Appendix A – Initial Interview With Client

Client: "Hey, I've been having trouble understanding projectile motion in my physics class. The concepts just don't click when I'm looking at them in a textbook."

Developer: "I see, visualizing the concepts in action might help. What if there was a way to interact with those principles directly?"

Client: "That would be amazing. I'm thinking, maybe a game where I could adjust angles and power to see how they affect the path of a projectile?"

Developer: "A game could certainly make learning more engaging. We could simulate different projectile types and show real-time trajectories based on your input adjustments."

Client: "Exactly, and if I could get immediate feedback on whether I hit a target, it might make the learning process quicker and clearer."

Developer: "We could include a summary of the physics after each shot, too—like showing how different angles and forces affect the motion. And progressively harder levels could help solidify those concepts."

Client: "That's perfect. If the game had that, I bet it wouldn't just help me, but lots of students struggling with the same concepts."

Developer: "Alright, I'll get started on developing a prototype. Your idea could become an invaluable educational tool!"

Appendix B – Success Criteria

Client: "I had a chance to think about what we discussed last time, and I've come up with some features that I think are key for the game. First off, I want to be able to choose from different projectiles. It's not just about seeing the path—it's about understanding how size and weight affect it."

Developer: "That's a good point. So, for our first success criterion, we'll ensure there's a variety of projectiles available for selection. Different weights and sizes will help illustrate the principles of mass and aerodynamics. We'll need to implement a system for you to select and view these differences in the game."

Client: "Exactly. And I want to control the launch. Like, actually changing the angle and power myself, so I can see what happens."

Developer: "Got it. The second criterion will allow you to adjust the launch angle and power. We'll include controls that are intuitive to use. This interactivity will definitely engage users more than static examples."

Client: "For sure. And when I make those changes, I want to see the path the projectile will take before I launch it—like a prediction."

Developer: "Okay, so real-time trajectory visualization will be our third criterion. As you adjust the settings, the projected path will update. This visual aid should help players predict the outcome based on their inputs."

Client: "Right, and after I take the shot, I need to know if I hit the target immediately. It's frustrating in games when you're left guessing."

Developer: "Understood. Providing immediate feedback after each launch will be our fourth success criterion. Whether it's a hit or miss, you'll know right away, and this should help in learning by trial and error."

Client: "Oh, and it would be awesome if after the shot, there was a brief explanation of what happened with the physics. Like, why did the projectile go this far, or why did it drop so quickly?"

Developer: "That would be our fifth criterion: offering a summary of the physics principles demonstrated by each launch. It should help connect the experience with the learning goals of the game."

Client: "Lastly, I don't want it to be too easy. I mean, it should get harder as I get better, right? Maybe add in some obstacles that I have to think about and use what I've learned to get past?"

Developer: "Absolutely, the sixth success criterion will involve level progression. We'll introduce new challenges at each stage, requiring you to apply the physics concepts you've learned. It'll keep the game engaging and educational."

Client: "That sounds perfect. I think if you can nail these criteria, we'll have a game that could really help people like me."

Developer: "Great, I'll draft up a development plan that aligns with these success criteria and get started right away. Your input is invaluable in making this game both fun and educational."

Appendix C: Post Development

Developer: "I'm pleased to report that development is complete. We've hit each success criterion we outlined. Let's walk through them. First, the game now allows players to select from a range of projectiles, each with unique weights and sizes. This feature is fully operational."

Client: "That's fantastic! It was important for me to see how different projectiles behave. Can I adjust the angles and power too?"

Developer: "Yes, that's our second success criterion. The launch angle and power are fully adjustable with intuitive controls, and the effects on trajectory are immediately evident in the gameplay."

Client: "Great to hear. Does it show me the projected path before launching?"

Developer: "It does. The third criterion, real-time trajectory visualization, is one of the highlights. It updates dynamically as you adjust the launch parameters, giving you a clear preview of the projectile's path."

Client: "Perfect. And what about feedback after I launch?"

Developer: "The fourth success criterion has been met with an immediate feedback system. After each launch, you'll know instantly if you've hit the target. It makes for a quick and effective learning cycle."

Client: "I was also hoping to see some kind of summary of the physics involved after each launch. Is that included?"

Developer: "Indeed, it is. Following each launch, the game provides a summary of the physics principles in action, fulfilling our fifth success criterion. It explains the outcome of your launch in terms of physics, linking the game experience back to educational content."

Client: "That's a relief. And the difficulty? Does it increase as I get better?"

Developer: "Absolutely. Our sixth and final success criterion is the progressive difficulty. New levels with more complex obstacles have been introduced, which will challenge you to use the projectile motion concepts you've learned. It's designed to scale with the player's growing understanding and skills."

Client: "This is more than I could've hoped for. It really feels like this game could change how we learn physics. Thank you for bringing this idea to life!"

Developer: "It was my pleasure. I believe we've created a tool that will make learning physics interactive and fun. I can't wait to see it in the hands of players and hear their feedback."

Appendix D – Code

Launcher

```
using UnityEngine;
using TMPro; // Namespace for TextMeshPro
using UnityEngine.UI; // Namespace for UI elements like Slider

public class launcher : MonoBehaviour
{
    [System.Serializable] // User-defined object
    public class ProjectileType
    {
        public string name;
        public GameObject prefab;
        public float weight; // Additional property to vary launch dynamics
        public float size;
    }

    public float rotationSpeed = 45f; // Speed of rotation
    public ProjectileType[] projectileTypes; // Array of ProjectileTypes
    private int currentProjectileIndex = 0; // Index to track the current projectile
    public bool canLaunch = true; // Control whether the launcher can fire
    private float currentAngle = 0f; // Current angle of the launcher

    // Adjusted for new min and max power
    public float launchSpeed = 0f; // Base launch speed now starts at min power
    public Transform firePoint;
    public float launchPowerAdjustmentSpeed = 1f; // Adjusted speed to cover the new range
effectively
```

```
// UI Elements
public TextMeshProUGUI angleText;
public TextMeshProUGUI powerText;
public Slider angleSlider;
public Slider powerSlider;
public TextMeshProUGUI projectileNameText; // Text element for displaying the projectile name



// Constants for min and max power
private const float MinPower = 0f;
private const float MaxPower = 0.5f;

void Start()
{
  SortProjectilesBySize();
}

void Update()
{
  AdjustLauncherAngle();
  SelectProjectile();
  AdjustLaunchPower();

  if (Input.GetKeyDown(KeyCode.Space) && canLaunch)
  {
    FireProjectile();
  }
```

```
    UpdateUI(); // Update UI elements based on the current angle and power
}
    // Bubble Sort algorithm to sort projectiles by size
void SortProjectilesBySize()
{
    int n = projectileTypes.Length;
    for (int i = 0; i < n - 1; i++) // Outer loop
    {
        for (int j = 0; j < n - i - 1; j++) // Inner loop
        {
            if (projectileTypes[j].size > projectileTypes[j + 1].size)
            {
                // Swap the elements
                ProjectileType temp = projectileTypes[j];
                projectileTypes[j] = projectileTypes[j + 1];
                projectileTypes[j + 1] = temp;
            }
        }
    }
}
ProjectileType FindProjectileByName(string name)
{
    foreach (var projectile in projectileTypes)
    {
        if (projectile.name == name)
            return projectile;
    }
    return null; // Return null if not found
}
```

```
  void AdjustLauncherAngle()
  {
    float angleChange = 0f;
    if (Input.GetKey(KeyCode.UpArrow) || Input.GetKey(KeyCode.W))
    {
      angleChange = rotationSpeed * Time.deltaTime;
    }
    else if (Input.GetKey(KeyCode.DownArrow) || Input.GetKey(KeyCode.S))
    {
      angleChange = -rotationSpeed * Time.deltaTime;
    }

    currentAngle = Mathf.Clamp(currentAngle + angleChange, 0f, 90f);
    transform.rotation = Quaternion.Euler(0f, 0f, currentAngle);
  }

  void SelectProjectile() // Complex selection using simple user input
  {
    if (Input.GetKeyDown(KeyCode.RightArrow))
    {
      currentProjectileIndex = (currentProjectileIndex + 1) % projectileTypes.Length;
    }
    else if (Input.GetKeyDown(KeyCode.LeftArrow))
    {
      currentProjectileIndex = (currentProjectileIndex - 1 + projectileTypes.Length) %
projectileTypes.Length;
    }
```

```
  }

  void AdjustLaunchPower() // User-defined method with parameters
  {
    if (Input.GetKey(KeyCode.Q))
    {
      launchSpeed = Mathf.Min(MaxPower, launchSpeed + launchPowerAdjustmentSpeed *
Time.deltaTime); // Increase power within max limit
    }
    else if (Input.GetKey(KeyCode.A))
    {
      launchSpeed = Mathf.Max(MinPower, launchSpeed - launchPowerAdjustmentSpeed *
Time.deltaTime); // Decrease power within min limit
    }
  }

  void FireProjectile()
  {
    if (canLaunch)
    {
      GameObject selectedPrefab = projectileTypes[currentProjectileIndex].prefab;
      GameObject projectileInstance = Instantiate(selectedPrefab, firePoint.position,
firePoint.rotation);
      Vector3 initialVelocity = transform.right * launchSpeed *
projectileTypes[currentProjectileIndex].weight; // Calculate initial velocity considering weight
      projectileInstance.GetComponent<Transform>().localScale *=
projectileTypes[currentProjectileIndex].size;

      Projectile projectileController = projectileInstance.GetComponent<Projectile>();
      if (projectileController != null)
```

```
        {
            projectileController._velocity = initialVelocity; // Assuming Launch() is a method handling velocity
        }

        canLaunch = false; // Prevent further launches
        Invoke("ResetLaunch", 2); // Reset launch capability after 2 seconds
      }
   }

   void ResetLaunch() // User-defined method to reset launch capability
   {
      canLaunch = true;
   }

   void UpdateUI() // Modified method to update UI elements
   {
      angleText.text = $"Angle: {currentAngle.ToString("F1")} ";
      float tempLaunchSpd = launchSpeed * 100;
      powerText.text = $"Power: {tempLaunchSpd.ToString("F1")}";
      angleSlider.value = currentAngle / 90f;
      powerSlider.value = (launchSpeed - MinPower) / (MaxPower - MinPower);
      projectileNameText.text = projectileTypes[currentProjectileIndex].name; // Update the projectile name text
   }
}
```

ObjectPlacer

```
using UnityEngine;


public class ObjectPlacer : MonoBehaviour
{
    public GameObject objectToPlaceFirstPrefab;
    public GameObject objectToPlaceSecondPrefab;


    void Start()
    {
        PlaceObjects();
    }


    void PlaceObjects()
    {
        GameObject firstObjectInstance = null;
        GameObject secondObjectInstance = null;


        if (objectToPlaceFirstPrefab != null)
        {
            // Instantiate and place the first object at y = 0 with a random x between 0 and 8.5
            float randomXFirst = Random.Range(0f, 8.5f);
            firstObjectInstance = Instantiate(objectToPlaceFirstPrefab, new Vector3(randomXFirst, 0f, 0f),
Quaternion.identity);
        }


        if (objectToPlaceSecondPrefab != null)
        {
            // Ensure the second object is placed to the left of the first object by 2 to 5 units
```

```
        float randomXSecondOffset = Random.Range(2f, 5f);
        float secondObjectX = 0f;
        if (firstObjectInstance != null)
        {
            float firstObjectX = firstObjectInstance.transform.position.x;
            secondObjectX = Mathf.Clamp(firstObjectX - randomXSecondOffset, 0f, 8.5f); // Clamp to ensure it remains within bounds
        }
        else
        {
            // If the first object isn't instantiated, just place the second object randomly within the specified range
            secondObjectX = Random.Range(0f, 8.5f - randomXSecondOffset); // Adjust range to account for offset
        }

        // Instantiate and place the second object at y = 5 with the calculated x position
        secondObjectInstance = Instantiate(objectToPlaceSecondPrefab, new Vector3(secondObjectX, 5f, 0f), Quaternion.identity);

        // Randomly adjust the second object's scale in the y direction between 0.5 and 4
        float randomYScale = Random.Range(0.5f, 4f);
        Vector3 currentScale = secondObjectInstance.transform.localScale;
        secondObjectInstance.transform.localScale = new Vector3(currentScale.x, randomYScale, currentScale.z);
    }
  }
}
```

Projectile

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

public class Projectile : MonoBehaviour
{
    private Rigidbody2D rb; // Rigidbody component for physics calculations
    public Vector3 _velocity;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        rb.velocity = _velocity;
    }
    private void OnCollisionEnter2D(Collision2D collision)
    {
        //Debug.Log("Hit");
        if (collision.gameObject.CompareTag("Target"))
        {
            //Debug.Log("Target");
            StartCoroutine(ResetAfterDelay());
        }
    }
    IEnumerator ResetAfterDelay()
    {
```

```
        yield return new WaitForSeconds(1); // Wait for 1 second
        SceneManager.LoadScene(SceneManager.GetActiveScene().name); // Reset the scene
    }

}
```