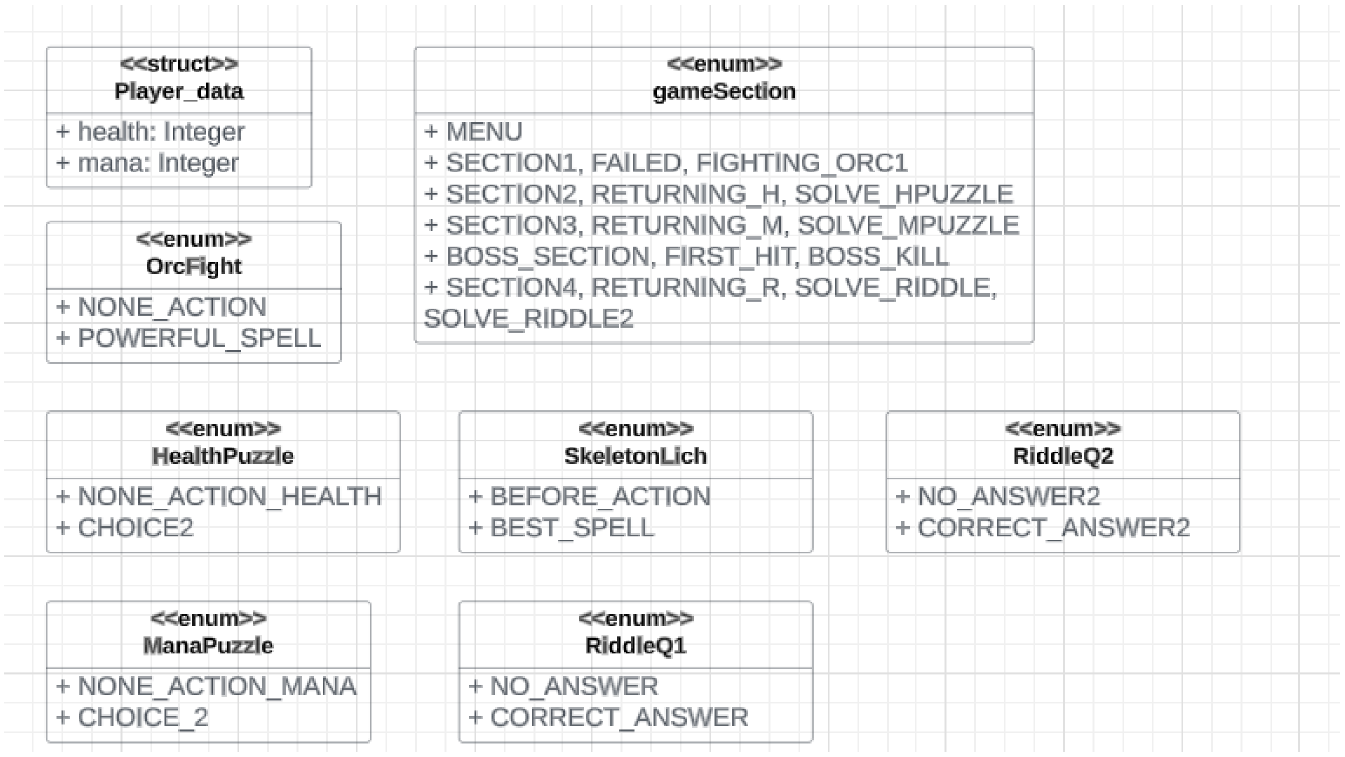
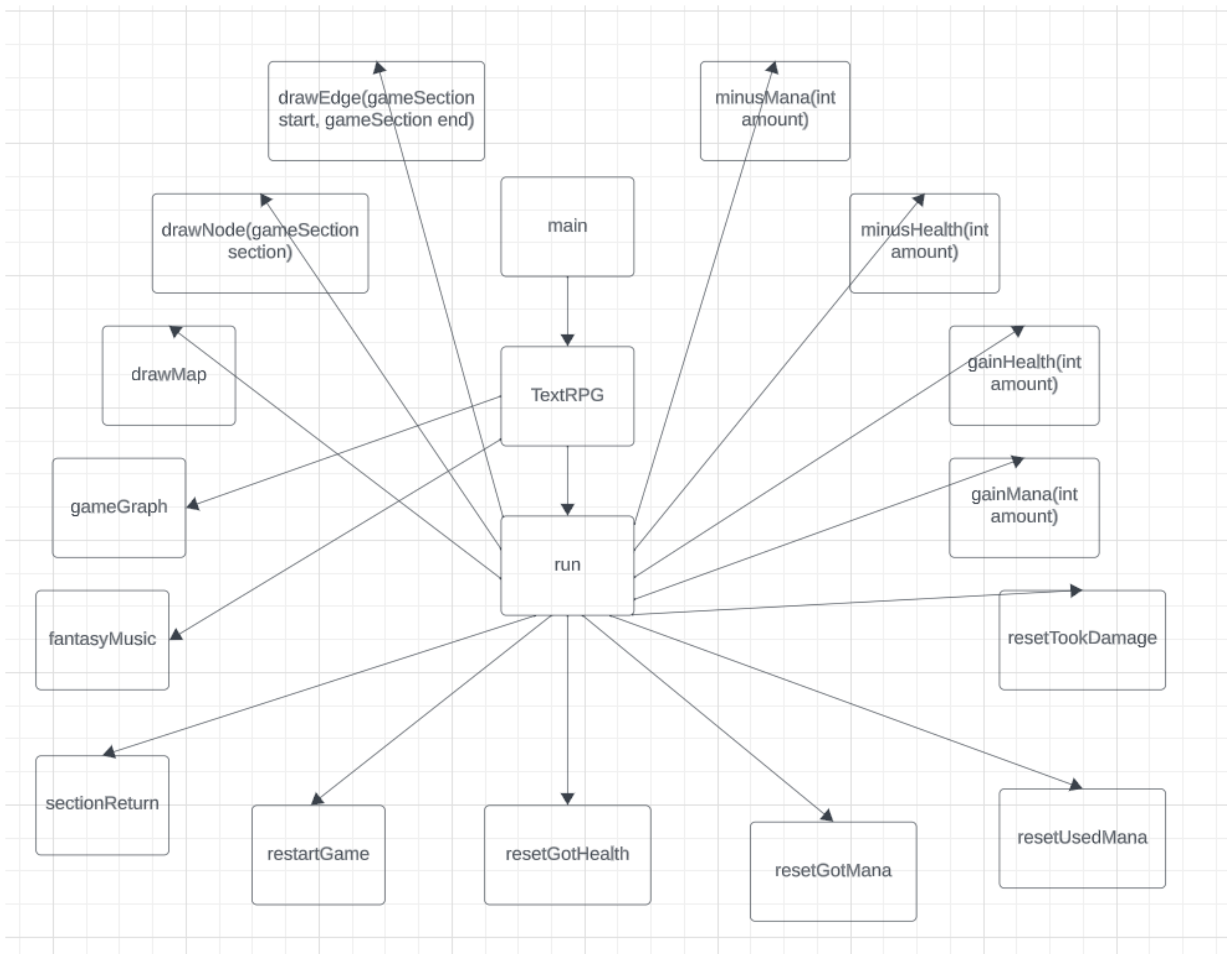
Research question and background

The research question is ‘how I make a Text RPG in C++ using Splashkit?’ The background to this research question or project is related to me, its creator. I have always liked RPG games, especially the Witcher series. From the characters to the world-building, everything is done so well that it left a void in me after I finished playing. It inspires me to create my own game, albeit a simple one, but nonetheless, it brings me great joy in making it.

Programming methodologye





The two picture provides an overview of my Text RPG.
I will divide this report into 3 sections which are object-oriented programming, important parts and graph theory.

OOP

I have encapsulated my code into a class. It makes my code more manageable, and I can easily scale it (adding more functionalities). I have put my structs, enum, variables and a function in private to prevent direct access to it. This ensures my code is controlled and integrity, meaning it is consistent and valid. I put mostly every functions in public because I want those functions to interact directly with the game. For example, I want maps to be drawn, player health and mana to be changed, the game to be restarted and the player to return all by using their corresponding functions.

Important parts

Player data

```
    bool tookDamage = false;
    bool usedMana = false;
    bool gotHealth = false;
    bool gotMana = false;
```

```
    void minusHealth(int amount){
        if(!tookDamage){
            player.health -= amount;
            tookDamage = true;
        }
    }
    void minusMana(int amount){
        if(!usedMana){
            player.mana -= amount;
            usedMana = true;
        }
    }
    void gainHealth(int amount){
        if(!gotHealth){
            player.health += amount;
            gotHealth = true;
        }
    }
    void gainMana(int amount){
        if(!gotMana){
            player.mana += amount;
            gotMana = true;
        }
    }
    void resetTookDamage(){
        tookDamage = false;
    }
    void resetUsedMana(){
        usedMana = false;
    }
    void resetGotHealth(){
         gotHealth = false;
    }
    void resetGotMana(){
         gotMana = false;
    }
```

This code controls player health and mana. I first set some boolean values to false to prevent the player constantly losing health and mana when calling ‘gainMana()’, ‘gainHealth()’, ‘minusMana()’, ‘minusHealth()’ in the game loop.

Allowing player to type letters and numbers

```
       stringstream current_input;
      for(char i = '0'; i <= '9'; ++i) {
          if(key_typed(static_cast<key_code>(i))) {
             current_input << i;
             break;
             }
           }

        for(char i = 'a'; i <= 'z'; i++){
            if(key_typed(static_cast<key_code>(i))) {
               current_input << i;
                break;
                }
             }
```

This code allows the player to type numbers between 0 to 9 and characters from ‘a’ to ‘z’ with stringsteam which converts any data types into a string and ‘key_typed(static_cast<key_code>(i)’ which checks if the character ‘i’ is typed.

User input and move scence

```
string user_input;
else if (key_typed(RETURN_KEY)) {
      user_input = current_input.str();

      // game menu
      if(game_section == MENU){
         if(user_input == "1" && visitedSection.find(SECTION1)== visitedSection.end())
{ // player can not re-enter section 1
             moveToSection(SECTION1);
             }
          else if(user_input == "2") {
                  moveToSection(SECTION2);
          }
          else if(user_input == "3") {
                  moveToSection(SECTION3);
          }
          else if(user_input == "4") {
                  moveToSection(SECTION4);
          }
       }

       // the rest of the code
       current_input.str("");
```

```
        current_input.clear();
}
```

This code manages the player input with `user_input = current_input.str();` to convert to stringstream to `user_input` and `current_input.clear();` to clear the text for the following input of the player.

Text and picture in each section

```
            if(game_section == MENU){
                draw_bitmap("dnd", -1500,-500,     option_scale_bmp(0.05,0.05));
                drawMap();
                draw_text("Welcome young adventurer to an adventure in another 
world!", COLOR_WHITE, 10, 30);
                draw_text("You are in a cave with 4 sections. Type 1, 2, 3, 4 to 
choose where you want to go.", COLOR_WHITE, 10, 50);
            }
            else if(game_section == SECTION1){
                visitedSection.insert(SECTION1);
                draw_bitmap("orc", -280, -380, option_scale_bmp(0.3,0.3));
                draw_text("Upon entering section 1, an orc blocks your path. ", 
COLOR_WHITE, 10, 30);
                draw_text("Type '1' to cast a powerful spell or '2' to run while cast 
a weak spell", COLOR_WHITE, 10, 50);
            }
            else if(game_section == FAILED){
                draw_bitmap("orc", -280, -380, option_scale_bmp(0.3,0.3));
                draw_text("You cast a powerful spell. It dealt some damange, but 
ultimately, the orc got you.", COLOR_WHITE, 10, 30);
                minusMana(50);
                minusHealth(100);
            }
```

This code outputs the text and images of each section by using Splashkit ‘draw_text’ and ‘draw_bitmap’.

Graph theory

Graph theory is a great way to clean my code and add more functionalities, and below are 3 ways it helps my code.

Transit between scene

```
unordered_map<gameSection, vector<gameSection>> graph;
void gameGraph() {
        // rules for each section
        graph[MENU] = {SECTION1, SECTION2, SECTION3, SECTION4};
        graph[SECTION1] = {MENU, FAILED, FIGHTING_ORC1};
        graph[SECTION2] = {MENU, RETURNING_H, SOLVE_HPUZZLE, BOSS_SECTION};
        graph[SECTION3] = {MENU, RETURNING_M, SOLVE_MPUZZLE, BOSS_SECTION};
        graph[BOSS_SECTION] = {FIRST_HIT, BOSS_KILL};
        graph[SECTION4] = {MENU, RETURNING_R, SOLVE_RIDDLE, SOLVE_RIDDLE2,
BOSS_SECTION};
       }

void moveToSection(gameSection nextSection) {
        if(find(graph[game_section].begin(), graph[game_section].end(), nextSection)
!= graph[game_section].end()) {
            game_section = nextSection;
        }
    }
```

This approach is a nice way to organize and transition between scenes. By setting rules for each section of the game state, the scene can be transited only if the condition is met. The logic here is that the enum ‘gameSection’ is the vertex, ‘MENU’, ‘SECTION1,2,3,4’ are nodes, and each node has its own nodes. The first thing I did was to declare an unordered map named ‘graph’ for the enum. Then, I set rules for each one. The ‘moveToSection’ checks whether the section can transit to the other scene. It will transit if the condition is met.

Draw a map

```
    map<gameSection, pair<int, int>> position;
    map<gameSection, string> name;

    void gameGraph() {
        // position for the nodes
        position[MENU] = {400, 100};
        position[SECTION1] = {100, 200};
        position[FAILED] = {50, 300};
        position[FIGHTING_ORC1] = {150, 300};
        position[SECTION2] = {300, 200};
        position[RETURNING_H] = {250, 300};
        position[SOLVE_HPUZZLE] = {350, 300};
        position[SECTION3] = {500, 200};
        position[RETURNING_M] = {450, 300};
        position[SOLVE_MPUZZLE] = {550, 300};
        position[BOSS_SECTION] = {300, 400};
        position[FIRST_HIT] = {250, 500};
        position[BOSS_KILL] = {350, 500};
        position[SECTION4] = {700, 200};
        position[RETURNING_R] = {650, 300};
        position[SOLVE_RIDDLE] = {750, 300};
        position[SOLVE_RIDDLE2] = {850, 300};
        // names for a few section
        name[MENU] = "  Start";
        name[SECTION1] = "Section 1";
        name[SECTION2] = "Section 2";
        name[SECTION3] = "Section 3";
        name[BOSS_SECTION] = "  Boss";
        name[SECTION4] = "Section 4";

    }
```

This code sets up the foundation of the map with the positions of each section and its corresponding names.

```
   void drawNode(gameSection section) {
        int x = position[section].first;
        int y = position[section].second;

        draw_text(name[section], COLOR_WHITE, x - 30, y - 5);
    }



    void drawEdge(gameSection start, gameSection end) {
        int x1 = position[start].first;
        int y1 = position[start].second;
        int x2 = position[end].first;
        int y2 = position[end].second;

        draw_line(COLOR_GREEN, x1, y1, x2, y2);
    }

    void drawMap() {
        for(auto& start : graph) {
            for(auto& end : start.second) {
                drawEdge(start.first, end);
            }
        }
        for(auto& section : position) {
            drawNode(section.first);
        }
    }
```

The ‘drawNode’ function takes the position and name of each node to draw it on the screen with white colour. The ‘drawEdge’ function draws a line or edge between those nodes. Each node has a start and end for the line to be drawn. The ‘drawMap’ function uses the two previous functions to iterate to draw the entire map. This function uses ‘auto’, which auto detects the variable without stating what it is. The ‘for(auto& start : graph)’ detects the ‘unordered_map<gameSection, vector<gameSection>>’. The ‘(auto& end : start.second)’ detects the ‘gameSection&’ and ‘vector<gameSection>’. The ‘for(auto& section : position)’ detects ‘map<gameSection, pair<int, int>>’. After this, I would call the drawMap() inside my game loop, and it will show the map.

Prevent player from re-enter section 1

```
set<gameSection> visitedSection;
// the rest of the code
else if(key_typed(RETURN_KEY)) {
      user_input = current_input.str();

      // game menu
      if(game_section == MENU){
         if(user_input == "1" && visitedSection.find(SECTION1)== visitedSection.end())
{ // player can not re-enter section 1
             moveToSection(SECTION1);
             }
              // the rest of the code
       }
}
// the rest of the code
else if(game_section == SECTION1){
         visitedSection.insert(SECTION1);
         draw_bitmap("orc", -280, -380, option_scale_bmp(0.3,0.3));
         draw_text("Upon entering section 1, an orc blocks your path. ", COLOR_WHITE,
10, 30);
          draw_text("Type '1' to cast a powerful spell or '2' to run while cast a weak
spell", COLOR_WHITE, 10, 50);
}
```

This code demonstrates how section 1 can only be entered once. The ‘visitedSection’ keeps track of what section has been visited. In the game menu, a condition that the player input is ‘1’ and check ‘visitedSection.find(SECTION1)== visitedSection.end()’ if ‘SECTION1’ is in ‘visitedSection’. Also, `visitedSection.insert(SECTION1);` insert ‘SECTION1’ to ‘visitedSection’ if the player enters ‘SECTION1’.

Outcome

The outcome of the text RPG goes as follows. The player is an adventurer exploring a cave. The cave has four sections. In the first section, the player will encounter an orc, and the player will either defeat it or get defeated. Then, they will return to the conjunction. The second and third sections are almost the same as there are three options, and answering the right one will be rewarded and vice versa. The only difference is that the second section gives and minus health and the third section is with mana. The fourth section asks the player two riddles by typing a single word. If the player answers correctly, they will be rewarded greatly, but if they answer incorrectly, it will cost them their life. The second, third and fourth option will lead them to the final boss section, where they must defeat it. The game is over once the player finishes with the boss.

Conclusion

In conclusion, this game captures all the basic features of a TextRPG. There are choices the player can make and each choice they made will have a consequence. The game also has health and mana system and the game will have to restart if their either of those data reaches 0 or lower. Still, there are a lot of improvement needs to be make to transform this game into a full fledge TextRPG. Some of which are an inventory system, the depth of the game, interaction with NPCs and quests.

Video link: https://www.youtube.com/watch?v=ZH1-P1v8r2o&ab_channel=Tomadonna