PEC 1 – IA para juegos en primera persona

Joaquín López Soriano

Inteligencia artificial para videojuegos

Contenido

[Herramientas 3](#_Toc150108932)

[Actividad 1: escenario 3](#_Toc150108933)

[Actividad 2: agentes runners 5](#_Toc150108934)

[Código 7](#_Toc150108935)

[FsmCitizen.cs 7](#_Toc150108936)

[ICitizenMovementState.cs 10](#_Toc150108937)

[IdleMovementState.cs 10](#_Toc150108938)

[WalkingMovementState.cs 11](#_Toc150108939)

[RunningMovementState.cs 12](#_Toc150108940)

[ICitizenDestinationState.cs 13](#_Toc150108941)

[FindDestinationState.cs 13](#_Toc150108942)

[TargetDestinationState.cs 15](#_Toc150108943)

[Video 15](#_Toc150108944)

[Actividad 3: agentes abuelos 16](#_Toc150108945)

[Código 17](#_Toc150108946)

[IElderWaitingState.cs 17](#_Toc150108947)

[DelayWaitingState.cs 17](#_Toc150108948)

[Video 18](#_Toc150108949)

[Repositorio 18](#_Toc150108950)

[Bibliografía 19](#_Toc150108951)

# Herramientas

Para realizar esta actividad, se han utilizado las siguientes herramientas:

* ***Unity v2022.3.11f1***: plataforma de desarrollo de videojuegos
* ***JetBrains Rider v2022.3.3***: IDE para programar en el lenguaje de programación C#.
* ***AI Navigation v2.0.0***: Sistema inteligente que permite trasladar personajes en un escenario.
* ***Mixamo***: empresa que ofrece personajes 3D y animaciones.
* ***Hithub***: plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git.
* ***Microsoft Word v2018***: software de tratamiento de textos.

# Actividad 1: escenario

Para desarrollar un escenario 3D con obstáculos, se ha implementado una manzana que contiene un parque con entre otros los siguientes obstáculos:

* Farola
* Piedra
* Seto
* Muro
* Banco
* Buzón
* Para de autobús
* Carro de perritos calientes

A continuación, se muestra una vista cenital del escenario.

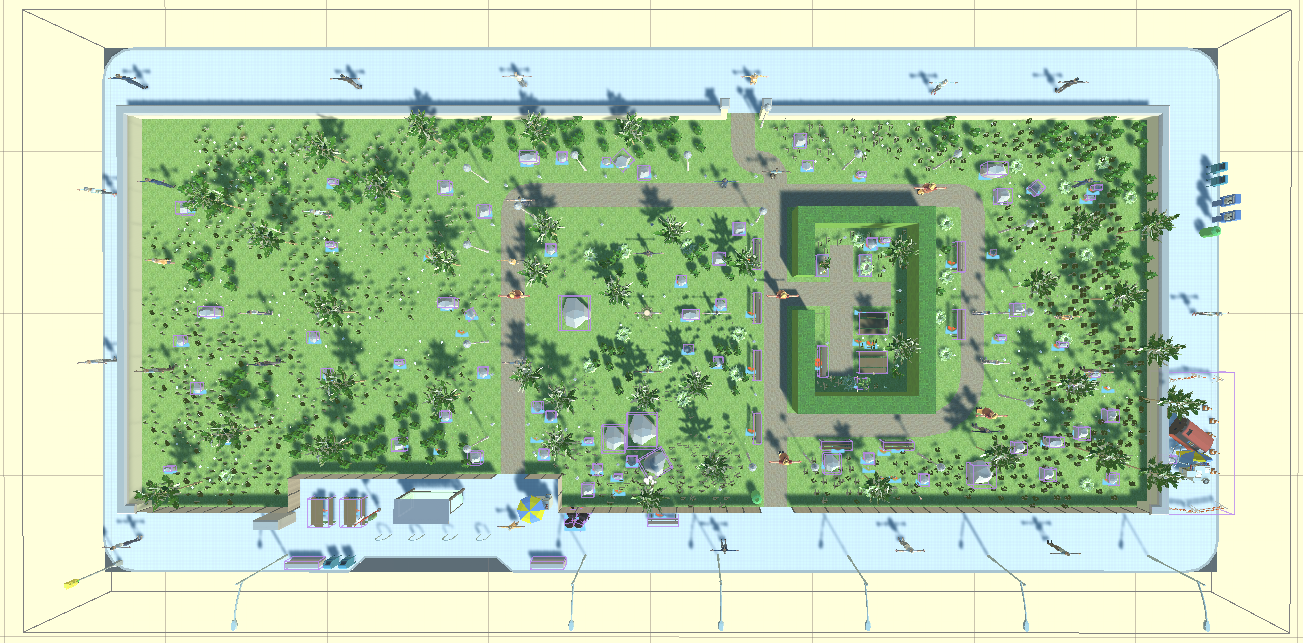


Ilustración : Vista cenital del escenario

Para definir las zonas válidas, *walkable*, y no válidas, *not walkable*, por la que circular los agentes, se ha utilizado el sistema *AI Navigation* de *Unity*, usando para definir las dos zonas un NavMesh.

En la siguiente imagen se muestra de color azul las zonas válidas para circular y en rojo las no válidas.

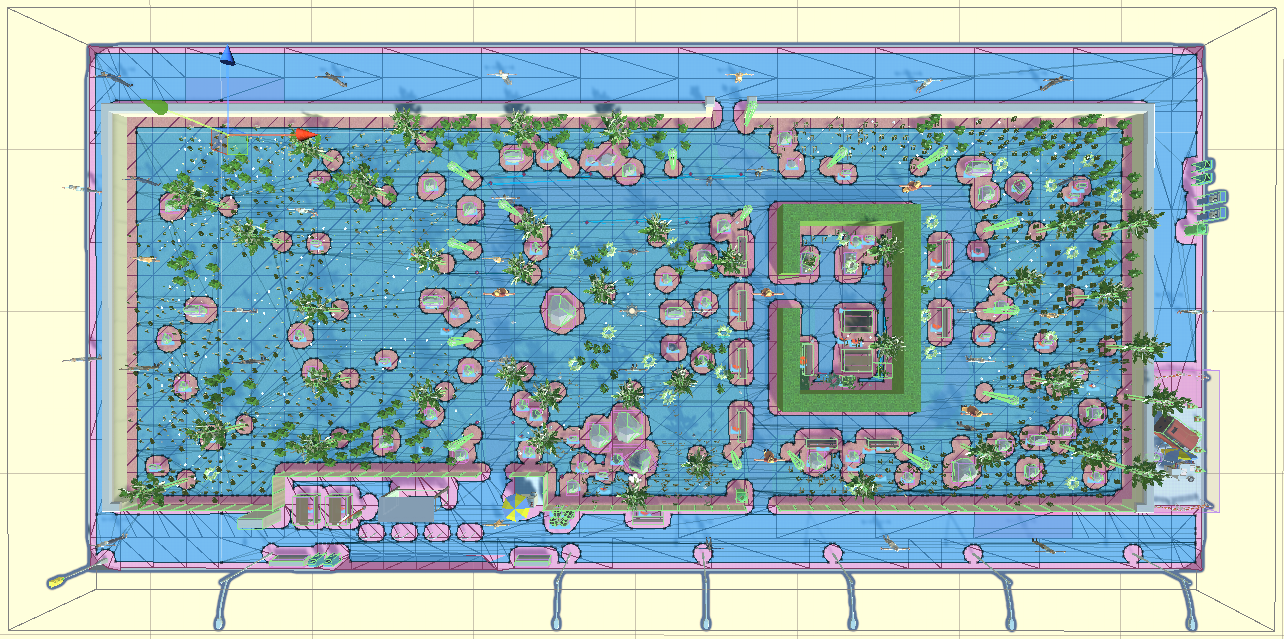
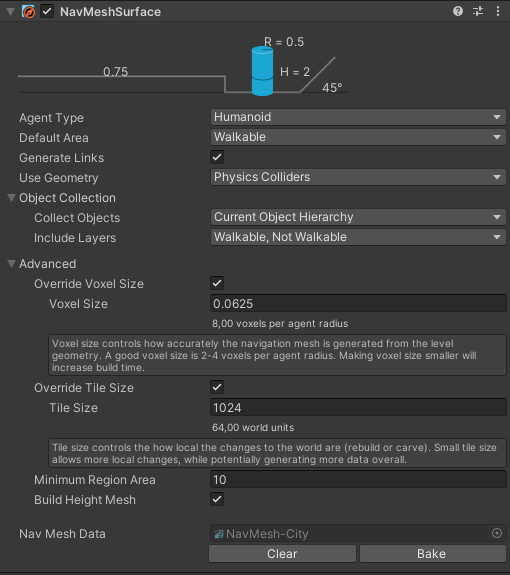


Ilustración : NavMesh del escenario

Para definir el *NavMesh*, en el objeto que contiene todo el escenario, *City*, se ha agregado el componente *NavMeshSurface* para crear la malla adecuada.

A continuación, para definir cada zona, se ha asignado un valor al layer del objeto. Si es una zona válida para moverse, como la acera o el césped, se ha asignado al layer del objeto el valor “*Walkable”*.  
 Y para una zona no válida, como bancos o muros, se ha asignado al layer del objeto el valor “*Not* *Walkable”*.

Por último, se ha generado la malla. Se ha utilizado la opción de generar la geometría de la malla usando las colisiones físicas porque ajusta mejor las zonas.

Los elementos cuyas zonas no válidas no se generaban de forma correcta, se ha agregado el componente *NavMeshModifierVolume* definido su volumen. Se ha usado en los obstáculos de banco y piedra principalmente.  
 Asimismo, se ha agregado el componente *NavMeshModifier* en objetos como las plantas o plantas para que no se tuvieran en cuenta a la hora de generar la malla.

## Incluid vuestro nombre y año

La inclusión de nombre y año en la escena ha sido en la marquesina de la parada de autobús y en la parte trasera del poste de autobús.



Ilustración : inclusión de nombre y año en escena

# Actividad 2: agentes runners

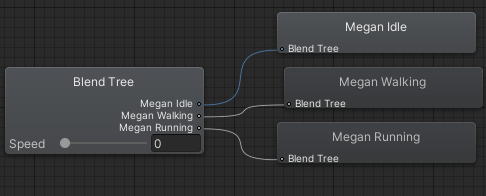
Para definir a este tipo de agentes, se ha agregado el componente *NavMesh Agent*, donde se define entre otros 2 parámetros importantes:

* la estructura del tipo de agente, *Agent Type -> Humanoide*
* el área válida de movimiento: *Area Mask -> Walkable*

Cada vez que el agente inicia el trayecto hacia el punto de destino, la velocidad de movimiento varía.

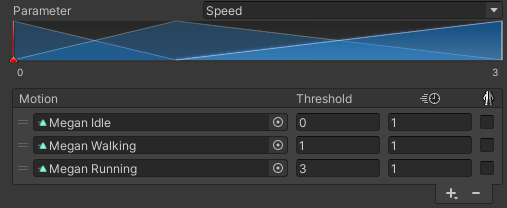
También se ha agregado el script de la clase *FsmCitizen*, definiendo un patrón de máquina de estados finita para modelar su comportamiento. Se han definido 2 máquinas de estado finitas para moldear los siguientes comportamientos:

* ***movimiento***: según el valor de la velocidad se ejecutará una animación diferente. La interfaz donde se definen los estados es *ICitizenMovementState*. Los estados son:
  + quieto *(Idle)*: en reposo
  + caminando *(walking)*: andando
  + corriendo *(running)*: andando de forma rápida
* ***destino:*** si el agente está moviéndose hacia su destino o está buscando uno nuevo. La interfaz donde se definen los estados es *ICitizenDestinationState*. Los estados son:
  + yendo al destino *(target)*
  + buscando destino *(find)*

Se ha usado un modelo 3D para definir a serie de personaje y asociado un animador con un *árbol de mezcla (Blend Tree)* donde se define el comportamiento movimiento, asociando una animación a cada estado de en función del valor de su velocidad de movimiento definida en el parámetro *Speed*:

* quieto *(Idle)*
* caminando *(walking)*
* corriendo *(running)*

Según el valor del movimiento, definido en el parámetro *speed* del componente *NavMesh Agent*, se ejecutará la siguiente animación:

* ***Velocidad menor a 1:*** estado quieto
* ***Velocidad entre 1 y 3:*** estado caminando
* ***Velocidad mayor a 3:*** estado corriendo

## Código

### FsmCitizen.cs

namespace Script.Citizen  
{  
 */\*  
 \* Citizen movement state enumeration  
 \*/* internal enum CitizenMovementState  
 {  
 **IdleState**,  
 **WalkingState**,  
 **RunningState** }  
  
 */\*  
 \* Handles the behaviour of a citizen using the FSM pattern  
 \* The citizen wander through the walkable areas of the scene  
 \*/* public class **FsmCitizen** : MonoBehaviour  
 {  
 *// Fields* [Header("Movement")]   
 [Tooltip("The speed at which the npc walks")] [SerializeField] [Range(0, 1.5f)]  
 protected float **walk** = 1f;  
 public float Walk => walk;  
   
 [Tooltip("The speed at which the npc runs")] [SerializeField] [Range(0, 4)]  
 protected float **sprint** = 3f;  
 public float Sprint => sprint;  
  
 [Tooltip("The max distance to move")] [SerializeField] [Range(0, 15)]  
 protected float **movementDistance** = 7f;  
 public float MovementDistance => movementDistance;  
  
 [Tooltip("The max distance for detecting the destination point")] [SerializeField] [Range(0, 2)]  
 protected float **detectionDistance** = 1f;  
 public float DetectionDistance => detectionDistance;  
  
 [SerializeField] private CitizenMovementState **citizenMovementState** = CitizenMovementState.**IdleState**;  
   
 [Tooltip("Max speed of movement transitions")] [Range(0, 20)]  
 public float **maxAcceleration** = 10f;  
  
 *// script classes* private NavMeshAgent \_navMeshAgent;  
 public NavMeshAgent NavMeshAgent => \_navMeshAgent;  
 private Animator \_animator;  
 public Animator Animator => \_animator;  
  
 *// private CapsuleCollider \_capsuleCollider;  
  
 //variables* private float \_movementSpeed; *// max speed* private Vector3 \_destinationPoint; *// point to go to* public Vector3 DestinationPoint  
 {  
 get => \_destinationPoint;  
 set => \_destinationPoint = value;  
 }  
   
 *// Citizen states  
 // Movement states* private ICitizenMovementState \_currentMovementState;  
 public ICitizenMovementState CurrentMovementState  
 {  
 set => \_currentMovementState = value;  
 }  
 private IdleMovementState \_idleMovementState;  
 public IdleMovementState IdleMovementState => \_idleMovementState;  
  
 private WalkingMovementState \_walkingMovementState;  
 public WalkingMovementState WalkingMovementState => \_walkingMovementState;  
  
 private RunningMovementState \_runningMovementState;  
 public RunningMovementState RunningMovementState => \_runningMovementState;  
   
 *// Destination states* private ICitizenDestinationState \_currentDestinationState;  
 public ICitizenDestinationState CurrentDestinationState  
 {  
 set => \_currentDestinationState = value;  
 }  
 private FindDestinationState \_findDestinationState;  
 public FindDestinationState FindDestinationState => \_findDestinationState;  
  
 private TargetDestinationState \_targetDestinationState;  
 public TargetDestinationState TargetDestinationState => \_targetDestinationState;  
  
 *// Initializes values* protected void **Awake**()  
 {  
 \_idleMovementState = new IdleMovementState(this);  
 \_walkingMovementState = new WalkingMovementState(this);  
 \_runningMovementState = new RunningMovementState(this);  
 \_findDestinationState = new FindDestinationState(this);  
 \_targetDestinationState = new TargetDestinationState(this);  
 \_navMeshAgent = GetComponent<NavMeshAgent>();  
 *// \_capsuleCollider = GetComponent<CapsuleCollider>();* \_animator = GetComponent<Animator>();  
 *// max speed between the walk and the sprint value* \_movementSpeed = Random.Range(walk, sprint);  
 *// max aceleration between half the acceleration and the acceleration value* \_navMeshAgent.acceleration = Random.Range(maxAcceleration / 2, maxAcceleration);   
  
 }  
   
 *// Start is called before the first frame update* protected void **Start**()  
 {  
 *// set the initial movement state and its speed* switch (citizenMovementState)  
 {  
 case CitizenMovementState.**IdleState**:  
 \_navMeshAgent.speed = 0;  
 \_currentMovementState = \_idleMovementState;  
 break;  
 case CitizenMovementState.**WalkingState**:  
 \_navMeshAgent.speed = walk;  
 \_currentMovementState = \_walkingMovementState;  
 break;  
 case CitizenMovementState.**RunningState**:  
 \_navMeshAgent.speed = sprint;  
 \_currentMovementState = \_runningMovementState;  
 break;  
 }  
 *// set the initial destination state* CurrentDestinationState = \_findDestinationState;  
 }  
  
 *// Update is called once per frame* protected void **Update**()  
 {  
 *// set the speed at the animator controller* \_animator.SetFloat(Constant.Animation.**SPEED**, \_navMeshAgent.speed);  
 \_currentMovementState.UpdateMovementState();  
 \_currentDestinationState.UpdateDestinationState();  
 }  
  
 *// Obtain a maximum speed* public void GetMovementSpeed()  
 {  
 \_movementSpeed = Random.Range(walk, sprint);  
 }  
  
 *// increase the citizen's speed up to the movementSpeed value* public void IncreaseMovementSpeed()  
 {  
 \_navMeshAgent.speed = Mathf.Lerp(\_navMeshAgent.speed, \_movementSpeed,  
 \_navMeshAgent.acceleration \* Time.fixedTime);  
 }  
 }  
}

### ICitizenMovementState.cs

namespace Script.Citizen.State.Movement  
{  
 */\*  
 \* Movement state interface  
 \*/* public interface ICitizenMovementState  
 {  
 public void UpdateMovementState(); *//Update the movement state* public void ToIdleState(); *// Transition to idle state* public void ToWalkingState(); *// Transition to walking state* public void ToRunningState(); *// Transition to running state* }

}

### IdleMovementState.cs

namespace Script.Citizen.State.Movement  
{  
 */\*  
 \* Idle state, the movement speed has to be less than 1  
 \*  
 \* Transition to  
 \* Idle -> Walking  
 \*/* public class IdleMovementState : ICitizenMovementState  
 {  
 *// FSM reference* private readonly FsmCitizen \_fsm;  
 public IdleMovementState (FsmCitizen fsmCitizen)  
 {  
 \_fsm = fsmCitizen;  
 }  
   
 *//Update the movement state* public void UpdateMovementState()  
 {  
 \_fsm.IncreaseMovementSpeed(); *// update the movement speed  
 // check the transition to the other states according to the the movement speed value* if (\_fsm.NavMeshAgent.speed >= \_fsm.Walk) *// if the the movement speed value is greater or equal to 1* { *// transition to walking state* ToWalkingState();  
 }  
 }  
  
 public void ToIdleState()  
 {  
 Debug.Log ("Can't switch to its own state");  
 }  
  
 *// Transition to walking state* public void ToWalkingState()  
 {  
 \_fsm.CurrentMovementState = \_fsm.WalkingMovementState;  
 }  
  
 public void ToRunningState()  
 {  
 Debug.Log ("Not reachable state");  
 }  
 }  
}

### WalkingMovementState.cs

namespace Script.Citizen.State.Movement  
{  
 */\*  
 \* Walking state, the movement speed has to be between 1 and 3 included  
 \*  
 \* Transition to  
 \* Walking -> Running  
 \* Walking -> Idle  
 \*/* public class WalkingMovementState : ICitizenMovementState  
 {  
 *// FSM reference* private readonly FsmCitizen \_fsm;  
 public WalkingMovementState (FsmCitizen fsmCitizen)  
 {  
 \_fsm = fsmCitizen;  
 }  
  
 *//Update the movement state* public void UpdateMovementState()  
 {  
 \_fsm.IncreaseMovementSpeed(); *// update the movement speed  
 // check the transition to the other states according to the the movement speed value* if (\_fsm.NavMeshAgent.speed < \_fsm.Walk) *// if the the movement speed value is less than 1* { *// transition to idle state* ToIdleState();  
 }  
 else if (\_fsm.NavMeshAgent.speed > \_fsm.Sprint) *// if the the movement speed value is greater than 3* { *// transition to running state* ToRunningState();  
 }  
 }  
  
 *// Transition to idle state* public void ToIdleState()  
 {  
 \_fsm.CurrentMovementState = \_fsm.IdleMovementState;  
 }  
  
 public void ToWalkingState()  
 {  
 Debug.Log ("Can't switch to its own state");  
 }  
  
 *// Transition to running state* public void ToRunningState()  
 {  
 \_fsm.CurrentMovementState = \_fsm.RunningMovementState;  
 }  
 }  
}

### RunningMovementState.cs

namespace Script.Citizen.State.Movement  
{  
 */\*  
 \* Running state, the movement speed has to be greater than 3  
 \*  
 \* Transition to  
 \* Running -> Walking  
 \*/* public class RunningMovementState : ICitizenMovementState  
 {  
 *// FSM reference* private readonly FsmCitizen \_fsm;  
 public RunningMovementState (FsmCitizen fsmCitizen)  
 {  
 \_fsm = fsmCitizen;  
 }  
   
 *//Update the movement state* public void UpdateMovementState()  
 {  
 \_fsm.IncreaseMovementSpeed(); *// update the movement speed  
 // check the transition to the other states according to the the movement speed value* if (\_fsm.NavMeshAgent.speed <= \_fsm.Sprint) *// if the the movement speed value is less or equal to 3* { *// transition to walking state* ToWalkingState();  
 }  
 }  
  
 public void ToIdleState()  
 {  
 Debug.Log ("Not reachable state");  
 }  
   
 *// Transition to walking state* public void ToWalkingState()  
 {  
 \_fsm.CurrentMovementState = \_fsm.WalkingMovementState;  
 }  
  
 public void ToRunningState()  
 {  
 Debug.Log ("Can't switch to its own state");  
 }  
 }  
}

### ICitizenDestinationState.cs

namespace Script.Citizen.State.Destination  
{  
 */\*  
 \* Destination state interface  
 \*/* public interface ICitizenDestinationState  
 {  
 public void UpdateDestinationState(); *//Update the destination state* public void ToFindDestinationState(); *// Transition to Find state* public void ToTargetDestinationState(); *// Transition to Target state* }  
}

### FindDestinationState.cs

namespace Script.Citizen.State.Destination  
{  
 */\*  
 \* Find state, on the way to the destination point  
 \*  
 \* Transition to  
 \* Find -> Target  
 \*/* public class FindDestinationState : ICitizenDestinationState  
 {  
 *// FSM reference* private readonly FsmCitizen \_fsm;  
  
 *//variables* private readonly int \_areaMask; *// walkable layer area number* public FindDestinationState (FsmCitizen fsmCitizen)  
 {  
 \_fsm = fsmCitizen;  
 \_areaMask = NavMesh.GetAreaFromName(Constant.NavMesh.**CITIZEN\_WALK**)+1; *// plus 1 so the resulted value is 1* }  
   
 *//Update the destination state  
 // Search for a random point in the walkable area to go to* public void UpdateDestinationState()  
 {  
 *// loop until find a random point in the walkable area* Vector3 destinationPoint; *// the random point in the walkable area gotten* while (!RandomPoint(\_fsm.transform.position, \_fsm.MovementDistance, out destinationPoint))  
 {  
 }  
 \_fsm.DestinationPoint = destinationPoint;   
 \_fsm.NavMeshAgent.SetDestination(destinationPoint); *// target the destination point* \_fsm.GetMovementSpeed(); *// Obtain a random maximum speed* ToTargetDestinationState(); *// transition to target state* }  
  
 public void ToFindDestinationState()  
 {  
 Debug.Log ("Can't switch to its own state");  
 }  
  
 *// Transition to target state* public void ToTargetDestinationState()  
 {  
 \_fsm.CurrentDestinationState = \_fsm.TargetDestinationState;  
 }  
   
 *// Find a random point in the walkable area* private bool RandomPoint(Vector3 center, float range, out Vector3 result)  
 {  
 *// get a random 3D point and extends it with a certain range* Vector3 randomPoint = center + Random.insideUnitSphere \* range;  
 NavMeshHit hit;  
 *// checks if the random 3D point is in the walkable area* if (NavMesh.SamplePosition(new Vector3(randomPoint.x, 0.5f, randomPoint.z), out hit, 1.0f, \_areaMask))  
 {  
 result = hit.position; *// if found, returns the random 3D point* return true;  
 }  
 *// returns vector zero if not* result = Vector3.zero;  
 return false;  
 }  
 }  
}

### TargetDestinationState.cs

namespace Script.Citizen.State.Destination  
{  
 */\*  
 \* Target state, on the way to the destination point  
 \*  
 \* Transition to  
 \* Target -> Find  
 \*/* public class TargetDestinationState : ICitizenDestinationState  
 {  
 *// FSM reference* private readonly FsmCitizen \_fsm;  
 public TargetDestinationState (FsmCitizen fsmCitizen)  
 {  
 \_fsm = fsmCitizen;  
 }  
   
 *//Update the destination state  
 // checks if the the citizen position is close enough to the destination point* public void UpdateDestinationState()  
 {  
 *// checks the transition to the other state when reached the destination point* if (\_fsm.DetectionDistance >= Vector3.Distance(\_fsm.transform.position, \_fsm.DestinationPoint)) *// if the the citizen position is close* { *// enough to the destination point* ToFindDestinationState(); *// transition to find state* }  
 }  
   
 *// transition to find state* public void ToFindDestinationState()  
 {  
 \_fsm.CurrentDestinationState = \_fsm.FindDestinationState;  
 }  
  
 public void ToTargetDestinationState()  
 {  
 Debug.Log ("Can't switch to its own state");  
 }  
 }  
}

Video

<https://youtu.be/--ScXGySnkk>

# Actividad 3: agentes abuelos

Para definir este nuevo agente, se ha partido del agente definido en el punto anterior agregado el componente *NavMesh Agent* con los mismos valores de los parámetros.  
 Para agregar la nueva funcinalidad, cuando se acerca a un banco se queda un tiempo descansando, se ha creado una nueva clase, *FsmElder*, que hereda de la clase *FsmCitizen* y una nueva interfaz *IElderWaitingState* que hereda de la interfaz *ICitizenMovementState*.

El script *FsmElder* agregado a este agente define el siguiente nuevo patrón de máquina de estados finita a partir del estado movimiento ya definido:

* ***descanar***: según el valor de la velocidad se ejecutará una animación diferente. La interfaz donde se definen los estados es *IElderWaitingState*. Los estados son:
  + quieto *(Idle)*: en reposo
  + caminando *(walking)*: andando
  + corriendo *(running)*: andando de forma rápida
  + descansar (delay): al pasar cerca de un banco, se para y espera unos segundos antes de proseguir su movimiento.

Para comprobar la cercanía de este agente con un banco, se ha agregado el componente *Capsule Collider*, definido y agregado el *tag* *“bench”* a cada objeto banco de la escena e implementado el método *OnTriggerEnter* comprobando si colisiona contra este objeto comprobando el valor de su *tag*.

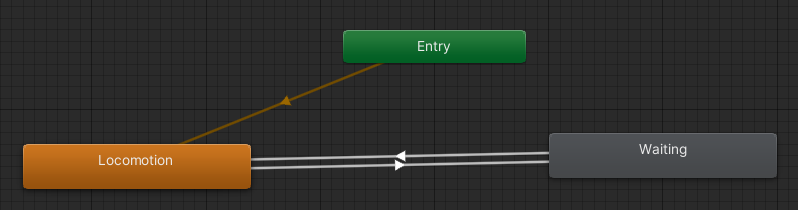
Se ha definido un tiempo de espera entre cada descanso para que este estado se espacie a lo largo del tiempo.

Por último, al igual que con los agentes anteriores, se ha usado un modelo 3D y asociado un animador para definir sus diferentes estados del comportamiento movimiento. Los estados del animador son:

* locomotion: contiene el *árbol de mezcla (Blend Tree)* que define los estados:
  + quieto *(Idle)*
  + caminando *(walking)*
  + corriendo *(running)*
* waiting: define el estado descansar. Al detectar una colisión entre el agente y un banco salta el disparador *Wait*, forzando la transición de *locomotion* a *waiting*.

Mientras esta en este estado se ejecuta una animación de espera.

Transcurridos unos segundos, salta el disparador *Leave*, forzando la transición de *waiting* a *locomotion*.

A continuación, se muestra la imagen de la máquina de estados del animador.

## Código

### IElderWaitingState.cs

namespace Script.Citizen.Elder.State.WaitingState  
{  
 */\*  
 \* Waiting state interface, inherits from the movement state interface  
 \*/* public interface IElderWaitingState : ICitizenMovementState  
 {  
 public void ToDelayWaitingState(); *// Transition to delay state* }  
}

### DelayWaitingState.cs

namespace Script.Citizen.Elder.State.WaitingState  
{  
 */\*  
 \* Waiting state, waits stopped an amount of time until continue moving again  
 \*  
 \* Transition from the other movement states  
 \* Transition to  
 \* Delay -> Idle  
 \*/* public class DelayWaitingState : IElderWaitingState  
 {  
 *// FSM reference* private readonly FsmElder fsm;  
 public DelayWaitingState (FsmElder fsmElder)  
 {  
 fsm = fsmElder;  
 }  
  
 public void UpdateMovementState()  
 {  
 *// do nothing updatable* }  
  
 *// Transition to idle state* public void ToIdleState()  
 {  
 *// return to the locomotion state at the animator controller and start moving again* fsm.Animator.SetTrigger(Constant.Animation.**LEAVE**);  
 fsm.CurrentMovementState = fsm.IdleMovementState;   
 }  
  
 public void ToWalkingState()  
 {  
 Debug.Log ("Not reachable state");  
 }  
  
 public void ToRunningState()  
 {  
 Debug.Log ("Not reachable state");  
 }  
  
 *// Transition to delay state* public void ToDelayWaitingState()  
 {   
 *// jump to the waiting state at the animator controller* fsm.Animator.SetTrigger(Constant.Animation.**WAIT**);  
 fsm.NavMeshAgent.speed = 0; *// stop the elder citizen* fsm.CurrentMovementState = fsm.DelayWaitingState; *// update the state to delay state* fsm.LaunchCoroutine(DelaySecond(fsm.MinWaitingTime, fsm.MaxWaitingTime)); *// launch a coroutine* }  
  
 *// Waits a random time between a minimum and maximum value* private IEnumerator DelaySecond(float minWaitingTime, float maxWaitingTime)  
 {  
 *//yield on a new YieldInstruction that waits for random seconds* yield return new WaitForSeconds( Random.Range(minWaitingTime, maxWaitingTime));  
 fsm.ResetWaitingTime(); *// update the passed by time* ToIdleState(); *// update the state to idle state* }  
 }  
}

## Video

<https://youtu.be/4RUqto72KC8>

# Repositorio

La URL del repositorio GIT es: <https://github.com/JoaquinLopezSoriano/IA-PEC1.git>

La URL del repositorio de Google es: <https://drive.google.com/file/d/1DiOKcU0727c-ELFkZ4RCf6zrWh7hW9MF/view?usp=sharing>

# Bibliografía

* <https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)>
* https://es.wikipedia.org/wiki/IntelliJ\_IDEA
* https://forum.unity.com/threads/theres-a-new-navigation-package-in-town.1371633/
* <https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/index.html>
* https://en.wikipedia.org/wiki/Navigation\_mesh
* <https://es.wikipedia.org/wiki/GitHub>
* <https://es.wikipedia.org/wiki/Microsoft_Word>
* <https://www.mixamo.com/>
* https://en.wikipedia.org/wiki/Mixamo