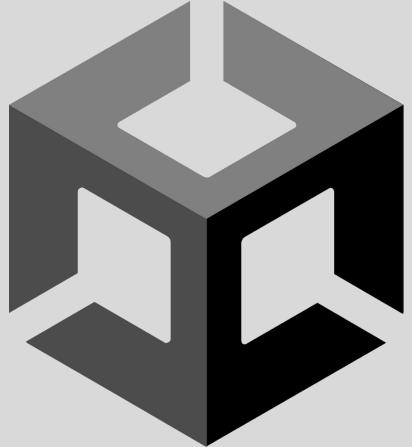


**Arturo Vilar Carretero**  
**2023-2024**

# **Memoria**

# **ESNE RTS**



**Grado en diseño y desarrollo  
de Videojuegos**

# Índice

**Descripción general - Página 3**

**Descripción Técnica - Página 4-9**

# **Descripción general**

**Este proyecto es una tarea de la asignatura de  
Programación gráfica.**

**La intención de la misma es aprender sobre los  
sistemas relacionados con los RTSs como son las  
elección de unidades y uso de máquinas de  
estados**

# Descripción técnica

El juego presenta un sistema principal basado en 2 conceptos básicos del contexto RTS que son el Team, representado por el script CTeam y el que representa la Life que es CLife

CLife lo único de lo que se encarga es de manejar todas las actividades relacionadas con la vida en las unidades

```
3 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public bool Damage(float damageAmount)
{
    if (!invincible)
    {
        currentLife -= damageAmount;

        if (currentLife <= 0f)
            currentLife = 0f;

        _currentNormalized = currentLife / maxLife;
    }

    if (currentLife <= 0f)
    {
        // die
        SendMessage("UnitDied");

        return true;
    }
    else
        return false;
}

0 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void Heal(float healAmount)
{
    _currentNormalized = currentLife / maxLife;
}
```

```
public float maxLife = 100f;
public float currentLife;
private float _currentNormalized;
0 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public float currentNormalized
{
    private set { _currentNormalized = value; }
    get { return _currentNormalized; }
}

public bool invincible = false;

// Start is called before the first frame update
@ Mensaje de Unity | 0 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
void Start()
{
    ResetLife();
}

0 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public bool IsAlive()
{
    return currentLife > 0f;
}

1 referencia | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void ResetLife()
{
    currentLife = maxLife;
    _currentNormalized = currentLife / maxLife;
}
```

CTeam es lo mismo pero con el concepto de los teams que señalan la pertenencia a un equipo o otro en la partida

```
public Color teamColor;
public int teamNumber;

@ Mensaje de Unity | 0 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
private void Start()
{
    ResetColor();
}

4 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void ResetColor()
{
    CSelectable selectable = GetComponent<CSelectable>();
    if (selectable)
        selectable.SetColor(teamColor);
}
```

# Descripción técnica

Tras esto pasamos al script que nos permite seleccionar a las unidades CSelectable que como en los otros componentes de tipo C son encargados de manejo de sus actividades correspondientes

```
[HideInInspector]
public bool selected = false;

public MeshRenderer mesh;

private Collider collider;
private Material outlineMaterial;

private void Awake()
{
    collider = GetComponent<Collider>();
    outlineMaterial = mesh.GetComponent<Renderer>().materials[1];
}

// Start is called before the first frame update
private void Start()
{
    outlineMaterial.SetFloat("_Outline_Width", 0f);
}

public void SetSelected()
{
    outlineMaterial.SetFloat("_Outline_Width", 0.08f);
    selected = true;
}
```

```
3 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void SetDeselected()
{
    outlineMaterial.SetFloat("_Outline_Width", 0f);
    selected = false;
}

1 referencia | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void SetColor(Color color)
{
    outlineMaterial.SetColor("_Outline_Color", color);
}

private void OnDestroy()
{
    Destroy(outlineMaterial);
}
```

A continuación entramos en las unidades o unitFSM ya que todas se rigen por máquinas de estados

```
void Update()
{
    screenPosition = Camera.main.WorldToScreenPoint(transform.position);

    switch (currentState)
    {
        case State.Idle:
            UpdateIdle();
            break;
        case State.GoingTo:
            UpdateGoingTo();
            break;
        case State.Attacking:
            UpdateAttacking();
            break;
        case State.Dying:
            UpdateDying();
            break;
        default:
            break;
    }
}
```

```
1 referencia | Xohat, Hace 5 horas | 1 autor, 1 cambio
protected void UpdateGoingTo()
{
    Vector3 offset = destiny - transform.position;
    float offsetSqrMagnitude = offset.sqrMagnitude;

    if (currentEnemy)
    {
        if (offsetSqrMagnitude <= boundingRadius2 + currentEnemy.boundingRadius2)
        {
            // close enough to attack!
            nmAgent.isStopped = true;

            lastState = currentState;
            currentState = State.Attacking;
        }
        else if (offsetSqrMagnitude <= destinyThreshold2)
        {
            // on destiny but the enemy is no on sight
            nmAgent.isStopped = true;

            lastState = currentState;
            currentState = State.Idle;
        }
    }
    else
    {
        if (offsetSqrMagnitude <= destinyThreshold2)
        {
            nmAgent.isStopped = true;

            lastState = currentState;
            currentState = State.Idle;
        }
    }
}
```

# Descripción técnica

```
i referencia | Xohat, Hace 5 horas | 1 autor, 1 cambio
protected void UpdateAttacking()
{
    if (currentEnemy)
    {
        attackCadenceAux += Time.deltaTime;

        Vector3 offset = currentEnemy.transform.position - transform.position;
        float offsetSqrMagnitude = offset.sqrMagnitude;

        if (offsetSqrMagnitude <= boundingRadius2 + currentEnemy.boundingRadius2)
        {
            if (attackCadenceAux >= attackCadence)
            {
                // Attack
                currentEnemy.life.Damage(meleeAttack);

                attackCadenceAux = 0f;
            }
        }
        else if (offsetSqrMagnitude <= visionSphereRadius2)
        {
            destiny = currentEnemy.transform.position;
            nmAgent.SetDestination(destiny);
            nmAgent.isStopped = false;

            lastState = currentState;
            currentState = State.GoingTo;
        }
        else
        {
            nmAgent.isStopped = true;

            lastState = currentState;
            currentState = State.Idle;

            currentEnemy = null;
        }
    }
    else
    {
        lastState = currentState;
        currentState = State.Idle;
    }
}
```

```
i referencia | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void RightclickOnFloor(Vector3 position)
{
    currentEnemy = null;
    GoTo(position);
}

i referencia | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void RightclickOnTransform(Transform destTransform)
{
    CTeam otherTeam = destTransform.GetComponent<CTeam>();
    if (otherTeam)
    {
        if (otherTeam.teamNumber != team.teamNumber)
        {
            // click on enemy
            GoTo(destTransform.position);
            currentEnemy = destTransform.GetComponent<UnitFSMBase>();
        }
        else
            currentEnemy = null;
    }
}
```

```
2 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public void GoTo(Vector3 destiny)
{
    if (currentState != State.Dying)
    {
        this.destiny = destiny;
        nmAgent.isstopped = false;
        nmAgent.SetDestination(destiny);

        lastState = currentState;
        currentState = State.GoingTo;
    }
}

4 referencias | Xohat, Hace 5 horas | 1 autor, 1 cambio
public virtual void EnemyEntersInVisionSphere(UnitFSMBase enemy)
{
    enemiesInVisionSphere.Add(enemy);
}

public virtual void EnemyLeavesVisionSphere(UnitFSMBase enemy)
{
    enemiesInVisionSphere.Remove(enemy);
}

public void UnitDied()
{
    army?.UnitDied(this);
}

protected virtual void OnGUI()
{
    GUI.Label(new Rect(screenPosition.x - 20f, screen.height - screenPosition.y - 30f, 200f, 20f), currentState.ToString());
}

private void OnDrawGizmos()
{
    Gizmos.color = Color.white;
    Gizmos.DrawWireSphere(transform.position, visionSphereRadius);
}
```

# Descripción técnica

Debido a la duración de esta memoria voy a hablar de la implementación de la unidad de artillería por ultimo. Como todas las unidades se basan en una máquina de estados en este caso con None, Alert, Attacking y Chasing

```
switch (currentArtilleryState)
{
    case ArtilleryState.None:
        base.UpdateIdle();
        break;

    case ArtilleryState.Alert:
    {
        if(alertHitTimerAux <= 0)
        {
            SearchForEnemies();
            alertHitTimerAux = alertHitTimer;
        }
        else
        {
            alertHitTimerAux -= Time.deltaTime;
        }
        break;
    }

    case ArtilleryState.Attacking:
        FireProjectile(enemiesInVisionSphere[0].gameObject.transform.position);
        break;

    case ArtilleryState.Chasing:
        break;
}
```

None es equivalente a un estado Idle.

Alert es un estado basado en principalmente enemigos que estén dentro del área de visión y por medio del método SearchForEnemies marcar si están a la vista de la unidad

# Descripción técnica

```
1 referencia | Xohat, Hace 24 minutos | 1 autor, 2 cambios
private void SearchForEnemies()
{
    if(enemiesInVisionSphere.Count > 0)
    {
        // Throw a ray for each enemy inside vision sphere
        for (int i = 0; i < enemiesInVisionSphere.Count; i++)
        {
            Debug.DrawLine(eyePosition, enemiesInVisionSphere[i].transform.position, Color.yellow);

            Vector3 dir = enemiesInVisionSphere[i].transform.position - (transform.position + eyePosition);
            dir.Normalize();

            RaycastHit hit;

            if (Physics.Raycast(transform.position + eyePosition, dir, out hit, visionSphereRadius))
            {
                CTeam enemy = hit.transform.GetComponent<CTeam>();
                Vector3 enemyPosition = enemy.transform.position;

                if (enemy)
                {
                    // Attack only if we can see the target we are aiming for
                    if (hit.transform.gameObject == enemiesInVisionSphere[i].gameObject)
                    {
                        currentEnemy = enemiesInVisionSphere[i];
                        currentArtilleryState = ArtilleryState.Attacking;
                        attackCadenceAux = 0;
                        return;
                    }
                }
            }
        }
    }
}
```

Principalmente se usa la Lista de enemiesInVisionSphere para comprobar todo esto ya que es el área de alerta. Aquí se traza un raycast hacia cada una de las unidades en el área de alerta y si el raycast conecta con la unidad enemiga se pasa al estado de Attacking si no, se continúa en estado de alerta.

El estado de Attacking llama a este método que lo que hace básicamente es comprobar si el enemigo actual asignado está dentro de la esfera de alerta y si es así se gira hacia la misma para así poder apuntar el proyectil correctamente. tras esto instancia el proyectil y le configura los valores a él mismo

```
1 referencia | Xohat, Hace 30 minutos | 1 autor, 1 cambio
public void FireProjectile(Vector3 targetPosition)
{
    if (enemiesInVisionSphere.Contains(currentEnemy))
    {
        transform.LookAt(currentEnemy.transform.position);
        attackCadenceAux += Time.deltaTime;

        if (attackCadenceAux >= attackCadence)
        {
            // Instantiate the projectile
            GameObject projectileObject = Instantiate(
                projectilePrefab,
                transform.position,
                Quaternion.LookRotation(targetPosition - transform.position)
            );

            ArtilleryProjectile projectile = projectileObject.GetComponent<ArtilleryProjectile>();

            // Set the reference to the spawner, its team, and the target position
            projectile.SetData(this, team);
            projectile.SetTargetPosition(targetPosition);

            attackCadenceAux = 0;
        }
    }
}
```

# Descripción técnica

El objeto instanciado anteriormente es un ArtilleryProjectile que contiene los siguientes datos

```
// Reference to the spawner unit
4 referencias | Xohat, Hace 33 minutos | 1 autor, 1 cambio
public UnitFSMArtillery artillery_unit { get; private set; }

2 referencias | Xohat, Hace 33 minutos | 1 autor, 1 cambio
public CTeam Team { get; private set; }

// Target position
private Vector3 targetPosition;

// These variables will be used to define the parabolic path
public float speed = 10f;

// The amount of damage the explosion will cause
public float explosionDamage = 30f;
```

```
④ Mensaje de Unity | 0 referencias | Xohat, Hace 33 minutos | 1 autor, 1 cambio
void Start()
{
    // Initialize the position to the offset from the artillery unit
    Vector3 offsettedSpawnPositon = new Vector3(artillery_unit.transform.position.x + 2, artillery_unit.transform.position.y + 5, artillery_unit.transform.position.z);

    transform.position = offsettedSpawnPositon;
}
```

```
④ Mensaje de Unity | 0 referencias | Xohat, Hace 33 minutos | 1 autor, 1 cambio
private void Update()
{
    Vector3 offsettedTargetPosition = new Vector3(targetPosition.x, targetPosition.y + 1.5f, targetPosition.z);

    // Move towards the target
    transform.position = Vector3.MoveTowards(transform.position, offsettedTargetPosition, speed * Time.deltaTime);
}

1 referencia | Xohat, Hace 33 minutos | 1 autor, 1 cambio
public void SetData(UnitFSMArtillery spawner, CTeam team)
{
    this.artillery_unit = spawner;
    this.Team = team;
}

1 referencia | Xohat, Hace 33 minutos | 1 autor, 1 cambio
public void SetTargetPosition(Vector3 targetPosition)
{
    this.targetPosition = targetPosition;
}
```

```
④ Mensaje de Unity | 0 referencias | Xohat, Hace 33 minutos | 1 autor, 1 cambio
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.layer == LayerMask.NameToLayer("Selectable"))
    {
        // Check if the object has a CTeam with value 1 component and if so, deal damage
        CTeam team = other.GetComponent<CTeam>();
        CLife life = other.GetComponent<CLife>();

        if (team != null && life != null)
        {
            Debug.Log("has Team");
            if (team.teamNumber == 1 && team.teamNumber == 0)
            {
                Debug.Log("is Valid");
                life.Damage(explosionDamage);
            }
            else if (team.teamNumber == 1)
            {
                Debug.Log("is Valid");
                life.Damage(explosionDamage);
            }
        }

        Destroy(gameObject);
    }
}
```

# Conclusión

Debido a no tener que hacer muchos apartados distintos no ha sido extremadamente desafiante, pero el spawneo del proyectil de manera correcta me ocasionó bastantes problemas.

Estos principalmente radicaban en la parte del movimiento ya que inicialmente quería que hiciera una parábola calculada matemáticamente pero me proporcionaba muchos problemas que no era capaz de debugear, pero por lo que intente parecía principalmente que era por algo relacionado con el dot product de la dirección de visión de la unidad. Al final tras hablar con compañeros opté por el MoveTowards ya que no era un requisito y estaba costando mucho para que no fuera reconocido como requisito de la entrega.

También al principio quería que el impacto del proyectil hiciera un efecto en área y no salió a buen puerto tampoco.

# Descripción técnica