Spike Outcome Report

Number: 13

Spike Title: The "Planning" in GOAP

Personal: Michael Williams (7668481)

**Goals:**

* Create a GOAP simulation that demonstrates the effectiveness of the technique in considering long-term outcomes of actions (related to side-effects and/or time delays) and can plan and act intelligently.

**Technologies, Tools, and Resources used:**

* Python IDE
* Sample Lab Code
* Previous Spike Code
* Lecture Material

**Tasks undertaken:**

1. The first step of this spike was to look over the lecture material. The primary objective here was to understand how the modified requirements would change the structure of the code from the original lab 3 code.
2. Once this was understood, we modified the code as simply as possible to add in the GameState object, which would hold the current goal insistence as well as the “currency” to spend on items to reduce goal insistence.
3. At this point, we tested the outcome to make sure it was the same as the original code, it was.
4. Now that we had the baseline code, the next step was to implement the code similar to the lecture material to allow for planning. As we are reasonably unfamiliar with python, this code took us quite some time to understand and work out so that we could implement it using our objects. The final code for this looks like:

**while** states:  
 current\_value = states[-1][0].GetCurrentStateDiscontent()  
  
 **if** len(states) >= 3:  
 **if** current\_value < best\_value:  
 best\_value = current\_value  
 *#get the best plan* best\_plan = [state[1] **for** state **in** states **if** state[1]] + [best\_value]  
 states.pop()  
 **continue** next\_action = states[-1][0].next\_action()  
 **if** next\_action:  
 next\_state = deepcopy(states[-1][0])  
 states.append([next\_state,**None**])  
 states[-1][1] = next\_action  
 next\_state.reset()  
 *##############* states[-1][0].apply\_action(next\_action)  
 changed = **True  
 else**:  
 states.pop()

1. At this point we tested our results, knowing what the expected outcome should be. But it didn’t work. The way in which we had refactored out code to add in the game state object did not work correctly when we called the deepcopy, which we suspected may have been the case. This error was caused as the deepcopy does not understand the context of the references, only that they exist. We then simplified our Action and Goal classes to give all the responsibility to the GameState class. This worked much better.

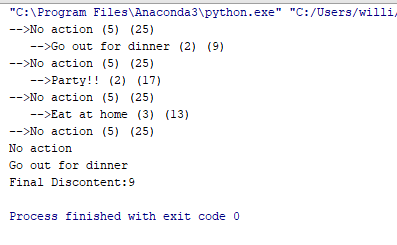
**What we found out:**

We found out about the importance and effect of implementing some sort of future planning into the goal-oriented behaviour architecture. It was quite interesting to see the effects that could be had when changing the depth to which the planning would occur. When it was set to 1 level, just the next action with no planning, the model would react the same as in Spike 3. But once we added the ability for the model to plan its next action and take into account the total discontent after multiple iterations, you end up with a much better, strategic decision.

Further activities that could be undertaken would be to work out what happens when the environment changes and suddenly the plan is out of date and needs to be recalculated. How this should be signalled and what performance effects there are for this. Additionally, how many iterations should be planned for optimal behaviour and how does this change for different environments?

**Screenshots**

No Planning



With Planning

