Report of Wang Weixiao 1155141608

1. Advantages of Dynamic Typing:

When the type of a data in a dataset changed during runtime, like we change the value of a concept with a different data type. dynamic typing will easily handle the condition while we need to point out each of the possible types of this concept if no use dynamic typing.

Also, it also takes advantage when the possible types of a data changed during updating. E.g. If we add a new kind of occupant, we do not need to change the function that set the occupant if we use the dynamic typing, that's much simpler for coding.

Example codes: Set the occupant for a cell

Java:

```
public boolean setOccupant(Object obj) {
        if (this.getOccupant() == null) {
            this.occupant = obj;
            return true;
        } else {
            if (this.getOccupant() instanceof GameCharacter) {
                if (((GameCharacter)this.getOccupant()).interactWith(obj)) {
                    this.occupant = obj;
                    return true;
                } else {
                    return false;
                }
            } else if (this.getOccupant() instanceof Trap) {
                if (((Trap)this.getOccupant()).interactWith(obj)) {
                    this.occupant = obj;
                    return true;
                } else {
                    return false;
                }
            } else {
                return false;
            }
        }
    }
```

Python:

```
def set_occupant(self, obj):
        if self._occupant != None:
            if self._occunpant.interact_with(obj) == True:
                self._occupant = obj
                return True
            else:
                return False
```

```
        else:
            self._occupant = obj
            return True
```

2. Advantages of Duck Typing:

When we call functions with "similar format" we can use duck typing to write a more general calling method instead of list all possible conditions. That's, when data in different classes have a similar function, we can call it generally, and that's useful when the number of classes are many or varying during time or we add new classes with a similar function when updating our programs. We don't need to change relevant calling method.

Examples:

1. both Trap and character have "interactWith", but we don't need to discriminate them in python

Java:

```
if (this.getOccupant() instanceof GameCharacter) {
                    if (((GameCharacter)this.getOccupant()).interactWith(obj)) {
                        this.occupant = obj;
                        return true;
                    } else {
                        return false;
                    }
                } else if (this.getOccupant() instanceof Trap) {
                    if (((Trap)this.getOccupant()).interactWith(obj)) {
                        this.occupant = obj;
                        return true;
                    } else {
                        return false;
                    }
```

Python:

```
            if self._occunpant.interact_with(obj) == True:
                self._occupant = obj
                return True
            else:
                return False
```

2. both volcano and character has act() but we don't need to to discriminate them in python

Java:

```
if (this.actors.get(i) instanceof GameCharacter) {
                        if (((GameCharacter)this.actors.get(i)).getActive()) {
                            ((GameCharacter)this.actors.get(i)).act(this.map);
                        }
                    } else if (this.actors.get(i) instanceof Volcano) {
```

```
                    if (((Volcano)this.actors.get(i)).getActive()) {
                        ((Volcano)this.actors.get(i)).act(this.map);
                    }
                }
```

Python:

```
for obj in self._actors:
    if obj.active:
        obj.act(self._map)
```