Loshan Sundaramoorthy

Programmerings Sprog – CPP Obligatorisk Opgave

1. Begrund din motivation for at vælge din opgave

I første omgang havde jeg valgt oversætter, men det blev hurtigt for komplekst og tidskrævende, hvis jeg skulle nå at opfylde min mål. Jeg valgte derfor at lave skiftet over til spil, og fandt frem til jeg vil kode et Snake spil. Kodning af dette spil indebærer grundlæggende spiludvikling. Opgaven kræver implementering af spilmekanikker, brugerinput, grafik, lyd og datalagring, hvilket er med til at give mig en omfattende forståelse af forskellige aspekter af C++ og raylib. Opgaven giver også god mulighed for at anvende objektorienteret programmering (OOP) og standardbiblioteket (STL). Jeg har tidligere udviklet en game-engine i C ved brug af SDL, så det gav god mening for mig at inkorporere RayLib for af udfordre mig selv yderligere.

2. Anvendelse af STL

Der anvendes **`std::deque`**, som bruges til at repræsentere slangens krop. Deque tillader hurtig indsættelse og fjernelse af elementer både i for- og bagenden, hvilket er nødvendigt for slangens bevægelse:

```
std::deque<Vector2> segments;
```

Der anvendes **`std::string`** til at håndtere filnavne og tekststrenge i spillet, såsom højeste score filnavnet:

```
const std::string HIGH_SCORE_FILE = "high_score.txt";
```

Der anvendes **`std::ifstream`** og **`std::ofstream`**, til at læse og skrive højeste score fra og til en fil:

```
std::ifstream file(filename);
```

```
std::ofstream file(filename);
```

3. Anvendelse af pointere

Pointere anvendes til at håndtere dynamisk hukommelse og ressourcer, specielt for food-objektet.

Food objekt: Pointeren til Food klassen bruges til at dynamisk allokere og deallokere hukommelse:

```
Food* food;
```

Hukommelsen allokeres i Game klassens konstruktør og deallokeret i desutruktøren:

```
food = new Food(snake.segments);
```

```
delete food;
```

4. C++ klasser

Klasser anvendes gennem hele projektet til at organisere koden og implementere OOP.

- Game: Håndtere hele spillets logik, opdateringer og rendering.
- Snake: Repræsenterer slangen og dens adfærd (bevægelse, draw, reset).
- Food: Repræsentere food og dens adfærd (placering, tegning).

- Utils: Indeholder konfigurationer og hjælpefunktioner.

5. **Vis og forklar assembler kode**

write_high_score(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>> const&, int):

```
push    rbp
mov     rbp, rsp
sub     rsp, 560
mov     qword ptr [rbp - 8], rdi
mov     dword ptr [rbp - 12], esi
```

Initialisering af stack ved at gemme pointer rbp og justere stackpointer. Lagrer filename og high_score i stacken.

```
mov     rsi, qword ptr [rbp - 8]
lea     rdi, [rbp - 528]
mov     qword ptr [rbp - 552], rdi      # 8-byte Spill
mov     edx, 16
call    std::basic_ofstream<char, std::char_traits<char>>::basic_ofstream(std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char>> const&, std::_Ios_Openmode)@PLT
```

Initialiserer ofstream object med filename.

```
mov     rdi, qword ptr [rbp - 552]      # 8-byte Reload
call    std::basic_ofstream<char, std::char_traits<char>>::is_open()@PLT
mov     byte ptr [rbp - 541], al        # 1-byte Spill
jmp     .LBB0_1
.LBB0_1:
mov     al, byte ptr [rbp - 541]        # 1-byte Reload
test    al, 1
jne     .LBB0_2
jmp     .LBB0_6
```

Kalder is_open og tjekker resultatet. Springer til .LBB0_2 hvis filen er åben, ellers .LBB0_6.

```
.LBB0_2:
    mov    esi, dword ptr [rbp - 12]
    lea    rdi, [rbp - 528]
    call   std::ostream::operator<<(int)@PLT
    jmp    .LBB0_3
.LBB0_3:
    lea    rdi, [rbp - 528]
    call   std::basic_ofstream<char, std::char_traits<char>>::close()@PLT
    jmp    .LBB0_4
```

Kalder operator for at skrive high score. Kalder close for at lukke filen.

```
.LBB0_6:
    lea    rdi, [rbp - 528]
    call   std::basic_ofstream<char, std::char_traits<char>>::~basic_ofstream()@PLT
    add    rsp, 560
    pop    rbp
    ret
```

Kalder destruktør for ofstream, justerer stackpointer, gendanner basepointer og returnerer.

6. Organisering af C++ kode

Projektet er organiseret i flere filer for at sikre en klar struktur:

- **Header filer (.h):**
  - **Utils.h**
  - **Game.h**
  - **Snake.h**
  - **Food.h**
  - **Level.h**
- **Kilde filer (.cpp):**
  - **Utils.cpp**
  - **Game.cpp**
  - **Snake.cpp**
  - **Food.cpp**
  - **Level.cpp**
  - **SnakeGame.cpp (Main)**

7. Anvendelse af C++ core guidelines

Ressource Management (RAII)

Ressourcer som lyde indlæses i konstruktøren og frigives i destruktører.

```
Game::Game() : snake(), is_running(true), score(0), high_score(read_high_score(HIGH_SCORE_FILE)), state(MENU), current_level(1), snake_speed(1.0f) {
    InitAudioDevice();
    eat_sound = LoadSound("C:/Users/Moorthy/Desktop/food.mp3");
    game_over_sound = LoadSound("C:/Users/Moorthy/Desktop/gameover.mp3");
    background_music = LoadMusicStream("C:/Users/Moorthy/Desktop/music.mp3");
    PlayMusicStream(background_music);
    SetMusicVolume(background_music, 0.1f);
    food = new Food(snake.segments);
    level = new Level(current_level);
}

Game::~Game() {
    UnloadSound(eat_sound);
    UnloadSound(game_over_sound);
    UnloadMusicStream(background_music);
    CloseAudioDevice();
    delete food;
    delete level;
}
```

Type Safety

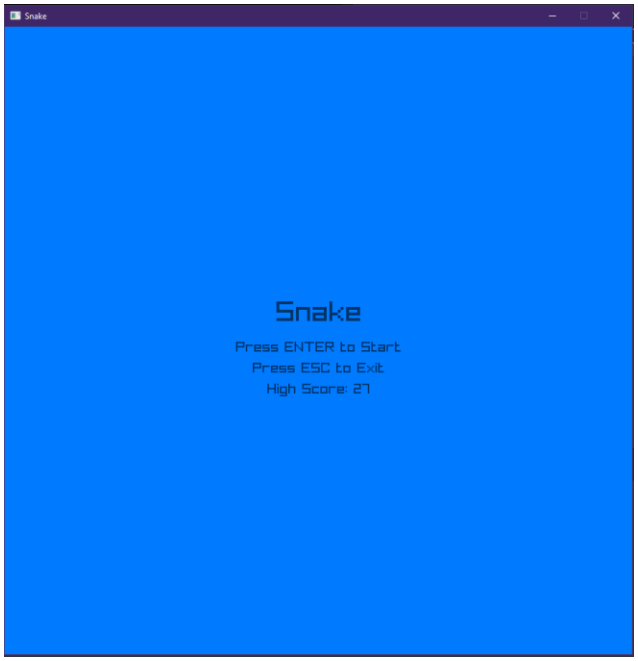
Jeg bruger enum class til at definere spiltilstande.

```
enum GameState {
    MENU,
    PLAYING,
    GAME_OVER
};
```

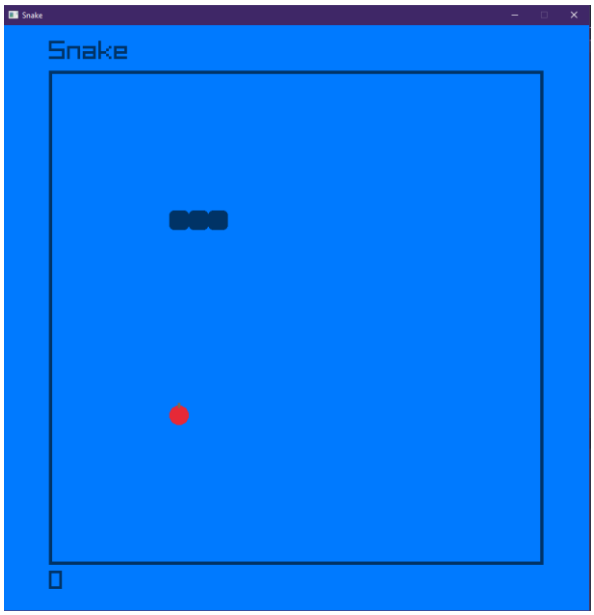
8. Dokumentation af C++ programmet

Her er nogle screenshots.

Screenshot af Menu Skærm



Screenshot af Spillet



Screenshots Snake::render()

Funktionen tegner slangen på skærmen ved at iterere over alle segmenter og tegne dem som (runde) rektangler

```
void Snake::render() {
    for (const auto& segment : segments) {
        float x = segment.x;
        float y = segment.y;
        Rectangle rect =
        {
            screen_margin + x * cell_dimension, screen_margin + y * cell_dimension,
            (float)cell_dimension, (float)cell_dimension
        };
        DrawRectangleRounded(rect, 0.5, 6, deep_blue);
    }
}
```

1. For hvert segment i segments deque'en hentes segmentets x og y koordinater.
2. Position af segmentet på skærmen regnes ved at justere x og y med screen_margin og plusser med cell_dimension.
3. Hvert segment tegnes som en rektangel med DrawRectangleRounded-funktionen, med farven deep_blue.

read_high_score & write_high_score

Funktionerne håndtere læsning og skrivning af højeste score fra og til en fil.

```
int read_high_score(const std::string& filename) {
    std::ifstream file(filename);
    int high_score = 0;
    if (file.is_open()) {
        file >> high_score;
        file.close();
    }
    else {
        std::ofstream new_file(filename);
        if (new_file.is_open()) {
            new_file << high_score;
            new_file.close();
        }
    }
    return high_score;
}

void write_high_score(const std::string& filename, int high_score) {
    std::ofstream file(filename);
    if (file.is_open()) {
        file << high_score;
        file.close();
    }
}
```

read_high_score:

1. Forsøger at åbne filen med std::ifstream.
2. Hvis filen åbnes, læser den højeste score og lukker filen.
3. Hvis filen ikke åbnes (filen ikke findes) oprettes en ny fil med std::ofstream og en standardværdi på 0 skrives til filen.

write_high_score

1. Åbnler filen std::ofstream.
2. Hvis filen åbnes, skriver den højeste score til filen og lukker filen.