CSCI3180 Assignment 2 Report

Wong Pui Yee 1155127529

Dynamic Typing

Dynamic typing provides the variables with reusability and flexibility, which allows programmers to make full use of the variables in the codes and save the lines of codes written.

In the section of the goblin initialization in Engine.py, the variable gno is of type integer originally when using with range() in for loop in the first line as shown in Fig.1, assigning with the numbers from 0 to num_of_goblins – 1. In the seventh line, gno has become an object of the class Goblin within the for loop.

```
for gno in range(num_of_goblins):
    # TODO: initilize each Goblin on the map
    #       using the set_occupant and occupying setter;
    #       add each Goblin to _actors array
    line = fp.readline()
    items = line.split()
    gno = Goblin(int(items[0]), int(items[1]), items[2:])
    gno.occupying = self._map.get_cell(int(items[0]), int(items[1])).set_occupant(gno)
    self._actors.append(gno)
    # END TODO
```

Fig. 1

Duck Typing

Duck Typing provides the codes with compatibility and maintainability. When there are new features (i.e. classes) that are going to be implemented in the function built before, only a few more lines of codes (i.e. same attributes as the old classes) are needed for the new feature to be used in the function and no modification is needed for the existing function.

In the extension, volcano, as the new feature, is implemented into the game. The volcano is a class that inherited from the class Mountain and is appended into the list named self._actors as shown in Fig.2 in NewEngine.py, with the objects of class Player and Goblin, inherited from the class GameCharacter. The attributes of the classes GameCharacter and Volcano are as shown in Fig.3 and Fig.4 respectively. After each round in the game, clean_up(), in NewEngine.py, removes the objects that are inactive in the list self._actors as shown in Fig.5. Since the volcanoes are appended to the list, the attribute active and its getter are needed to be implemented in the Class Volcano for clean_up() as shown in Fig.4 and Fig.6. Although volcanoes are not objects of the class GameCharacter, it shares the same attribute with GameCharacter to work well in the function clean_up().

```
for vno in range(num_of_volcanoes):
    # TODO: initilize each Volcano of the map object
    #       using the build_cell method
    #       add each volcano to _actors array
    line = fp.readline()
    items = line.split()
    vno = Volcano(int(items[0]), int(items[1]), int(items[2]))
    self._map.build_cell(int(items[0]), int(items[1]), vno)
    self._actors.append(vno)
    # END TODO
```

Fig. 2

```
class GameCharacter(metaclass=ABCMeta):
    def __init__(self, row, col):
        self._row = row
        self._col = col
        self._occupying = None
        self._name = None
        self._active = True
        self._character = None
        self._color = '\033[1;31m'
```

Fig. 3

```
class Volcano(Mountain):
    def __init__(self, row, col, freq):
        Mountain.__init__(self, row, col)
        self._countdown = freq
        self._frequency = freq
        self._color = '\u001b[41m'
        self._active = True
```

Fig. 4

```
def clean_up(self):
    # TODO: remove all objects in _actors which is not active
    i = 0
    for obj in self._actors:
        if obj.active == False:
            self._actors.pop(i)
        i += 1
    # END TODO
```

Fig. 5

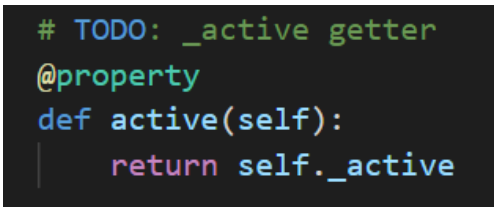
TODO: _active getter
@property
def active(self):
    return self._active

Fig. 6