VRun: An Endless Runner Third-Person Virtual Reality Game

Eric Dittus* Lanique L. Peterson†

Computer Science Department
Hunter College, City University of New York
https://github.com/La-Nique/VRun
https://youtu.be/rQ-VMlocOac

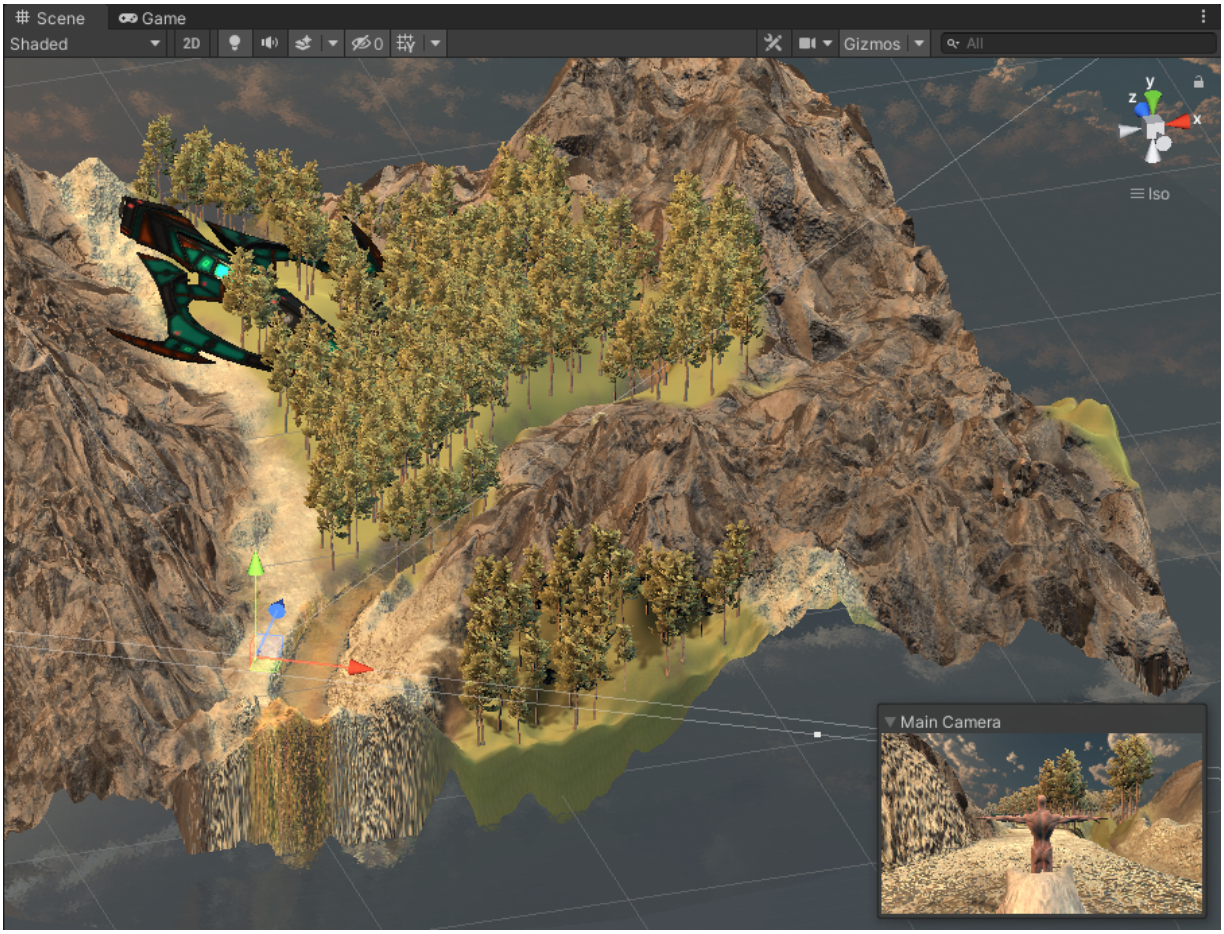


Figure 1: Birds-eye and Main Camera view of Scene One of VRun

ABSTRACT

A curious alien landed their ship on earth. While wandering through the forest, they encountered a hungry wolf. The hungry wolf starts chasing the alien, and the alien then flees back to their ship, dodging obstacles, to escape from the wolf! Once the alien makes it back onto its ship, the alien takes off through space to land back on his home planet. Once they land on their home planet, they must run through dangerous terrain to make it back home. Our game is called VRun [19]. VRun is a third-person endless runner game that uses natural head movement to direct and steer the alien to their destination. Natural head movement plays a significant role in the immersion levels of virtual reality systems. Virtual reality endless runner systems require a lot of head movement to enable users to feel immersed in the virtual environment. This research paper outlines the design and implementation of the system. An evaluation of the system demonstrates that this system works.

*e-mail: eric.dittus@macaulay.cuny.edu
†e-mail:lanique.peterson02@myhunter.cuny.edu

1 Introduction

2 Related Work

3 Methodology

3.1 Unity Parameters and Google VR Cardboard Integration

3.1.1 Unity Version

Unity Version 2020.3.36 was used at the direction of Professor Wole Oyekoya.

3.1.2 Google VR Cardboard[1]

To ensure properly established Google VR Cardboard integration with a Unity3D project, it is first important to use a Unity 2020 Version. Then, build a Unity3D project with iOS and MacOS support. Add the Google VR Cardboard Package to your project as a url from the googleVr/cardboard GitHub page [12]. Ensure your build settings are set for iOS, and then add an XRRig to the scene. Lastly, build on XCode.

3.1.3 XCode Deployment[2]

In the Unity3D VRun project, go to File -¿ Build Settings -¿ iOS -¿ "Switch Platform", and then select "Build and Run" when the project is finished processing. You will then be prompted to choose a location for the XCode build. After selecting, the project will be built onto XCode and then open and begin to process.

While processing on XCode, it may be required to go to "Signing and Capabilities", click "Automatically Manage Signing", and then select "Team" as your registered apple id account. After build completion, connect your iOS device to the Mac laptop via USB C charging cable, and unlock the phone, which will install the app and start if given permission.

3.2 Input Device

3.2.1 VR Integrated Input

An Iphone SE using iOS 15.5 mounted in a Google VR Cardboard was used as input. To turn left and right, the device's accelerometer x-axis input must be between the float value of -0.2 and -1.0, and 0.2 and 1, respectively. For jumping, the accelerometer y-axis input must be between -0.9 and 0.5. VRun-Keyboard Accessible, alternatively, can receive "a", "d", and "space bar" button commands for left, right, and jump, respectively.

3.2.2 Recommended Specs

An iOS device with a minimum iOS Version of 11.0 and a high performing CPU and GPU is recommended to play VRun with googleVR.

3.2.3 Keyboard Integrated Input For Testing

VRun also contains the capability to run using keyboard input in the Unity3D editor with minimal code manipulation.

3.3 Map Building[3]

3.3.1 Terrain

The terrain painting tool (See Figure 2) was used to change the terrain, and imported texture packages used for realism [16] [22].

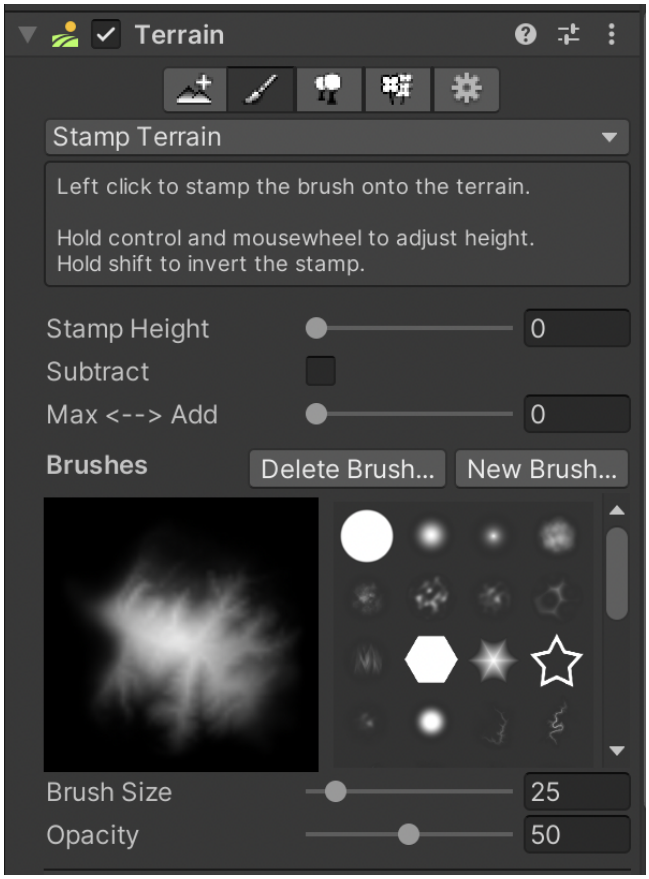


Figure 2: Terrain painting tool

3.3.2 The SkyBox[4]

3.4 Obtaining Character Model and Movement Animations[5]

The character model and animations for "Maynard", such as idle, walking, running, walk/run right/left, and jump, were obtained from Mixamo.com [17].

3.5 Character Movement[6]

3.5.1 Animator, Animation State Controller, and 2d Blend Tree

The Unity Animator (See Figure 4) is a tool to design the character animation logic within a Unity3D project. To control character movement an animator must be attached as a component of a character. A C script known as the Animation State Controller should be attached to the Animator to assist animation transitions.

Parameters can be added to the Animator that defines a transition to a new state. For example, the boolean variable isJump was used to define the transition to and from the jump animation. The Animation State Controller is the where the manipulation of

[1]The methodology for Google VR integration with VRun and section 3.1.2 is from The Wonderland of Virtual Reality on Youtube [18]

[2]The methodology for deployment of VRun on XCode and in section 3.1.3 is from The Wonderland of Virtual Reality on Youtube [18]

[3]Map building for VRun was learned from Brackeys on Youtube [8] and Unity Terrain Editor Tutorial [23]

[4]The SkyBox methodology used in VRun was learned from XyloBetaGames on Youtube [25]

[5]The methodology and implementation included in section 3.6 was at the direction of iHeartGameDev on Youtube [14]

[6]The methodology and implementation of the 2D Blend Tree and non-VR integrated movement capabilities from iHeartGameDev on Youtube was the basis for section 3.7's functionality. [14]

Figure 3: Alien character ”Maynard”.

these variables take place.

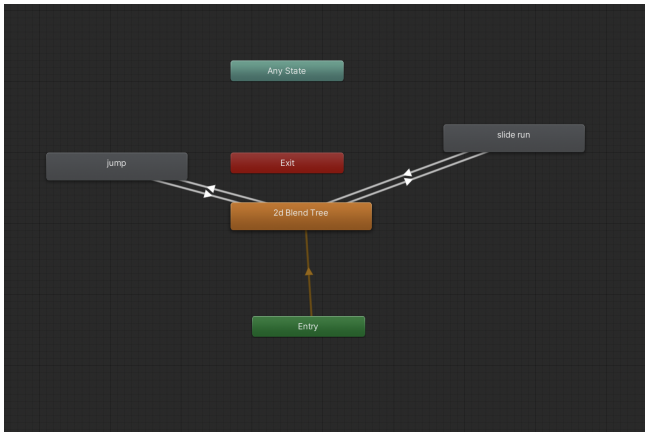


Figure 4: Animator attached to the Alien Character.

Two float variable parameters in the Animator attached to ”Maynard” played a major role in the implementation of character movement in VRun: BlendZ and BlendX. These floats were implemented in the 2D Blend Tree set as the default state of the Animator layer.

The 2-Dimensional Blend Tree with blend type 2D Freeform Directional (See Figure 5) controls transitions between character animations by the use of the parameter floats BlendX and BlendZ.

The 2-Dimensional Blend Tree graph (See Figure 6) contains the desired character motions to be ”blended” together. The top contains a coordinate grid, and the bottom lists each motions’ coordinate mapping. The forward motions are positioned at (0, Pos Y), where running has the highest Pos Y value. The turning motions are given positions at (Pos X, 0), where (-Pos X, 0), and (Pos X, 0) represent a left or right turn, respectively. The BlendZ (Pos Y) and BlendX (Pos X) float parameters act as the coordinates of the red dot defaulted at (0,0). As the values of BlendZ and BlendX change and shift from one motion coordinate to another, it causes the character to emulate the blended animations, resulting in seamless movement transitions.

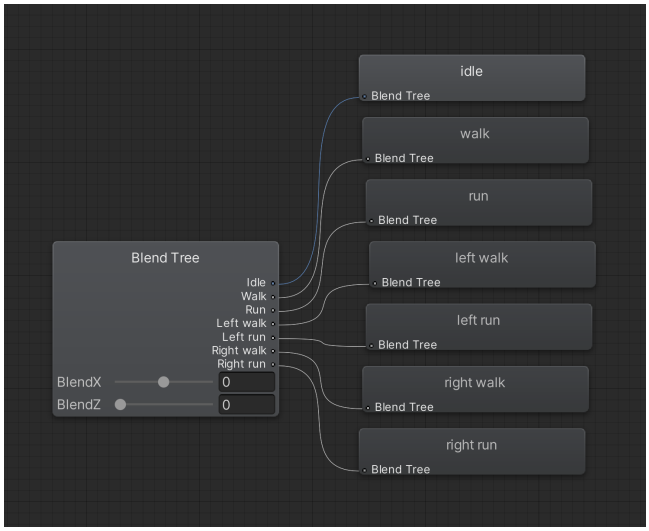


Figure 5: 2-Dimensional Blend Tree in Animator of Figure 3

3.5.2 VR Integration

Every modern smart phone is equipped with an accelerometer, which accesses the physical position information of the phone relating to its x, y, and z tilt axis.

The accelerometer information can be accessed using Input.acceleration.axis (See Figure 7), which returns a float between -1 and 1 representing that axis’ tilt. Here, the float variable ”turning” obtains the input’s x axis data, and the ”jumpOrSlide” variable obtains it’s y axis data.

The algorithm in Figure 8 splits the VRun field of view into 3 segments. When accelerometer x-axis value is between -1 and -0.2, and 0.2 and 1, the blendX variable representing the blendX parameter in the Animator decreases or increases, causing the character to turn left or right, respectively, whereas the valued between -0.2 and 0.2 negates turning. In that case, the character resumes running in a straight line.

Jumping is done using the same methodology, except it receives the input’s accelerometer y-axis information, and only jumps when the ”jumpOrSlide” has a value of between -0.9 and 0.5. Which in terms of VR represents the rotation from the top of the forehead, up and around the back of the user’s head, and to their chin. Therefore, the character can jump when the user looks both up and down, but does not react when the user looks straight ahead.

3.6 3rd Person Camera Tracking and Wolf Following Method

Figure 9 exists in the camera script of the XRRig that trails the Maynard. The public ”Transform” variable charPosition references the character. Distance is the following distance of the camera, offset updates its height to remain at a fixed location above the character despite character descension/ascension, and delay can also be used to change the distance between the character and the camera. transform.LookAt() from the Unity3D scripting API [6] updates the rotation of an object to ensure it continues to look at the target game object, which allowed the XRRig to remain fixed on the character despite head movement from the user.

The Wolf trailing algorithm (Shown in Figure 11), is similar to the camera tracking algorithm, with slight alterations. Distance re-

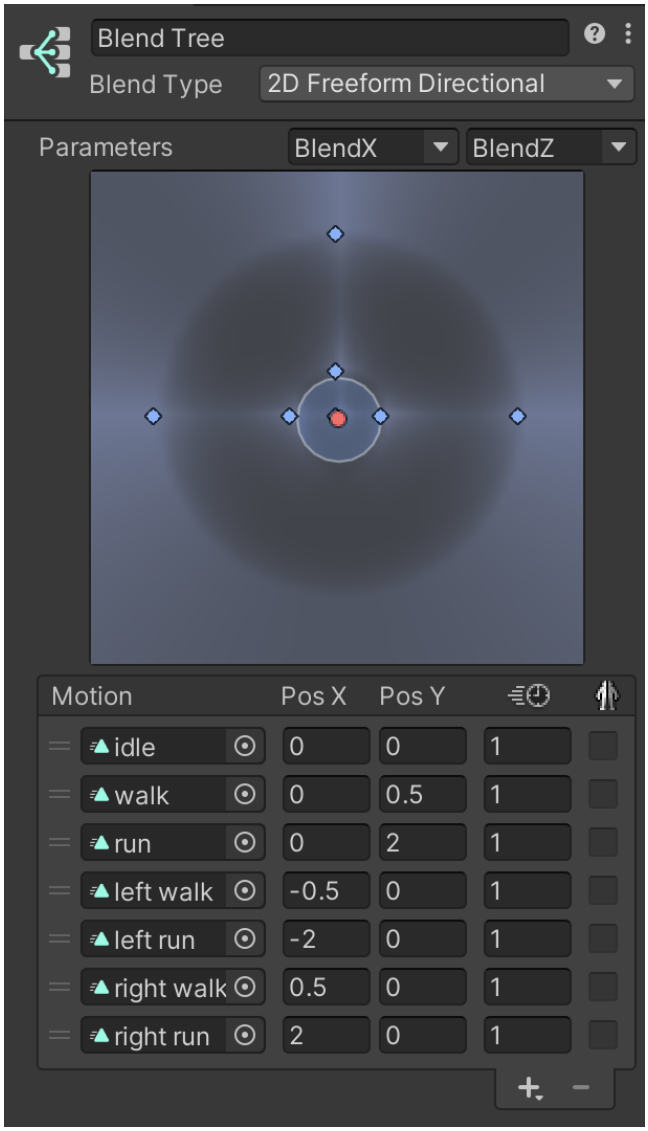


Figure 6: 2-Dimensional Blend Tree Graph

```
float turning = Input.acceleration.x;
float jumpOrSlide = Input.acceleration.y;
```

Figure 7: Accessing input accelerometer values [2]

maining constant can cause the Wolf to begin ahead of Maynard, and therefore must start at Distance = 8decreases until -6, allowing a more gradual transition to a fixed location trailing behind Maynard. However, when the character turns, how does one ensure the wolf turns as well, seemingly in pursuit? VRun adapts the rotation methodology presented by James Brady [9] to rotate the wolf game object around its y-axis based on the left and right turns of the character, using the ”blendX” variable in Maynard’s animation state controller, which references the parameter in its animator, to calculate the rotation magnitude. The transform.LookAt method, however, replaces the need for rotational change.

```
// turn left
if((turning <-0.2f && turning >-1f) && (blendX > -2.0f) ){
    if(blendX > 0){
        blendX = 0;
    }
    blendX -= Time.deltaTime * 0.5f;
}
// center stop turning left
if((turning >-0.2f && turning <0f) && blendX<0.0f){
    blendX = 0f;
    //blendX += Time.deltaTime * 1f;
}
//right
if((turning > 0.2f && turning < 1f) && (blendX < 2f)){
    if(blendX < 0){
        blendX = 0;
    }
    blendX += Time.deltaTime * 0.5f;
}
//center stop turning right
if((turning < 0.2f && turning > 0f) && (blendX > 0f)){
    blendX = 0f;
}
```

Figure 8: Accelerometer integrated turning algorithm within the Animation State Controller of the character. blendX methodology from iHeartGameDev on Youtube [14]

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

public class CharacterTracking : MonoBehaviour
{
    public Transform charPosition;
    public float distance = 1f;
    public float offset = 3f;
    public float delay = 0.02f;
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        Vector3 follow = charPosition.position - charPosition.forward * distance;
        follow.y += offset;
        transform.position +=(follow - transform.position) * delay;
        transform.LookAt(charPosition.transform);
    }
}
```

Figure 9: The 3rd person camera tracking and wolf following algorithm in VRun, credited to James Brady on Youtube [9]

3.7 Camera Switching and the Training Room

3.7.1 Camera Switching

The code seen in Figure 12 takes advantage of the GameObject.SetActive() function described in Unity Documentation [1]. The desired camera GameObjects should be dragged into the script component where the code from Figure 12 is located. The cameras are activated and deactivated per developer intent, and allow a wait time using the WaitForSeconds() command according to methodology desribed in Unity Documentation [7].

3.7.2 The Training Room

A thread started by derHugo on stack overflow was was follow to learn how to make a transparent object [10]. A sperical Training Room GameObject was used, and it’s Material used a shader from Ciconia Studio [21].

Figure 10: Wolf character obtained from Unity Asset Store [11]

```
if(animationStateController.startGame == false){
    distance = 8;
    source.mute = true;
}else if(distance > -6 && animationStateController.startGame == true){
    source.mute = false;
    distance--;
}
bool isAttack = animator.GetBool("isAttack");

Vector3 follow = charPosition.position - charPosition.forward * distance ;
transform.position +=(follow - transform.position) * delay;
if(animationStateController.blendX != 0){
    transform.Rotate(new Vector3(0f, animationStateController.blendX, 0f));
}
transform.LookAt(charPosition.transform);
```

Figure 11: The Wolf following algorithm, credited to James Brady on Youtube [9]

3.8 Sound Integration[7]

Audio files can be downloaded and added to the Unity3D project files. A GameObject can then be given an Audio Source component. The ”mute” state of this component can then be manipulated through

3.9 Scene Changing

Unity3D project Build Settings contain the scene ranking (See Figure 14), the highest being the scene that loads first. To switch scenes, one must first use ”using UnityEngine.SceneManagement;” at the top of the scripts. Then, load the scene via a scene’s buildIndex (As shown in Figure 13) [5]. To obtain the scene by name, use SceneManager.GetSceneByName [4], and to obtain a buildIndex, use Scene.buildIndex [3]

3.9.1 The Death Method

If Maynards has a -10 or lower y-axis positional coordinate (Image 2 of Figure 15), he has fallen into the river (Image 1 of Figure 15). This calls the Failed() function in Maynard’s Animation State Controller, which loads the ”Failed Sreen” scene (Image 3 of Figure 15).

3.9.2 The Winning Method

In order to finish a VRun level, the character must traverse to the Trident (See Figure 16). The 3-dimensional distance formula is used to calculate the distance between Maynard and the ship (As shown in Figure 17). Firstly, the Unity3d editor ”tag” feature was used to create a unique ”CharacterScene1” tag, which is then associated

[7]The methodology for sound integration in VRun was at the direction of Jimmy Vegas on Youtube [24]

```
void Start()
{
    startCam.SetActive(false);
    mountainCam.SetActive(true);
    alienChar = start.GetComponent<AlienAnimationStateController>();
    StartCoroutine(waiter());
}

IEnumerator waiter()
{
    // Wait for 12 seconds
    yield return new WaitForSecondsRealtime(12);
    mountainCam.SetActive(false);
    startCam.SetActive(true);
    alienChar.startGame = true;
}
```

Figure 12: Camera switching code.

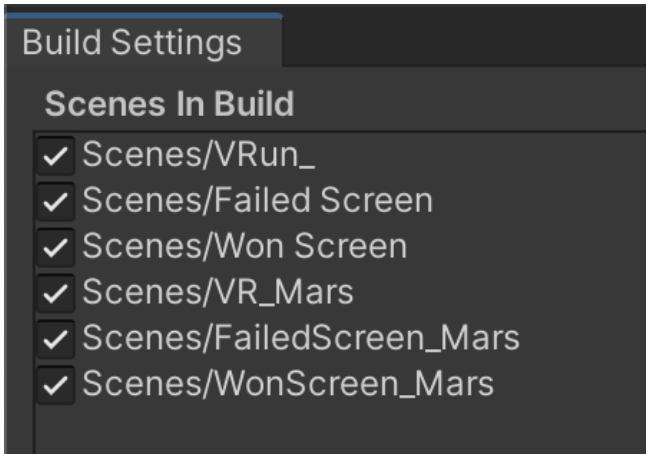


Figure 13: Scene Ranking in Build Settings

```
SceneManager.LoadScene(
    SceneManager.GetSceneByName("Failed Screen").buildIndex +2);
```

Figure 14: Scene Loading Methodology

with the character GameObject present in scene one. charPosition is assigned the transform.position (x, y, z coordinates) of the object with tag ”CharacterScene1”, and shipPosition is assigned transform.position, which simply accesses the (x, y, z) coordinates of the GameObject the script is attached to. After this assignment, the 3d distance formula calculates the distance between the ship and character until they are within a distance of 75, whereby a scene change activation occurs.

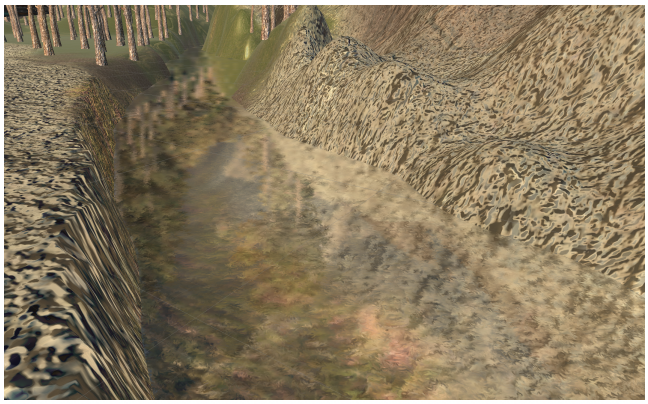
4 User Evaluation

An anonymous user was given the Google Cardboard and iPhone, shown where the VRun app is located, and given no further instruction.

4.1 Results

4.1.1 Google VR Cardboard Ease of Use

The ease of use of the Google Cardboard the Google VR Cardboard headset was noted. A unique way of selecting the VRun app on the



```
Vector3 charPosition = transform.position;
if(charPosition.y < -10){
    Failed();
}
```

```
public void Failed(){
    if(gameHasEnded == false){
        gameHasEnded = true;
        Debug.Log("YOU DIED. GAME OVER.");
        // Restart game.
        SceneManager.LoadScene(SceneManager.GetSceneByName("Failed Screen").buildIndex +2);
    }

}
```

Figure 15: River and dying methodology

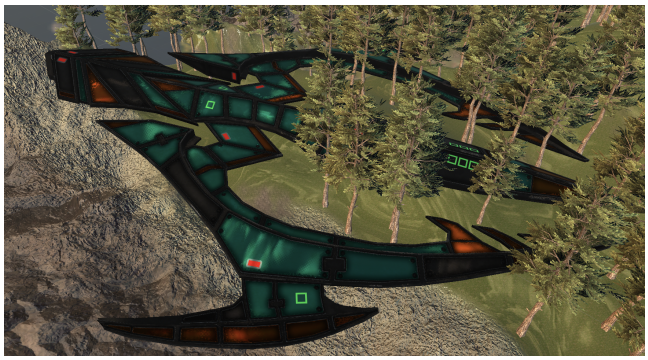


Figure 16: The Trident GameObject [13]

```
void Update()
{
    Vector3 charPosition = GameObject.FindGameObjectWithTag("CharacterScene1").transform.position;
    Vector3 shipPosition = transform.position;
    float distance = Mathf.Sqrt(((shipPosition.x - charPosition.x)* (shipPosition.x - charPosition.x)) +
                                ((shipPosition.y - charPosition.y)* (shipPosition.y - charPosition.y)) +
                                ((shipPosition.z - charPosition.z)* (shipPosition.z - charPosition.z)));
    if(distance < 75f){
        SceneManager.LoadScene(SceneManager.GetSceneByName("Won Screen").buildIndex+3);
    }
}
```

Figure 17: Excerpt from Trident script component

phone by reaching through the nose area was discovered.

4.1.2 Controls and Lag

Head movement controls were confusing at first, but learned quickly. Additionally, the jump trigger was too sensitive.
Lag was significant, particularly in scene two. However, because they desired the game to be more difficult, lag made the game more fun because it produced a challenge.
VRun did not make them disoriented or nauseous, but warned that a more sensitive person may be impacted.

4.1.3 Recommendations

The user would recommend VRun to others, but had feature suggestions. The dog should have the potential to kill the character to make the game experience more "intense". The jump feature should be less sensitive, and latency should be improved. Lastly, the addition of obstacles to jump over and hit would make the game more dynamic.

5 Peer Evaluation

5.1 Work Distribution

5.1.1 Eric Dittus

VRun Project

- Character [17] and Wolf [11] (including Wolf sound [20] at the direction of Lanique Peterson) import and animation with help from iHeartGameDev on Youtube [14]
- Movement and accelerometer integration [14] [2]
- XCode implementation (with help from The Wonderland of Virtual Reality [18] suggested through the research by Lanique Peterson) and VRun VR testing
- XRRig camera and wolf tracking with algorithm credited to James Brady on Youtube [9]
- Scene Two terrain [16] [8] and city [15] (except for blood pool and death scene change integration, as well as the skybox by Lanique Peterson)
- Final GitHub organization and deployment (including ReadMes and second GitHub repo for keyboard compatability, as well as final XCode package)

VRun Research Paper

- Methodology Section

5.1.2 Lanique Peterson

VRun Project

-

VRun Research Paper

-

5.2 Results and Evaluation

5.2.1 Eric Dittus

5.2.2 Lanique Peterson

6 Discussions and Conclusions

VRun has strong potential. In future work, we want to expand the complexity of VRun's interaction. Adding obstacles to the environment is the best way to implement additional interaction between the user and the virtual environment. Obstacles would be objects such as rocks, bushes, or animals. When the user hits an obstacle, the alien slows down; if the alien slows down to a certain distance from the wolf, an attack sequence activates. Once the wolf's attack sequence occurs, the alien will react in pain and decrease in health points. If the alien's health goes below a specific integer, the alien dies.

Immersion is significant within virtual reality systems; therefore, we will focus on the user's sensory factors of seeing, touch, and hearing. To achieve this, we will have to increase the realism within the game by understanding the mechanics of earth science.

We will add additional audio of the alien running throughout various terrains. Blowing wind and grass will be added to the game to enhance the user's perception. Hand controllers will be an added advantage for gameplay. The hand controllers will vibrate slightly as the alien runs through each world, and stronger vibration will activate when the alien collides with an obstacle or gets attacked.

In addition, we will add a user interface to toggle between different worlds. The training room will be interactive and optional. Users can run and jump within the training room before entering the world. There is also more work to be done to correct latency issues. Correcting the latency issues will improve overall user immersion and prevent vertigo.

7 Visual Display Recommendations

We recommend iOS devices with a minimum iOS Version of 11.0 to play VRun.

References

[1] Gameobject.setactive. Unity Documentation.
[2] Input.acceleration. Unity Documentation.
[3] Scene.buildindex. Unity Documentation.
[4] Scenemanager.getscenebyname. Unity Documentation.
[5] Scenemanager.loadscene. Unity Documentation.
[6] Transform.lookat. Unity Documentation.
[7] Waitforseconds. Unity Documentation.
[8] Brackeys. How to make terrain in unity! Youtube.com, June 2019.
[9] J. Brady. Unity tutorial - a temple run prototype in 10 minutes. Youtube.com, February 2021.
[10] derHugo. How to make object transparent on the inside on unity? Stack Overflow.
[11] DzenGames. Wolf animated. Unity Asset Store, May 2016.
[12] Google. Google cardboard. github.com.
[13] J. Grafskiy. Uav trident. Unity Asset Store, September 2016.
[14] iHeartGameDev. Unity's animation system. Youtube.com, May 2021.
[15] KitBash3D. Neo city free sample kit (built-in). Unity Asset Store, August 2020.
[16] LowlyPoly. Stylized terrain texture. Unity Asset Store, September 2019.
[17] Mixamo.com.
[18] T. W. of Virtual Reality. Unity3d demo 1 - how to create a google cardboard vr app and run it on iphone. Youtube.com, October 2020.
[19] L. Peterson and E. Dittus. Vrun. GitHub, July 2022.
[20] ross sinc. Dog growling and running.wav. pixabay.com, January 2022.
[21] C. Studio. Ciconia studio shaders. Unity Asset Store.
[22] Unity. Unity Asset Store.
[23] Unity Technologies. *Working With The Terrain Editor*, December 2020.
[24] J. Vegas. Mini unity tutorial - how to add sound effects. Youtube.com, October 2017.
[25] XyloBetaGames. How to change the skybox in unity. Youtube.com.