Documentation: 1.- Unity3D – Easy Finite State Machine (C#)

State machines are a very effective way to manage game state, either on your main game play object (Game Over, Restart, Continue etc) or on individual actors and NPCs (AI behaviours, Animations, etc). The following is a simple state machine that should work well within any Unity context.

Designed with simplicity in mind

Most state machines come from the world of C# enterprise, and are wonderfully complicated or require a lot of boilerplate code. State Machines however are an incredibly useful pattern in game development, administrative overhead should never be a burden that discourages you from writing good code.

•Simple use of Enums as state definition.

•Minimal initialization – one line of code.

•Incredibly easy to add/remove states

•Uses reflection to avoid boiler plate code – only write the methods you actually need.

•Compatible with Coroutines. •Tested on iOS and Android

Usage

An example project is included (Unity 5.0) to show the State Machine in action.

To use the state machine you need a few simple steps

Include the StateMachine package

```
using AxlPlay; //Remember the using statement
before the class declaration

public class MyManagedComponent : MonoBehaviour{ }
```

Define your states using an Enum

```
public enum States{
    Init,
    Play,
    Win,
    Lose
}
```

Create a variable to store a reference to the State Machine

```
StateMachine<States> fsm;
```

Get a valid state machine for your MonoBehaviour

```
fsm = StateMachine<States>.Initialize(this);
```

This is where all of the magic in the StateMachine happens: in the background it inspects your MonoBehaviour (`this`) and looks for any methods described by the

convention shown below. You can call this at any time, but generally `Awake()` is a safe choice.

You are now ready to manage state by simply calling `ChangeState()`

```
fsm.ChangeState(States.Init);
```

State callbacks are defined by underscore convention (`StateName_Method`)

```
void Init_Enter(){
```

```
    Debug.Log("We are now ready");
} //Coroutines are supported, simply return
IEnumerator IEnumerator Play_Enter(){

Debug.Log("Game Starting in 3"); yield return new
WaitForSeconds(1);

Debug.Log("Game Starting in 2"); yield return new
WaitForSeconds(1);

Debug.Log("Game Starting in 1"); yield return new
WaitForSeconds(1); Debug.Log("Start");

}
void Play_Update(){
    Debug.Log("Game Playing");
}
void Play_Exit(){
    Debug.Log("Game Over");
}
```

Currently supported methods are:

•`Enter` •`Exit` •`FixedUpdate` •`Update` •`LateUpdate` •`Finally` •`OnCollisionEnter` •`OnTriggerEnter`

These methods can be private or public. The methods themselves are all optional, so you only need to provide the ones you actually intend on using.

Couroutines are supported on Enter and Exit, simply return `IEnumerator`. This can be great way to accommodate animations.

Note:`FixedUpdate`, `Update` and `LateUpdate` calls won't execute while an Enter or Exit routine is running.

Finally is a special method guaranteed to be called after a state has exited. This is a good place to perform any hygiene

operations such as removing event listeners. Note: Finally does not support coroutines.

Transitions

There is simple support for managing asynchronous state changes with long enter or exit coroutines.

`fsm.ChangeState(States.MyNextState, StateTransition.Safe);` The default is `StateTransition.Safe`. This will always allows the current state to finish

both it's enter and exit functions before transitioning to any new states.

```
fsm.ChangeState(States.MyNextState, StateTransition.Overwrite);
```

`StateMahcine.Overwrite` will cancel any current transitions, and call the next state immediately. This means any code which has yet to run in enter and exit routines will

be skipped. If you need to ensure you end with a particular configuration, the finally function will always be called:

```
void MyCurrentState_Finally(){ //Reset object to desired configuration

}
```

EXAMPLE:

using AxlPlay; //Remember the using statement before the class

```
declaration
 // init FSM
public enum States {
```

```
      Idle,
Pursue,

ArrivedEvent
} public StateMachine<States> fsm; void Start(){

//Initialize State Machine Engine fsm =
StateMachine<States>.Initialize(this, States.Idle);

 }
void Idle_Enter(){
fsm.ChangeState(States.Pursue) }

void Pursue_Enter(){
}
void Pursue_Update(){
}
```

2.- FSM Example:

```
// init FSM

    public enum States{
        Idle,
Flee,

Finish

    }
    public StateMachine<States> fsm;
    void Awake() {
      navMeshAgent = GetComponent<NavMeshAgent>();
    }
void Start() { //Initialize State Machine Engine fsm =
StateMachine<States>.Initialize(this, States.Idle);

} void Idle_Enter(){

        Debug.Log("Flee => Idle_Enter");
    }
```

3.- Actions:

Flee:

Flee will move the agent away from the target with pathfinding. fleedDistance The agent has fleed when the magnitude is greater than this value

lookAheadDistance

The distance to look ahead when fleeing

target

The GameObject that the agent is fleeing from

Pursue:

Pursue predicts where the target is going to be in the future. This allows the agent to arrive at the target earlier than it would have with the Set Destination Action.

targetDistPrediction

How far to predict the distance ahead of the target. Lower values indicate less distance should be predicated

targetDistPredictionMult

Multiplier for predicting the look ahead distance

target

The GameObject that the agent is pursuing

EVADE:

Evade is similar to the Flee except the Evade predicts where the target is going to be in the future. This allows the agent to flee from the target earlier than it would have with the Flee.

evadeDistance

The agent has evaded when the magnitude is greater than this value.

lookAheadDistance

The distance to look ahead when evading.

targetDistPrediction

How far to predict the distance ahead of the target. Lower values indicate less distance should be predicated.

targetDistPredictionMult

Multiplier for predicting the look ahead distance.

target

The GameObject that the agent is evading.

PATROL:

Patrol moves from waypoint to waypoint.

randomPatrol

Should the agent patrol the waypoints randomly?

waypointPauseDuration

The length of time that the agent should pause when arriving at a waypoint

waypoints

The waypoints to move to

CAN SEE OBJECT:

Can See Object returns the object when it sees an object in front of the current agent.

usePhysics2D

Should the 2D version be used?

targetObject

The object that we are searching for. If this value is null then the objectLayerMask will be used

objectLayerMask

The LayerMask of the objects that we are searching for

ignoreLayerMask

The LayerMask of the objects to ignore when performing the line of sight check

fieldOfViewAngle

The field of view angle of the agent (in degrees)

viewDistance

The distance that the agent can see

offset

The offset relative to the pivot position

targetOffset

The target offset relative to the pivot position

returnedObject

The object that is within sight

CAN HEAR OBJECT:

The Can Hear Object return the object when it hears another object.

usePhysics2D

Should the 2D version be used?

targetObject

The object that we are searching for. If this value is null then the objectLayerMask will be used

objectLayerMask

The LayerMask of the objects that we are searching for

hearingRadius

How far away the unit can hear

audibilityThreshold

The further away a sound source is the less likely the agent will be able to hear it. Set a threshold for the the minimum audibility level that the agent can hear

offset

The offset relative to the pivot position

returnedObject

The returned object that is heard

WANDER:

Wander moves the agent randomly throughout the map.

wanderDistance

How far ahead of the current position to look ahead for a wander

wanderRate

The amount that the agent rotates direction

SEARCH:

Search will search the map by wandering until it finds the target. It can find the target by seeing or hearing the target.

wanderDistance

How far ahead of the current position to look ahead for a wander

wanderRate

The amount that the agent rotates direction

fieldOfViewAngle

The field of view angle of the agent (in degrees)

viewDistance

The distance that the agent can see

ignoreLayerMask

The LayerMask of the objects to ignore when performing the line of sight check

senseAudio

Should the search end if audio was heard?

hearingRadius

How far away the unit can hear

offset

The offset relative to the pivot position

targetOffset

The target offset relative to the pivot position

objectLayerMask

The LayerMask of the objects that we are searching for

linearAudibilityThreshold

The further away a sound source is the less likely the agent will be able to hear it. Set a threshold for the the minimum audibility level that the agent can hear

returnedObject

The object that is within sight

-SHOOT

The scene ShooterDemo is an example of this action. This action shoots a gun.

timeBetweenBullets The time that the agent waits between each shoot **range**

The distance of the bullet.

ShootableMask

The layer of the gameobjects that the agent can shoot

GunGameobject

The gameobject where the bullet comes out

PlayShootAnimation

Would you like to play an animation when the agent shoot?

TriggerShootAnimation

The animation that will be played when the agent shoot. If PlayShootAnimation is false you haven't to fill this

OnPlayerHit

The event that will be fire when the agent hit the target

AgentCanReload

Do you want to the agent can reload?

AmmunitionForCartridge

A number of bullets that the agent can shoot before of reload. If AgentCanReload is false don't fill this field

TriggerAnimationReload

The parameter in the animator.If AgentCanReload is false do not fill this field

GunLight

Do you want to turn on a light when the agent shoot?.

PlayAudioOnShoot

Would you like to play an audio when the agent shoot?.

SecondsReloading

The seconds that the agent will take reloading

PlayMuzzleFlash

Do you want that the pistol plays a muzzle flash when the agent shoot?.

Audio

The audio that will be played when the agent shoot. If PlayAudioOnShoot is false don't fill this field.

HitPoint

The hit point of the shoot(Only if hit a ShootableMask) .

-TARGET REACHABLE

Field Of View returns an object when it sees an object within of the field of view, without an object blocking the target.

viewDistance

Is the distance that the agent can see.

viewAngle

Is the field of view of the agent.

targetMask

Is the layer of gameobjects that you want to detect.

obstacleMask

Is the layer of obstacles that can block the sight between agent and target. Example: A wall.

numRaysScene

Is the number of rays that you can see in scene view (Gizmos).

StoreTarget

Store the gameobject seen closest

hitEvent

Event to send if get a target object

-AGENT STOP

This action stop the NavMesh agent

-AGENT RESUME

This action resume the NavMesh agent

-CAN SEE FROM SIGHT

Can see from sight.

goFromSight

The object From sight.

usePhysics2D

The field of view angle of the agent (in degrees).

fieldOfViewAngle

The distance that the agent can see.

viewDistance

The distance that the agent can see.

offset

The offset relative to the pivot position.

returnedObject

The object that is within sight

finishEvent

This event will fire when the target was seen

targetObject

The object that we are searching for. If this value is null then the objectLayermask will be used

HitLayerMask

The LayerMask of the objects that we are searching for.

IgnoreLayerMask

The LayerMask of the objects to ignore when performing the line of sight check

targetOffset

The target offset relative to the pivot position

angleOffset2D

The angle offset relative to the pivot position 2D

If you need any help with the implementation or if you think we should add some other new features, please don't hesitate to contact

me

oliver@axlplay.com https://axlplay.com