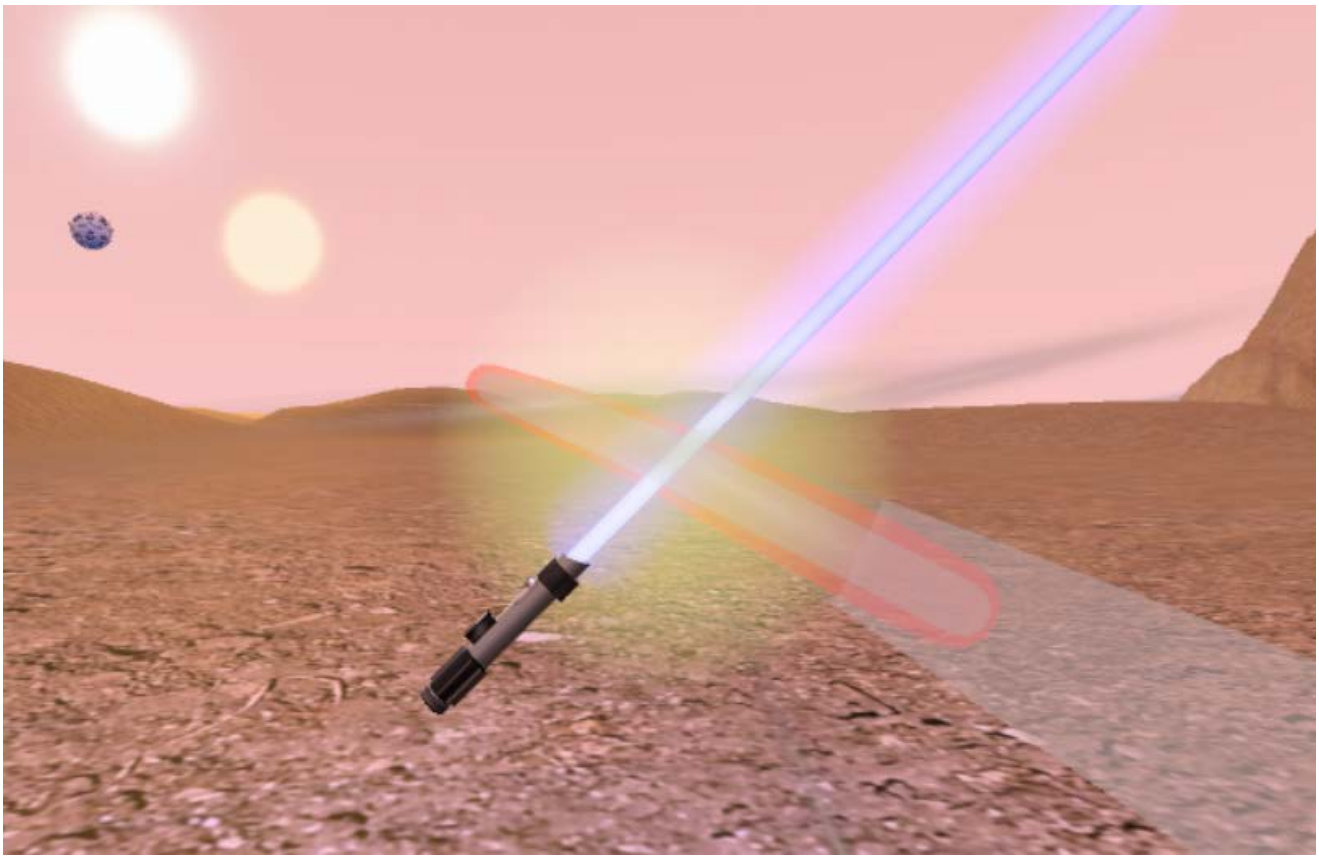

# SPACE KNIGHT TRAINER

## Fundamentals of Virtual Reality

Nicholas P. Doherty
ndoherty@ltu.edu

Video demonstration can be found at:
https://youtu.be/rax_GwNbMHg

# Introduction

The Space Knight Trainer is a Virtual Reality (VR) mini-game that seeks to fully immerse the player in an alternate environment. The goal of the game is to use the laser sward to block blaster bolts originating from the floating droid. Originally, the project was to be run off of an Android phone using Google Cardboard. However, the lack of input controls ultimately led to the game being a fully-fledged game running off of a computer using Steam VR and an HTC Vive headset. The game features a functioning state machine for the floating droid, working colliders to detect and deflect the “Blaster Bolts”, volumetric lines and a system to automatically remove the bolts after use.

# Goals

It did not take long to realize that the google cardboard severely hampered the immersion in VR due to its lack of input. A touch screen phone only allows the user to interact with it via the touch screen; when that is monopolized by a headset, then the user has no way of touching the screen and thus interacting with the game. To solve such an issue, it was decided that the game would be run using controllers and tracking equipment more akin to a traditional VR setup.

It was also important to get the orb within the game to randomly move in a way that seemed plausible. It was inherent that the orb did not follow a preset path, otherwise it would be too easy for the player to simply predict the path and loose a sense of immersion. The same intent applied to the bolts fired from the orb. An orb that simply shot at a preset place would be too simple to ignore by moving to the side.

Finally, there was the laser sword and blaster bolts. It did not take long to find a sword model that looked and sounded like one might expect. The bolts on the other hand, had to be made entirely by scratch, but it did not take long to create bolts that looked realistic from any angle using volumetric lines. However, the game itself depended on the ability of the sword to effect the flight of the bolts in a meaningful way. The challenge would come to be creating a bolt that knew how to act depending on what it hit.

# Solutions

## Immersive environment

In order to solve the issue with user immersion, the decision was made to use a system that allowed the user to control aspects of the game using their hands rather than their phone screen. After it was discovered that LTU currently owns such an HTC Vive, the project was designed with the Vive in mind. Alongside the sounds included in the project, the high framerate and excellent tracking supplied by the Vive do an incredible job of providing a realistic experience.

## Remote movement

The remote's movement was accomplished using a state machine so that it would move between individual states for *idle*, *hover*, and *move*.

### The *idle* state

The *idle* state is for when the orb is going to or from the "off state". It activates the orb when the orb is off and the sword is activated. Conversely, it deactivates the orb when it is in use but the sword is turned off. Figure 1 shows how the state machine controls the movement of the orb depending on the state of the sword. Figure 2 shows how the orb actually cancels all other invokes when *idle* is called to clear the stack of waiting commands; thus ensuring the orb does not try to move and shoot after it has been shut off.

```
case State.Idle:
    if (Lightsaber.GetComponent<LaserSwordScript>().State == 1 || Lightsaber.GetComponent<LaserSwordScript>().State == 3)
    {
        transform.position = Vector3.Lerp(transform.position, new Vector3(transform.position.x, Initheight, transform.position.z), 0.1f);
    }
    else
    {
        CancelInvoke();
        transform.position = Vector3.Lerp(transform.position, new Vector3(transform.position.x, 0.5f, transform.position.z), 0.1f);
    }
    break;
```

Figure 1. moving the orb around onscreen depending on the state of the sword

```
void idle()
{
    Debug.Log("Switching to Idle");
    currentState = State.Idle;
    CancelInvoke();
}
```

Figure 2. Switching to the *idle* state

## The *hover* state

The *hover* state is a simple state that waits a random interval of time (anywhere from one to three seconds) before calling a *fire* function. Figure 3 shows how it then waits again before calling a function that will choose a new position and activate the *move* state.

```
void hover()
{
    Debug.Log("Switching to Hover");
    currentState = State.Hovering;
    Invoke("fire", Random.Range(1.0f, 3.0f));
    Invoke("GotoNewPos", Random.Range(1.0f, 3.0f));
}
```

Figure 3. The *hover* state calls the *fire* and *GotoNewPos* functions

## The *move* state

The *move* state uses the Lerp functionality in C# to smoothly move to the position selected while in the *hover* state. As the orb moves, it will rotate counterclockwise or clockwise depending on its velocity on the x axis. As can be seen in Fig. 4, the orb will actually terminate movement when it is within 0.05 units of the new position. Such a decision was made because the Lerp function will slow the object it is moving as it approaches the endpoint so that the movement appears to be smooth. Without the early termination, the orb could enter what is known as Zeno's dichotomy paradox; where the orb never actually reaches its new location, or at least takes a long time to do so.

```
case State.Moving:
    transform.Rotate(0, ((transform.position.x - newTrainerPos.x) / (Mathf.Abs(transform.position.x - newTrainerPos.x))) * rotateSpeed * Time.deltaTime, 0);
    transform.position = Vector3.Lerp(transform.position, newTrainerPos, 0.1f);
    if(Vector3.Distance(transform.position, newTrainerPos) < 0.05f)
    {
        hover();
    }
    break;
```

Figure 4. The Zeno's paradox avoidance system

## Blaster bolts

The bolts are created using two spheres, one inside the other. The mesh renderers for each are turned off, though the models for each are a gradual fade from solid color to transparency. They also have trail renderers, which draw a trail as they move. The resulting effect is a bolt that has a bright white interior and a red surrounding glow when viewed from any angle so long as it is in motion. Otherwise, the bolt will not be visible. The scripting behind them is a simple *fire* function (Fig. 5) that is called during the *hover* state. The bolt script will then choose a target within a half of a unit of the player's position on the x and z plane, and anywhere from one half of a unit to one and a half units above the ground. The bolts then move towards that point at a given velocity. If they hit the ground, they will be immediately destroyed. If they miss the player and fly into space, they will be destroyed after 10 seconds. If they hit the sword, then the "normal" angle to the contact angle is calculated and the bolts are reflected off the resulting angle. They will then fly for five seconds before being destroyed if they do not hit the ground; in which case they are destroyed immediately. The script that shows the collision detection can be viewed in Fig. 6. The script also includes a *flash* function to touch off a flash of light and a sparking effect whenever the bolts make contact with the blade.

```
void fire()
{
    firing = true;
    var bolt = Instantiate(BlasterBolt);
    var boltComponent = bolt.GetComponent<BlasterBolt>();
    boltComponent.TrainingRemote = this.gameObject;
    boltComponent.Lightsaber = Lightsaber;
    boltComponent.Player = Player;
    bolt.transform.position = this.transform.position;
    AudioSource.PlayClipAtPoint(Blaster, transform.position);
    firing = false;
}
```

Figure 5. The *fire* function that initializes the bolt

```
void OnCollisionEnter(Collision collision)
{
    foreach (ContactPoint contact in collision.contacts)
    {
        Debug.DrawRay(contact.point, contact.normal, Color.white, 1);
        Debug.Log(contact.otherCollider.gameObject);
        if(contact.otherCollider.gameObject.name == "LaserSwordRoot")
        {
            Velocity = Vector3.Reflect(Velocity, contact.normal);
            transform.rotation = Quaternion.LookRotation(Velocity);
            GetComponent<AudioSource>().PlayOneShot(BlockAudio);
            flasher = Instantiate(lightFlash);
            flasher.transform.position = contact.point;
            Destroy(flasher, 0.1f);
            Destroy(gameObject, 5f);
        }
        else
        {
            Destroy(gameObject);
        }
    }
}
```

Figure 6. The collision detection and reaction system is based on the object with which the bolt collides

## Lasersword

Once the blaster bolts where created, not much had to be done to the sword in order to tell it to interact with the bolts. The sword has a collider mesh around the blade that detects contact with the blaster bolts, thus allowing the game to know when the player has successfully blocked incoming fire from the orb. As stated earlier, the meshes on the blade and bolts allow for the bolts to be deflected and cause a bright flash of light. Most of the scripting behind the sword's activation and deactivation came with the model. As such, it made it easy to simply focus on the way the sword interacted with the rest of the game instead of worrying about how to make it work.

# Acknowledgements

I'd like to thank Professor Gordon stein for his enormous contribution to this project. Since I was only allowed to use the Vive unit at LTU when he was supervising, he had the not-so-glorious task of watching me fulfill my 8-year-old dreams of playing with a lightsaber and answering a plethora of coding questions for hours on end. Thank you, Professor Stein; without you, this project would have never happened.

There is also Dr. CJ Chung. I would not have known about this competition were it not for his guidance. Due to his advice, I've not only had a goal to work towards all semester, but I now have an excellent project to put on my resume.

Lastly, I would like to thank Jeff Johnson from digitalruby.com for his lasersword model and Deviant Art user David Guzman for his excellent Jedi Training Remote (also known as the orb). Both models were listed as free to use on their sites. As a result, these individuals saved me many hours of frustration.