Homework 3 - Decision Making

COS498/598 - Video Game AI - Spring 2025

by Zachary Hutchinson

Due Date: Mar 14, 2025, Midnight, Brightspace

Goal

Get experience engineering a decision making system.

Introduction

The existing code base for this assignment is a bit complex. The game consists of a simulator for a single individual. Basically, it is a tester for an unimplemented behavior system.

The simulator allows the user to advance time at different speeds and alter different stats of the agent. Your decision making system should react to these changes and alter the action of the agent. For example, if the agent has a *bored* stat and the user increases the stat, the agent might go to the movies or play a game.

I originally intended to have you create a behavior tree for this assignment, but I've decided to let you implement whichever system you like: finite state machine, decision tree, behavior tree or something homegrown. You can base your system on most anything you like, but you must have a verbalizable approach (see below the section on Decision System). It is worthwhile to attempt to engineer something complex on your own for which there exist well-trod solutions. It makes you appreciate (and understand) well-trod solutions after hacking it yourself.

The Scenario

Once you have a system through which to implement the agent's behavior, you need to come up with a idea. Where is the agent? What is the world it lives in? What is its job? Friends, Family, Likes, Dislikes, Dreams, etc. Use this idea to create a series of behaviors for the agent. What do the agent do on Sunday afternoon? Is it always the same?

But we're not trying to simulate every aspect of real life. Don't get bogged down in too much detail. Consider the above as a character in a video game that might be observed by a player going about their business. Think of the Action messages in this way: what does the player see.

Files

What follows is a description of the files associated with this project:

hw3.py

This is the main file containing the simulation logic, game loop, initialization and pygame stuff. There really isn't anything in this file you need to change. You are welcome to change whatever you like if it helps your code.

defs.py

This file holds constant information used by the simulator. You will definitely want to change the STATS dictionary to suit your scenario, theme, etc. You can also change the Agent's name and time/day the sim starts.

You are welcome to add more game speeds if it helps.

Please change the agent's name.

ztime.py

This is a small day/time class. It is used by the system to keep track of time (to the minute). You don't need to modify this file, but you will need to understand it. You need to create Time objects for new Actions.

agent.py

Contains the Agent class. An agent is a name, a stats object, an idle message and an action. Basically, it is a container for several objects that make up the functionality of an Agent.

The idle message is what gets displayed when the agent's action expires and it has not gotten a new one. It's fine if your agent from time to time goes idle. But it shouldn't be there for a very long time.

See the Stat object in *agent_stat.py* for the Stat class.

NOTE: The job of your decision system is to set an agent's action data member such that as an agent's stats change, its action also changes.

action.py

An Action object abstracts what an agent is doing. You can think of this as animation for a text-based sim. Your decision system will be comprised of a set of actions that are placed in the agent if the right conditions are met.

An action consists of a *msg* which is the text printed to the screen while the action is live. A start time and end time. The start time isn't used by the action or agent but you might need it for your decision system. An action also has an *end time*. When the current time reaches the *end time*, the action is done. When an agent's current action is done, it will delete the action and display its idle_message.

NOTE: Your decision system does not need to wait for an action to expire to replace it. It can replace them at any time and start a new action. This is the fundamental difference between systems that are reactive and those that are not (the agent is on rails until it finishes what it is doing). Reactive agents are hard to get right because external events (or internal) ones can cause them to jump from one action to another inappropriately or out of context.

agent_stat.py

This file defines the Stat class. Stats are defined in the *defs.py* file. This class stores the name, an index of current value and a list with all possible values. You probably don't need to change anything here, but you should understand how to use objects of this class. Your decision system will need to use stat values to determine which action the agent should be doing.

decision.py

This is the file for all your code. It defines two functions required by the simulator: *tick_decisionsys* and *make_decisionsys*.

make_decisionsys: This is the entry point for your implementation. The sim calls this function before starting the main loop. It doesn't return anything. It takes the agent object as its only parameter. In this function, you should create the decision system for your agent. NOTE: You can store the instantiated system anywhere you like if the system itself maintains any state. You are welcome to store additional state in the agent as well by modifying that class.

tick_decisionsys: This function is called each turn by the sim to check the decision system to see if the agent should change to another action. The agent and the current time are passed into the function. It does not return anything. This function should interface with your decision system to check whether the agent should change its current action.

NOTE: If your decision system requires more of an interface than two functions, you are welcome to add more calls to hw3.py in appropriate places.

Decision System

As I stated above, you are welcome to implement any system you like (Behavior Tree, Decision Tree, FSM, H-FSM, etc.).

However, I would like your system to have some level of *modularity* and *reusability*. This means that you create a bit of software that could be used to create a decision system for another agent. Parts of it are reusable. There are no wrong answers here except a system that would be completely worthless if you tried to make a second or third agent.

And then you should use this modular system to create a decision-making system for your agent.

But don't let the above stop you from jumping into the code. Initially, you will implement a very narrow system but if you keep working with it, you'll see how to generalize parts. Iterate, iterate, iterate.

Using the Simulator

Here are a list of the hot keys for the simulator:

- UP/DOWN ARROW: cycles through the agent stats.
- LEFT/RIGHT ARROW: cycles through values of a stat.
- LEFT/RIGHT BRACKET: change the sim speed.
- P: pauses/unpauses the sim. Unpausing will restore the same speed before the sim was paused.
- Q: quits the sim

Submission

Submit **ALL** code and resource files in a zip file to Brightspace. In your zip you need to include a write-up describing your system, the scenario/theme/setting, the agent and what you tried to create using the agent's behavior.

Be detailed in your write-up. With enough stat variables across a 24 hour clock and 7 days of the week, I might not see all the behaviors you've included. I want to. So tell me about them.

Tell me about your decision system. Did you base it on one of the approaches mentioned in class or the book? Or did you hack something out yourself. Describe how it works. Is it reusable? Is it modular? How could you improve it?

Grading

- 30% Write-up
- 30% Decision System
- 40% Agent scenario