Halloween Defense Reference Code

John (한찬응)

**[Player Script]**

**C# TankShooting**

|  |
| --- |
| using UnityEngine; using UnityEngine.UI;  public class TankShooting : MonoBehaviour {     public int m\_PlayerNumber = 1;            public Rigidbody m\_Shell;                 public Transform m\_FireTransform;         public Image powerGauge;     public AudioSource m\_ShootingAudio;       public AudioClip m\_ChargingClip;          public AudioClip m\_FireClip;              public float m\_MinLaunchForce = 15f;      public float m\_MaxLaunchForce = 30f;      public float m\_MaxChargeTime = 0.75f;               private float m\_CurrentLaunchForce;       private float m\_ChargeSpeed;                private bool buttonDown= false;     private bool buttonUp=false;       private void OnEnable()     {         m\_CurrentLaunchForce = m\_MinLaunchForce;     }       private void Start()     {         //m\_FireButton = "Fire" + m\_PlayerNumber;          m\_ChargeSpeed = (m\_MaxLaunchForce - m\_MinLaunchForce) / m\_MaxChargeTime;         powerGauge.rectTransform.localScale = new Vector3(1,0,1);     }       private void Update()     {         if (!buttonDown && !buttonUp)          {             m\_CurrentLaunchForce = m\_MinLaunchForce;         }         else if (buttonDown)          {             // Change the clip to the charging clip and start it playing.             m\_ShootingAudio.clip = m\_ChargingClip;             m\_ShootingAudio.Play ();                 if (m\_CurrentLaunchForce <= m\_MaxLaunchForce)              {                 // Increment the launch force and update the slider.                 m\_CurrentLaunchForce += m\_ChargeSpeed \* Time.deltaTime;             }              else              {                 m\_CurrentLaunchForce = m\_MaxLaunchForce;             }             FireGauge(m\_CurrentLaunchForce);         }              else if (buttonUp)              {                 Button\_Reset ();                 Fire ();             }     }     private void FireGauge(float currentForce)     {         //powerGauge.rectTransform.localScale.Set (1, 0, 1);         float ratio = (currentForce-15f) / (m\_MaxLaunchForce-15f);         powerGauge.rectTransform.localScale = new Vector3(1,ratio,1);      }        private void Fire()     {         powerGauge.rectTransform.localScale = new Vector3(1,0,1);         // Instantiate and launch the shell.         //m\_Fired = true;          Rigidbody shellInstance = Instantiate (m\_Shell, m\_FireTransform.position, m\_FireTransform.rotation) as Rigidbody;          shellInstance.velocity = m\_CurrentLaunchForce \* m\_FireTransform.forward;          m\_ShootingAudio.clip = m\_FireClip;         m\_ShootingAudio.Play ();          m\_CurrentLaunchForce = m\_MinLaunchForce;     }      public void Button\_Down()     {         buttonDown = true;         buttonUp = false;     }     public void Button\_Up()     {         buttonDown=false;         buttonUp=true;     }     public void Button\_Reset()     {         buttonDown=false;         buttonUp=false;     }  } |

**C# PlayerHealth**

|  |
| --- |
| using UnityEngine;  using UnityEngine.UI;  using System.Collections;  using UnityEngine.SceneManagement;  namespace CompleteProject  {  public class PlayerHealth : MonoBehaviour  {  public Image black;  public Animator anim;    public int startingHealth = 100;  public int currentHealth;  public Slider healthSlider;  public Image damageImage;  public float flashSpeed = 1f;  public Color flashColour = new Color(1f, 0f, 0f, 0.1f);  public Color flashColourPlus = new Color(0f,1f,0f,0.1f);  bool isDead;  bool damaged;  bool healed;  void Awake ()  {  currentHealth = startingHealth;  }  void Update ()  {  // If the player has just been damaged...  if(damaged)  {  // ... set the colour of the damageImage to the flash colour.  damageImage.color = flashColour;  }  if (healed)  {  damageImage.color = flashColourPlus;  }  // Otherwise...  else  {  // ... transition the colour back to clear.  damageImage.color = Color.Lerp (damageImage.color, Color.clear, flashSpeed \* Time.deltaTime);  }  // Reset the damaged flag.  damaged = false;  healed = false;  }  public void TakeDamage (int amount)  {  // Set the damaged flag so the screen will flash.  damaged = true;  // Reduce the current health by the damage amount.  currentHealth -= amount;  // Set the health bar's value to the current health.  healthSlider.value = currentHealth;  // If the player has lost all it's health and the death flag hasn't been set yet...  if(currentHealth <= 0 && !isDead)  {  StartCoroutine (Fading("GameOver"));  }  }  public void HealPoint(int amount)  {  // Set the healed flag so the screen will flash.  healed = true;  if (currentHealth != startingHealth) {  currentHealth += amount;  // Set the health bar's value to the current health.  healthSlider.value = currentHealth;  } else {  Debug.Log("The Player health is full");  }  }  void Death ()  {  // Set the death flag so this function won't be called again.  isDead = true;  }  IEnumerator Fading(string SceneName)  {  anim.SetBool ("Fade", true);  yield return new WaitUntil (() => black.color.a == 1);  SceneManager.LoadScene (SceneName);  }  }  } |

**[Enemy Script]**

**C# EnemyAttack**

|  |
| --- |
| using UnityEngine;  using System.Collections;  namespace CompleteProject  {  public class EnemyAttack : MonoBehaviour  {  public int attackDamage = 10;  Animator anim;  GameObject player;  PlayerHealth playerHealth;  void Awake ()  {  // Setting up the references.  player = GameObject.FindGameObjectWithTag ("Player");  playerHealth = player.GetComponent <PlayerHealth> ();  anim = GetComponent <Animator> ();  }  //OnTriggerStay 한 다음에, count seconds to give damage  void OnTriggerEnter (Collider other)  {  // If the entering collider is the player...  if(other.gameObject == player)  {  anim.SetBool("isAttacking",true);  Attack ();  }  }  void Attack ()  {  // If the player has health to lose...  if(playerHealth.currentHealth > 0)  {  // ... damage the player.  playerHealth.TakeDamage (attackDamage);  }  }  }  } |

**C# EnemyAnim**

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  namespace CompleteProject  {  public class EnemyAnim : MonoBehaviour {  Animator anim;  GameObject player;  PlayerHealth playerHealth;  EnemyHealth enemyHealth;  UnityEngine.AI.NavMeshAgent nav;  bool isSinking;  public float sinkSpeed = 2.5f;  public int attackDamage = 10;  void Awake ()  {  // Setting up the references.  anim = GetComponent <Animator> ();  player = GameObject.FindGameObjectWithTag ("Player");  playerHealth = player.GetComponent <PlayerHealth> ();  enemyHealth = GetComponent <EnemyHealth> ();  nav = GetComponent <UnityEngine.AI.NavMeshAgent> (); //jh  }  //OnTriggerStay 한 다음에, count seconds to give damage  void OnTriggerEnter (Collider other)  {  // If the entering collider is the player...  if(other.gameObject == player)  {  // ... the player is in range.  anim.SetBool("isAttacking",true);  enemyHealth.currentHealth = -1;  StartSinking ();  }  }  // Update is called once per frame  void Update () {  // If the enemy and the player have health left...  if(enemyHealth.currentHealth > 0 && playerHealth.currentHealth > 0)  {  // ... set the destination of the nav mesh agent to the player.  nav.SetDestination (player.transform.position);  }  // Otherwise...  else  {  // ... disable the nav mesh agent.  nav.enabled = false;  }  // If the player has zero or less health...  if(playerHealth.currentHealth <= 0)  {  // ... tell the animator the player is dead.  GameObject.Destroy(gameObject);  }  if(isSinking)  {  // ... move the enemy down by the sinkSpeed per second.  transform.Translate (-Vector3.up \* sinkSpeed \* Time.deltaTime);  }  }  public void StartSinking ()  {  // Find and disable the Nav Mesh Agent.  GetComponent <UnityEngine.AI.NavMeshAgent> ().enabled = false;  // Find the rigidbody component and make it kinematic (since we use Translate to sink the enemy).  GetComponent <Rigidbody> ().isKinematic = true;  // The enemy should no sink.  isSinking = true;  // After 2 seconds destory the enemy.  Destroy (gameObject, 0.5f);  }  }  } |

**C# EnemyHealth**

|  |
| --- |
| using UnityEngine;  namespace CompleteProject  {  public class EnemyHealth : MonoBehaviour  {  public int startingHealth = 100;  public int currentHealth;  public float sinkSpeed = 2.5f;  public int scoreValue = 10;  Animator anim;  bool isDead;  bool isSinking;  public int shellDamage=100;  void Awake ()  {  // Setting up the references.  anim = GetComponent <Animator> ();  // Setting the current health when the enemy first spawns.  currentHealth = startingHealth;  }  void OnTriggerEnter (Collider other)  {  // If the entering collider is the player...  if(other.gameObject == GameObject.FindGameObjectWithTag ("shell"))  {  Debug.Log("enemy got hit");  TakeDamage (shellDamage);  }  }  void Update ()  {  // If the enemy should be sinking...  if(isSinking)  {  // ... move the enemy down by the sinkSpeed per second.  transform.Translate (-Vector3.up \* sinkSpeed \* Time.deltaTime);  }  }  public void TakeDamage (int amount)  {  // If the enemy is dead...  if(isDead)  // ... no need to take damage so exit the function.  return;  // Reduce the current health by the amount of damage sustained.  currentHealth -= amount;  // If the current health is less than or equal to zero...  if(currentHealth <= 0)  {  // ... the enemy is dead.  Death ();  }  }  void Death ()  {  // The enemy is dead.  isDead = true;  // Tell the animator that the enemy is dead.  anim.SetBool("isDead",true);  StartSinking ();  }  public void StartSinking ()  {  // Find and disable the Nav Mesh Agent.  GetComponent <UnityEngine.AI.NavMeshAgent> ().enabled = false;  // Find the rigidbody component and make it kinematic (since we use Translate to sink the enemy).  GetComponent <Rigidbody> ().isKinematic = true;  // The enemy should no sink.  isSinking = true;  // After 0.5 seconds destory the enemy.  Destroy (gameObject, 0.5f);  }  }  } |

**[Radar System Script]**

**C# Radar**

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  public class Radar : MonoBehaviour {  public GameObject[] trackObjects;  List<GameObject> radarObjects;  public GameObject radarPrefab;  List<GameObject> borderObjects;  public float switchDistance;  public Transform helpTransform;  // Use this for initialization  void Start () {  creatRadarObjects ();  }    // Update is called once per frame  void Update () {  for (int i = 0; i < radarObjects.Count; i++) {  if(Vector3.Distance(radarObjects[i].transform.position, transform.position) > switchDistance){  //switch to the boader Objects  helpTransform.LookAt(radarObjects[i].transform);  borderObjects [i].transform.position = transform.position + switchDistance \* helpTransform.forward;  borderObjects [i].layer = LayerMask.NameToLayer ("RadarLayer");  radarObjects[i].layer = LayerMask.NameToLayer("InvisibleLayer");  }else{  //switch back to radar Objects  borderObjects [i].layer = LayerMask.NameToLayer ("InvisibleLayer");  radarObjects[i].layer = LayerMask.NameToLayer("RadarLayer");  }  }    }  void creatRadarObjects(){  radarObjects = new List<GameObject> ();  borderObjects = new List<GameObject> ();  foreach (GameObject o in trackObjects) {  GameObject k = Instantiate (radarPrefab, o.transform.position, Quaternion.identity) as GameObject;  radarObjects.Add (k);  GameObject j = Instantiate (radarPrefab, o.transform.position, Quaternion.identity) as GameObject;  borderObjects.Add (j);  }  }  } |

**[Game Manager Script]**

**C# Enemy Manager** WaveSpawner

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  using UnityEngine.UI;  public class WaveSpawner : MonoBehaviour {  public enum SpawnState {SPAWNING, WAITING, COUNTING};  [System.Serializable]  public class Wave  {    public string name;  public Transform[] enemy;  //public Transform enemy2;  public int count; //count of enemy created  public float rate; //spawn rate  }  public Text instruction;  public Wave[] waves;  private int nextWave = 0; //index of the wave that we want to store  public Transform[] spawnPoints;  public float timeBetweenWaves = 5f;  private float waveCoundown;  private float searchCountdown = 1f;  private SpawnState state = SpawnState.COUNTING;  void Start ()  {  if (spawnPoints.Length == 0)  {  Debug.LogError ("No spawn points referenced.");  }  waveCoundown = timeBetweenWaves;  //instruction = GetComponent<Text> ();  }  void Update()  {  if (state == SpawnState.WAITING)  {  //check if enemies are still alive  if (!EnemyIsAlive ())  {  WaveCompleted ();  } else  {  return;  }  }  if (waveCoundown <= 0)  {  //check if we already spawn  if (state != SpawnState.SPAWNING) {  // start spawning wave  StartCoroutine( SpawnWave (waves[nextWave]));  }  }  else  {  instruction.text = "WAVE " + (nextWave + 1);  waveCoundown -= Time.deltaTime;  }  }  void WaveCompleted ()  {  //begin a new round  Debug.Log("Wave Completed");  state = SpawnState.COUNTING;  waveCoundown = timeBetweenWaves;  //check if this is the last wave / if it is, start again from the first wave, for now  if (nextWave + 1 > waves.Length - 1) {  nextWave = 0;  } else  {  nextWave++;  }  }  //checking when enemy is alive  bool EnemyIsAlive()  {  searchCountdown -= Time.deltaTime;  if (searchCountdown <= 0f)  {  searchCountdown = 1f;  if (GameObject.FindGameObjectsWithTag ("enemy").Length == 0)  {  Debug.Log ("all enemy are dead");  return false;  }  }  return true;  }  IEnumerator SpawnWave(Wave \_wave)  {  Debug.Log ("Spawning Wave: " + \_wave.name);  state = SpawnState.SPAWNING;  //spawn  for (int i = 0; i < \_wave.count; i++)  {  for (int j = 0; j < \_wave.enemy.Length; j++)  {  instruction.text = "";  SpawnEnemy (\_wave.enemy[j] );  }  //SpawnEnemy (\_wave.enemy2 );  yield return new WaitForSeconds (1f / \_wave.rate); //wait for second to create new enemy  }  state = SpawnState.WAITING;  yield break;  }  void SpawnEnemy(Transform \_enemy)  {  // Spawn enemy  Debug.Log("Spawning enemy: " + \_enemy.name);  Transform \_sp = spawnPoints [Random.Range (0, spawnPoints.Length)];  Instantiate(\_enemy, \_sp.position, \_sp.rotation);  }  } |

**C# Heal Manager**

|  |
| --- |
| using System.Collections;  using System.Collections.Generic;  using UnityEngine;  namespace CompleteProject  {  public class HealManager : MonoBehaviour  {  public GameObject heart; // The enemy prefab to be spawned.  public float min=20f;  public float max=40f;  public Transform[] healSpawnPoints;  private float healCoundown;  private float timeBetweenHeals;  void Start ()  {  if (healSpawnPoints.Length == 0)  {  Debug.LogError ("No spawn points referenced.");  }  setRandomHealTime ();  }  void Update()  {  healCoundown-=Time.deltaTime;  if (healCoundown <= 0)  {  SpawnHeal ();  setRandomHealTime ();  }  }  void setRandomHealTime()  {  timeBetweenHeals = Random.Range (min, max);  healCoundown = timeBetweenHeals;  }  void SpawnHeal()  {  Transform \_sp = healSpawnPoints [Random.Range (0, healSpawnPoints.Length)];  Instantiate(heart, \_sp.position, \_sp.rotation);  }  }  } |