Search tutorials...

Tutorial Downloads

File	Size
Follower FSM example.capx	175KB

Translations

Portugese (Brazillian) (*pt-br*)

Know another language? Translate this tutorial!

Stats

3.3K visitors
13.2K page views
175 translation visitors
218 translation page views

How to simplify your AI code with Finite State Machines

Tutorials > Intermediate > AI > *How to simplify your AI code with Finite State Machines*

Tutorial written by Valerien
Originally published on 25th, August 2014 - 3 revisions

Favourite
56 favourites

This article will introduce you to Artificial Intelligence in video games. In this tutorial, you'll learn what Finite State Machines are, and how to apply this powerful concept to your construct 2 AIs. But let's start off with a quick overview of game AI. You will need the Platform moveTo behavior from rexrainbow in order to open the attached capx. Extract the archive and put the behavior's folder into your Program Files\Construct 2\exporters\html5\behaviors folder. And if you like what I'm doing, you can follow me on twitter and on facebook!

A short introduction to game AI

In video games, artificial intelligence is used to **build and describe the behaviors of elements that move and act by themselves, be it a non-playable character, a seeking rocket…** Once applied to games, artificial intelligence becomes more artificial than smart: **our aim is to provide the player with compelling obstacles, to spice up his experience. It is not about simulating self-aware, human-like intelligence.**

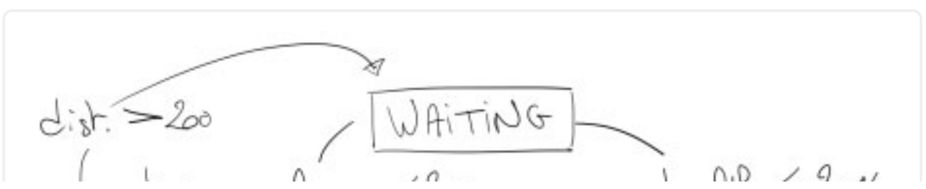
Actually, gameplay-wise, it's better if the AI isn't too smart: in an infiltration game, you may have noticed how guards can forget you and get back to their patrol after a while (cf. Mark of the ninja, Metal Gear Solid). In games that include real time fighting mechanisms, often only one or two foes will attack you at once, making the fight manageable (Dishonored, Assassin's Creed). **Enemies rarely anticipate the player's next move, but will rather react to his previous assaults. This is why, in essence, game AI is more artificial than intelligent.**

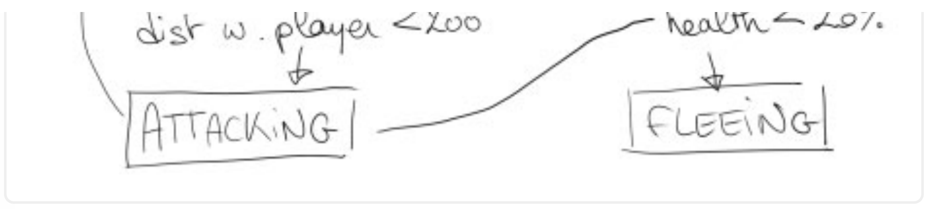
Really basic AI, we can manage with plain, intuitive logic. But by now, you may have noticed that it can get tricky to manage a growing list of behaviors with standard conditions and actions in Construct 2. It gets tedious to edit and extend really fast! We can use a powerful, yet accessible concept to model AI with multiple states or behaviors: **Finite State Machines**.

What's a Finite State Machine?

Finite State Machines (FSM), or **State machines** are models of computation used to abstract an object or machine in such a way it can only be in a unique state at a time, picked in a set of states.

It should be clearer with a little illustration. Let's say we have an enemy AI, in an RPG game. We want it to attack the player when the player gets close. When it has a low health however, we want to flee. Our AI's behavior can be modeled as a set of three states: it can be attacking, fleeing, or waiting.





The boxed text represents a state, while the arrows and floating text represent the conditions and and directions of the logic's flow. At any time, our AI can only be in one state, and can only move from one state to another. The idea is, as far as your main programming loop is concerned, to only tell the engine what state your character or object should be in. Each state becomes a self-contained, unique bit of code that you can edit any time, without risking to break your game!

4 reasons to use Finite State Machines

They are easy to implement and manage

Finite State Machines are the first step towards elegant AI development. This programming model is one the simplest concepts to implement and to manage: it is close enough to using basic if/else statements, yet much more flexible. It basically makes your AI easy to modify and extend, even when it gets big.

They are accessible and intuitive

FSMs aren't abstract at all: it's quite natural to think about an artificial intelligence as an entity with a set of behaviors or states.

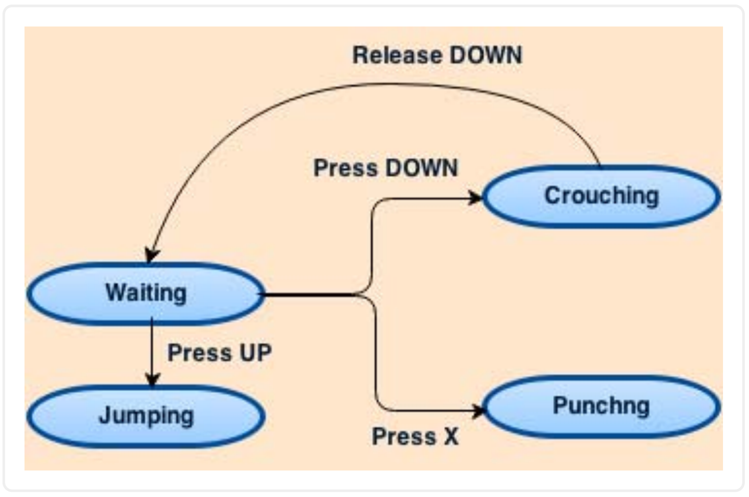
They cover many AI needs

Although they have their limits, FSMs **will cover most of your AI needs for 2d game development**. It can easily model a variety of enemies, characters, and even objects (for example, a tracking rocket with 3 states: "launching",

Index

They can model even UI or control schemes

As I wrote earlier, FSMs can also be used to model objects, **but also UI or control schemes!** They are actually quite powerful when it comes to representing a player's set of moves. I'll leave you with an example of a basic platformer input FSM.

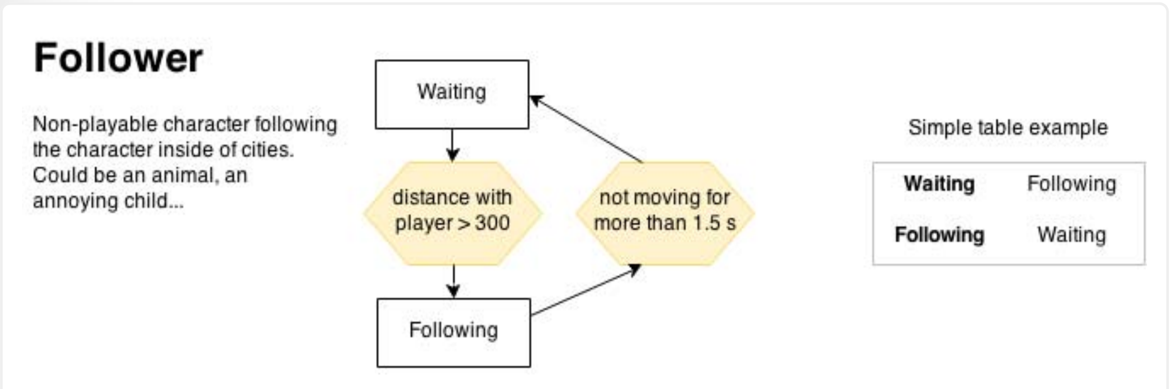


Planning your FSM AI

Before programming any artificial intelligence, it is absolutely crucial to plan it carefully: the more precise and coherent your planning is, the more time you will save while coding. Finite State Machines are generally represented and built using a graph. Graphs capture very well the flow and restrictions of an FSM.

There are many flowcharts or graph drawing tools, starting with pen and paper. You could use Office Visio, LibreOffice draw, or an online alternative like Draw.io. For simpler graphs, a pencil is more than enough.

Your planning has to capture the states, the flow of the states, and the conditions of flow of your FSM-based AI. You can write it in any detail you are comfortable with. Here's an example of how I would go with a simple AI. I chose to think about a follower type of non-playable character. It follows the player and has 2 states: waiting and following, as represented in the graph below. A table sums up the flow of our FSM.


Follower
Non-playable character following the character inside of cities. Could be an animal, an annoying child...
Waiting
distance with player > 300
not moving for more than 1.5 s
Following
Simple table example

Waiting	Following
Following	Waiting

Open full size image

How to implement the follower FSM in construct 2 (by example)

The process is shown and described in the **commented capx example attached to this tutorial** (*you can find it in the top-left portion of the page*). The AI-related code is basically split into two major blocks:

- A list of conditions, used to switch states
- A list of functions, one per state
This way, our AI can be easily extended with new behaviors, like sleeping or scouting the area!

Last words

Would you like me to cover something specific? **Drop me a message!** You can follow me and ask me questions directly on twitter, facebook and google plus. [illegible]ials about game design directly on my website: http://GDquest.com/ . Finally, you can find all of my Construct 2 tutorials on this forum post.

Index

Unlock your full gamedev potential

Upgrade to the Personal Edition of Construct 2, it has way more features and won't holding back from making money and using your full creativity like the free edition does. It's a one off payment and all Construct 2 editor updates are free for life!

Plus, it's got a lot of additional features that will help you save time and make more impressive games!

View deals

Congratulations on finishing this tutorial!
Did you learn a lot from it? Share it now with your friends!

Tweet 2 Like G+

Comments

StefanSava 1,160 rep 1

Nice tutorial! It would be cool if you removed the MoveTo Behaviour and leave soemthing generic construct already has. It's a bummer having to search and install that behaviour just to try out the tutorial file.

about a year ago

yrragjerios 231 rep

I have an error it says "Cannot find action with the ID 8 in the behavior 'MoveTo' (Platform-MoveTo)

about a year ago

KoolEcky 16.2k rep

0

Valerien, excellent tutorial on AI using FSMs.

10 months ago

1

Leave a comment

Everyone is welcome to leave their thoughts! Register a new account or login.

Index