CODI I DISSENY

1. JUGADOR

```
Frequently called | 2 usages | Héctor Lara García +4 *
public IEnumerator InteractRaycast()
{
    while (true)
    {
        Debug.DrawRay( start: _Camera.transform.position, dir: _Camera.transform.forward, Color. ■ magenta, duration: 5f);
        if (Physics.Raycast( origin: _Camera.transform.position, direction: _Camera.transform.forward, out RaycastHit hit, maxDistance: 5f, interactLayerMask))
        {
            if (hit.transform.TryGetComponent<IInteractuable>(out IInteractuable interactuable))
            {
                if (interactuable.isInteractuable)
                {
                    //Traiem el material de l'objecte interactuable si es remarkable i és un objecte diferent amb el que vam interactuar
                    if (interactiveGameObject != null && !interactiveGameObject.Equals(hit.transform.gameObject) && interactuable.isRemarkable && interactiveGameObject.GetComponent<MeshRenderer>() != null)
                    {
                        interactiveGameObject.GetComponent<MeshRenderer>().materials = new Material[] { interactiveGameObject.GetComponent<MeshRenderer>().materials[0] };
                        interactiveGameObject = null;
                    }
                    //Guardem una referència a l'objecte interactuable
                    interactiveGameObject = hit.transform.gameObject;
                    //Si l'objecte interactuable és remarkable, li afegim el material
                    if (interactuable.isRemarkable)
                    {
                        interactiveGameObject.GetComponent<MeshRenderer>().materials = new Material[] { interactiveGameObject.GetComponent<MeshRenderer>().materials[0] };
                        baseMaterial = interactiveGameObject.GetComponent<MeshRenderer>().materials[0];
                        interactiveGameObject.GetComponent<MeshRenderer>().materials = new Material[]
                        {
                        interactiveGameObject.GetComponent<MeshRenderer>().materials[0],

                        material
                        };
                    }
                    OnInteractuable?.Invoke(interactiveGameObject);
                }
            }
        }
```

Aquesta corutina és la que el *Player* fa servir per a poder interactuar amb el món. Si té un objecte interactuable al davant ens guardem la referència a aquell objecte i si és *remarkable* modifiquem els seus materials per tal d'afegir i treure l'*outliner*.

```
//Si no hi ha cap objecte interactuable a la vista, netegem la referència i invoquem l'event OnNotInteractuable
else if (!Physics.Raycast( origin: _Camera.transform.position, direction: _Camera.transform.forward, out RaycastHit hit2, maxDistance: 5f, interactLayerMask))
{
    if (interactiveGameObject != null)
    {
        if (interactiveGameObject.GetComponent<IInteractuable>().isRemarkable)
        {
            interactiveGameObject.GetComponent<MeshRenderer>().materials = new Material[] { interactiveGameObject.GetComponent<MeshRenderer>().materials[0] };
            interactiveGameObject = null;
        }
        else
        {
            interactiveGameObject = null;
        }
    }
    OnNotInteractuable?.Invoke();
}
yield return new WaitForSeconds(0.1f);
```

Si no trobem res interactuable al davant i l'últim objecte amb el qual hem interactuat és *remarkable* li traiem el material i posem l'objecte interactuable a null.

```
private void Interact(InputAction.CallbackContext context)
{
    //Si el jugador està interactuant amb un objecte
    if (interactiveGameObject != null)
    {
        //Si és una porta cridem a l'Interact de la porta, ja que és diferent (necessita la posició del jugador per obrir-se).
        if (interactiveGameObject.GetComponent<IInteractuable>() is InteractuableDoor)
        {
            if (Physics.Raycast( origin: _Camera.transform.position,  direction: _Camera.transform.forward, out RaycastHit hit,  maxDistance: 5f,  interactLayerMask))
            {
                if (hit.collider.TryGetComponent<InteractuableDoor>(out InteractuableDoor door))
                {
                    door.SoundPlayer();
                    door.Interact(this.transform);
                }
            }
        }
        //Si és un llibre, cridem a l'Interact del llibre, ja que necessita l'objecte equipat per interactuar.
        else if (interactiveGameObject.GetComponent<IInteractuable>() is InteractuableCellBook)
        {
            if (equipedObject != null)
            {
                if (equipedObject.GetComponent<PickItem>().item is BookItem)
                {
                    interactiveGameObject.GetComponent<InteractuableCellBook>().Interact(equipedObject.gameObject);
                }
            }
        }
        else
        {
            interactiveGameObject.GetComponent<IInteractuable>().Interact();
        }
        //Si l'objecte interactuable és remarkable, li tornem a posar el material base per tal que no es vegi l'outliner quan es fa el puzle per exemple.
        if (interactiveGameObject.GetComponent<IInteractuable>().isRemarkable)
            interactiveGameObject.GetComponent<MeshRenderer>().materials = new Material[] { interactiveGameObject.GetComponent<MeshRenderer>().materials[0] };
        //Netegem la referència de l'objecte interactuable per tal que no es pugui tornar a interactuar fins que no es torni a fer un raycast.
        if (interactiveGameObject != null)
            interactiveGameObject = null;
        {
            OnNotInteractuable?.Invoke();
        }
    }
}
```

Quan el jugador està interactuant amb alguna cosa i pressiona la E, aquesta funció s'executa. Bàsicament el que es fa es cridar a la funció Interact de l'objecte i ell ja s'encarrega de la lògica.

2. PUZLES

2.1. PUZLE MORSE

```
//El dictionari de cada lletra traduida a morse
private Dictionary<char, string> morseCode = new Dictionary<char, string>()
{
    {'A', ".-"}, {'D', "-.."},
    {'E', "."},   {'I', ".."},  {'P', ".--."},
    {'R', ".-."},   {'S', "..."},  {'T', "-"},
    {' ', " "}
};
```

Aquest és el diccionari que la llum fa servir per a fer la seva funció. Només vam posar en aquest cas les paraules que ens interessen, tot i que es podrien posar totes i no canviaria res.

```
private IEnumerator PlayMorse(string message)
{
    while (true)
    {
        morseLight.enabled = false;
        yield return new WaitForSeconds(5f);
        Debug.Log("Empiezo morse");

        foreach (char c in message)
        {
            //agafem el codi en morse segons el caracter
            string code = morseCode[c];

            if (code == " ")
            {
                yield return new WaitForSeconds(letterSpace); //espai entre paraula
                continue;
            }
            //Recorrem cada caracter del morse
            foreach (char symbol in code)
            {
                morseLight.enabled = true;

                if (symbol == '.')
                {
                    yield return new WaitForSeconds(dotDuration);
                }
                else if (symbol == '-')
                {
                    yield return new WaitForSeconds(dashDuration);
                }

                morseLight.enabled = false;

                yield return new WaitForSeconds(spaceDuration); //espai entre simbols
            }

            yield return new WaitForSeconds(letterSpace); //espai entre lletres
        }
```

Aquesta és la funció que fa que la llum parpellegi en Morse segons la paraula donada. Com es pot veure és una corutina i va recorrent caràcter a caràcter de la paraula, agafa el caràcter traduït a Morse gràcies al diccionari i recorre cada símbol per fer encendre's i apagar-se segons les pauses pertinents.

```
private Dictionary<string, char> morseCode = new Dictionary<string, char>()
{
    {".-", 'A'},    {"-...", 'B'},  {"-.-.", 'C'},  {"-..", 'D'},
    {".", 'E'},     {"..-.", 'F'},  {"--.", 'G'},   {"....", 'H'},
    {"..", 'I'},    {".---", 'J'},  {"-.-", 'K'},   {".-..", 'L'},
    {"--", 'M'},    {"-.", 'N'},    {"---", 'O'},   {".--.", 'P'},
    {"--.-", 'Q'},  {".-.", 'R'},   {"...", 'S'},   {"-", 'T'},
    {"..-", 'U'},   {"...-", 'V'},  {".--", 'W'},   {"-..-", 'X'},
    {"-.--", 'Y'},  {"--..", 'Z'}

};
```

Per al panell on s'ha d'introduir la combinació també tenim un diccionari de Morse, però al revés i amb totes les lletres.

```
public void AddInput(string input)
    switch (input)
        case "enter":
            CheckCombo();
            break;
        default:
            //Si la paraula actual és més llarga que 14 caràcters, no acceptem més entrades
            if (currentInput != null && currentInput.Length ==14)
            {
                return;
            }
            //Si l'entrada és un punt o un guió, l'afegim al caràcter morse actual
            if (input == "." || input == "-")
            {
                currentMorseCharacter += input;
            }
            //Si l'entrada és "endLetter", comprovem si el caràcter morse actual és al diccionari i si ho és, afegim la lletra corresponent a l'entrada actual. Si no, afegim un "?".
            else if (input == "endLetter")
            {
                if (morseCode.ContainsKey(currentMorseCharacter))
                {
                    currentInput += morseCode[currentMorseCharacter];
                }
                else if (currentMorseCharacter != "")
                {
                    currentInput += "?";
                }

                currentMorseCharacter = "";
            }
            //Si l'entrada és un espai, l'afegim a l'entrada actual
            else if (input == "/")
            {
                currentInput += " ";
            }
            //Si l'entrada és "delete", eliminem l'últim caràcter de l'entrada actual
            else if (input == "delete")
            {
                char[] aux= currentInput.ToCharArray();
                currentInput = "";
                for (int i = 0; i < aux.Length; i++)
                {
                    if (i < aux.Length - 1)
                    {
                        currentInput += aux[i];
                    }
                }
            }
            keypadDisplayText.text = currentInput;
            break;
```

Aquesta funció s'executa quan el jugador pressiona un dels botons del panell. Fa una cosa diferent dependent del *input* que li arriba sempre i quan la frase actual sigui més curta que 14 caràcters.

2.2. PUZLE AMB IA

```
public Texture2D CaptureDrawing()
{
    RenderTexture currentRT = RenderTexture.active;
    RenderTexture.active = renderTexture;

    Texture2D image = new Texture2D(renderTexture.width, renderTexture.height, TextureFormat.RGBA32, mipChain: false);
    image.ReadPixels(new Rect(0, 0, renderTexture.width, renderTexture.height), destX: 0, destY: 0);
    float threshold = 0.05f; //Posem aquest threshold per tal que reconeixi les linies i no el fons gris

    for (int x = 0; x < image.width; x++)
    {
        for (int y = 0; y < image.height; y++)
        {
            Color pixel = image.GetPixel(x, y);

            //Calculem la brillantor del pixel i mirem si es mes petit que el threshold, per tant el considerem negre
            float luminance = 0.2126f * pixel.r + 0.7152f * pixel.g + 0.0722f * pixel.b;

            //Si es obscur el pixel, el considerem negre si no blanc
            image.SetPixel(x, y, luminance < threshold ? Color.black : Color.white);
        }
    }
    image.Apply();

    RenderTexture.active = currentRT;
    byte[] bytes = image.EncodeToPNG();
    File.WriteAllBytes(Application.persistentDataPath + "/Captured.png", bytes);
    Classify(image);
    return image;
}
```

Aquesta funció agafa la *RenderTexture*, la qual captura el dibuix que el jugador ha fet amb la pissarra, i creem una textura 2D a partir d'aquesta. Vam tenir un problema amb la IA, ja que a la RenderTexture li afectava la llum i la manca d'aquesta. Per aquest motiu hem de recórrer la textura i els píxels que no siguin negres els pintem com a blanc. Per fer això vam haver de fer un *threshold*, ja que si no detectava píxels que no volíem com a negre. D'aquesta manera només és blanc el fons i el dibuix és veu perfectament.

```
public void Classify(Texture2D inputImage)
{

    Tensor input = new Tensor(inputImage, channels: 3); // 3 = RGB
    worker.Execute(input);

    Tensor output = worker.PeekOutput();
    float[] scores = output.ToReadOnlyArray();

    for (int i = 0; i < scores.Length; i++)
    {
        Debug.Log($"Clase {i+1}: {scores[i]}");
    }
```

Aquesta funció és la que fa que la nostra IA s'executi i reconegui la imatge passada per paràmetre.

3. WAVE EFFECT

```
void Draw()
{
    //Step és la freqüència entre l'amplada de la textura (points és amplada).
    float step = frequency / (float)points;
    if (texture != null)
        texture.SetPixels(colors);

    //Recorrem l'amplada de la textura i per cada punt calculem el valor de la funció de Perlin.
    for (int currentPoint = 0; currentPoint < points; currentPoint++)
    {
        float sample = PerlinNoiseOctaves(currentPoint, 5, step, (int)movementSpeed);
        //Escalem l'amplitud segons la freqüència i l'amplitud màxima (Això es va fer per als canvis sobtats com trets). Vam posar 8 perquè quedés bé, més gran pot ser massa.
        float currentAmplitude = (Mathf.Clamp01(frequency / 8f) * amplitude);
        //Centrem l'ona verticalment a la texura
        sample = ((sample - .5f) * currentAmplitude * 0.5f) + amplitude * 0.5f;
        //Ajustem el valor de la mostra perquè no surti de l'amplitud màxima.
        int y = sample < 0 ? 1 : sample > amplitude ? amplitude-1 : (int) sample;

        //Pintem tres pixels verticals per cada punt de la textura per fer l'ona més visible, més gruixuda.
        texture.SetPixel(currentPoint, y-1, color);
        texture.SetPixel(currentPoint, y, color);
        texture.SetPixel(currentPoint, y+1, color);
    }
    if (texture != null)
        texture.Apply();
}
```

Aquesta funció és la que crea l'ona de soroll gràcies a un so de *Perlin.*

```
private float PerlinNoiseOctaves(float x, float y, float step, int octaves = 0)
{
    //Això fa que l'ona és mogui horitzontalment amb el temps.
    float xCoord = Time.timeSinceLevelLoad + x * step;
    //El valor de la y no es toca
    float yCoord = y;
    //Fem la mostra de Perlin amb les coordenades x i y.
    float result = Mathf.PerlinNoise(xCoord, yCoord);

    //Per cada octava augmentem el valor de la freqüència i afegim una mostra de Perlin amb un valor més baix.
    for (int octave = 1; octave < octaves; octave++)
    {
        //Augmentem la freqüència de l'octava, per exemple, si l'octava és 1, la freqüència serà el doble de la inicial.
        float newStep = step * 2 * octave;
        //Tornem a calcular les coordenades x i y per a l'octava.
        float xOctaveCoord = Time.timeSinceLevelLoad + x * newStep;
        float yOctaveCoord = y;
        //Tornem a fer un perlin amb les noves coordenades
        float octaveSample = Mathf.PerlinNoise(xOctaveCoord, yOctaveCoord);
        //Augmentem la freqüència de l'octava i es redueix l'amplitud del perlin a cada octava.
        octaveSample = (octaveSample - .5f ) * (.8f / octave);

        result += octaveSample;
    }

    return Mathf.Clamp01(result);
}
```

A més del *Perlin* afegim octaves per tal que l'efecte de l'onada es noti més quan hi ha més so.

```
private IEnumerator ImpactCoroutine(int noiseIntensity)
{
    currentTargetMovementSpeed = this.targetMovementSpeed;
    currentTargetFrequency = this.targetFrequency;
    currentSmoothTime = this.smoothTime;

    this.smoothTime = 0.1f;
    this.targetMovementSpeed = noiseIntensity;
    this.targetFrequency = noiseIntensity;
    yield return new WaitForSeconds(0.5f);
    this.targetFrequency = currentTargetFrequency;
    if (targetFrequency < 1) targetFrequency = 1;
    this.targetMovementSpeed = currentTargetMovementSpeed;
}
```

Aquesta funció es crida quan es fa un cop de so (fer un tret, llençar un objecte...). El que fa és augmentar els valors de l'ona durant 0.5 segons per a, seguidament, tornar a l'estat anterior.

4. Enemics

4.1. Enemic cec

```
// Funcio per moure l'enemic pel mapa
IEnumerator Patrol()
{
    Vector3 point = Vector3.zero;
    while (true)
    {
        if (!_Patrolling)
        {
            if (!_Search)
            {
                while (true)
                {
                    //Aquest monstre pot anar entre dues parts del mapa, però una d'aquestes
                    //està tancada fins a superar el puzle dels jeroglífics,
                    //per això la comprovació de la porta.
                    if (_DoorLocked.isLocked)
                    {
                        point = _WaypointsFirstPart[Random.Range(0, _WaypointsFirstPart.Count)].transform.position;
                    }
                    else
                    {
                        int random = Random.Range(0, 3);

                        switch (random)
                        {
                            case 0:
                                point = _WaypointsFirstPart[Random.Range(0, _WaypointsFirstPart.Count)].transform.position;
                                break;
                            case 1:
                            case 2:
                                point = _WaypointsSecondPart[Random.Range(0, _WaypointsSecondPart.Count)].transform.position;
                                break;
                        }
                    }
                    if (point != _NavMeshAgent.destination)
                        break;
                }
            }
            else
            {
                //Aquesta comprovació és per evitar que si ha sentit un so
                //molt fort elimini el SetDestination() cap al punt
                if(_RangeSearchSound > 0)
                {
                    RandomPoint(_SoundPos, _RangeSearchSound, out Vector3 coord);
                    point = coord;
                }
            }
            _Animator.enabled = true;
            _Patrolling = true;
            _NavMeshAgent.SetDestination(point);
        }

        if (!_NavMeshAgent.pathPending && _NavMeshAgent.remainingDistance <= _NavMeshAgent.stoppingDistance)
        {
            //Si l'enemic està investigant el so (el booleà _Search) i no s'ha activat la corutina
            //específica l'activem. Aquesta corutina és la que farà canviar al monstre d'estat perquè
            //torni a patrullar pels seus waypoints.
            if (_Search && _CurrentState != EnemyStates.ATTACK && _ChangeStateToPatrol == null)
            {
                Debug.Log("Activo la corutina WakeUp");
                _ChangeStateToPatrol = StartCoroutine(WakeUp(10));
            }
            _Animator.enabled = false;
            _Patrolling = false;
            yield return new WaitForSeconds(1);
        }
        else
            yield return new WaitForSeconds(0.5f);
    }
}
```

La funció de patrulla de l'enemic cec s'utilitza tant per patrullar pels *waypoints* especificats en el mapa com per voltar al voltant del punt de so si el monstre ha escoltat un so. Els *waypoints* estan dividits en dues parts, la primera part és la part del mapa on apareix i on està localitzat el puzle dels jeroglífics. Quan el jugador resol aquest puzle la porta s'obre i permet al monstre anar a la nova zona amb els nous *waypoints*.

```
//Funcio que criden tant els objectes llençables com el jugador
//per notificar als enemics que han fet un so, i que aquests actuin acorde
public override void ListenSound(Vector3 pos, int lvlSound)
{
    bool wall = false;
    _SoundPos = pos;
    //Fem un raycast des de la posicio de l'enemic fins a l'origen del so i recollim tots els objectes que hem tocat en el cami
    RaycastHit[] hits = Physics.RaycastAll(transform.position, _SoundPos - transform.position, Vector3.Distance(_SoundPos,
transform.position));

    //Per cada objecte que hem tocat mirem si són objectes que atenuen el so
    //Si ho són reduïm el valor del so rebut per paràmetre pel valor que ens dona l'objecte.
    foreach (RaycastHit hit in hits)
    {
        if (hit.collider.TryGetComponent<IAttenuable>(out IAttenuable a))
        {
            lvlSound = a.AttenuateSound(lvlSound);
            wall = true;
        }
    }

    //Distància del cec fins al punt del so
    float dist = Vector3.Distance(transform.position, _SoundPos);
    Debug.Log($"Distància entre cec i punt de so: {dist}");

    //Si la distància entre l'origen del so i el monstre és menor a 10 i el monstre té visió directa amb el jugador,
    //el monstre passarà al valor més agressiu de resposta al so.
    if (dist < 10 && Physics.Raycast(transform.position, (_Player.transform.position - transform.position), out RaycastHit info, dist,
_LayerObjectsAndPlayer))
    {
        if (info.collider.TryGetComponent<Player>(out _))
        {
            lvlSound = 8;
        }
        else
        {
            lvlSound = 3;
        }
    }

    Debug.Log($"Nivell so: {lvlSound}");

    //Llista de resposta al so per part del monstre. Com més fort és el nivell de so
    //més a prop d'aquest va, fins que si és molt alt el monstre atacarà.
    if (lvlSound > 0 && _CurrentState == EnemyStates.PATROL)
    {
        if (lvlSound > 0 && lvlSound <= 2)
        {
            _RangeSearchSound = 1.5f;
            _Search = true;
        }
        else if (lvlSound > 2 && lvlSound <= 3)
        {
            _RangeSearchSound = 1;
            _Search = true;
        }
        else if (lvlSound > 3 && lvlSound <= 5)
        {
            _RangeSearchSound = 0.5f;
            _Search = true;
        }
        else if (lvlSound > 5)
        {
            //Si la distància calculada anteriorment està entre 1.5 i 8, no té cap paret davant i pot saltar,
            //saltarà cap a la font de so, independentment de si aquesta és produïda pel jugador o per un objecte.
            if (dist > 1.5f && dist <= 8 && !wall && !_Jumping)
            {
                Debug.Log("Faig salt!");
                _Jumping = true;
                ChangeState(EnemyStates.ATTACK);
                Vector3 start = transform.position;
                Vector3 end = _SoundPos;

                _Rigidbody.AddForce((end - start) * 140);
                OnJump?.Invoke();

                //No necessiten comprovació ja que només entrarà aquí quan pugui tornar a saltar,
                //que es recupera quan acaba la corutina de RecoverJump
                _BlindAudioSource.PlayOneShot(_blindAttackAudioClip);
                StartCoroutine(RecoverAgent());
                StartCoroutine(RecoverJump());
            }
            //Si la distància és massa gran només anirà fins al punt d'orígen del so.
            else if (Vector3.Distance(_SoundPos, transform.position) > 8)
            {
                _RangeSearchSound = 0;
                _Search = true;
                _NavMeshAgent.SetDestination(_SoundPos);
                Debug.Log($"Soc l'enemic cec i vaig al punt {_NavMeshAgent.destination}");
            }
        }

        if(lvlSound <= 5)
            ChangeState(EnemyStates.PATROL);
    }
}
```

Aquesta funció és la que criden tant el jugador com els objectes llençables quan realitzen un so. L'enemic crea un *raycastall* des de la seva posició fins a la d'origen del so, agafant tots els objectes. Després els parseja mirant si cap d'aquests atenua el so, i si ho fan es redueix el so que es passa per paràmetre. Una vegada reduït el so calcula la distància entre ell i el punt de so i si aquest és menor a 10 i no té cap paret davant canvia el so calculat anteriorment per un valor que supera el llindar màxim del monstre, sinó posa un valor entremig.

Aquest valor després el comprova amb uns paràmetres per reaccionar d'acord amb aquests. Com és fort és més precís serà el monstre a buscar al que ha provocat aquest so. Si el volum és extremadament fort i la distància és idònia salta cap a la font de so, atacant el punt, tant si no hi ha ningú com si hi està el jugador.

4.2. Enemic gras

```
private void ActivateChaseCoroutine()
{
    if(_CurrentState != EnemyStates.KNOCKED)
        ChangeState(EnemyStates.CHASE);
}

private void DeactivateChaseCoroutine()
{
    _Search = false;
    ChangeState(EnemyStates.PATROL);
}

private void ActivateAttack()
{
    if (_CurrentState != EnemyStates.KNOCKED)
    {
        //Si el jugador està davant de l'enemic quan entra dins de l'àrea d'atac (i aquest no està noquejat) l'atacarà.
        Physics.Raycast(transform.position, (_Player.transform.position - transform.position), out RaycastHit thing, 12,
            _LayerObjectsAndPlayer);
        if(thing.transform.gameObject.TryGetComponent(out Player player))
        {
            _Animator.SetBool("Chase", false);
            ChangeState(EnemyStates.ATTACK);
        }
    }
}

private void DeactivateAttack()
{
    _Animator.SetBool("Chase", true);
    if(_ChangeToChase == null)
        StartCoroutine(ChangeToChase());
}

IEnumerator ChangeToChase()
{
    while(true)
    {
        //Esperem que el monstre acabi l'animació d'atac per poder canviar d'estat
        if (!_Animator.GetCurrentAnimatorStateInfo(0).IsName("Attack"))
        {
            //Si aquest està a prop del jugador el perseguirà, si no passarà a patrullar.
            if(Vector3.Distance(transform.position, _Player.transform.position) <= 3.5f)
                ChangeState(EnemyStates.CHASE)
            else
                ChangeState(EnemyStates.PATROL);

            break;
        }

        yield return new WaitForSeconds(0.1f);
    }
}
```

Quan el jugador entra dins de l'àrea de persecució del monstre gras aquest començarà a perseguir al jugador, i si tot seguit entra a l'àrea d'atac canviarà l'estat a atacar. Si el

jugador en cap moment surt d'aquestes àrees canvia els estats menys quan està atacant, que s'espera a que acabi l'animació per canviar d'estat, comprovant on està el jugador per canviar a l'estat adient (si està molt a prop a *CHASE* si no a *PATROL*).

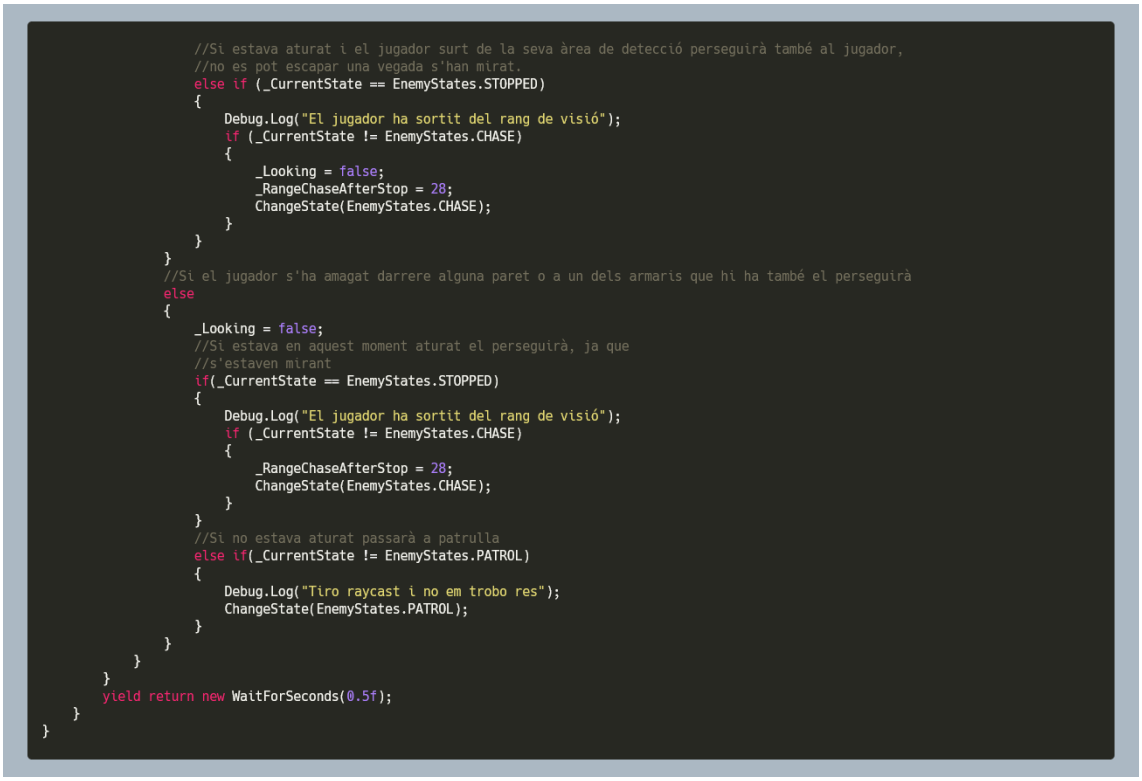
4.3. Enemic ràpid

```
//Corutina que realitza tot el càlcul per saber si el jugador i l'enemic s'estan mirant i actuar
IEnumerator LookingPlayer()
{
    float alphaLook = 0;
    while (true)
    {
        if(_CurrentState != EnemyStates.KNOCKED)
        {
            //Primer mirem si el jugador està dins de l'àrea per actuar.
            Collider[] aux = Physics.OverlapSphere(transform.position, _RangeChaseAfterStop, _LayerPlayer);
            if (aux.Length > 0)
            {
                //Això és el càlcul de les direccions de les mirades, si s'estan mirant l'un a l'altre serà -1 i anira augmentant
                //fins 1 quan estan mirant cap a la mateixa direcció
                alphaLook = Vector3.Dot(transform.forward, _Player.transform.forward);

                //Raycast amb les layers de paret i player i si tenim la paret no seguim, si no seguim el jugador
                //El raycast és de distància 12 per a tenir un marge amb el trigger de l'esfera de deteccio, perquè no es torni boig
                Debug.DrawRay(transform.position, (_Player.transform.position - transform.position), Color.blue, 4);
                if (Physics.Raycast(transform.position, (_Player.transform.position - transform.position), out RaycastHit info, 12,
_LayerObjectsAndPlayer))
                {
                    if (info.transform.tag == "Player")
                    {
                        if (_FirstTime)
                        {
                            _FastAudioSource.PlayOneShot(_fastShoutAudioClip, 6.5f);
                            _FirstTime = false;
                        }
                        if (!_Looking)
                        {
                            transform.LookAt(new Vector3(info.transform.position.x, 0.9f, info.transform.position.z));
                            _Looking = true;
                        }

                        //Si el càlcul del Vector3.Dot ens dona que l'enemic i el jugador s'estan mirant
                        //l'enemic passarà a l'estat d'aturat, on no es mourà, només mirarà al jugador constantment.
                        if (alphaLook < -0.8f)
                        {
                            if (_CurrentState != EnemyStates.STOPPED)
                            {
                                _FastAudioSource.resource = _fastAudioClip;
                                _FastAudioSource.Play();
                                _NavMeshAgent.SetDestination(transform.position);
                                ChangeState(EnemyStates.STOPPED);
                            }
                        }
                        else
                        {
                            //Tornem a calcular l'alphaLook per si el jugador encara ens segueix mirant
                            transform.LookAt(new Vector3(info.transform.position.x, 0.9f, info.transform.position.z));
                            alphaLook = Vector3.Dot(transform.forward, _Player.transform.forward);

                            //Si després de tornar a calcular el Vector3.Dot l'enemic i el jugador no s'estan mirant
                            //canviarem l'estat de l'enemic per actuar contra el jugador
                            if (alphaLook >= -0.8f)
                            {
                                //Si el jugador està lluny canviarà a chase i el perseguirà
                                if (Vector3.Distance(info.transform.position, transform.position) > 2)
                                {
                                    Debug.Log("Veig al jugador lluny");
                                    if (_CurrentState != EnemyStates.CHASE)
                                    {
                                        _FastAudioSource.resource = _fastRunAudioClip;
                                        _FastAudioSource.Play();
                                        _RangeChaseAfterStop = 28;
                                        ChangeState(EnemyStates.CHASE);
                                    }
                                }
                                //Si es troba prou a prop l'atacarà directament
                                else if (Vector3.Distance(info.transform.position, transform.position) <= 2)
                                {
                                    Physics.Raycast(transform.position,
                                        (_Player.transform.position - transform.position), out RaycastHit thing, 12,
                                        _LayerObjectsAndPlayer);
                                    if (thing.transform.CompareTag("Player"))
                                    {
                                        Debug.Log("Tinc al jugador al davant!");
                                        _NavMeshAgent.SetDestination(transform.position);
                                        if (_CurrentState != EnemyStates.ATTACK)
                                        {
                                            ChangeState(EnemyStates.ATTACK);
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
```



```
                //Si estava aturat i el jugador surt de la seva àrea de detecció perseguirà també al jugador,
                //no es pot escapar una vegada s'han mirat.
                else if (_CurrentState == EnemyStates.STOPPED)
                {
                    Debug.Log("El jugador ha sortit del rang de visió");
                    if (_CurrentState != EnemyStates.CHASE)
                    {
                        _Looking = false;
                        _RangeChaseAfterStop = 28;
                        ChangeState(EnemyStates.CHASE);
                    }
                }
            }
            //Si el jugador s'ha amagat darrere alguna paret o a un dels armaris que hi ha també el perseguirà
            else
            {
                _Looking = false;
                //Si estava en aquest moment aturat el perseguirà, ja que
                //s'estaven mirant
                if(_CurrentState == EnemyStates.STOPPED)
                {
                    Debug.Log("El jugador ha sortit del rang de visió");
                    if (_CurrentState != EnemyStates.CHASE)
                    {
                        _RangeChaseAfterStop = 28;
                        ChangeState(EnemyStates.CHASE);
                    }
                }
                //Si no estava aturat passarà a patrulla
                else if(_CurrentState != EnemyStates.PATROL)
                {
                    Debug.Log("Tiro raycast i no em trobo res");
                    ChangeState(EnemyStates.PATROL);
                }
            }
        }
    }
    yield return new WaitForSeconds(0.5f);
  }
}
```

Aquesta funció és la més important de l'enemic ràpid, on està tot el processament de les mirades entre l'enemic i el jugador. Primer mirem si el jugador està dins de l'àrea de detecció. Si aquest es troba dins calculem amb un Vector3.Dot si els *forwards* de les mirades estan l'una davant de l'altre per utilitzar-lo més endavant. Fem un *raycast* des de la posició de l'enemic fins a la del jugador i si l'objecte que troba és el jugador i utilitzant el .Dot anterior canviarà l'estat a aturat. Quan aquest .Dot canviï de valor canviarà o a *chase* o a *attack* depenent de la distància a al que estigui el jugador de l'enemic.

5. Guardat

```
//Canviem el tipus de la llista per poder guardar tota la info
List<ItemSlotSave> Inventory = new List<ItemSlotSave>();

foreach (ItemSlot item in _Inventory.items)
{
    if(!_Temp && (item.item is ThrowableItem || item.item is SilencerItem || item.item is HealingItem || item.item is AmmunitionItem) && !
_Delete)
    {
        _Delete = true;
    }
    else
        Inventory.Add(new ItemSlotSave(item.item.id, item.amount, item.stackable));
}

List<ItemSlotSave> ChestInventory = new List<ItemSlotSave>();

foreach (ItemSlot item in _ChestInventory.items)
{
    if (!_Temp && (item.item is ThrowableItem || item.item is SilencerItem || item.item is HealingItem || item.item is AmmunitionItem) &&
!_Delete)
    {
        _Delete = true;
    }
    else
        ChestInventory.Add(new ItemSlotSave(item.item.id, item.amount, item.stackable));
}

List<PickObjectSave> ImportantSpawnsList = new List<PickObjectSave>();

foreach(GameObject item in ImportantSpawns)
{
    ImportantSpawnsList.Add(new PickObjectSave(item.GetComponent<PickObject>().Object.id, item.GetComponent<PickObject>().Id,
item.GetComponent<PickObject>().Picked));
}

List<PickObjectSave> NormalSpawnsList = new List<PickObjectSave>();

foreach (GameObject item in NormalSpawns)
{
    NormalSpawnsList.Add(new PickObjectSave(item.GetComponent<PickObject>().Object.id, item.GetComponent<PickObject>().Id,
item.GetComponent<PickObject>().Picked));
}

List<CellBookSave> CellBooks = new List<CellBookSave>();

foreach(CellBook book in _Cells)
{
    if (book.bookGO != null)
        CellBooks.Add(new CellBookSave(book.bookGO.GetComponent<PickItem>().item.id, book.cellId));
}
```

Per a que el guardat no ens guardés posicions de memòria dels objectes del joc vam crear unes classes amb els mateixos valors que aquestes però amb variables bàsiques.

Aquestes llistes son les conversions de les classes amb variables complexes a unes que sí ho son.

```
[Serializable]
    public class ItemSlotSave
    {
        [SerializeField] public int itemId;

        [SerializeField] public int amount;

        [SerializeField] public bool stackable;

        public ItemSlotSave(int itemId, int amount, bool stackable)
        {
            this.itemId = itemId;
            this.amount = amount;
            this.stackable = stackable;
        }
    }
```

```
[Serializable]
public class ItemSlot
{
    [SerializeField]
    public Item item;
    [SerializeField]
    public int amount;
    [SerializeField]
    public bool stackable;

    public ItemSlot(Item obj, bool stackable)
    {
        item = obj;
        amount = 1;
        this.stackable = stackable;
    }
}
```

Aquests son uns exemples dels canvis de classe, d'una que té una variable Item a una que només guarda la id d'aquesta.