Spike: Task 4

Title: Goal-Oriented Behaviour

Author: Sam Huffer, 101633177

# Goals / Deliverables

* Working code to demonstrate goal-oriented behaviour (GOB) using simple goal insistence (SGI). See “\04 – Spike – Goal Oriented Behaviour\SimpleGOB\SimpleGOB\SimpleGOB.py”.
* Demonstrations of where SGI does and does not work well.

# Technologies, Tools, and Resources Used

* Visual Studio (VS) 2017 (for editing code)
* Learning materials on Canvas (for instructions and sample code)
* Command prompt (for executing and testing the code)

# Tasks Undertaken

* Created a python3 project in VS and copied the sample code into it.
* Read the sample code to get a gist of what was going on, paying attention to comments denoting missing functionality.
* Compiled and ran the project to see what happened when the unaltered sample code ran.
* Went back to the comments about missing functionality, picked a feature, and implemented it (see below).
* Re-compiled the project and ran it again. If it broke or did not work as intended, I edited and re-ran the code until it worked as intended.
* Once one feature was done, I picked another piece of missing functionality and repeated.
* Once all missing features commented on in the code were implemented, I split the code into a Game class and an AI class according to object-oriented principles. I then ran the code and fixed any bugs that appeared.

# Procedural Code Added / Edited

Figure 1: Added side effects to each action listed.

Figure 2: Swapped the “Pythonic” code for the “non-Pythonic” code for ease of reading.

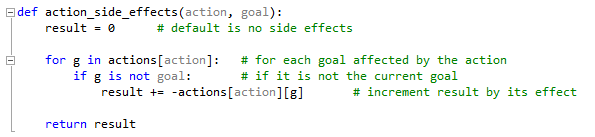


Figure 3: Added function for returning the total side-effects of an action.

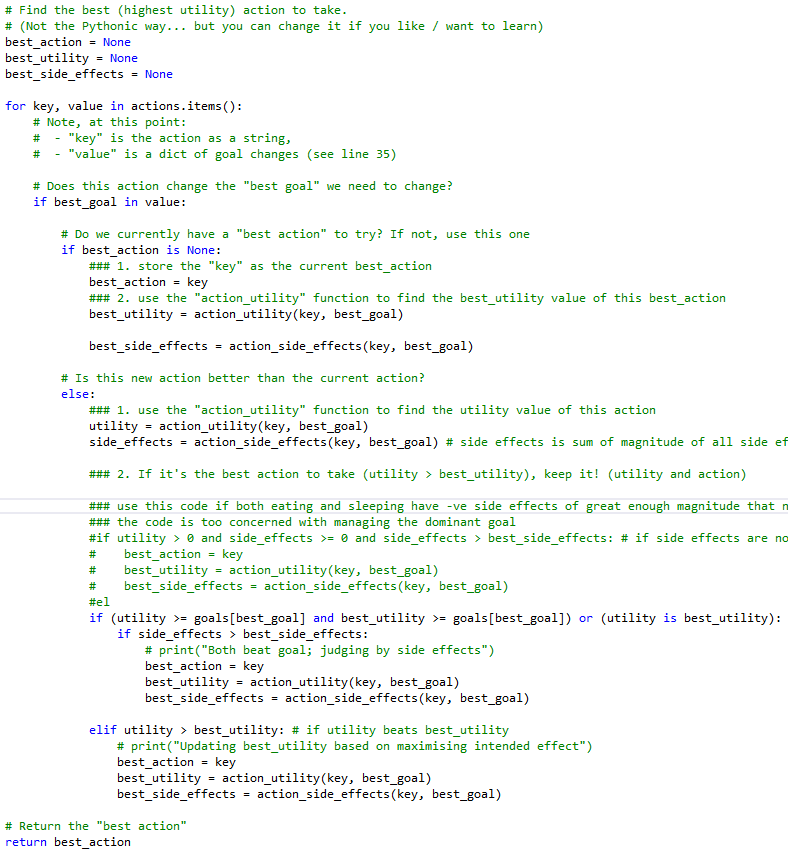


Figure 4: Implemented code for selecting the best action to take.

# OOP Classes

AI

bool self.VERBOSE

dict{str, int} self.goals

dict{str, dict{str, int }} self.actions

str self.name

void \_\_init\_\_(self, verbose, actions, goals, name)

void apply\_action(self, action)

int action\_utility(self, action, goal)

int action\_side\_effects(self, action, goal)

str choose\_action(self)

void play\_until\_all\_goals\_zero(self)

Game

bool self.VERBOSE

dict{str, int} self.goals

dict{str, dict{str, int }} self.actions

void \_\_init\_\_(self)

void print\_actions(self, actions)

void main(self)

# What I Found Out

If the intention of the code provided to students was that we should modify it to address all the addi­tional functionality listed in the comments and have the program reach a “Done” state:

* When only one value needed to be evaluated (the effect on the goal being worked towards), the program worked perfectly fine and reached a “Done” state within a few loops.
* When evaluating the intended effect *and* side effects of an action, it was dependent on the magnitude of all the effects of the actions as to whether the program got stuck in a loop where reducing to one goal to 0 required the other goal to be addressed, and so on.
* If multiple goals are to be completed through GOB and SGI in a manner based upon that used in this spike, there need to be actions that will satisfy one goal and not negatively impact others; this is not a concern if the AI is not expected to reach a “Done” state where all goals are completed and the program ends.

With regard to the creation of the object-oriented version of the code:

* Object-oriented programming encourages the division of fields and methods into more man­ageable segments that themselves can be passed around the code to access their encapsu­lated data or asked to perform tasks expected of them.
* Object-oriented programs can take advantage of inheritance and polymorphism such that dif­ferent AIs have the same methods and fields but they’re implemented / populated differently, allowing for a variety of behaviours when the same actions of each sibling class are called by others.
* One downside of object-oriented python code specifically is the requirement of specifying self.[FIELD NAME] or self.[METHOD NAME] when a class accesses its own fields or methods, something that feels tiresome when other languages, such as C#, don’t require it as it is assumed that the field / method belongs to the class calling it unless otherwise specified.