Unity
Services Search Forums Recent Posts
Made with Unity
Learn
Community
Asset Store
Get Unity

Unity Account
You need a Unity Account to shop in the Online and Asset Stores, participate in the Unity Community and manage your license portfolio. Login Create account
Forums
Answers
Feedback
Issue Tracker
Blog
Evangelists
User Groups
User Research

Navigation
Home
Unity
Services
Made with Unity
Learn
Community
Forums
Answers
Feedback
Issue Tracker
Blog
Evangelists
User Groups
User Research
Asset Store
Get Unity
Unity Account
You need a Unity Account to shop in the Online and Asset Stores, participate in the Unity Community and manage your license portfolio. Login Create account
Language
中文
한국어
Portugués
Русский
Español
日本語

Forums > Unity Community Support > **Scripting**

Reddit AMA with Adam Myhill on Cinemachine, Thurs July 27 - 10AM PT. More info.

Unity Starter Kits feedback | 2017.2 Beta | 2017.1 released | Unity Essentials Packs | Reddit AMA | Patch releases | Calling all artists and designers

Implementing finite-state-machine AI. (C#)

Search this thread...

Ecoste

Joined: Nov 2, 2014
Posts: 23

I was reading this book about state driven AI(Programming Game AI By Example) which clearly explains the structure, however it does so in C++.

I want to re-create this structure, however I'm having a plethora of problems already. This is especially complicated for me because I usually do everything in JS.

(Basic structure)

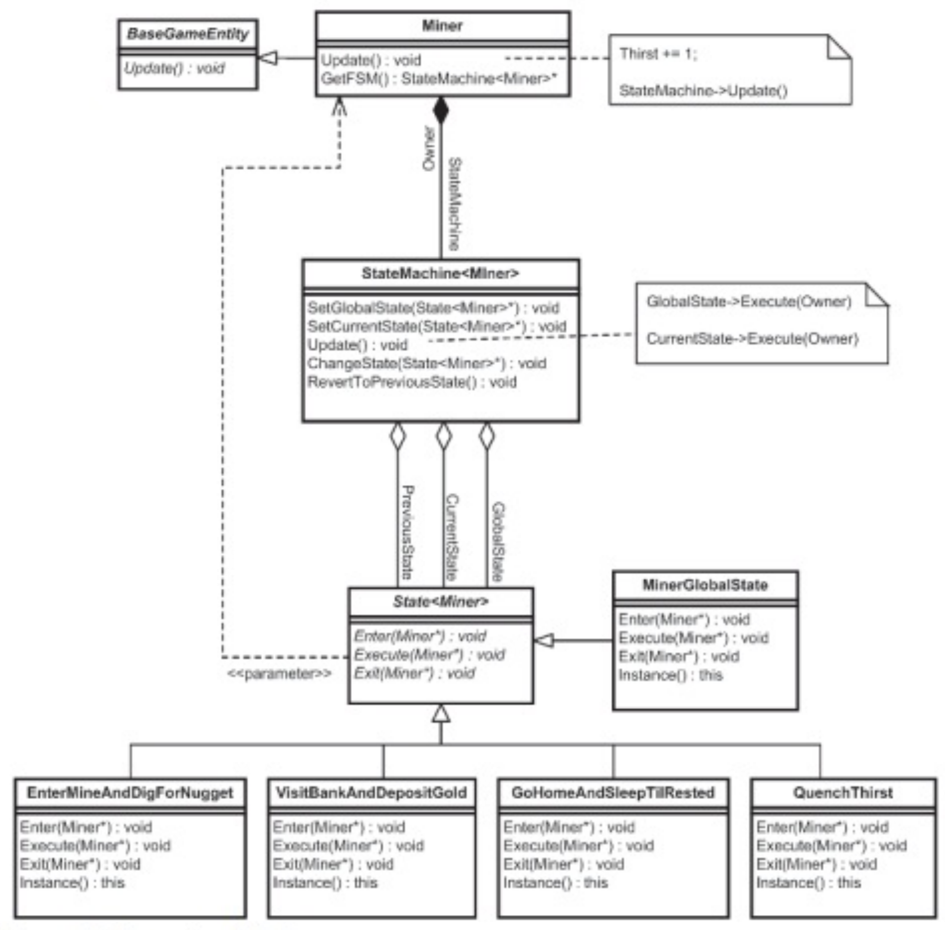


Figure 2.4. The updated design

And here's my code so far:

Code (CSharp):

```csharp
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

//Imagine that every class here only has a single instance. A singleton without the actual implementation.

public class AI : MonoBehaviour {
    Human human;
    // Use this for initialization
    void Start () {
        human = new Human(); //You are trying to create a MonoBehaviour using the 'new' keyword.  This is not allowed.  Mono

//Well, Human inherits AI which inherits MonoBehaviour.

//I thought however that inheritance only inherits and doesn't create a new instance. But now that I think about it, that woulc

//What is the work-around for this problem? I can't fathom one at the moment.

        human.Start ();
    }

    // Update is called once per frame
    void Update () {
        human.Update();
    }
}



class Human : AI{
    StateMachine stateMachine;
    internal void Start(){
        stateMachine = new StateMachine ();

        stateMachine.Start ();
    }

    internal void Update(){
        stateMachine.Update ();
    }
}



class StateMachine:Human{
    State state;
    public pointer globalState; //A pointer to the current instance of any State class that the AI is currently in.

    internal void Start(){
        state = new State ();
    }

    internal void ChangeState(/*pointer*/){ //Pointer to a state's class instance.
        //globalState = /*pointer*/
    }

    internal void Update(){
        //state.Execute(globalState);
    }
}

class State:Human{
    internal void Enter(){

    }

    internal void Execute(/*delegate*/){
        //delegate.Execute;
    }
}

class Idle:State{
    internal void Enter(){
        //Do whatever when starting this state.
    }

    internal void Execute(){
        //Do whatever Idle state does.
    }

    internal void Exit(){
        //Do whatever when exiting this state.
    }
```

So, as you can see there are 2 problems here. The pointer one(which can be solved using a string for the state and a getter that returns the instance of said state. But I didn't try doing this yet, and it wouldn't be as elegant.) and the cannot create MonoBehavior(I still want all the state classes to be able to access the MonoBehavior freely.)

So, C# Gurus, what kind of code structure would I have to implement in order to make this kind of higher structure possible? I can't think of anything that would accommodate for the structure described in the book, but then again I'm very new at C# as a whole.

Thanks.

Ecoste, Dec 24, 2014 #1

LightStriker

Joined: Aug 3, 2013
Posts: 2,198

Don't derive StateMachine from Human.

Also, ignore the notion of "pointer" in C#, everything is a pointer, you just don't have to deal with up/down casting.

LightStriker, Dec 24, 2014 #2

---

shaderop

Joined: Nov 24, 2010
Posts: 940

You're confusing yourself with the many C++-isms in the original code, e.g. trying to shoehorn pointers into a language that doesn't have such a construct (as @LightStriker) already pointed out.

Game AI by Example is an excellent book, but I would take the concepts and skip the actual implementation. You don't need to implement your own state machine code as there are already a few options out there that will do that for you. PlayMaker is one example that is built specifically for Unity and allows for visual design of states, transitions and actions.

There's also an open source library for C# called Stateless that does a great job of implementing FSMs in C#. I have used in the past with Unity and found it to be powerful and bug-free.

I also wrote my own library called TinyStateMachine which is even simpler and more lightweight (just one C# file), which is based on writing state transition tables directly in code. But I couldn't be bothered to document it except by writing a few unit tests.

TL;DR: Finite State Machines are good. Game AI by example explains the concepts well, but more elegant implementations can already be found for C#.

shaderop, Dec 24, 2014 #3

Ecoste

I see. Thanks for the replies guys.

I looked at some of the libraries and I don't think they'd suit me at the moment. They add a lot of complexity which I have no control over, and I don't understand them very well.

Joined: Nov 2, 2014
Posts: 23

I really do understand and like the structure that is in the OP. However, I have still found no way to do that. Shame indeed.

I have found other implementations, particularly in this stackoverflow thread (http://stackoverflow.com/questions/5923767/simple-state-machine-example-in-c)

I however fail to see how they're different from a bunch of 'if' and 'switch' statements.

What I like about the structure in the OP is that it is very hierarchical and thus manageable and easy to understand(for me.)

I'll post here once I find out what thing I'll do.

Ecoste, Dec 25, 2014 #4

Tomnnn

Joined: May 23, 2013
Posts: 4,138

Maybe your first a.i. should be something less complicated than a human. Try making a statemachine for something not even intelligent. How about a door or a light? It'll get you used to whatever you're unfamiliar with and get you in the right mindset to make a state machine.

I myself as a programmer for over a decade now find using PlayMaker a fun way to organize code into a series of state machines. It's cool to visualize and fun to use, and ridiculously easy to make changes to. But since that's money, here's what I would do for a door by implementing my own little state machine.

Code (CSharp):

```csharp
enum States{closed, opening, open, closing}
int doorState = States.closed;

...

switch(doorState)
{
    case States.closed:
    //execute closed code & listener for interaction
    break;
    case States.opening:
    //execute code for animating the door open, switch to open when done
    break;
    case States.open:
    //listening code for event to close door
    break;
    case States.closing:
    //code to animate door closed and switch to closed state
    break;
}
```

Tomnnn, Dec 25, 2014 #5

Ecoste

Joined: Nov 2, 2014
Posts: 23

Tomnnn said: ↑

Maybe your first a.i. should be something less complicated than a human. Try making a statemachine for something not even intelligent. How about a door or a light? It'll get you used to whatever you're unfamiliar with and get you in the right mindset to make a state machine.

I myself as a programmer for over a decade now find using PlayMaker a fun way to organize code into a series of state machines. It's cool to visualize and fun to use, and ridiculously easy to make changes to. But since that's money, here's what I would do for a door by implementing my

Click to expand...

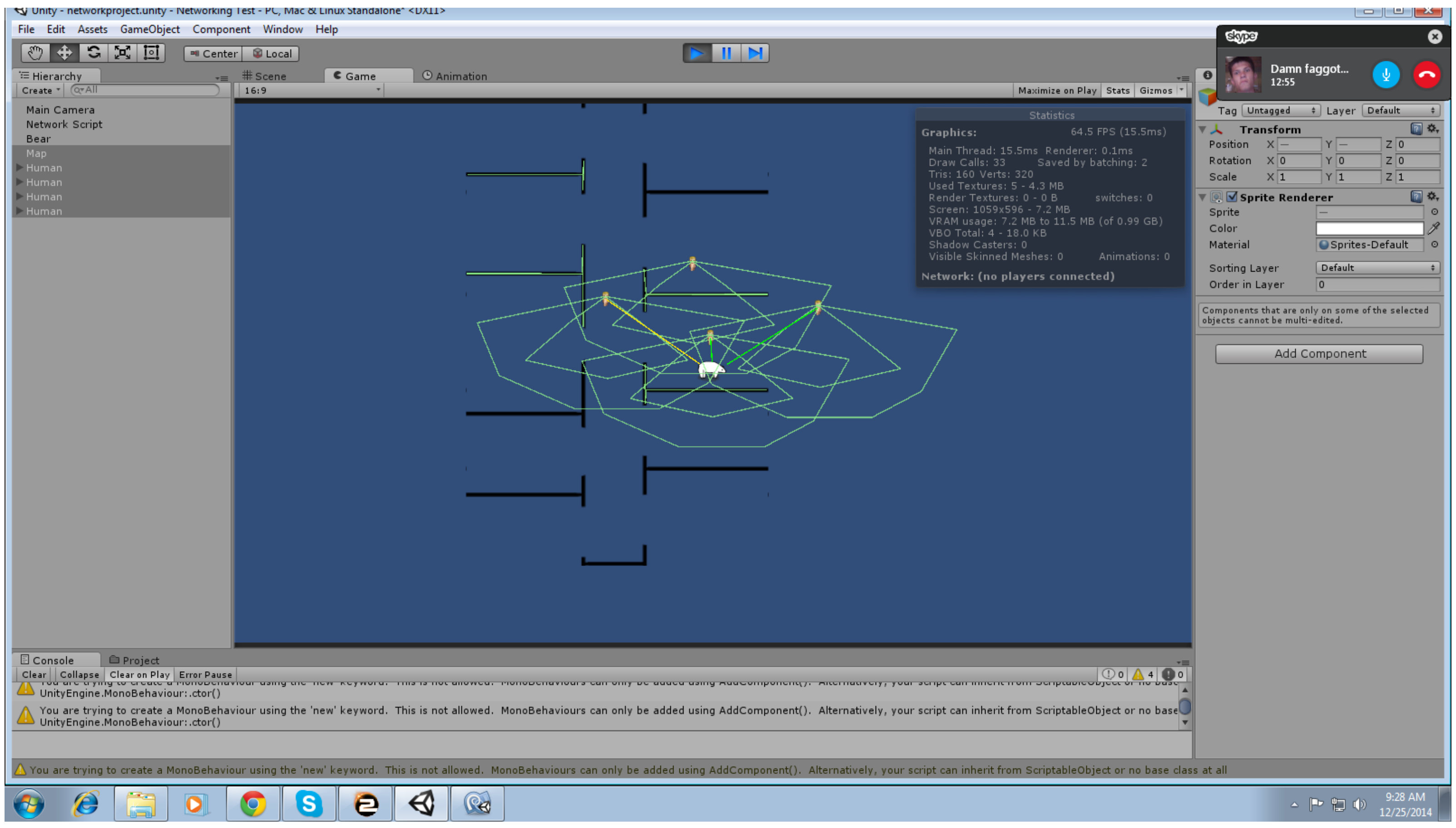
```csharp
enum States{closed, opening, open, closing}
int doorState = States.closed;

...

switch(doorState)
{
    case States.closed:
    //execute closed code & listener for interaction
    break;
    case States.opening:
    //execute code for animating the door open, switch to open when done
    break;
    case States.open:
    //listening code for event to close door
    break;
    case States.closing:
    //code to animate door closed and switch to closed state
    break;
}
```

Well, that kind of implementation is simple. I'll give it a try. Although, I wouldn't want to continue with this kind of implementation.

But, to give everyone a broad idea of what I'm doing - I'm just making a stealth-game prototype.



```
Statistics
Graphics:                  64.5 FPS (15.5ms)
  Main Thread: 15.5ms  Renderer: 0.1ms
  Draw Calls: 33         Saved by batching: 2
  Tris: 160  Verts: 320
  Used Textures: 5 - 4.3 MB
  Render Textures: 0 - 0 B          switches: 0
  Screen: 1059x596 - 7.2 MB
  VRAM usage: 7.2 MB to 11.5 MB (of 0.99 GB)
  VBO Total: 4 - 18.0 KB
  Shadow Casters: 0
  Visible Skinned Meshes: 0         Animations: 0
Network: (no players connected)
```

```
UnityEngine.MonoBehaviour:.ctor()
You are trying to create a MonoBehaviour using the 'new' keyword.  This is not allowed.  MonoBehaviours can only be added using AddComponent().  Alternatively, your script can inherit from ScriptableObject or no base
UnityEngine.MonoBehaviour:.ctor()

You are trying to create a MonoBehaviour using the 'new' keyword.  This is not allowed.  MonoBehaviours can only be added using AddComponent().  Alternatively, your script can inherit from ScriptableObject or no base class at all
```

And I have a script within each human that is called 'DetectBear', this script is in each Human. The script 'AI' is also in each human. Each human also contains 2 trigger colliders, which both contain scripts that simply return a list of objects that are within those colliders.

Code (CSharp):

```csharp

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class DetectBear : MonoBehaviour {

    private GameObject bear;
    private LayerMask mask;
    //private PolygonCollider2D polygonCol;
    private whoIsInside insideScriptImmidiate;
    private whoIsInsideLong insideScriptLong;
    private bool bearDetected = false;

    private List<GameObject> listImmediate;
    private List<GameObject> listLong;
    private RaycastHit2D ray;
    // Use this for initialization
    void Start () {
        bear = GameObject.Find ("Bear"); //Gets our player.
        mask = ~(1 << LayerMask.NameToLayer ("FoV")); //Gets FoV layer and puts it into mask so that we ignore the FoV layer.
        //polygonCol = transform.Find("FoV").GetComponent<PolygonCollider2D>(); //Gets FoV polygon within the 'FoV' gameob
        insideScriptImmidiate = transform.Find("FoVImmediate").GetComponent<whoIsInside>(); //Get who is inside script withi
        insideScriptLong = transform.Find("FoVLong").GetComponent<whoIsInsideLong>();
    }

    // Update is called once per frame
    private float timerThreshold = 3f; //Time.Delta is in seconds, so is this.
    private float timer = 0f;
    void Update () {
        listImmediate = insideScriptImmidiate.getList ();
        listLong = insideScriptLong.getList ();

        ray = Physics2D.Raycast(transform.position,
                                bear.transform.position - transform.position,
                                Mathf.Infinity,
                                mask); //Draw ray from Human to Bear.

        if(listLong.Contains(bear) && ray.collider.gameObject == bear){//If bear is in range and visible.
            if(listImmediate.Contains (bear)){
                bearDetected = true;
            } else if(listLong.Contains (bear)){
                timer += Time.deltaTime;

                if (timer > timerThreshold) {
                    bearDetected = true;
                }else{
                    Debug.DrawRay (ray.point, (new Vector3 (ray.point.x, ray.point.y, 0) - transform.position) * -1, Color.yellow);
                }
            }
        }else{
            timer = 0f;
            bearDetected = false;
        }

        if (bearDetected) {
            Debug.DrawRay(ray.point, (new Vector3(ray.point.x, ray.point.y, 0) - transform.position) * -1, Color.green);
            gameObject.GetComponent<AI>().setState(AI.States.panick);
        }
    }
}

```

This basically detects the bear in one of the two ways. And as you can see, it also has an if statement at the end that just draws a green ray if the bear is detected, which also changes the state of that particular human. In the future, I'd like a 'global' controller script that will set the state of a human based on its instance id. But, that is not needed right now.

And here's the AI script:

Code (CSharp):

```

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class AI : MonoBehaviour {
    public enum States{idle, panick}
    States state = States.idle;
    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

        switch (state) {
            case States.idle:
                Debug.Log ("I'm idle.");
                break;
            case States.panick:
                Debug.Log ("I'm panicking.");
                break;
        }

    }

    public void setState(States s){
        state = s;
    }
}
```

I like this implementation, except for the switch statement. I want small, manageable chunks of code that are easy to follow.

So, this AI structure and path finding is left. But, I'm stuck at the AI structure right now, especially since I've never done this type of thing before.

Ecoste, Dec 25, 2014 #6

Tomnnn

Joined: May 23, 2013
Posts: 4,138

Finite State machines are all about them switches, bro

Your code will not get more complicated than you see it there. Another class can contain definitions for what needs to be done by each state, and your class you have there can just be the state machine itself. The key will be to design your algorithms to run in single steps, to also match the design of state machines.

Action
Goal test
Success / fail / other event event where you switch states and set values

Tomnnn, Dec 26, 2014 #7

Kiwasi

Joined: Dec 5, 2013
Posts: 14,025

I implemented a FSM based on the concepts in Programming Game AI By Example. From memory the main point of his code was to keep the states as separate classes from the state machine. It went something like this. Be aware this is all just from memory, and may not work properly.

The state machine code. I seem to recall needing to abstract this one step further for flexibility. IE this was a regular C# class owned by a MonoBehaviour. But you should be able to see the general structure here.

Code (CSharp):

```csharp
public class StateMachine : MonoBehaviour {
    private State currentState;

    public void SwitchState (State newState){
        currentState.EndState(this);
        currentState = newState;
        currentState.BeginState(this);
    }

    void Update (){
        if (currentState != null){
            currentState.UpdateState(this);
        }
    }
}
```

Here is the base class for a State.

Code (CSharp):

```csharp
public abstract class State {
    public abstract void BeginState(StateMachine stateMachine);
    public abstract void UpdateState(StateMachine stateMachine);
    public abstract void EndState(StateMachine stateMachine);
}
```

Making the individual states is simply a case of inheriting from the base class.

Kiwasi, Dec 26, 2014 #8

jister likes this.

LightStriker

Joined: Aug 3, 2013
Posts: 2,198

Tomnnn said: ↑

Finite State machines are all about them switches, bro

No, it's not. I never have switch/case in my state machine. Each state is a class, and it's to each state to determine how and when to change to another state.

LightStriker, Dec 26, 2014 #9

jister and Kiwasi like this.

Kiwasi

LightStriker said: ↑

No, it's not. I never have switch/case in my state machine. Each state is a class, and it's to each state to determine how and when to change to another state.

That is very true. An abundance of switches is the quickest way to build an unmaintainable state machine.

Kiwasi, Dec 26, 2014 #10

Joined: Dec 5, 2013
Posts: 14,025

jister likes this.

Tomnnn

Joined: May 23, 2013
Posts: 4,138

@LightStriker I've been under the impression that a simple finite state machine can be implemented with a switch statement. I learned from reading examples (FSMs implemented in C, C++ and Java using switches) and also, a switch seemed to be a sensible way to turn a diagram for an FSM into code.

I've made classes just to group relevant methods together, but using those classes as the states sounds like transitions / expanding transitions would be a pain. Wouldn't they need a reference to each other since the Type is going to be that class?

@BoredMormon is that so? I've yet to have a problem. How can someone get disorganized if they're using enums?

--edit

Read your post that I missed after mine. I see, so the state-class thing is the implementation style for that author (maybe others? never heard of it myself). I'll look into that. I should know these things if I'm going to be a teacher in 3 years, haha.

It's so easy to find implementation examples that use switches. My learning (for most things) came from tearing apart examples, I've never read the 'proper' ways in how-to books.

Last edited: Dec 26, 2014

Tomnnn, Dec 26, 2014 #11

Kiwasi

Joined: Dec 5, 2013
Posts: 14,025

Switches are fine for a machine with 3 or 4 states. However try implementing a nested state machine with a dozen states on each level, plus global transition, state entrance and exit behaviour and multiple state variables. Do this all with switch statements. Then go back and change the behaviour inside one state completely six months later. You'll get the idea.

Having each state as a separate self contained script has several advantages

- Easier to read an entire states functionality in one place
- The base state machine can be reused every where. Simply plug in a new set of states and you are good to go
- The code complexity does not increase significantly with extra states
- Changing one state does not effect the others. The system is modular enough that you can replace a state entirely without effecting the system.

There are several other ways to implement a state machine, including graphical (with a tool like playmaker) and an event based system (Unity gems has a good frame work). There are others.

As with any frame work there is a trade off between functionality and complexity. For simple jobs the complexity of a fully fledged state machine is not warranted. If you are making a space invaders style AI then a switch statement is probably good enough. But if you want something like Skyrim then you'll need to spend a fair bit more time on your frame work.

Kiwasi, Dec 26, 2014 #12

LightStriker

Joined: Aug 3, 2013
Posts: 2,198

It really doesn't take much for a switch/case state machine to become a bloated unmaintainable mess. You ends up having a bit of everything all over the place, methods off different state near each other, and a very hard time to insert or remove a state. This kind of state machine is fine when you know exactly what you need, and it is very unlikely to change later - which the design never does, right?

In one game, I even did a door in object-oriented FSM, and it helped... when the design decided the door could be blocked, slammed, crush the player, destroyed... and not just open, opening, closed, closing.

Once you know how an OO FSM works, which is dead easy really, you won't go back. OO FSM really starts to shine when the complexity increases, like with any kind of AI. Using polymorphism for your state is a blessing... By example, if you have a collection of state in the air, a collection on ground, a collection underwater, etc. And then you have player, NPC, enemy, and other that can all share the same states, which is simply impossible with switch/case.

LightStriker, Dec 26, 2014 #13

Tomnnn

Joined: May 23, 2013
Posts: 4,138

@LightStriker I'm sure I'll figure it out when I get that far. My plan for my doctorate, if I can go with the option, is to make a complex state machine to be a good a.i. of my mother's favorite dog who will be near death at that point in time. As for your door example, isn't expanding a switch statement as easy as adding the case to the switch and appending a new state to the enum?

In the event to switch crushed, I would set the state to States.crushed. What you said about them sharing states made me realize my misunderstanding... that's just because your states are methods xP You can share them because they *are* the behaviors. If that's the point, then I get it now. My [probably outdated] idea of a state machine is that a state is more like a label to group actions together with. I can see now that the correct idea for a state machine is that the states are the actions? Or are both valid but the state-class-function idea scales better?

@BoredMormon I have done such with a switch I went back to a horror game from when I was still new to unity. I added a state to the monster behavior between tracking and killing the player. I added 1 more case to check for the final listen state in which the player might escape if they make no noise.

Maybe I made a flaw in designing my state machine and violated a convention. By making the current state a public variable I could just set it into another state from any point by accessing that variable in the script. And for your fourth point... if you're adding a new state to happen between two other states, don't you have to modify the events to transition into that state? Sounds about the same work for having my function choose the new state instead of the previous one. If this should be obvious, then it's probably a flaw with switch based finite state machines that I overcame by violating a convention for making them I wouldn't know, my top concern has been making things work above all else.

Tomnnn, Dec 26, 2014 #14

Kiwasi

Joined: Dec 5, 2013
Posts: 14,025

If it works for you, then fair enough. But my experience is that switches turn to spaghetti very quickly.

Kiwasi, Dec 26, 2014 #15

LightStriker

Joined: Aug 3, 2013
Posts: 2,198

Tomnnn said: ↑

@LightStriker I'm sure I'll figure it out when I get that far. My plan for my doctorate, if I can go with the option, is to make a complex state machine to be a good a.i. of my mother's favorite dog who will be near death at that point in time. As for your door example, isn't expanding a switch statement as easy as adding the case to the switch and appending a new state to the enum?

In the event to switch crushed, I would set the state to States.crushed. What you said about them sharing states made me realize my misunderstanding... that's just because your states are methods xP You can share them because they are the behaviors. If that's the point, then I get it now. My [probably outdated] idea of a state machine is that a state is more like a label to group actions together with. I can see now that the correct idea for a state machine is that the states are the actions? Or are both valid but the state-class-function idea scales better?

I guess I'm not explaining myself properly... A state is not a single action; it has properties, it may have multiple methods, and by experience it is very helping to have a kind of constructor/destruction for when the state starts and when it stops. It's never "as easy as adding an entry in an enum".

They can be shared not because they are method, but because they are object. Each AI can have its own instance of "NPCWalkState", while feeding it with its own parameter, like where a NPC can walk, at what forward speed, what turn speed, etc.

Here's an example of the class hierarchy used in a game I worked on;

Character > Player
Character > NPC
Character > Enemy

State > CharacterState > GroundState > WalkState
State > CharacterState > GroundState > RunState
State > CharacterState > GroundState > SlideState
State > CharacterState > GroundState > StumbleState
State > CharacterState > GroundState > RespawnState
State > CharacterState > AirState > JumpState
State > CharacterState > AirState > DoubleJumpState
State > CharacterState > AirState > DashState
State > CharacterState > AirState > SlamState
State > CharacterState > AirState > DeadState
(and 15 other states like that)

Player, NPC and Enemy have different stats and behaviour. However, they all can use those states;

Code (CSharp):

```
machine.SwitchState(new JumpState(machine, character));
```

Example of a state;

Code (CSharp):

```
public class JumpState : AirState
{
    // Velocity, jump height, and other parameter

    public JumpState(StateMachine machine, Character character)
        : base(machine, character) { }

    public override void Constructor()
    {
        base.Constructor();

        Character.Animator.SetTrigger("Jump");
        Character.Visual.FXHandler.Jump();

        if (Character is Player)
            StatsManager.Instance.Jumped();
    }

    public override void Update()
    {
        UpdateBallisticCurve();

        // Evaluate other possible action in mid air
        if (Character.Jump()) // The character has its own input method for deciding an action
            Machine.SwitchState(new DoubleJumpState(Machine, Character));

        if (Character.Dash())
            Machine.SwitchState(new DashState(Machine, Character));

        if (Character.Slam())
            Machine.SwitchState(new SlamState(Machine, Character));

```

Last edited: Dec 26, 2014

LightStriker, Dec 26, 2014 #16

Tomnnn

Joined: May 23, 2013
Posts: 4,138

BoredMormon said: ↑

If it works for you, then fair enough. But my experience is that switches turn to spaghetti very quickly.

Execution wise? Without a doubt lol. If the behavior is very complex and cannot be broken up, it's entirely up to knowing what enum to search for in your giant switch xD But if the methods are broken up and designed like PlayMaker state machines, your class can be just the switch. Then if you've got a case, method call and break, managing 100 states would be 300 lines.

@LightStriker it's still sounding like a design thing to me. That point regarding the ease of expansion sounds unfortunate. I'll ask a few of the cs graduates about state machines, they might be able to explain it better since they know me lol.

Tomnnn, Dec 26, 2014 #17

Ecoste

Joined: Nov 2, 2014
Posts: 23

Code (CSharp):

```

using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class AI : MonoBehaviour {
    StateMachine stateMachine;

    void Start () {
        stateMachine = new StateMachine(this);
        stateMachine.addState ("Idle", new State_Idle (this));
        stateMachine.addState ("Panick", new State_Panick (this));
    }

    void Update () {
        stateMachine.Update ();
    }
}

public sealed class StateMachine{
    AI ai = null;

    public StateMachine(AI t)
    {
        ai = t;
    }

    ////////////////////////////////////////////////////////
    public AI GetAI(){
        return ai;
    }
    ////////////////////////////////////////////////////////
    Dictionary<string, State> states = new Dictionary<string, State>();

    public void addState(string s, State o){
        states.Add(s, o);
    }

    public object getState(string s){
        return states[s];
    }

    public void Update(){
        foreach(KeyValuePair<string, State> state in states)
        {
            state.Value.Update();
        }
    }
}

public abstract class State
{
    public string Name { get; set; }

    public abstract void Entry();
    public abstract void Update();
    public abstract void Exit();
}

class State_Idle : State{
    string Name = "Idle";
    AI ai = null;

    public State_Idle(AI t){
```

This is my solution thus far. I'm happy with in, and I hope it will be sustainable and won't need any changes in the future.

This is only the bare-bones structure, I obviously didn't add anything to it yet.

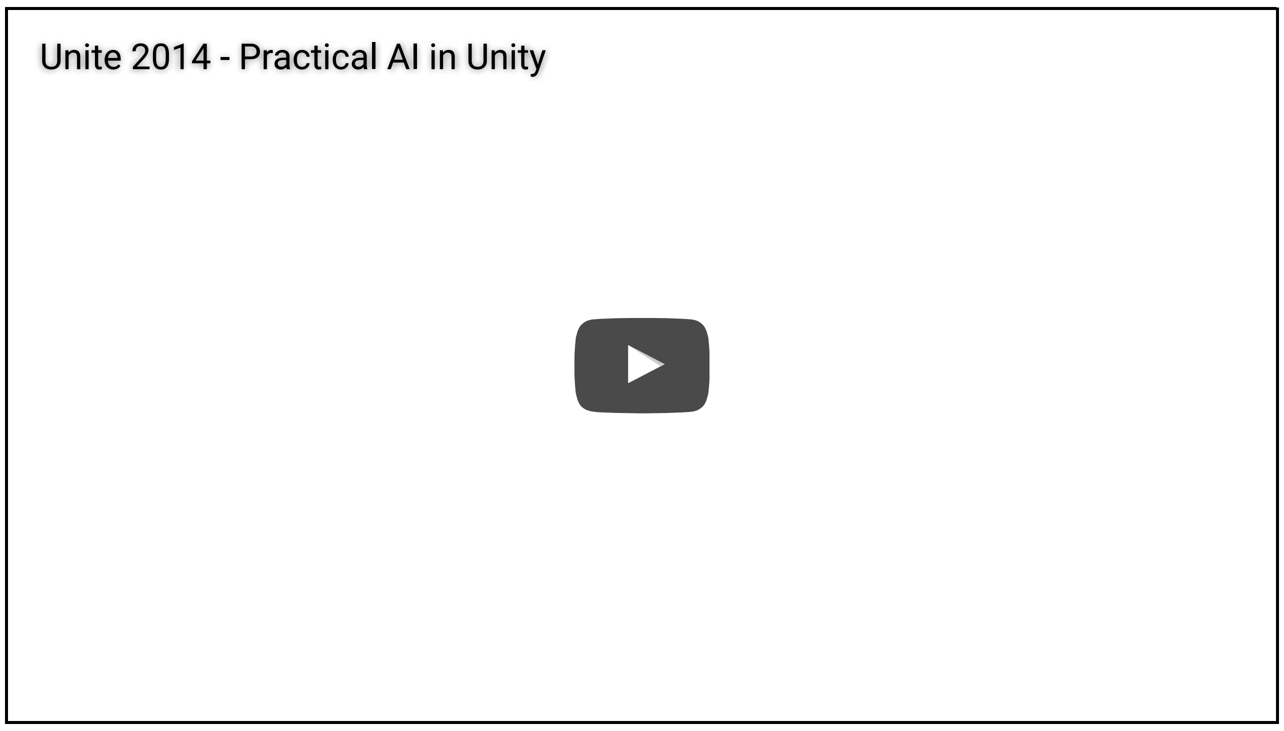
Last edited: Dec 27, 2014

Ecoste, Dec 27, 2014 #18

---

rakkarage

Joined: Feb 3, 2014
Posts: 506



rakkarage, Dec 27, 2014 #19

Kiwasi

Joined: Dec 5, 2013
Posts: 14,025

Ecoste said: ↑

This is my solution thus far. I'm happy with in, and I hope it will be sustainable and won't need any changes in the future.

How long have you been in this business for? Everything requires changes

Kiwasi, Dec 27, 2014 #20

Ecoste

BoredMormon said: ↑

How long have you been in this business for? Everything requires changes

I've never been in the business):

Ecoste, Dec 27, 2014 #21

Posts: Nov 2, 2014 23

Tomnnn

Joined: May 23, 2013
Posts: 4,138

> Ecoste said: ↑
>
> *I've never been in the business):*

Knowledge of state machines will be a good resume piece for getting into that business.

Tomnnn, Dec 27, 2014 #22

Kiwasi

Joined: Dec 5, 2013
Posts: 14,025

Actually reading your code this time there is nothing that jumps out at me as bad. You may want to include BeginState and EndState functionality as you go. You also need to define transitions between states.

Kiwasi, Dec 27, 2014 #23

LightStriker

Joined: Aug 3, 2013
Posts: 2,198

I would dump the dictionary and the strings. Why would you need those?

LightStriker, Dec 28, 2014 #24

Kiwasi likes this.

Kiwasi

Joined: Dec 5, 2013
Posts: 14,025

Strings are a double edged sword. On the one hand the offer flexibility that is not otherwise easy to produce. But typos in string introduce hard to spot bugs. In general the conventional wisdom is to avoid strings.

Kiwasi, Dec 28, 2014 #25

Ecoste

Joined: Nov 2, 2014
Posts: 23

> LightStriker said: ↑
>
> *I would dump the dictionary and the strings. Why would you need those?*

My alternative to that would be to just create new states every time instead of reusing them. Is that better?

Ecoste, Dec 28, 2014 #26

---

fire7side

Joined: Oct 15, 2012
Posts: 616

> *My alternative to that would be to just create new states every time instead of reusing them. Is that better?*

You'll basically be doing that anyway. I haven't bumped up against it yet, but I think I'll just attach scripts as states and enable and disable. Coroutines that check every second would probably be all that's necessary, and the code is going to be unique for the type of npc. Events are more important.

fire7side, Dec 28, 2014 #27

LightStriker

Joined: Aug 3, 2013
Posts: 2,198

> Ecoste said: ↑
>
> *My alternative to that would be to just create new states every time instead of reusing them. Is that better?*

It's not like you change state every frame, so the garbage collection should be a non-issue.

LightStriker, Dec 28, 2014 #28

Kiwasi

Joined: Dec 5, 2013
Posts: 14,025

There are multiple different ways to handle the states. You can use a dictionary and specific state instances if the states need to remember their own data. There is a memory cost to doing this.

You can create new states, there will be a performance and gc cost to soon this. There will also be difficulty storing data between states.

Static states save on memory and GC. But they cannot store any data.

Ultimately you choose what structure works for your project.

Pro tip for working with strings: whenever possible use a variable instead of a hard coded string. In your case State.name. This will dramatically reduce typos.

Kiwasi, Dec 28, 2014 #29

NickArgall

Joined: Dec 21, 2014
Posts: 2

I note that the more experienced programmers are talking about how difficult it is to read a piece of code, while the less experienced programmers are talking about how difficult it is to write a piece of code. Over the long term, 'readable' becomes progressively more important.

The reason to use an Enum instead of a string when possible is to get compiler errors instead of runtime errors. Simple compiler errors are typically fixed in seconds. Simple runtime errors are typically fixed in minutes. Every programmer has a non-zero error rate, and it has an embarrassing tendency to be stable, even though we learn and improve.

NickArgall, Dec 30, 2014 #30

LightStriker

Joined: Aug 3, 2013
Posts: 2,198

NickArgall said: ↑

> *The reason to use an Enum instead of a string when possible is to get compiler errors instead of runtime errors. Simple compiler errors are typically fixed in seconds. Simple runtime errors are typically fixed in minutes. Every programmer has a non-zero error rate, and it has an embarrassing tendency to be stable, even though we learn and improve.*

And the reason to use classes over enum is for flexibility, maintainability and reusability. Viva el polymorphism!

LightStriker, Dec 30, 2014 #31

NickArgall

Joined: Dec 21, 2014
Posts: 2

LightStriker said: ↑

> *And the reason to use classes over enum is for flexibility, maintainability and reusability. Viva el polymorphism!*

Sorry, yes, you're absolutely right. I'm a huge fan of the Strategy pattern, which is what the OO-based approach to state representation seems to be based on. I was weighing in on the discussion about 'strings' without really paying attention to the code sample. (Post #18 is the one where strings entered into the discussion.) Now that I've gotten over my 'half asleep posting' fail, my comment on post #18 is:

If your states have a Name property, then using a dictionary where you add the name as a separate variable compromises maintainability because you have two different entities that are storing the same information. Better to simply use a list and use StateList[index].Name if you want the name of one of the entities in the list.

NickArgall, Dec 30, 2014 #32

(You must log in or sign up to reply here.)