Wong Cho Hin
1155070355

CSCI3180 Assignment2 Report

1. For advantage 1, functions can be applied on arguments of different types.

```python
def foo(x):
    if(isinstance(x,int)):
        print "it is int"
    elif(isinstance(x,str)):
        print "it is string"
    elif(isinstance(x,float)):
        print "it is float"
    elif(isinstance(x,bool)):
        print "it is boolean"
    else:
        print "it is something else"

foo(1)    #it is int
foo(0.5)  #it is float
foo(True) #it is boolean
foo("hi") #it is string
```

We can pass a variable with different types into function foo(). The function can check the type of the variable and do something with respect to the type of the variable.

For advantage 2, it is possible for mixed type collection data structure.

```python

student=[[0 for x in range(3)]for y in range(4)]
student[0]=["1155070355","Wong Cho Hin","CSCI",3,3.08]
student[1]=["1155012345","Chan Tai Man","IBBA",1,3.42]
student[2]=["1155054321","Li Siu Ming","QFIN",4,3.64]
for i in range(3):
    print student[i]
```

With mixed type list, it is convenient for us to store data within one list. Also, we can do something meaningful with this advantage. Variables with different types can be stored into one list and print the items easily.

2. This is the first scenario that python is better than java because of the usage of duck typing.

```java
    private  void gameStart() {
        int turn = 0;
        int numOfAlivePlayers = n;
        while (numOfAlivePlayers > 1) {
            // teleport after every N turns
            if (turn == 0) {
                for (Object obj : teleportObjects) {
                    if (obj instanceof Human)
                        ((Human) obj).teleport();
                    else if (obj instanceof Chark)
                        ((Chark) obj).teleport();
                    else if (obj instanceof Obstacle)
                        ((Obstacle) obj).teleport();
                }
                System.out.println("Everything gets teleported..");
            }
```

```python
    def gameStart(self):
        turn=0
        numOfAlivePlayers = self.n
        while numOfAlivePlayers>1:
            if turn==0:
                for obj in self.teleportObjects:
                    obj.teleport()
                print "Everything gets teleported.."
```

In Java, we need to check the type of an object and cast it back to the corresponding type in order to use that method. However,as classes Human, Chark and Obstacle have the method teleport(), we can use duck typing. In Python, we can just simply loop through all “teleportObjects” and call their instance method, teleport().

The second scenario that python is better than java is below.

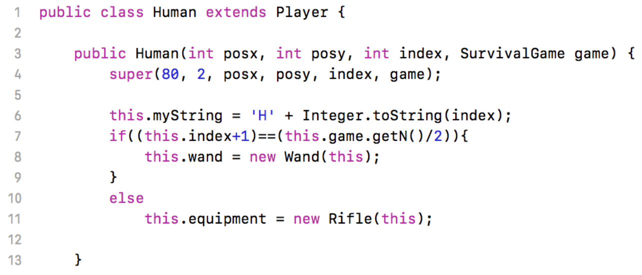
```java
    public void printBoard() {
        String printObject[][] = new String[D][D];

        // init printObject
        for (int i = 0; i < D; i++)
            for (int j = 0; j < D; j++)
                printObject[i][j] = "  ";

        for (int i = 0; i < n; i++) {
            Pos pos = ((Player)teleportObjects[i]).getPos();
            printObject[pos.getX()][pos.getY()] = ((Player)
                teleportObjects[i]).getName();
        }

        for (int i = n; i < n+O; i++) {
            Pos pos = ((Obstacle)teleportObjects[i]).getPos();
            printObject[pos.getX()][pos.getY()] = "O" + Integer.toString(i-n);
        }
```

```python
    def printBoard(self):
        printObject=[[0 for i in range(self.D)]for j in range(self.D)]
        for i in range(0,self.D):
            for j in range(0,self.D):
                printObject[i][j]="  "

        for i in range(0,self.n):
            pos=self.teleportObjects[i].getPos()
            printObject[pos.getX()]
            [pos.getY()]=self.teleportObjects[i].getName()

        for i in range(self.n,(self.n+self.O)):
            pos=self.teleportObjects[i].getPos()
            printObject[pos.getX()][pos.getY()]="O"+str(i-self.n)
```

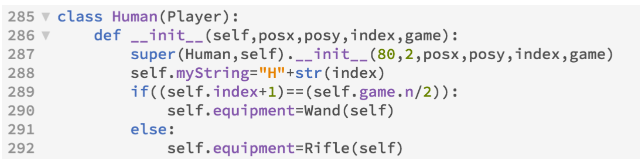
Wong Cho Hin
1155070355

In Java, although both classes Player and Obstacle have the method getPos(), they are not in parent-child-classes relation. We need to cast the type of “teleportObjects” to their corresponding type in order to call the method getPos(). But with dynamic typing in Python, we do not need to do casting.

3. I would like to show further advantages of Dynamic Typing first. In Java, as variable “equipment” in Player class was declared as type Weapon, “equipment” cannot be changed into type Wand. Since class Wand is not a children class of Weapon, I have to declare a new variable wand in Player class in order to use to in class Human. But with dynamic typing in Python, I can continue to use the variable “self.equipment”, declared in Player class, in Human class.

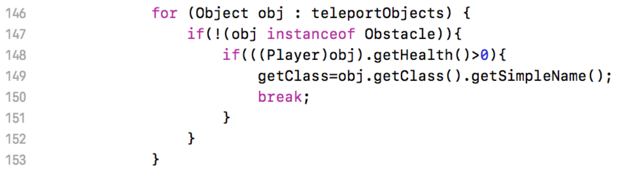


```java
public class Human extends Player {

    public Human(int posx, int posy, int index, SurvivalGame game) {
        super(80, 2, posx, posy, index, game);

        this.myString = 'H' + Integer.toString(index);
        if((this.index+1)==(this.game.getN()/2)){
            this.wand = new Wand(this);
        }
        else
            this.equipment = new Rifle(this);

    }
```

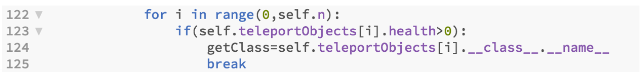


```python
class Human(Player):
    def __init__(self,posx,posy,index,game):
        super(Human,self).__init__(80,2,posx,posy,index,game)
        self.myString="H"+str(index)
        if((self.index+1)==(self.game.n/2)):
            self.equipment=Wand(self)
        else:
            self.equipment=Rifle(self)
```

Another advantage of dynamic typing is that I do not need to check the type of an object when using its instance method. In Java, as “teleportObjects” consist of type Player and type Obstacle, I have to check the type of each “teleportObjects” in order to use getHealth(), which can only be used in Player type object. But with dynamic typing in Python, I can directly access the instance variable “health” in Player only. If the type is Obstacle, Python will just ignore the condition.

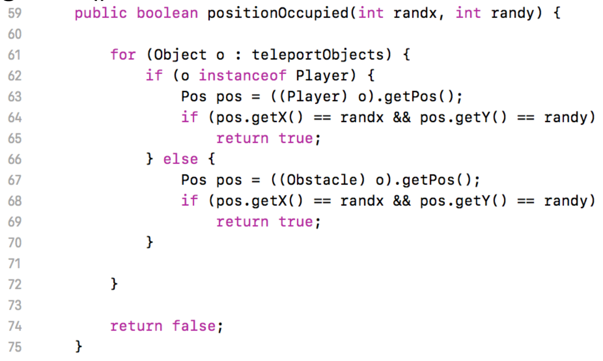


```java
            for (Object obj : teleportObjects) {
                if(!(obj instanceof Obstacle)){
                    if(((Player)obj).getHealth()>0){
                        getClass=obj.getClass().getSimpleName();
                        break;
                    }
                }
            }
```

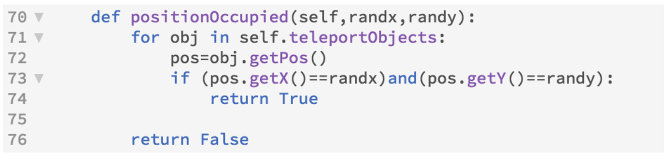


```python
            for i in range(0,self.n):
                if(self.teleportObjects[i].health>0):
                    getClass=self.teleportObjects[i].__class__.__name__
                    break
```

We can see further advantage of Duck Typing in Task4. In Java, we need to cast the variable “o” into type Player and Obstacle even though both classes Player and Obstacle have the method getPos(). But in Python, we can use duck typing since both classes Player and Obstacle have the method getPos().



```java
    public boolean positionOccupied(int randx, int randy) {

        for (Object o : teleportObjects) {
            if (o instanceof Player) {
                Pos pos = ((Player) o).getPos();
                if (pos.getX() == randx && pos.getY() == randy)
                    return true;
            } else {
                Pos pos = ((Obstacle) o).getPos();
                if (pos.getX() == randx && pos.getY() == randy)
                    return true;
            }

        }

        return false;
    }
```



```python
    def positionOccupied(self,randx,randy):
        for obj in self.teleportObjects:
            pos=obj.getPos()
            if (pos.getX()==randx)and(pos.getY()==randy):
                return True

        return False
```