DOCUMENTACIÓN

Clase: Inteligencia Artificial	Práctica: 3 [Máquinas de estados]
Alumno: Miguel Rodríguez Gallego	

Motivos de mi práctica

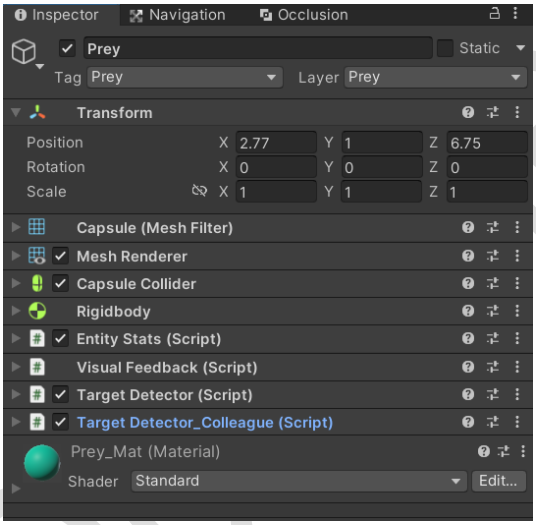
Quise hacer una práctica que llegase a los mínimos de forma adecuada con algún toque de más visual para facilitar la visión de los estados con feedback visual.

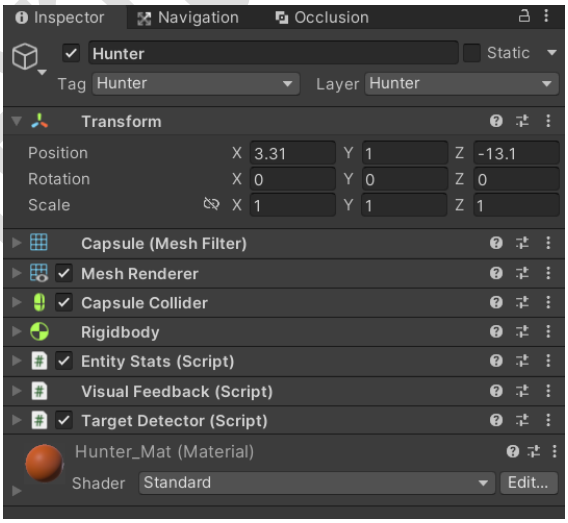
Proceso de creación

Comencé recogiendo los scripts de la anterior práctica de detección de criaturas, imágenes de feedback y demás…

Luego re hice varios scripts de detección y estadísticas de entidad para no tener tantos scripts concretos y que sean así más generalistas, permitiendo trabajar mejor sobre todas las entidades a la hora de referenciar desde la máquina de estados.

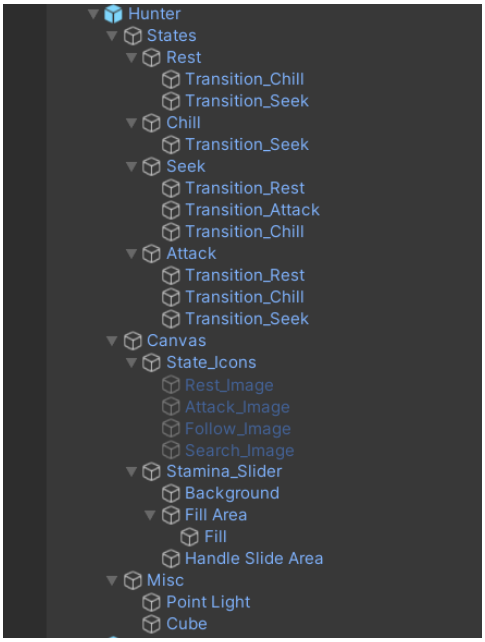
En cuanto a código son todos simples, hay stamina y distancias de detección x colliders como en las anteriores…

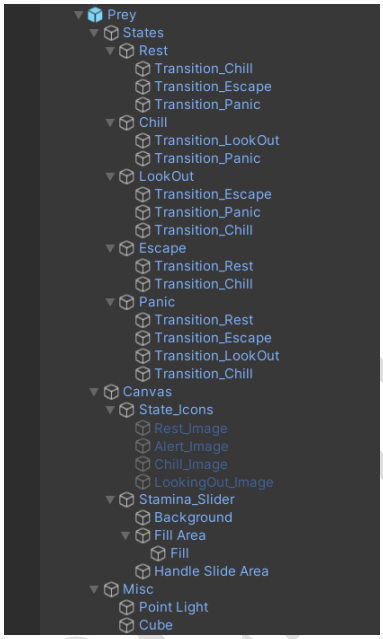




En cuanto a código principal de las entidades son todos simples, hay stamina y distancias de detección x colliders principalmente como en las anteriores prácticas…

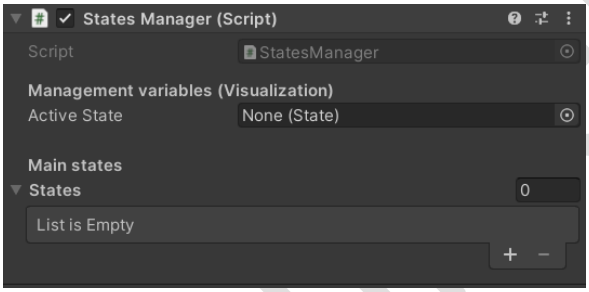
Luego, ya en cuanto a máquinas de estados formé esta organización de gameObjects en el inspector para la presa y el cazador:





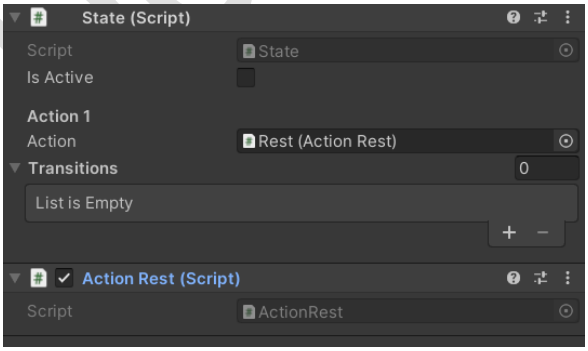
Se puede apreciar como lo organicé dentro de las propias entidades, para poder referenciar fácilmente en el awake todo x código sin volverme loco.

También, dentro de **States**, el cual consta con un State Manager que maneja los estados hijos y sus comprobaciones. Este es el que tiene el Update() que registra la acción y comprueba los cambios que se dan cada frame.

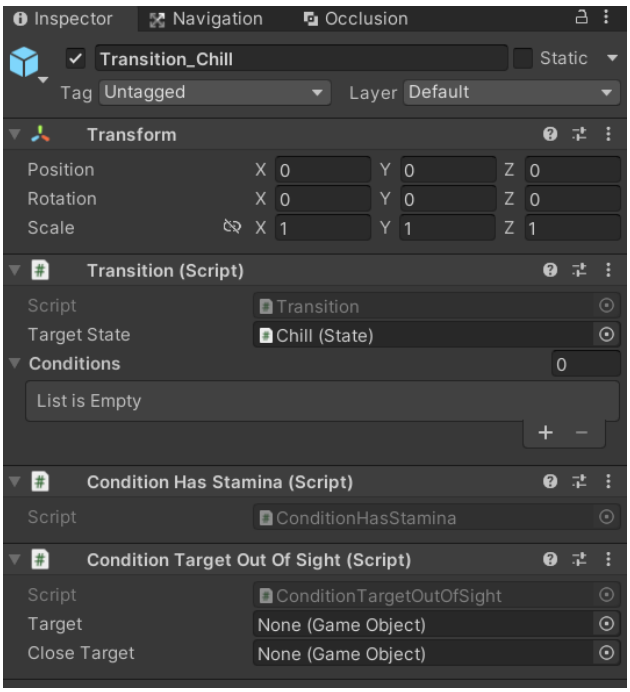


Recalcaré que lo hice escalable y simple, haciéndolo de tal forma que si arrastras un gameobject con las cualidades de estado como hijo del State manager y tiene sus respectivas condiciones y demás, automáticamente se incluye en la lista de estados al arrancar. Así con todos los componentes de esta jerarquía.

Volviendo al funcionamiento, dentro de cada estado se encuentra la clase estado y la acción a ejecutar en sí. Este simplemente contiene información que recoge de los hijos en la jerarquía.



Dentro de cada transición en si encontramos el script de la transición en si que consta de varias condiciones que recoge del mismo componente en el Awake(), además del estado al que quiere ir si se cumplen las condiciones al funcionar.



Entonces visto todo esto, se puede apreciar el funcionamiento de listas de forma jerárquica con sus respectivas funcionalidades. Es una organización simple pero que funciona.

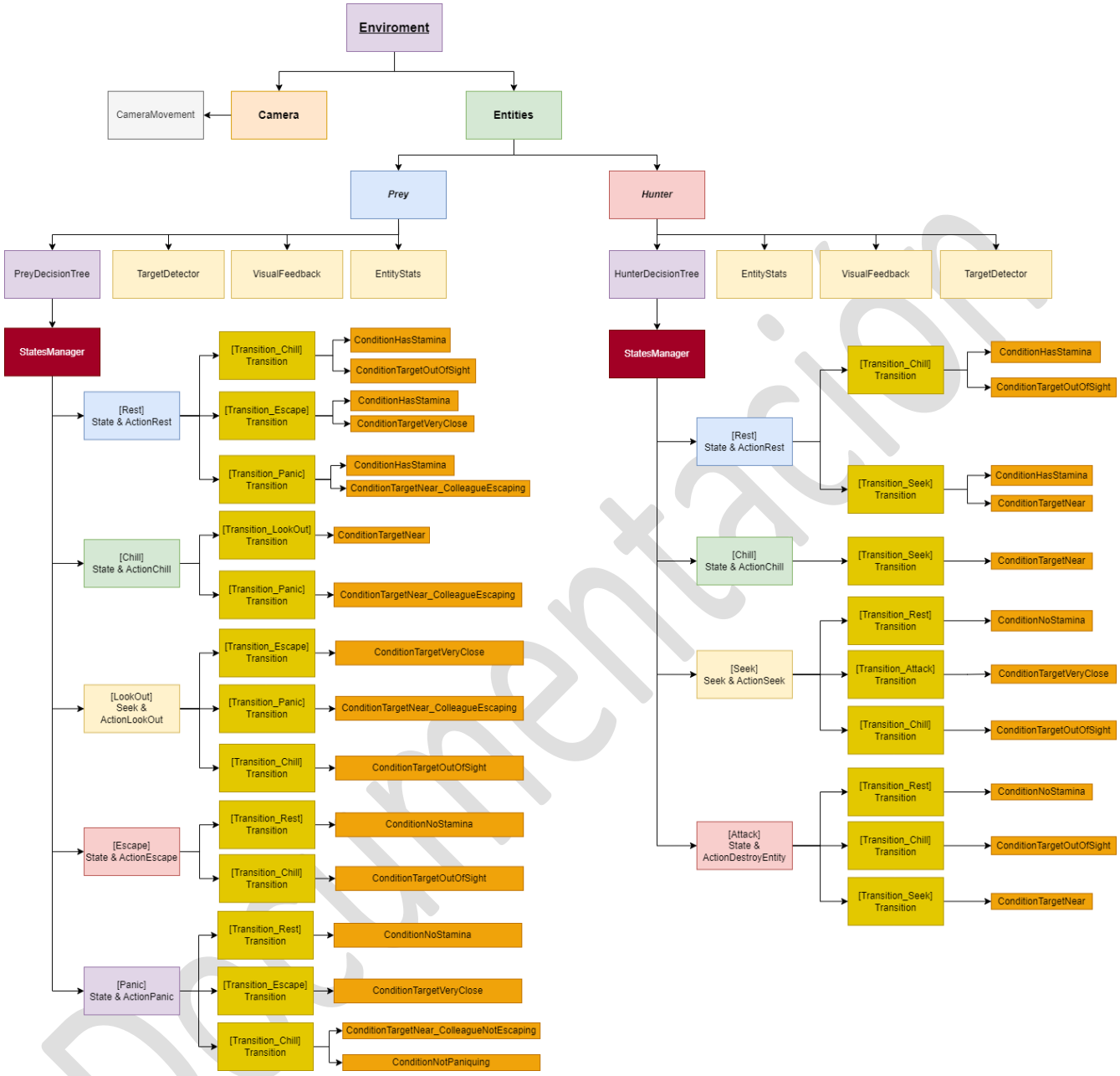
Dificultades y como se resolvieron

La dificultad principalmente fue a la hora de plantear que transiciones tenía que poner en cada estado y las condiciones de cada uno para pasar al otro.

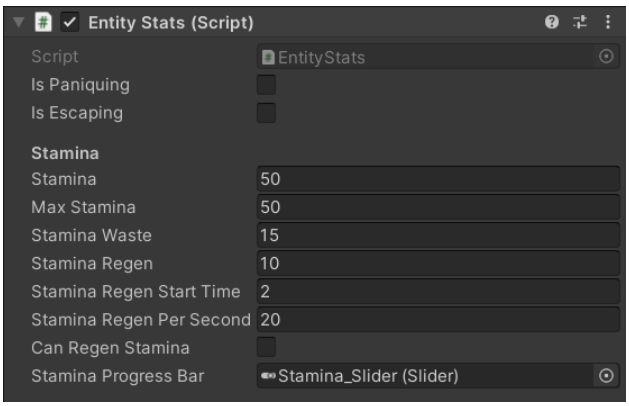
Todo esto lo tuve que solucionar a prueba y error durante bastante tiempo probando las combinaciones que mejor se hacían a la idea, pero tuve que crear más scripts de comprobación de los que creía que iba a necesitar, como el de comprobar si no tiene stamina.

Además, al principio no me di cuenta de que necesitaría hacer más de 1 comprobación en algunas transiciones, x lo que cambiaba continuamente de estado sin parar. Esto fue lo que más tiempo me llevó resolver.

Diagrama de clases



- Camera
  - **CameraMovement**: Poder mover la cámara cuando presionas espacio y tmb pausarla con la misma tecla para que salga el cursor y puedas interactuar con el ambiente.
- Entity management
  - **EntityStats**: Estadísticas de estamina y algunos estados concretos de la entidad.

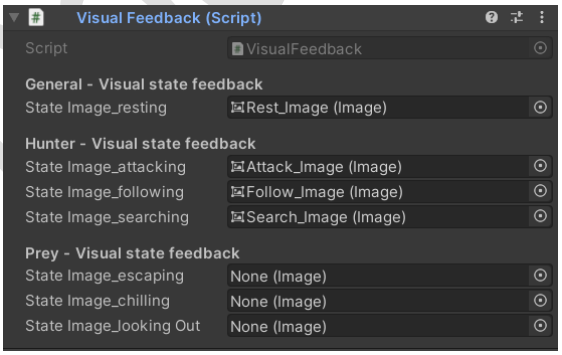



```
1 referencia
void UpdateStamina()
{
    CheckStaminaState();
}
/// <summary> Checks stamina state to make an action or another
2 referencias
void CheckStaminaState()
{
    if (stamina <= 0)
        canRegenStamina = true;

    if (stamina >= maxStamina)
        canRegenStamina = false;
}

/// <summary> Lower stamina automatically with time
3 referencias
public void UseStamina()
{
    stamina -= staminaWaste * Time.deltaTime;
    UpdateStamina_UISlider();
}
/// <summary> Regens stamina automatically with time
1 referencia
public void RegenStamina()
{
    stamina += staminaRegen * Time.deltaTime;
    UpdateStamina_UISlider();
}
/// <summary> HUD Methods to show correctly stamina in slider bar
3 referencias
void UpdateStamina_UISlider()
{
    float fillAmount = stamina / maxStamina;
    staminaProgressBar.value = fillAmount;
}
/// <summary> Get stamina private value
2 referencias
public float GetStamina() => stamina;
/// <summary> Get stamina can regen private value
```

- VisualFeedback: Feedback visual del estado y la estamina.

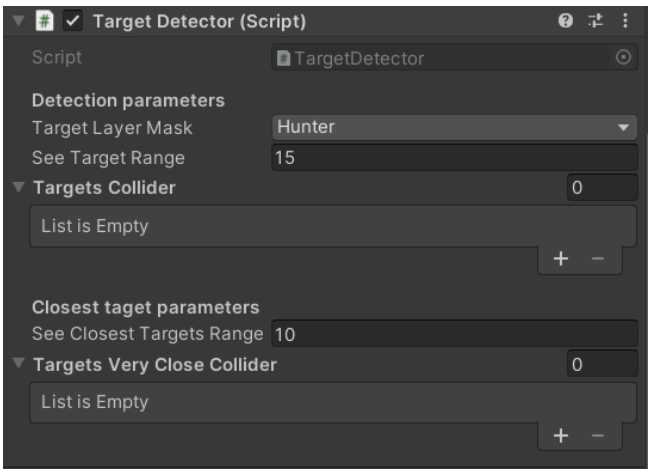



```
public class VisualFeedback : MonoBehaviour
{
    [Header("General - Visual state feedback")]
    public Image stateImage_resting;

    [Header("Hunter - Visual state feedback")]
    public Image stateImage_attacking;
    public Image stateImage_following;
    public Image stateImage_searching;

    [Header("Prey - Visual state feedback")]
    public Image stateImage_escaping;
    public Image stateImage_chilling;
    public Image stateImage_lookingOut;
}
```

- TargetDetector: Detecta las entidades cercanas.



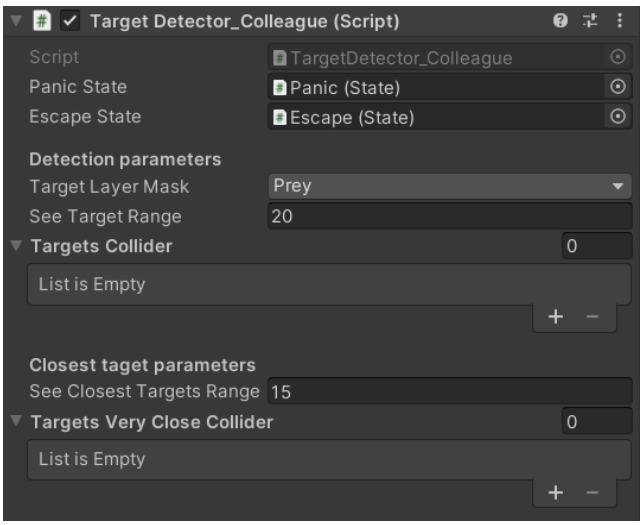
Los métodos que se updatean y registran los targets

```
47      void FindTargets_CanSee()
48      {
49          targetsCollider = Physics.OverlapSphere(transform.position, seeTargetRange, targetLayerMask);
50
51          if (targetsCollider.Length != 0)
52          {
53              closestVisibleTarget = GetClosestEntity(targetsCollider);
54              target = closestVisibleTarget.gameObject;
55              lastPositionSeen = closestVisibleTarget.gameObject.transform.position; // Gets last position of the entitytoZ
56          }
57          else
58          {
59              targetsCollider = null;   // No citizen around in sight
60              target = null;
61          }
62      }
63      /// <summary> When prey is extremely near, detect it and attack it
        1 referencia
66      void FindTargets_CanAttack()
67      {
68          targetsVeryCloseCollider = Physics.OverlapSphere(transform.position, seeClosestTargetsRange, targetLayerMask);
69
70          if (targetsVeryCloseCollider.Length != 0)
71          {
72              closestTarget = GetClosestEntity(targetsVeryCloseCollider);
73              veryCloseTarget = closestTarget.gameObject;
74              lastPositionSeen = closestTarget.gameObject.transform.position; // Gets last position of the entitytoZ
75          }
76          else
77          {
78              targetsVeryCloseCollider = null;   // No citizen around in sight
79              veryCloseTarget = null;
80          }
81      }
```

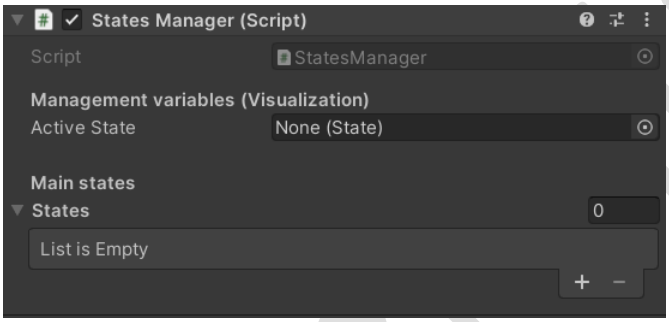
El que coge el target más cercano

```
83      /// <summary>
84      ///     Gets closer entity in checking collider
85      /// </summary>
        2 referencias
86      Transform GetClosestEntity(Collider[] entity)
87      {
88          Collider tMin = null;
89          float minDist = Mathf.Infinity;
90          Vector3 currentPos = transform.position;
91          foreach (Collider t in entity)
92          {
93              float dist = Vector3.Distance(t.transform.position, currentPos);
94              if (dist < minDist)
95              {
96                  tMin = t;
97                  minDist = dist;
98              }
99          }
100         return tMin.transform;
101     }
102 }
```

- TargetDetector_Colleague: Detecta las entidades más cercanas del mismo tipo.

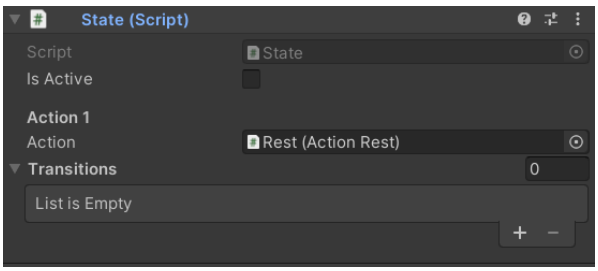


- State
  - StatesManager: Coge todos los estados hijos y los maneja. Tiene el método de las comprobaciones.




```
50      /// <summary>
51      ///     Check conditions in "State" of active state & if it has to transition
52      ///     [Main logic about transitioning]
53      /// </summary>
        1 referencia
54      void CheckConditions()
55      {
56          for (int i = 0; i < activeState.transitions.Count; i++)
57          {
58              int positiveConditionCount = 0;
59
60              for (int x = 0; x < activeState.transitions[i].conditions.Length; x++)
61              {
62                  if (activeState.transitions[i].conditions[x].Test() == true)
63                  {
64                      positiveConditionCount++;
65                  }
66                  if (positiveConditionCount == activeState.transitions[i].conditions.Length)
67                  {
68                      activeState.isActive = false;
69                      activeState = activeState.transitions[i].targetState;
70                      activeState.isActive = true;
71                      return;
72                  }
73              }
74          }
75      }
76  }
```

  - State: Estado en si con sus transiciones que se comprueban a cada frame para ver si cambiar al estado target de cada uno. Solo es un contenedor de información




```
/// <summary>
///     State main info
/// </summary>
Script de Unity (9 referencias de recurso) | 8 referencias
public class State : MonoBehaviour
{
    StatesManager statesManager;

    public bool isActive = false;

    [Header("Action 1")]
    public IAction action;              // Action to make if state is activated
    public List<Transition> transitions;    // Transitions to check

    Mensaje de Unity | 0 referencias
    void Awake()
    {
        statesManager = GetComponentInParent<StatesManager>();
        action = GetComponent<IAction>();
        transitions.AddRange(GetComponentsInChildren<Transition>().ToArray());
    }
}
```

- Actions:
  - IAction: Clase base para las decisiones.
  - ActionChill: Tranquilo por el mapa sin ningún enemigo cerca.

```
public override void Act()
{
    if (entityStats.gameObject.tag == "Hunter")
    {
        visualFeedback.stateImage_attacking.gameObject.SetActive(false);
        visualFeedback.stateImage_following.gameObject.SetActive(false);
        visualFeedback.stateImage_searching.gameObject.SetActive(true);
        visualFeedback.stateImage_resting.gameObject.SetActive(false);
    }
    if (entityStats.gameObject.tag == "Prey")
    {
        entityStats.isEscaping = false;

        visualFeedback.stateImage_escaping.gameObject.SetActive(false);
        visualFeedback.stateImage_chilling.gameObject.SetActive(true);
        visualFeedback.stateImage_lookingOut.gameObject.SetActive(false);
        visualFeedback.stateImage_resting.gameObject.SetActive(false);
    }

    ChillRoutineLogic();
}
/// <summary> Regens stamina automatically with time
1 referencia
void ChillRoutineLogic()
{
    if (chillingTimer <= 0)
    {
        walkWait = Random.Range(1, 3);
        walkTime = walkWait + Random.Range(0, 3);
        rotateWait = walkTime + Random.Range(0, 3);
        rotateOrNot = Random.Range(1, 2);
        rotationTime = rotateWait + Random.Range(0, 3);
        rotationQuantity = Random.Range(-180, 180);

        rb.constraints = RigidbodyConstraints.FreezeRotation;
    }

    chillingTimer += Time.deltaTime;
    Wander();
}
```

```
92      void Wander()
93      {
94          entity_gameObject = rb.gameObject;
95
96          if (chillingTimer <= walkWait)
97              return;
98
99          if (chillingTimer <= walkTime)
100         {
101             Vector3 acceleration = entity_gameObject.transform.forward * movementSpeed * Time.fixedDeltaTime;
102             acceleration.y = 0;
103             rb.velocity = acceleration;
104             return;
105         }
106         if (chillingTimer <= rotateWait)
107             return;
108
109         if (rotateOrNot == 1)
110         {
111             if (chillingTimer <= rotationTime)
112             {
113                 entity_gameObject.transform.Rotate(entity_gameObject.transform.up * Time.deltaTime * rotationQuantity);
114                 return;
115             }
116         }
117         chillingTimer = 0;
118     }
119 }
```

- ActionDestroyEntity: Atacar a la presa.

```
22      /// <summary>
23      ///     Kill enemy action logic & visual feedback image change
24      /// </summary>
        2 referencias
25      public override void Act()
26      {
27          Kill();
28          rb.velocity = Vector3.zero;
29
30          visualFeedback.stateImage_attacking.gameObject.SetActive(true);
31          visualFeedback.stateImage_following.gameObject.SetActive(false);
32          visualFeedback.stateImage_searching.gameObject.SetActive(false);
33          visualFeedback.stateImage_resting.gameObject.SetActive(false);
34      }
35
        1 referencia
36      public void Kill()
37      {
38          if (targetDetector.target == null) return;
39
40          targetDetector.target.SetActive(false);
41      }
42  }
43
```

- ActionEscape: Escapar del cazador.

```
32      public override void Act()
33      {
34          EscapeOpositeDirection(targetDetector.target, escapeSpeed);
35          entityStats.isPaniquing = false;
36          entityStats.isEscaping = true;
37
38          visualFeedback.stateImage_escaping.gameObject.SetActive(true);
39          visualFeedback.stateImage_chilling.gameObject.SetActive(false);
40          visualFeedback.stateImage_lookingOut.gameObject.SetActive(false);
41          visualFeedback.stateImage_resting.gameObject.SetActive(false);
42      }
43      /// <summary>
44      ///     Escape opposite direction of target logic
45      /// </summary>
        1 referencia
46      void EscapeOpositeDirection(GameObject target, float speed)
47      {
48          entity_gameObject = rb.gameObject;
49
50          Vector3 direction = -(target.transform.position - entity_gameObject.transform.position).normalized;
51          direction.y = 0;
52          Vector3 acceleration = direction * speed * Time.fixedDeltaTime;
53
54          entityStats.UseStamina();
55
56          if (acceleration.magnitude > maxSpeed)
57          {
58              acceleration.Normalize();
59              acceleration *= maxSpeed;
60          }
61
62          entity_gameObject.transform.LookAt(direction * rotationSpeed * Time.fixedDeltaTime);
63          rb.velocity = acceleration;
64      }
65  }
66
```

- ActionLookOut: Ver si está cerca en el campo de visión de la presa el cazador.

```
/// <summary>
/// The function sets the hiding variable to false, sets the velocity of the rigidbody to zero & activates visual feedback
/// </summary>
2 referencias
public override void Act()
{
    entityStats.isPaniquing = false;
    entityStats.isEscaping = false;

    entity_gameObject = rb.gameObject;

    if (targetDetector.target == null)
        return;

    Vector3 direction = -(targetDetector.target.transform.position - entity_gameObject.transform.position).normalized;
    direction.y = 0;
    entity_gameObject.transform.LookAt(direction * rotationSpeed * Time.fixedDeltaTime);

    rb.velocity = Vector3.zero;
    rb.constraints = RigidbodyConstraints.FreezeRotation;

    visualFeedback.stateImage_escaping.gameObject.SetActive(false);
    visualFeedback.stateImage_chilling.gameObject.SetActive(false);
    visualFeedback.stateImage_lookingOut.gameObject.SetActive(true);
    visualFeedback.stateImage_resting.gameObject.SetActive(false);
}
}
```

- ActionPanic: Si eres una presa y hay otra presa cerca escapando, salir corriendo en dirección contraria asustada hasta cansarte.

```
public override void Act()
{
    if (targetDetector.target == null)
        entityStats.isPaniquing = false;

    EscapeOpositeDirection(targetDetector.target, escapeSpeed);

    visualFeedback.stateImage_escaping.gameObject.SetActive(true);
    visualFeedback.stateImage_chilling.gameObject.SetActive(false);
    visualFeedback.stateImage_lookingOut.gameObject.SetActive(false);
    visualFeedback.stateImage_resting.gameObject.SetActive(false);
}
/// <summary> Escape opposite direction of target logic
1 referencia
void EscapeOpositeDirection(GameObject target, float speed)
{
    entity_gameObject = rb.gameObject;

    entityStats.isPaniquing = true;
    entityStats.isEscaping = false;


    if (target != null)
        targetLastPosition = target.transform.position;

    Vector3 direction = -(targetLastPosition - entity_gameObject.transform.position).normalized;
    direction.y = 0;
    Vector3 acceleration = direction * speed * Time.fixedDeltaTime;

    entityStats.UseStamina();

    if (acceleration.magnitude > maxSpeed)
    {
        acceleration.Normalize();
        acceleration *= maxSpeed;
    }

    entity_gameObject.transform.LookAt(direction * rotationSpeed * Time.fixedDeltaTime);
    rb.velocity = acceleration;
}
```

- ActionRest: Descansar para recuperar la estamina.

```
/// <summary>
/// The function sets the hiding variable to false, sets the velocity of the rigidbody to zero & activates visual feedback
/// </summary>
2 referencias
public override void Act()
{
    if (entityStats.gameObject.tag == "Hunter")
    {
        visualFeedback.stateImage_attacking.gameObject.SetActive(false);
        visualFeedback.stateImage_following.gameObject.SetActive(false);
        visualFeedback.stateImage_searching.gameObject.SetActive(false);
        visualFeedback.stateImage_resting.gameObject.SetActive(true);
    }
    if (entityStats.gameObject.tag == "Prey")
    {
        entityStats.isPaniquing = false;
        entityStats.isEscaping = false;

        visualFeedback.stateImage_escaping.gameObject.SetActive(false);
        visualFeedback.stateImage_chilling.gameObject.SetActive(false);
        visualFeedback.stateImage_lookingOut.gameObject.SetActive(false);
        visualFeedback.stateImage_resting.gameObject.SetActive(true);
    }

    rb.velocity = Vector3.zero;
    entityStats.RegenStamina();
}
}
```

- ActionSeek: Perseguir a la presa.

```
public override void Act()
{
    visualFeedback.stateImage_attacking.gameObject.SetActive(false);
    visualFeedback.stateImage_following.gameObject.SetActive(true);
    visualFeedback.stateImage_searching.gameObject.SetActive(false);
    visualFeedback.stateImage_resting.gameObject.SetActive(false);

    MoveTowards(targetDetector.target, followSpeed);
}

/// <summary>
///     Regens stamina automatically with time
/// </summary>
1 referencia
void MoveTowards(GameObject target, float speed)
{
    entity_gameObject = rb.gameObject;

    if (targetDetector.target == null)
        return;

    Vector3 direction = (target.transform.position - entity_gameObject.transform.position).normalized;
    direction.y = 0;
    Vector3 acceleration = direction * speed * Time.fixedDeltaTime;

    entityStats.UseStamina();

    if (acceleration.magnitude > maxSpeed)
    {
        acceleration.Normalize();
        acceleration *= maxSpeed;
    }

    entity_gameObject.transform.LookAt(direction * rotationSpeed * Time.fixedDeltaTime);
    rb.velocity = acceleration;
}
}
```

- Conditions:
  - ICondition: Clase base de las condiciones.
  - ConditionHasStamina: Comprueba si la entidad tiene estamina.
  - ConditionNoStamina: Comprueba si la entidad tiene estamina.
  - ConditionTargetNear_ColleagueEscaping: Comprueba si la presa cercana está escapando.
  - ConditionTargetActive: Ver si la entidad tiene un objetivo.
  - ConditionTargetNear: Ver si hay un objetivo muy cerca.

- **ConditionTargetNear_ColleagueEscaping:** Si es presa, ver si hay otra presa cerca que esté en estado de escapar.

- **ConditionTargetNear_ColleagueNotEscaping:** Si es presa, ver si hay otra presa cerca que no esté en estado de escapar.

- **ConditionTargetOutOfSight:** Comprobar si ya no hay un objetivo a la vista.

- **ConditionTargetVeryClose:** Ver si hay un objetivo muy cerca.

- **ConditionNotPaniquing:** Ver si la entidad está en pánico.

Autocrítica

Creo que en esta práctica mejoré la organización bastante y que logré un código bastante escalable de cara a futuro y meter nuevos estados, condiciones y acciones en general de manera simple.

Se que es muy mejorable también en muchos aspectos, pero estoy contento con el resultado, es bastante limpio y funcional.