Problem Set 4

Contents

Task 1

Description

This is a simple game program where the player is meant to go through a set of rooms in the dungeon. The player's permission to go through a certain room is accessed through a set of conditions which determine whether or not the user can access the room. The player starts the game in the entry room and if the player is able to slay more than three monsters then they qualify for access both the left (nextRoomOne) and right node (nextRoomTwo) otherwise they can only access the room represented by the left node(nextRoomOne). This is the challenge on level one.

On Level Two the player has four challenges but may choose at least one to tackle. Each condition is link to permission to access specified room (see Figure 1) if the player manages to satisfy all four conditions then they can easily move around the whole dungeon freely with access to any room. The game is considered solved at this point.

Diagram

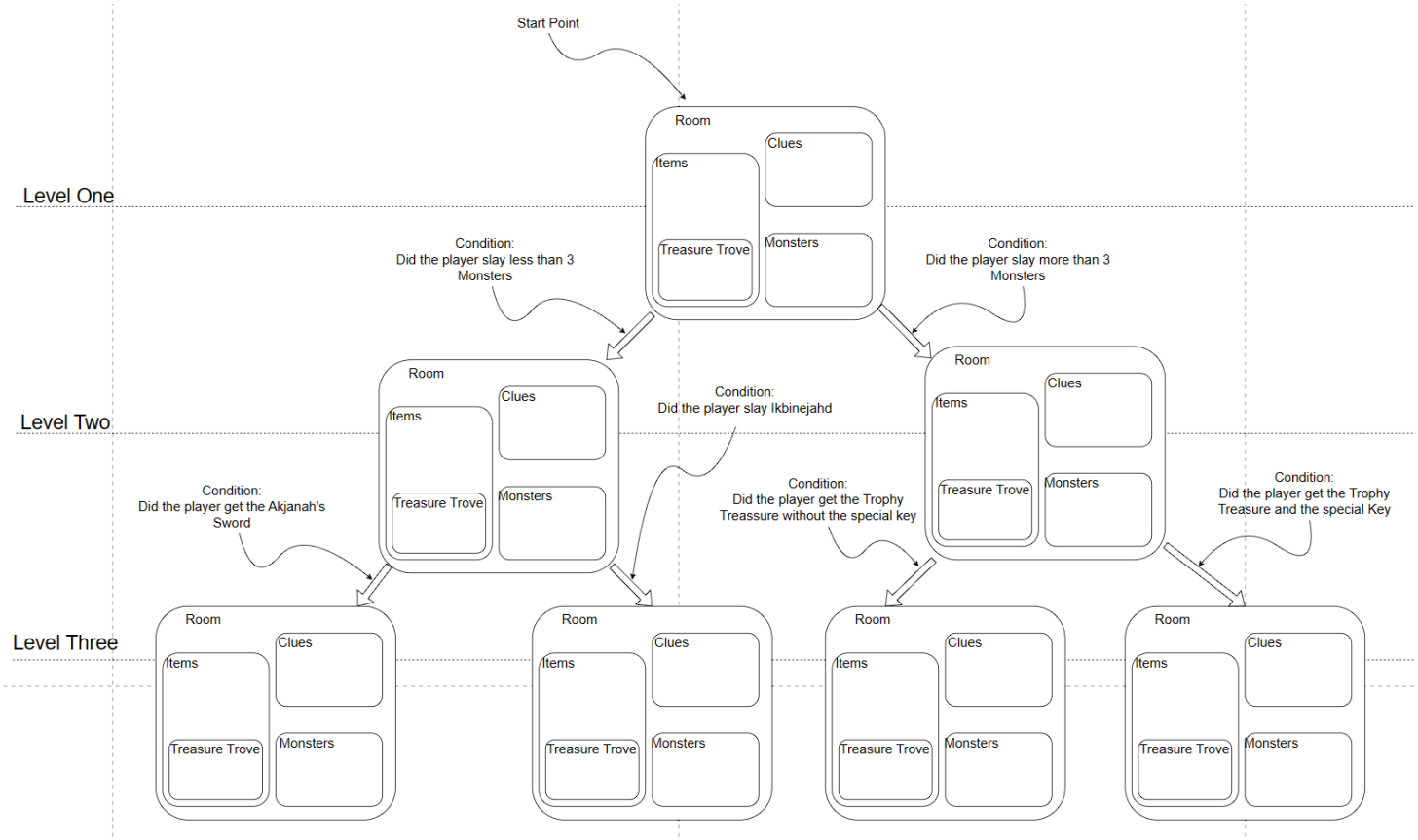
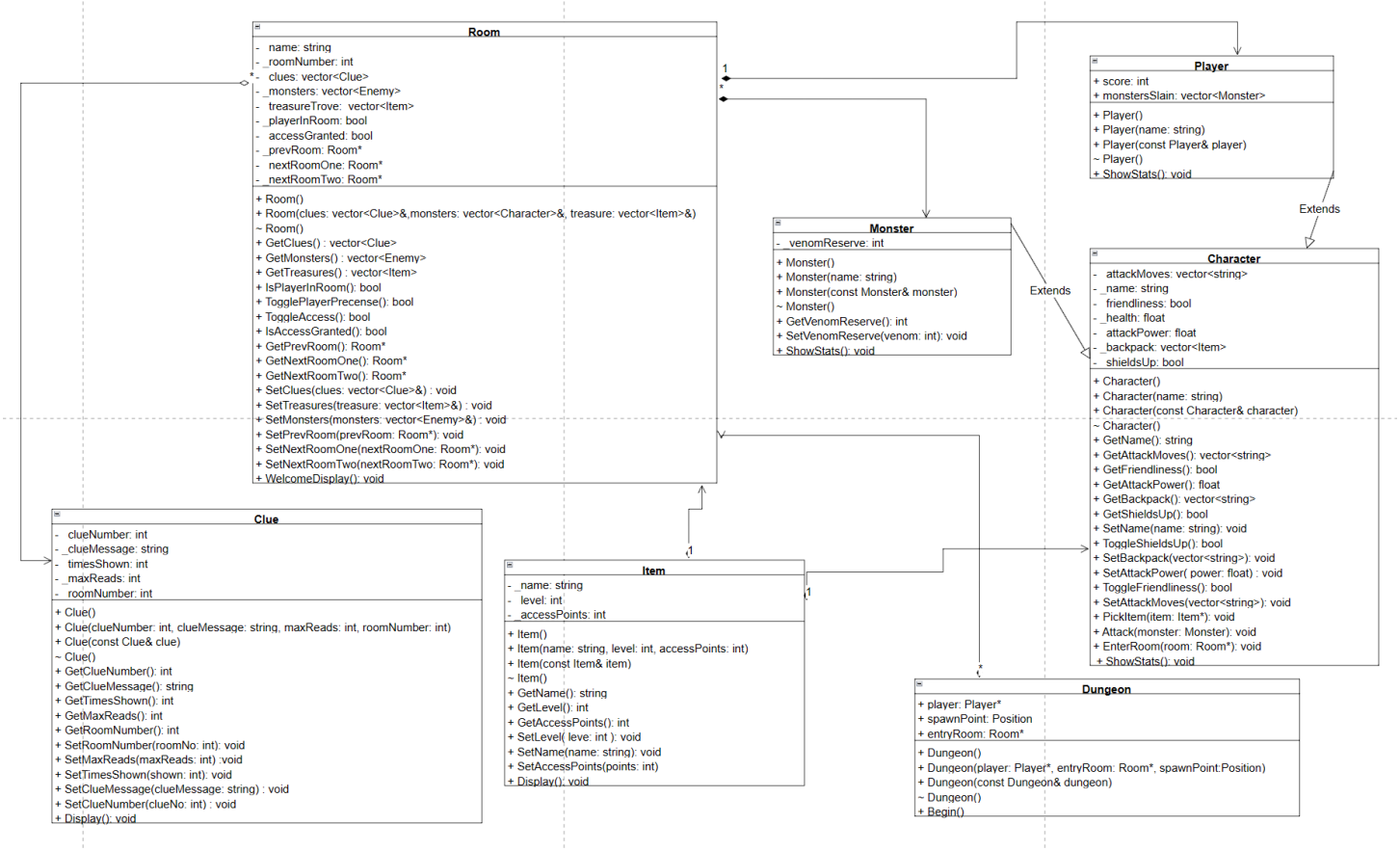


Figure 1 The Tree representation of a Dungeon

Task 2

UML Diagram



Task 3

Code Implementation

```cpp
#include "Dungeon.h"
#include <algorithm>
#include <iostream>

Dungeon::Dungeon()
{
    _player = new Player();

    std::vector<std::string> roomNames = {
        "Entry Room", "Room 1", "Room 2", "Room 3",
        "Room 4", "Room 5", "Room 6", "Room 7"
    };
    // Level 0 Rooms
    _entryRoom = new Room(roomNames[0], roomNumber: 10);
    // Level 1 Rooms
    Room* room1 = new Room(roomNames[1], roomNumber: 20);
    Room* room2 = new Room(roomNames[2], roomNumber: 22);
    // Level 2 Rooms
    Room* room3 = new Room(roomNames[3], roomNumber: 30);
    Room* room4 = new Room(roomNames[4], roomNumber: 31);
    Room* room5 = new Room(roomNames[5], roomNumber: 32);
    Room* room6 = new Room(roomNames[6], roomNumber: 33);
    _entryRoom->SetNextRoomOne(room1);
    _entryRoom->SetNextRoomTwo(room2);
    // room1
    room1->SetPrevRoom(_entryRoom);
    room1->SetNextRoomOne(room3);
    room1->SetNextRoomTwo(room4);
    // room2
    room2->SetPrevRoom(_entryRoom);
    room2->SetNextRoomOne(room5);
    room2->SetNextRoomTwo(room6);
    // room3
    room3->SetPrevRoom(room1);
    // room4
    room4->SetPrevRoom(room1);
    // room5
    room5->SetPrevRoom(room2);
    // room6
    room6->SetPrevRoom(room2);

    _spawnPoint = { .x: 0, .y: 0 };
}
```

```cpp
void Dungeon::DFSTraversal(Room* room)
{
    if (room == Room::Sentinel)
    {
        return;
    }


    std::cout << room->GetName() << std::endl;
    DFSTraversal(room->GetNextRoomOne());
    DFSTraversal(room->GetNextRoomTwo());
}

void Dungeon::Traverse(Room* room)
{
    while(true)
    {
        // room->WelcomeMessage();
        std::cout << "You are in " << room->GetName() << std::endl;
        std::cout << "You have the following options: " << std::endl;
        std::cout << "1. Go to " << (
            (room->GetNextRoomOne() != Room::Sentinel)
            ? room->GetNextRoomOne()->GetName()
            : "None (This room is an end point. You'll need to go back)") << std::endl;
        std::cout << "2. Go to " << (
            (room->GetNextRoomTwo() != Room::Sentinel)
            ? room->GetNextRoomTwo()->GetName()
            : "None (This room is an end point. You'll need to go back)") << std::endl;
        std::cout << "3. Go back to " << (
            (room->GetPrevRoom()!= Room::Sentinel)
            ? room->GetPrevRoom()->GetName()
            : "None (This is the entry Room)") << std::endl;
        std::cout << "4. Quit" << std::endl;
        std::cout << "Enter a command: " << std::endl;
        std::string response;
```

```cpp
        std::cout << "Enter a command: " << std::endl;
        std::string response;
        getline([&]std::cin, [&]response);

        if (response == "1")
        {
            if (room->GetNextRoomOne() == Room::Sentinel)
            {
                std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;


            }
            else
            {
                room = room->GetNextRoomOne();
            }
        }
        else if (response == "2")
        {
            if (room->GetNextRoomTwo() == Room::Sentinel)
            {
                std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;


            }
            else
            {
                room = room->GetNextRoomTwo();
            }
        }
        else if (response == "3")
        {
            room = room->GetPrevRoom();
        }
        else if (response == "4")
```

```
175         else if (response == "4")
176         {
177             std::cout << "Quitting game ..." << std::endl;
178             break;
179         }
180         else
181         {
182             std::cout << "Invalid command" << std::endl;
183         }
184     }
185 }
186
187 void Dungeon::Task3()
188 {
189     auto* dungeon = new Dungeon();
190
191     std::cout << "=============================================================" << std::endl;
192     std::cout << "============    TASK 3 [START]   ============================" << std::endl;
193     std::cout << "=============================================================" << std::endl;
194     std::cout << "You have the following rooms to traverse and explore" << std::endl;
195
196     dungeon->DFSTraversal(dungeon->_entryRoom);
197
198     std::cout << "We Start in the entry Room: " << std::endl;
199     std::cout << std::endl;
200     dungeon->_entryRoom->WelcomeMessage();
201     dungeon->Traverse(dungeon->_entryRoom);
202     std::cout << "============    TASK 3 [END]   ============================" << std::endl;
203 }
```

Output

```
You are in Entry Room
You have the following options:
1. Go to Room 1
2. Go to Room 2
3. Go back to None (This is the entry Room)
4. Quit
Enter a command:
1
You are in Room 1
You have the following options:
1. Go to Room 3
2. Go to Room 4
3. Go back to Entry Room
4. Quit
Enter a command:
1
You are in Room 3
You have the following options:
1. Go to None (This room is an end point. You'll need to go back)
2. Go to None (This room is an end point. You'll need to go back)
3. Go back to Room 1
4. Quit
Enter a command:
3
You are in Room 1
You have the following options:
1. Go to Room 3
2. Go to Room 4
3. Go back to Entry Room
4. Quit
Enter a command:
2
You are in Room 4
You have the following options:
1. Go to None (This room is an end point. You'll need to go back)
2. Go to None (This room is an end point. You'll need to go back)
3. Go back to Room 1
4. Quit
Enter a command:
3
You are in Room 1
You have the following options:
1. Go to Room 3
2. Go to Room 4
3. Go back to Entry Room
4. Quit
```

```
You are in Room 1
You have the following options:
1. Go to Room 3
2. Go to Room 4
3. Go back to Entry Room
4. Quit
Enter a command:
3
You are in Entry Room
You have the following options:
1. Go to Room 1
2. Go to Room 2
3. Go back to None (This is the entry Room)
4. Quit
Enter a command:
2
You are in Room 2
You have the following options:
1. Go to Room 5
2. Go to Room 6
3. Go back to Entry Room
4. Quit
Enter a command:
1
You are in Room 5
You have the following options:
1. Go to None (This room is an end point. You'll need to go back)
2. Go to None (This room is an end point. You'll need to go back)
3. Go back to Room 2
4. Quit
Enter a command:
3
You are in Room 2
You have the following options:
1. Go to Room 5
2. Go to Room 6
3. Go back to Entry Room
4. Quit
Enter a command:
2
You are in Room 6
You have the following options:
1. Go to None (This room is an end point. You'll need to go back)
2. Go to None (This room is an end point. You'll need to go back)
3. Go back to Room 2
4. Quit
Enter a command:
```

```
You are in Room 6
You have the following options:
1. Go to None (This room is an end point. You'll need to go back)
2. Go to None (This room is an end point. You'll need to go back)
3. Go back to Room 2
4. Quit
Enter a command:
3
You are in Room 2
You have the following options:
1. Go to Room 5
2. Go to Room 6
3. Go back to Entry Room
4. Quit
Enter a command:
3
You are in Entry Room
You have the following options:
1. Go to Room 1
2. Go to Room 2
3. Go back to None (This is the entry Room)
4. Quit
Enter a command:
quit
Invalid command
You are in Entry Room
You have the following options:
1. Go to Room 1
2. Go to Room 2
3. Go back to None (This is the entry Room)
4. Quit
Enter a command:
Quit
Invalid command
You are in Entry Room
You have the following options:
1. Go to Room 1
2. Go to Room 2
3. Go back to None (This is the entry Room)
4. Quit
Enter a command:
4
Quitting game...
============    TASK 3 [END]   ==============
How would you like to proceed? Please type 'help' to view commands
```

Task 4

Code Implementation

```
27  class Player : public Character
28  {
```

```
90
91      void AcceptVisitor(PlayerVisitor* visitor);
92  };
93
```

```
105 class PlayerVisitor
106 {
107 public:
108     void VisitPlayer(Player* player);
109 };
```

```cpp
void Player::AcceptVisitor(PlayerVisitor* visitor)
{
    visitor->VisitPlayer(this);
}

void PlayerVisitor::VisitPlayer(Player* player)
{
    std::cout << "Player Visitor Upgraded Health to: " << player->GetHealth() + 30 << std::endl;
    player->SetHealth(player->GetHealth() + 30);
    player->SetScore(player->GetScore() + 40);
    std::cout << "################################################" << std::endl;
    std::cout << "\tPlayer Name: " << player->GetName() << "\tHealth: " << player->GetHealth() << std::endl;
    std::cout << "################################################" << std::endl;
    std::cout << "Attack Moves: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& move :string& : player->GetAttackMoves())
    {
        std::cout << "}⇒" << move << std::endl;
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Friendliness: " << ((player->GetFriendliness()) ? "Friendly" : "Not Friendly") << std::endl;
    std::cout << "Backpack: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& item : player->GetBackpack())
    {
        item.Display();
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Shields Up: " << ((player->GetShieldsUp()) ? "Shielded" : "Not Shielded") << std::endl;
    std::cout << "Player Attack Power: " << player->GetAttackPower() << std::endl;
    std::cout << "Player Score Updated (by the visitor): " << player->GetScore() << std::endl;
    std::cout << "Player Monsters Slain: " << player->GetMonstersSlain() << std::endl;
}
```

Output

```
==================================================
======        WELCOME TO THE DUNGEON        =========
==================================================
How would you like to proceed? Please type 'help' to view commands
task4
================================================
============    TASK 4 [START]   ==============
================================================
Player Visitor Upgraded Health to: 130
##################################################
        Player Name: Default Player Name        Health: 130
##################################################
Attack Moves:
----------------------------
}=>Double Jab Punch
}=>[COMBO] Low Right Kick, Right Jab Punch
}=>Tackle
}=>Round Kick, Floor Opponent
----------------------------
Friendliness: Not Friendly
Backpack:
----------------------------
----------------------------
Shields Up: Not Shielded
Player Attack Power: 50
Player Score Updated (by the visitor): 40
Player Monsters Slain: 0
============    TASK 4 [END]   ==============
How would you like to proceed? Please type 'help' to view commands
```

Reflection

1. **Did you run into any problems while completing this PS? How did you solve it? What references you used?**
    a. I encountered no issues working on problem set 4; in fact, t's been much easier in comparison to my other problem sets I attribute this improvement of my skill to having to battle with my programming project which was pretty challenging but bringing on the lessons from that project onto this on, I've had a much easier time.
2. **Comment on your experience using the Tree ADT and Visitor pattern (how challenging was it to learn, was it practical for your program, what else can you use it for, etc.)**
    a. I didn't find it challenging, as mentioned this is due to having learnt the hard way during my programming project. I now know when it is a good idea to use pointers and when they might become a source of pain. I've used what I learnt in my programming project to implement this task

Code Reference

Item.h Declarations

```
24  class Item
25  {
26  private:
27      std::string _name;
28      int _level;
29      int _accessPoints;
30
31  public:
32      /**
33       * @brief This is the default constructor.
34       */
35      Item();
36
37      /**
38       * @brief This is the overloaded constructor.
39       * @param name - the name of the item.
40       * @param level - the level of the item.
41       * @param accessPoints - the access points the player gets from picking up the item.
42       */
43      Item(std::string& name, int level, int accessPoints);
44
45      /**
46       * @brief Ths is the copy constructor.
47       * @param item - the item to be copied.
48       */
49      Item(const Item& item);
50
51      /**
52       * @brief This is the destructor.
53       */
54      ~Item();
```

```
56      /**
57       * @brief Getter for the _name property.
58       * @return a string representing the name of the item.
59       */
60      std::string& GetName();
61
62      /**
63       * @brief Setter for the _name property.
64       * @param name - the name of the item.
65       */
66      void SetName(std::string& name);
67
68      /**
69       * @brief Getter for the _level property.
70       * @return a string representing the level of the item.
71       */
72      int GetLevel() const;
73
74      /**
75       * @brief Setter for the _level property.
76       * @param level - the level of the item.
77       */
78      void SetLevel(int level);
79
80      /**
81       * @brief Getter for the _accessPoints property.
82       * @return an integer representing the access points the player gets from picking up the item.
83       */
84      int GetAccessPoints() const;
```

```
86      /**
87       * @brief Setter for the _accessPoints property.
88       * @param accessPoints - the access points the player gets from picking up the item.
89       */
90      void SetAccessPoints(int accessPoints);
91
92      /**
93       * @brief This method displays the details of the item.
94       */
95      void Display() const;
96  };
```

Clue.h

```
25  class Clue
26  {
27  private:
28      int _clueNumber;
29      std::string _clueMessage;
30      int _timesShown;
31      int _maxReads;
32      int _roomNumber;
33
34  public:
35      /**
36       * @brief This is the default constructor.
37       */
38      Clue();
39
40      /**
41       * @brief This is the overloaded constructor.
42       * @param clueNumber - hte clue number.
43       * @param clueMessage - the clue message.
44       * @param maxReads - the maximum number of times the clue can be read.
45       * @param roomNumber - the room number in which the clue can be read.
46       */
47      Clue(int clueNumber, std::string& clueMessage, int maxReads, int roomNumber);
48
49      /**
50       * @brief This is the copy constructor.
51       * @param clue - the clue to be copied
52       */
53      Clue(const Clue& clue);
54
55      /**
56       * @brief This is the destructor.
57       */
58      ~Clue();
```

```
60      /**
61       * @brief Getter for the clue number.
62       * @return an integer representing the clue number.
63       */
64      int GetClueNumber() const;
65
66      /**
67       * @brief Setter for the clue number.
68       * @param clueNumber - the desired clue number.
69       */
70      void SetClueNumber(int clueNumber);
71
72      /**
73       * @brief Getter for the clue message.
74       * @return a string representing the clue message.
75       */
76      std::string& GetClueMessage();
77
78      /**
79       * @brief Setter for the clue message.
80       * @param clueMessage - the desired clue message.
81       */
82      void SetClueMessage(std::string& clueMessage);
83
84      /**
85       * @brief Getter for the number of times the clue has been shown.
86       * @return an integer representing the number of times the clue has been shown.
87       */
88      int GetTimesShown() const;
89
90      /**
91       * @brief Setter for the number of times the clue has been shown.
92       * @param timesShown - the desired number of times the clue has been shown.
93       */
94      void SetTimesShown(int timesShown);
```

```
96      /**
97       * @brief (Read only) Getter for the maximum number of times the clue can be read.
98       * @return an integer representing the maximum number of times the clue can be read.
99       */
100     int GetMaxReads() const;
101
102     /**
103      * @brief Getter for the room number in which the clue can be read.
104      * @return an integer representing the room number in which the clue can be read.
105      */
106     int GetRoomNumber() const;
107
108     /**
109      * @brief Setter for the room number in which the clue can be read.
110      * @param roomNumber - the desired room number in which the clue can be read.
111      */
112     void SetRoomNumber(int roomNumber);
113
114     /**
115      * @brief This function displays the clue.
116      */
117     void Display() const;
118 };
```

Enemy.h

```
24  class Enemy : public Character
25  {
26  private:
27      int _venomReserve;
28
29  public:
30      /**
31       * @brief This is the default constructor.
32       */
33      Enemy();
34
35      /**
36       * @brief This is the overloaded constructor.
37       * @param name - the name of the enemy.
38       * @param venomReserve - the amount of venom the enemy has.
39       */
40      Enemy(std::string& name, int venomReserve);
41
42      /**
43       * @brief This is a copy constructor that takes an Enemy type.
44       * @param enemy - the enemy to be copied.
45       */
46      Enemy(const Enemy& enemy);
47
48      /**
49       * @brief The is a copy constructor that takes a Character type.
50       * @param character - the character to be copied.
51       * @param venomReserve - the amount of venom the enemy has.
52       */
53      Enemy(const Character& character, int venomReserve);
54
55      /**
56       * @brief This is the destructor.
57       */
58      ~Enemy() override;
```

```
60      /**
61       * @brief Getter for the _venomReserve property.
62       * @return an integer representing the amount of venom the enemy has.
63       */
64      int GetVenomReserve() const;
65
66      /**
67       * @brief Setter for the _venomReserve property.
68       * @param venomReserve - the amount of venom the enemy has.
69       */
70      void SetVenomReserve(int venomReserve);
71
72      /**
73       * @brief This method displays the enemy's stats.
74       */
75      void ShowStats() const override;
76
77
78  };
```

Player.h Declarations

```
27  class Player : public Character
28  {
29  private:
30      int _score;
31      int _monstersSlain;
32
33  public:
34      /**
35       * @brief This is the default constructor.
36       */
37      Player();
38
39      /**
40       * @brief This is the overloaded constructor.
41       * @param name - the name of the player.
42       */
43      Player(std::string& name);
44
45      /**
46       * @brief This is a copy constructor that takes a Player type.
47       * @param player - the player to be copied.
48       */
49      Player(const Player& player);
50
51      /**
52       * @brief This is a copy constructor that takes a Character type.
53       * @param character - the character to be copied.
54       */
55      Player(const Character& character);
56
57      /**
58       * @brief This is the destructor.
59       */
60      ~Player() override;
```

```
62      /**
63       * @brief Getter for the score.
64       * @return an integer representing the score.
65       */
66      int GetScore() const;
67
68      /**
69       * @brief Setter for the score.
70       * @param score - the score.
71       */
72      void SetScore(int score);
73
74      /**
75       * @brief Getter for the monsters slain.
76       * @return - an integer representing the number of monsters slain.
77       */
78      int GetMonstersSlain() const;
79
80      /**
81       * @brief Setter for the monsters slain.
82       * @param monstersSlain - the number of monsters slain.
83       */
84      void SetMonstersSlain(int monstersSlain);
85
86      /**
87       * @brief This method displays the player's stats.
88       */
89      void ShowStats() const override;
90
91      void AcceptVisitor(PlayerVisitor* visitor);
92  };
```

```
105 class PlayerVisitor
106 {
107 public:
108     void VisitPlayer(Player* player);
109 };
```

Character.h Declarations

```cpp
class Character
{
protected:
    std::string _name;
    float _health;
    std::vector<std::string> _attackMoves;
    bool _friendliness;
    std::vector<Item> _backpack;
    bool _shieldsUp;
    float _attackPower;

public:
    /**
    * @brief This is the default constructor.
    */
    Character();

    /**
    * @brief This is the overloaded constructor.
    * @param name - the name of the character.
    */
    Character(std::string& name);

    /**
    * @brief This is the copy constructor.
    * @param character - the character to be copied.
    */
    Character(const Character& character);

    /**
    * @brief This is the destructor.
    */
    virtual ~Character();
```

```cpp
    /**
    * @brief This is the dereference operator.
    * @return a reference to the character.
    */
    const Character& operator*() const;

    /**
    * @brief Getter for the _name property.
    * @return a string representing the name of the character.
    */
    std::string& GetName();

    /**
    * @brief Setter for the _name property.
    * @param name - the name of the character.
    */
    void SetName(std::string& name);

    /**
    * @brief Getter for the _health property.
    * @return a float representing the health of the character.
    */
    float GetHealth() const;

    /**
    * @brief Setter for the _health property.
    * @param health - the health of the character.
    */
    void SetHealth(float health);

    /**
    * @brief Getter for the _attackMoves property.
    * @return a vector of strings representing the attack moves of the character.
    */
    std::vector<std::string>& GetAttackMoves();
```

```cpp
    /**
     * @brief Setter for the _attackMoves property.
     * @param attackMoves - the attack moves of the character.
     */
    void SetAttackMoves(std::vector<std::string>& attackMoves);

    /**
     * @brief Getter for the _friendliness property.
     * @return a bool representing the friendliness of the character.
     */
    bool GetFriendliness() const;

    /**
     * @brief Getter for the _backpack property.
     * @return a vector of items representing the backpack of the character.
     */
    std::vector<Item>& GetBackpack();

    /**
     * @brief Setter for the _backpack property.
     * @param backpack - the backpack of the character.
     */
    void SetBackpack(std::vector<Item>& backpack);

    /**
     * @brief Getter for the _shieldsUp property.
     * @return a bool representing the shields up status of the character.
     */
    bool GetShieldsUp() const;

    /**
     * @brief Getter for the _attackPower property.
     * @return a float representing the attack power of the character.
     */
    float GetAttackPower() const;
```

```cpp
    /**
     * @brief Setter for the _attackPower property.
     * @param attackPower - the attack power of the character.
     */
    void SetAttackPower(float attackPower);

    /**
     * @brief This method toggles the shields up status of the character.
     */
    bool ToggleShieldsUp();

    /**
     * @brief This method toggles the friendliness of the character.
     */
    bool ToggleFriendliness();

    /**
     * @brief This method adds an item to the character's backpack.
     * @param item - the item to be added.
     */
    void PickItem(Item* item);

    /**
     * @brief This method attacks another character.
     * @param character - the character to be attacked.
     */
    void Attack(Character* character) const;

    /**
     * @brief This method allows the character to enter a room.
     * @param room - the room to be entered.
     */
    void EnterRoom(Room* room);
```

```
    /**
     * @brief This method displays the character's status.
     */
    virtual void ShowStats() const;

};
```

------ Room Declaration -----

```
class Room
{
public:
    typedef Room* RoomPtr;
    static RoomPtr Sentinel;

private:
    std::string _name;
    std::vector<Clue> _clues;
    std::vector<Character> _monsters;
    std::vector<Item> _treasureTrove;
    bool _playerInRoom;
    bool _accessGranted;
    int _roomNumber;
    Room* _prevRoom;
    Room* _nextRoomOne;
    Room* _nextRoomTwo;

public:
    /**
     * @brief This is the default constructor.
     */
    Room();

    /**
     * @brief This is the overloaded constructor.
     * @param name - the name of the room.
     * @param roomNumber - the room number.
     */
    Room(const std::string& name, int roomNumber);

```

```
243     /**
244      * @brief This is the copy constructor.
245      * @param room - the room to be copied
246      */
247     Room(const Room& room);
248
249     /**
250      * @brief This is the destructor.
251      */
252     ~Room();
253
254     void Setup();
255
256     /**
257      * @brief This is the dereference operator.
258      * @return a reference to the room.
259      */
260     const Room& operator*() const;
261
262     /**
263      * @brief Getter for the _name property.
264      * @return a string representing the name of the room.
265      */
266     std::string GetName() const;
267
268     /**
269      * @brief Setter for the _name property.
270      * @param name - the name of the room.
271      */
272     void SetName(const std::string& name);
```

```
274     /**
275      * @brief Getter for the _clues property.
276      * @return a vector of strings representing the clues in the room.
277      */
278     std::vector<Clue>& GetClues();
279
280     /**
281      * @brief Setter for the _clues property.
282      * @param clues - the vector of clues in the room.
283      */
284     void SetClues(const std::vector<Clue>& clues);
285
286     /**
287      * @brief Getter for the _monsters property.
288      * @return a vector of Enemy objects representing the monsters in the room.
289      */
290     std::vector<Character>& GetMonsters();
291
292     /**
293      * @brief Setter for the _monsters property.
294      * @param monsters - the vector of monsters in the room.
295      */
296     void SetMonsters(const std::vector<Character>& monsters);
297
298     /**
299      * @brief Getter for the _treasureTrove property.
300      * @return a vector of Item objects representing the items in the room.
301      */
302     std::vector<Item> GetTreasureTrove() const;
303
304     /**
305      * @brief Setter for the _treasureTrove property.
306      * @param treasureTrove - the vector of items in the room.
307      */
308     void SetTreasureTrove(const std::vector<Item>& treasureTrove);
```

```
310     /**
311      * @brief Getter for the _playerInRoom property.
312      * @return a bool representing the player in room status.
313      */
314     bool IsPlayerInRoom() const;
315
316     /**
317      * @brief Getter for the _accessGranted property.
318      * @return a bool representing the access granted status.
319      */
320     bool IsAccessGranted() const;
321
322     /**
323      * @brief Getter for the _roomNumber property.
324      * @return an integer representing the room number.
325      */
326     int GetRoomNumber() const;
327
328     /**
329      * @brief Setter for the _roomNumber property.
330      * @param roomNumber - the room number.
331      */
332     void SetRoomNumber(int roomNumber);
333
334     /**
335      * @brief Getter for the _prevRoom property.
336      * @return a pointer to the previous room.
337      */
338     Room* GetPrevRoom() const;
339
340     /**
341      * @brief Setter for the _prevRoom property.
342      * @param prevRoom - the previous room.
343      */
344     void SetPrevRoom(Room* prevRoom);
```

```
346     /**
347      * @brief Getter for the _nextRoomOne property.
348      * @return a pointer to the first next room.
349      */
350     Room* GetNextRoomOne() const;
351
352     /**
353      * @brief Setter for the _nextRoomOne property.
354      * @param nextRoomOne - the first next room.
355      */
356     void SetNextRoomOne(Room* nextRoomOne);
357
358     /**
359      * @brief Getter for the _nextRoomTwo property.
360      * @return a pointer to the second next room.
361      */
362     Room* GetNextRoomTwo() const;
363
364     /**
365      * @brief Setter for the _nextRoomTwo property.
366      * @param nextRoomTwo - the second next room.
367      */
368     void SetNextRoomTwo(Room* nextRoomTwo);
369
370     /**
371      * @brief This method toggles the player in room status.
372      * @return a bool representing the player in room status.
373      */
374     bool TogglePlayerPresence();
375
376     /**
377      * @brief This method toggles the access granted status.
378      * @return a bool representing the access granted status.
379      */
380     bool ToggleAccess();
```

```
382     /**
383      * @brief This method displays the welcome message.
384      */
385     void WelcomeMessage();
386 };
```

Dungeon.h

```cpp
class Dungeon
{
public:
    Player* _player;
    Room* _entryRoom;
    Position _spawnPoint;

    /**
     * @brief Ths is the default constructor.
     */
    Dungeon();

    /**
     * @brief This is the overloaded constructor.
     * @param player - the player.
     * @param entryRoom - the entry room.
     * @param spawnPoint - the spawn point.
     */
    Dungeon(Player* player, Room* entryRoom, Position spawnPoint);

    /**
     * @brief This is the copy constructor.
     * @param dungeon - the dungeon to be copied.
     */
    Dungeon(const Dungeon& dungeon);

    /**
     * @brief This is the destructor.
     */
    ~Dungeon();

    /**
     * @brief This method displays the details of the dungeon.
     */
    void Display() const;

    /**
     * @brief This method displays the details of the dungeon.
     * @param room - the room to be traversed.
     */
    void Traverse(Room* room);

    /**
     * @brief This method recursively traverses the dungeon using DFS algorithm.
     * @param room - the room to start with
     */
    void DFSTraversal(Room* room);

    /**
     * @brief This function implements the Task 3 in pSet4
     */
    static void Task3();

    /**
     * @brief This function implements the Task 4 in pSet4
     */
    static void Task4();

    /**
     * @brief This function implements the Task 5 in pSet4
     * @param someString - the string to be converted to lower case.
     * @return
     */
    static std::string ToLower(std::string& someString);
```

```
95
96      /**
97       * @brief This method begins the game.
98       */
99      static void Begin();
100 };
```

Item.cc Implementation

```
1  #include "Item.h"
2  #include <iostream>
3
4  Item::Item()
5  {
6      _name = "Default Item Name";
7      _level = 0;
8      _accessPoints = 45;
9  }
10
11 Item::Item(std::string& name, int level, int accessPoints)
12 {
13     _name = name;
14     _level = level;
15     _accessPoints = accessPoints;
16 }
17
18 Item::Item(const Item& item)
19 {
20     _name = item._name;
21     _level = item._level;
22     _accessPoints = item._accessPoints;
23 }
24
25 Item::~Item()
26 {
27     _name = "";
28     _level = 0;
29     _accessPoints = 0;
30 }
31
32 std::string& Item::GetName()
33 {
34     return _name;
35 }
```

```cpp
void Item::SetName(std::string& name)
{
    _name = name;
}

int Item::GetLevel() const
{
    return _level;
}

void Item::SetLevel(int level)
{
    _level = level;
}

int Item::GetAccessPoints() const
{
    return _accessPoints;
}

void Item::SetAccessPoints(int accessPoints)
{
    _accessPoints = accessPoints;
}

void Item::Display() const
{
    std::cout << "\t=======================================" << std::endl;
    std::cout << "\t> " << _name << " Details" << std::endl;
    std::cout << "\t=======================================" << std::endl;
    std::cout << "\tItem Name: " << _name << std::endl;
    std::cout << "\tItem Level: " << _level << std::endl;
    std::cout << "\tItem Access Points: " << _accessPoints << std::endl;
    std::cout << "\t_______________________________________" << std::endl;
    std::cout << std::endl;
}
```

Clue.cc Implementation

```cpp
#include "Clue.h"
#include <iostream>

Clue::Clue()
{
    _clueNumber = 0;
    _clueMessage = "Default Clue Message";
    _timesShown = 0;
    _maxReads = 3;
    _roomNumber = 000;
}

Clue::Clue(int clueNumber, std::string& clueMessage, int maxReads, int roomNumber)
{
    _clueNumber = clueNumber;
    _clueMessage = clueMessage;
    _timesShown = 0;
    _maxReads = maxReads;
    _roomNumber = roomNumber;
}

Clue::Clue(const Clue& clue)
{
    _clueNumber = clue._clueNumber;
    _clueMessage = clue._clueMessage;
    _timesShown = clue._timesShown;
    _maxReads = clue._maxReads;
    _roomNumber = clue._roomNumber;
}
```

```cpp
Clue::~Clue()
{
    _clueNumber = 0;
    _clueMessage = "";
    _timesShown = 0;
    _maxReads = 0;
    _roomNumber = 0;
}

int Clue::GetClueNumber() const
{
    return _clueNumber;
}

void Clue::SetClueNumber(int clueNumber)
{
    _clueNumber = clueNumber;
}

std::string& Clue::GetClueMessage()
{
    return _clueMessage;
}

void Clue::SetClueMessage(std::string& clueMessage)
{
    _clueMessage = clueMessage;
}

int Clue::GetTimesShown() const
{
    return _timesShown;
}
```

```cpp
void Clue::SetTimesShown(int timesShown)
{
    _timesShown = timesShown;
}

int Clue::GetMaxReads() const
{
    return _maxReads;
}

int Clue::GetRoomNumber() const
{
    return _roomNumber;
}

void Clue::SetRoomNumber(int roomNumber)
{
    _roomNumber = roomNumber;
}

void Clue::Display() const
{
    std::cout << "=========================================" << std::endl;
    std::cout << "=  CLUE: " << _clueNumber << "  =====  Room Number: "<< _roomNumber << " ==" << std::endl;
    std::cout << "=========================================" << std::endl;
    std::cout << "=========================================" << std::endl;
    std::cout << _clueMessage << std::endl;
    std::cout << "-----------------------------------" << std::endl;
    std::cout << "Number of Times shown: " << _timesShown << std::endl;
    std::cout << "Maximum Reads Set to : " << _maxReads << std::endl;
    std::cout << "=========================================" << std::endl;
}
```

Enemy.cc

```cpp
#include "Enemy.h"
#include <iostream>

Enemy::Enemy()
{
    _name = "Default Character Name";
    _health = 100;
    _attackMoves = {};
    _friendliness = false;
    _backpack = {};
    _shieldsUp = false;
    _attackPower = 50;
    _venomReserve = 0;
}

Enemy::Enemy(std::string& name, int venomReserve)
    : Character(name)
{
    _health = 100;
    _attackMoves = {};
    _friendliness = false;
    _backpack = {};
    _shieldsUp = false;
    _attackPower = 60;
    _venomReserve = venomReserve;
}

Enemy::Enemy(const Enemy& enemy)
    : Character(enemy)
{
    _venomReserve = enemy._venomReserve;
}
```

```cpp
Enemy::Enemy(const Character& character, int venomReserve)
: Character(character)
{
    _venomReserve = venomReserve;
}

Enemy::~Enemy()
{
    _name = "";
    _health = 0;
    _attackMoves.clear();
    _friendliness = false;
    _backpack.clear();
    _shieldsUp = false;
    _attackPower = 0;
    _venomReserve = 0;
}

int Enemy::GetVenomReserve() const
{
        return _venomReserve;
}

void Enemy::SetVenomReserve(int venomReserve)
{
        _venomReserve = venomReserve;
}
```

```cpp
void Enemy::ShowStats() const
{
    std::cout << "***********************************************" << std::endl;
    std::cout << "\tEnemy Name: " << _name << "\tHealth: " << _health << std::endl;
    std::cout << "***********************************************" << std::endl;
    std::cout << "Attack Moves: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& move:const string&: _attackMoves)
    {
        std::cout << "}⇒" << move << std::endl;
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Friendliness: " << ((_friendliness) ? "Friendly" : "Not Friendly") << std::endl;
    std::cout << "Backpack: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& item: _backpack)
    {
        item.Display();
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Shields Up: " << ((_shieldsUp) ? "Shielded" : "Not Shielded") << std::endl;
    std::cout << "Attack Power: " << _attackPower << std::endl;
    std::cout << "Venom Reserve: " << _venomReserve << std::endl;
    std::cout << std::endl;
}
```

Character.cc & Room Implementation

```cpp
#include "Character.h"
#include "Clue.h"
#include <iostream>

Character::Character()
{
    std::vector<std::string> attackMoves = {
        "Double Jab Punch",
        "[COMBO] Low Right Kick, Right Jab Punch",
        "Tackle",
        "Round Kick, Floor Opponent"
    };

    _name = "Default Character Name";
    _health = 100;
    _attackMoves = {
        attackMoves[0],
        attackMoves[1],
        attackMoves[2],
        attackMoves[3]
    };
    _friendliness = false;
    _backpack = {
    };
    _shieldsUp = false;
    _attackPower = 120;
}

Character::Character(std::string& name)
{
    _name = name;
    _health = 100;
    _attackMoves = {};
    _friendliness = false;
    _backpack = {};
```

```
35      _backpack = {};
36      _shieldsUp = false;
37      _attackPower = 120;
38  }
39
40  Character::Character(const Character& character)
41  {
42      _name = character._name;
43      _health = character._health;
44      _attackMoves = character._attackMoves;
45      _friendliness = character._friendliness;
46      _backpack = character._backpack;
47      _shieldsUp = character._shieldsUp;
48      _attackPower = character._attackPower;
49  }
50
51  Character::~Character()
52  {
53      _name = "";
54      _health = 0;
55      _attackMoves.clear();
56      _friendliness = false;
57      _backpack.clear();
58      _shieldsUp = false;
59      _attackPower = 0;
60  }
61
62  const Character& Character::operator*() const
63  {
64      return *this;
65  }
```

```
67  std::string& Character::GetName()
68  {
69      return _name;
70  }
71
72  void Character::SetName(std::string& name)
73  {
74      _name = name;
75  }
76
77  float Character::GetHealth() const
78  {
79      return _health;
80  }
81
82  void Character::SetHealth(float health)
83  {
84      _health = health;
85  }
86
87  std::vector<std::string>& Character::GetAttackMoves()
88  {
89      return _attackMoves;
90  }
91
92  void Character::SetAttackMoves(std::vector<std::string>& attackMoves)
93  {
94      _attackMoves = attackMoves;
95  }
96
97  bool Character::GetFriendliness() const
98  {
99          return _friendliness;
100 }
```

```cpp
std::vector<Item>& Character::GetBackpack()
{
    return _backpack;
}

void Character::SetBackpack(std::vector<Item>& backpack)
{
    _backpack = backpack;
}

bool Character::GetShieldsUp() const
{
    return _shieldsUp;
}

float Character::GetAttackPower() const
{
    return _attackPower;
}

void Character::SetAttackPower(float attackPower)
{
    _attackPower = attackPower;
}

bool Character::ToggleShieldsUp()
{
    _shieldsUp = !_shieldsUp;
    return _shieldsUp;
}
```

```cpp
bool Character::ToggleFriendliness()
{
    _friendliness = !_friendliness;
    return _friendliness;
}

void Character::PickItem(Item* item)
{
    _backpack.push_back(*item);
}

void Character::Attack(Character* character) const
{
    if (_shieldsUp)
    {
        std::cout << _name << " is defending." << std::endl;
    }
    else
    {
        std::cout << _name << " is attacking." << std::endl;
        character->SetHealth(character->GetHealth() - _attackPower);
    }
}

void Character::EnterRoom(Room* room)
{
    std::cout<< _name << " has entered " << room->GetName() << "." << std::endl;
    room->TogglePlayerPresence();
}
```

```cpp
void Character::ShowStats() const
{
    std::cout << "-----------------------------------------------" << std::endl;
    std::cout << "Default Name: " << _name << "Health: " << _health << std::endl;
    std::cout << "-----------------------------------------------" << std::endl;
    std::cout << "Attack Moves: " << std::endl;

    for (auto& move :const string& : _attackMoves)
    {
        std::cout << move << std::endl;
    }

    std::cout << "Friendliness: " << ((_friendliness) ? "Friendly" : "Not Friendly") << std::endl;
    std::cout << "Backpack: " << std::endl;

    for (auto& item : _backpack)
    {
        item.Display();
    }

    std::cout << "Shields Up: " << ((_shieldsUp) ? "Shielded" : "Not Shielded") << std::endl;
    std::cout << "Attack Power: " << _attackPower << std::endl;
}
```

```cpp
//////////////////////////////////////
////////    ROOM IMPLEMENTATIONS   //////
//////////////////////////////////////

Room::Room()
{
    _name = "Default Room Name";
    _roomNumber = 10;

    Setup();

    _playerInRoom = false;
    _accessGranted = false;
    _prevRoom = Sentinel;
    _nextRoomOne = Sentinel;
    _nextRoomTwo = Sentinel;
}

Room::Room(const std::string& name, int roomNumber)
{


    _name = name;
    _roomNumber = roomNumber;

    Setup();

    _playerInRoom = false;
    _accessGranted = false;
    _prevRoom = Sentinel;
    _nextRoomOne = Sentinel;
    _nextRoomTwo = Sentinel;

}
```

```cpp
Room::Room(const Room& room)
{
    _name = room._name;
    _clues = room._clues;
    _monsters = room._monsters;
    _treasureTrove = room._treasureTrove;
    _playerInRoom = room._playerInRoom;
    _accessGranted = room._accessGranted;
    _roomNumber = room._roomNumber;
    _prevRoom = room._prevRoom;
    _nextRoomOne = room._nextRoomOne;
    _nextRoomTwo = room._nextRoomTwo;
}

Room::~Room()
{
    _name = "";
    _clues.clear();
    _monsters.clear();
    _treasureTrove.clear();
    _playerInRoom = false;
    _accessGranted = false;
    _roomNumber = 0;
    _prevRoom = Sentinel;
    _nextRoomOne = Sentinel;
    _nextRoomTwo = Sentinel;
}

void Room::Setup()
{
    std::vector<std::string> cluesMsgs = {
    "This is clue message number 101",
    "This is clue message number 102",
    "This is clue message number 103",
    "This is clue message number 104",
```

```
    "This is clue message number 104",
    };
    std::vector<std::string> monstersNames = {
        "Fire Breathing Horse",
        "Stinging Robotic Wasp",
        "Multi-Head Black Mamba",
        "Hop Hop Toad"
    };
    std::vector<std::string> itemNames = {
        "Troy's Sword",
        "Belinda's Precious Necklace",
        "South African Diamond Jewel Pack",
        "Shariffah's Golden Knife"
    };

    _clues = {
        Clue(clueNumber: 101, [&]cluesMsgs[0], maxReads: 3, _roomNumber),
        Clue(clueNumber: 101, [&]cluesMsgs[1], maxReads: 3, _roomNumber),
        Clue(clueNumber: 101, [&]cluesMsgs[2], maxReads: 3, _roomNumber),
        Clue(clueNumber: 101, [&]cluesMsgs[3], maxReads: 3, _roomNumber)
    };
    _monsters = {
        Character([&]monstersNames[0]),
        Character([&]monstersNames[1]),
        Character([&]monstersNames[2]),
        Character([&]monstersNames[3])
    };
    _treasureTrove = {
        Item([&]itemNames[0], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100),
        Item([&]itemNames[1], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100),
        Item([&]itemNames[2], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100),
        Item([&]itemNames[3], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100)

    };
}
```

```cpp
std::string Room::GetName() const
{
    return _name;
}

void Room::SetName(const std::string& name)
{
    _name = name;
}

std::vector<Clue>& Room::GetClues()
{
    return _clues;
}

void Room::SetClues(const std::vector<Clue>& clues)
{
    _clues = clues;
}

std::vector<Character>& Room::GetMonsters()
{
    return _monsters;
}

void Room::SetMonsters(const std::vector<Character>& monsters)
{
    _monsters = monsters;
}

std::vector<Item> Room::GetTreasureTrove() const
{
    return _treasureTrove;
}

```

```cpp
void Room::SetTreasureTrove(const std::vector<Item>& treasureTrove)
{
    _treasureTrove = treasureTrove;
}

bool Room::IsPlayerInRoom() const
{
    return _playerInRoom;
}

bool Room::IsAccessGranted() const
{
    return _accessGranted;
}

int Room::GetRoomNumber() const
{
    return _roomNumber;
}

void Room::SetRoomNumber(int roomNumber)
{
    _roomNumber = roomNumber;
}

const Room& Room::operator*() const
{
    return *this;
}

Room* Room::GetPrevRoom() const
{
    return _prevRoom;
}
```

```cpp
void Room::SetPrevRoom(Room* prevRoom)
{
    _prevRoom = prevRoom;
}

Room* Room::GetNextRoomOne() const
{
    return _nextRoomOne;
}

void Room::SetNextRoomOne(Room* nextRoomOne)
{
    _nextRoomOne = nextRoomOne;
}

Room* Room::GetNextRoomTwo() const
{
    return _nextRoomTwo;
}

void Room::SetNextRoomTwo(Room* nextRoomTwo)
{
    _nextRoomTwo = nextRoomTwo;
}

bool Room::TogglePlayerPresence()
{
    _playerInRoom = !_playerInRoom;
    return _playerInRoom;
}
```

```cpp
bool Room::ToggleAccess()
{
    _accessGranted = !_accessGranted;
    return _accessGranted;
}

void Room::WelcomeMessage()
{
    std::cout << "=================================================" << std::endl;
    std::cout << "= Welcome to "<< _name << "    Room Number: " << _roomNumber << "  =" << std::endl;
    std::cout << "=================================================" << std::endl;
    std::cout << "In this room there both Monsters and Treasures" << std::endl;
    std::cout << "There a total of " << _treasureTrove.size() << " treasure Items as listed below:" << std::endl;
    std::cout << "----------   Treasures   ----------" << std::endl;
    for (auto& item : _treasureTrove)
    {
        item.Display();
    }
    std::cout << "Additionally you'll face some Monstrous enemies:" << std::endl;
    std::cout << "----------   Enemies   ----------" << std::endl;
    for (auto& monster:Character& : _monsters)
    {
        std::cout << monster.GetName() << " \t|\tAttack Power: " << monster.GetAttackPower() << std::endl;
    }
    std::cout << "Brace Yourself!!!" << std::endl;
    std::cout << "=========   END OF WELCOME MESSAGE   ========" << std::endl;
    std::cout << std::endl;
}


Room::RoomPtr Room::Sentinel;
```

Player.cc Implementation

```cpp
#include "Player.h"
#include <iostream>

Player::Player()
{
    std::vector<std::string> attackMoves = {
        "Double Jab Punch",
        "[COMBO] Low Right Kick, Right Jab Punch",
        "Tackle",
        "Round Kick, Floor Opponent"
    };

    _name = "Default Player Name";
    _health = 100;
    _attackMoves = {
        attackMoves[0],
        attackMoves[1],
        attackMoves[2],
        attackMoves[3]
    };
    _friendliness = false;
    _backpack = {};
    _shieldsUp = false;
    _attackPower = 50;

    _score = 0;
    _monstersSlain = 0;
}

Player::Player(std::string& name)
    : Character([&]name)
{
    _score = 0;
    _monstersSlain = 0;
}
```

```cpp
Player::Player(const Player& player)
    : Character(player)
{
    _score = player._score;
    _monstersSlain = player._monstersSlain;
}

Player::Player(const Character& character)
    : Character(character)
{
    _score = 0;
    _monstersSlain = 0;
}

Player::~Player()
{
    _name = "";
    _health = 0;
    _attackMoves.clear();
    _friendliness = false;
    _backpack.clear();
    _shieldsUp = false;
    _attackPower = 0;

    _score = 0;
    _monstersSlain = 0;
}

int Player::GetScore() const
{
    return _score;
}
```

```cpp
void Player::SetScore(int score)
{
    _score = score;
}

int Player::GetMonstersSlain() const
{
    return _monstersSlain;
}

void Player::SetMonstersSlain(int monstersSlain)
{
    _monstersSlain = monstersSlain;
}
```

```cpp
void Player::ShowStats() const
{
    std::cout << "################################################" << std::endl;
    std::cout << "\tPlayer Name: " << _name << "\tHealth: " << _health << std::endl;
    std::cout << "################################################" << std::endl;
    std::cout << "Attack Moves: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& move:const string& : _attackMoves)
    {
        std::cout << "}⇒" << move << std::endl;
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Friendliness: " << ((_friendliness) ? "Friendly" : "Not Friendly") << std::endl;
    std::cout << "Backpack: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& item : _backpack)
    {
        item.Display();
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Shields Up: " << ((_shieldsUp) ? "Shielded" : "Not Shielded") << std::endl;
    std::cout << "Player Attack Power: " << _attackPower << std::endl;
    std::cout << "Player Score: " << _score << std::endl;
    std::cout << "Player Monsters Slain: " << _monstersSlain << std::endl;
}

void Player::AcceptVisitor(PlayerVisitor* visitor)
{
    visitor->VisitPlayer(this);
}
```

```cpp
void PlayerVisitor::VisitPlayer(Player* player)
{
    std::cout << "Player Visitor Upgraded Health to: " << player->GetHealth() + 30 << std::endl;
    player->SetHealth(player->GetHealth() + 30);
    player->SetScore(player->GetScore() + 40);
    std::cout << "################################################" << std::endl;
    std::cout << "\tPlayer Name: " << player->GetName() << "\tHealth: " << player->GetHealth() << std::endl;
    std::cout << "################################################" << std::endl;
    std::cout << "Attack Moves: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& move:string& : player->GetAttackMoves())
    {
        std::cout << "}⇒" << move << std::endl;
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Friendliness: " << ((player->GetFriendliness()) ? "Friendly" : "Not Friendly") << std::endl
    std::cout << "Backpack: " << std::endl;
    std::cout << "------------------------------" << std::endl;

    for (auto& item : player->GetBackpack())
    {
        item.Display();
    }
    std::cout << "------------------------------" << std::endl;
    std::cout << "Shields Up: " << ((player->GetShieldsUp()) ? "Shielded" : "Not Shielded") << std::endl;
    std::cout << "Player Attack Power: " << player->GetAttackPower() << std::endl;
    std::cout << "Player Score Updated (by the visitor): " << player->GetScore() << std::endl;
    std::cout << "Player Monsters Slain: " << player->GetMonstersSlain() << std::endl;
}
```

Dungeon.cc Implementations

```cpp
#include "Dungeon.h"
#include <algorithm>
#include <iostream>

Dungeon::Dungeon()
{
    _player = new Player();

    std::vector<std::string> roomNames = {
        "Entry Room", "Room 1", "Room 2", "Room 3",
        "Room 4", "Room 5", "Room 6", "Room 7"
    };
    // Level 0 Rooms
    _entryRoom = new Room(roomNames[0], roomNumber: 10);
    // Level 1 Rooms
    Room* room1 = new Room(roomNames[1], roomNumber: 20);
    Room* room2 = new Room(roomNames[2], roomNumber: 22);
    // Level 2 Rooms
    Room* room3 = new Room(roomNames[3], roomNumber: 30);
    Room* room4 = new Room(roomNames[4], roomNumber: 31);
    Room* room5 = new Room(roomNames[5], roomNumber: 32);
    Room* room6 = new Room(roomNames[6], roomNumber: 33);
    _entryRoom->SetNextRoomOne(room1);
    _entryRoom->SetNextRoomTwo(room2);
    // room1
    room1->SetPrevRoom(_entryRoom);
    room1->SetNextRoomOne(room3);
    room1->SetNextRoomTwo(room4);
    // room2
    room2->SetPrevRoom(_entryRoom);
    room2->SetNextRoomOne(room5);
    room2->SetNextRoomTwo(room6);
    // room3
    room3->SetPrevRoom(room1);
```

```cpp
    room3->SetPrevRoom(room1);
    // room4
    room4->SetPrevRoom(room1);
    // room5
    room5->SetPrevRoom(room2);
    // room6
    room6->SetPrevRoom(room2);

    _spawnPoint = { .x: 0, .y: 0 };
}

Dungeon::Dungeon(Player* player, Room* entryRoom, Position spawnPoint)
{
    _player = player;

    std::vector<std::string> roomNames = {
        "Entry Room", "Room 1", "Room 2", "Room 3",
        "Room 4", "Room 5", "Room 6", "Room 7"
    };
    // Level 0 Rooms
    _entryRoom = entryRoom;
    // Level 1 Rooms
    Room* room1 = new Room(roomNames[1], roomNumber: 20);
    Room* room2 = new Room(roomNames[2], roomNumber: 22);
    // Level 2 Rooms
    Room* room3 = new Room(roomNames[3], roomNumber: 30);
    Room* room4 = new Room(roomNames[4], roomNumber: 31);
    Room* room5 = new Room(roomNames[5], roomNumber: 32);
    Room* room6 = new Room(roomNames[6], roomNumber: 33);
    _entryRoom->SetNextRoomOne(room1);
    _entryRoom->SetNextRoomTwo(room2);
    // room1
    room1->SetPrevRoom(_entryRoom);
    room1->SetNextRoomOne(room3);
    room1->SetNextRoomTwo(room4);
```

```
67      room1->SetNextRoomOne(room3);
68      room1->SetNextRoomTwo(room4);
69      // room2
70      room2->SetNextRoomOne(room5);
71      room2->SetNextRoomTwo(room6);
72      //
73      room3->SetPrevRoom(room1);
74      // room4
75      room4->SetPrevRoom(room1);
76      // room5
77      room5->SetPrevRoom(room2);
78      // room6
79      room6->SetPrevRoom(room2);
80
81
82      _spawnPoint = spawnPoint;
83  }
84
85  Dungeon::Dungeon(const Dungeon& dungeon)
86  {
87      _player = dungeon._player;
88      _entryRoom = dungeon._entryRoom;
89      _spawnPoint = dungeon._spawnPoint;
90  }
91
92  Dungeon::~Dungeon()
93  {
94      delete _player;
95      delete _entryRoom;
96      _spawnPoint = { .x: 0, .y: 0 };
97  }
98
```

(local variable) Room* room2

```
99  void Dungeon::Display() const
100 {
101     std::cout << "Dungeon: " << std::endl;
102     std::cout << "Player: " << _player->GetName() << std::endl;
103     std::cout << "Entry Room: " << _entryRoom->GetName() << std::endl;
104     std::cout << "Spawn Point: (" << _spawnPoint.x << ", " << _spawnPoint.y << ")" << std::endl;
105 }
106
107 void Dungeon::DFSTraversal(Room* room)
108 {
109     if (room == Room::Sentinel)
110     {
111         return;
112     }
113
114
115     std::cout << room->GetName() << std::endl;
116     DFSTraversal(room->GetNextRoomOne());
117     DFSTraversal(room->GetNextRoomTwo());
118 }
119
120 void Dungeon::Traverse(Room* room)
121 {
122     while(true)
123     {
124         // room->WelcomeMessage();
125         std::cout << "You are in " << room->GetName() << std::endl;
126         std::cout << "You have the following options: " << std::endl;
127         std::cout << "1. Go to " << (
128             (room->GetNextRoomOne() != Room::Sentinel)
129             ? room->GetNextRoomOne()->GetName()
130             : "None (This room is an end point. You'll need to go back)") << std::endl;
131         std::cout << "2. Go to " << (
132             (room->GetNextRoomTwo() != Room::Sentinel)
```

```
            (room->GetNextRoomTwo() != Room::Sentinel)
            ? room->GetNextRoomTwo()->GetName()
            : "None (This room is an end point. You'll need to go back)") << std::endl;
        std::cout << "3. Go back to " << (
            (room->GetPrevRoom()!= Room::Sentinel)
            ? room->GetPrevRoom()->GetName()
            : "None (This is the entry Room)") << std::endl;
        std::cout << "4. Quit" << std::endl;
        std::cout << "Enter a command: " << std::endl;
        std::string response;
        getline(std::cin, response);

        if (response == "1")
        {
            if (room->GetNextRoomOne() == Room::Sentinel)
            {
                std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;

            }
            else
            {
                room = room->GetNextRoomOne();
            }
        }
        else if (response == "2")
        {
            if (room->GetNextRoomTwo() == Room::Sentinel)
            {
                std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;

            }
            else
            {
                room = room->GetNextRoomTwo();
            }
```

```
                room = room->GetNextRoomTwo();
            }
        }
        else if (response == "3")
        {
            room = room->GetPrevRoom();
        }
        else if (response == "4")
        {
            std::cout << "Quitting game ... " << std::endl;
            break;
        }
        else
        {
            std::cout << "Invalid command" << std::endl;
        }
    }
}
```

```
void Dungeon::Task3()
{
    auto* dungeon = new Dungeon();

    std::cout << "═══════════════════════════════════════════════" << std::endl;
    std::cout << "═══════════    TASK 3 [START]   ═══════════" << std::endl;
    std::cout << "═══════════════════════════════════════════════" << std::endl;
    std::cout << "You have the following rooms to traverse and explore" << std::endl;

    dungeon->DFSTraversal(dungeon->_entryRoom);

    std::cout << "We Start in the entry Room: " << std::endl;
    std::cout << std::endl;
    dungeon->_entryRoom->WelcomeMessage();
    dungeon->Traverse(dungeon->_entryRoom);
    std::cout << "═══════════    TASK 3 [END]   ═══════════" << std::endl;
}

void Dungeon::Task4()
{
    auto* dungeon = new Dungeon();
    PlayerVisitor* playerVisitor = new PlayerVisitor();
    std::cout << "═══════════════════════════════════════════════" << std::endl;
    std::cout << "═══════════    TASK 4 [START]   ═══════════" << std::endl;
    std::cout << "═══════════════════════════════════════════════" << std::endl;
    playerVisitor->VisitPlayer(dungeon->_player);
    std::cout << "═══════════    TASK 4 [END]   ═══════════" << std::endl;
}
```

```cpp
std::string Dungeon::ToLower(std::string& someString)
{
    std::string result = someString;
    std::transform(_First:result.begin(), _Last:result.end(), _Dest:result.begin(), ::tolower);
    return result;
}

void Dungeon::Begin()
{
    std::vector<std::string> commands = {
        "description", "help", "task3", "task4", "quit", "clear"
    };
    std::string response;

    std::cout << "=========================================================" << std::endl;
    std::cout << "======        WELCOME TO THE DUNGEON        =============" << std::endl;
    std::cout << "=========================================================" << std::endl;
    while (true)
    {
        std::cout << "How would you like to proceed? Please type 'help' to view commands" << std::endl;
        getline([&]std::cin, [&]response);


        if (ToLower([&]response) == commands[0])
        {
            std::cout << "=========================================================" << std::endl;
            std::cout << "======          GAME    DESCRIPTION         =============" << std::endl;
            std::cout << "=========================================================" << std::endl;
            std::cout << "This is a simple game program where the player is meant to go through a set of rooms in the dungeon.\n"
                << "The player's permission to go through a certain room is accessed through a set of conditions which determine\n"
                << "whether or not the user can access the room. The player starts the game in the entry room and if the player is\n"
                << "able to slay more than three monsters then they qualify for access both the left (nextRoomOne) and right node (nextRoomTwo)\n"
                << "otherwise they can only access the room represented by the left node(nextRoomOne). This is the challenge on level one.\n";

            std::cout << "On Level Two the player has four challenges but may choose at least one to tackle.Each condition is link \n"
                << "to permission to access specified room(see Figure 1) if the player manages to satisfy all four conditions then \n"
                << "they can easily move around the whole dungeon freely with access to any room.The game is considered solved at this point." << st
            std::cout << "---------------------------------------------------------" << std::endl;
            std::cout << "---------------------------------------------------------" << std::endl;
        }
        else if (ToLower([&]response) == commands[1])
        {
            std::cout << "=========================================================" << std::endl;
            std::cout << "======          HELP PAGE [commands]        =============" << std::endl;
            std::cout << "=========================================================" << std::endl;
            std::cout << "\tdescription: \tShows you the description of the game" << std::endl;
            std::cout << "\thelp: \tOpens up this help page." << std::endl;
            std::cout << "\ttask3 \tRuns the implementation of Task 3" << std::endl;
            std::cout << "\ttask4 \tRuns the implementation of Task 4" << std::endl;
            std::cout << "\tquit: \tQuits the game" << std::endl;
            std::cout << "---------------------------------------------------------" << std::endl;
        }
        else if (ToLower([&]response) == commands[2])
        {
            Task3();
        }
        else if (ToLower([&]response) == commands[3])
        {
            Task4();
        }
        else if (ToLower([&]response) == commands[4])
        {
            std::cout << "Quitting game ... " << std::endl;
            break;
        }
        else if (ToLower([&]response) == commands[5])
        {
            #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
                system(_Command:"cls");
            #elif __APPLE__   #if defined(WIN32) || defined(_WIN32) || defined(__WIN32__) || defined(__NT__)
                              system("clear");
            #elif __linux__   #elif __APPLE__
                              system("clear");
            #endif  #elif __linux__
        }
        else
        {
            std::cout << "Invalid command" << std::endl;
        }
    }

}
```