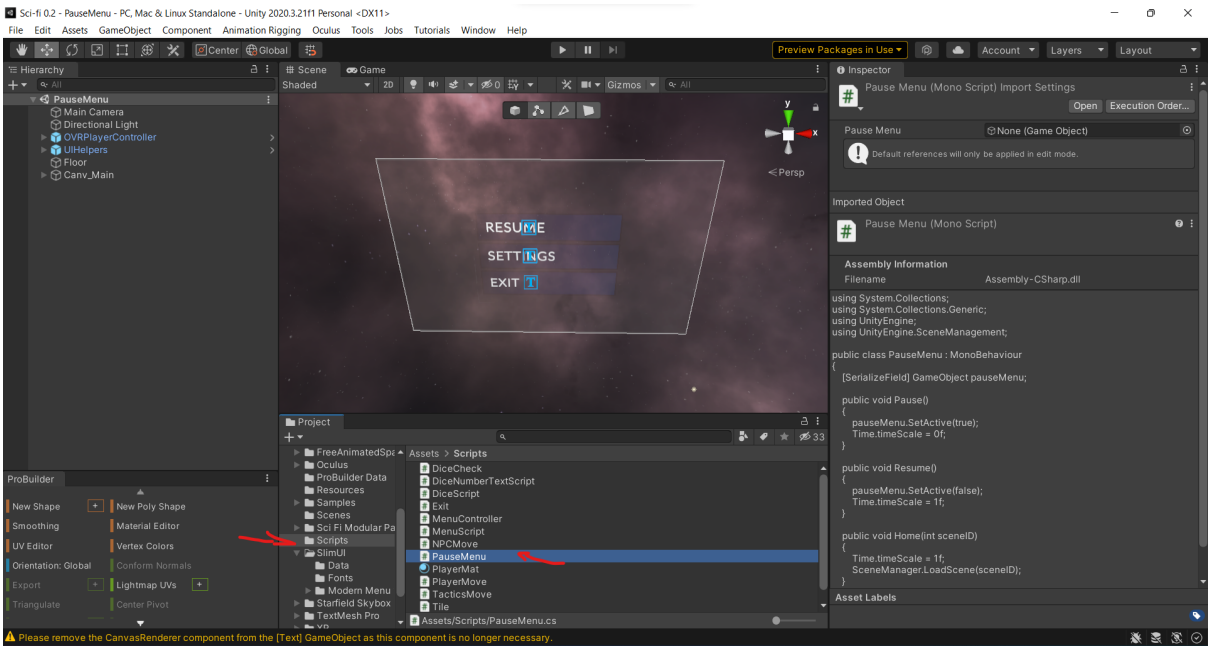
This is what happens when you open a game downloaded from Github from Unity. This is Dice designed for the Dice Rolling System in the early stage of the game. As shown here, there is a six-sided DICE, and there is an area under DICE to check the dice number. In the Dice folder under the Project window, you can see the Resources and Scripts folders, and developers can add Dice material changes or run script changes in these two folders in future updates.

5.2.2 Pause Menu

The game has been designed with a canvas of a pause menu, and in future updates it is hoped that more Pause Menu options will be developed. The following shows what Pause Menu looks like in the game and where its script files are located.



In the Hierarchy window on the left side of Unity, you can click to view the Pause Menu. As shown in the Scene window, this is what pausemenu looked like in early game design. In the Scenes folder in the Project window below, developers can click on the Pause Menu scene and then go to that scene.



The image above shows what PauseMenu looks like after entering the PauseMenu scenario, where developers can individually modify PauseMenu. Developers can modify and update the PauseMenu script file in scripts in the project window below.

5.2.3 Advanced Combat System

In the current version of the project, we implemented health bars to identify the health status of the characters on the board. However, the combat system was not added to

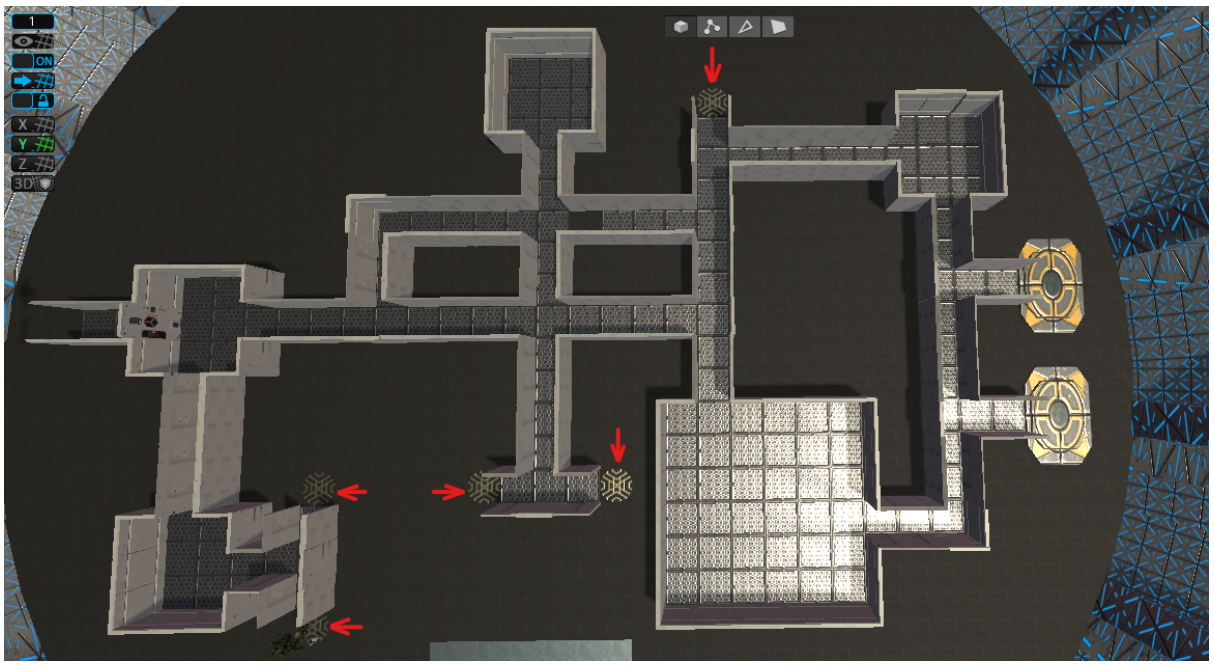
make the health bars work. We designed the basic combat system which allows the health turns to 0 if the character was touched by a character of the enemy team, then the character disappears from the board. The code is shown in the figure below. Future developments can combine the dice rolling functionality with the combat system, for example, if an enemy character is in the attack range of an alley character, the player can roll the dice to attack if the number of the dice is 5 or 6. This can help to add more uncertainty to the game and make the game more entertaining.

```
public void onStartAttack(){
        isAttackActive = true;
        lineObject = new GameObject("lineObject");
        LineRenderer attackLine = lineObject.AddComponent<LineRenderer>();
        attackLine.material = Resources.Load("Attack", typeof(Material)) as
Material;
        attackLine.widthMultiplier = 0.01f;
        StartCoroutine(SetLinePos());
    }

    private IEnumerator SetLinePos(){
        int layer_mask = LayerMask.GetMask("Map");
        while (isAttackActive){
            RaycastHit hit;
            if (Physics.Raycast(new
Vector3(gameObject.transform.position.x,gameObject.transform.position.y,game
Object.transform.position.z), (previousGridSpace.transform.position -
gameObject.transform.position).normalized, out hit, layer_mask)){
                //var wall = hit.collider.gameObject;
                lineObject.SetActive(false);
            }
            else{
                lineObject.SetActive(true);
                LineRenderer attackLine =
lineObject.GetComponent<LineRenderer>();
                attackLine.SetPosition(0, new
Vector3(gameObject.transform.position.x,gameObject.transform.position.y,game
Object.transform.position.z));
                attackLine.SetPosition(1, new
Vector3(previousGridSpace.transform.position.x,previousGridSpace.transform.p
osition.y,previousGridSpace.transform.position.z));
            }

            yield return null;
        }
    }

    public void onEndAttack(){
        isAttackActive = false;
        if (lineObject != null){
```

```
            Destroy(lineObject);
        }
    }
```

5.2.4 Alien Spawning System

The game has the positions where the alien can spawn as shown in the image below. However in the current version of the game, only a certain number of aliens are initialised when the game starts. Future developments can be made to add functionalities to the models to enable alien spawn every 5 minutes or after 5 rounds of the game. Developers can also add spawn conditions, for example, if there are more than 3 aliens on the board, the spawn system will delay its trigger time to ensure the fairness of the game.



5.2.5 Win Condition

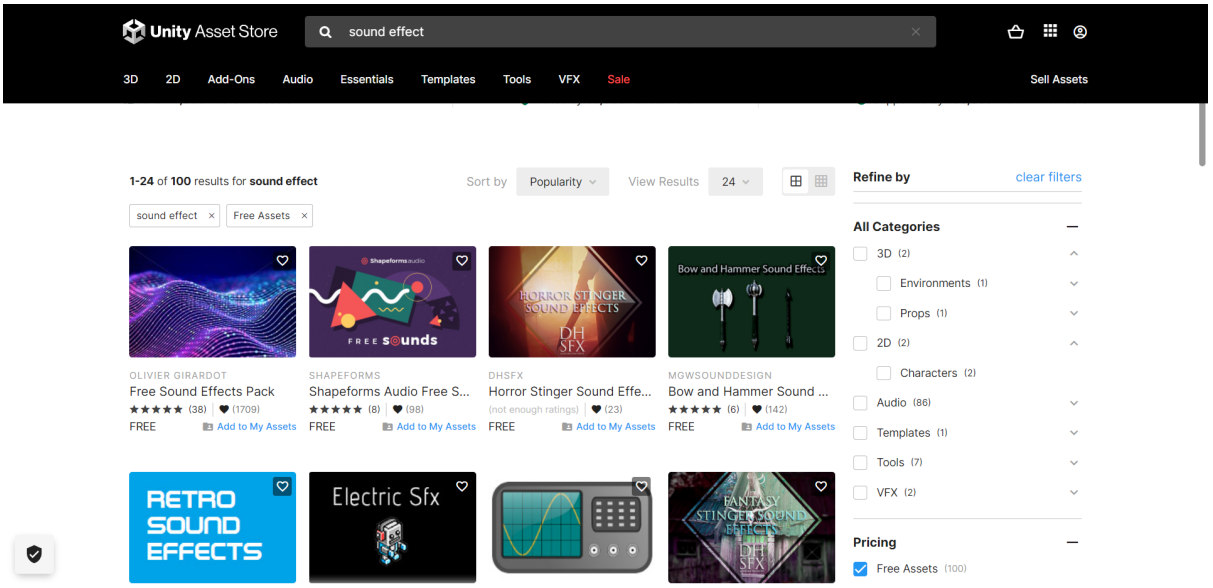
Win condition is one of the most important parts of functionalities. When space marines team successfully rescues the hostage and reaches the escape point, or the aliens team kills all the space marines on the board, the win condition is triggered and a canvas shows up. The canvas for winning was designed as the following image, where the player is informed of winning and a button allows the user to go back to the start menu.

5.3 Future Non-Functional development

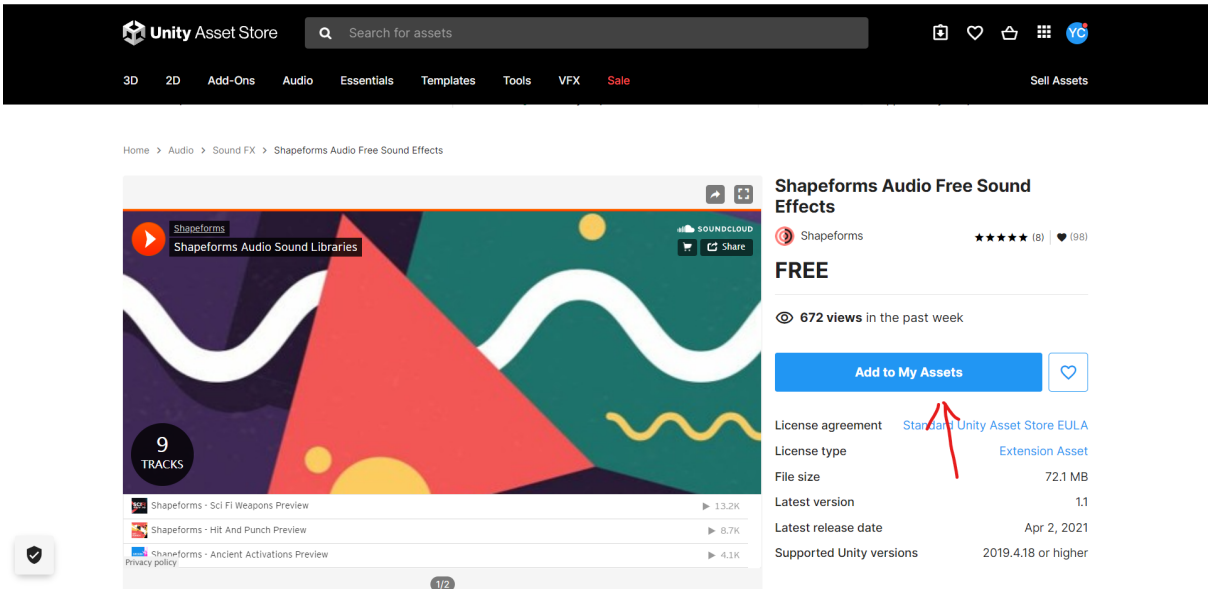
This section shows the non-functional Requirement that we will implement in future updates. These updates are also in Unity.

5.3.1 Sound effects

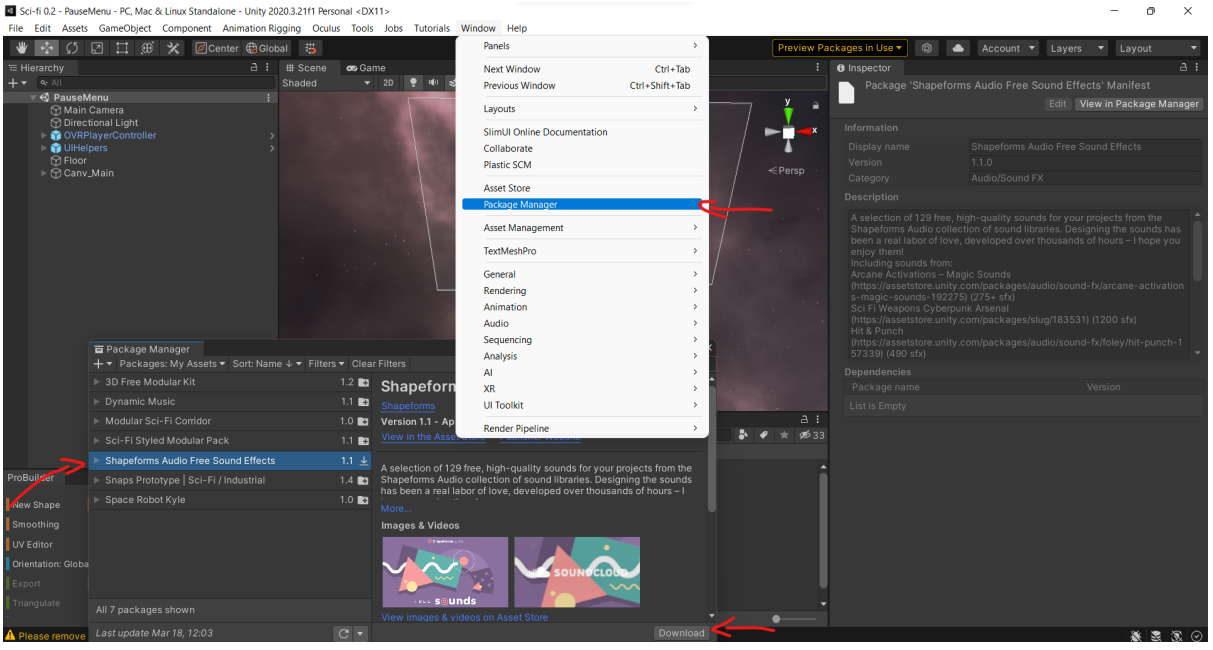
In the current version of the product, there is only background music in the game. For adding sound effects to future versions of the game we recommend using the Unity Asset Store to introduce sound effects with a sound pack and some assets. Using the Unity Asset Store makes the cost of the game easier and helps us update and design the game.



The above is the result of a search for Sound Effects in the Unity Asset Store, which has a wide selection of sound Effects for free and for free.



Example: We can randomly select a sound pack and click the button at the arrow to add our own Unity Assets.



Developers can click On Package Manager in The window at the top of the unity toolbar and click on the Sound Effect package to download and import.

5.3.2 Character Animations

The current version only has static characters in the game, but we aim to add animations to the characters to make the game more attractive. The space marine model can walk periodically but can not stop walking depending on events in the game.

The components of the space marine are shown in the figure below. Future developments can be made by adjusting the animator component or using bone renderer to create new animations.