Program Report

Name: Bo Hu

Student number: 23021010036

1. The problem statement

My program is a game about a knight whose goal is to rescue a princess. The princess is trapped in a big castle which has ten rooms, and each room has a monster. So the player control the knight to fight against the monster at each room. After player defeats a monster in one room, knight can choose a reward and add it to backpack, then choose a next room to go. A reward is a weapon or a potion, which is an item knight can use in the next fight to increase the probability of win. Knight will successfully rescue the princess and win the game after defeating all the ten monsters in each rooms.

Here is some details of the game:

1. A fight might have many rounds. Before each round the knight can choose an item to be used in this round. Items are weapons and potions. A weapon can cause higher damage on monster and a potion can restore knight's hp.
2. If the knight wins the fight in a room, he can choose a reward provided by the room. Each reward item has its attribute `weight` and the knight has a attribute `capacity`. The total weight of all the items in knight's backpack can't exceed the capacity. Knight can drop items to satisfy the capacity.
3. The neighboring relationship between rooms is depicted in the *Picture 1.* below. If knight wins room A and choose room B to continue adventure, he can return back to room A without repeating the fight here if he fails the fight in room B. If knight returns back to room A, he can choose another neighbor room to continue his adventure, like room C.

More details are introduced below. To save space I omit some text based attribute like `description`.

1.1. Monsters

Table 1. The details of monsters.

name	hp	attack	level
Witch	4	1	1
Ogre	6	2	2
Vampire	9	3	3
Night Demon	12	4	4
Dark elf	15	5	5
Werewolf	18	6	6
Zombie	21	7	7
Warlock	24	8	8
Ghost	27	9	9
Dark Dragon	30	10	10

1.2. Rooms

Table 2. The details of rooms.

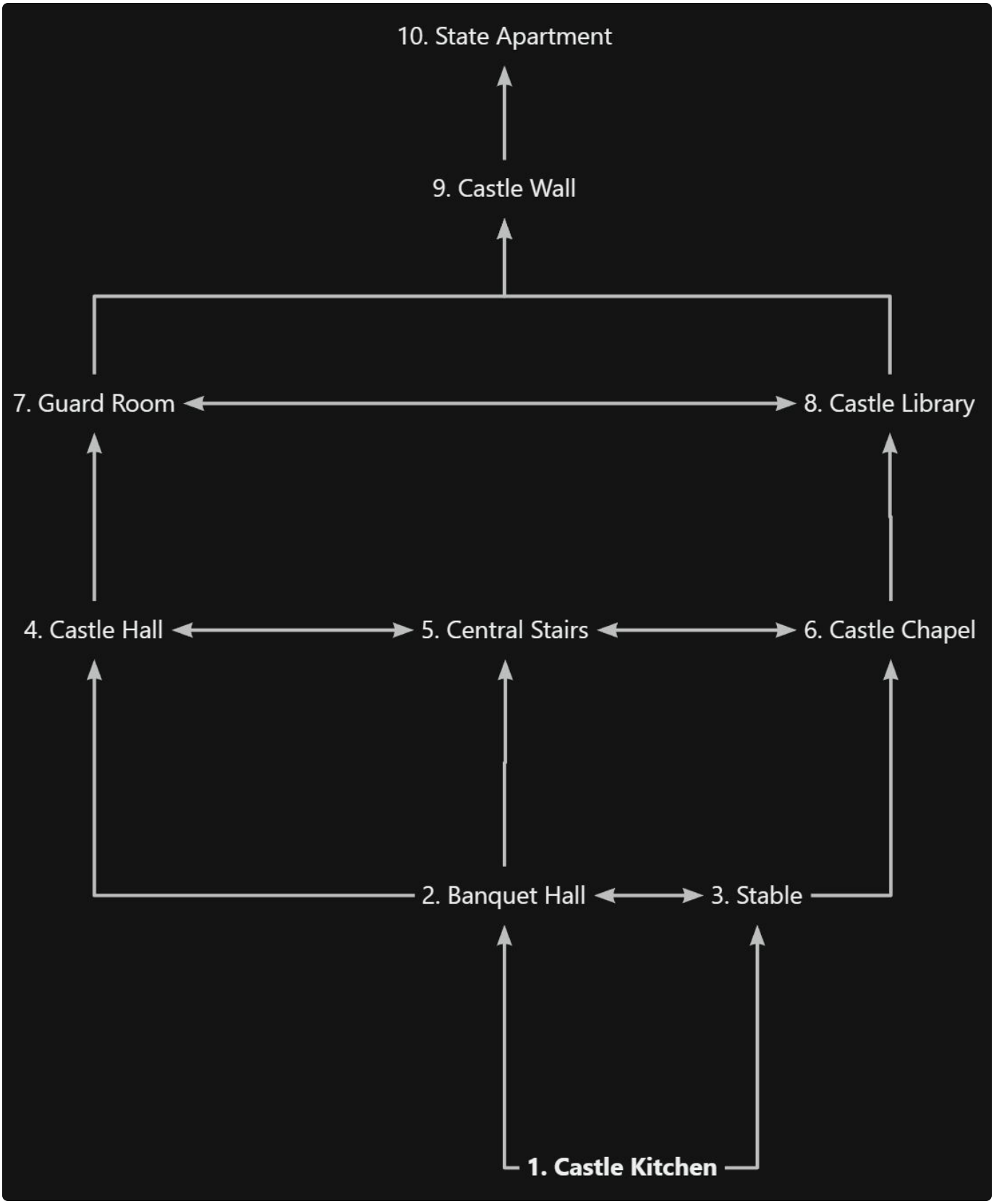
> Each room has its monster and a list of rewards for knight to choose.

name	monster	rewards
Castle Kitchen	Witch	1. British Hunting Knife 2. Hatchet
Banquet Hall	Ogre	1. Apple flavored potion 2. Stiletto sword
Stable	Vampire	1. Tang Sword 2. Tablet
Guard Hall	Night Demon	1. Roman Short sword 2. Battle Axe

Central Stairs	Dark elf	1. Elixir 2. Long sword
Castle Chapel	Werewolf	1. Longbow 2. stimulant
Guard Room	Zombie	1. Composite Bow 2. Recurve Bow
Castle Library	Warlock	1. Pill 2. Crossbow 3. Ointment
Castle Wall	Ghost	1. War Hammer 2. Berken hatchet 3. Bomb
State Apartments	Dark Dragon	

Graph 1. The neighbor relationships of rooms.

> Knight starts with `Castle Kitchen` and the final BOSS is in `State Apartment`.



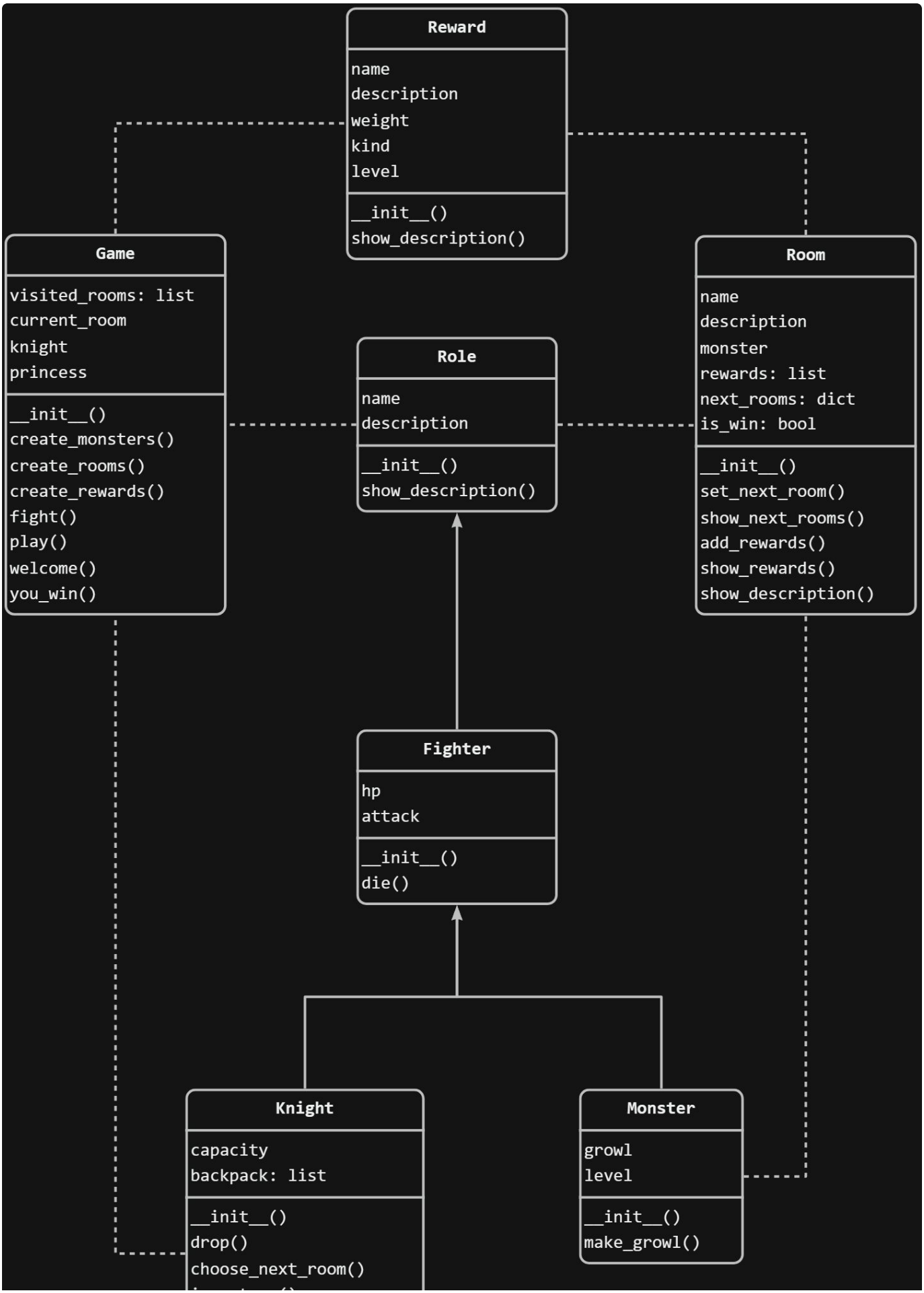
1.3. Rewards

Table 3. The details of rewards.

> A reward is a weapon or potion with its weight. The `level` is corresponding to its effect.

name	weight	kind	level
British Hunting Knife	1	weapon	1
Apple flavored potion	1	potion	1
Hatchet	2	weapon	2
Stiletto sword	2	weapon	2
Tang Sword	3	weapon	3
Tablet	3	potion	3
Roman Short sword	4	weapon	4
Battle Axe	4	weapon	4
Elixir	5	potion	5
Long sword	5	weapon	5
Longbow	6	weapon	6
stimulant	6	potion	6
Composite Bow	7	weapon	7
Recurve Bow	7	weapon	7
Pill	8	potion	8
Crossbow	8	weapon	8
Ointment	8	potion	9
War Hammer	9	weapon	9
Berken hatchet	9	weapon	10
Bomb	9	weapon	10

2. The class diagram


Reward
name
description
weight
kind
level
__init__()
show_description()
Game
visited_rooms: list
current_room
knight
princess
__init__()
create_monsters()
create_rooms()
create_rewards()
fight()
play()
welcome()
you_win()
Role
name
description
__init__()
show_description()
Room
name
description
monster
rewards: list
next_rooms: dict
is_win: bool
__init__()
set_next_room()
show_next_rooms()
add_rewards()
show_rewards()
show_description()
Fighter
hp
attack
__init__()
die()
Knight
capacity
backpack: list
__init__()
drop()
choose_next_room()
Monster
growl
level
__init__()
make_growl()

3. Description of my classes

I delete the whole python file `TextUI.py` in the start code and rewite a new `class Room` in `Room.py`. And I add some attributes and methods in `class Game` in `Game.py` which you can check in my diagram above.

I create a new `Reward.py` to implement a `class Reward`, and every reward item after winning a room is an instance of this class.

I create a new `Role.py` to implement four classes: `class Role`, `class Fighter`, `class Knight` and `class Monster`. The specific roles in the game are instances of these classes.

- `class Role` is a super-class of the other three classes.
- `class Fighter` is a sub-class of `class Role` and it intialize a fighter. Both knight and monster are fighters so `class Fighter` is also the super-class of `class Knight` and `class Monster`.
- `class Knight` is a sub-class of `class Fighter` and it initialize a knight. A knight participates in the fight with monsters and a knight has a special attribute `capacity`, which is the max weight the knight can afford. In this class I design some methods to implement some actions the knight have to do in the game, like the method `choose_next_room()` to go to the next room and `inventory()` to inventory backpack.

I create a new `Errors.py` to implement some exception handling.

I create a new `Test.py` to implement some automated unit testings.

4. Problems I encounterd

I first draw a diagram and write my codes according to it. But with more classes and methods written I realize that the mechanism of the game is not so simple as what I have designed before. In the following work I create more methods to implement every important mechanism which I

think will make the game more interesting. Finally the new diagram according to my codes is much more complex than the original version.

Another problem is the settings of fight. The proper values of hp and attack of monsters are corresponding to the degree of difficulty of the game, and I have tried a lot to work out a series of proper settings to make the game interesting.

5. Exception handling

Player will interact with the game by entering number or string in the command line. For example when choosing a reward after winning a room, player have to enter a reward id to add the item in backpack. However if the id number player entered is not in the reward list provided by the room, the game will raise an error `RewardIdNotExistError` and be handled.

I have created three error classes in `Errors.py` and write corresponding `try...except...` codes to implement the exception handling. It will make sure that whatever the player enters in the command line the game will not be interrupted by errors.

- `ItemIdNotExistError` and `RewardIdNotExistError`: When player enters an id number while choosing an item in a fight round or choosing a reward after winning, make sure the id player entered must have corresponding item.
- `DirectionNotExistError`: When player enter a direction to choose the next room, make sure the entered direction is existing.

6. Automated Unit Testing

I write a `class Test` in `Test.py` to implement three automated unit tests. The method `Set Up()` initialize some instances before the test and `TearDown()` recover the environment after the test finished.

- `test_monster()`: Test whether the monster has the right growl.
- `test_room()`: Test whether the rooms have the right corresponding monster, rewards and next rooms.
- `test_knight()`: Test whether the relationship between the knight's capacity and the total weight of his backpack.