Push or Pull? Turning keypresses into velocity for in-game vehicles

Asked 16 years, 2 months ago Modified 16 years, 2 months ago

Viewed 593 times

4

Should I push keypresses to vehicles when they're pressed, or should vehicles pull keys pressed from the engine?

I have a vehicle object, which has location, velocity and accelleration members (among other things) and an update method, during which it updates its location based on its velocity, and its vevlocity based on its accelleration.

I have a game object which contains the game loop, which calls the update method on the vehicle.

If the player controls the vehicle with the arrow keys, should a keypress set the accelleration (push) and a key-release clear the velocity, or should the vehicle ask the game-engine if the accellerate key is pressed (pull)? I think a push would mean that the keyboard control module would need to know about vehicles, whereas pull would mean a vehicle needs to know specific keyboard controls.

I think a related question would be something like: should all objects know about all other objects, or should there be a strict hierarchy, so objects can ask things / tell things to other objects up the tree, but not down (or vice-versa)?

language-agnostic

Share

Improve this question

Follow

asked Sep 28, 2008 at 16:58

Tim Gradwell

2,402 ● 5 ● 25 ● 25

5 Answers

Sorted by: Highest score (default)

8

You should try and follow an [Subscribing/Observer]() pattern. You put all the key capture code into one [singleton]() InputManager and then each object that requires reaction to input registers with the manager.

The manager holds the list of subscribed objects and sends events to them when the keys are pressed/depressed. Just don't forget to unsubscribe when the object is deleted or 'loses focus'.

This avoids the polling problem. There are very few exceptions where a polling solution is desirable.

Share Improve this answer

Follow

edited Oct 2, 2008 at 20:08

answered Sep 28, 2008 at 17:04

Caerbanog
1,335 ● 1 ● 12 ● 18

Say i have two keys that do the same thing (say, WASD and Arrows keys both accel the vehicle). in order to avoid 2x accels, you have to have a bit in the vehicle 'have i acceld due to keypress this frame' (lame) or you need to put the key->game command logic in between (which has exceptions itself). – Jeff Oct 2, 2008 at 6:35

not really jeff... if (key == key_A) accelleration = 10; if (key == upArrow) accelleration = 10; Or have I missed your point? – Tim Gradwell Oct 2, 2008 at 13:21

yes, you turned the operation of the key from: V = V+10; into A = 10; and deferred V+=A; This is exactly the 'bit in the vehicle that says i've acceled this frame'. Now if you have another key for 'brakes' you're now stuck either at A=10 or A=-10 (unless you go and add more state A_braking. – Jeff Oct 2, 2008 at 15:52

(and even worse, the + or - 10 is dependant on the order of registering the observers!) – Jeff Oct 2, 2008 at 15:55

3

IMO, your vehicle shouldn't know anything about keyboards, mice, or gamepads. And neither should your input handling code know anything about your vehicles. The input handling code should read the input for each player, and translate it into some sort of instruction specific for their context. For example, if player one is driving a car, his instruction might include steering wheel rotation, acceleration, and brake values. While a player piloting a plane might require pitch, yaw, etc.

By translating the gamepad input (or whatever) to the appropriate instruction type, you can decouple input mechanisms from game logic. One thing that would be possible with this level of decoupling would be to create a "CarInstruction" from network input.

Share Improve this answer Follow

answered Sep 29, 2008 at 13:21

Joel Martinez
47.7k ● 26 ● 134 ● 185

1

Answering this question is hard without more intimate knowledge about how your game engine works. That being said, I'll take a stab at it. The "push keyboard presses" approach reads to me like an "event" or "callbacks" strategy. You define a function somewhere that looks like `def handle_key_event(name_of_key):` that gets called whenever a key event occurs. The advantage of this is that from a readability perspective, you know exactly where key events are being handled. On the downside, each key press needs to be treated as an atomic operation. If you need to keep lots of state variables around on the state of other keys to determine what to on each press, it can get a little messy.

On the other hand, if you pull key presses, you introduce an inherent delay in catching key presses. You won't catch key events any faster than your tickrate/framerate. This is fine if your game is ticking away nice and fast, but you don't want to have the UI become all jumpy/laggy when your framerate slows down.

Just food for thought, I guess. Above all, pick a strategy and stick with it. If keyboard events are a callback, don't use the "pull" approach for your mouse events, for instance. Consistency is more important than correctness here IMO.

Share Improve this answer Follow

answered Sep 28, 2008 at 17:06

community wiki
Ryan

1

@Joel: I agree - vehicles shouldn't know about specific hardware controls, and input handling code shouldn't know anything about vehicles. There should be some intermediate class which maps from keys to vehicles. Thanks for the contribution!

Share Improve this answer Follow

answered Sep 29, 2008 at 14:05
Tim Gradwell
2,402 ● 5 ● 25 ● 25

1

you want to poll:

```
void UpdateVehicleFromInput()
{
   if (InputSystem()->IsKeyDown(key))
      DoSomething();
}
```

And this is of course 'somewhere in your update loop where appropriate for you're design' If you want to call that particular place 'part of your input system' or 'part of your game logic' or whatever else, knock yourself out.

This way you know why your doing something (cause key is down), you can adjust the conditions trivially, you know you're doing something once and exactly once (and you can change it w/o ramification, particulary if the vehicle doesn't exist), and you know when you are doing something (before or after you say, respond to damage, or position your particle effects, or who knows what else).

Abstracting the input system can be valid iif you really are doing cross platform development. for casual development, it's very unnecessary (but a fun technical distraction when you run out of game design ideas to implement).

Contrary to irrational popular belief, there's no downside to polling. Processors do > 1B things a second, one IF a frame is irrelevant (pretty much the only relevant cpu operations are N^2 where N>100 and blowing your l2 cache and of course busy waiting for disk access). Polling input is O(1).

Share Improve this answer Follow

edited Oct 2, 2008 at 6:52

answered Oct 2, 2008 at 6:47

Jeff

1,053 ● 1 ● 8 ● 14