Blightwood Abyss – Survival Game

Autor: Wojciech Pilch

1. Opis projektu

1.1. Główne założenie projektu

Założeniem projektu jest napisanie gry w gatunku Survival/Horde Mode, w której gracz mierzy się z kolejnymi falami przeciwników. Wraz z postępem rozgrywki przeciwnicy stają się coraz liczniejsi.

Gracz eliminuje wrogów, strzelając w ich kierunku pociskami zwanymi umownie „łzami". Za pokonanie przeciwnika oraz ukończenie fali gracz otrzymuje punkty, które pod koniec rozgrywki są zapisywane w bazie danych. Dodatkowo, przeciwnicy mogą po sobie pozostawić losowy łup wzmacniający lub osłabiający określone statystyki postaci, takie jak zadawane obrażenia, szybkość ruchu czy maksymalne zdrowie. Część przedmiotów może zostać odblokowanych przez gracza, jeżeli spełni on określone warunki. Inne będą od razu dostępne dla każdego.

Gra kończy się, gdy zdrowie gracza spadnie do zera. Wówczas jego wynik jest porównywany z dotychczasowym rekordem – jeśli jest wyższy, zostaje zapisany w bazie danych jako jego nowy najlepszy rezultat. Na ekranie wyświetla się również stosowna informacja.

1.2. Technologia

Cały projekt jest realizowany w języku C++, z wykorzystaniem paradygmatu programowania obiektowego. Za stworzenie interfejsu graficznego odpowiada, dedykowana tworzeniu aplikacji oraz gier, biblioteka graficzna <raylib>. Ponadto, aby zapewnić pełną funkcjonalność programu, wykorzystano następujące biblioteki zewnętrzne:

1) <iostream>
2) <vector>
3) <regex>
4) <string>
5) <fstream>
6) <random>
7) <mutex>
8) <sstream>
9) <filesystem>
10) <map>
11) <thread>
12) <memory>
13) <atomic>

Niektóre biblioteki, takie jak <string>, czy <vector>, były niezbędne do prawidłowego działania programu, podczas gdy inne, jak <random> lub <memory>, istotnie ułatwiały implementację i zwiększały czytelność kodu.

Podczas implementacji programu, kod został podzielony na szereg klas, z których każda odpowiada za konkretną funkcjonalność aplikacji. Główne elementy programu obsługiwane są przez klasy *Game* oraz *Menu*. Logika związana z postaciami, przeciwnikami, pociskami czy przedmiotami sotała zaimplementowana w klasach *Character*, *Enemy*, *Tears* oraz *Items*. Za warstwę graficzną w dużej mierze odpowiada klasa *GameUI*. Z kolei działania wykonywane przed właściwym uruchomieniem gry – takie jak ustawienia ekranu, ładowanie zasobów graficznych, danych użytkowników czy statystyk postaci zostały przypisane do klas *ScreenSettings, LoadingTextures, UserInfo* oraz *CharacterData*.

Taka struktura programu umożliwiła efektywne wykorzystanie paradygmatów programowania obiektowego, w szczególności dziedziczenia oraz polimorfizmu. W wielu klasach zastosowano również wzorzec projektowy *Singleton*, który w tym przypadku okazał się szczególnie użyteczny – pozwolił m.in. uniknąć wielokrotnego ładowania tych samych tekstur.

2. Specyfikacja zewnętrzna

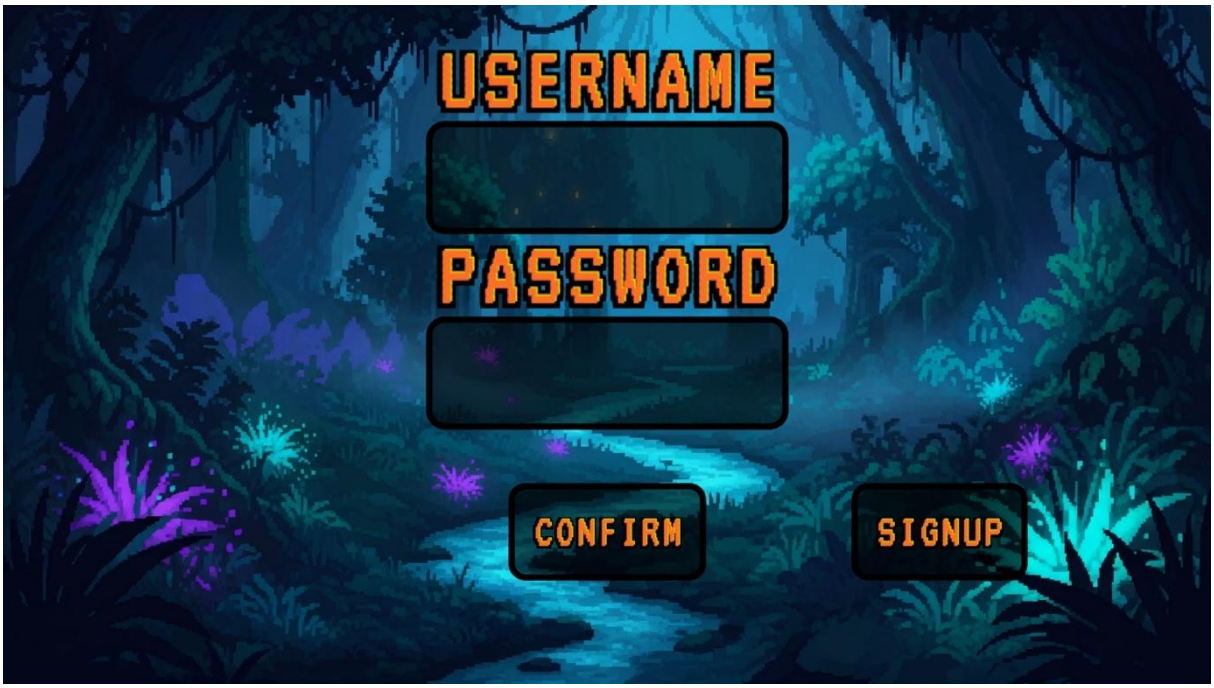
Interfejs programu składa się z dwóch, głównych elementów – menu oraz rozgrywki. Domyślnie program uruchamiany jest w trybie pełnoekranowym, jednakże istnieje możliwość zmiany trybu na okienkowy po naciśnięciu przycisku F11 – w takim przypadku okno programu zmniejsza się do około 2/3 wielkości ekranu.

2.1. Menu

Po włączeniu programu użytkownik jest witany ekranem startowym, który przedstawia nazwę gry. Użytkownik po naciśnięciu lewego przycisku myszy zostaje przekierowany do kolejnego menu.

Rys. 2.1. Ekran startowy programu

Kolejnym menu jest sekcja umożliwiająca logowanie użytkownika. W celu zalogowania się, użytkownik musi wprowadzić swoje dane, a następnie je potwierdzić.

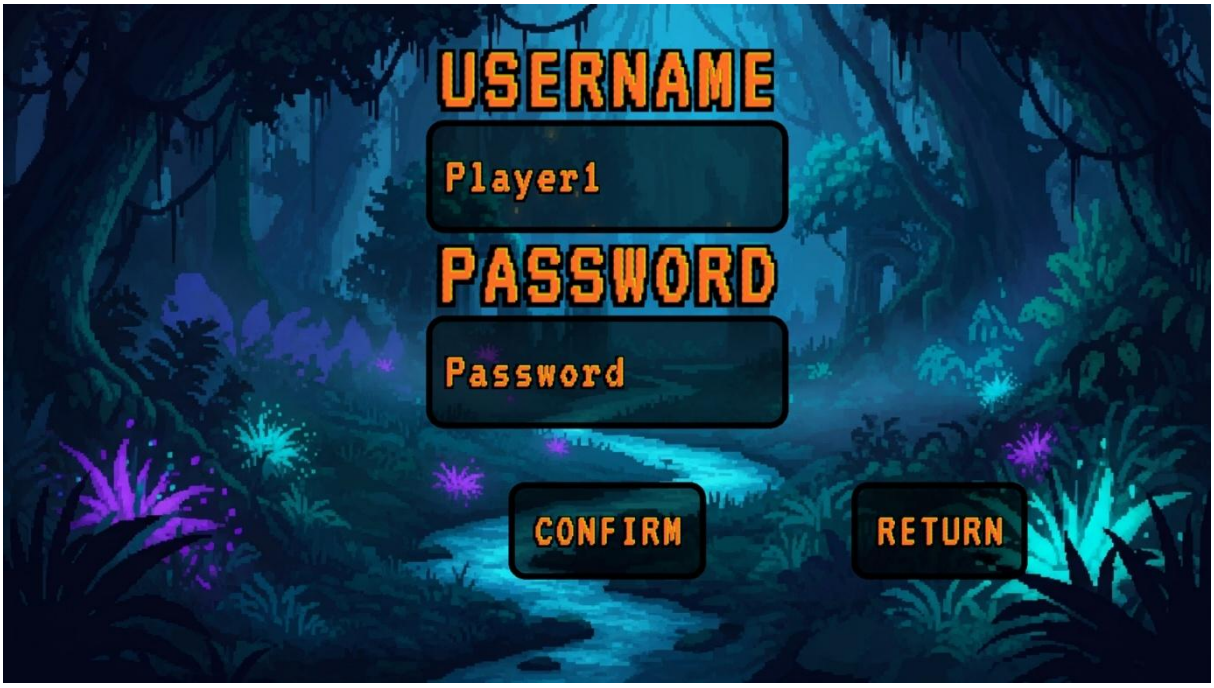


Rys. 2.2. Ekran logowania

Jeśli podane dane są nieprawidłowe, program wyświetli na ekranie odpowiedni komunikat.

Rys. 2.3. Ekran logowania – wyświetlanie błędu

Jeśli użytkownik korzysta z programu po raz pierwszy, może on wybrać opcję „*rejestracja*”, która pozwoli na zarejestrowanie nowego konta gracza w bazie danych. Istnieje możliwość powrotu do pierwotnego ekranu logowania, poprzez wciśnięcie przycisku „*powrót*”.



Rys. 2.4. Ekran logowania – rejestracja nowego konta

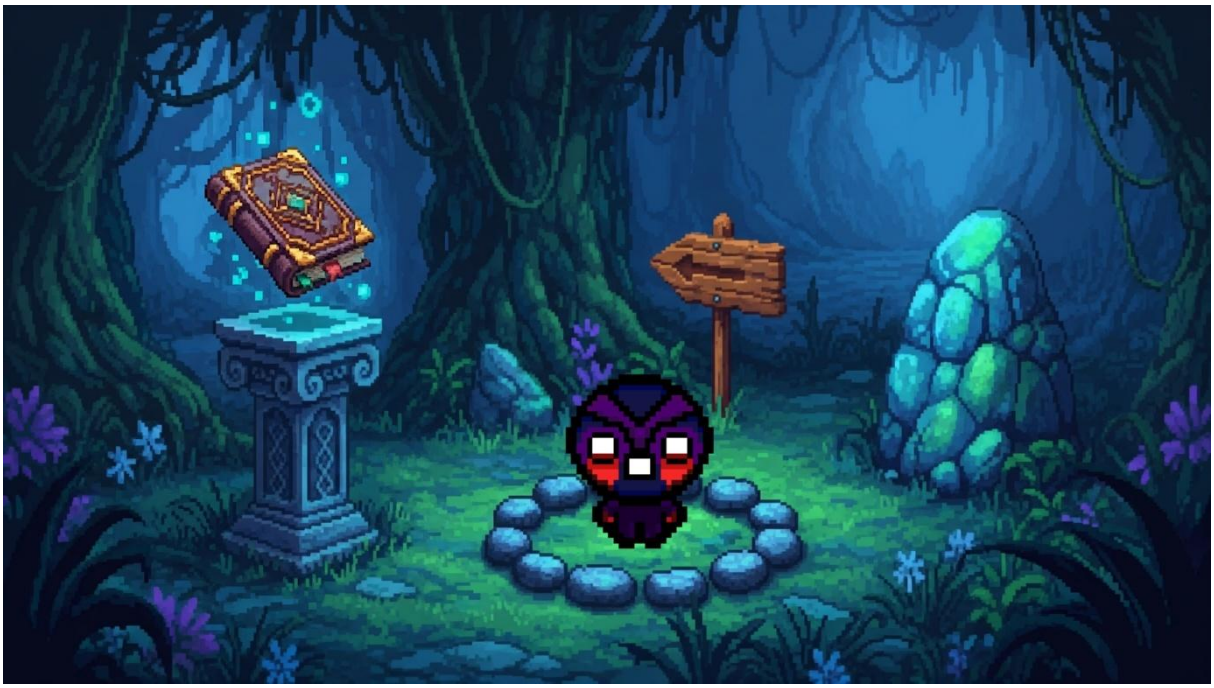
Po podaniu właściwych danych bądź utworzeniu nowego konta, gracz zostaje przeniesiony do głównego menu programu. Z tego poziomu użytkownik może wybrać jedną z dostępnych opcji, klikając na odpowiadający napis lewym przyciskiem myszy:

1) Nowa gra
2) Reguły gry
3) Kolekcja
4) Najwyższe wyniki
5) Wyjście

Rys. 2.5. Menu główne programu

1) Nowa gra – wybór tej opcji przenosi gracza do menu selekcji postaci.

W menu wyświetlany jest wygląd aktualnie przeglądanego bohatera. Najechanie kursorem na poszczególne elementy menu wyświetli odpowiednie komunikaty.
Skierowanie kursora na książkę widoczną na rys. 2.6. wyświetli bazowe statystki postaci takie jak maksymalne zdrowie, zadawane obrażenia, szybkość poruszania się, częstotliwość strzałów, czy prędkość pocisków.
Gracz może przeglądać dostępne postacie, używając strzałki widocznej na środku ekranu. Istnieje również opcja powrotu do menu głównego gry – wystarczy kliknąć kamień po prawej stronie.
Po dokonaniu wyboru wystarczy nacisnąć lewym przyciskiem myszy na wybraną postać, aby rozpocząć rozgrywkę.



Rys. 2.6. Selekcja postaci

2) Reguły gry – menu przedstawia szczegółowy opis rozgrywki, przeciwników, postaci oraz przedmiotów.

Gracz przemieszcza się po menu klikając przyciski widoczne na ekranie. Pierwsze strony zawierają słowny opis rozgrywki, natomiast kolejne strony przedstawiają szczegółowy opis każdej postaci, każdego przeciwnika oraz każdego przedmiotu.

Rys. 2.7. Zasady rozgrywki – opis rozgrywki

Rys. 2.8. Zasady rozgrywki – przykładowa strona z opisem bohatera

3) Odblokowane przedmioty – menu zawiera wgląd do odblokowanych przez użytkownika przedmiotów.

Jeśli użytkownik odblokował dany przedmiot, na odpowiedniej stronie zostanie wyświetlona jego tekstura, nazwa oraz sposób działania (rys. 2.9.) W przypadku przedmiotów zablokowanych, użytkownik zobaczy jedynie ich nazwę i opis, natomiast tekstura będzie zasłonięta paskiem z informacją o zablokowaniu. (rys. 2.10.) Pod spodem pojawi się wskazówka, w jaki sposób można odblokować dany przedmiot.

Rys. 2.9. Kolekcja – strona z odblokowanym przedmiotem

Rys. 2.10. Kolekcja – strona z zablokowanym przedmiotem

4) Najwyższe wyniki – menu wyświetla najlepsze wyniki uzyskane przez graczy.

Użytkownicy są posortowani względem swoich wyników – od najwyższego do najniższego. Jeżeli liczba użytkowników w bazie danych przekracza możliwości wyświetlania nazw użytkowników na danej stronie, w dolnej części ekranu wyświetlą się strzałki pozwalające na przeglądanie kolejnych stron z wynikami.

Rys. 2.11. Najwyższe wyniki graczy - strona 1

Rys. 2.12. Najwyższe wyniki graczy - strona 2

2.2. Rozgrywka

Po wybraniu postaci, gracz zostaje przeniesiony do ekranu gry, gdzie będzie mógł rywalizować o uzyskanie jak najlepszego wyniku. Gracz porusza się, po pomieszczeniu o określonych wymiarach, przy pomocy klawiszy WSAD, natomiast strzela za pomocą strzałek. Nad głową gracza oraz przeciwników wyświetlają się paski zdrowia. Na ekranie widoczne są aktualne statystyki gracza, wynik gracza oraz numer aktualnej fali.

Rys. 2.13. Ekran rozgrywki

Pokonani przeciwnicy mają szansę upuszczenie przedmiotu, domyślnie odblokowanego dla gracza. Przedmioty mogą m.in. losowo zwiększać lub zmniejszać statystyki gracza.

Rys. 2.14. Ekran rozgrywki – przedmioty upuszczone przez przeciwników

Podniesienie jakiegokolwiek przedmiotu skutkuje wyświetleniem informacji o zmianie odpowiedniej statystyki – pokazana zostaje wartość, o jaką dana statystyka została zwiększona lub zmniejszona.

Rys. 2.15. Ekran rozgrywki – zmiana wartości poszczególnych statystyk postaci

Po oczyszczeniu obszaru ze wszystkich przeciwników, następuje kilkusekundowa przerwa, po której generowana jest nowa fala przeciwników.

Rys. 2.16. Ekran rozgrywki – odliczenie czasu do rozpoczęcia następnej fali

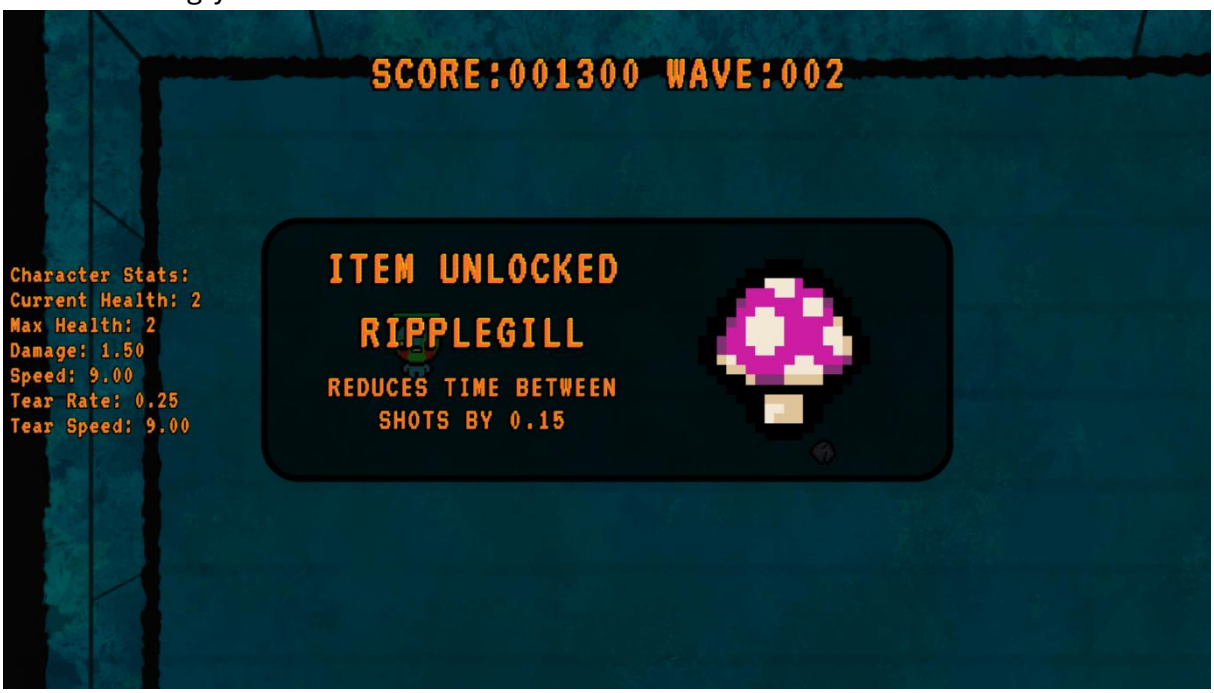
Co kilka fal pojawiają się znacznie potężniejsze jednostki, zwane bossami, wyróżniające się zwiększoną liczbą zdrowia lub unikalnymi sposobami ataku.

Rys. 2.17. Ekran rozgrywki - fala z bossem

Pokonanie bossa nagradza gracza specjalnym przedmiotem, zwiększającym wybraną statystykę postaci.

Rys. 2.18. Ekran rozgrywki – przedmiot upuszczony przez bossa

Po podniesieniu takiego przedmiotu na ekranie zostanie wyświetlona informacja, w formie analogicznej do tej, zaprezentowanej na rysunku 2.15.
Zgodnie z informacjami dostępnymi w ramach menu kolekcji bądź menu zasad rozgrywki, użytkownik może odblokować nowe przedmioty, wykonując odpowiednie akcje.
Po spełnieniu wymagań związanych z danym przedmiotem, na ekranie pojawi się komunikat potwierdzający jego odblokowanie. Przedmiot ten od razu staje się dostępny w ramach rozgrywki.



Rys. 2.19. Ekran rozgrywki – informacja o odblokowanym przedmiocie

Jeżeli gracz utraci wszystkie punkty zdrowia, ekran rozgrywki zatrzyma się, elementy interfejsu przestaną być widoczne, a na ekranie wyświetli się plansza informująca o zakończeniu rozgrywki. Plansza przedstawia liczbę zdobytych przez gracza punktów. Jeżeli jest ona większa od dotychczasowej, to na planszy pojawia się stosowna informacja. Plansza zawiera również przycisk, którego wciśnięcie powoduje powrót do menu głównego.

Rys. 2.20. Plansza po zakończeniu rozgrywki – wariant z nowym rekordem

Rys. 2.21. Plansza po zakończeniu rozgrywki – wariant bez nowego rekordu

Jeśli gracz uzyska nowy najlepszy wynik, tabela rekordów zostanie zaktualizowana, a jego pozycja w rankingu odpowiednio podniesiona.

Rys. 2.22. Menu najwyższych wyników – zaktualizowana tabela

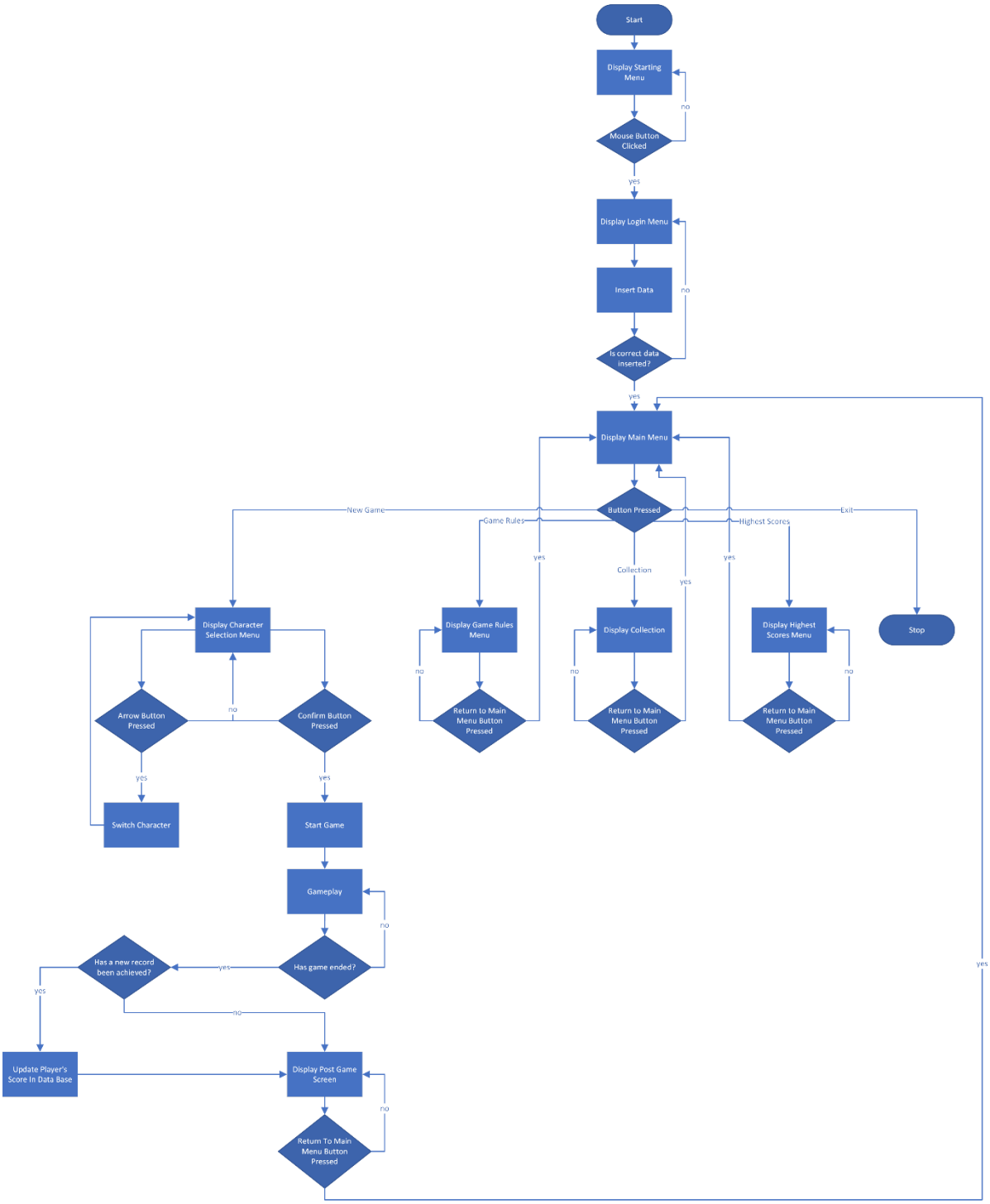
W przypadku odblokowania nowego przedmiotu w trakcie rozgrywki (rys. 2.19.), po jej zakończeniu przedmiot ten zostanie oznaczony jako odblokowany w kolekcji.

Rys. 2.23. Kolekcja – nowo odblokowany przedmiot

3. Specyfikacja wewnętrzna:

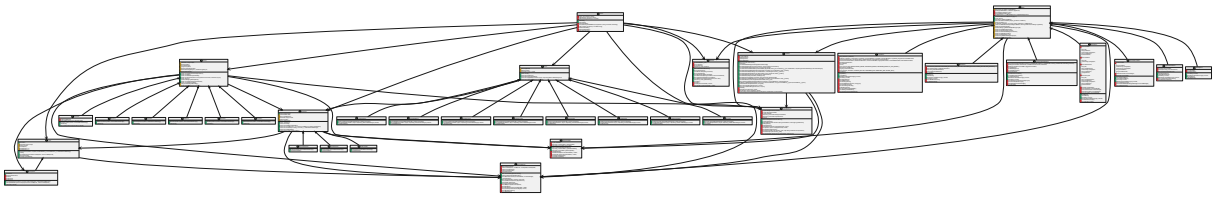
Ze względu na złożoność projektu do sprawozdania dołączono oddzielne pliki zawierające diagram klas i diagram aktywności. Dodatkowo z powodu ograniczeń edytora używanego do utworzenia diagramu klas, diagramy klas *Game, Character*, *Enemy, ScreenSettings* nie zawierają wszystkich metod i pól – pominięto przede wszystkim standardowe metody typu *get* lub *set*, które nie wnoszą istotnych informacji do ogólnego obrazu struktury klas.

3.1. Diagram czynności w notacji UML



Rys. 3.1. Diagram aktywności w notacji UML

3.2. Diagram hierarchii klas w notacji UML



Rys. 3.2. Diagram klas w notacji UML

3.3. Opis metod oraz pól poszczególnych klas

Poniżej znajduje się opis najważniejszych metod oraz pól każdej z klas.

1) Klasa czysto wirtualna Menu

Zmienne:

static unique_ptr<Menu> selectedMenu – pole przechowujące wskaźnik na wybrane menu,
int pageNumber – pole przechowujące numer aktualnie wyświetlanej strony

Metody:

virtual void Draw() = 0 – metoda odpowiadająca za wyświetlanie wszystkich elementów w ramach danego menu
virtual MenuResult handleMenuLogic()=0 – metoda czysto wirtualna, nadpisywana przez klasy pochodne, odpowiada za logikę poszczególnego menu,
static void setSelectedMenu(unique_ptr<Menu> newMenu) – metoda aktualizująca wskaźnik selectedMenu,
virtual void CheckCollisions() – metoda sprawdzająca, czy został wciśnięty przycisk,
virtual void drawMenuElements() – metoda wyświetlająca przyciski w odpowiednim menu,
virtual void switchPage(int number) – metoda odpowiadająca za zmianę numeru aktualnej strony,
virtual void setAmountOfPages(int number) – metoda ustawiająca maksymalną liczbę stron,
virtual void setButtonVisibility() – metoda ustawiająca widoczność przycisków na ekranie,

1.1) Klasa pochodna StartingMenu

Zmienne:

string titleName – zmienna przechowująca nazwę wyświetlanego tytułu gry,

1.2) Klasa pochodna LoginMenu

Zmienne:

rectangle LoginMenu_ConfirmArea, LoginMenu_UsernameBarArea, LoginMenu_PasswordBarArea, LoginMenu_SingupArea, ReturnButtonArea – zmienne przechowujące obszary kolizji przycisków,

int setAction – zmienna przechowująca aktualnie wybrane działanie np. wpisywanie nazwy użytkownika, czy hasła,

Metody:

bool checkIsLoginCorrect() – metoda sprawdzająca zgodność wpisywanych przez użytkownika danych, z przechowywanymi w bazie danych,

int isButtonClicked() – metoda sprawdzająca, czy i który przycisk został wciśnięty,

bool checkIsPlayerInDataBase() – metoda sprawdzająca, czy użytkownik znajduje się w bazie danych,

void addPlayerToDataBase() – metoda dodająca nowego użytkownika do bazy danych,

1.3) Klasa pochodna MainMenu

Zmienne:

map<string, ButtonData>Buttons – mapa przechowująca informacje o przyciskach dostępnych w menu <nazwa przycisku, obszar kolizji przycisku>,

int setAction – zmienna przechowująca wybrane przez użytkownika działanie,

Metody:

int isButtonClicked() – metoda sprawdzająca, czy i który przycisk został wciśnięty,

1.4) Klasa pochodna CharacterSelectionMenu

Zmienne:

Rectangle SmallerCommentsBar, BiggerCommentsBar – zmienne przechowujące obszary aktywne paska komentarzy;

Metody:

void DrawComments(CommentType type) – metoda wyświetlająca na ekranie odpowiedni komentarz, na podstawie wybranego typu

void chooseExplanationType() – metoda ustawiająca typ komentarza,

void DrawPlayerCharacterImage() – metoda wyświetlająca wizerunek aktualnie przeglądanej postaci,

1.5) Klasa pochodna RulesMenu

Zmienne:

struct ItemsInfo, GameInfo, CharacterInfo, EnemyInfo – struktury zawierające różne zestawy zmiennych potrzebnych do prawidłowego wyświetlenia informacji na ekranie,

vector<GameInfo> GameInfoPages – wektor przechowujący informacje o rozgrywce,

vector<CharacterInfo> CharInfoPages – wektor przechowujący informacje o dostępnych postaciach,

vector<EnemyInfo> EnemyInfoPages – wektor przechowujący informacje o przeciwnikach,

vector<ItemsInfo> ItemsInfoPages – wektor przechowujący informacje o dostępnych przedmiotach,

Metody:

void DrawRules(int page) – metoda wyświetlająca odpowiednie informacje na podstawie aktualnej strony,

void setPagesContent() – metoda wstawiająca do wektorów odpowiednie informacje,

1.6) Klasa pochodna UnlockedItemsMenu

Zmienne:

struct ItemsInfo - struktura zestaw zmiennych potrzebnych do prawidłowego wyświetlenia informacji na ekranie,

vector<ItemsInfo> ItemsInfoPages – wektor przechowujący informacje o przedmiotach

Metody:

void DrawPage(int page) – metoda wyświetlająca informacje na stronie,

void DrawLockedBar(float beginBarHeight, float endBarHeight) – metoda wyświetlająca pasek „Zablokowane” na stronie w przypadku, kiedy przedmiot jest niedostępny,

void setPageContent() – metoda wstawiająca do wektora odpowiednie informacje

1.7) Klasa pochodna HighestScoreMenu

Zmienne:

vector<pair<string, int>>UsersScores – wektor przechowujący nazwy graczy oraz opowiadające im wyniki,

bool areUsersLoadedIntoVector – wartownik sprawdzający, czy wartości z bazy danych zostały wstawione do wektora,

Metody:

void LoadUsersScoresIntoVector() – metoda wczytująca wartości z bazy danych,

void DrawPlayersScores() – metoda wyświetlająca wyniki graczy na ekranie.

2) Klasa Game

Zmienne:

int playerTotalScore – zmienna przechowująca aktualny wynik gracza,
unique_ptr<Character> Player – inteligetny wskaźnik wskazujący na aktualnie wybrany typ postaci,
vector<shared_ptr<Enemy>> enemies – wektor przechowujący wskaźniki na obiekty typu Enemy,
vector<enemyTears> EnemyTears – wektor przechowujący pociski przeciwników,
vector<shared_ptr<Items>> items – wektor przechowujący wskaźniki na obiekty typu Items,
int amountOfEnemies – zmienna przechowująca liczbę przeciwników, zwiększa się przy rozpoczęciu każdej fali,

Metody:

void Draw() – metoda wyświetlająca wszystkie elementy rozgrywki,
void Update() – metoda aktualizująca wszystkie elementy rozgrywki,
void CollisionCheck() – metoda sprawdzająca kolizje zachodzące między graczem, przeciwnikami, przedmiotami i aktualizująca odpowiednie wartości,
void createRandomLoot(Vector2 enemyPos, bool condition) – metoda odpowiadająca za tworzenie losowego przedmiotu,
vector <shared_ptr<Enemy>> CreateEnemy() – metoda tworząca przeciwników w każdej fali,
void DeleteInactiveTears() – metoda usuwająca wszystkie pociski, które nie są aktywne, tzn. wykroczyły po za obszar rozgrywki, trafiły w przeciwnika lub gracza,
void InputHandle() – metoda zarządzająca logiką poruszania się oraz strzelania,
bool updatePlayerInDataBase(int playerScore, string username, bool& flag) – metoda aktualizująca wynik w bazie danych po zakończonej rozgrywce,

3) Klasa wirtualna Character

Zmienne:

vector<Tears> tearsy – wektor przechowujący pociski gracza;

characterStats stats – zmienna typu strukturalnego, przechowuje wszystkie statystyki postaci,

Texture2D image=nullptr – wskaźnik, którego zadaniem jest wskazywać na odpowiednią teksturę załadowaną w ramach klasy LoadingTextures,*

Metody:

virtual void loadOwnStats() – metoda aktualizująca zmienną stats odpowiednimi wartościami, wczytanymi z pliku

virtual void Draw() – metoda wyświetlająca wybraną postać,

virtual void DrawPlayerHealthBar() – metoda wyświetlająca pasek zdrowia nad postacią,

virtual void movePlayer(int x, int y, Vector2 minMapLimit, Vector2 maxMapLimit) – metoda zmieniająca położenia gracza na obszarze gry,

virtual void shootTears(int tearD_X, int tearD_Y, Texture2D& loadedImage) – metoda odpowiadająca za tworzenie pocisków i wstawianie ich do wektora tearsy,

3.1) Klasy pochodne XCharacter*

XCharacter(Texture2D& loadedImage) – konstruktor ustawiający odpowiednie wartości zmiennym typu chronionego zdefiniowanych w ramach klasy bazowej,*

4) Klasa wirtualna Enemy

Zmienne:

Texture2D image=nullptr - wskaźnik, którego zadaniem jest wskazywać na odpowiednią teksturę załadowaną w ramach klasy LoadingTextures,*

enemyStats stats - zmienna typu strukturalnego, przechowuje wszystkie statystyki danego przeciwnika,

Metody:

virtual void Update(Vector2 PlayerPosition) – metoda aktualizująca położenie przeciwnika,

virtual void Draw() - metoda wyświetlająca wybraną postać,

virtual void DrawEnemyHealthBar() - metoda wyświetlająca pasek zdrowia nad przeciwnikiem,

virtual void shootTears(vector<enemyTears>& enemTears, Character player) – odpowiadająca za tworzenie pocisków i wstawianie ich do odpowiedniego wektora,*

virtual void UpdateColl(Vector2 Direction) – metoda aktualizująca położenie przeciwnika w przypadku wykrycia kolizji między dwoma przeciwnikami

virtual void loadEnemyStats() – metoda aktualizująca zmienną stats odpowiednimi wartościami, wczytanymi z pliku,

4.1) Klasy pochodne MonsterX*

MonsterX(Vector2 postion, Texture2D& loadedImage) – konstruktor wczytujący odpowiednie wartości zmiennym typu chronionego zdefiniowanych w klasie bazowej,*

5) Klasa czysto wirtualna Items

Zmienne:

Texture2D image=nullptr - wskaźnik, którego zadaniem jest wskazywać na odpowiednią teksturę załadowaną w ramach klasy LoadingTextures,*

Metody:

virtual void DrawItems() – metoda wyświetlająca przedmiot,

virtual void applyEffect(Character player, map<string,float>& statsChanges)=0 – metoda czysto wirtualna, która w klasach pochodnych wywołuje efekt, za który klasa jest odpowiedzialna,*

virtual void setPosition(Vector2 enemyPos) – metoda ustawiająca pozycję przedmiotu, w taki sposób, aby znajdował się w miejscu zniszczenia przeciwnika,

6) Klasa wirtualna Tears

Zmienne:

bool active – zmienna przechowująca stan pocisku;

Metody:

void UpdatePosition(Vector2 minMapLimit, Vector2 maxMapLimit) – metoda aktualizująca pozycję pocisku,
virtual void Draw() – metoda wyświetlająca pocisk,

6.1) Klasa pochodna EnemyTears

Metody:

void UpdatePosition(Vector2 Playerpos, Vector2 minMapLimit, Vector2 maxMapLimit) – metoda aktualizująca pozycję pocisku w oparciu o położenie gracza

7) Klasa CharactersData – klasa oparta o wzorzec Singleton

Zmienne:

map<string, characterStats> CharacterStats – mapa przechowująca statystyki postaci o danej nazwie,
map<string, enemyStats> EnemyStats – mapa przechowująca statystyki przeciwnika o danej nazwie,

Metody:

void LoadCharacterStats() – metoda ładująca statystyki wszystkich postaci do mapy,
void LoadEnemyStats() – metoda ładująca statystyki wszystkich przeciwników do mapy,

8) Klasa UserInfo – klasa oparta o wzorzec Singleton

Zmienne:

string username – zmienna przechowująca nazwę użytkownika,
string password – zmienna przechowująca hasło użytkownika,
map<string, bool> UserItems – zmienna przechowująca informacje o tym, czy przy gracz posiada odblokowane przedmioty,

Metody:

void updateUserItems(string itemName, bool value) – metoda, która aktualizuje wartości odblokowania przedmiotu
bool getUserItemValue(string itemName) – metoda zwracająca wartość posiadania przedmiotu,

9) Klasa LoadingTextures – klasa oparta o wzorzec Singleton

Zmienne:

map<string, Texture2D> BackgroundTextures – mapa przechowująca nazwy tekstur oraz samą teksturę tła,
map<string, Texture2D> ObjectTextures - mapa przechowująca nazwy tekstur oraz samą teksturę obiektów,

Metody:

void loadTextures(fs::path directory_path, map<string,Texture2D>& mapTextures) – metoda wczytująca tekstury do mapy,
Texture2D& passCorrectTexture(string textureName, textureType typeOfTexture) – metoda przekazująca referencje do odpowiedniej tekstury,
void unloadTextures(map<string, Texture2D>& mapTextures) – metoda rozładowująca tekstury, po wyłączeniu programu,

10) Klasa ScreenSettings – klasa oparta o wzorzec Singleton

Zmienne:

Vector2 resolutionFactor – zmienna przechowująca mnożnik względem bazowej rozdzielczości,
Camera2D camera – zmienna przechowująca informacje o kamerze stosowanej w trakcie rozgrywki,

Metody:

Vector2 getScreenResolutionFactor() – metoda przekazująca mnożnik, aby móc edytować rozmiar poszczególnych napisów, tekstur, itp.
void ToggleFullScreenWindow() – metoda zmieniająca tryb, w którym jest uruchomiony program,
void updateCamera(Vector2 playerPosition) – metoda zmieniająca położenie kamery na podstawie pozycji gracza,
void setMapLimit(Vector2& mapLimit) – metoda ustawiająca limity obszaru, po którym może poruszać się gracz w trakcie rozgrywki,

11) Klasa GameUI – klasa oparta o wzorzec Singleton

Zmienne:

Color MyOrange – zmienna przechowująca wartości RGB specjalnego koloru pomarańczowego,

Color MyYellow - zmienna przechowująca wartości RGB specjalnego koloru żółtego,

Metody:

void DrawTextWithOutline(const string& text, Vector2 position, float fontSize) – metoda odpowiedzialna za tworzenie i wyświetlanie napisu w kolorystyce pomarańczowo-żółtej,

void DrawBlackBar(Rectangle border, unsigned char opacity) – metoda wyświetlająca czarne tło o podanej przejrzystości o podanym obszarze,

void DrawTextOnBar(Rectangle bar, float fontSize, const string& text, float y_position) – metoda wyświetlająca wyśrodkowany tekst na określonym obszarze

string ConvertToString(float number, int prec) – metoda zamieniająca liczbę typu float na string wraz z uwzględnieniem precyzji,

void DrawCharacterStatsInGame(characterStats playerStats, float x_pos, float starting_y_pos, float fontSize, map<string,float>& statsChange, bool drawChanges) – metoda wyświetlająca statystyki gracza w grze. Metoda wyświetla również zmianę statystyk, jeżeli takie się pojawiły,

**- w miejscu znaku „X” znajduje się odpowiednia nazwa/wartość, która reprezentuje osobny typ klasy pochodnej*

4. Testowanie

W trakcie implementacji program był wielokrotnie testowany pod kątem poprawności działania. Na etapie tworzenia projektu wyeliminowano wiele różnych błędów. Najpoważniejsze błędy dotyczyły głównie niewłaściwego zarządzania pamięcią. Przykładowo wykryto błąd, polegający na wielokrotnym zwalnianiu tej samej tekstury.

5. Podsumowanie

Projekt został napisany w celu zrealizowania gry o wysokiej regrywalności, osiągniętej poprzez elementy losowości oraz system odblokowywania nowych przedmiotów.

W przyszłości projekt może zostać rozbudowany o nowe funkcjonalności, m.in. nowe przedmioty, nowych przeciwników, czy nowych bohaterów o różnych zdolnościach i statystykach.