Task 2: Demonstrating Advantages of Dynamic Typing

Dynamic typing not requires to input data type of the variable very strictly. Here are some examples.

```
19  if __name__ == '__main__':
20      if len(sys.argv) > 1:
21          if sys.argv[1] == 'basic':
22              from Engine import Engine
23              eng = Engine('map-basic.txt')
24              if eng != None:
25                  eng.run()
26          elif sys.argv[1] == 'extension':
27              from NewEngine import NewEngine
28              eng = NewEngine('map-extension.txt')
29              if eng != None:
30                  eng.run()
31      else:
32          print('usage: python3 StrangePlaent.py [basic/extension]')
```

Line 24, 29, in StrangePlanet.py

```
21  class Engine:
22      def __init__(self, data_file):
23          self._actors = []
24          self._map = None
25          self._player = None
26          with open(data_file, 'r') as fp:
27              line = fp.readline()
28              if not line:
29                  return None
30              else:
31                  items = line.split()
32                  if len(items) != 5:
33                      print('INVALID DATA FILE.')
34                      return None
```

Line 29, in Engine.py

In line 24 and 29 of *StrangePlanet.py*, it allows `eng` can be a `None` type variable, or a real `Engine` object. However, in Java or C/C++, we can only set the object's variable to some other thing which obey the nature of the kind of variable declared. Even in the printing function, Python only requires `print(myVariable)`, but C requires to decide the output type: `printf("%d", myVariable)`.

```
20  from Trap import Trap
21  from Volcano import Volcano
22  class NewEngine:
23      def __init__(self, data_file):
24          self._actors = []
25          self._map = None
26          self._player = None
27          with open(data_file, 'r') as fp:
```

Line 24, from NewEngine.py

```
51
52      def display(self):
53          # TODO: print a string to display the cell
54          #       and the occupant in the cell (done)
55          if (self._occupant != None):
56              #print("%shaha %s%s \033[0m   " % (self._color, (self._occu
57              print(self._color, end="")
58              print(" ", end="")
59              print(self._occupant.display(), end="")
60              print(self._color, end="")
61              print(" \033[0m   ", end="")
62          else:
63              print(self._color, end="")
64              print("   \033[0m   ", end="")
65          # END TO
66
67  class Plain(Cell):
68      def __init__(self, row, col):
```

Line 57-64, Cell.py

Also, in line 24 of *NewEngine.py*, `self._actors` are defined as a list of items. It is a big bucket that can append ANYTHING into it. For example, Player, Goblin, Volcano.
However, in a typical array in C/C++, array should be implemented with a specified data type. For example, if you want to declare an array which mainly for storing `Player`, we can declare it like `Player myArray[25]`, but it can only store `Player`, cannot store `Goblin`.

There are the advantages of dynamic typing:

1. Error handling, in Java, if we want to raise an error, for example, in File I/O, we have to use try and catch to do it, which in the try, it require a very BIG bracket to include almost ALL the lines when initializing in `Engine`. But in python, we can change the variable type to `None` by return and we can back to the function to print the error message. Which make it more convenient

```
9   public Engine(String dataFile) {
10          this.actors = new ArrayList<Object>();
11          this.map = null;
12          this.player = null;
13
14          try {
                BufferedReader reader = new BufferedReader(new FileReader(dataFile));
                String tempString = null;
17
18              tempString = reader.readLine();
19              String[] arr = tempString.split("\\s+");
20              int rowNum = Integer.parseInt(arr[0]);
21              int colNum = Integer.parseInt(arr[1]);
```

Line 14 in Engine.java

2. Convenient Implementation. As mentioned before, in python, we can declare a variable without specify it as any type of variable by explicit variable declaration. Sometimes we may encounter some complicated data type like iterator, object array, etc. we can declare the variable without specify the type. And also, if we encounter some variable name that we want to use for more than one function , like `input` that is for handling input of each part of function, we only need 1 variable to handle them all, even some are integer and some are string, but in C, we need to separate it into `inputString` and `inputInt` to handle them

A small disadvantage of Dynamic Typing

Sometimes we may not able to find the type of the variable or object easily. For example, `isinstance()` is a function that to find the type of object in Python, Java also have one called `instanceof` which can return is the input is equal to the type of some object. However, if Python without any imports, we may not be able to find it by `isinstance()` (error will be raised) , because it does not really care for the object name of it.

```
18  public boolean interactWith(Object comer) {
19          this.occupying.removeOccupant();
20          if (comer instanceof Goblin) {
21              System.out.printf("\033[1;31;43mA goblin entered a trap at (%d, %d)and died.\033[0;0m%n", this.row, this.col);
22              ((Goblin)comer).setActive(false);
23              return false;
24          } else if (comer instanceof Player) {
25              System.out.printf("\033[1;31;43mYou entered a trap at (%d, %d)! HP - 1.\033[0;0m%n", this.row, this.col);
26              ((Player)comer).setHp(((Player)comer).getHp() - 1);
27              ((Player)comer).setOxygen(((Player)comer).getOxygen() - 1);
28              return true;
29          } else {
30              return false;
31          }
32  }
33
```

Trap.java, Line 20 and 24

Task 4: Demonstrating Advantages of Duck Typing

Rather than caring about "what it is", duck typing cares more about "what can it do". For example, in line 45 and 48 in *Volcano.py*, instead of using `isinstance()`, I can directly call Object's value. In Line 45 and 48, `occ._name` is used. If that object do not have the attribute, `None` will be returned.

```
38          print("\033[1;33;41mVolcano erupts! \033[0;0m")
39          # add game logic
40          self._countdown = self._frequency
41          cells = map.get_neighbours(self._row, self._col)
42          for i in range (0, len(cells)):
43              occ = cells[i].getOccupant()
44              if (occ != None):
45                  if (occ._name == "Goblin"):
46                      occ.setActive(False)
47                      occ._occupying.remove_occupant()
48                  elif (occ._name == "Player"):
49                      occ.setHP(occ.getHP() - 1)
50      # END TODO
```

Line 45, 48 in Volcano.py

Advantages of duck typing

In the traditional implementation of object oriented programming, polymorphism can be implemented only if declaring inheritance. (Like volcano extends mountain, Goblin extends GameCharacter), etc, to use the variable or function `act()`. However, python allows ALL do do the `act()` function, we do not need to link it into a category again, which can reduce many unnecessary casting and make the code more concise. For example, we have distinguish whether it is a `GameCharacter` or a `Volcano` and further specify the Object type in the further sentence, but not required in python.

```
def run(self):
    # main rountine of the game
    self.print_info()
    while not self.state():
        for obj in self._actors:
            if obj._active:
                obj.act(self._map)
        self.print_info()
        self.clean_up()
    self.print_result()
```

.act can be used in both Player, Goblin, and Volcano!

```
96      public void run() {
97              this.printInfo();
98              while (this.state() == 0) {
99                      for (int i = 0; i < this.actors.size(); ++i) {
                                if (this.actors.get(i) instanceof GameCharacter) {
101                                     if (((GameCharacter)this.actors.get(i)).getActive()) {
102                                             ((GameCharacter)this.actors.get(i)).act(this.map);
103                                     }
                                } else if (this.actors.get(i) instanceof Volcano) {
105                                     if (((Volcano)this.actors.get(i)).getActive()) {
106                                             ((Volcano)this.actors.get(i)).act(this.map);
107                                     }
108                             }
109                     }
110                     this.printInfo();
111                     this.cleanUp();
112             }
113             this.printResult();
114     }
```

much more complicated in the non duck typing programming

A small disadvantage of duck typing

As the `act()` function can come from any part of program. If there is an error or bug founded in the program code, it will be a little bit more difficult to chase back which `act()` function goes wrong.