Final project FRA143

Game : First Journey

กลุ่มที่ 8

ผู้ดำเนินงาน

นายกิตติ์ธเนศ ผาติสว่างพันธ์	67340500049
นางสาวชนัยชนม์ ฉันทวรกุลชัย	67340500051
นายปรธร จันทร์จำรัส	67340500056

1.บทนำ

โปรแกรมเกม RPG นี้เป็นเกมแนว Turn-Based ที่ผู้เล่นจะต้องเลือกใช้สกิลหรือไอเท็มในการโจมตีมอนสเตอร์และบอสสุดท้ายเพื่อพิชิตชัยชนะ โดยมีระบบพลังชีวิต (HP), พลังเวท (Mana), และไอเท็ม Potion ที่ช่วยฟื้นฟู HP เกมนี้ถูกพัฒนาด้วยภาษา C++ โดยใช้หลักการเขียนโปรแกรมเชิงวัตถุ (OOP) เพื่อให้โครงสร้างโปรแกรมมีความชัดเจนและง่ายต่อการพัฒนาเพิ่มเติม

2.รายละเอียดการออกแบบระบบ

2.1 ประเภทเกม

- RPG แบบ Turn-Based
- ผู้เล่นจะต้องต่อสู้กับมอนสเตอร์ทีละตัวโดยเลือกใช้สกิลหรือไอเท็ม
- มีมอนสเตอร์ 3 ตัวและบอสตัวสุดท้ายที่ต้องกำจัดให้หมดเพื่อชนะเกม

2.2 วิธีเล่นเกม

- เลือกระดับความยาก (ง่าย / กลาง / ยาก)
- ต่อสู้กับมอนสเตอร์แบบสุ่ม 3 ตัว
- ต่อสู้กับบอสตัวสุดท้าย
- ในแต่ละเทิร์นผู้เล่นสามารถเลือกใช้สกิล, ใช้ Potion หรือโจมตีธรรมดา
- มอนสเตอร์จะทำการโจมตีสวนกลับแบบสุ่ม
- Mana ของผู้เล่นจะเพิ่มขึ้น +10 ต่อเทิร์น (ไม่เกิน 100) โดยการใช้สกิลบางสกิลจะต้องใช้มานาในการร่าย
- หาก HP ผู้เล่นหมด เกมจะจบทันที
- คะแนนจะถูกคำนวณตามระดับความยากและประเภทของมอนสเตอร์ที่กำจัดได้
- ผู้เล่นสามารถบันทึกคะแนนสูงสุดและเล่นต่อได้

3.รายละเอียดการทำงานของเกม (Game Mechanics)

3.1 ค่าพลังและสกิล

ชื่อสกิล	Mana ที่ใช้	ความเสียหาย (Damage)	หมายเหตุ
Normal Attack	o	15	การโจมตีธรรมดา
Fire Bolt	10	20	สกิลเวท ดาเมจแรงกว่าการโจมตีธรรมดา แต่ใช้มานา
EXCALIBURR (Ultimate)	70	80	สกิลอัลติเมท (ใช้มานามาก)
Potion	0	-	ฟื้นฟู HP 30 หน่วย (มีจำนวนจำกัด)

3.2 การนับคะแนน

ประเภทมอนสเตอร์	คะแนนต่อมอนสเตอร์	การคำนวณคะแนน
มอนสเตอร์ทั่วไป	10 x ความยาก	ตัวอย่าง ความยาก = 2 ได้ 20 คะแนน
บอส	50 x ความยาก	ตัวอย่าง ความยาก = 3 ได้ 20 คะแนน

3.3 Functional Requirements และ Non-Functional Requirements

3.3.1. Functional

1. เลือกระดับความยาก
    - ผู้เล่นสามารถเลือกระดับความยากได้ 3 ระดับ: Easy, Medium, Hard
    - ความยากส่งผลต่อ HP และพลังโจมตีของมอนสเตอร์
2. ระบบต่อสู้
    - ผู้เล่นสามารถเลือกใช้การโจมตีปกติ, เวท Fire Bolt, หรือ EXCALIBURR ได้
    - สามารถใช้น้ำยาเพิ่ม HP ระหว่างต่อสู้ได้
    - การต่อสู้เป็นแบบเทิร์นเบส สลับการโจมตี
3. ระบบ HP และ Mana
    - ตัวละครมี HP และ Mana ซึ่งลดลงตามการต่อสู้และใช้งานเวท
    - Mana จะเพิ่มขึ้นในแต่ละเทิร์นโดยอัตโนมัติ
4. สกิลและเวทของผู้เล่น
    - โจมตีปกติไม่ใช้มานา
    - Fire Bolt ใช้มานา 10 และทำดาเมจ 20
    - EXCALIBURR ใช้มานา 70 และทำดาเมจ 80
5. มอนสเตอร์และบอส
    - มอนสเตอร์มีการสุ่มชื่อ HP และพลังโจมตีตามระดับความยาก
    - ด่านสุดท้ายมีบอส "Dragon King" ซึ่งมี HP และพลังโจมตีสูงมาก
6. ระบบคะแนน
    - ได้คะแนนเมื่อชนะมอนสเตอร์โดยคะแนนที่ได้คำนวณจากระดับความยากของศัตรู
    - แสดงคะแนนรวมเมื่อจบเกม

7. ระบบเล่นซ้ำ

- หลังจบเกม ผู้เล่นสามารถเลือกได้ว่าจะเล่นใหม่หรือจบเกม

3.3.2. Non-Functional

1. ประสิทธิภาพของโค้ด
    - โค้ดสามารถรันได้อย่างราบรื่น ไม่มีข้อผิดพลาด
    - ไม่มีการหน่วงเวลาในเกมเพลย์
2. ความเข้าใจง่าย
    - โค้ดมีโครงสร้างชัดเจน อ่านง่าย
    - การตั้งชื่อคลาส ฟังก์ชัน และตัวแปรบ่งบอกหน้าที่ชัดเจน เช่น Player, Monster, attack()
3. การพัฒนาในอนาคต
    - โครงสร้างโปรแกรมรองรับการเพิ่มระบบใหม่ เช่น เพิ่มคลาสตัวละคร, เพิ่มสกิล, เพิ่มด่านได้ง่าย
    - สามารถเพิ่มมอนสเตอร์หรือบอสใหม่ได้โดยไม่ยุ่งยาก
4. การโต้ตอบกับผู้เล่น
    - มีข้อความอธิบายทุกตัวเลือก เช่น เมื่อเลือกระหว่างเวท, โจมตี, ใช้ยา
    - แสดงสถานะ HP, Mana, และจำนวน Potion ทุกเทิร์น

4.โครงสร้างโปรแกรม

4.1 คลาสหลัก (Main classes)

คลาส	**รายละเอียด**
Character	คลาสพื้นฐานสำหรับตัวละคร Player และ Monster มี attribute และ method สำหรับจัดการ HP และโจมตี
Player	สืบทอดจาก Character มีระบบ Mana, Potion, สกิลต่าง ๆ และแสดงสถานะผู้เล่น
Monster	สืบทอดจาก Character มีระบบสร้างมอนสเตอร์แบบสุ่มตามระดับความยาก และบอสพิเศษ
Game	ควบคุมการทำงานทั้งหมดของเกม เช่น เริ่มเกม สุ่มมอนสเตอร์ ต่อสู้ แสดงผลคะแนน

4.2 รายละเอียดแต่ละคลาส

1. Class Character

- Attributes:
    - name (string)
    - hp (int)
    - attackPower (int)
- Methods:
    - getHP() : คืนค่า HP ปัจจุบัน
    - takeDamage(int) : ลด HP ตามความเสียหายที่ได้รับ
    - isAlive() : ตรวจสอบว่ามี HP มากกว่า 0 หรือไม่
    - attack() : คืนค่าพลังโจมตีแบบสุ่ม

2. Class Player (สืบทอดจาก Character)

- Attributes:
    - mana (int) เริ่มต้น 100
    - potions (int) เริ่มต้น 3

- Methods:
    - skill1() : Normal Attack (โจมตี 15 หน่วย ไม่ใช้ Mana)
    - fireBolt() : Fire Bolt (โจมตี 20 หน่วย ใช้ Mana 10)
    - ultimate() : EXCALIBURR (โจมตี 80 หน่วย ใช้ Mana 70)
    - usePotion() : ฟื้นฟู HP 30 หน่วย
    - gainMana() : เพิ่ม Mana 10 หน่วยต่อเทิร์น (ไม่เกิน 100)
    - showStatus() : แสดงค่า HP, Mana, Potion ปัจจุบัน

3. Class Monster (สืบทอดจาก Character)

- สร้างมอนสเตอร์โดยสุ่มค่าพลังชีวิตและพลังโจมตีตามระดับความยาก
- บอสจะมี HP สูงกว่าและชื่อเฉพาะเช่น "Dragon King"

4. Class Game

- Attributes:
    - difficulty (int) เลือกระดับความยาก (1-3)
    - score (int) คะแนนสะสมของผู้เล่น
- Methods:
    - start() : เริ่มเกม เลือกระดับความยาก
    - getRandomMonster() : สุ่มสร้างมอนสเตอร์ทั่วไป
    - createBoss() : สร้างบอสตามระดับความยาก
    - fight(monster) : ลูปต่อสู้ระหว่างผู้เล่นกับมอนสเตอร์
    - showHighScore() : แสดงคะแนนสูงสุด
    - saveHighScore() : บันทึกคะแนนลงไฟล์ highscore.txt

5. ส่วนประกอบของโปรแกรม

5.1 การนำเข้าไลบรารี

ใช้สำหรับเรียกใช้งานฟังก์ชันมาตรฐาน เช่น การรับ-ส่งข้อมูล (cin, cout), การจัดการข้อความ, ลิสต์, การสุ่มตัวเลข และการจัดการเวลา

```
1   #include <iostream>
2   #include <string>
3   #include <vector>
4   #include <cstdlib>
5   #include <ctime>
6   using namespace std;
```

5.2 คลาส Character (คลาสพื้นฐาน)

เป็นคลาสแม่ที่เก็บข้อมูลพื้นฐานของตัวละคร เช่น ชื่อ, พลังชีวิต (HP), พลังโจมตี และฟังก์ชันที่ใช้ในเกม เช่น การโจมตี, การรับความเสียหาย และตรวจสอบสถานะการมีชีวิต

```
8   class Character {
9   protected:
10      string name;
11      int hp;
12      int attackPower;
13  public:
14      Character(string n, int h, int a) : name(n), hp(h), attackPower(a) {}
15      virtual ~Character() {}
16
17      string getName() { return name; }
18      int getHP() { return hp; }
19      void takeDamage(int dmg) { hp -= dmg; if (hp < 0) hp = 0; }
20      bool isAlive() { return hp > 0; }
21      virtual int attack() { return attackPower; }
22  };
```

```
    int useExcaliburr() {
        if (mana >= 70) {
            mana -= 70;
            cout << name << " uses EXCALIBURR! Massive damage! (-70 Mana)\n";
            return 90;
        } else {
            cout << "Not enough mana!\n";
            return 0;
        }
    }

    void showStatus() {
        cout << "[" << name << "] HP: " << hp << " | Mana: " << mana << " | Potions: " << potions << "\n";
    }
};
```

5.3 คลาส Player (ผู้เล่น)

สืบทอดมาจาก Character เพิ่มคุณสมบัติเฉพาะของผู้เล่น เช่น ค่ามานา (Mana), น้ำยา (Potion) และทักษะเวทมนตร์ เช่น Fire Bolt และ EXCALIBURR พร้อมทั้งสามารถฟื้นฟูมานาและแสดงสถานะได้

```
class Player : public Character {
private:
    int potions;
    int mana;
public:
    Player(string n) : Character(n, 100, 20), potions(5), mana(100) {}

    void usePotion() {
        if (potions > 0) {
            hp += 30;
            if (hp > 100) hp = 100;
            potions--;
            cout << name << " uses a potion! (+30 HP, Potions left: " << potions << ")\n";
        } else {
            cout << "No potions left!\n";
        }
    }

    void gainMana(int amount) {
        mana += amount;
        if (mana > 100) mana = 100;
    }

    int getMana() { return mana; }

    int useFireBolt() {
        if (mana >= 10) {
            mana -= 10;
            cout << name << " casts Fire Bolt! (-10 Mana)\n";
            return 25;
        } else {
            cout << "Not enough mana!\n";
            return 0;
        }
    }
```

5.4 คลาส Monster (มอนสเตอร์)

สืบทอดจาก Character ใช้สำหรับสร้างศัตรูทั่วไปและบอส มีเพียงค่าพลังชีวิตและพลังโจมตี ไม่ซับซ้อน

5.5 คลาส Game (ควบคุมการทำงานของเกม)

```
76  class Monster : public Character {
77  public:
78      Monster(string n, int h, int a) : Character(n, h, a) {}
79  };
81  class Game {
82  private:
83      Player* player;
84      int score;
85      int difficulty;
86  public:
87      Game(string playerName) {
88          player = new Player(playerName);
89          score = 0;
90      }
91
92      ~Game() {
93          delete player;
94      }
95
96      void chooseDifficulty() {
97          cout << "Choose difficulty level:\n1. Easy\n2. Medium\n3. Hard\n> ";
98          cin >> difficulty;
99          if (difficulty < 1 || difficulty > 3) difficulty = 2;
100     }
101
102     Monster getRandomMonster() {
103         vector<string> names = {"Slime", "Goblin", "Orc", "Wolf"};
104         string mName = names[rand() % names.size()];
105         int mHP = 40 + (rand() % 20) + difficulty * 5;       // ลด HP
106         int mAtk = 6 + difficulty * 2;                       // ลดพลังโจมตี
107         return Monster(mName, mHP, mAtk);
108     }
109
110     Monster createBoss() {
111         return Monster("Dragon King", 400, 20);              // ลด HP และ Atk
112     }
```

```cpp
    void fight(Monster& enemy) {
        cout << "A wild " << enemy.getName() << " appears! (HP: " << enemy.getHP() << ")\n";

        while (enemy.isAlive() && player->isAlive()) {
            player->showStatus();
            cout << "Enemy HP: " << enemy.getHP() << "\n";
            cout << "\n=== Choose an action ===\n";
            cout << "1. Normal Attack (0 Mana, 20 dmg)\n";
            cout << "2. Fire Bolt (10 Mana, 25 dmg)\n";
            cout << "3. EXCALIBURR (70 Mana, 90 dmg)\n";
            cout << "4. Use Potion (+30 HP)\n> ";

            int choice;
            cin >> choice;
            int dmg = 0;
            switch (choice) {
                case 1:
                    dmg = player->attack();
                    cout << "You attacked for " << dmg << " damage!\n";
                    break;
                case 2:
                    dmg = player->useFireBolt();
                    break;
                case 3:
                    dmg = player->useExcaliburr();
                    break;
                case 4:
                    player->usePotion();
                    break;
                default:
                    cout << "Invalid action!\n";
            }
```

จัดการลอจิกหลักของเกม เช่น

- ให้ผู้เล่นเลือกความยาก
- สุ่มมอนสเตอร์
- ดำเนินการต่อสู้
- คำนวณคะแนน
- สร้างบอสและตรวจสอบว่าผู้เล่นยังรอดอยู่หรือไม่

ฟังก์ชันภายในที่สำคัญ:

- chooseDifficulty(): เลือกระดับความยาก
- getRandomMonster(): สร้างมอนสเตอร์สุ่ม
- createBoss(): สร้างบอสสุดท้าย
- fight(): การต่อสู้ระหว่างผู้เล่นกับมอนสเตอร์
- start(): เริ่มเกม (รอบ 3 ด่าน + บอส)

5.6 ฟังก์ชัน main (จุดเริ่มต้นของโปรแกรม)

```cpp
int main() {
    srand(time(0));
    string name;
    char playAgain;

    do {
        cout << "Enter your character's name: ";
        cin >> name;

        Game g(name);
        g.start();

        cout << "Do you want to play again? (y/n): ";
        cin >> playAgain;
        cout << "-----------------------------------\n";
    } while (playAgain == 'y' || playAgain == 'Y');

    cout << "Thanks for playing!\n";
    return 0;
}
```

```cpp
            if (dmg > 0) enemy.takeDamage(dmg);

            if (enemy.isAlive()) {
                int enemyDmg = enemy.attack();
                player->takeDamage(enemyDmg);
                cout << enemy.getName() << " attacks you for " << enemyDmg << " damage!\n";
            }

            player->gainMana(20); // เพิ่มการฟื้น Mana
            cout << "You regained 20 Mana.\n";
            cout << "-------------------------\n";
        }

        if (player->isAlive()) {
            cout << "You defeated " << enemy.getName() << " Victory!\n";
            score += (enemy.getName() == "Dragon King") ? 100 : 20;
        } else {
            cout << "You were defeated... Game Over.\n";
        }
    }

    void start() {
        chooseDifficulty();

        for (int i = 0; i < 3 && player->isAlive(); i++) {
            Monster m = getRandomMonster();
            fight(m);
        }

        if (player->isAlive()) {
            cout << " Final Boss Appears! \n";
            Monster boss = createBoss();
            fight(boss);
        }

        cout << "Final Score: " << score << "\n";
    }
};
```

- เริ่มต้นเกมโดยให้ผู้เล่นกรอกชื่อ
- สร้างออบเจ็กต์ของ Game และเรียกใช้งาน
- หลังจบเกม ถามว่าผู้เล่นต้องการเล่นใหม่หรือไม่
- ใช้ do-while วนซ้ำหากผู้เล่นต้องการเล่นอีก

6. การไหลของโปรแกรม (Program Flow)

1. เริ่มเกมและให้ผู้เล่นเลือกระดับความยาก
2. สุ่มมอนสเตอร์ 3 ตัวตามระดับความยาก
3. ต่อสู้กับมอนสเตอร์แต่ละตัวทีละตัว
4. เมื่อชนะมอนสเตอร์ 3 ตัว ให้ต่อสู้กับบอสสุดท้าย
5. หาก HP ผู้เล่นหมด เกมจบทันที
6. หลังชนะทั้งหมด คำนวณคะแนนรวม และถามผู้เล่นว่าจะเล่นใหม่หรือไม่
7. บันทึกคะแนนสูงสุดลงไฟล์และแสดงผลคะแนนสูงสุดในเกม

7. สรุป (Conclusion)

โปรแกรมเกม RPG นี้ออกแบบมาเพื่อให้ผู้เล่นได้สัมผัสประสบการณ์การต่อสู้แบบ Turn-Based ผ่าน Console ด้วยฟีเจอร์สกิลหลากหลาย การจัดการทรัพยากร Mana และ Potion รวมทั้งระบบการนับคะแนนและบันทึกผลคะแนนสูงสุด โปรแกรมนี้พัฒนาด้วยภาษา C++ โดยใช้หลัก OOP ทำให้โครงสร้างโปรแกรมมีความยืดหยุ่นและง่ายต่อการดูแลรักษาและขยายในอนาคต