

Problem Set 4

Contents

Problem Set 4	1
Task 1	1
Description	1
Diagram.....	2
Task 2	3
UML Diagram	3
Task 3	3
Task 4	7
Code Reference.....	10

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

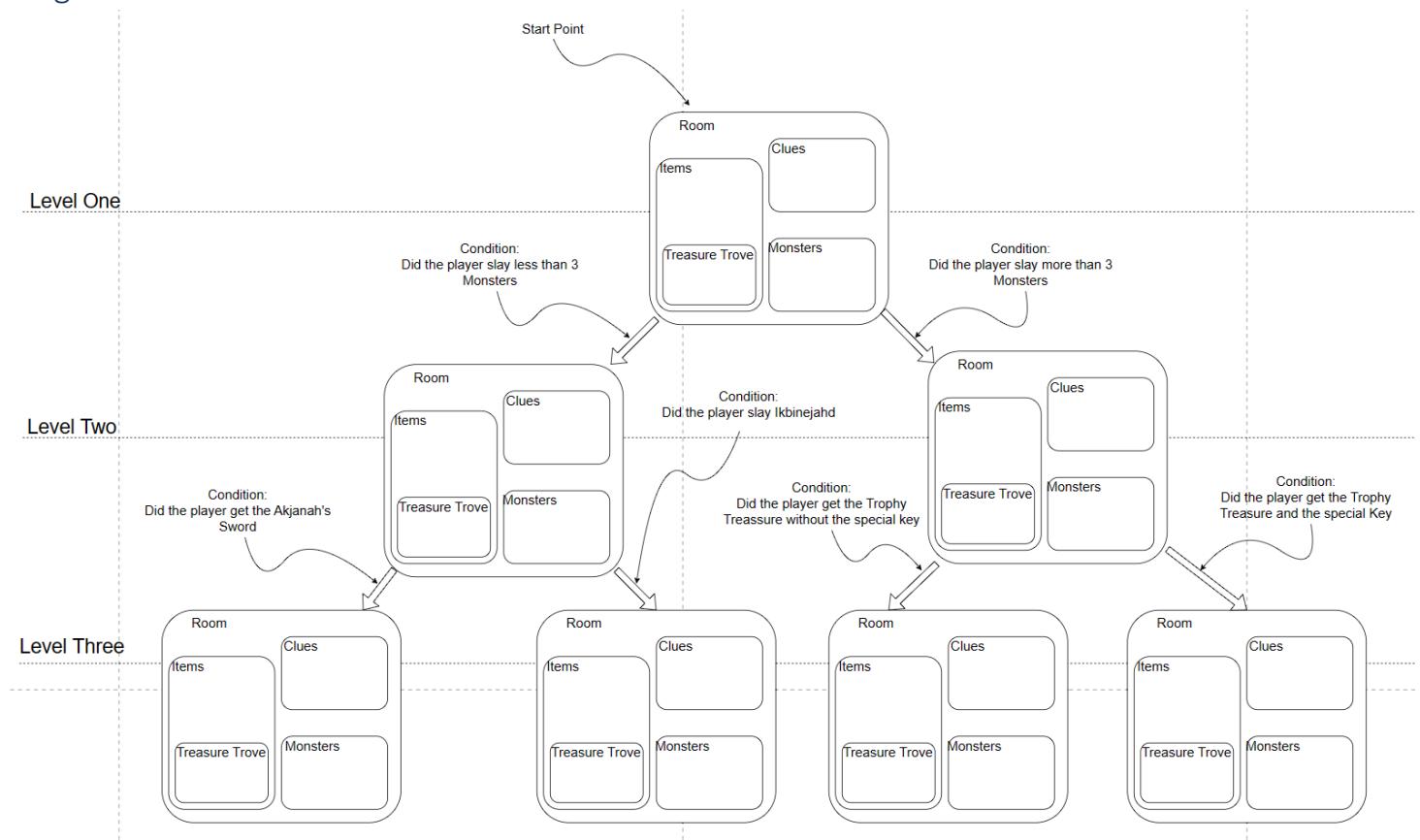
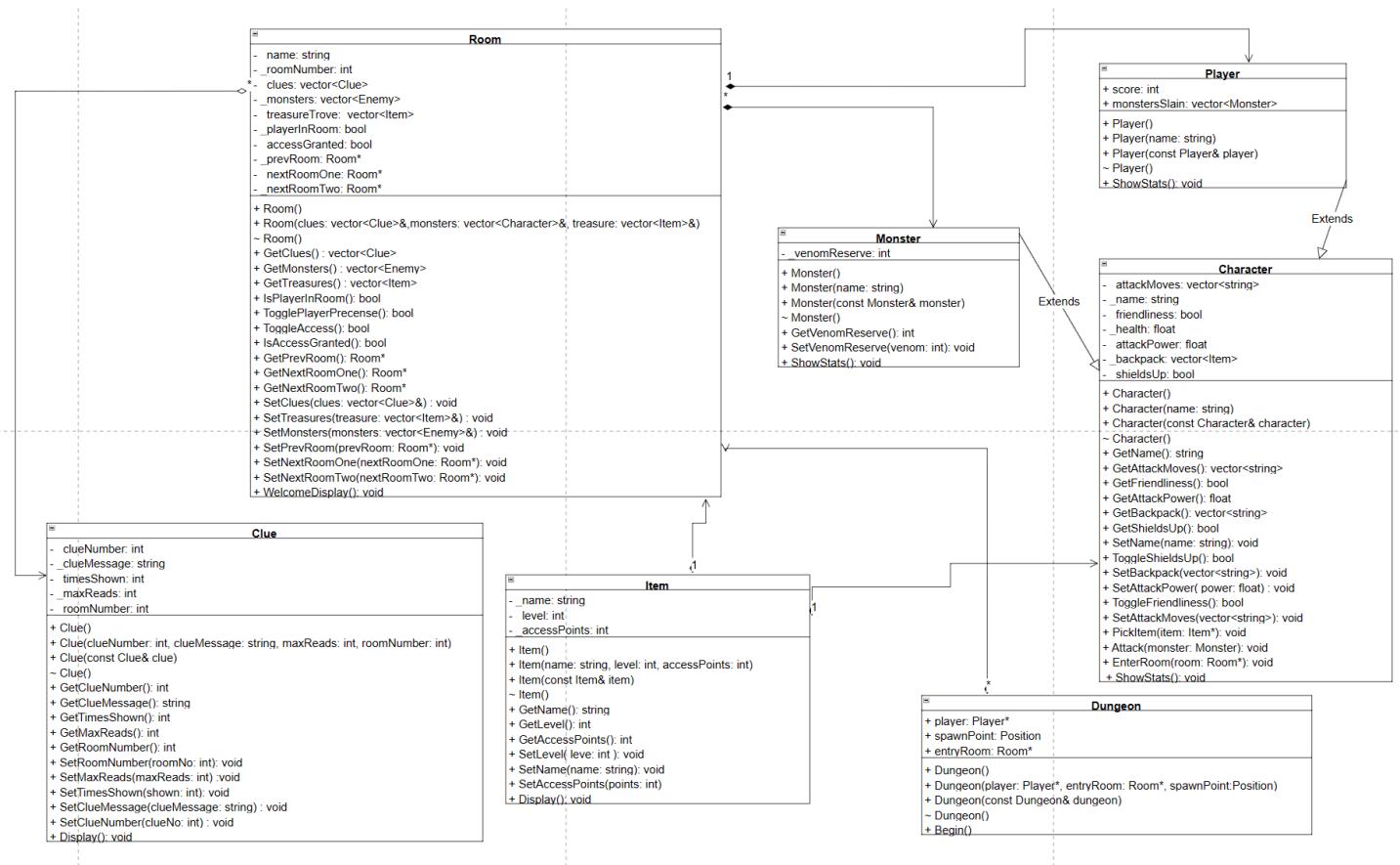


Figure 1 The Tree representation of a Dungeon

Task 2

UML Diagram



Task 3

Code Implementation

```
1 #include "Dungeon.h"
2 #include <algorithm>
3 #include <iostream>
4
5 Dungeon::Dungeon()
6 {
7     _player = new Player();
8
9     std::vector<std::string> roomNames = {
10         "Entry Room", "Room 1", "Room 2", "Room 3",
11         "Room 4", "Room 5", "Room 6", "Room 7"
12     };
13     // Level 0 Rooms
14     _entryRoom = new Room(roomNames[0], roomNumber: 10);
15     // Level 1 Rooms
16     Room* room1 = new Room(roomNames[1], roomNumber: 20);
17     Room* room2 = new Room(roomNames[2], roomNumber: 22);
18     // Level 2 Rooms
19     Room* room3 = new Room(roomNames[3], roomNumber: 30);
20     Room* room4 = new Room(roomNames[4], roomNumber: 31);
21     Room* room5 = new Room(roomNames[5], roomNumber: 32);
22     Room* room6 = new Room(roomNames[6], roomNumber: 33);
23     _entryRoom->SetNextRoomOne(room1);
24     _entryRoom->SetNextRoomTwo(room2);
25     // room1
26     room1->SetPrevRoom(_entryRoom);
27     room1->SetNextRoomOne(room3);
28     room1->SetNextRoomTwo(room4);
29     // room2
30     room2->SetPrevRoom(_entryRoom);
31     room2->SetNextRoomOne(room5);
32     room2->SetNextRoomTwo(room6);
33     // room3
34     room3->SetPrevRoom(room1);
35     // room4
36     room4->SetPrevRoom(room1);
37     // room5
38     room5->SetPrevRoom(room2);
39     // room6
40     room6->SetPrevRoom(room2);
41
42     _spawnPoint = { .x: 0, .y: 0 };
43 }
44
```

```

110 void Dungeon::DFSTraversal(Room* room)
111 {
112     if (room == Room::Sentinel)
113     {
114         return;
115     }
116
117
118     std::cout << room->GetName() << std::endl;
119     DFSTraversal(room->GetNextRoomOne());
120     DFSTraversal(room->GetNextRoomTwo());
121 }
122
123 void Dungeon::Traverse(Room* room)
124 {
125     while(true)
126     {
127         // room->WelcomeMessage();
128         std::cout << "You are in " << room->GetName() << std::endl;
129         std::cout << "You have the following options: " << std::endl;
130         std::cout << "1. Go to " << (
131             (room->GetNextRoomOne() != Room::Sentinel)
132             ? room->GetNextRoomOne()->GetName()
133             : "None (This room is an end point. You'll need to go back)") << std::endl;
134         std::cout << "2. Go to " << (
135             (room->GetNextRoomTwo() != Room::Sentinel)
136             ? room->GetNextRoomTwo()->GetName()
137             : "None (This room is an end point. You'll need to go back)") << std::endl;
138         std::cout << "3. Go back to " << (
139             (room->GetPrevRoom() != Room::Sentinel)
140             ? room->GetPrevRoom()->GetName()
141             : "None (This is the entry Room)") << std::endl;
142         std::cout << "4. Quit" << std::endl;
143         std::cout << "Enter a command: " << std::endl;
144         std::string response;
145         getline( [&] std::cin, [ &] response);
146
147         if (response == "1")
148         {
149             if (room->GetNextRoomOne() == Room::Sentinel)
150             {
151                 std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;
152             }
153             else
154             {
155                 room = room->GetNextRoomOne();
156             }
157         }
158         else if (response == "2")
159         {
160             if (room->GetNextRoomTwo() == Room::Sentinel)
161             {
162                 std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;
163             }
164             else
165             {
166                 room = room->GetNextRoomTwo();
167             }
168         }
169         else if (response == "3")
170         {
171             room = room->GetPrevRoom();
172         }
173         else if (response == "4")
174         {
175

```

```

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 << "===== TASK 3 [START] =====" << std::endl;
192     std::cout << "You have the following rooms to traverse and explore" << std::endl;
193
194     dungeon->DFSTraversal(dungeon->_entryRoom);
195
196     std::cout << "We Start in the entry Room: " << std::endl;
197     std::cout << std::endl;
198     dungeon->_entryRoom->WelcomeMessage();
199     dungeon->Traverse(dungeon->_entryRoom);
200     std::cout << "===== TASK 3 [END] =====" << std::endl;
201
202 }
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:
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     void AcceptVisitor(PlayerVisitor* visitor);
91 };
92
93 }
```

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

```
114 void Player::AcceptVisitor(PlayerVisitor* visitor)
115 {
116     visitor->VisitPlayer(this);
117 }
118
119 void PlayerVisitor::VisitPlayer(Player* player)
120 {
121     std::cout << "Player Visitor Upgraded Health to: " << player->GetHealth() + 30 << std::endl;
122     player->SetHealth(player->GetHealth() + 30);
123     player->SetScore(player->GetScore() + 40);
124     std::cout << "#####" << std::endl;
125     std::cout << "\tPlayer Name: " << player->GetName() << "\tHealth: " << player->GetHealth() << std::endl;
126     std::cout << "#####" << std::endl;
127     std::cout << "Attack Moves: " << std::endl;
128     std::cout << "-----" << std::endl;
129
130     for (auto& move : string& : player->GetAttackMoves())
131     {
132         std::cout << "}" << move << std::endl;
133     }
134     std::cout << "-----" << std::endl;
135     std::cout << "Friendliness: " << ((player->GetFriendliness()) ? "Friendly" : "Not Friendly") << std::endl;
136     std::cout << "Backpack: " << std::endl;
137     std::cout << "-----" << std::endl;
138
139     for (auto& item : player->GetBackpack())
140     {
141         item.Display();
142     }
143     std::cout << "-----" << std::endl;
144     std::cout << "Shields Up: " << ((player->GetShieldsUp()) ? "Shielded" : "Not Shielded") << std::endl;
145     std::cout << "Player Attack Power: " << player->GetAttackPower() << std::endl;
146     std::cout << "Player Score Updated (by the visitor): " << player->GetScore() << std::endl;
147     std::cout << "Player Monsters Slain: " << player->GetMonstersSlain() << std::endl;
148 }
```

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, it'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 one, 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 This 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();
55
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 - the 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;
59
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

```
39 class Character
40 {
41     protected:
42         std::string _name;
43         float _health;
44         std::vector<std::string> _attackMoves;
45         bool _friendliness;
46         std::vector<Item> _backpack;
47         bool _shieldsUp;
48         float _attackPower;
49
50     public:
51     /**
52      * @brief This is the default constructor.
53      */
54     Character();
55
56     /**
57      * @brief This is the overloaded constructor.
58      * @param name - the name of the character.
59      */
60     Character(std::string& name);
61
62     /**
63      * @brief This is the copy constructor.
64      * @param character - the character to be copied.
65      */
66     Character(const Character& character);
67
68     /**
69      * @brief This is the destructor.
70      */
71     virtual ~Character();

73     /**
74      * @brief This is the dereference operator.
75      * @return a reference to the character.
76      */
77     const Character& operator*() const;

79     /**
80      * @brief Getter for the _name property.
81      * @return a string representing the name of the character.
82      */
83     std::string& GetName();

85     /**
86      * @brief Setter for the _name property.
87      * @param name - the name of the character.
88      */
89     void SetName(std::string& name);

91     /**
92      * @brief Getter for the _health property.
93      * @return a float representing the health of the character.
94      */
95     float GetHealth() const;

97     /**
98      * @brief Setter for the _health property.
99      * @param health - the health of the character.
100     */
101    void SetHealth(float health);

103     /**
104      * @brief Getter for the _attackMoves property.
105      * @return a vector of strings representing the attack moves of the character.
106     */
107     std::vector<std::string>& GetAttackMoves();
```

```
109 /**
110  * @brief Setter for the _attackMoves property.
111  * @param attackMoves - the attack moves of the character.
112  */
113 void SetAttackMoves(std::vector<std::string>& attackMoves);
114
115 /**
116  * @brief Getter for the _friendliness property.
117  * @return a bool representing the friendliness of the character.
118  */
119 bool GetFriendliness() const;
120
121 /**
122  * @brief Getter for the _backpack property.
123  * @return a vector of items representing the backpack of the character.
124  */
125 std::vector<Item>& GetBackpack();
126
127 /**
128  * @brief Setter for the _backpack property.
129  * @param backpack - the backpack of the character.
130  */
131 void SetBackpack(std::vector<Item>& backpack);
132
133 /**
134  * @brief Getter for the _shieldsUp property.
135  * @return a bool representing the shields up status of the character.
136  */
137 bool GetShieldsUp() const;
138
139 /**
140  * @brief Getter for the _attackPower property.
141  * @return a float representing the attack power of the character.
142  */
143 float GetAttackPower() const;
```

```
145 /**
146  * @brief Setter for the _attackPower property.
147  * @param attackPower - the attack power of the character.
148  */
149 void SetAttackPower(float attackPower);
150
151 /**
152  * @brief This method toggles the shields up status of the character.
153  */
154 bool ToggleShieldsUp();
155
156 /**
157  * @brief This method toggles the friendliness of the character.
158  */
159 bool ToggleFriendliness();
160
161 /**
162  * @brief This method adds an item to the character's backpack.
163  * @param item - the item to be added.
164  */
165 void PickItem(Item* item);
166
167 /**
168  * @brief This method attacks another character.
169  * @param character - the character to be attacked.
170  */
171 void Attack(Character* character) const;
172
173 /**
174  * @brief This method allows the character to enter a room.
175  * @param room - the room to be entered.
176  */
177 void EnterRoom(Room* room);
```

```
178
179     /**
180      * @brief This method displays the character's status.
181      */
182     virtual void ShowStats() const;
183
184 };
185
```

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

```
212 class Room
213 {
214 public:
215     typedef Room* RoomPtr;
216     static RoomPtr Sentinel;
217
218 private:
219     std::string _name;
220     std::vector<Clue> _clues;
221     std::vector<Character> _monsters;
222     std::vector<Item> _treasureTrove;
223     bool _playerInRoom;
224     bool _accessGranted;
225     int _roomNumber;
226     Room* _prevRoom;
227     Room* _nextRoomOne;
228     Room* _nextRoomTwo;
229
230 public:
231     /**
232      * @brief This is the default constructor.
233      */
234     Room();
235
236     /**
237      * @brief This is the overloaded constructor.
238      * @param name - the name of the room.
239      * @param roomNumber - the room number.
240      */
241     Room(const std::string& name, int _roomNumber);
242
```

```
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 /**
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

```
31 class Dungeon
32 {
33 public:
34     Player* _player;
35     Room* _entryRoom;
36     Position _spawnPoint;
37
38     /**
39      * @brief This is the default constructor.
40      */
41     Dungeon();
42
43     /**
44      * @brief This is the overloaded constructor.
45      * @param player - the player.
46      * @param entryRoom - the entry room.
47      * @param spawnPoint - the spawn point.
48      */
49     Dungeon(Player* player, Room* entryRoom, Position spawnPoint);
50
51     /**
52      * @brief This is the copy constructor.
53      * @param dungeon - the dungeon to be copied.
54      */
55     Dungeon(const Dungeon& dungeon);
56
57     /**
58      * @brief This is the destructor.
59      */
60     ~Dungeon();
```

```
62     /**
63      * @brief This method displays the details of the dungeon.
64      */
65     void Display() const;
66
67     /**
68      * @brief This method displays the details of the dungeon.
69      * @param room - the room to be traversed.
70      */
71     void Traverse(Room* room);
72
73     /**
74      * @brief This method recursively traverses the dungeon using DFS algorithm.
75      * @param room - the room to start with
76      */
77     void DFSTraversal(Room* room);
78
79     /**
80      * @brief This function implements the Task 3 in pSet4
81      */
82     static void Task3();
83
84     /**
85      * @brief This function implements the Task 4 in pSet4
86      */
87     static void Task4();
88
89     /**
90      * @brief This function implements the Task 5 in pSet4
91      * @param someString - the string to be converted to lower case.
92      * @return
93      */
94     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 }
```

```
37 void Item::SetName(std::string& name)
38 {
39     _name = name;
40 }
41
42 int Item::GetLevel() const
43 {
44     return _level;
45 }
46
47 void Item::SetLevel(int level)
48 {
49     _level = level;
50 }
51
52 int Item::GetAccessPoints() const
53 {
54     return _accessPoints;
55 }
56
57 void Item::SetAccessPoints(int accessPoints)
58 {
59     _accessPoints = accessPoints;
60 }
61
```

```
62 void Item::Display() const
63 {
64     std::cout << "\t=====" << std::endl;
65     std::cout << "\t> " << _name << " Details" << std::endl;
66     std::cout << "\t=====" << std::endl;
67     std::cout << "\tItem Name: " << _name << std::endl;
68     std::cout << "\tItem Level: " << _level << std::endl;
69     std::cout << "\tItem Access Points: " << _accessPoints << std::endl;
70     std::cout << "\t=====" << std::endl;
71 }
72 }
```

Clue.cc Implementation

```
1 #include "Clue.h"
2 #include <iostream>
3
4 Clue::Clue()
5 {
6     _clueNumber = 0;
7     _clueMessage = "Default Clue Message";
8     _timesShown = 0;
9     _maxReads = 3;
10    _roomNumber = 000;
11 }
12
13 Clue::Clue(int clueNumber, std::string& clueMessage, int maxReads, int roomNumber)
14 {
15     _clueNumber = clueNumber;
16     _clueMessage = clueMessage;
17     _timesShown = 0;
18     _maxReads = maxReads;
19     _roomNumber = roomNumber;
20 }
21
22 Clue::Clue(const Clue& clue)
23 {
24     _clueNumber = clue._clueNumber;
25     _clueMessage = clue._clueMessage;
26     _timesShown = clue._timesShown;
27     _maxReads = clue._maxReads;
28     _roomNumber = clue._roomNumber;
29 }
30
31 Clue::~Clue()
32 {
33     _clueNumber = 0;
34     _clueMessage = "";
35     _timesShown = 0;
36     _maxReads = 0;
37     _roomNumber = 0;
38 }
39
40 int Clue::GetClueNumber() const
41 {
42     return _clueNumber;
43 }
44
45 void Clue::SetClueNumber(int clueNumber)
46 {
47     _clueNumber = clueNumber;
48 }
49
50 std::string& Clue::GetClueMessage()
51 {
52     return _clueMessage;
53 }
54
55 void Clue::SetClueMessage(std::string& clueMessage)
56 {
57     _clueMessage = clueMessage;
58 }
59
60 int Clue::GetTimesShown() const
61 {
62     return _timesShown;
63 }
64
```

```

65 void Clue::SetTimesShown(int timesShown)
66 {
67     _timesShown = timesShown;
68 }
69
70 int Clue::GetMaxReads() const
71 {
72     return _maxReads;
73 }
74
75 int Clue::GetRoomNumber() const
76 {
77     return _roomNumber;
78 }
79
80 void Clue::SetRoomNumber(int roomNumber)
81 {
82     _roomNumber = roomNumber;
83 }
84
85 void Clue::Display() const
86 {
87     std::cout << "====" << std::endl;
88     std::cout << "= CLUE: " << _clueNumber << " === Room Number: " << _roomNumber << " ===" << std::endl;
89     std::cout << "====" << std::endl;
90     std::cout << "====" << std::endl;
91     std::cout << _clueMessage << std::endl;
92     std::cout << "-----" << std::endl;
93     std::cout << "Number of Times shown: " << _timesShown << std::endl;
94     std::cout << "Maximum Reads Set to : " << _maxReads << std::endl;
95     std::cout << "====" << std::endl;
96 }

```

Enemy.cc

```

1 #include "Enemy.h"
2 #include <iostream>
3
4 Enemy::Enemy()
5 {
6     _name = "Default Character Name";
7     _health = 100;
8     _attackMoves = {};
9     _friendliness = false;
10    _backpack = {};
11    _shieldsUp = false;
12    _attackPower = 50;
13    _venomReserve = 0;
14 }
15
16 Enemy::Enemy(std::string& name, int venomReserve)
17     : Character(& name)
18 {
19     _health = 100;
20     _attackMoves = {};
21     _friendliness = false;
22     _backpack = {};
23     _shieldsUp = false;
24     _attackPower = 60;
25     _venomReserve = venomReserve;
26 }
27
28 Enemy::Enemy(const Enemy& enemy)
29     : Character(& enemy)
30 {
31     _venomReserve = enemy._venomReserve;
32 }

```

```

34 Enemy::Enemy(const Character& character, int venomReserve)
35 : Character(&character)
36 {
37     _venomReserve = venomReserve;
38 }
39
^o 40 ~Enemy::~Enemy()
41 {
42     _name = "";
43     _health = 0;
44     _attackMoves.clear();
45     _friendliness = false;
46     _backpack.clear();
47     _shieldsUp = false;
48     _attackPower = 0;
49     _venomReserve = 0;
50 }
51
^o 52 int Enemy::GetVenomReserve() const
53 {
54     return _venomReserve;
55 }
56
^o 57 void Enemy::SetVenomReserve(int venomReserve)
58 {
59     _venomReserve = venomReserve;
60 }
61
62 void Enemy::ShowStats() const
63 {
64     std::cout << "*****" << std::endl;
65     std::cout << "\tEnemy Name: " << _name << "\tHealth: " << _health << std::endl;
66     std::cout << "*****" << std::endl;
67     std::cout << "Attack Moves: " << std::endl;
68     std::cout << "-----" << std::endl;
69
70     for (auto& move:const string&: _attackMoves)
71     {
72         std::cout << "}=>" << move << std::endl;
73     }
74     std::cout << "-----" << std::endl;
75     std::cout << "Friendliness: " << (_friendliness) ? "Friendly" : "Not Friendly" << std::endl;
76     std::cout << "Backpack: " << std::endl;
77     std::cout << "-----" << std::endl;
78
79     for (auto& item: _backpack)
80     {
81         item.Display();
82     }
83     std::cout << "-----" << std::endl;
84     std::cout << "Shields Up: " << (_shieldsUp) ? "Shielded" : "Not Shielded" << std::endl;
85     std::cout << "Attack Power: " << _attackPower << std::endl;
86     std::cout << "Venom Reserve: " << _venomReserve << std::endl;
87     std::cout << std::endl;
88 }
```

Character.cc & Room Implementation

```
1 #include "Character.h"
2 #include "Clue.h"
3 #include <iostream>
4
5 Character::Character()
6 {
7     std::vector<std::string> attackMoves = {
8         "Double Jab Punch",
9         "[COMBO] Low Right Kick, Right Jab Punch",
10        "Tackle",
11        "Round Kick, Floor Opponent"
12    };
13
14     _name = "Default Character Name";
15     _health = 100;
16     _attackMoves = {
17         attackMoves[0],
18         attackMoves[1],
19         attackMoves[2],
20         attackMoves[3]
21     };
22     _friendliness = false;
23     _backpack = {};
24
25     _shieldsUp = false;
26     _attackPower = 120;
27 }
28
29 Character::Character(std::string& name)
30 {
31     _name = name;
32     _health = 100;
33     _attackMoves = {};
34     _friendliness = false;
35     _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 }
```

```
66
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 }
```

```
 52 102 std::vector<Item>& Character::GetBackpack()
 53 103 {
 54 104     return _backpack;
 55 105 }
 56
 57 106
 58 107 void Character::SetBackpack(std::vector<Item>& backpack)
 59 108 {
 60 109     _backpack = backpack;
 61 110 }
 62
 63 111
 64 112 bool Character::GetShieldsUp() const
 65 113 {
 66 114     return _shieldsUp;
 67 115 }
 68
 69 116
 70 117 float Character::GetAttackPower() const
 71 118 {
 72 119     return _attackPower;
 73 120 }
 74
 75 121
 76 122 void Character::SetAttackPower(float attackPower)
 77 123 {
 78 124     _attackPower = attackPower;
 79 125 }
 80
 81 126
 82 127 bool Character::ToggleShieldsUp()
 83 128 {
 84 129     _shieldsUp = !_shieldsUp;
 85 130     return _shieldsUp;
 86 131 }
 87
```

```
 52 133 bool Character::ToggleFriendliness()
 53 134 {
 54 135     _friendliness = !_friendliness;
 55 136     return _friendliness;
 56 137 }
 57
 58 138
 59 139 void Character::PickItem(Item* item)
 60 140 {
 61 141     _backpack.push_back(*item);
 62 142 }
 63
 64 143
 65 144 void Character::Attack(Character* character) const
 66 145 {
 67 146     if (_shieldsUp)
 68 147     {
 69 148         std::cout << _name << " is defending." << std::endl;
 70 149     }
 71 150     else
 72 151     {
 73 152         std::cout << _name << " is attacking." << std::endl;
 74 153         character->SetHealth(character->GetHealth() - _attackPower);
 75 154     }
 76 155 }
 77
 78 156
 79 157 void Character::EnterRoom(Room* room)
 80 158 {
 81 159     std::cout << _name << " has entered " << room->GetName() << "." << std::endl;
 82 160     room->TogglePlayerPresence();
 83 161 }
 84
```

```
163 void Character::ShowStats() const
164 {
165     std::cout << "-----" << std::endl;
166     std::cout << "Default Name: " << _name << "Health: " << _health << std::endl;
167     std::cout << "-----" << std::endl;
168     std::cout << "Attack Moves: " << std::endl;
169
170     for (auto& move : const string& : _attackMoves)
171     {
172         std::cout << move << std::endl;
173     }
174
175     std::cout << "Friendliness: " << (_friendliness) ? "Friendly" : "Not Friendly" << std::endl;
176     std::cout << "Backpack: " << std::endl;
177
178     for (auto& item : _backpack)
179     {
180         item.Display();
181     }
182
183     std::cout << "Shields Up: " << (_shieldsUp) ? "Shielded" : "Not Shielded" << std::endl;
184     std::cout << "Attack Power: " << _attackPower << std::endl;
185 }
```

```
187 ///////////////////////////////////////////////////////////////////
188 ////////////// ROOM IMPLEMENTATIONS //////////
189 ///////////////////////////////////////////////////////////////////
190
191 Room::Room()
192 {
193     _name = "Default Room Name";
194     _roomNumber = 10;
195
196     Setup();
197
198     _playerInRoom = false;
199     _accessGranted = false;
200     _prevRoom = Sentinel;
201     _nextRoomOne = Sentinel;
202     _nextRoomTwo = Sentinel;
203 }
204
205 Room::Room(const std::string& name, int roomNumber)
206 {
207
208     _name = name;
209     _roomNumber = roomNumber;
210
211     Setup();
212
213     _playerInRoom = false;
214     _accessGranted = false;
215     _prevRoom = Sentinel;
216     _nextRoomOne = Sentinel;
217     _nextRoomTwo = Sentinel;
218 }
219
220 }
```

```
222 Room::Room(const Room& room)
223 {
224     _name = room._name;
225     _clues = room._clues;
226     _monsters = room._monsters;
227     _treasureTrove = room._treasureTrove;
228     _playerInRoom = room._playerInRoom;
229     _accessGranted = room._accessGranted;
230     _roomNumber = room._roomNumber;
231     _prevRoom = room._prevRoom;
232     _nextRoomOne = room._nextRoomOne;
233     _nextRoomTwo = room._nextRoomTwo;
234 }
235
236 ~Room()
237 {
238     _name = "";
239     _clues.clear();
240     _monsters.clear();
241     _treasureTrove.clear();
242     _playerInRoom = false;
243     _accessGranted = false;
244     _roomNumber = 0;
245     _prevRoom = Sentinel;
246     _nextRoomOne = Sentinel;
247     _nextRoomTwo = Sentinel;
248 }
249
250 void Room::Setup()
251 {
252     std::vector<std::string> cluesMsgs = {
253         "This is clue message number 101",
254         "This is clue message number 102",
255         "This is clue message number 103",
256         "This is clue message number 104",
```

```
256     "This is clue message number 104",
257 };
258 std::vector<std::string> _monstersNames = {
259     "Fire Breathing Horse",
260     "Stinging Robotic Wasp",
261     "Multi-Head Black Mamba",
262     "Hop Hop Toad"
263 };
264 std::vector<std::string> _itemNames = {
265     "Troy's Sword",
266     "Belinda's Precious Necklace",
267     "South African Diamond Jewel Pack",
268     "Shariffah's Golden Knife"
269 };
270
271 _clues = {
272     Clue(clueNumber: 101, [&] cluesMsgs[0], maxReads: 3, _roomNumber),
273     Clue(clueNumber: 101, [&] cluesMsgs[1], maxReads: 3, _roomNumber),
274     Clue(clueNumber: 101, [&] cluesMsgs[2], maxReads: 3, _roomNumber),
275     Clue(clueNumber: 101, [&] cluesMsgs[3], maxReads: 3, _roomNumber)
276 };
277 _monsters = {
278     Character([&] monstersNames[0]),
279     Character([&] monstersNames[1]),
280     Character([&] monstersNames[2]),
281     Character([&] monstersNames[3])
282 };
283 _treasureTrove = {
284     Item([&] itemNames[0], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100),
285     Item([&] itemNames[1], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100),
286     Item([&] itemNames[2], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100),
287     Item([&] itemNames[3], level: div(_Numerator: _roomNumber, _Denominator: 10).quot, accessPoints: 100)
288 };
289 }
290 }
```

```
291 std::string Room::GetName() const
292 {
293     return _name;
294 }
295
296 void Room::SetName(const std::string& name)
297 {
298     _name = name;
299 }
300
301 std::vector<Clue>& Room::GetClues()
302 {
303     return _clues;
304 }
305
306 void Room::SetClues(const std::vector<Clue>& clues)
307 {
308     _clues = clues;
309 }
310
311 std::vector<Character>& Room::GetMonsters()
312 {
313     return _monsters;
314 }
315
316 void Room::SetMonsters(const std::vector<Character>& monsters)
317 {
318     _monsters = monsters;
319 }
320
321 std::vector<Item> Room::GetTreasureTrove() const
322 {
323     return _treasureTrove;
324 }
325
```

```
326 void Room::SetTreasureTrove(const std::vector<Item>& treasureTrove)
327 {
328     _treasureTrove = treasureTrove;
329 }
330
331 bool Room::IsPlayerInRoom() const
332 {
333     return _playerInRoom;
334 }
335
336 bool Room::IsAccessGranted() const
337 {
338     return _accessGranted;
339 }
340
341 int Room::GetRoomNumber() const
342 {
343     return _roomNumber;
344 }
345
346 void Room::SetRoomNumber(int roomNumber)
347 {
348     _roomNumber = roomNumber;
349 }
350
351 const Room& Room::operator*() const
352 {
353     return *this;
354 }
355
356 Room* Room::GetPrevRoom() const
357 {
358     return _prevRoom;
359 }
360
```

```
361 void Room::SetPrevRoom(Room* prevRoom)
362 {
363     _prevRoom = prevRoom;
364 }
365
366 Room* Room::GetNextRoomOne() const
367 {
368     return _nextRoomOne;
369 }
370
371 void Room::SetNextRoomOne(Room* nextRoomOne)
372 {
373     _nextRoomOne = nextRoomOne;
374 }
375
376 Room* Room::GetNextRoomTwo() const
377 {
378     return _nextRoomTwo;
379 }
380
381 void Room::SetNextRoomTwo(Room* nextRoomTwo)
382 {
383     _nextRoomTwo = nextRoomTwo;
384 }
385
386 bool Room::TogglePlayerPresence()
387 {
388     _playerInRoom = !_playerInRoom;
389     return _playerInRoom;
390 }
```

```

392 bool Room::ToggleAccess()
393 {
394     _accessGranted = !_accessGranted;
395     return _accessGranted;
396 }
397
398 void Room::WelcomeMessage()
399 {
400     std::cout << "====" << std::endl;
401     std::cout << "= Welcome to " << _name << " Room Number: " << _roomNumber << " = " << std::endl;
402     std::cout << "====" << std::endl;
403     std::cout << "In this room there both Monsters and Treasures" << std::endl;
404     std::cout << "There a total of " << _treasureTrove.size() << " treasure Items as listed below:" << std::endl;
405     std::cout << "----- Treasures -----" << std::endl;
406     for (auto& item : _treasureTrove)
407     {
408         item.Display();
409     }
410     std::cout << "Additionally you'll face some Monstrous enemies:" << std::endl;
411     std::cout << "----- Enemies -----" << std::endl;
412     for (auto& monster:Character& : _monsters)
413     {
414         std::cout << monster.GetName() << "\t|\tAttack Power: " << monster.GetAttackPower() << std::endl;
415     }
416     std::cout << "Brace Yourself!!!" << std::endl;
417     std::cout << "===== END OF WELCOME MESSAGE =====" << std::endl;
418     std::cout << std::endl;
419 }
420
421
422 Room::RoomPtr Room::Sentinel;

```

Player.cc Implementation

```

1 #include "Player.h"
2 #include <iostream>
3
4 Player::Player()
5 {
6     std::vector<std::string> attackMoves = {
7         "Double Jab Punch",
8         "[COMBO] Low Right Kick, Right Jab Punch",
9         "Tackle",
10        "Round Kick, Floor Opponent"
11    };
12
13    _name = "Default Player Name";
14    _health = 100;
15    _attackMoves = {
16        attackMoves[0],
17        attackMoves[1],
18        attackMoves[2],
19        attackMoves[3]
20    };
21    _friendliness = false;
22    _backpack = {};
23    _shieldsUp = false;
24    _attackPower = 50;
25
26    _score = 0;
27    _monstersSlain = 0;
28 }
29
30 Player::Player(std::string& name)
31     : Character( [&] name)
32 {
33     _score = 0;
34     _monstersSlain = 0;
35 }

```

```
37 Player::Player(const Player& player)
38     : Character(&player)
39 {
40     _score = player._score;
41     _monstersSlain = player._monstersSlain;
42 }
43
44 Player::Player(const Character& character)
45     : Character(&character)
46 {
47     _score = 0;
48     _monstersSlain = 0;
49 }
50
51 Player::~Player()
52 {
53     _name = "";
54     _health = 0;
55     _attackMoves.clear();
56     _friendliness = false;
57     _backpack.clear();
58     _shieldsUp = false;
59     _attackPower = 0;
60
61     _score = 0;
62     _monstersSlain = 0;
63 }
64
65 int Player::GetScore() const
66 {
67     return _score;
68 }
69
```

```
70 void Player::SetScore(int score)
71 {
72     _score = score;
73 }
74
75 int Player::GetMonstersSlain() const
76 {
77     return _monstersSlain;
78 }
79
80 void Player::SetMonstersSlain(int monstersSlain)
81 {
82     _monstersSlain = monstersSlain;
83 }
84
```

```

^ 85 void Player::ShowStats() const
86 {
87     std::cout << "#####" << std::endl;
88     std::cout << "\tPlayer Name: " << _name << "\tHealth: " << _health << std::endl;
89     std::cout << "#####" << std::endl;
90     std::cout << "Attack Moves: " << std::endl;
91     std::cout << "-----" << std::endl;
92
93     for (auto& move : const string& : _attackMoves)
94     {
95         std::cout << "}=>" << move << std::endl;
96     }
97     std::cout << "-----" << std::endl;
98     std::cout << "Friendliness: " << (_friendliness) ? "Friendly" : "Not Friendly" << std::endl;
99     std::cout << "Backpack: " << std::endl;
100    std::cout << "-----" << std::endl;
101
102    for (auto& item : _backpack)
103    {
104        item.Display();
105    }
106    std::cout << "-----" << std::endl;
107    std::cout << "Shields Up: " << (_shieldsUp) ? "Shielded" : "Not Shielded" << std::endl;
108    std::cout << "Player Attack Power: " << _attackPower << std::endl;
109    std::cout << "Player Score: " << _score << std::endl;
110    std::cout << "Player Monsters Slain: " << _monstersSlain << std::endl;
111}
112
113 void Player::AcceptVisitor(PlayerVisitor* visitor)
114 {
115     visitor->VisitPlayer(this);
116 }
117

```

```

118 void PlayerVisitor::VisitPlayer(Player* player)
119 {
120     std::cout << "Player Visitor Upgraded Health to: " << player->GetHealth() + 30 << std::endl;
121     player->SetHealth(player->GetHealth() + 30);
122     player->SetScore(player->GetScore() + 40);
123     std::cout << "#####" << std::endl;
124     std::cout << "\tPlayer Name: " << player->GetName() << "\tHealth: " << player->GetHealth() << std::endl;
125     std::cout << "#####" << std::endl;
126     std::cout << "Attack Moves: " << std::endl;
127     std::cout << "-----" << std::endl;
128
129     for (auto& move : string& : player->GetAttackMoves())
130     {
131         std::cout << "}=>" << move << std::endl;
132     }
133     std::cout << "-----" << std::endl;
134     std::cout << "Friendliness: " << ((player->GetFriendliness()) ? "Friendly" : "Not Friendly") << std::endl;
135     std::cout << "Backpack: " << std::endl;
136     std::cout << "-----" << std::endl;
137
138     for (auto& item : player->GetBackpack())
139     {
140         item.Display();
141     }
142     std::cout << "-----" << std::endl;
143     std::cout << "Shields Up: " << ((player->GetShieldsUp()) ? "Shielded" : "Not Shielded") << std::endl;
144     std::cout << "Player Attack Power: " << player->GetAttackPower() << std::endl;
145     std::cout << "Player Score Updated (by the visitor): " << player->GetScore() << std::endl;
146     std::cout << "Player Monsters Slain: " << player->GetMonstersSlain() << std::endl;
147 }
148

```

Dungeon.cc Implementations

```
1 #include "Dungeon.h"
2 #include <algorithm>
3 #include <iostream>
4
5 Dungeon::Dungeon()
6 {
7     _player = new Player();
8
9     std::vector<std::string> roomNames = {
10         "Entry Room", "Room 1", "Room 2", "Room 3",
11         "Room 4", "Room 5", "Room 6", "Room 7"
12     };
13     // Level 0 Rooms
14     _entryRoom = new Room(roomNames[0], roomNumber: 10);
15     // Level 1 Rooms
16     Room* room1 = new Room(roomNames[1], roomNumber: 20);
17     Room* room2 = new Room(roomNames[2], roomNumber: 22);
18     // Level 2 Rooms
19     Room* room3 = new Room(roomNames[3], roomNumber: 30);
20     Room* room4 = new Room(roomNames[4], roomNumber: 31);
21     Room* room5 = new Room(roomNames[5], roomNumber: 32);
22     Room* room6 = new Room(roomNames[6], roomNumber: 33);
23     _entryRoom->SetNextRoomOne(room1);
24     _entryRoom->SetNextRoomTwo(room2);
25     // room1
26     room1->SetPrevRoom(_entryRoom);
27     room1->SetNextRoomOne(room3);
28     room1->SetNextRoomTwo(room4);
29     // room2
30     room2->SetPrevRoom(_entryRoom);
31     room2->SetNextRoomOne(room5);
32     room2->SetNextRoomTwo(room6);
33     // room3
34     room3->SetPrevRoom(room1);
35
36     room3->SetPrevRoom(room1);
37     // room4
38     room4->SetPrevRoom(room1);
39     // room5
40     room5->SetPrevRoom(room2);
41     // room6
42     room6->SetPrevRoom(room2);
43
44     _spawnPoint = { .x: 0, .y: 0 };
45 }
46 Dungeon::Dungeon(Player* player, Room* entryRoom, Position spawnPoint)
47 {
48     _player = player;
49     std::vector<std::string> roomNames = {
50         "Entry Room", "Room 1", "Room 2", "Room 3",
51         "Room 4", "Room 5", "Room 6", "Room 7"
52     };
53     // Level 0 Rooms
54     _entryRoom = entryRoom;
55     // Level 1 Rooms
56     Room* room1 = new Room(roomNames[1], roomNumber: 20);
57     Room* room2 = new Room(roomNames[2], roomNumber: 22);
58     // Level 2 Rooms
59     Room* room3 = new Room(roomNames[3], roomNumber: 30);
60     Room* room4 = new Room(roomNames[4], roomNumber: 31);
61     Room* room5 = new Room(roomNames[5], roomNumber: 32);
62     Room* room6 = new Room(roomNames[6], roomNumber: 33);
63     _entryRoom->SetNextRoomOne(room1);
64     _entryRoom->SetNextRoomTwo(room2);
65     // room1
66     room1->SetPrevRoom(_entryRoom);
67     room1->SetNextRoomOne(room3);
68     room1->SetNextRoomTwo(room4);
```

```

67     room1->SetNextRoomOne(room3);
68     room1->SetNextRoomTwo(room4);
69     // room2
70     room2->SetNextRoomOne(room5);
71     room2->SetNextRoomTwo(room6);
72     // (local variable) Room* room2
73     room3->SetPrevRoom(room1);
74     // room4
75     room4->SetPrevRoom(room1);
76     // room5
77     room5->SetPrevRoom(room2);
78     // room6
79     room6->SetPrevRoom(room2);
80
81     _spawnPoint = spawnPoint;
82 }
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

```

```

99 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 Dungeon::DFSTraversal(Room* room)
108 {
109     if (room == Room::Sentinel)
110     {
111         return;
112     }
113
114     std::cout << room->GetName() << std::endl;
115     DFSTraversal(room->GetNextRoomOne());
116     DFSTraversal(room->GetNextRoomTwo());
117 }
118
119
120 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)

```

```

132     (room->GetNextRoomTwo() != Room::Sentinel)
133     &&? room->GetNextRoomTwo()->GetName()
134     : "None (This room is an end point. You'll need to go back)" << std::endl;
135     std::cout << "3. Go back to " << (
136         (room->GetPrevRoom() != Room::Sentinel)
137         &&? room->GetPrevRoom()->GetName()
138         : "None (This is the entry Room)" << std::endl;
139     std::cout << "4. Quit" << std::endl;
140     std::cout << "Enter a command: " << std::endl;
141     std::string response;
142     getline(&std::cin, [&]response);
143
144     if (response == "1")
145     {
146         if (room->GetNextRoomOne() == Room::Sentinel)
147         {
148             std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;
149         }
150         else
151         {
152             room = room->GetNextRoomOne();
153         }
154     }
155
156     else if (response == "2")
157     {
158         if (room->GetNextRoomTwo() == Room::Sentinel)
159         {
160             std::cout << "No doors out of this room. You might need to go back. End of traversal" << std::endl;
161         }
162         else
163         {
164             room = room->GetNextRoomTwo();
165         }
166     }
167
168     {
169         room = room->GetNextRoomTwo();
170     }
171
172     else if (response == "3")
173     {
174         room = room->GetPrevRoom();
175     }
176     else if (response == "4")
177     {
178         std::cout << "Quitting game ..." << std::endl;
179         break;
180     }
181     else
182     {
183         std::cout << "Invalid command" << std::endl;
184     }
185 }
```

```

184 void Dungeon::Task3()
185 {
186     auto* dungeon = new Dungeon();
187
188     std::cout << "===== TASK 3 [START] =====" << std::endl;
189     std::cout << "===== TASK 3 [END] =====" << std::endl;
190     std::cout << "You have the following rooms to traverse and explore" << std::endl;
191
192     dungeon->DFSTraversal(dungeon->_entryRoom);
193
194     std::cout << "We Start in the entry Room: " << std::endl;
195     std::cout << std::endl;
196     dungeon->_entryRoom->WelcomeMessage();
197     dungeon->Traverse(dungeon->_entryRoom);
198     std::cout << "===== TASK 3 [END] =====" << std::endl;
199
200 }
201
202 void Dungeon::Task4()
203 {
204     auto* dungeon = new Dungeon();
205     PlayerVisitor* playerVisitor = new PlayerVisitor();
206     std::cout << "===== TASK 4 [START] =====" << std::endl;
207     std::cout << "===== TASK 4 [END] =====" << std::endl;
208     std::cout << std::endl;
209     playerVisitor->VisitPlayer(dungeon->_player);
210     std::cout << "===== TASK 4 [END] =====" << std::endl;
211 }
```

```

213 std::string Dungeon::ToLower(std::string& someString)
214 {
215     std::string result = someString;
216     std::transform(_First: result.begin(), _Last: result.end(), _Dest: result.begin(), ::tolower);
217     return result;
218 }
219
220 void Dungeon::Begin()
221 {
222     std::vector<std::string> commands = {
223         "description", "help", "task3", "task4", "quit", "clear"
224     };
225     std::string response;
226
227     std::cout << "_____" << std::endl;
228     std::cout << "===== WELCOME TO THE DUNGEON =====" << std::endl;
229     std::cout << "_____" << std::endl;
230
231     while (true)
232     {
233         std::cout << "How would you like to proceed? Please type 'help' to view commands" << std::endl;
234         getline(&std::cin, [&]response);
235
236         if (ToLower([&]response) == commands[0])
237         {
238             std::cout << "_____" << std::endl;
239             std::cout << "===== GAME DESCRIPTION =====" << std::endl;
240             std::cout << "_____" << std::endl;
241             std::cout << "This is a simple game program where the player is meant to go through a set of rooms in the dungeon.\n"
242             << "The player's permission to go through a certain room is accessed through a set of conditions which determine\n"
243             << "whether or not the user can access the room. The player starts the game in the entry room and if the player is\n"
244             << "able to slay more than three monsters then they qualify for access both the left (nextRoomOne) and right node (nextRoomTwo)\n"
245             << "otherwise they can only access the room represented by the left node(nextRoomOne). This is the challenge on level one.\n";
246
247             std::cout << "On Level Two the player has four challenges but may choose at least one to tackle.Each condition is link \n"

```

```

248             std::cout << "On Level Two the player has four challenges but may choose at least one to tackle.Each condition is link \n"
249             << "to permission to access specified room(see Figure 1) if the player manages to satisfy all four conditions then \n"
250             << "they can easily move around the whole dungeon freely with access to any room.The game is considered solved at this point." << std::endl;
251             std::cout << "-----" << std::endl;
252         }
253         else if (ToLower([&]response) == commands[1])
254         {
255             std::cout << "_____" << std::endl;
256             std::cout << "===== HELP PAGE [commands] =====" << std::endl;
257             std::cout << "_____" << std::endl;
258             std::cout << "\tdescription: \tShows you the description of the game" << std::endl;
259             std::cout << "\thelp: \tOpens up this help page." << std::endl;
260             std::cout << "\ttask3 \tRuns the implementation of Task 3" << std::endl;
261             std::cout << "\ttask4 \tRuns the implementation of Task 4" << std::endl;
262             std::cout << "\tquit: \tQuits the game" << std::endl;
263             std::cout << "-----" << std::endl;
264         }
265         else if (ToLower([&]response) == commands[2])
266         {
267             Task3();
268         }
269         else if (ToLower([&]response) == commands[3])
270         {
271             Task4();
272         }
273         else if (ToLower([&]response) == commands[4])
274         {
275             std::cout << "Quitting game ..." << std::endl;
276             break;
277         }
278         else if (ToLower([&]response) == commands[5])
279         {
280             #if defined(WIN32) || defined(_WIN32) || defined(_WIN32_) || defined(_NT_)
281
282             else if (ToLower([&]response) == commands[5])
283             {
284                 #if defined(WIN32) || defined(_WIN32) || defined(_WIN32_) || defined(_NT_)
285                     system(_Command: "cls");
286                 #elif __APPLE__ #if defined(WIN32) || defined(_WIN32) || defined(_WIN32_) || defined(_NT__)
287                     system("clear");
288                 #elif __linux__ #elif __APPLE__
289                     system("clear");
290                 #endif #elif __linux__
291             }
292         }
293     }
294 }
295

```