MEMORIA

Programación concurrente

GDDV 4.3

Convocatoria ordinaria
Primera Práctica del Segundo Semestre - 2023/24.

Prototipo RTS

Profesor: Maximiliano Miranda Esteban

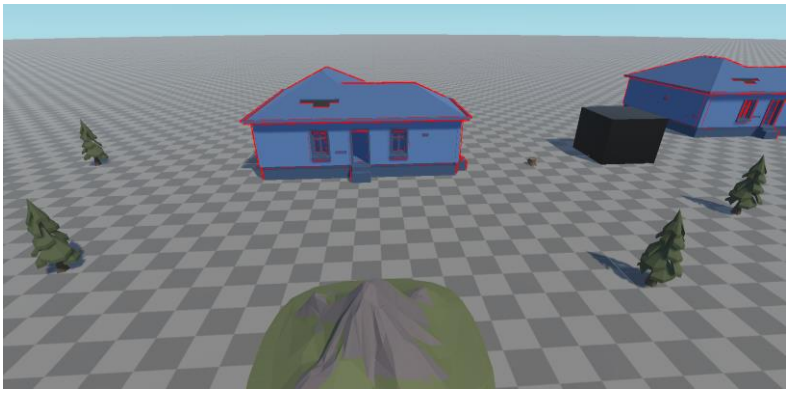
Alumno: Miguel Rodríguez Gallego

Contenido

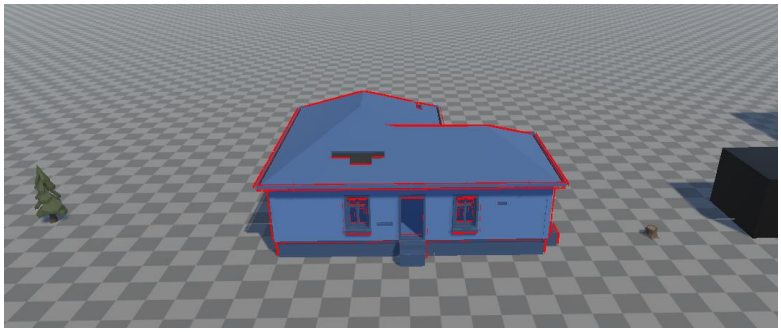
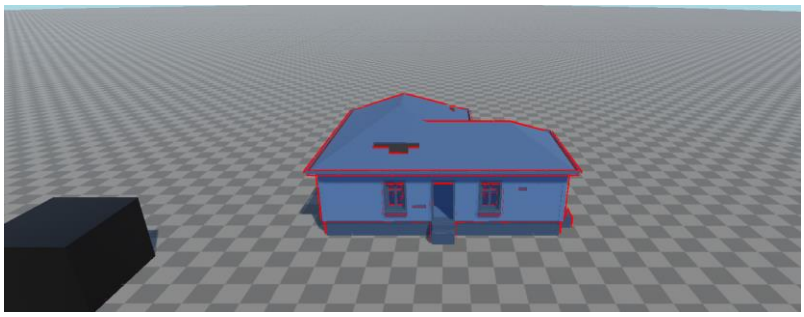
Desarrollo del proyecto

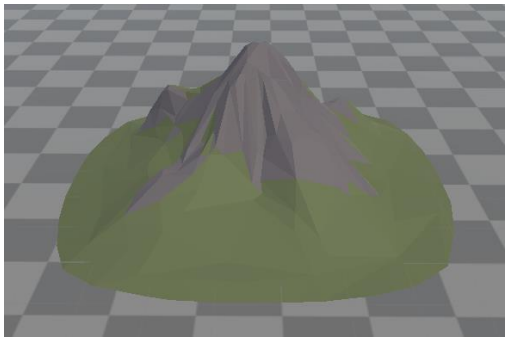
Gestión de escena - Elementos del juego

1 escena.



Composición:

- **Equipos:**
  - Base equipo 1
    - 
  - Base equipo 2
    - 
- **Recursos:**

- Minas - Piedra
  - 
- Árboles - Madera
  - 

- **Entidades:**
  - Cuerpo a cuerpo
    - 
  - Artillero

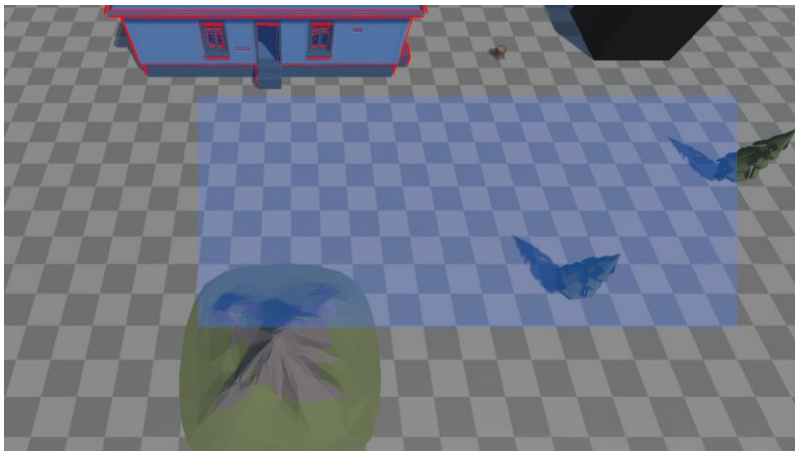
- 

    - Cosechador

- 

- **Canvas:**

    - Seleccionador de unidades

- 

Sistemas

- **Unidades** ➔ Clase base "**UnitBase**" con los métodos y atributos base para todas las unidades.

  Las unidades funcionan por un sistema de herencia con jerarquías, cogiendo todos los métodos y atributos principales de "**UnitBase**".

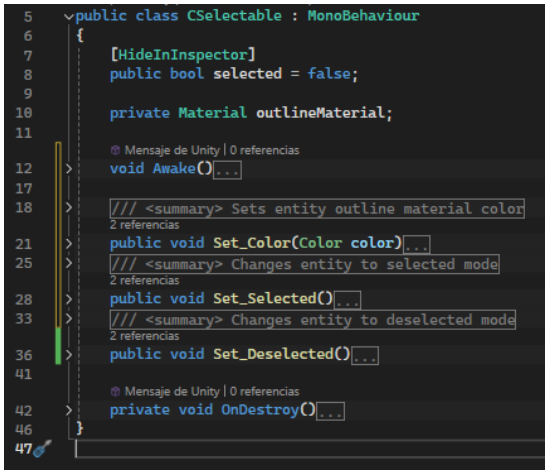
```
11      [HideInInspector] public NavMeshAgent nmAgent;
12      [HideInInspector] public CSelectable selectable;
13      [HideInInspector] public CTeam team;
14      [HideInInspector] public CLife life;
15
        16 referencias
16      public enum State_base...
23      public State_base currentState  = State_base.Idle;
24      public State_base prevState     = State_base.Idle;
25
26      public float destinyThreshold = 0.5f;
27      [HideInInspector] public float destinyThreshold2;
28
29      public float visionSphereRadius = 10;
30      [HideInInspector] public float visionSphereRadius2;
31
32      public float boundingRadius = 1f;
33      [HideInInspector] public float boundingRadius2;
34
35      [Header("Attack")]
36      public float attackRate = 2f;
37      protected float attackRateAux = 0f;
38      public float meleeAttack = 5f;
39
40      [HideInInspector] public Vector3 screenPosition;
41
42      Vector3 currentDestination;
43
44      public ArmyBaseController armyBase;
45      public GameObject armyBase_gameobject;
46
47      protected UnitBase currentEnemy = null;
48      protected List<UnitBase> enemiesOnVisionSphere = new List<UnitBase>();
49
50      Texture2D progressBarEmpty, progressBarFull;
51
52      public UnityEvent<UnitBase> unitDiedEvent;
```

```
        Mensaje de Unity | 0 referencias
54      void Awake()...
        Mensaje de Unity | 4 referencias
61      public virtual void Start()...
        Mensaje de Unity | 0 referencias
74      void Update()...
91
        5 referencias
92      protected virtual void Update_Idle()...
        5 referencias
96      protected virtual void Update_GoingTo()...
        1 referencia
135     protected virtual void Update_Attacking()...
        1 referencia
178     protected virtual void Update_Dying()...
182
183     /// <summary> Cambia el color al del color del equipo
        1 referencia
187     public void Set_TeamColor(Color color)...
192     /// <summary> Cambio del estado actual de la unidad a un estado de idle
        8 referencias
195     protected void Set_Idle()...
202
203     /// <summary> Gestión del click derecho (dirección de movimiento de las unidades ...
        5 referencias
207     public virtual void RightClick_OnFloor(Vector3 position)...
212     /// <summary> Gestión de interacción con edificios
        4 referencias
216     public virtual void RightClick_OnTransform(Transform transform)...
235
        9 referencias
236     public void GoTo(Vector3 destination)...
248
249     /// <summary> Adds enemy to "enemiesOnVisionSphere" list
        4 referencias
253     public virtual void Enemy_Enters_VisionSphere(UnitBase enemy)   => enemiesOnVisionSphere.Add    (enemy);
254     /// <summary> Removes enemy to "enemiesOnVisionSphere" list
        10 referencias
258     public virtual void Enemy_Leaves_VisionSphere  (UnitBase enemy)   => enemiesOnVisionSphere.Remove (enemy);
259
        2 referencias
260     protected virtual void UnitDied()...
271
        Mensaje de Unity | 4 referencias
272     protected virtual void OnGUI()...
```

Las unidades cuentan con unos estados por defecto que son los que se encuentran en este Script, y luego tienen un segundo estado que es el que se introduce en las clases que heredan ya de cada unidad en concreto, como el script de "**UnitHarvester**".

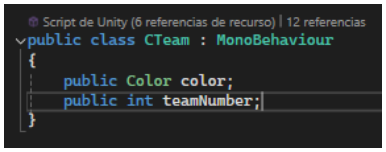
```
25 referencias
public enum State
{
    None,
    GoingToMine,
    GoingToChop,
    Waiting,
    Harvesting,
    ReturningToBase
}
public State state = State.None;
```

Todas las unidades tienen varios scripts de funcionamiento básico.

- **CSelectable**: Para gestionar los cambios visuales al seleccionar a la unidad.
  - 

```
public class CSelectable : MonoBehaviour
{
    [HideInInspector]
    public bool selected = false;

    private Material outlineMaterial;

    Mensaje de Unity | 0 referencias
    void Awake()...

    /// <summary> Sets entity outline material color
    2 referencias
    public void Set_Color(Color color)...
    /// <summary> Changes entity to selected mode
    2 referencias
    public void Set_Selected()...
    /// <summary> Changes entity to deselected mode
    2 referencias
    public void Set_Deselected()...

    Mensaje de Unity | 0 referencias
    private void OnDestroy()...
}
```

    - Este también se encuentra en los edificios de creación de unidades.

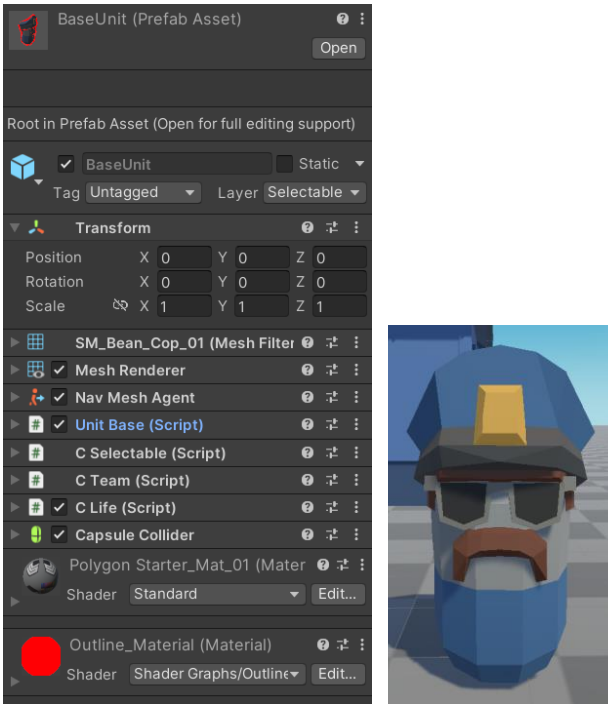
- **CTeam**: Variables básicas para la gestión de equipo.
  - 

```
Script de Unity (6 referencias de recurso) | 12 referencias
public class CTeam : MonoBehaviour
{
    public Color color;
    public int teamNumber;
}
```

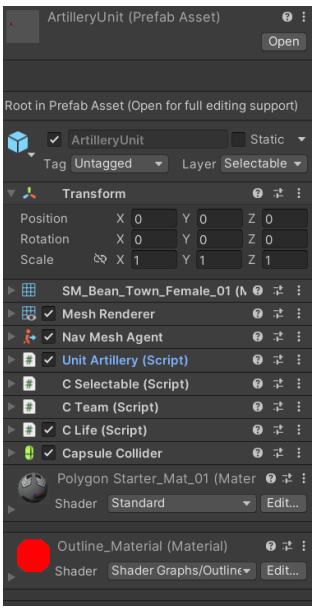
- **CLife**: Gestión de la salud de la entidad

```
Script de Unity (3 referencias de recurso) | 4 referencias
public class CLife : MonoBehaviour
{
    public float maxLife = 100f;
    public float currentLife;
    private float _currentNormalized;
    1 referencia
    public float currentNormalized
    {
        private set { _currentNormalized = value; }
        get { return _currentNormalized; }
    }

    public bool invencible = false;

    private bool hasDied = false;

    Mensaje de Unity | 0 referencias
    private void Start()...

    /// <summary> Resets Entity life
    1 referencia
    public void ResetLife()...
    /// <summary> Checks if health is above 0, so entity
    0 referencias
    public bool IsAlive() => currentLife > 0f;

    /// <summary> Inflicts damage to Entity
    3 referencias
    public bool Damage(float meleeAttack)...
    /// <summary> Heals Entity
    0 referencias
    public void Heal(float healAmount)...
}
```

Gameobjects de las unidades:

- **Entidades:**
  - **Cuerpo a cuerpo**:
    - 

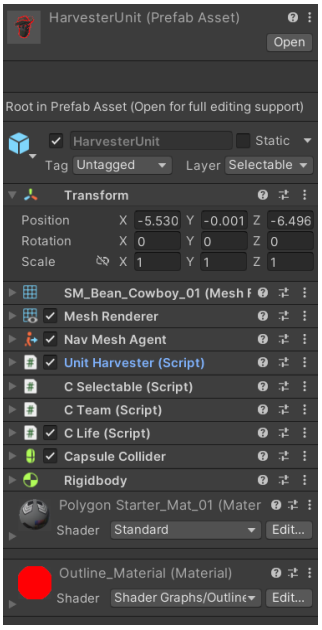
    - Script concreto: "**UnitBase**"
  - **Artillero**: Realiza disparos a distancia a los enemigos de su alrededor.

- 

- Script concreto de unidad: "**UnitArtillery**"
  -

```
public class UnitArtillery : UnitBase
{
    private LineRenderer gunLineRenderer;

    18 referencias
    public enum State
    {
        None,
        Alert,
        Melee,
        Shooting,
        Chasing
    }
    public State currentArtilleryState = State.None;

    public  float shotAttackRate = 1f;
    private float shotAttackRateAux = 0f;

    public float hitDamage = 2f;

    private Vector3 eyesPosition = new Vector3(0f, 1.2f, 0f);

    private float alertHitTimer = 1f;
    private float alertHitTimerAux = 0f;

    private int currentEnemyIndex = 0;

    Mensaje de Unity | 3 referencias
    public override void Start()...

    4 referencias
    protected override void Update_Idle()...
    4 referencias
    protected override void Update_GoingTo()...
    1 referencia
    protected void Update_Shooting()...

    0 referencias
    private void HideGun()...

    /// <summary> Gestión del click derecho (dirección de movimiento de las unidades ...
    4 referencias
    public override void RightClick_OnFloor(Vector3 position)...

    /// <summary> Adds enemy to "enemiesOnVisionSphere" list
    4 referencias
    public override void Enemy_Enters_VisionSphere(UnitBase enemy)...
    /// <summary> Removes enemy to "enemiesOnVisionSphere" list
    10 referencias
    public override void Enemy_Leaves_VisionSphere(UnitBase enemy)...

    /// <summary> Searchs for an enemy in the radious sphere around him
    2 referencias
    private void Search_Enemy(UnitBase enemyToSearch)...
    /// <summary> Searchs for all the enemies in the radious sphere around him
    1 referencia
    private void Search_AllEnemies()...

    Mensaje de Unity | 3 referencias
    protected override void OnGUI()...
}
```

- **Cosechador**: Recolecta recursos para llevarlos a la base de su equipo.

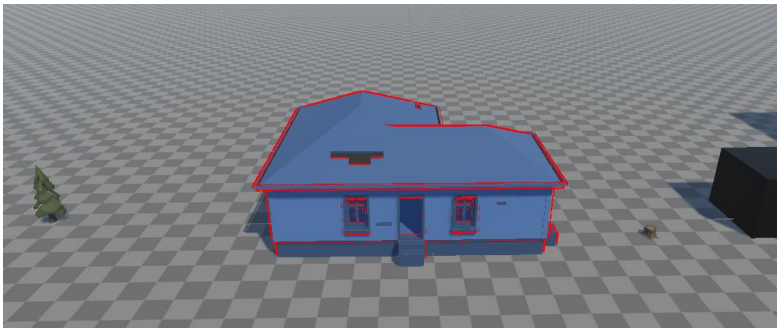


- Script propio de unidad: "**UnitHarvester**"

```
public class UnitHarvester : UnitBase
{

    25 referencias
    public enum State
    {
        None,
        GoingToMine,
        GoingToChop,
        Waiting,
        Harvesting,
        ReturningToBase
    }
    public State state = State.None;

    [Header("Resources")]
    public int    storage_total = 0;
    public int    storage_stone = 0;
    public int    storage_wood = 0;
    public int    harvestAmountPerChop = 1;
    public int    storage_max = 10;
    public  float harvestRate = 1f;
    private float harvestRateAux = 0f;

    // last mine
    private CResources currentMine;
    private Vector3 lastHarvestPosition;
    private int lastHarvestIndex;

    [SerializeField] bool onBase = false;

    Mensaje de Unity | 3 referencias
    public override void Start()...

    4 referencias
    protected override void Update_Idle()...
    4 referencias
    protected override void Update_GoingTo()...

    /// <summary> Gestión del click derecho (dirección de movimiento de las unidades ...
    4 referencias
    public override void RightClick_OnFloor(Vector3 position)...
    /// <summary> Gestión de interacción con edificios
    4 referencias
    public override void RightClick_OnTransform(Transform transform)...

    /// <summary> After waiting for a space to harvest in actual resource, this lead ...
    1 referencia
    public void Finish_Waiting(Vector3 harvestPosition, int index)...
    /// <summary> Unlinks entity completely from current mine he is working on
    3 referencias
    private void Leave_CurrentMine()...

    2 referencias
    protected override void UnitDied()...

    Mensaje de Unity | 0 referencias
    void OnTriggerEnter(Collider other)...

    Mensaje de Unity | 3 referencias
    protected override void OnGUI()...
```

Army base

Script propio base: "**ArmyBaseController**".



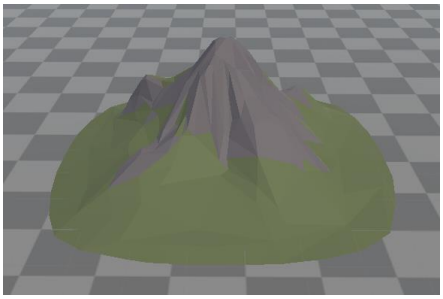
Gestiona principalmente las acciones de spawn y entrada de recursos en la base por parte del jugador y sus unidades.

```
public class ArmyBaseController : MonoBehaviour
{

    [HideInInspector]
    public CTeam team;
    [HideInInspector]
    public CSelectable selectable;

    private Transform spawnPoint;

    [Header("Unit Prefabs")]
    public GameObject baseUnitPrefab;
    public GameObject artilleryUnitPrefab;
    public GameObject harvesterUnitPrefab;

    [Header("Resources")]
    public int actualResources;
    public int Resources_max = 1000;
    public int stone_resource;
    public int wood_resource;


    Mensaje de Unity | 0 referencias
    void Awake()...
    Mensaje de Unity | 0 referencias
    void Start()...

    /// <summary> Set oveline material color of base
    1 referencia
    public void Set_TeamColor(Color color)...

    /// <summary> Base actions: Spawn entites
    1 referencia
    public GameObject Action(int actionType)...

    /// <summary> Resource introduction in base for harvesters
    2 referencias
    public int Introduce_Resources(int cant, Resource resourceType)...
}
```

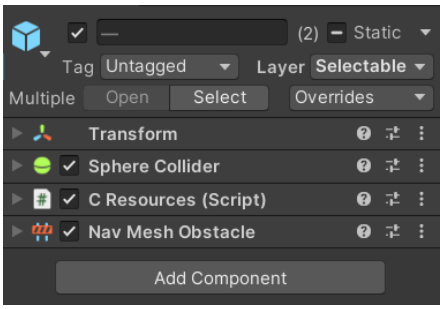
Mines – Resources

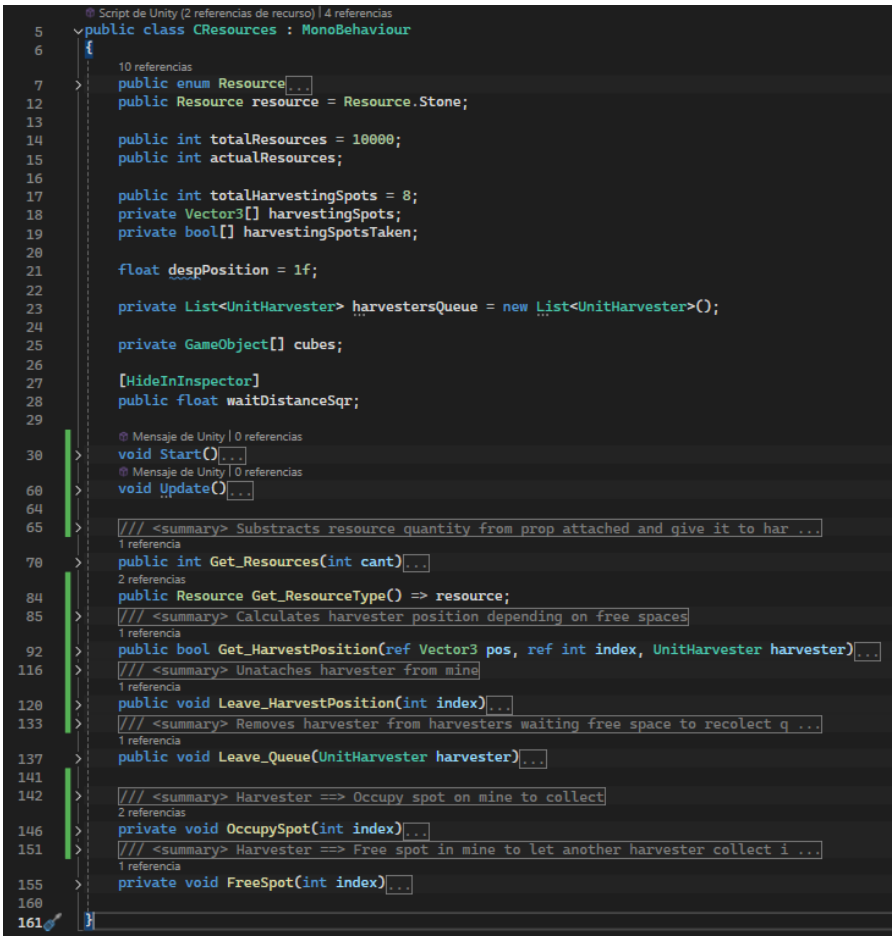
Script principal: "**CResources**".

Gestiona la recolecta de recursos junto a las unidades recolectoras de minería, distinguiendo entre dos tipos de materiales, piedra y madera.



Utiliza una esfera a modo de interacción con el click derecho sobre la estructura y para establecer los puntos de recolecta calculados a su alrededor mediante los valores que introduzcamos en el script CResources en la variable "totalHarvestingSpots".





```
Script de Unity (2 referencias de recurso) | 4 referencias
public class CResources : MonoBehaviour
{
    10 referencias
    public enum Resource...
    public Resource resource = Resource.Stone;

    public int totalResources = 10000;
    public int actualResources;

    public int totalHarvestingSpots = 8;
    private Vector3[] harvestingSpots;
    private bool[] harvestingSpotsTaken;

    float despPosition = 1f;

    private List<UnitHarvester> harvestersQueue = new List<UnitHarvester>();

    private GameObject[] cubes;

    [HideInInspector]
    public float waitDistanceSqr;

    Mensaje de Unity | 0 referencias
    void Start()...
    Mensaje de Unity | 0 referencias
    void Update()...

    /// <summary> Substracts resource quantity from prop attached and give it to har ...
    1 referencia
    public int Get_Resources(int cant)...
    2 referencias
    public Resource Get_ResourceType() => resource;
    /// <summary> Calculates harvester position depending on free spaces
    1 referencia
    public bool Get_HarvestPosition(ref Vector3 pos, ref int index, UnitHarvester harvester)...
    /// <summary> Unataches harvester from mine
    1 referencia
    public void Leave_HarvestPosition(int index)...
    /// <summary> Removes harvester from harvesters waiting free space to recolect q ...
    1 referencia
    public void Leave_Queue(UnitHarvester harvester)...

    /// <summary> Harvester ==> Occupy spot on mine to collect
    2 referencias
    private void OccupySpot(int index)...
    /// <summary> Harvester ==> Free spot in mine to let another harvester collect i ...
    1 referencia
    private void FreeSpot(int index)...

}
```

Implementación

Complicaciones

Clase

Las partes más complicadas fueron las que vimos en clase acerca de la matriz de distancias entre entidades en la escena.

Fue complicado plantear la lógica de la gestión de distancias, su funcionamiento hasta darle varias vueltas.

Recolectores

De la parte restante que quedó cara a la entrega, con las unidades recolectoras no hubo mayores complicaciones más allá de como plantear en un inicio el esquema de lógica con el que funcionaría.

Su lógica era muy sencilla y las bases al realizar el sistema base de jerarquía heredando estados y métodos concretos fue rápido y sencillo, no hubo que aplicar mucho nuevo código.

Aspectos de mayor interés

El esquema de funcionamiento de las unidades y como está planteado con jerarquía y herencias para organizar mejor el juego.

El sistema de selección de personajes mediante un plano de un rectángulo en un canvas que cambia según mueves el ratón. Fue interesante la manera en que hubo que plantear su funcionamiento para seleccionar varias unidades mediante Unity

Conclusiones

Práctica sencilla al haber visto muchas partes complicadas del mismo en clase.

La programación de la matriz de distancias y del sistema de escoger unidades me parecieron curiosos por debajo en cuanto a funcionamiento y algo complicados de primeras sin tener ninguna idea.

Me parece que un RTS es un juego bastante más complejo y llevadero de lo que parece por diversos puntos, y menos mal que no nos metimos con diseño para balancear nada.