Prácticas Movimiento

Práctica 1

Usando el proyecto de ejemplo implementad un seek steering behavior. Este proyecto genera un ejecutable con el mismo comportamiento que moai.exe que ya hemos visto. A este ejecutable hay que pasarle un parámetro con el nombre del fichero .lua que ponga en marcha nuestro juego.

Dentro de la carpeta sample está el fichero test.lua que podéis usar para testear el correcto funcionamiento. Es el fichero que se usará para ver que la práctica funciona correctamente, así que aseguraos de que funciona bien con él antes de cambiarlo a vuestro gusto.

El programa debe leer de un fichero xml llamado params.xml localizado en la carpeta sample, toda la información necesaria:

- la posición del target
- la velocidad máxima del personaje
- la aceleración máxima
- radio de destino

y dirigirse a ella usando el **steering behavior de Seek**.

El fichero tiene este formato:

```
<root>
    <params>
        <max_velocity value=""> </max_velocity>
        <max_acceleration value=""> </max_acceleration>
        <dest_radius value=""> </dest_radius>
        <targetPosition x="" y=""> </targetPosition>
    </params>
</root>
```

Para la lectura de este fichero, dentro del esqueleto de prácticas suministrado existe una clase Params (en el fichero params.h) que lee los datos de un fichero. Está programado para el fichero de esta práctica, y deberéis modificarlo para los parámetros necesarios en el resto de prácticas.

Para la implementación de la práctica disponéis de una clase Character donde podéis añadir el código que necesitéis. Las funciones más importantes de la clase son:

- void OnUpdate(float step): Se llama en cada iteración del bucle principal con el parámetro step indicando el tiempo transcurrido desde la última llamada en segundos
- void DrawDebug(): Podéis usar esta función para pintar información de debug en la pantalla. En IA es fundamental usar bien la información de debug para saber lo que está pasando en el juego en todo momento y poder encontrar los posibles problemas de forma sencilla. Dibujar información es una parte fundamental de ese proceso.

También tenéis definida ya la variable miembro mLinearVelocity donde se almacenará la velocidad del personaje y mAngularVelocity donde se almacenará la velocidad angular del personaje.

Para modificar las condiciones iniciales del personaje no deben estar definidas en el código C++. En un juego no queremos que el personaje se coloque siempre en una posición inicial, o con una velocidad inicial por código. Lo mejor es modificar el fichero test.lua para ir modificando las condiciones iniciales del personaje. Para ello usad el código ya escrito y modificad los valores según vuestras necesidades:

entity:setLoc(0, 0) → Pone al personaje en las coordenadas X, Y proporcionadas

entity:setRot(0) → Pone al personaje con la rotación especificada con el ángulo en grados.

entity:setLinearVel(0, 0) → Pone como velocidad inicial la suministrada en las coordenadas X, Y

entity:setAngularVel(0) → Poner como velocidad angular inicial la suministrada en grados.

Consejos:

Empezad paso a paso:

1. Poned una velocidad inicial en test.lua
2. Usad esa velocidad para actualizar la posición del personaje en OnUpdate
3. Ver como el personaje se mueve por la pantalla con la velocidad configurada en test.lua
4. Añadir la aceleración
    a. Definir una variable local dentro de OnUpdate para definir una aceleración fija
    b. Usar esta aceleración para actualizar la velocidad en OnUpdate
    c. Ver como el personaje va cambiando su velocidad con el tiempo y modificando su movimiento. Experimentad con diferentes velocidades iniciales para ver cómo se modifica la dirección del movimiento al aplicar la aceleración
5. Añadir una clase SeekSteering para el cálculo de la aceleración
    a. Inputs:
        i. Personaje al que se va a aplicar (para obtener su estado: posición, velocidad)
        ii. Target al que queremos ir
    b. Outputs:
        i. Aceleración
6. SeekSteering tendrá una función GetSteering que realizará el cálculo de la nueva aceleración

7. Implementad esa función utilizando el método explicado en clase para Seek
    a. Calcular velocidad deseada: resta entre posición del target y posición del personaje
    b. Calcular aceleración necesaria para conseguir esa velocidad: resta entre velocidad deseada y velocidad actual
    c. Poned la longitud del vector aceleración a un número (máxima aceleración): Normalizar vector y escalar por la máxima aceleración.
8. SeekSteering tendrá una función DrawDebug donde se pintarán:
    a. Vector (Línea) de la velocidad deseada calculada en el frame anterior
    b. Vector (Línea) de la aceleración calculada en el frame anterior
9. Usad la aceleración calculada por SeekSteering para actualizar la velocidad en lugar de la variable local que estáis creando en el código.
10. Disfrutad con vuestra práctica terminada.

Práctica 2

Implementar ahora un **steering behavior de Arrive**.

Los parámetros también se especifican en un fichero params.xml igual que la práctica anterior.

Formato:

```
<root>
    <params>
        <max_velocity value=""> </max_velocity>
        <max_acceleration value=""> </max_acceleration>
        <dest_radius value=""> </dest_radius>
        <arrive_radius value=""> </arrive_radius>
        <targetPosition x="" y=""> </targetPosition>
    </params>
</root>
```

Práctica 3

Implementar ahora un **steering behavior de Align**, para que independientemente de cual sea la orientación inicial, termine mirando a un ángulo definido en params.xml

Formato:

```
<root>
    <params>
        <max_angular_velocity value=""> </max_angular_velocity>
        <max_angular_acceleration value=""> </max_angular_acceleration>
        <angular_dest_radius value=""> </angular_dest_radius>
        <angular_arrive_radius value=""> </angular_arrive_radius>
        <targetRotation value=""> </targetRotation>
    </params>
</root>
```

Los ángulos se especifican en grados

Práctica 4

Utilizad un **steering behavior de Arrive** para el movimiento y un **steering behavior AlignToMovement que use Align** como **delegado** para la rotación de forma que el personaje mire hacia donde se mueve.

Los parámetros también se especifican en un fichero params.xml igual que la práctica anterior.

Formato:

```
<root>
    <params>
        <max_velocity value=""> </max_velocity>
        <max_acceleration value=""> </max_acceleration>
        <dest_radius value=""> </dest_radius>
        <arrive_radius value=""> </arrive_radius>
        <targetPosition x="" y=""> </targetPosition>

        <max_angular_velocity value=""> </max_angular_velocity>
        <max_angular_acceleration value=""> </max_angular_acceleration>
        <angular_arrive_radius value=""> </angular_arrive_radius>
        <angular_dest_radius value=""> </angular_dest_radius>
        <targetRotation value=""> </targetRotation>
    </params>
</root>
```

Opcionales

Práctica 5

En esta práctica hay que implementar dos personajes: un enemigo y un personaje principal

Hay que implementar el comportamiento para estos dos personajes

Enemigo

Opción 1 (más fácil)

- Implementad un personaje que se mueva a lo largo del eje X de un lado a otro de la pantalla, con la coordenada Y fija a -200.

Opción 2 (más interesante)

- Pensad como implementaríais un personaje que se dedique a moverse de forma aleatoria por la pantalla, como si estuviera explorando o paseando.

Personaje principal

El personaje 2 utiliza el **steering behavior de Pursue** para perseguir al personaje 1. Cuando llega a la posición de personaje 1 se resetea su posición y vuelve al 0,0.

Opción 1:

params.xml:

```
<root>
    <params>
        <max_velocity value=""> </max_velocity>
        <max_acceleration value=""> </max_acceleration>
        <dest_radius value=""> </dest_radius>
        <arrive_radius value=""> </arrive_radius>
        <targetPosition x="" y=""> </targetPosition>

        <max_angular_velocity value=""> </max_angular_velocity>
        <max_angular_acceleration value=""> </max_angular_acceleration>
        <angular_arrive_radius value=""> </angular_arrive_radius>
        <angular_dest_radius value=""> </angular_dest_radius>
        <targetRotation value=""> </targetRotation>

        <enemy_speed value=""></enemy_speed>
        <enemy_minPosition x="" y=""></enemy_minPosition>
        <enemy_maxPosition x="" y=""></enemy_maxPosition>
    </params>
</root>
```

Opción 2:

params.xml:

```
<root>
    <params>
        <max_velocity value=""> </max_velocity>
        <max_acceleration value=""> </max_acceleration>
        <dest_radius value=""> </dest_radius>
        <arrive_radius value=""> </arrive_radius>
        <targetPosition x="" y=""> </targetPosition>

        <max_angular_velocity value=""> </max_angular_velocity>
        <max_angular_acceleration value=""> </max_angular_acceleration>
        <angular_arrive_radius value=""> </angular_arrive_radius>
        <angular_dest_radius value=""> </angular_dest_radius>
        <targetRotation value=""> </targetRotation>
    </params>
</root>
```

enemy_params.xml:

```
<root>
    <params>
        <max_velocity value=""> </max_velocity>
        <max_acceleration value=""> </max_acceleration>
        <dest_radius value=""> </dest_radius>
        <arrive_radius value=""> </arrive_radius>
        <targetPosition x="" y=""> </targetPosition>

        <max_angular_velocity value=""> </max_angular_velocity>
        <max_angular_acceleration value=""> </max_angular_acceleration>
        <angular_arrive_radius value=""> </angular_arrive_radius>
        <angular_dest_radius value=""> </angular_dest_radius>
        <targetRotation value=""> </targetRotation>
    </params>
</root>
```