Spike: Spike_1
Title: Spike_GOB & SGI

Author: LE BAO DUY NGUYEN - 102449993

Goals / deliverables:

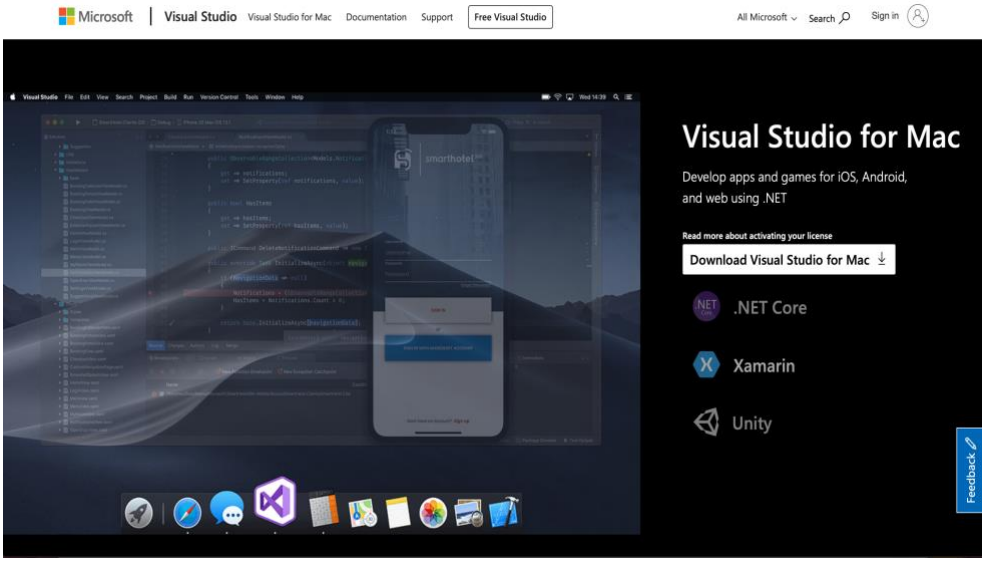
- Code see /spikes/spike04/
- Short report titled “Spike_GOB & SGI”
- Completed gob_simple.py

Technologies, Tools, and Resources used:

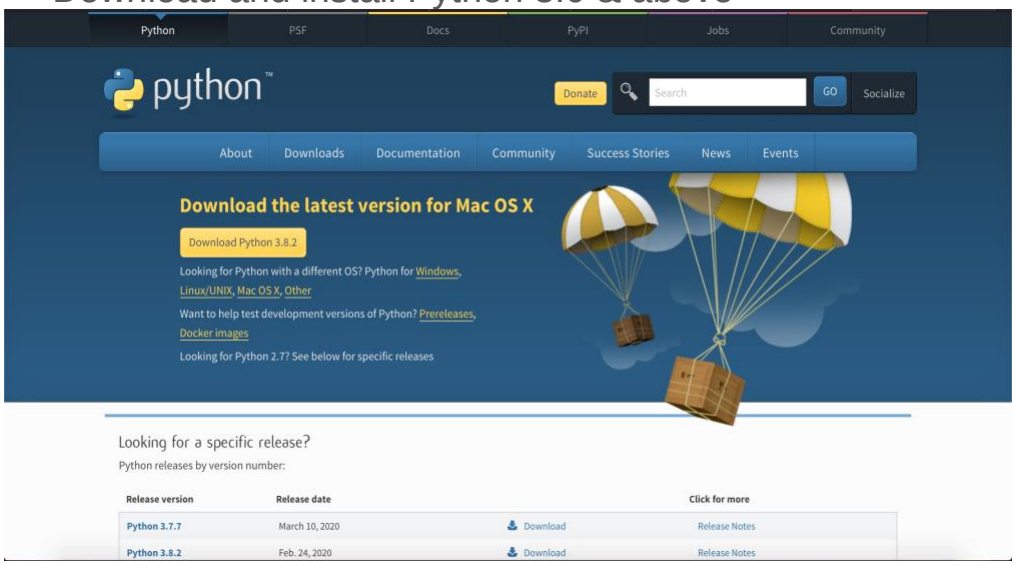
- Visual Studio Code
- Python 3.0+

Tasks undertaken:

- Download and install Visual Studio Code

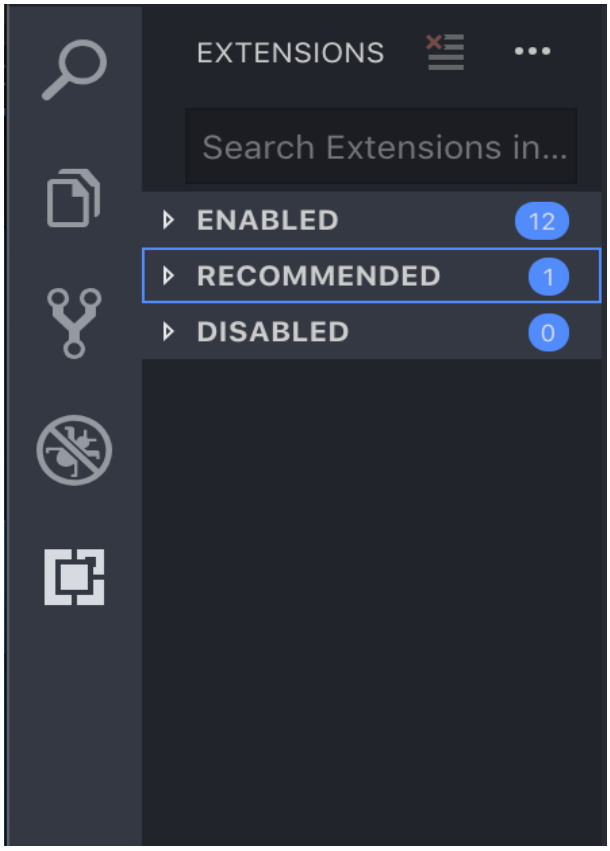


- Download and install Python 3.0 & above



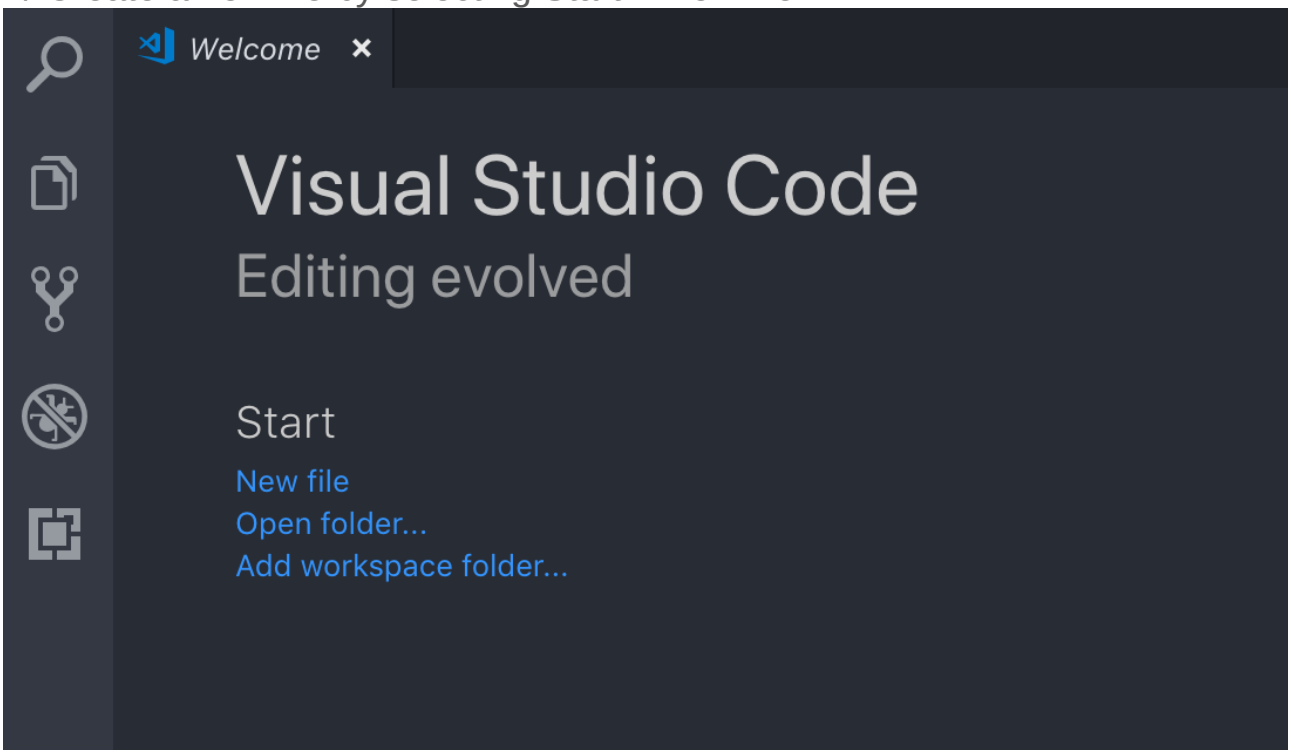
- Download and install Python extension within Visual Studio Code

1/ Go to Extensions

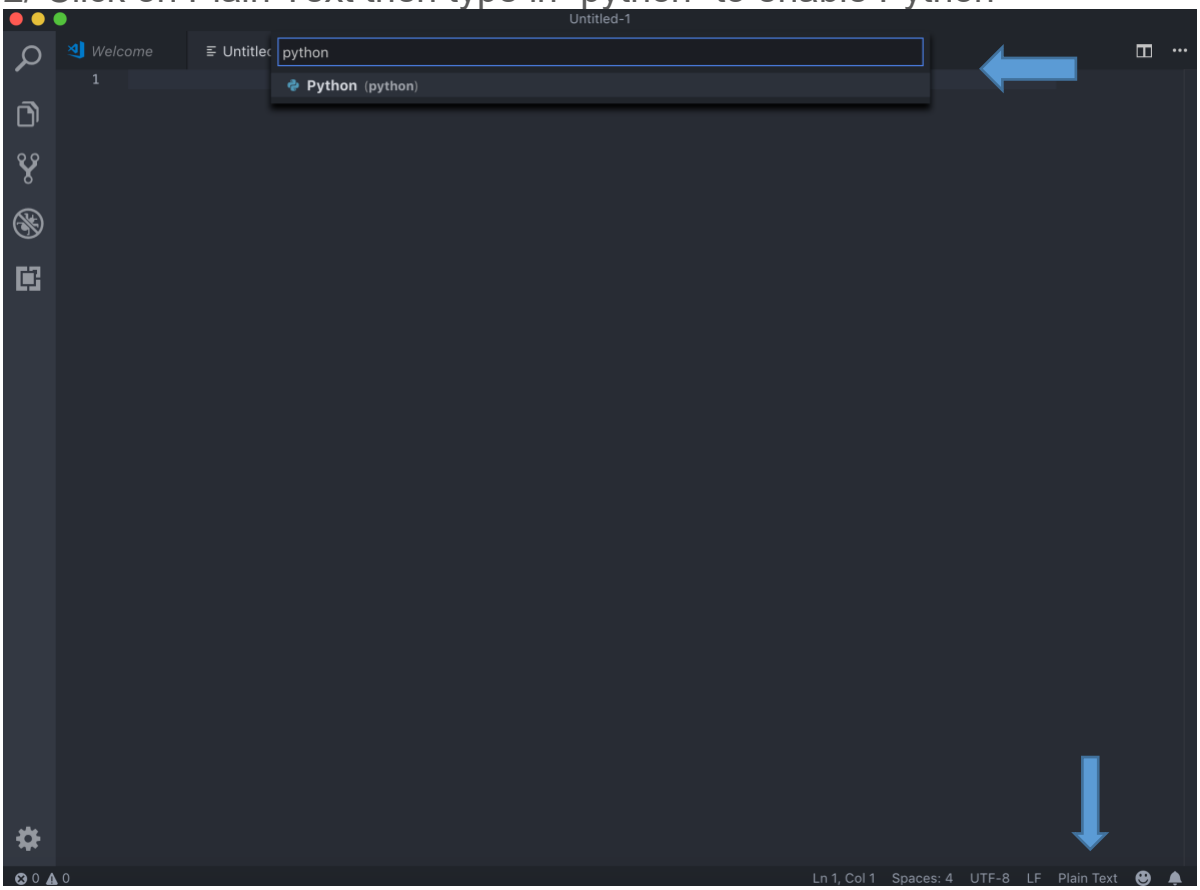


2/ Search “Python” then Install

- Add missing code into sample code
  1/ Create a new file by selecting Start > New file



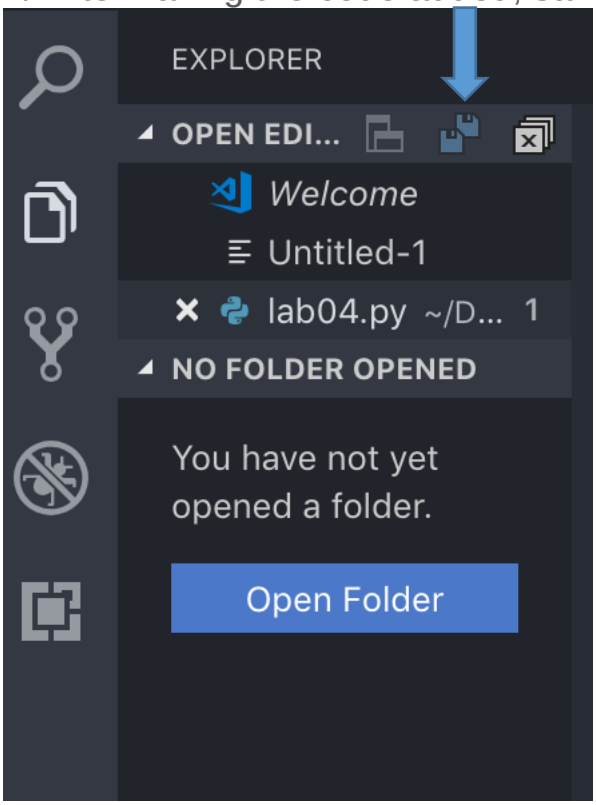
2/ Click on Plain Text then type in “python” to enable Python



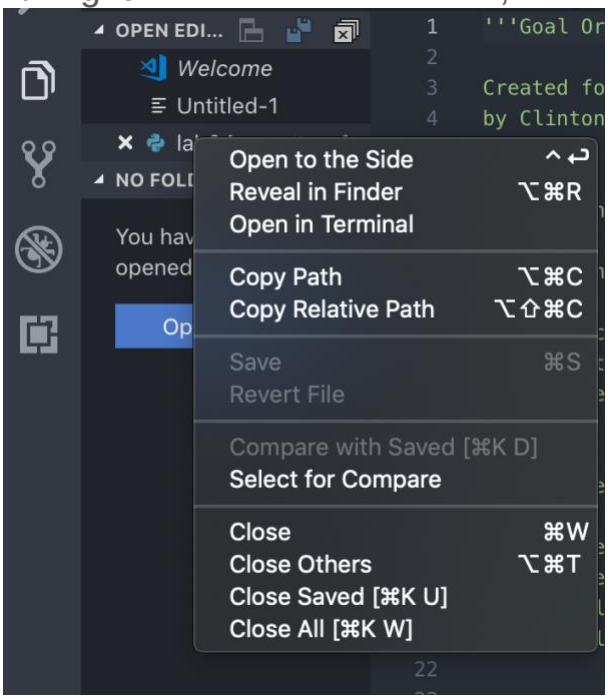
3/ Add missing code

```
        if best_goal in value:

            # Do we currently have a "best action" to try? If not, use this one
            if best_action is None:
                ### 1. store the "key" as the current best_action
                best_action = key
                ### 2. use the "action_utility" function to find the best_utility value
of this best_action
                best_utility = action_utility(best_action, best_goal)

            # Is this new action better than the current action?
            else:
                ### 1. use the "action_utility" function to find the utility value of
this action
                utility = action_utility(key, best_goal)
                ### 2. If it's the best action to take (utility > best_utility), keep
it! (utility and action)
                if utility > best_utility:
                    best_action = key
                    best_utility = utility

    # Return the "best action"
    return best_action
```

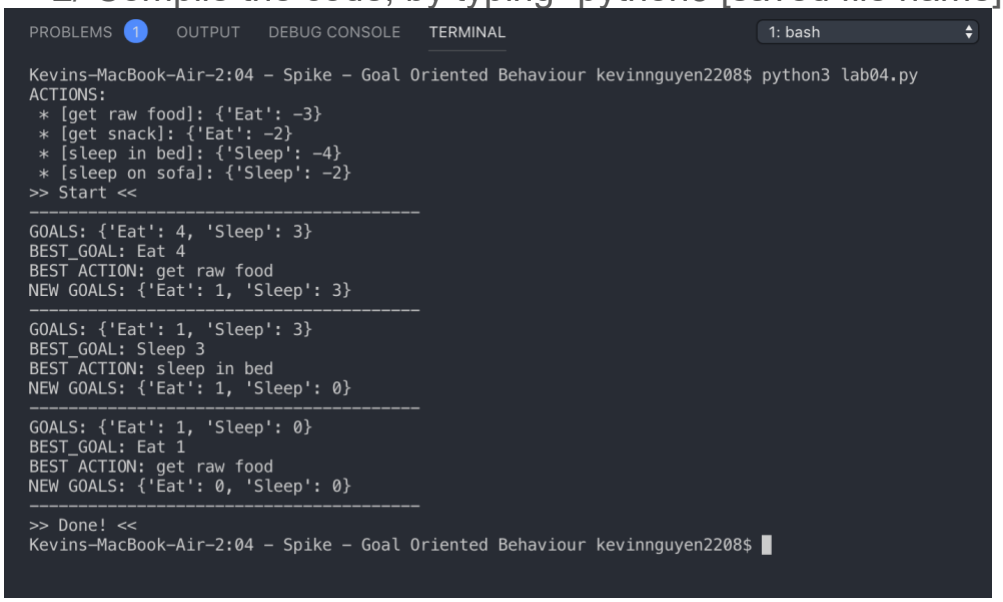
4/ After having the code added, save the file by clicking this icon



- Compile sample code
  1/ Right-click on the saved file, select “Open in Terminal”



  2/ Compile the code, by typing “python3 [saved file name].py”



What we found out:

```
def choose_action():
    '''Return the best action to respond to the current most insistent goal.
    '''
    assert len(goals) > 0, 'Need at least one goal'
    assert len(actions) > 0, 'Need at least one action'

    # Find the most insistent goal - the 'Pythonic' way...
    best_goal, best_goal_value = max(list(goals.items()), key=lambda item: item[1])

    # ...or the non-Pythonic way. (This code is identical to the line above.)
    #best_goal = None
    #for key, value in goals.items():
    #    if best_goal is None or value > goals[best_goal]:
    #        best_goal = key

    if VERBOSE: print('BEST_GOAL:', best_goal, goals[best_goal])

    # Find the best (highest utility) action to take.
    # (Not the Pythonic way... but you can change it if you like / want to learn)
    best_action = None
    best_utility = None
    for key, value in actions.items():
        # Note, at this point:
        #  - "key" is the action as a string,
        #  - "value" is a dict of goal changes (see line 35)

        # Does this action change the "best goal" we need to change?
        if best_goal in value:

            # Do we currently have a "best action" to try? If not, use this one
            if best_action is None:
                ### 1. store the "key" as the current best_action
                best_action = key
                ### 2. use the "action_utility" function to find the best_utility value of this best_action
                best_utility = action_utility(best_action, best_goal)

            # Is this new action better than the current action?
            else:
                ### 1. use the "action_utility" function to find the utility value of this action
```

```
        utility = action_utility(key, best_goal)
        ### 2. If it's the best action to take (utility > best_utility), keep it! (utility and action)
        if utility > best_utility:
            best_action = key
            best_utility = utility


    # Return the "best action"
    return best_action
```

```
ACTIONS:
 * [get snack]: {'Eat': -2}
 * [sleep on sofa]: {'Sleep': -2}
 * [get raw food]: {'Eat': -3}
 * [sleep in bed]: {'Sleep': -4}
>> Start <<
----------------------------------------
('GOALS:', {'Sleep': 3, 'Eat': 4})
('BEST_GOAL:', 'Eat', 4)
('BEST ACTION:', 'get raw food')
('NEW GOALS:', {'Sleep': 3, 'Eat': 1})
----------------------------------------
('GOALS:', {'Sleep': 3, 'Eat': 1})
('BEST_GOAL:', 'Sleep', 3)
('BEST ACTION:', 'sleep in bed')
('NEW GOALS:', {'Sleep': 0, 'Eat': 1})
----------------------------------------
('GOALS:', {'Sleep': 0, 'Eat': 1})
('BEST_GOAL:', 'Eat', 1)
('BEST ACTION:', 'get raw food')
('NEW GOALS:', {'Sleep': 0, 'Eat': 0})
----------------------------------------
>> Done! <<
Kevins-MacBook-Air-2:04 - Spike - Goal Oriented Behaviour kevinnguyen2208$  $
```

The effectiveness of SGI is that it allows the game bots to perform actions based on the insistence or goal varying on what it is. Referring to the output below, the best goal changes after an action, the higher the goal, the more it will influence the agent into making a certain action.

The disadvantage of using Simple Goal Instance is that it leaves little options for the bots to do anything afterwards (unless the goal was too simple). It doesn't take into consideration the side effects because so far, there were only positive outcomes, to make it more convincing there needs to be negative effects as well, such as certain values to balance things out. Add some additional functions or values to make it more realistic.

Risks/ Issues

_ Visual Studio Code issue when installing and running Python + its in-app extension.
_ Python version needs to be updated to the latest and run using "python3 …" command line.