CS175 Final Project

Julia Kennish
Vanessa Chambers

May 2022

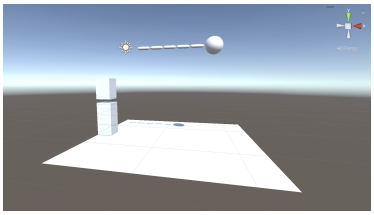
1 Introduction

Our final project "Giants" aimed to simulate features of rigid bodies and joint hierarchies through a simple game of figures spinning around a horizontal bar, akin to gymnasts doing giants on uneven bars or high bar.[1] We implemented this game through the use of custom humanoid models and C# scripts to simulate swinging movement. Specifically, we want to examine the differences between the origination point of the force in the motion, and whether or not this aligns with what is most physically advantageous for gymnasts in real life.

2 On boarding

As this was the first time either of the members of our team had ever used Unity Engine, we began with basic on boarding tutorials to develop a sense of the application. After some experimentation into the features of Unity, we were most interested in the use of joints and simulation of momentum. We began implementing this idea through a wrecking ball (Figure 1) for which we implemented a "knockback" code file that allows for physics interactions (further discussed in a later section).

A: At start



B: In motion

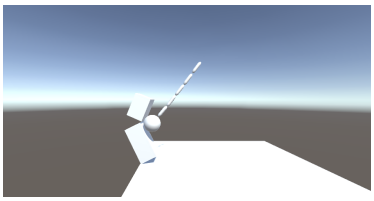


Figure 1: Our wrecking ball experiment

[1] Giants refer to a gymnastics skill in which the gymnast holds onto a horizontal bar and makes a full vertical revolution around the bar.

We then moved on to simulate a person in a similar position, we began with a rudimentary custom model of a person hanging from a bar, made from built-in 3D Unity elements such as cubes, spheres and cylinders. At each joint, a Unity "Hinge" component is used to connect the parts of the body (Figure 2).

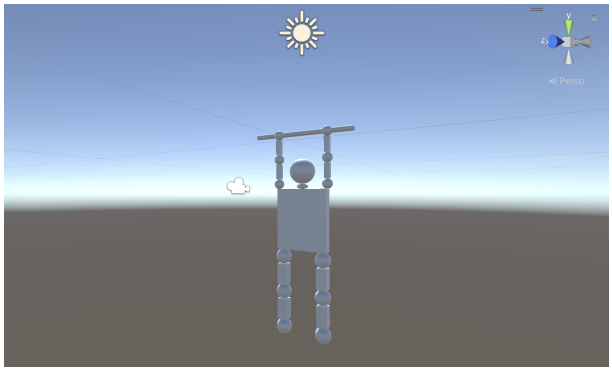


Figure 2: Humanoid figure simulation

After again experimenting with the use of custom code to create a swinging motion (further discussed in the next section), we decided to forge ahead with this project.

3 Simulating the Giant

3.1 The Physics Behind the Giant

We began this simulation by researching the physical intuition behind the giant. We found that Gymnasts generate more momentum to swing around the bar by using their abdominal muscles to position their center of mass closer and farther to the bar. The gymnast begins the motion in a resting hanging position at the bottom of the bar, with their entire body in a stretched position (A). They then use their abdominal muscles to do work to bend their hips upward, bringing their lower body to form a piked position (B). The energy generated from this motion gives the gymnast enough momentum to swing around to the top of the bar; at this position, the gymnast then stretches their body out in a handstand position (C) before swinging back down.

A: stretched position B: piked position C: handstand position

Figure 3: Simone Biles performing a giant on uneven bars at the 2018 gymnastics world championships

The gymnast creates these body positions with the goal of bringing their center of mass close to the bar at the bottom of the swing and far from the bar at the top of the swing. This ultimately creates more centripetal force which swings the gymnast around the bar.

3.2 Modeling the Physics in Unity

We used the human and bar simulation shown in Figure 3 to model the actual gymnast and bar. Our model locks the humanoid figures' hands into place such that they are always attached to the bar, making any grip simulation irrelevant. Because the most relevant muscle group in the giant is the abdomen, we upgraded this base model to include a 3-part torso using capsules, allowing us to further isolate body parts.

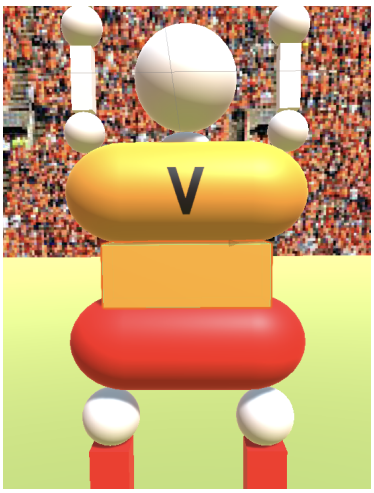


Figure 4: The three part torso modeling the chest, waist, and pelvis as three separate components

We then used this segmented humanoid model to experiment how different types of swings affected the giant.

To model the swinging motion, because we don't fully understand the muscular interactions that are used to generate the energy to complete the giant, we chose write a simplified physics script to create this motion:

```
public class SwingAbs : MonoBehaviour
{
    Vector3 Vec;
    // Update is called once per frame
    void Update()
    {
        if (Input.GetKey(KeyCode.Z)) {
            Vec = transform.localPosition;
            Vec.z += Input.GetAxis("Horizontal") * Time.deltaTime * 20;
            transform.localPosition = Vec;
        }
    }
}
```

In this code, we translate the pressing of the left and right arrow keys into movement in the positive and negative z direction respectively. This function is called every frame, so holding the arrow keys will increase the quantity of motion in either direction. When applied to a specific body part, it will move in the z-direction[2], with the remaining parts of the body following due to the hinge joints. You'll notice in this code that this is predicated on holding the "Z" key on the keyboard, this will be discussed further later in this section.

Due to the law of moments (Figure 5), the further away from the point of a pivot, the larger the impact of the force.

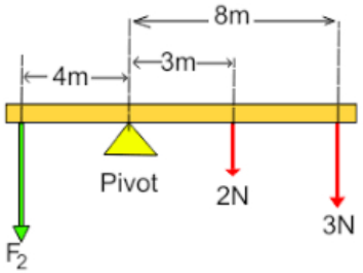


Figure 5: Law of moments

Therefore in our project, the most compelling results came from applying the motion to the feet (as these were the body parts farthest from the bar), despite this not being the muscle group that athletes use in real life. We therefore wanted to display a range of figures with a range of force origination points -

[2]The figures swing around the x-axis

including feet, abdominal muscles, thighs, and a combination of all three - to consider how this impacts their motion. From a UI perspective, we indicated this using a red highlight color on the body element. Because the motion ranges so greatly from figure to figure, we used the bottom row of the keyboard (keys Z, X, C & V) to be held to control a specific figure at a time. Holding the Z key and pressing the arrow key moves the leftmost figure, holding the X key moves the second leftmost figure and so on. In the above code example, the function is specific to the figure using the abdominal muscles and uses the key "Z."

Additionally, we want to add the ability for the player to release the figure from the bar in order to potentially "launch" them from the scene, simulating a dismount from the apparatus. We implemented the following code which uses the pressing of the space bar as the event that triggers this.

```
public class FreezePosition : MonoBehaviour
{
    public GameObject TheThingToFreeze;

    void Update()
    {
        if (Input.GetKeyDown("space"))
        {
            TheThingToFreeze.GetComponent<Rigidbody>().constraints =
                RigidbodyConstraints.None;
        }
    }
}
```

4 Blocks and Knockback

In order to add an additional element to our final project, we wanted to include a collision system to our game. In front of each figure, we have placed a stack of 3 cubes. If the player is able to gain enough momentum with each figure, the cubes will be hit by another rigid body and the knockback function will be triggered. We hope for this to incentivize players to test out all of the figures in order to knock down every stack of blocks, as the ability to knock down a stack with varying levels of difficulty indicates the force gained by the movement initialization from each body part. We wrote the following script for this:

```
public class Knockback : MonoBehaviour
{
    [SerializeField] private float knockbackStrength;

    private void OnCollisionEner(Collision collision) {
      Rigidbody rb = collision.collider.GetComponent<Rigidbody>();
```

```
      if (rb != null) {
         Vector3 direction = collision.transform.position -
             transform.position;
         direction.y = 0;

         rb.AddForce(direction.normalized * knockbackStrength,
             ForceMode.Impulse);
      }
    }
}
```

In this code, we use Unity's collision structure to create a new position with changes in the x and z directions due to contact with another rigid body.

5 Textures and Materials

To add some additional flare and to make our controls clear from a user perspective, we wanted to add some textures to create jerseys for the figures. We did this by applying textures to our capsules which had the following base images.

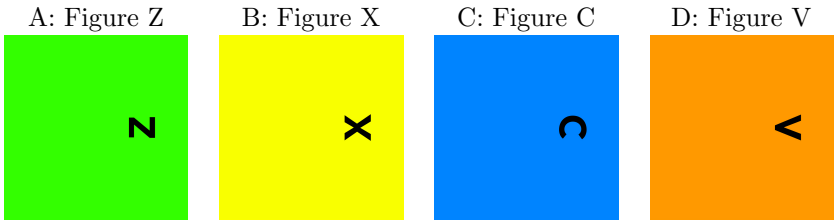


Figure 6: Textures to wrap around chest capsules for figures

We then applied the same hex code to the waist of each figure to extend the texture to create a shirt structure. The figures collectively are demonstrated in Figure 7, with the inclusion of their "shirts" and the highlighted body part that initiates their motion.

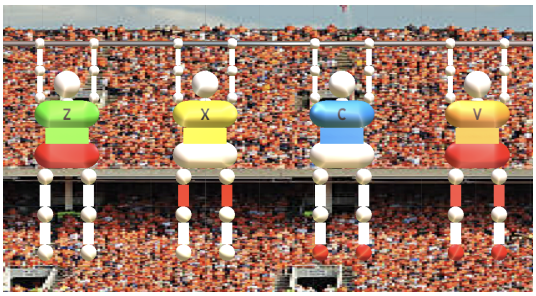


Figure 7: Textured figures

We additionally applied a stock brick texture to all of our cubes.

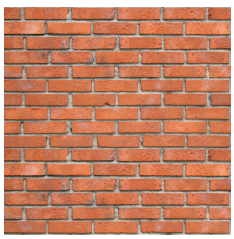


Figure 8: Brick texture

Our remaining material use was in our bar texture, as we used a light gray color with a highly metallic sheen to resemble a metal bar.

6 Skybox

For our background, we wanted to emulate the image of a stadium. We first attempted to use single images that Unity automatically wrapped around a sphere which created unrealistic results. We then progressed to creating a custom skybox. We used the following stock image as a base:

Figure 9: Base stock image

The Unity skybox template uses the format given by Figure 10.

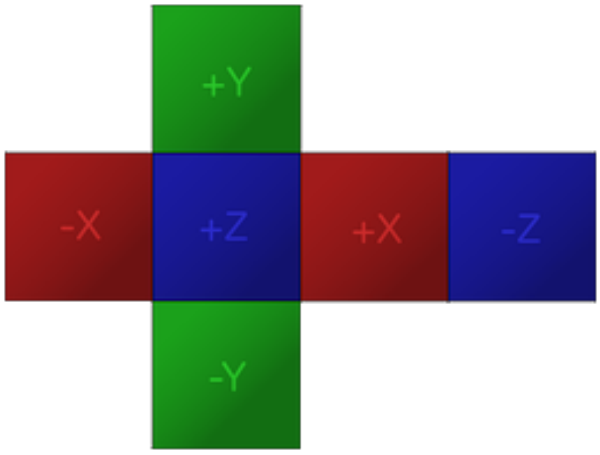


Figure 10: Unity skybox template

Using these two images we created the following skybox. A stock image of grass

was used for the bottom edge of each side which was blended into a solid green color for our -Y side. Similarly, we blended a solid blue color for the sky into the top edge of each side with our +Y being solid blue. For each side (-X, +Z, +X, -Z) we alternately flipped the image and applied some blending at the edges of the sky. The final result is shown in Figure 11 and 12.

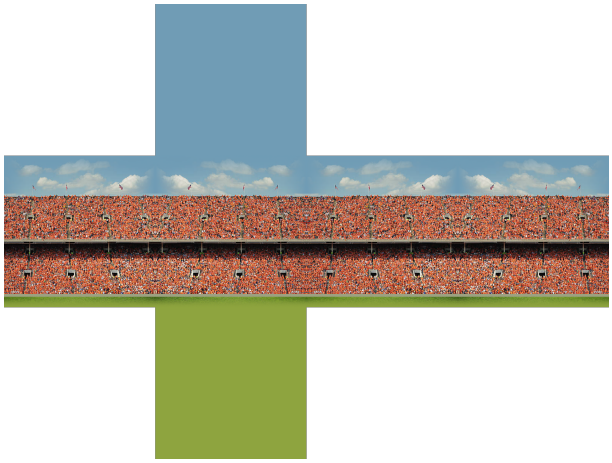


Figure 11: Our final skybox

In-game, our skybox appears as the following:

Figure 12: Our final skybox

7 Menu and Controls

In order to create a more professional user experience, we wanted to create a menu screen that explains the controls. To do these we needed to learn how to

switch between scenes. For this we use Unity's scene manger and buttons to enter the main game scene as detailed by the following section of code.

```
using UnityEngine.SceneManagement;

public class MainMenu : MonoBehaviour
{
    public string game;
    public void Play()
    {
      SceneManager.LoadScene(game);
    }

    public void Quit()
    {
      Application.Quit();
      Debug.Log("Quitting");
    }
}
```

We also wanted to create the option to reset the scene and return to the menu (if the controls are forgotten). This required another script of custom code as follows:

```
using UnityEngine.SceneManagement;

public class Reset : MonoBehaviour
{
    // Update is called once per frame
    void Update()
    {
       if (Input.GetKeyDown(KeyCode.R))
        {
            SceneManager.LoadScene("Giants");
        }
    }
}
```

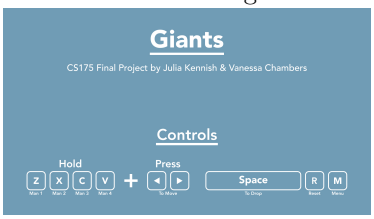
```
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour
{
    // Update is called once per frame
    void Update()
    {
       if (Input.GetKeyDown(KeyCode.M))
```

```
        {
            SceneManager.LoadScene("Main Menu");
        }
    }
}
```

On the press of he "R" key the main game scene is reloaded, and on the press of the "M" key the menu screen is loaded.

Our menu screen is comprised of a single raw image that was created in an external program (for more advanced customization) as well as two interactable buttons.

A: Raw image



B: With buttons

Figure 13: Our menu screen

This menu screen details clearly all of our controls and includes a "Quit" button as well as the "Play" button. The player can quit the game by either navigating to the menu screen and pressing "Quit," or by using CMD + Q on Mac or CTRL + Q on Windows.

8 Conclusion

After putting all of the pieces together, our final product for our game can be seen here in Figure 14 and 15.

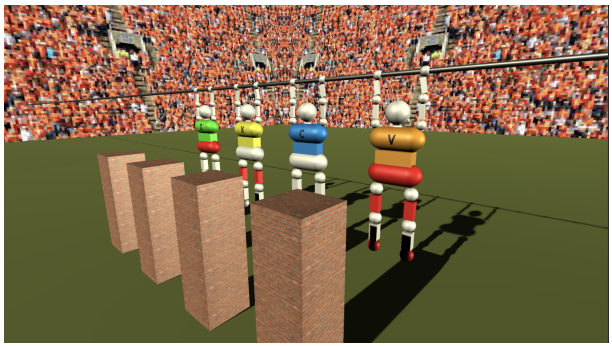


Figure 14: Game load in at start

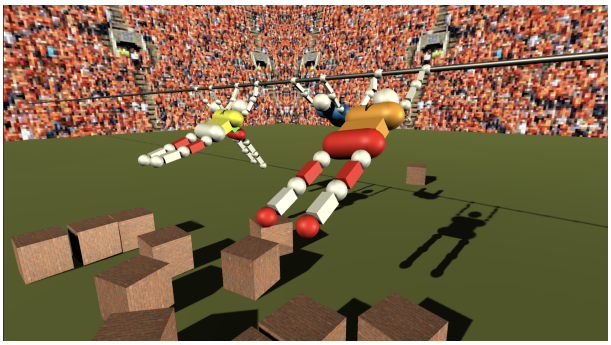


Figure 15: Game in motion

We found in the end that the most effective movement comes from the final figure which uses all three parts of the body (the figure labelled "V") as the highest quantity of force is used for it. This result makes intuitive sense given that the actual giant requires swing from the abdomen, while our physics model rewards work done further away from the pivot. We also found some slight graphical bugs in our code, as moving some elements of the bodies, particularly the thighs and the abdomen, causes some breaking in our hinge model.

Overall we are very happy with our final output in terms of play-ability. We would have loved to have done more research into the realism elements in order to improve it further. Specifically, we could continue to build on this model by proportionally weighting the body part components and fine tuning the joint interactions between these components. The unity "ragdoll" 3D object could be useful in creating this, however, the ragdoll simulates a relaxed body with no muscular support. Therefore, a better model would add some sort of musculature to the ragdoll to better simulate an gymnast's body.

Given these limitations and potential improvements, this project was a great way to begin to learn the basic elements of Unity and how to go about applying what we learned throughout CS175 to game graphical structures.

9 Acknowledgements and References

9.1 Acknowledgements

Thank you to Professor Gortler and the CS 175 teaching staff for guiding us through our first foray into computer graphics!

9.2 References

- How to Implement Knockback In Unity - Unity video tutorial
- [TUTORIAL] Make a Rope Bridge and Wrecking Ball in UNITY that uses Rigidbody Physics - Unity video tutorial

- Creating A Main Menu Screen - Complete Unity Menu System - Unity video tutorial
- Transforming Objects Movement Using C# Scripts In Unity - Unity article
- Unfreeze Position Unity - Unity article
- How to restart scene properly - Unity answer board
- HORIZONTAL BAR GIANT SWING CENTER OF GRAVITY MOTION COMPARISONS by Phillip J. Cheetham - Giant physics
- The Physics of the Gymnastics Giant - Rhett Allain — WIRED