Games Programming – GUIs Assignment 7

Game Manager:

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

public class GameManager : MonoBehaviour
{
    public enum GameState
    {
        Menu,
        Playing
    }

    public GameState gameState = GameState.Menu;
    private int score = 0;
    private int highScore = 0;
    private int lives = 3;
    public TextMeshProUGUI scoreText;
    public TextMeshProUGUI highScoreText;
    public TextMeshProUGUI livesText;
    public GameObject menuCanvas;
    public GameObject gameCanvas;
    public Button playButton;

    public int currentGameLevel = 1;
    public GameObject asteroidPrefab;
    public GameObject spaceshipPrefab;
    public GameObject bulletPrefab;

    List<GameObject> activeAsteroids = new List<GameObject>();

    void Start()
    {
        // Set the camera's position
        Camera.main.transform.position = new Vector3(0f, 40f, 0f);
        Camera.main.transform.LookAt(new Vector3(0f, 0f, 0f), Vector3.up);

        if (gameState == GameState.Menu)
        {
            DisplayMenuGUI();
        }

        CreatePlayerSpaceship();

        // Start button's click event.
        playButton.onClick.AddListener(PlayButtonClicked);

        // Call the method to start a new level
        StartNewLevel();
    }

    void StartNewLevel()
    {
        score = 0;

        // Destroy all leftover asteroids from the previous game
        DestroyLeftoverAsteroids();
```

```
        currentGameLevel++;
        AddToScore(10);

        // Calculate the number of asteroids based on the current game level
        int numAsteroids = currentGameLevel + 1;

        for (int i = 0; i < numAsteroids; i++)
        {
            // Generate a random spawn position within the screen boundaries
            Vector3 spawnPosition = new Vector3(Random.Range(-15f, 15f), 0f,
Random.Range(-15f, 15f));

            // Ensure the Y position is at ground level (0)
            spawnPosition.y = 0f;

            // Add a buffer to the Z position to prevent immediate wrap-around
            spawnPosition.z += 2f;

            // Instantiate asteroid
            GameObject asteroid = Instantiate(asteroidPrefab, spawnPosition,
Quaternion.identity);
            activeAsteroids.Add(asteroid);
        }
    }

    void CreatePlayerSpaceship()
    {
        Quaternion rot = spaceshipPrefab.transform.rotation;
        Instantiate(spaceshipPrefab, Vector3.zero, rot);
    }

    void DisplayMenuGUI()
    {
        // Enable or disable UI elements
        menuCanvas.SetActive(true);
        gameCanvas.SetActive(false);
        playButton.gameObject.SetActive(true);


        // Attach button click handlers
        playButton.onClick.AddListener(PlayButtonClicked);
    }

    void PlayButtonClicked()
    {
        // Switch the game state to "Playing."
        gameState = GameState.Playing;

        // Disable the menu canvas and enable the game canvas.
        menuCanvas.SetActive(false);
        gameCanvas.SetActive(true);

        // Call the StartNewGame() method.
        StartNewLevel();
    }

    void UpdateScoreGUI()
    {
        if (scoreText != null)
        {
            scoreText.text = "Score: " + score.ToString();
        }
```

```
        if (highScoreText != null)
        {
            highScoreText.text = "High Score: " + highScore.ToString();
        }

        if (livesText != null)
        {
            livesText.text = "Lives: " + lives.ToString();
        }
    }

    void AddToScore(int points)
    {
        score += points;

        // Update the high score if the current score is higher.
        if (score > highScore)
        {
            highScore = score;
        }

        UpdateScoreGUI();
    }

    // Decrease the player's lives and end the game if no lives are left.
    void DecreaseLives()
    {
        lives--;

        if (lives <= 0)
        {
            // Game over, switch back to the menu.
            gameState = GameState.Menu;
            DisplayMenuGUI();
        }
        else
        {
            // Update the UI to show the remaining lives.
            UpdateScoreGUI();
        }
    }

    // Method to destroy all leftover asteroids from the previous game
    void DestroyLeftoverAsteroids()
    {
        foreach (GameObject asteroid in activeAsteroids)
        {
            if (asteroid != null)
            {
                Destroy(asteroid);
            }
        }

        activeAsteroids.Clear();
    }
}
```

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

public class Spaceship : MonoBehaviour
```

```
{
    public float upForce = 8f;  // Adjust the force for forward acceleration.
    public float rotationSpeed = 5f;  // Adjust the rotation speed.
    private Rigidbody rb;
    public GameObject bulletPrefab;
    public float bulletSpeed = 10.0f;
    public float fireRate = 4.0f;
    private float nextFireTime;

    private void Start()
    {
        // Get the Rigidbody component attached to the spaceship.
        rb = GetComponent<Rigidbody>();

        InvokeRepeating("CheckScreenEdges", 0f, 0.2f);
    }

    private void Update()
    {
        // Accelerate forward when the Up arrow is held.
        if (Input.GetKey(KeyCode.UpArrow))
        {
            // Apply a forward force to the spaceship.
            rb.AddForce(transform.up * upForce);
        }

        // Rotate left when the Left arrow is held.
        if (Input.GetKey(KeyCode.LeftArrow))
        {
            Vector3 currentRotation = transform.rotation.eulerAngles;
            currentRotation.y -= 100 * Time.deltaTime;
            transform.rotation = Quaternion.Euler(currentRotation);
        }

        // Rotate right when the Right arrow is held.
        if (Input.GetKey(KeyCode.RightArrow))
        {
            Vector3 currentRotation = transform.rotation.eulerAngles;
            currentRotation.y += 100 * Time.deltaTime;
            transform.rotation = Quaternion.Euler(currentRotation);
        }

        // Fire Bullet
        if (Input.GetKeyDown(KeyCode.Space))
        {
            if (Time.time > nextFireTime)
            {
                FireBullet();
                nextFireTime = Time.time + 1.0f * fireRate;
            }
        }

    }

    void CheckScreenEdges()
    {
        Debug.Log("Current position: " + transform.position);
        // Check if the asteroid has left the screen
        if (Mathf.Abs(transform.position.x) > 30f ||
Mathf.Abs(transform.position.z) > 30f)
        {
            // Wrap around to the opposite side
```

```
            transform.position = new Vector3(-transform.position.x, 0, -
transform.position.z);
        }
    }

    void FireBullet()
    {
        GameObject bullet = Instantiate(bulletPrefab, transform.position,
transform.rotation);
        Vector3 bulletDirection = transform.up;
        bullet.GetComponent<Bullet>().FireBullet(bulletDirection);
    }
}



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

public class Asteroid : MonoBehaviour
{
    Rigidbody rb;
    public float moveSpeed = 6f;
    public GameObject smallAsteroidPrefab;
    public int numSmallAsteroidsToSpawn = 2; // Number of small asteroids to
spawn on collision.
    public GameObject playerShipPrefab; // Reference to the player ship prefab.

    void Start()
    {
        rb = GetComponent<Rigidbody>();

        // Calculate a random direction vector on the XZ plane
        Vector3 randomDirection = Random.onUnitSphere;
        randomDirection.y = 0f;

        // Set the initial velocity based on your moveSpeed
        rb.velocity = randomDirection * moveSpeed;

        // Generate random torque (angular velocity)
        Vector3 randomTorque = new Vector3(
            Random.Range(5f, 15f),
            Random.Range(5f, 15f),
            Random.Range(5f, 15f)
        );

        InvokeRepeating("CheckScreenEdges", 0f, 0.2f);
    }

    void CheckScreenEdges()
    {
        Debug.Log("Current position: " + transform.position);
        // Check if the asteroid has left the screen
        if (Mathf.Abs(transform.position.x) > 25f ||
Mathf.Abs(transform.position.z) > 25f)
        {
            // Wrap around to the opposite side
            transform.position = new Vector3(-transform.position.x, 0, -
transform.position.z);
        }
    }
```

```
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Player"))
        {
            // Destroy the player's ship.
            Destroy(collision.gameObject);

            // Re-create the player's ship in the center of the screen.
            RespawnPlayerShip();
        }
        else if (collision.gameObject.CompareTag("Bullet"))
        {
            // Destroy the bullet.
            Destroy(collision.gameObject);
            SpawnSmallAsteroids(collision.contacts[0].point);

            if (gameObject.CompareTag("Asteroid"))
            {
                // The asteroid was large; spawn small asteroids at the collision
point.
                SpawnSmallAsteroids(collision.contacts[0].point);
            }
        }
    }

    private void SpawnSmallAsteroids(Vector3 spawnPosition)
    {
        for (int i = 0; i < numSmallAsteroidsToSpawn; i++)
        {

            // Instantiate small asteroid prefab at the collision point.
            GameObject smallAsteroidInstance = Instantiate(smallAsteroidPrefab,
spawnPosition, Quaternion.identity);

            // Apply some random velocity to the small asteroids.
            Vector3 randomDirection = Random.onUnitSphere;
            randomDirection.y = 0f;
            smallAsteroidInstance.GetComponent<Rigidbody>().velocity =
randomDirection * moveSpeed;

            Destroy(smallAsteroidInstance, 2f);
        }
    }

    private void RespawnPlayerShip()
    {
        // Re-create the player ship in the center of the screen.
        Quaternion rot = Quaternion.Euler(90, 0, 0);
        Instantiate(playerShipPrefab, Vector3.zero, rot);
    }
}

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

public class Bullet : MonoBehaviour
{
    public float bulletSpeed = 10.0f;

    // You might want to add a reference to an impact effect prefab.
    public GameObject impactEffect;
```

```
    // This method is called when the bullet is created.
    public void FireBullet(Vector3 direction)
    {
        // Set the initial velocity of the bullet based on the direction and
speed.
        GetComponent<Rigidbody>().velocity = direction * bulletSpeed;
    }

    // This method is called when a collision occurs.
    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.CompareTag("Asteroid"))
        {
            // Handle asteroid collisions (e.g., destroy the asteroid and play an
impact effect).
            Destroy(collision.gameObject); // Destroy the asteroid.

            if (impactEffect != null)
            {
                Instantiate(impactEffect, transform.position,
Quaternion.identity);
            }
            Destroy(gameObject); // Destroy the bullet.
        }
    }

    void Update()
    {
        CheckScreenEdges();
    }

    void CheckScreenEdges()
    {
        // Check if the bullet has left the screen
        if (Mathf.Abs(transform.position.x) > 25f ||
Mathf.Abs(transform.position.z) > 25f)
        {
            Destroy(gameObject); // Destroy the bullet.
        }
    }
}
```