Published in Towards Data Science

Debby Nirwan Follow

Jan 16, 2021 · 5 min read ★ · Listen

GETTING STARTED

How I Implemented HFSM In Python

Learning the step-by-step implementation of HFSM concepts in Python.

```
49      def on_entry(self, callback: Callable[[Any], None]):
50          self._entry_callbacks.append(callback)
51
52      def on_exit(self, callback: Callable[[], None]):
53          self._exit_callbacks.append(callback)
54
55      def set_child_sm(self, child_sm):
56          if not isinstance(child_sm, StateMachine):
57              raise TypeError("child_sm must be the type of StateMachine")
58          if self._parent_state_machine and self._parent_state_machine == \
59                  child_sm:
60              raise ValueError("child_sm and parent_sm must be different")
61          self._child_state_machine = child_sm
62
63      def set_parent_sm(self, parent_sm):
64          if not isinstance(parent_sm, StateMachine):
65              raise TypeError("parent_sm must be the type of StateMachine")
66          if self._child_state_machine and self._child_state_machine == \
67                  parent_sm:
68              raise ValueError("child_sm and parent_sm must be different")
69          self._parent_state_machine = parent_sm
70
71      def start(self, data: Any):
72          logging.debug(f"Entering {self._name}")
73          for callback in self._entry_callbacks:
74              callback(data)
75          if self._child_state_machine is not None:
76              self._child_state_machine.start(data)
```

HFSM in Python (Image by Author)

Tree. In my experience, after reading books and papers of some technical topics, my comprehension of the topics would greatly improve if I implemented them in code.

Pacman AI is written in Python and so I tried searching HFSM implementation on Github, but couldn't find a good one for my implementation and decided to write one and release it on Github.

In this article, I want to share what I have learned when implementing it. We will see the step-by-step transformation from concepts to code. Hopefully, this is useful for you and encourages you to code and share your works.

If you are not familiar with HFSM y[111 | 1] to read my previous post below:

Hierarchical Finite State Machine for AI Acting Engine

Learning the details of Hierarchical Finite State Machine (HFSM), how it solves issues found in Finite State Machine...

towardsdatascience.com

Let's get started.

Step-by-step Implementation

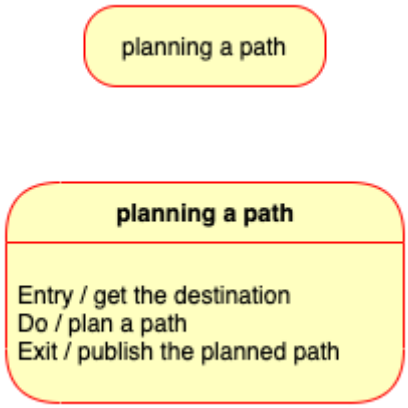
An HFSM consists of multiple FSMs, so we will start with implementing an FSM.

FSM Implementation

An FSM consists of the following building blocks:

- State
- Event
- Transition

We start with the first building block, state. Here is one example of a state:



A State (Image by Author)

A state can be as simple as having just a name, but it can also have some actions associated with it such as entry, do, and exit. We can create a class of a state as follows.

State Class (Code by Author)

Our state class simply has a name, which is important to distinguish one state from another which we use for equality check in the code snippet above.

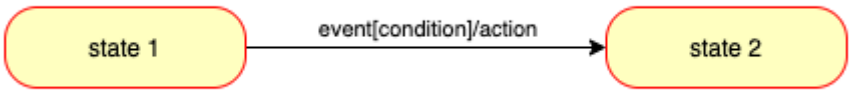
We also have two lists of callbacks, for entry and exit which can be set by ***on_entry()*** and ***on_exit()*** functions. ***do*** action is not implemented because it is easier to implement it as an ***action*** in Transition which we will turn to in a moment.

start() and ***stop()*** functions are used by the transition later to enter and exit a state when a transition occurs.

The next building block is an Event, it only has a name, so it is quite simple.

Event Class (Code by Author)

The last building block in an FSM is a Transition. Transition is depicted by an arrow in the picture below.



Transition (Image by Author)

It has four components:

1. Event
2. Condition
3. Action
4. State(s)

When an Event is raised, a Transition will occur if all of the following are satisfied:

- The current state matches “state 1”
- There is no condition set or the condition holds (returns True)

If the action exists, it will be executed.

There are three types of transitions:

- *Normal Transition*: a transition from a state to another state, the one that we use most
- *Self Transition*: a transition from and to itself
- *Null Transition*: a transition that only executes the action, no entry or exit. This type of transition is useful for ***do*** action in a state


Transition Base Class (Code by Author)

We have everything we need for a Transition here, but the ***__call__()*** function is not implemented because it will be different for the three types.

Normal Transition will look like this, where a source state is exited and the destination state is entered.

Normal Transition Class (Code by Author)

Self Transition is as follows. There is no change in state, but all callbacks are called if any.

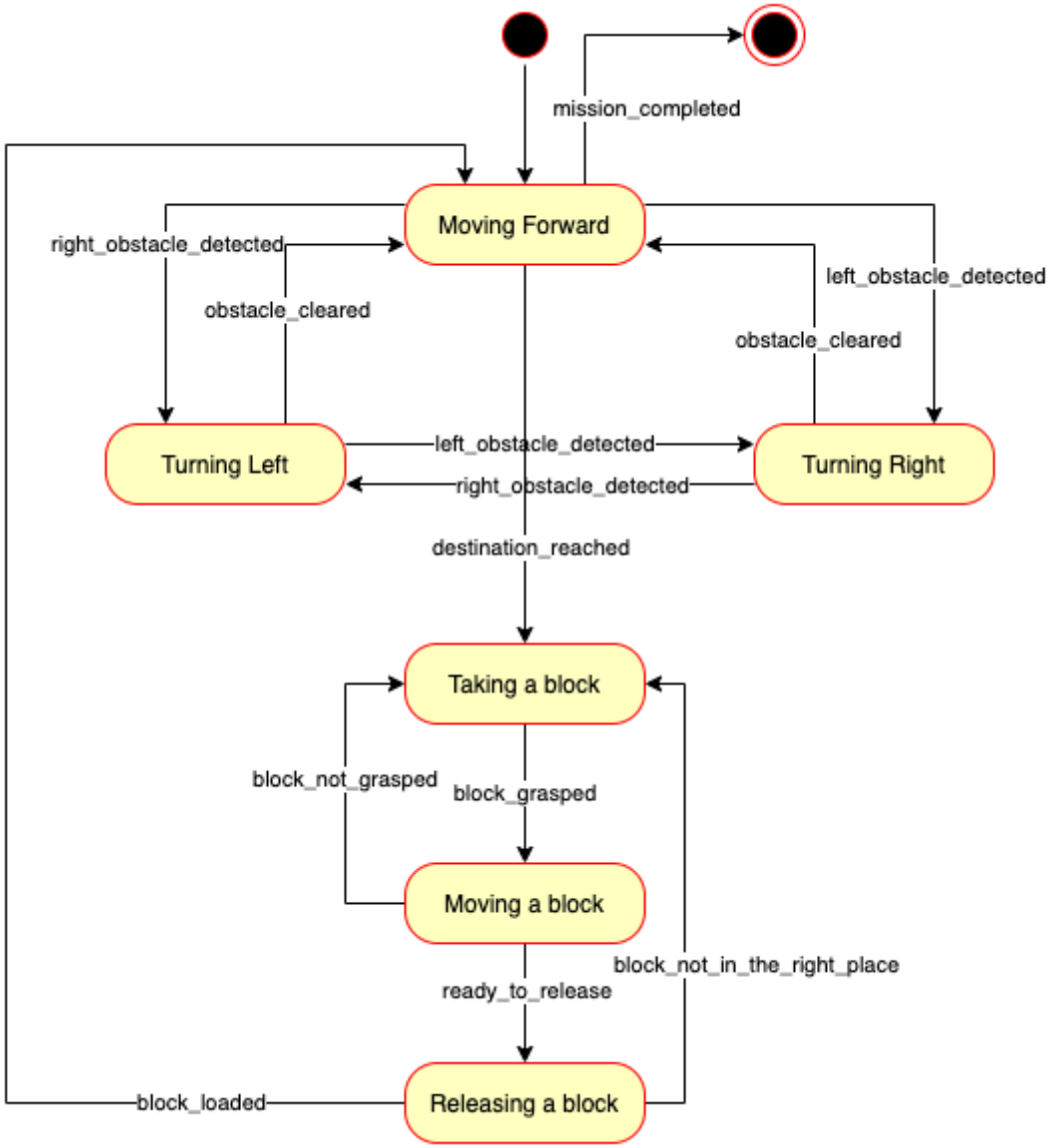
Self Transition (Code by Author)

And a Null Transition is as follows:

Null Transition (Code by Author)

We can see that a Null Transition is not really a transition because there are no exiting and entering states occurring. It only executes an action. If the action doesn't exist, it basically does nothing.

Now that we have all the building blocks for an FSM, we are ready to implement the main class the StateMachine class.



An FSM Example (Image by Author)

An FSM (or, we simply call the class StateMachine) has a collection of:

- and, Transitions

It also has a name, because we may have multiple FSMs in HFSM and also has to track its current state.

The other things that we want to add are:

- Exit State
- Initial State

They are the two black circles in the picture. And finally, we also want to add an exit callback, for us to check the result of an FSM when it has finished.

StateMachine class (Code by Author)

That is the full implementation of an FSM. We have a couple of functions to add states, events, and transitions into the FSM.

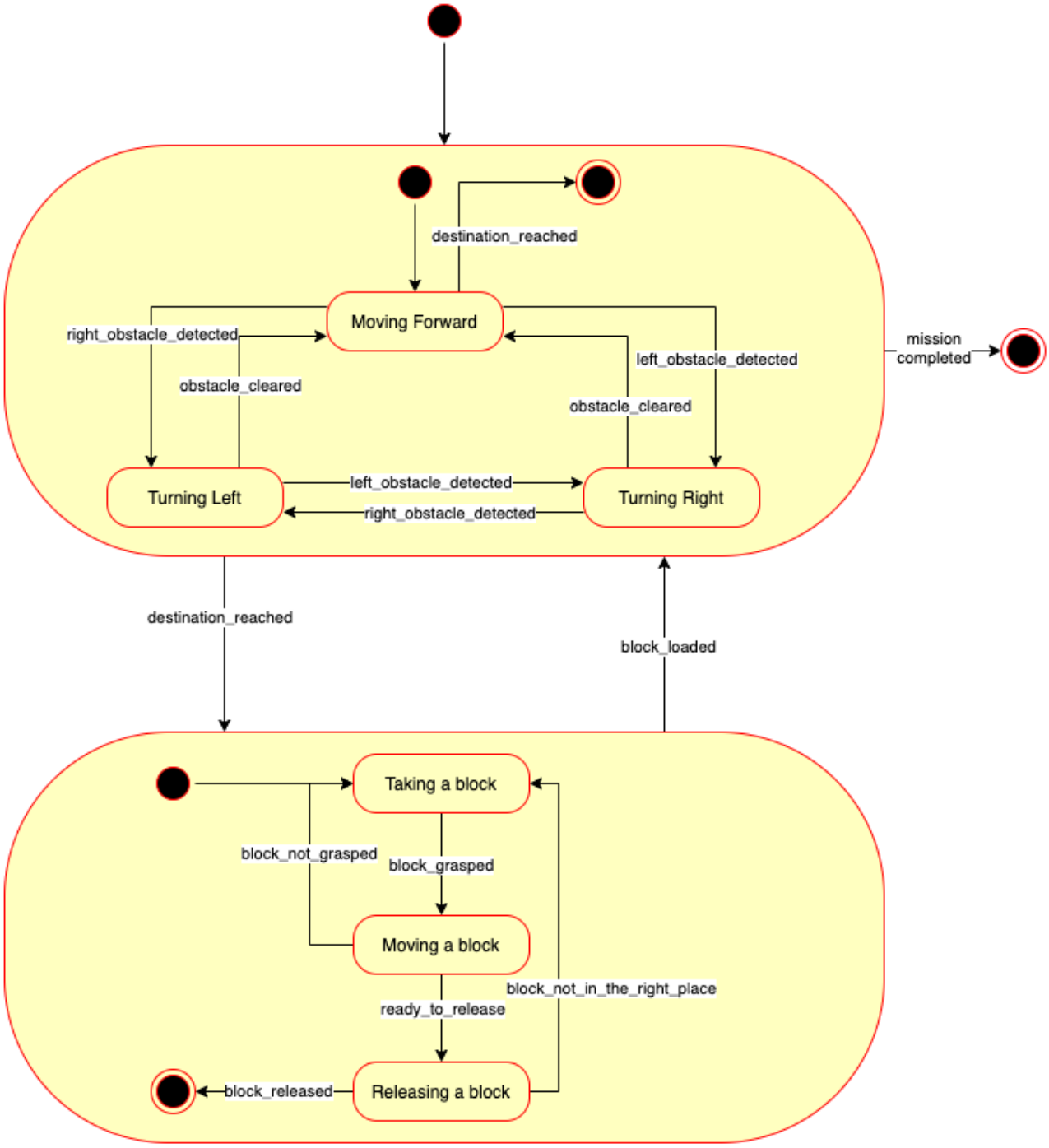
In the ***add_xxx_transition()*** functions we return the transition object so that the caller can simply add condition and action like in the example below:

Add Condition and Action to a Transition (Code by Author)


That's all we need for an FSM Implementation. Next, we turn our focus on the implementation of HFSM.

HFSM Implementation

An example of HFSM is shown in the picture below, it is an HFSM version of the FSM we saw earlier.



1. be able to add a child state machine to a state
2. start a child state machine when parent state is entered
3. stop a child state machine when parent state is exited
4. propagate an event to the lower-level state machine(s)

First, we need to modify our State Class to support (1), (2), and (3):

Final State Class (Code by Author)

The updated trigger_event function (Code by Author)

Wrap Up

Hopefully, this walkthrough is useful for you to see how I implemented HFSM from concepts to code and you can learn something from it. Just like I learned when I implemented it.

The full implementation can be seen in the repository hosted in Github below:

Or, if you want to use it, you can simply install it with

```
pip install hfsm
```

Thank you for reading!