«Talento Tech»

Desarrollo de Videojuegos

Unity 2D

Clase 09

«Talento Tech»

Buenos Aires Ciudad

Clase N° 09 | Decisiones

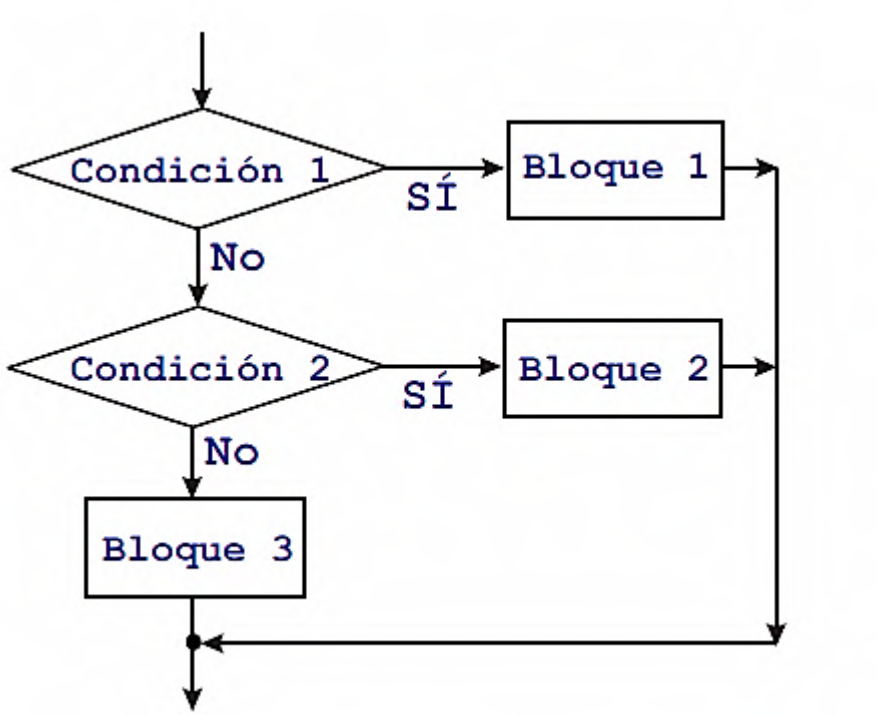
Temario:

- Toma de Decisiones usando Condicionales
- Creación de Situaciones

¿Qué es una IA?

Una IA, o inteligencia artificial, es un campo de la informática que se centra en crear sistemas capaces de realizar tareas que normalmente requieren inteligencia humana. Esto incluye habilidades como el aprendizaje, el razonamiento, la percepción y la toma de decisiones. Las IA pueden variar desde programas simples que realizan tareas específicas hasta sistemas complejos que pueden aprender de la experiencia y adaptarse a nuevas situaciones.

En nuestro caso crearemos IAs sencillas para VJ utilizando solo Condicionales como base.



Ejemplo 1: Movimiento básico hacia un objetivo.

En este ejemplo, el **NPC** (“Non playable Character”, es decir, cualquier personaje que no sea un jugador) se mueve hacia un objetivo si está lo suficientemente cerca. Si el objetivo está fuera del rango, el NPC se detiene. Se puede utilizar como base del código de un enemigo que debe seguirte al acercarte.

```
public class SimpleMovementAI : MonoBehaviour
{
    public Transform target; // El objetivo al cual se moverá
    public float speed = 2f; // Velocidad del NPC
    public float detectionRange = 5f; // Rango de detección

    void Update() {
        // Calculamos la distancia entre el NPC y el objetivo
        float distance = Vector3.Distance(transform.position, target.position);
        // Si la distancia es menor al rango de detección, el NPC se mueve hacia el objetivo
        if (distance < detectionRange) {
            Vector3 direction = (target.position - transform.position).normalized;
            transform.Translate  (direction * speed * Time.deltaTime);
        }
        else {
            // Si está fuera del rango, el NPC se detiene
            // Podríamos agregar otra lógica aquí si queremos que haga otra cosa.
        }
    }
}
```

Veamos el código:

```
public Transform target;
```

Recordemos que la clase **Transform**, encierra *Posición, Rotación y Escala*. Entonces, si necesitamos la posición de algún objeto, en vez de crear una variable de referencia a un **GameObject**, podemos hacerla directamente del tipo **Transform.** Recuerden **asignarlo**, en este caso desde el **Inspector**

```
float distance = Vector3.Distance(transform.position, target.position);
```

El **Vector3.Distance** es una función que interpola 2 objetos y del Vector obtenido genera un valor en **float** dando como resultado la “distancia” entre uno y otro.
Así podemos usar este dato para nuestra conveniencia.

Ejemplo 2: Patrullaje entre 2 puntos.

Este ejemplo permite que el **NPC** patrulle entre dos puntos y cambie de dirección al llegar a cada uno de ellos.

```
public class SimplePatrolAI : MonoBehaviour{
    public Transform pointA; // Punto A de patrullaje
    public Transform pointB; // Punto B de patrullaje
    public float speed = 2f; // Velocidad de patrullaje
    private Transform currentTarget; // Objetivo actual

    void Start() {
        // Inicia moviéndose hacia el punto A
        currentTarget = pointA;
    }

    void Update() {
        // Calcula la distancia hacia el objetivo actual
        float distance = Vector3.Distance(transform.position, currentTarget.position);

        // Si está cerca del objetivo, cambia al otro punto
        if (distance < 0.1f){
            currentTarget = currentTarget == pointA ? pointB : pointA;
        }
```

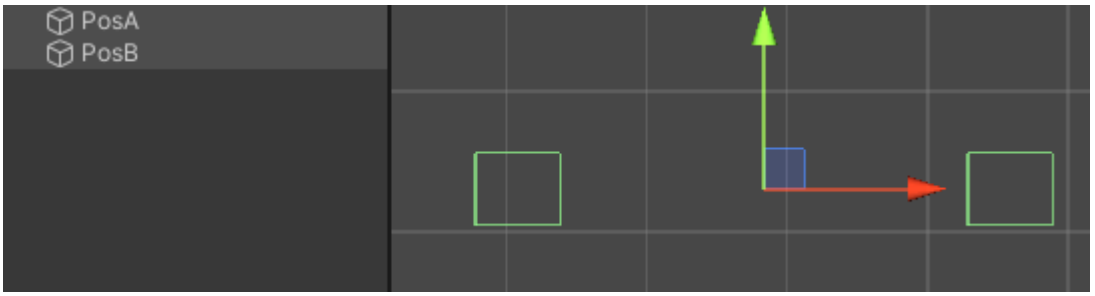
```
        // Mueve el NPC hacia el objetivo actual
        Vector3 direction = (currentTarget.position - transform.position).normalized;
        transform.Translate(direction * speed * Time.deltaTime);
    }
}
```

Explicando el código

```
    public Transform pointA; // Punto A de patrullaje
    public Transform pointB; // Punto B de patrullaje
```

Como en el ejemplo anterior, **debemos asignar** estas variables desde el **Inspector**. Les sugerimos crear 2 **EmptyObjects**, con 1 Collider cada uno por si más adelante nos interesa interactuar con ellos.

Imágen de los 2 Empties con solo sus Colliders2D:



Explicación del cambio de target:

```
currentTarget = currentTarget == pointA ? pointB : pointA;
```

Aunque no lo parezca este es un “if” pero utilizando un **Operador ternario**: Esta línea utiliza el operador ternario (? :), que es una forma compacta de realizar una expresión condicional. Tiene la siguiente estructura:

“*condición ? valorSiVerdadero : valorSiFalso*”

Condición: En este caso, la condición es currentTarget == pointA. Aquí se está verificando si currentTarget es igual a pointA.

Valores de resultado:

- Si la condición es verdadera (currentTarget es igual a pointA), se asigna pointB a currentTarget.
- Si la condición es falsa (es decir, currentTarget no es igual a pointA), se asigna pointA a currentTarget.

Ejemplo 3: Detección de Enemigos con tags.

En este ejemplo, el NPC detecta si un objeto con una tag/etiqueta específica ("Enemigo") entra en su rango y emite una alerta.

```
public class SimpleDetectionAI : MonoBehaviour
{
    public float detectionRange = 5f; // Rango de detección

    void Update()
    {
        // Busca todos los objetos con la etiqueta "Enemigo"
        GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");

        foreach (GameObject enemy in enemies)
        {
            float distance = Vector3.Distance(transform.position, enemy.transform.position);

            // Si un enemigo está dentro del rango de detección, imprime un mensaje
            if (distance < detectionRange)
            {
                Debug.Log("¡Enemigo detectado!");
                // Podrías agregar aquí otra lógica de respuesta
            }
        }
    }
}
```

Explicando algunas cosas:

```
GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemigo");
```

Esta línea lo que hace es crear un Array donde guardará cada enemigo disponible en el mapa. Recuerden que el Array es de tamaño estático así que, lo que hace este código es volver a crear el Array buscando todos los enemigos del mapa para así "actualizarlo".
Siempre existen otros métodos y los estaremos viendo más adelante.

```
foreach (GameObject enemy in enemies)
    {
        float distance = Vector3.Distance(transform.position, enemy.transform.position);

        // Si un enemigo está dentro del rango de detección, imprime un mensaje
        if (distance < detectionRange)
        {
            Debug.Log("¡Enemigo detectado!");
            // Podrías agregar aquí otra lógica de respuesta
        }
    }
```

Una vez creada el Array, se utiliza el ForEach para realizar la iteración, navegar dentro del Array y preguntar si el "enemigo seleccionado" se encuentra cerca de mi, sacando la distancia con el **Vector3.Distance**, si esto es asi, nos avisara que tenemos un enemigo cerca.
Es un código útil para juegos de sigilo que solo nos muestran un espacio reducido.

Ejemplo 4: Patrullaje con detección de jugador y cambio de estado.

En este ejemplo, el NPC patrulla entre dos puntos. Si detecta al jugador en su rango, deja de patrullar y lo persigue. Cuando el jugador sale del rango de detección, el NPC vuelve a patrullar.

```
public class PatrolAndChaseAI : MonoBehaviour
{
    public Transform pointA;
    public Transform pointB;
```

```
    public Transform player;
    public float speed = 3f;
    public float detectionRange = 7f;
    private Transform currentTarget;

    void Start(){
        currentTarget = pointA;
    }

    void Update(){
        float playerDistance = Vector3.Distance(transform.position, player.position);
        if (playerDistance < detectionRange) {
            // Si el jugador está dentro del rango, el NPC cambia a modo persecución
            MoveTowards(player.position);

        }
        else {
            // Si el jugador sale del rango, el NPC vuelve a patrullar
             Patrol();
        }


    void Patrol(){
        float distanceToTarget = Vector3.Distance(transform.position, currentTarget.position);

        if (distanceToTarget < 0.1f)
        {
            // Cambia de objetivo cuando llega a un punto de patrullaje
            currentTarget = currentTarget == pointA ? pointB : pointA;
        }
    }

        MoveTowards(currentTarget.position);
    }

    void MoveTowards(Vector3 targetPosition)
    {
        Vector3 direction = (targetPosition - transform.position).normalized;
        transform.Translate(direction * speed * Time.deltaTime);
    }
}
```

Explicación:

```
    public Transform pointA;
    public Transform pointB;
    public Transform player;
```

Creamos las 3 variables de tipo Transform para que el NPC se mueva entre ellas.

```
public float speed = 3f;
public float detectionRange = 7f;
```

Ponemos la velocidad de movimiento del NPC y su rango de detección. Si estamos dentro del rango, nos perseguirá.

```
private Transform currentTarget;
```

Creamos la variable que será el “Target”/objetivo/dirección a donde nuestro NPC irá. Este Target estará alternándose entre los 2 puntos y el Player.

```
  void Start(){
     currentTarget = pointA;
  }
```

Al comenzar se le asignará el puntoA

```
void Update()
  {
     float playerDistance = Vector3.Distance(transform.position, player.position);
     if (playerDistance < detectionRange) {
        // Si el jugador está dentro del rango, el NPC cambia a modo persecución
        isChasingPlayer = true;
        MoveTowards(player.position);

     }
     else {
        // Si el jugador sale del rango, el NPC vuelve a patrullar
        isChasingPlayer = false;
         Patrol();
     }
```

Y ahora empezaremos a detectar continuamente la cercanía del **Player**, si está “lejos”, seguirá patrullando del **punto A al B**, y si está cerca, cambiará rápidamente a perseguirnos.

```
void Patrol(){
     float distanceToTarget = Vector3.Distance(transform.position, currentTarget.position);

     if (distanceToTarget < 0.1f)
     {
        // Cambia de objetivo cuando llega a un punto de patrullaje
        currentTarget = currentTarget == pointA ? pointB : pointA;
     }

     MoveTowards(currentTarget.position);
  }
```

La función patrulla será la misma utilizada en el primer Ejemplo, dando el recorrido al NPC

```
  void MoveTowards(Vector3 targetPosition)
  {
      Vector3 direction = (targetPosition - transform.position).normalized;
      transform.Translate(direction * speed * Time.deltaTime);
  }
```

Y esta última función será la que asignará el Target, es decir, hacia donde nos debemos mover y producirá su movimiento.

Ejercicios prácticos:

Armen el sistema de patrullaje de sus enemigos utilizando el último ejemplo creado. Agreguenle alguna condición. Intenten agregar conceptos entretenidos o mecánicas para su juego.

Por ejemplo: Los enemigos que patrullan son “Guardias” que, dependiendo de tu “rango Criminal” te perseguirán. Esto lo podemos crear, haciendo que algunos items que obtenemos sean “robados”, por lo tanto cada ítem nos aumentará el rango. Y si tenemos el rango suficiente, el Guardia nos perseguirá.

Resolución:

Patrullaje y persecución:

```
public class PatrolChase : MonoBehaviour
{
   public Transform pointA;
   public Transform pointB;
   public Transform player;
   public float speed = 3f;
   public float detectionRange = 7f;
   private Transform currentTarget;
   Hero p;

  void Start()
   {
      currentTarget = pointA;
      p = player.GetComponent<Hero>();
   }


  void Update()
   {
      float playerDistance = Vector3.Distance(transform.position, player.position);
```

```
    if (playerDistance < detectionRange &&  p.rangoCriminal > 3)
    {

        // Si el jugador está dentro del rango, el NPC cambia a modo persecución

        MoveTowards(player.position);


    }
    else
    {
        // Si el jugador sale del rango, el NPC vuelve a patrullar

        Patrol();
    }

}

    void Patrol()
    {
        float distanceToTarget = Vector3.Distance(transform.position, currentTarget.position);


        if (distanceToTarget < 0.1f)
        {
            // Cambia de objetivo cuando llega a un punto de patrullaje
            currentTarget = currentTarget == pointA ? pointB : pointA;
        }


        MoveTowards(currentTarget.position);
    }


    void MoveTowards(Vector3 targetPosition)
    {
        Vector3 direction = (targetPosition - transform.position).normalized;
        transform.Translate(direction * speed * Time.deltaTime);
    }

}
```

En el player:

Debe haber una variable llamada “rangoCriminal” dentro del Script principal del Player, en este caso el Script “Hero”.

```
public class Hero : MonoBehaviour
{
[SerializeField]
public int rangoCriminal;
```

En el Script de Item colocar esta función:

```
      private void AumentoRangoCriminal(Collider2D col) {

       Hero p = col.GetComponent<Hero>();
       p.rangoCriminal += 1;
     }
```

Y llamarla de esta manera dentro del Script de OntriggerEnter2D:

```
  AumentoRangoCriminal(col);
```


Buenos Aires
aprende
Agencia de Habilidades para el Futuro
Buenos
Aires
Ciudad