FSM & Pathfinding

William Leduc, Pierre-Olivier Roy et Raphaël Duhamel

Table des matières

01

Structure FSM

Structure de la machine et de ses états

Final State Machine

Structure de la FSM:

1. Initialisation des composantes
2. Organisation des états
3. Cycle de vie des états
4. Transition d'état

Structure des états:

1. Référence à la FSM
2. Cycle de vie
3. Logique d'entrée et de sortie

```
public abstract class CharacterState : IState
{
    protected PlayerStateMachine m_stateMachine;

    1 référence
    public void OnStart(PlayerStateMachine stateMachine)...

    6 références
    public virtual void OnEnter()...

    5 références
    public virtual void OnExit()...

    5 références
    public virtual void OnFixedUpdate()...

    5 références
    public virtual void OnUpdate()...

    5 références
    public virtual bool CanEnter(IState currentState)...

    5 références
    public virtual bool CanExit()...

    1 référence
    public void OnStart()...

}
```

02

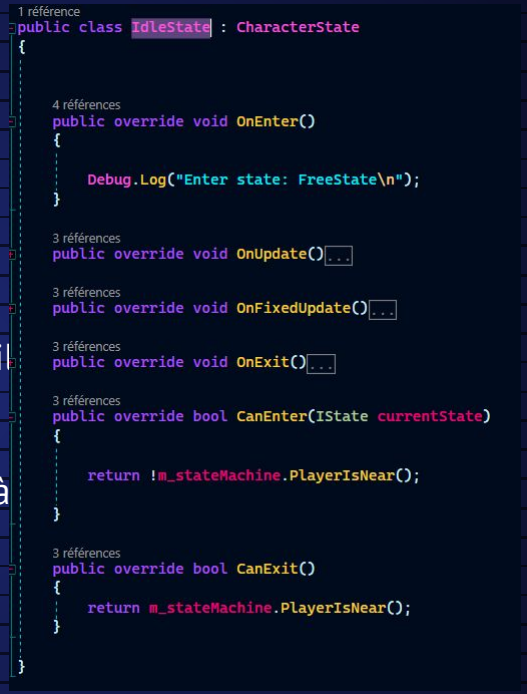
États de la FSM

Logique de la Final State Machine

État Idle

Le personnage non-joueur ne fait rien dans cette état, il attend la prochaine transition.

Entre lorsqu'un booléen est True et tourne se booléen à false après un délais.



État Poursuite

Lorsque le joueur est à proximité, le personnage non-joueur poursuit le joueur.

Détermine le joueur comme destination en Update.

Entre lorsque le joueur est suffisamment proche et sort lorsque le joueur est trop loin OU après un certain délais.

```
1 référence
public class PursuitState : CharacterState
{
    4 références
    public override void OnEnter()
    {
        m_stateMachine.GetAgent().isStopped = false;
        Debug.Log("Enter state: FreeState\n");
    }

    3 références
    public override void OnUpdate()
    {
        m_stateMachine.GetAgent().SetDestination(m_stateMachine.GetPlayerTransform().position);
    }

    3 références
    public override void OnFixedUpdate()...

    3 références
    public override void OnExit()
    {
        m_stateMachine.GetAgent().isStopped = true;
    }

    3 références
    public override bool CanEnter(IState currentState)
    {

        return m_stateMachine.PlayerIsNear();

    }

    3 références
    public override bool CanExit()
    {

        return !m_stateMachine.PlayerIsNear();
    }
}
```

État Roaming

Lorsque le joueur n'est pas détecté et que le personnage non-joueur n'est pas Idle, il patrouille un périmètre.

Itère à travers une liste pour gérer les destinations et change sa destination lorsqu'il atteint la cible.

Entre lorsque le joueur n'est pas à proximité et qu'il n'est pas idle. Sort lorsqu'il détecte le joueur.

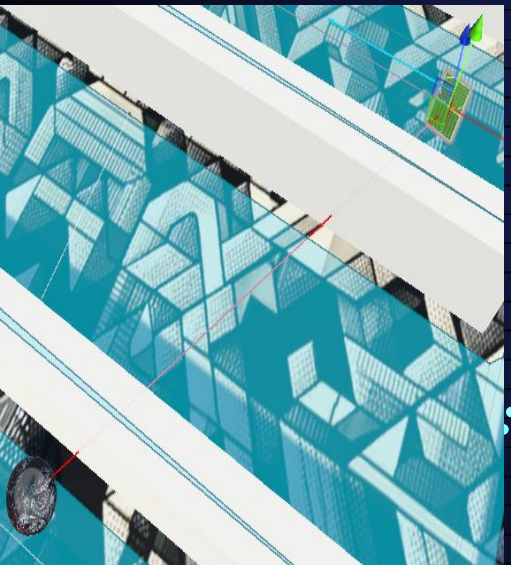
```
public class RoamingState : CharacterState
{
    4 références
    public override void OnEnter()...

    3 références
    public override void OnUpdate()
    {
        if (m_stateMachine.waypoints.Length == 0)
        {
            Debug.Log("No waypoints set");
            return; // Return if no waypoints are set
        }
        // Check if NPC has reached the current waypoint
        if (Vector3.Distance(m_stateMachine.transform.position, m_stateMachine.way
        {
            Debug.Log("Set destination");
            m_stateMachine.currentWaypointIndex = (m_stateMachine.currentWaypointI
        }
        else
        {
            //Debug.Log(m_stateMachine.GetAgent());
            //Debug.Log(m_stateMachine.waypoints[m_stateMachine.currentWaypointInd
            //Debug.Log(m_stateMachine.waypoints[m_stateMachine.currentWaypointInd
            m_stateMachine.GetAgent().SetDestination(m_stateMachine.waypoints[m_st
        }
    }

    3 références
    public override void OnFixedUpdate()...

    3 références
    public override void OnExit()...

    3 références
    public override bool CanEnter(IState currentState)
    {

        return !m_stateMachine.PlayerIsNear()
            && !m_stateMachine.IsIdle();

    }

    3 références
    public override bool CanExit()
    {

        return m_stateMachine.PlayerIsNear();
    }
}
```

03

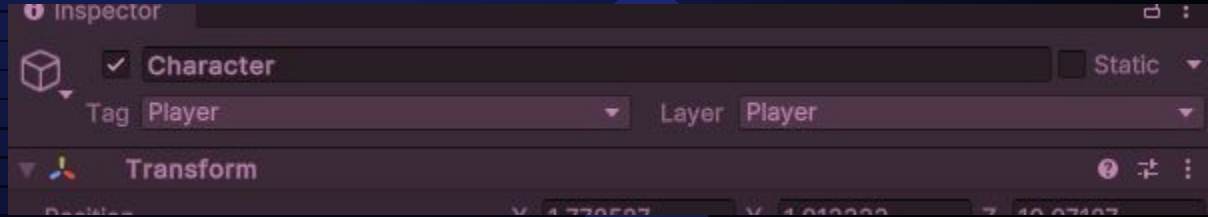
Détection : Raycasting

Logique de détection du joueur

Détection du joueur : Raycasting



Le raycasting est une technique utilisé pour détecter si un rayon invisible, émis depuis un point touche un objet. Cela permet de tracer une ligne entre le personnage non-joueur et le joueur en prenant compte des obstacles.



04

Pathfinding : Navmesh

Logique de Navmesh dans Unity

Composantes

Navmesh Agent

Attaché aux personnages.

Détermine leurs paramètre afin que le navmesh puisse déterminer leurs déplacements

Navmesh surface

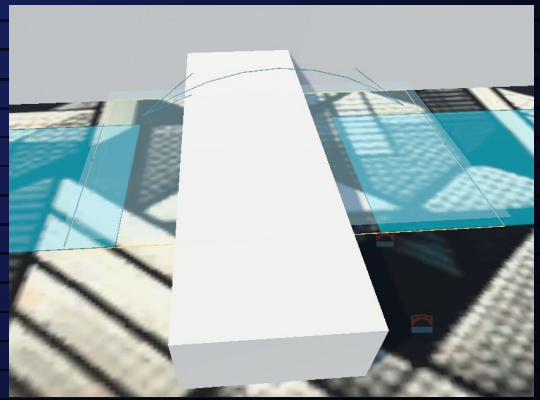
Attacher à un GameObject.

Permet de choisir les paramètres de ce qui est un chemin possible et le fabrique.

Outils : Navmesh Link

Navmesh Link

Le navmesh link permet de lier des blocs de navmesh afin de permettre des sauts ou autre déplacement désiré.



05

Bonnes pratiques de travail

Pratiques de travails mises en place lors du projet

Bonnes pratiques de travail

Sourcetree

Partage dynamique et suivi des modifications

Refactorization

Accélération du projet en prenant du code déjà existant

Nomenclature et Debug

Utilisation d'une nomenclature structuré pour une lecture rapide et de debug pour suivre lors de l'exécution du code les point focaux