Bishop's University

Computer Games Design

Assignment 1

Instructed By: Prof. Russell Butler

Submitted By:

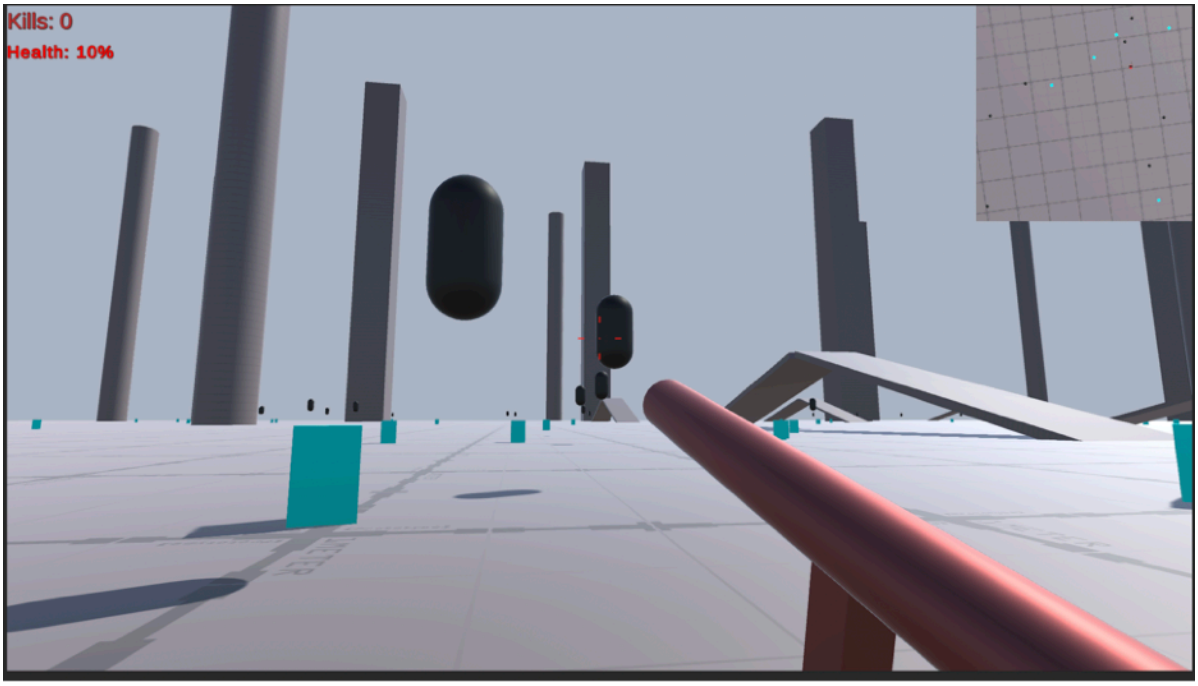
Jagjot Singh

1) Game features implemented:

We tried to implement the following features in the our assignment:

A) 100% functionality off the basic camera control for movement(WSAD), look, and jump with gravity.

B) 100% functionality of the dynamic crosshair but we changed how it behaves(bigger) when enemy is far.



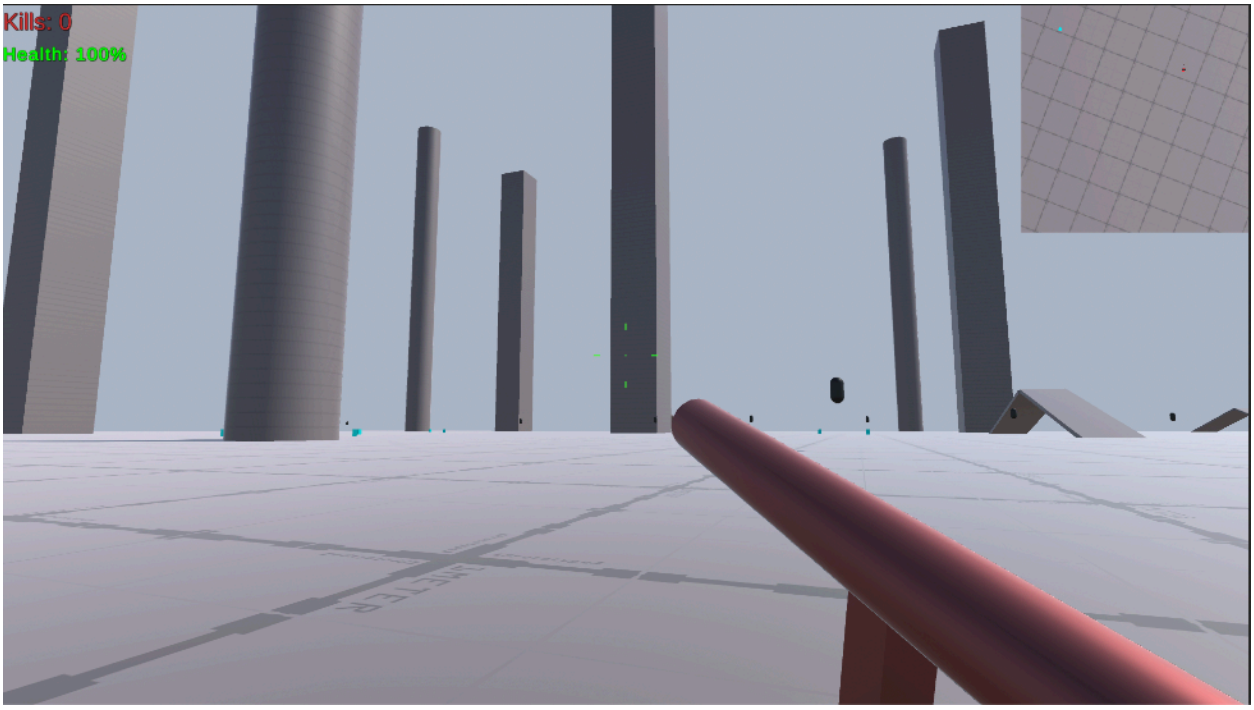
Crosshair on the enemy(capsule) and is in red color and is bigger.

Code screenshot for dynamic crosshair:

```
void Start()
{
    crosshairImages = GetComponentsInChildren<Image>();
    // Debug.Log("crosshairImages: " + crosshairImages.Length);
}

// Update is called once per frame
0 references
void Update()
{
    RaycastHit rHit;
    bool didHit = Physics.Raycast(fpsCam.transform.position, fpsCam.transform.forward, out rHit);
    if (didHit)
    {
        // Debug.Log("CrosshairOn: " + rHit.transform.name);
        // hit.rigidbody.AddForceAtPosition(transform.forward * impactForce, hit.point);
        // Debug.Log("Enemy: " + rHit.transform.CompareTag("Enemy"));
        bool isEnemy = rHit.transform.CompareTag("Enemy");
        // Debug.Log("Distance: " + rHit.distance);
        if (isEnemy)
        {
            foreach (var image in crosshairImages)
            {
                image.color = Color.red;

            }

        }
        else
        {
            foreach (var image in crosshairImages)
            {
                image.color = Color.green;

            }
        }
        if (rHit.distance < normalEnemyDistance)
        {
            currentSize = Mathf.Lerp(currentSize, minSize, Time.deltaTime * timeMultiplier);
        }
        else if (rHit.distance < normalEnemyDistance * 2)
        {
            currentSize = Mathf.Lerp(currentSize, idleSize, Time.deltaTime * timeMultiplier);
        }
        else
        {
            currentSize = Mathf.Lerp(currentSize, maxSize, Time.deltaTime * timeMultiplier);
        }
    }
    else
    {
        foreach (var image in crosshairImages)
        {
            image.color = Color.green;

        }
        currentSize = Mathf.Lerp(currentSize, maxSize, Time.deltaTime * 2f);
    }
    crosshair.sizeDelta = new Vector2(currentSize, currentSize);
}
```

We used ‘Raycast’ to implement our logic for the same.

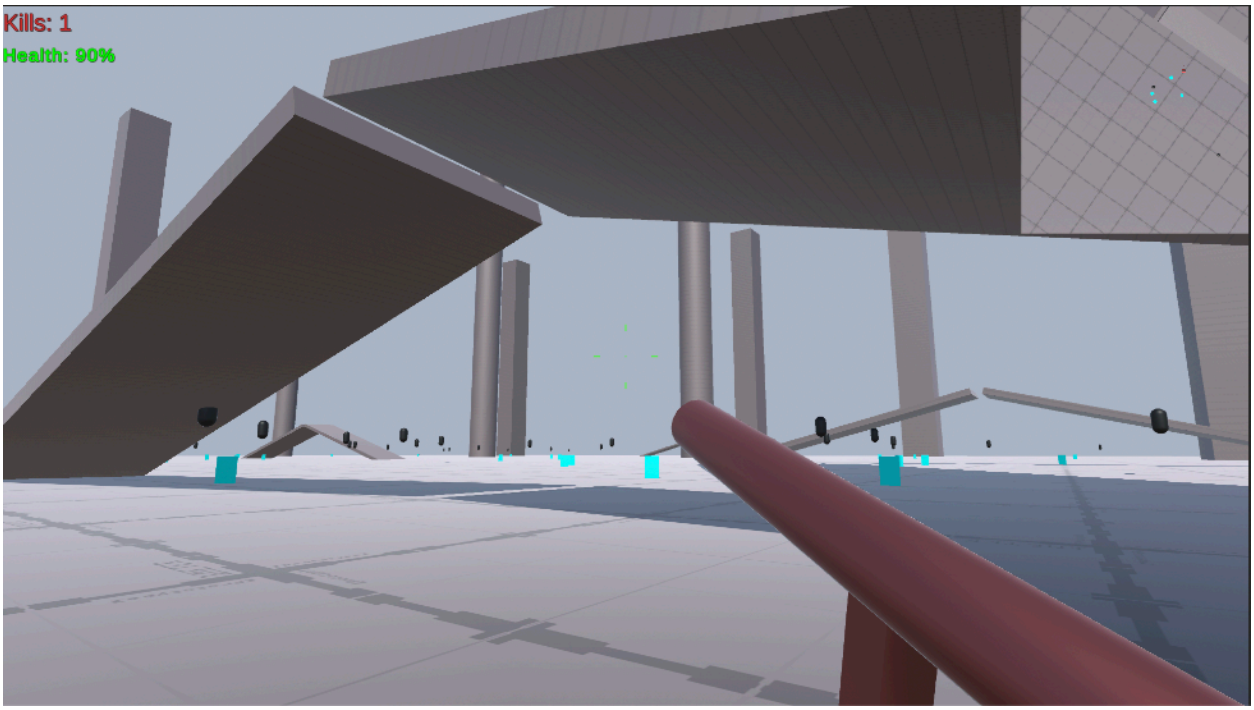
Screenshot when crosshair is not on the enemy:



Here color of the crosshair is green.

D) Game level appearance:

As this was my first assignment, I decided not to use additional assets but to focus on the extra functionality and game logic improvement. But I still tried to add couple of shapes as pillars and bridges.



E) Game sounds:

I added sound effects to the game, there is a game theme, enemy sounds and gun short sound effects.

F) Camera control:

Basic camera control with mouse added. Sensitivity depends on the system.

```
public class PlayerCamera : MonoBehaviour
{
    1 reference
    public float sensX;
    1 reference
    public float sensY;

    2 references
    public float lookLimit = 45f;
    5 references
    float xRotation;
    3 references
    float yRotation;

    1 reference
    public Transform orientation;
    // public Transform gun;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        // Get mouse input
        float mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime * sensX;
        float mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime * sensY;

        yRotation += mouseX;
        xRotation -= mouseY;
        xRotation = Mathf.Clamp(xRotation, -lookLimit, lookLimit);

        // Cam Roation and Orientation
        transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);
        orientation.rotation = Quaternion.Euler(0, yRotation, -xRotation);
        // gun.rotation = Quaternion.Euler(xRotation, 0, 0);
    }
}
```

G) Player and enemy behaviour:

We used “Raycast” to implement shooting. Player can shoot enemies and after taking a certain amount of damage from the player enemies dies and it is resisted as a “kill” for the player. If any enemy touches the player before dying then player will take some damage which will reflect on the player’s health.

```
rb.AddForce(Physics.gravity * (gravityScale - 1) * rb.mass);//Adding downward gravity force(for fast fall)
if (grounded)
    movePlayer();

if (Input.GetAxisRaw("Fire1") == 1 && Time.time >= nextTimeToFire)
{
    nextTimeToFire = Time.time + 1f / fireRate;
    muzzleFlash.Play();
    GameObject bs = Instantiate(bulletShell, muzzle.position, Quaternion.identity);
    bs.GetComponent<Rigidbody>().AddForce(bs.transform.forward * 2f, ForceMode.Impulse);
    RaycastHit rHit;
    bool didHit = Physics.Raycast(fpsCam.transform.position, fpsCam.transform.forward, out rHit, shootRange);

    if (didHit)
    {
        // Debug.Log("Hitted: " + rHit.transform.name);
        // hit.rigidbody.AddForceAtPosition(transform.forward * impactForce, hit.point);
        EnemyHandler handler = rHit.transform.GetComponentInParent<EnemyHandler>();

        if (handler != null)
        {
            handler.takeDamage(gunDamage);
        }
    }
```

```
void EnemyHandler.takeDamage(float amount)
```

```
void OnCollisionEnter(Collision collisionInfo)
{
    if (collisionInfo.transform.CompareTag("Enemy") && this.idOflastEnemyThatGaveDamage != collisionInfo.gameObject.GetInstanceID())
    {
        this.idOflastEnemyThatGaveDamage = collisionInfo.gameObject.GetInstanceID();
        EnemyHandler eh = collisionInfo.transform.GetComponentInParent<EnemyHandler>();
        this.playerCurrentHealth -= eh.hitDamage;
        // Debug.Log("PlayerHealth: " + this.playerCurrentHealth);
        // Debug.Log("EnemyId: " + collisionInfo.gameObject.GetInstanceID());
        // Destroy(collisionInfo.gameObject);// not working ???
        eh.die();
        GameInfo gi = this.gameInfo.GetComponent<GameInfo>();
        gi.updatehealth((this.playerCurrentHealth / this.playerMaxHealth) * 100);
        if (this.playerCurrentHealth <= 0f)
        {
            gi.gameOver();
        }
    }
}
```

H) Extra features implemented:

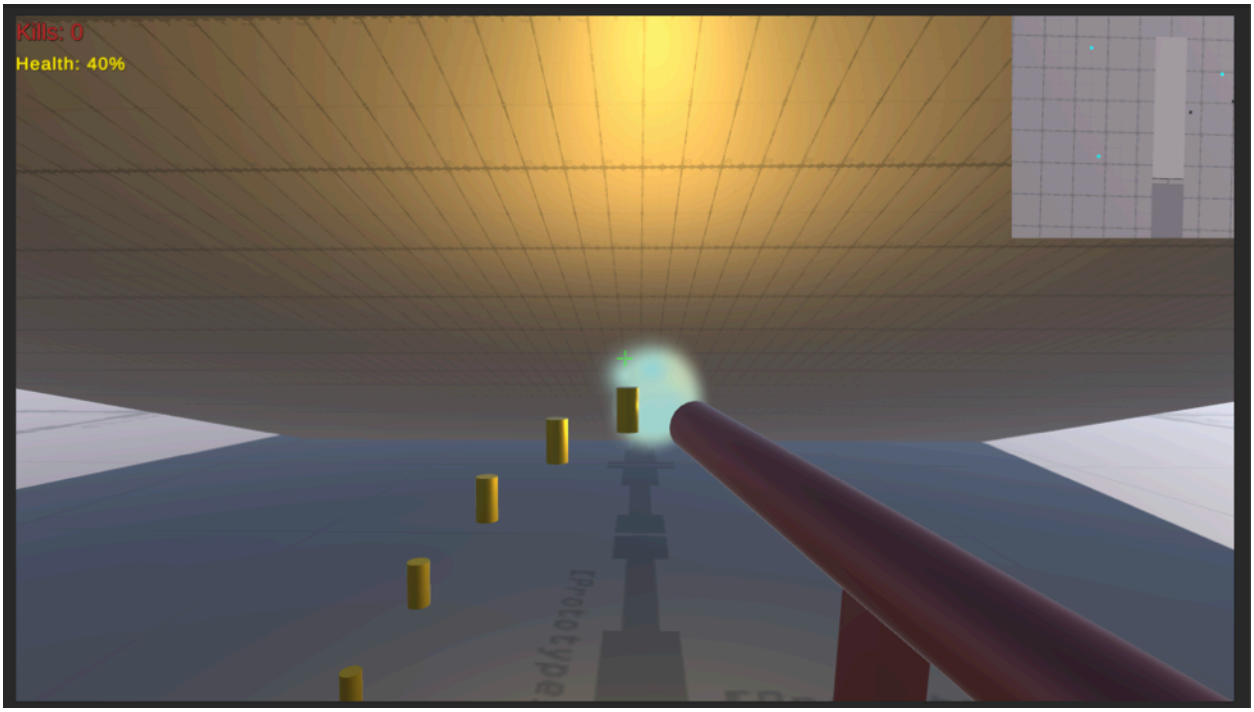
a) Bullet shell drop while shooting.

b) Particle effect and point light effect while shooting.

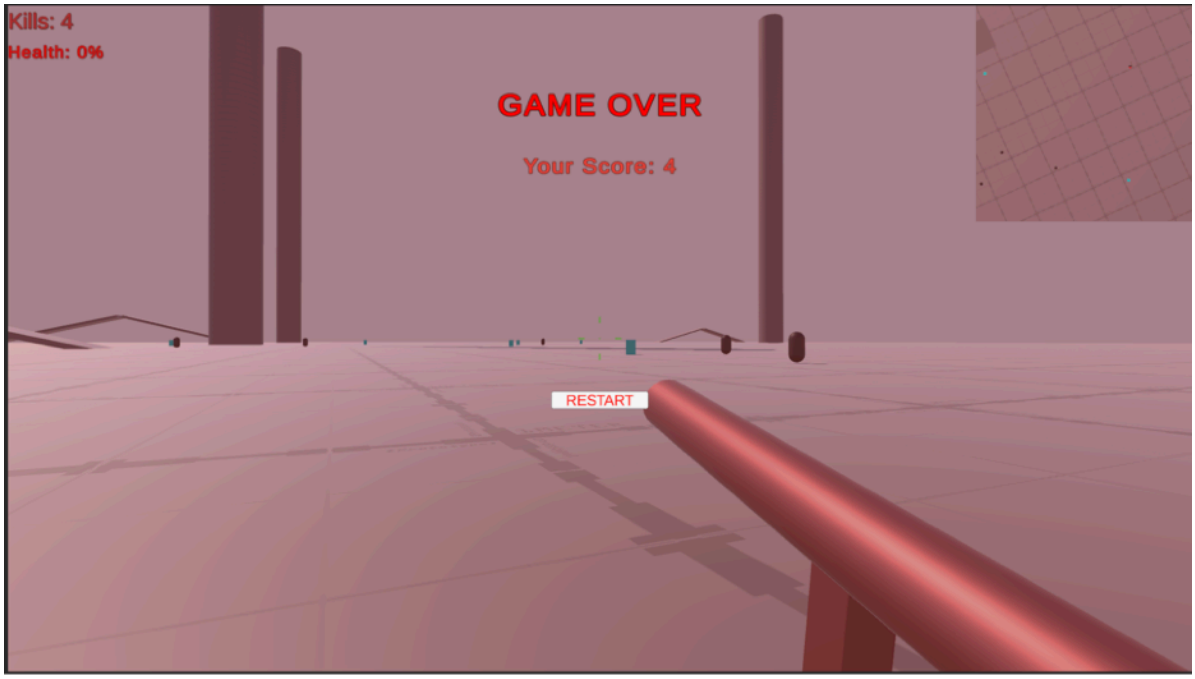
c) Minimap

d) Kill counter

e) Dynamic health presenter(Color will green when health >=70 % , yellow when health is >30% but < 70% and finally red when health is <30%)



f) Game over screen with restart button:



```
public void incrementKill()
{
    this.enemyKills++;
}

1 reference
public void updatehealth(float healthPercent)
{
    this.healthPercent = healthPercent;
}

1 reference
private void updateHealthText()
{
    playerHealthTMP.text = "Health: " + this.healthPercent + "%";
    if (this.healthPercent >= 70)
    {
        playerHealthTMP.color = Color.green;
    }
    else if (this.healthPercent > 30)
    {
        playerHealthTMP.color = Color.yellow;
    }
    else
    {
        playerHealthTMP.color = Color.red;
    }
}

0 references
public void restartGame()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
    Time.timeScale = 1f;

}

1 reference
public void gameOver()
{
    Transform gameOverPanelTransform = transform.Find("GameOverPanel");
    gameOverPanelTransform.gameObject.SetActive(true);
    playerScoreTMP.text = "Your Score: " + this.enemyKills;
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;
    Time.timeScale = 0f;
}
```

Code for changing health, kills and game over logic.

I) Features can be implemented in future:

a) More enemies with different behaviour.

b) Better visual and sound effects.

c) Pause and Start screen.

d) Hi score saving in memory.

e) Using better assets wherever necessary.

f) Changing the size of visual game objects based on the screen size.

J) Files submitted:

a) Assignment1final(windows64).zip: Windows version of the game. Game executable(Assignment1.exe) inside 'Assignment1final' folder.

b) source_jagjot_singh.zip: Source code files

c) Final_Report_Assign1.pdf: This file.