PROJECT SPECIFICATION

TEXT BASED ADVENTURE GAME

INTRODUCTION

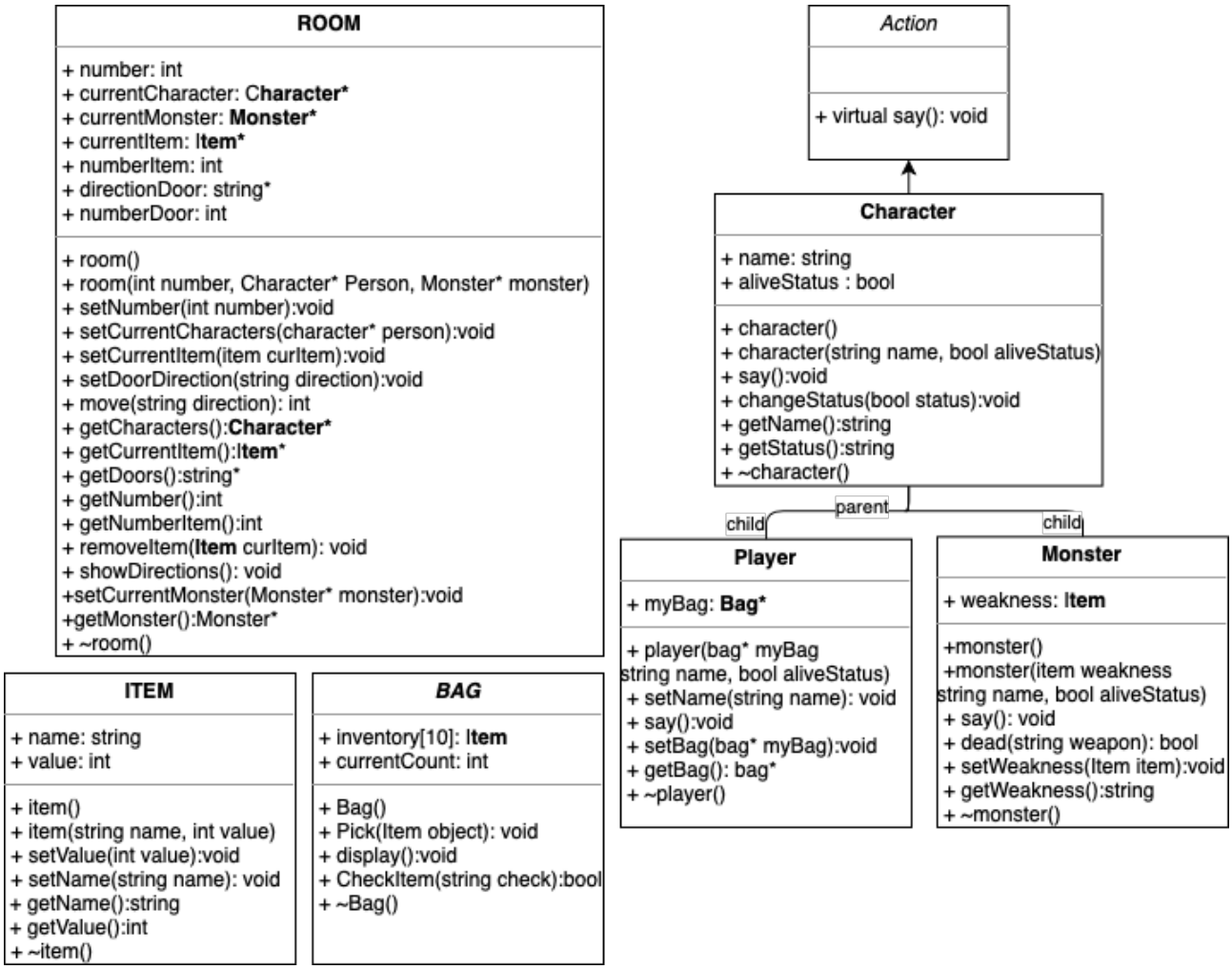
This is an a text based adventure game. There is a castle with 9 rooms and your are the hero tasked to rescue the princess who is locked in room 9. You can navigate through different rooms and pick the various item to defeat the monsters and gain valuable items. If you encounter a monster it will not attack you unless you attack the monster. Killing the monster will grant you access to the locked paths which hold far greater treasure that you have located till now. One of the monsters is Dracula in room 6 which can be killed using a dagger found in room 7. There is also a medusa in room 5 which can be killed using a shield found in room 3. Use four simple commands to navigate through the room and win the game.

DESIGN DESCRIPTION

ASSESSMENT CONCEPTS

1. Memory allocation from the **stack** and the **heap**
    - Arrays
    - Strings
    - Objects
2. User Input and Output
    - I/O of different data types
3. Object-oriented programming and design
    - Inheritance
    - Polymorphism
    - Abstract Classes
4. Testing
    - Test Inputs / Expected Outputs
    - Automated Testing

CLASS DIAGRAM



CLASS DESCRIPTIONS

Class Bag:

- **Attributes:**
  - Inventory[10]: item - stores all the items in an array
  - currentCount: count variable
- **Methods:**
  - Bag() - default constructor
  - Pick: add an object to the inventory array
  - Display: show all current objects
  - CheckItem: check is the item in the parameter is present in the inventory
  - ~Bag(): Destructor

Class Item:

- **Attributes:**
    - name: string - stores the name of the item
    - values: int - stores the value of the item
- **Methods:**
    - item() - default constructor
    - item(string name, int value, room currentRoom) - parameterized constructor
    - setName() - set function for name
    - getName() - get function for name
    - setValue() - set function for value
    - getValue() - get function for value
    - ~item() - Destructor

Class Room:

- **Attributes:**
    - number: int - number of the room
    - currentCharacter: character* - pointer to array storing all the character in room
    - currentMonster: monster* - pointer to array storing all the monster in room
    - currentItem[10]: item * - pointer to array storing all the item in room
    - numberItem: int - count variable
    - directionDoor[]: string
    - numberDoor: int - count variable
    - numberCharacter - count variable
- **Methods:**
    - room() - default constructor
    - room(int number) - parameterized constructor
    - setNumber(int number):void - set function for number
    - setCurrentCharacters(character char):void - set function for currentCharacter
    - setCurrentItem(item curItem):void - set function for currentItem
    - setDoorDirection(string direction):void - set function for directionDoor

- move(string direction): int - moves the player to a different room and return the room number. The function checks for the doors.
- getCharacters():**Character*** - get function for currentCharacter
- getCurrentItem():I**tem*** - get function for currentItem
- getDoors():string* - get function for directionDoor
- getNumber():int - get function for number
- removeItem(**Item** curItem **Character** person) - removes an item from the array
- setCurrentMonster(Monster* monster):void - get function for currentMonster
- getMonster():monster* - get function for Monster
- ~room() - Destructor

Class Character:

- **Attributes:**
  - name: string - stores the name of the Character
  - aliveStatus: bool - stores a bool value for alive status true means the character is alive
- **Methods:**
  - character() - default constructor
  - character(string name, bool aliveStatus) - parameterized constructor
  - say():void - return a sentence
  - changeStatus(bool status):void - changes the alive status
  - getName():string - get function for name
  - getStatus():string - get function for aliveStatus
  - ~character() - Destructor

Class Player:

- **Attributes:**
  - myBag* bag - stores the bag for the character
- **Methods:**
  - player(bag* myBag, string name, bool aliveStatus) - parameterized constructor

- setName(string name): void - set function for name
- say():void - virtual function
- setBag(Bag* myBag):void - set function for bag
- getBag(): bag* - get function for bag
- ~player() - Destructor

Class Character:

- **Attributes:**
  - myBag: bag - stores the bag for the character
- **Methods:**
  - monster() - default constructor
  - monster(item weakness, string name, bool aliveStatus) - parameterized constructor
  - say(): void - virtual function
  - dead(I**tem** weapon): bool - check if the monster can be killed with the weapon
  - setWeakness():void - set function for weakness
  - getWeakness():string - get function for weakness
  - ~monster() - Destructor

USER INTERFACE

A user is the player the game. They use the command-line. Users are presented with a list of actions they can perform in the game. The scenario is well explained and the commands they can use are clear.

```
                                                  ****************************
                                                  *   WELCOME TO ZELDA GAME  *
                                                  ****************************
Please enter your name:
player

These are some basic commands:
move then a direction to move to a new room
pick then item name to pick an item
attack then monster to attack the monster
exit then " Game " to exit the game

The princess of the kingdom of CPeria Princess Zelda. Has been captured by an evil wizard and placed in a castle gu
arded by monsters.
Your job is to set her free and bring her back outside the castle.
There are 9 rooms in the castle and one exit to it . Rooms have treasure and weapons too.
so do you have the potential to destroy the monsters and save Princess Zelda ?
Are you ready player?
Lets do it !!

Currently you are in Room 1. Navigate to different rooms and find items to defeat the monster and rescue the prince
ss.
There is a room to your: EAST.
There is a room to your: SOUTH.
There is a room to your: WEST.
The bag contains: 0
Currently, your bag is empty.
Current Cash : 0
```

CODE STYLE

All code in the program will be properly indented using 4 space tabs. Classes will begin with a capital letter. Function Comments will be given for each function that describes what the function does, what the return type represent and what any arguments are available for the function. Code Comments will be used to describe what each block of code performs if it is not obvious from just reading the code or the code is short. Comments will be written as the code is written.

Testing Plan

Unit Testing

Our unit tests will cover all public functions in our classes. Each class will have a corresponding test file like ClassName_Main.cpp, which will include a main method that will exhaustively test each function with a set of inputs. The tests will be laid out so that each function is tested in different scenarios.

Testing main

The main file will test three test conditions. Each condition will have a input and will be matched with an expected output. The fist condition test a number of cases with wrong input. The second test is a correct version which focuses on completion of the game. The condition focuses on mid termination of the game which is if the player wants to quit the game in between. The test for main runs each time we compile the files.

Schedule Plan

STRETCH GOALS

Our goal is to complete the whole project by week 8. The next two weeks will be dedicated to testing different cases and improving the user interface of the program and adding additional functionality if needed.

BREAK WEEK 1

Complete the class diagram and divide the work between the members and complete the classes.

BREAK WEEK 2

Start the main file and the function file together during joint sessions and add any new methods to the class if needed. Also comment the code will writing it.

WEEK 9

Complete the main file and prepare a makefile and discuss different strategies to test the program.

WEEK 10

Start creating the test file for the classes each member created and test the main file.

WEEK 10

Prepare for the final presentation and work on the user interface.