Spike: Spike_6
Title: **Goal Oriented Action Planning**

Author: Le Bao Duy Nguyen 102449993

Goals / deliverables:
Utilising search algorithms, we can create an appropriate plan for executing independent actions to achieve a goal in a flexible manner.

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
- Read the codes and guidelines. Researching Canvas materials as well as Google, YouTube, etc.

This GOAP is inspired by the Avengers: Infinity War movies. The Goal is to defeat Thanos through different steps. The possibility of the outcome is calculated according to the actual outcome mentioned in the movie.

First GOAP state is defined, along with the preconditions to change states.. It was also responsible for containing all the actions that were performed on the state.

```
class GOAP_State:
    def __init__(self, states):
        self.states ={}
        for state in states:
            self.add_state(state)

    def add_state(self, name, value = False):
            self.states[name] = value

    def preconditions_met(self, action):
        for precondition in action.preconditions:
            if self.states[precondition['State']] != precondition['Needed']:
                return False
        return True
```

```
def perform_action(self, action):
    for effect in action.effects:
        self.states[effect['State']] += effect['Result']
```

An action class was used to monitor the behaviour the agent would do in order to trigger off the state change event. The precondition & effects are required to combine together into a cohesive plan between each independent state. These were important to ensure the GOAP to determine a suitable path of actions to reach the goal.
The agent planned a series of tasks to complete the goal. DFS algorithm was used for memory utilisation.

```
class GOAP_Agent:
    def __init__(self):
        self.state = []
        self.actions = []
        self.running_cost = 0
        self.paths_evaluated = 0

 def perform_action(self, action):
        self.state.perform_action(action)
        self.running_cost += action.cost

    def plan(self, goal, state = None, path = None, start_action = None):
        if state is None:
            state = copy.deepcopy(self.state)

        if path is None:
            path = {'Actions': [], 'Ratio': 0}
            self.paths_evaluated = 0

        if start_action:
            path['Actions'].append(start_action)
            path['Ratio'] += start_action.cost
            state.perform_action(start_action)

        if state.states[goal_state]:
            self.paths_evaluated += 1
            return path

        available_actions = set()
        for action in self.actions:
            if state.preconditions_met(action):
                available_actions.add(action)
```

```
        easiest_path = None
        for action in available_actions:
            potential_path = self.plan(goal_state, copy.deepcopy(state), copy.deepcopy(path), action)
        if not easiest_path or potential_path['Ratio'] < easiest_path['Ratio']:
            easiest_path = potential_path

        return easiest_path
```

Different states and actions were declared. Each state can be chosen individually to be a goal state.

```
get_Gauntlet = GOAP_Action('get Gauntlet',5412989)
get_Gem = GOAP_Action('get Gem', 600860)
fight_Thanos = GOAP_Action('fight Thanos', 7986756)

agent = GOAP_Agent()
agent.state = GOAP_State(
    [
        'Has Gem',
        'Has Gauntlet',
        'Defeat Thanos'
    ])
agent.actions = [
    get_Gauntlet,
    get_Gem,
    fight_Thanos
]

get_Gem.add_precondition('Has Gem', False)
get_Gem.add_effect('Has Gem')
get_Gauntlet.add_precondition('Has Gem')
get_Gauntlet.add_precondition('Has Gauntlet', False)
get_Gauntlet.add_effect('Has Gauntlet')
fight_Thanos.add_precondition('Has Gauntlet')
fight_Thanos.add_precondition('Defeat Thanos', False)
fight_Thanos.add_effect('Defeat Thanos')




goal_state = 'Defeat Thanos'
path = agent.plan(goal_state)
```

```
print ('Goal: ' + goal_state + "\n")
for i in range(len((path['Actions']))):
    print(str(i+1) + ') ' + path['Actions'][i].name + ' (' + str(path['Actions'][i].cost) + ')')

print ('Win ratio: 1/' + str(path['Ratio']))
input ("Press any key to continue")
```

What we found out:

```
Kevins-MacBook-Air-2:16 - Spike - GOAP kevinnguyen2208$ python3 goap_Thanos.py
Goal: Defeat Thanos

1) get Gem (600860)
2) get Gauntlet (5412989)
3) fight Thanos (7986756)
Win ratio: 1/14000605
```

The implementation creates simple instantiations of actions and states, allowing a flexible set actions that could be combined with preconditions and effects. This would then allow the planner system to then calculate the best approach to take and execute it. The use of GOAP creates an extensible approach to designing AI for NPC's and other enemies that can create more dynamic behaviour easily.

While this was a simple plan, it outlined how GOAP is used to create a decoupled FSM that can easily plug in new actions and states which the planning algorithm utilised to create the best path based on the search method.