Spis treści

Informacje o grupie

Grupa:

Jakub Wasiczek

Michał Filip

Podział zadań

Jakub Wasiczek odpowiadał w **backendzie** za klasy:

1. Player
2. Equipment
3. Localization
4. Item
5. MyError

W GUI miał za zadanie stworzyć:

1. LocationSelectionWindow
2. PlayerEQWindow
3. CharacterCreation
4. System zapisu postaci
5. System wczytywania postaci

Oraz za sporządzenie Dokumentacji Technicznej.

Michał Filip odpowiadał w **backendzie** za klasy:

1. Enemy
2. Boss

W GUI miał za zadanie stworzyć:

1. MainWindow
2. DifficultyWindow
3. FightWindow

Backend

Klasa Player:

1. Pola:

- nickname: Przechowuje pseudonim gracza.
- hp: Reprezentuje punkty zdrowia gracza.
- dmg: Reprezentuje punkty obrażeń gracza.
- def: Reprezentuje punkty obrony gracza.
- eq_limit: Reprezentuje limit wagi ekwipunku, który gracz może nosić.
- playerClass: Enum HeroType wskazujący klasę postaci gracza (Wojownik, Łotr, Mag).
- key_count: Reprezentuje liczbę kluczy, jakie posiada gracz.
- bagpack: Instancja klasy Equipment reprezentująca plecak gracza.

2. Konstruktory:

Dwa konstruktory: konstruktor domyślny i konstruktor parametryczny, który przyjmuje nazwę gracza i typ postaci (HeroType). Konstruktor domyślny ustawia początkowe wartości.

3. Metoda character_creation:

Metoda odpowiadająca za stworzenie postaci gracza. Gracz podaje swoją nazwę, wybiera klasę postaci spośród dostępnych (Wojownik, Łotr, Mag), a następnie tworzona jest nowa instancja gracza z odpowiednimi statystykami.

4. Metoda HeroAttack:

Metoda pozwalająca zadawać graczowi obrażenia na podstawie losowej liczby.

5. Metoda WeightTest:

Metoda sprawdzająca, czy waga ekwipunku gracza nie przekroczyła limitu. W przypadku przekroczenia, gracz musi usunąć przedmiot z plecaka.

6. Metody HeroBuffEq i HeroLostEq:

Metody do dodawania (zwiększania statystyk) i usuwania (zmniejszania statystyk) efektów ekwipunku gracza w zależności od rodzaju przedmiotu.

Enum **HeroType**:

Enum definiujący dostępne klasy postaci: Wojownik, Łotr, Mag, przypisujący każdej z nich odpowiednią wartość liczbową (1, 2, 3).

Klasa Equipment:

1. Pola:

- bagpack: LinkedList przechowująca przedmioty w ekwipunku.
- bagpack_weight: Reprezentuje wagę całego ekwipunku.

2. Konstruktor:

Konstruktor domyślny ustawiający początkową wartość wagi ekwipunku na 0.

3. Metoda AddItem:

Metoda dodająca przedmiot do ekwipunku. Dodaje przedmiot na początku listy i aktualizuje wagę ekwipunku.

4. Metoda RemoveItem:

Metoda usuwająca przedmiot z ekwipunku. Aktualizuje wagę ekwipunku po usunięciu przedmiotu.

5. Metoda FindItem:

Metoda wyszukująca przedmiot w ekwipunku na podstawie określonego warunku. Wykorzystuje funkcję FirstOrDefault z LINQ.

6. Metoda BackpackContains:

Metoda wyświetlająca zawartość ekwipunku w uporządkowany sposób. Sortuje listę przedmiotów według ich typu.

7. Metoda RemoveItemAt:

Metoda usuwająca przedmiot z ekwipunku na podstawie indeksu. Aktualizuje wagę ekwipunku po usunięciu przedmiotu.

Ogólne informacje:

- Klasa Equipment jest odpowiedzialna za przechowywanie i zarządzanie ekwipunkiem gracza.
- Metoda FindItem umożliwia wyszukiwanie przedmiotów w ekwipunku na podstawie określonych warunków.
- Metoda BackpackContains prezentuje zawartość ekwipunku w formie uporządkowanej listy, informując o typie, wadze i opisie każdego przedmiotu.
- Metoda RemoveItemAt umożliwia usuwanie przedmiotów na podstawie indeksu.

Klasa Localization:

1. Pola:

- amount_of_items: Przechowuje liczbę dostępnych przedmiotów w lokalizacji.
- item: Przechowuje przedmiot dostępny w lokalizacji.
- localization_name: Przechowuje nazwę lokalizacji.
- localization_description: Przechowuje opis lokalizacji.

2. Konstruktor:

Konstruktor domyślny inicjalizuje domyślne wartości pól.

3. Metoda LookAround:

Metoda wirtualna wyświetlająca opis lokalizacji.

4. Metoda TakeItem:

Metoda wirtualna pozwalająca na zabranie przedmiotu z lokalizacji i dodanie go do ekwipunku. Aktualizuje liczbę dostępnych przedmiotów.

Po klasie abstrakcyjnej Lozalization dziedziczy 5 innych klas, każda z nich zawiera własny opis, oraz typy przedmiotów jakie możemy nich znaleźć:

1. Klasa **StartingLocation**
- Gracz rozpoczyna w niej rozgrywkę
- Nie ma w niej przedmiotów
- Nie ma w niej wrogów
2. Klasa **Valley**
3. Klasa **Port**
4. Klasa **Graveyard**
5. Klasa **Gates**
- Metoda OpenTheGates:

Metoda sprawdzająca, czy gracz może otworzyć wrota na podstawie liczby posiadanych kluczy. Jeśli gracz ma mniej niż 3 klucze, wrota nie zostaną otwarte, a metoda zwróci true. W przeciwnym wypadku zwraca false kończącym tym samym pętlę oraz rozgrywkę.

Ogólne informacje:

- Klasa Localization jest klasą bazową, która zawiera podstawowe informacje o lokalizacji, takie jak nazwa, opis, liczba przedmiotów, i dostępny przedmiot.
- Klasy dziedziczące po Localization reprezentują konkretne lokalizacje w grze, takie jak Początek, Dolina Magów, Port, Cmentarzysko, Wrota.

Klasa Item:

1. Pola:

- type: Reprezentuje typ przedmiotu (enum ItemType).
- weight: Przechowuje wagę przedmiotu.
- description: Przechowuje opis przedmiotu.

2. Konstruktory:

- Konstruktor domyślny inicjalizuje domyślne wartości wag i opisu.
- Konstruktor przyjmujący ItemType inicjalizuje przedmiot na podstawie przekazanego rodzaju.

3. Metoda zobacz_przedmiot:

Metoda zwracająca opis przedmiotu.

4. Metoda CompareTo:

Implementacja interfejsu IComparable.

Enum ItemType:

Reprezentuje różne typy przedmiotów, takie jak MagicScepter, Dagger, Health Potion Sword, Health Stone, Armor, Key.

Ogólne informacje:

- Klasa Item reprezentuje przedmiot w grze, zawierając informacje o jego typie, wadze i opisie.
- Wszystkie przedmioty po znalezieniu się w ekwipunku gracza zapewniają mu różne bonusy.

Klasa Enemy:

1. Pola:

- name: Reprezentuje nazwę przeciwnika.
- hp: Przechowuje ilość punktów życia przeciwnika.
- dmg: Przechowuje wartość zadawanego obrażenia przez przeciwnika.
- def: Przechowuje wartość obrony przeciwnika.
- enemyItem: Przechowuje przedmiot, jaki może wypaść z przeciwnika po jego śmierci.
- difficulty: Reprezentuje poziom trudności przeciwnika.

2. Konstruktory:

- Konstruktor przyjmujący podstawowe parametry (imię, punkty życia, obrażenia, obrona, poziom trudności) inicjalizuje te parametry i dostosowuje je w zależności od poziomu trudności.
- Konstruktor przyjmujący dodatkowo przedmiot przeciwnika korzysta z poprzedniego konstruktora i dodaje obsługę przedmiotu.

3. Metoda EnemyAttack:

Metoda symuluje atak przeciwnika, generując losową wartość obrażeń w określonym zakresie.

4. Metoda Clone:

Implementacja interfejsu ICloneable. Pozwala na sklonowanie obiektu przeciwnika.

5. Metoda ChangeName:

Metoda umożliwiająca zmianę nazwy przeciwnika.

Enum Difficulty:

Reprezentuje różne poziomy trudności dla przeciwników.

Klasa Boss:

1. Pole:

- artifact: Przechowuje klucz unikalny dla każdego bossa.

2. Konstruktory:

Konstruktor przyjmujący podstawowe parametry (imię, punkty życia, obrażenia, obrona, poziom trudności) oraz przedmiot bossa. Korzysta z konstruktora klasy bazowej Enemy i dodaje implementację przedmiotu bossa.

Klasa MyError:

1. Właściwość:

ExtraInfo: Właściwość tylko do odczytu, przechowująca dodatkowe informacje dotyczące błędu.

2. Konstruktor:

Konstruktor przyjmuje dwa parametry: message i extrainfo. Parametr message przekazywany jest do konstruktora klasy bazowej Exception, natomiast extrainfo jest przypisywany do właściwości ExtraInfo.

Metody znajdujące się w klasie Program:

```
public static void ZapisJson(string nazwaPliku, Player player)
{
    try
    {
        string jsonstr = JsonSerializer.Serialize(player, typeof(Player));
        File.WriteAllText(nazwaPliku, jsonstr);
        Console.WriteLine("Gra została zapisana do pliku JSON.");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Błąd podczas zapisu do pliku JSON: " + ex.Message);
    }
}
```

Opis:

Metoda służy do zapisu obiektu klasy Player do pliku JSON. W przypadku wystąpienia błędu (np. podczas zapisu do pliku), metoda przechwytuje wyjątek i wyświetla odpowiedni komunikat błędu.

```
public static Player OdczytJson(string nazwaPliku)
{
    try
    {
        if (!File.Exists(nazwaPliku))
        {
            Console.WriteLine("Plik JSON nie istnieje.");
            return null;
        }

        string jsonstr = File.ReadAllText(nazwaPliku);
        return JsonSerializer.Deserialize<Player>(jsonstr);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Błąd podczas odczytu pliku JSON: " + ex.Message);
        return null;
    }
}
```

Opis:

Metoda służy do odczytu obiektu klasy Player z pliku JSON. W przypadku wystąpienia błędu (np. brak pliku, błędny format JSON), metoda przechwytuje wyjątek i wyświetla odpowiedni komunikat błędu, a następnie zwraca null.

```
public static bool Fight(Player Character1, Enemy Enemy1)
{
    int Character_hp = Character1.Hp;
    int Enemy_hp = Enemy1.Hp;

    while (Character_hp > 0 && Enemy_hp > 0)
    {
        Console.WriteLine("Co chcesz zrobić");
        Console.WriteLine("1 - Atakuj");
        Console.WriteLine("2 - Ulecz się");
        Console.WriteLine("3 - Uciekaj");

        int choice = 0;
        try
        {
            if (!int.TryParse(Console.ReadLine(), out choice))
            {
                Console.WriteLine("Musisz podać liczbę od 1-3!");
            }

            if (choice != 1 && choice != 2 && choice != 3)
            {
                throw new Exception("Niepoprawna liczba!");
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine("Błąd: " + ex.Message);
            continue;
        }

        switch (choice)
        {
            case 1:
                int Character_att = Character1.HeroAttack(Character1.Dmg);
                Enemy_hp -= (1 - (Enemy1.Def / 100)) * Character_att;
                Console.WriteLine($"Przeciwnikowi zostało {Enemy_hp} życia\n");
                if (Enemy_hp <= 0)
                {
                    if (Enemy1.EnemyItem is not null)
                    {
                        Console.WriteLine($"Zdobyłeś {Enemy1.EnemyItem.type}\n");
                        Character1.bagpack.AddItem(Enemy1.EnemyItem);
                    }
                    Console.WriteLine("Wygrałeś\n");
                    return true;
                }
                break;
            case 2:
                Item healthPotion = Character1.bagpack.FindItem(item => item.Type == ItemType.HealthPotion);
                if (healthPotion != null)
                {
                    Character_hp = Character1.Hp;
                    Character1.bagpack.RemoveItem(healthPotion);
                    Console.WriteLine($"Wróciłeś do pełni zdrowia!\nTwoje HP: {Character1.Hp}\n");
                }
                else { Console.WriteLine("W twoim ekwipunku nie ma mikstur zdrowia!\n"); }
                    break;
            case 3:
                Console.WriteLine("Uciekasz...\n");
                return false;
        }

        Console.WriteLine($"{Enemy1.Name} atakuje!\n");
        int Enemy_att = Enemy1.EnemyAttack(Enemy1.Dmg);
        Character_hp -= (1 - (Character1.Def / 100)) * Enemy_att;
        Console.WriteLine($"Zostało ci {Character_hp} życia\n");
        if (Character_hp <= 0)
        {
            Console.WriteLine("Przegrałeś\n");
            return false;
        }
```

Opis:

Metoda odpowiada za mechanikę walki pomiędzy graczem (Character1) a przeciwnikiem (Enemy1).

1. W pętli while trwa walka, dopóki zdrowie gracza i przeciwnika są większe niż zero.
2. W każdym przebiegu pętli gracz ma trzy opcje: atakować, uleczyć się lub uciec.
3. W przypadku błędnego wyboru, program zgłasza błąd, wyświetla komunikat i kontynuuje pętlę.
4. Opcje gracza są obsługiwane poprzez switch.
5. Dla opcji ataku (case 1):
    a. Oblicza obrażenia zadane przez gracza przeciwnikowi, uwzględniając obronę przeciwnika.
    b. Aktualizuje zdrowie przeciwnika.
    c. Jeśli przeciwnik umiera, gracz zdobywa ewentualny przedmiot z przeciwnika.
6. Dla opcji uleczenia (case 2):
    a. Sprawdza, czy gracz posiada miksturę zdrowia w ekwipunku.
    b. Jeśli tak, ulecz gracza i usuń miksturę z ekwipunku.
7. Dla opcji ucieczki (case 3):
    a. Zakończ walkę i zwróć false.
8. Po każdej turze przeciwnik atakuje gracza, obliczając obrażenia i aktualizując zdrowie gracza.
9. Po zakończeniu walki, zwraca true, jeśli gracz wygrał, lub false, jeśli przegrał.
10. W przypadku błędnego wprowadzenia danych (np. nieprawidłowy wybór opcji), metoda przechwytuje wyjątek i wyświetla odpowiedni komunikat błędu.

```
0 odwołania
public static void Encounter(Player player, Localization currentLocalization, Difficulty enum_difficulty_level, ref int valley_counter, ref int port_counter, ref int graveyard_counter)
{
    Enemy v1 = new("Zając", 41, 51, 5, enum_difficulty_level);
    Enemy v2 = new("Wilk", 47, 31, 7, enum_difficulty_level);
    Enemy v3 = new("Skrzat", 51, 31, 11, enum_difficulty_level, new Item(ItemType.HealthPotion));
    Boss v4 = new("Leśny golem", 57, 37, 13, enum_difficulty_level, new Item(ItemType.Key));

    Enemy p1 = new("Wsciekła mewa", 57, 41, 17, enum_difficulty_level);
    Enemy p2 = new("Pirat", 61, 47, 19, enum_difficulty_level, new Item(ItemType.HealthPotion));
    Enemy p3 = p2.Clone() as Enemy;
    p3.ChangeName("Korsarz");
    Boss p4 = new("Kraken", 67, 53, 23, enum_difficulty_level, new Item(ItemType.Key));

    Enemy g1 = new("Szkielet", 67, 59, 29, enum_difficulty_level, new Item(ItemType.HealthPotion));
    Enemy g2 = new("Zombie", 71, 61, 29, enum_difficulty_level);
    Enemy g3 = new("Duch złodzieja", 89, 67, 31, enum_difficulty_level);
    Boss g4 = new("Nekromanta", 101, 73, 43, enum_difficulty_level, new Item(ItemType.Key));
```

Opis:

Metoda odpowiada za spotkania gracza z wrogiem, w zależności od lokalizacji.

1. Tworzy przeciwników (Enemy i Boss) zdefiniowanych dla danej lokalizacji.
2. W zależności od lokalizacji i liczby przeciwników pokonanych wcześniej, inicjuje walkę z odpowiednim przeciwnikiem.

```
bool won = false;
Console.Clear();
if (currentLocalization is Valley)
{
    if (valley_counter == 1) { Console.WriteLine($"Walczysz z {v1.Name}\n"); won = Fight(player, v1); valley_counter += won ? 1 : 0; return; }
    if (valley_counter == 2) { Console.WriteLine($"Walczysz z {v2.Name}\n"); won = Fight(player, v2); valley_counter += won ? 1 : 0; return; }
    if (valley_counter == 3) { Console.WriteLine($"Walczysz z {v3.Name}\n"); won = Fight(player, v3); valley_counter += won ? 1 : 0; return; }
    if (valley_counter == 4)
    {
        Console.WriteLine($"Walczysz z {v4.Name}\n"); won = Fight(player, v4); valley_counter += won ? 1 : 0;
        player.Key_count += won ? 1 : 0; if (won == true) Console.WriteLine("Znalazłeś klucz!\n"); return;
    }
    if (valley_counter == 5) { Console.WriteLine("Pokonałeś już wszystkich przeciwników w tej lokalizacji\n"); return; }
}
if (currentLocalization is Port)
{
    if (port_counter == 1) { Console.WriteLine($"Walczysz z {p1.Name}\n"); won = Fight(player, p1); port_counter += won ? 1 : 0; return; }
    if (port_counter == 2) { Console.WriteLine($"Walczysz z {p2.Name}\n"); won = Fight(player, p2); port_counter += won ? 1 : 0; return; }
    if (port_counter == 3) { Console.WriteLine($"Walczysz z {p3.Name}\n"); won = Fight(player, p3); port_counter += won ? 1 : 0; return; }
    if (port_counter == 4)
    {
        Console.WriteLine($"Walczysz z {p4.Name}\n"); won = Fight(player, p4); port_counter += won ? 1 : 0;
        player.Key_count += won ? 1 : 0; if (won == true) Console.WriteLine("Znalazłeś klucz!\n"); return;
    }
    if (port_counter == 5) { Console.WriteLine("Pokonałeś już wszystkich przeciwników w tej lokalizacji\n"); return; }
}
if (currentLocalization is Graveyard)
{
    if (graveyard_counter == 1) { Console.WriteLine($"Walczysz z {g1.Name}\n"); won = Fight(player, g1); graveyard_counter += won ? 1 : 0; return; }
    if (graveyard_counter == 2) { Console.WriteLine($"Walczysz z {g2.Name}\n"); won = Fight(player, g2); graveyard_counter += won ? 1 : 0; return; }
    if (graveyard_counter == 3) { Console.WriteLine($"Walczysz z {g3.Name}\n"); won = Fight(player, g3); graveyard_counter += won ? 1 : 0; return; }
    if (graveyard_counter == 4)
    {
        Console.WriteLine($"Walczysz z {g4.Name}\n"); won = Fight(player, g4); graveyard_counter += won ? 1 : 0;
        player.Key_count += won ? 1 : 0; if (won == true) Console.WriteLine("Znalazłeś klucz!\n"); return;
    }
    if (graveyard_counter == 5) { Console.WriteLine("Pokonałeś już wszystkich przeciwników w tej lokalizacji\n"); return; }
}
```

3. Po każdej walce aktualizuje licznik pokonanych przeciwników i, jeśli przeciwnik jest bossem, dodaje klucz do ekwipunku gracza.
4. Wykorzystuje funkcję Fight do przeprowadzenia walki.
5. Informuje gracza o rezultatach walki oraz ewentualnym zdobyciu klucza.

Metoda **start()** wywoływana w funkcji main():

inicjuje rozpoczęcie gry, pozwalając graczowi:

1. wybrać poziom trudności,

```
int difficulty_level = 0;
Difficulty enum_difficulty_level = Difficulty.easy;

bool valid_input_difficulty = false;
do
{
    try
    {
        if (!int.TryParse(Console.ReadLine(), out difficulty_level))
        {
            Console.WriteLine("Musisz podać liczbę od 1-3!");
        }

        if (difficulty_level != 1 && difficulty_level != 2 && difficulty_level != 3)
        {
            throw new Exception("Niepoprawna liczba!");
        }

        switch (difficulty_level)
        {
            case 1:
                enum_difficulty_level = Difficulty.easy;
                valid_input_difficulty = true;
                break;
            case 2:
                enum_difficulty_level = Difficulty.hard;
                valid_input_difficulty = true;
                break;
            case 3:
                enum_difficulty_level = Difficulty.master;
                valid_input_difficulty = true;
                break;
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Błąd: " + ex.Message);
    }
} while (!valid_input_difficulty);
```

2. tworzyć postać wywołując metodę character_creation z klasy Player,

```
Player player = new Player();
Player new_player = player.character_creation();
if(new_player == null)
{
    Console.WriteLine("Coś poszło nie tak przy tworzeniu postaci!");
    return;
}
```

3. przechodzić między lokalizacjami,

```
Localization currentLocalization = new StartingLocation();


static Localization ChooseNewLocalization(Localization currentLocalization)
{
    Console.WriteLine("Wybierz nową lokalizację:");
    Console.WriteLine("1 - Dolina Magów [Zagrożenie - niskie]");
    Console.WriteLine("2 - Port [Zagrożenie - średnie]");
    Console.WriteLine("3 - Cmentarz [Zagrożenie - wysokie]");
    Console.WriteLine("4 - Wrota [???]");

    int choice = 0;
    if (int.TryParse(Console.ReadLine(), out choice))
    {
        switch (choice)
        {
            case 1:
                Console.WriteLine("Wyruszyłeś do Doliny Magów!");
                return new Valley();
            case 2:
                Console.WriteLine("Wyruszyłeś do Portu!");
                return new Port();
            case 3:
                Console.WriteLine("Wyruszyłeś w kierunku cmentarza..");
                return new Graveyard();
            case 4:
                Console.WriteLine("Wyruszyłeś w kiernku gigantycznych wrót..");
                return new Gates();
            default:
                Console.WriteLine("Niepoprawny wybór. Pozostajesz w obecnej lokalizacji.");
                return currentLocalization;
        }
    }
    else
    {
        Console.WriteLine("Niepoprawny wybór. Pozostajesz w obecnej lokalizacji.");
        return currentLocalization;
    }

}
```

4. wykonywać różne akcje,

```
switch (intTemp)
{
    case 1:
        currentLocalization.LookAround();
        break;
    case 2:
        if (currentLocalization.amount_of_items != 0 && currentLocalization.item != null)
        {
            Console.WriteLine("Znalazłeś: " + currentLocalization.item.type);
            new_player.HeroBuffEq(currentLocalization.item);
            currentLocalization.TakeItem(currentLocalization.item,new_player.bagpack);
        }
        else { Console.WriteLine("Nic nie znalazłeś."); }
        break;
    case 3:
        new_player.WeightTest();
        currentLocalization = ChooseNewLocalization(currentLocalization);
        break;
    case 4:
        if (currentLocalization is StartingLocation)
        {
            Console.WriteLine("Jesteś na beztroskiej polanie.\nNic ci tutaj nie grozi.\n");
        }
        else
        {
            Encounter(new_player, currentLocalization, enum_difficulty_level, ref valley_counter, ref port_counter, ref graveyard_counter);
        }
        break;
    case 5:
        new_player.bagpack.BackpackContains();
        break;
    case 6:
        Console.Clear();
        Console.WriteLine($"Nazwa gracza: {new_player.Nickname}\nLimit ekwipunku: {new_player.Eq_limit}\nDMG: " +
            $"{new_player.Dmg}\nDEF: {new_player.Def}\nHP: {new_player.Hp}");
        break;
    case 7:
        save_name = new_player.Nickname;
        save_name = save_name + ".json";
        ZapisJson(save_name, new_player);
        break;
    case 8:
        Console.WriteLine("Podaj nazwe postaci, którą chcesz wczytać:\n");
        save_name = Console.ReadLine();
        Console.WriteLine("Wczytywanie postaci..\n");
        save_name += ".json";
        new_player = OdczytJson(save_name) ?? new_player;
        break;
```

5. prowadzić rozgrywkę do momentu spełnienia warunku zwycięstwa.

```
bool winCondition = true;
while (winCondition)
{
```

```
if (currentLocalization is Gates)
{
    Gates end = new Gates();
    winCondition = end.OpenTheGates(new_player);

}
```

GUI

MainWindow

1. Pola klasy:

- player: Pole przechowujące obiekt klasy Player, reprezentującego gracza w grze.
- currentLocalization: Pole przechowujące obiekt klasy Localization, reprezentującego aktualną lokalizację gracza w grze.
- valley_counter, port_counter, graveyard_counter: Pola statyczne przechowujące liczniki przeciwników pokonanych w poszczególnych lokalizacjach.
- difficulty: Pole przechowujące obiekt klasy Difficulty, reprezentujący poziom trudności gry.
- Pola reprezentujące przeciwników (v1, v2, ..., g4) w różnych lokalizacjach.

2. Metody:

- Btn_save_click: Obsługuje zapis postaci do pliku JSON.
- Btn_load_Click: Obsługuje wczytywanie postaci z pliku JSON.
- Btn_Go: Obsługuje przycisk „Nowa Lokalizacja”, umożliwiający przemieszczanie się między lokalizacjami.
- BtnFind_Click: Obsługuje przycisk "Przeszukaj lokalizacje", umożliwiający znalezienie przedmiotów w aktualnej lokalizacji.
- BtnCheckStats_Click: Obsługuje przycisk "Sprawdź staty", wyświetlający statystyki postaci.
- BtnCheckBagPack_Click: Obsługuje przycisk "Sprawdź plecak", wyświetlający zawartość plecaka gracza.
- BtnDifficulty_Click: Obsługuje przycisk "Ustaw poziom trudności", umożliwiający ustawienie poziomu trudności gry.
- UpdateEnemyDifficulty: Aktualizuje poziom trudności przeciwników.
- BtnFight_Click: Obsługuje przycisk "Szukaj guza", rozpoczynający walkę z przeciwnikami w danej lokalizacji.
- BtnCreation_Click: Obsługuje przycisk "Stwórz postać", umożliwiający utworzenie nowej postaci.

3. Konstruktor:

Inicjalizuje obiekty reprezentujące przeciwników w różnych lokalizacjach.

LocationSelectionWindow

1. Właściwość:

SelectedLocalization: Właściwość przechowująca wybraną lokalizację.

2. Metody:

- LocationSelectionWindow(): Konstruktor klasy, tworzy obiekty reprezentujące różne lokalizacje (Valley, Port, Graveyard), a następnie wyświetla ich opisy w odpowiednich polach tekstowych na oknie.
- Valley_Click: Obsługuje kliknięcie przycisku "Dolina" i ustawia SelectedLocalization na obiekt reprezentujący Dolinę, po czym zamyka okno z wynikiem true.
- Port_Click: Obsługuje kliknięcie przycisku "Port" i ustawia SelectedLocalization na obiekt reprezentujący Port, po czym zamyka okno z wynikiem true.
- Cementary_Click: Obsługuje kliknięcie przycisku "Cmentarz" i ustawia SelectedLocalization na obiekt reprezentujący Cmentarz, po czym zamyka okno z wynikiem true.

DifficultyWindow

1. Pole:

Difficulty: Pole przechowujące aktualny poziom trudności, które może być modyfikowane przez to okno.

2. Metody:

- DifficultyWindow(Difficulty difficulty): Konstruktor klasy, inicjalizuje komponenty okna oraz ustawia wartość pola Difficulty na przekazany poziom trudności.
- BtnEasy_Click: Obsługuje kliknięcie przycisku "Łatwy" i ustawia Difficulty na łatwy, po czym wyświetla stosowny komunikat i zamyka okno.
- BtnHard_Click: Obsługuje kliknięcie przycisku "Trudny" i ustawia Difficulty na trudny, po czym wyświetla stosowny komunikat i zamyka okno.
- BtnMaster_Click: Obsługuje kliknięcie przycisku "Mistrzowski" i ustawia Difficulty na mistrzowski, po czym wyświetla stosowny komunikat i zamyka okno.

CharacterCreation

1. Pole:

player: Pole przechowujące obiekt klasy Player, reprezentującego gracza, którego wybiera użytkownik.

2. Pole tekstowe i wzory dla imienia gracza:

- pattern: Wzór używany do sprawdzania poprawności wprowadzonego imienia gracza. Imię musi składać się z liter alfabetu, bez cyfr ani znaków specjalnych.
- PlayerName: Pole tekstowe, w którym gracz wprowadza imię postaci.

3. Metody:

- CharacterCreation(): Konstruktor klasy, inicjalizuje komponenty okna oraz tworzy trzech różnych bohaterów (wojownika, maga, łotra) i wyświetla ich statystyki na formularzu.
- BWarrior_Click, BMage_Click, BRogue_Click: Obsługują kliknięcia przycisków wyboru postaci (Wojownika, Maga, Łotra). Sprawdzają poprawność wprowadzonego imienia, tworzą obiekt klasy Player z wybranym typem bohatera i przypisanym imieniem, a następnie ustawiają to pole na utworzonego gracza. DialogResult zostaje ustawione na true w przypadku poprawnie wprowadzonego imienia, co umożliwia zamknięcie okna.

FightWindow

1. Pola klasy:
- player: Pole przechowujące obiekt klasy Player
- enemy: Pole przechowujące obiekt klasy Enemy
- Won: Pole logiczne określające, czy gracz wygrał walkę.
2. Pola lokalne:
- characterHp: Pole przechowujące aktualne punkty życia gracza.
- enemyHp: Pole przechowujące aktualne punkty życia przeciwnika.
3. Metody:
- UpdateUI(): Metoda aktualizująca interfejs użytkownika, wyświetlająca informacje o aktualnych punktach życia gracza i przeciwnika.
- BtnAttack_Click: Obsługuje kliknięcie przycisku "Atak". Wywołuje atak gracza na przeciwnika, a następnie odpowiedź przeciwnika na gracza. Aktualizuje UI i sprawdza, czy walka została zakończona.
- BtnHeal_Click: Obsługuje kliknięcie przycisku odpowiadającego za leczenie. Sprawdza, czy gracz ma miksturę zdrowia w ekwipunku, a następnie przywraca mu pełne punkty życia i usuwa miksturę.
- BtnEscape_Click: Obsługuje kliknięcie przycisku "Ucieczka". Za pomocą tego przycisku gracz wychodzi z walki.

PlayerEQWindow

1. Pole klasy:

player: Pole przechowujące obiekt klasy Player, reprezentującego gracza, którego ekwipunek jest wyświetlany w oknie.

2. Metody:

- PlayerEQWindow(Player player): Konstruktor klasy, inicjalizuje komponenty okna oraz przekazuje obiekt gracza, którego ekwipunek ma być wyświetlony. Sortuje listę przedmiotów w ekwipunku i dodaje je do ItemList w interfejsie użytkownika.
- BtnDelete_Click: Obsługuje kliknięcie przycisku "Usuń". Usuwa zaznaczony przedmiot z ekwipunku gracza i odświeża listę w interfejsie.
- BtnInfo_Click: Obsługuje kliknięcie przycisku "Opis” Wyświetla okno z opisem i wagą zaznaczonego przedmiotu.
- BtnEscEQ_Click: Obsługuje kliknięcie przycisku "Wyjdź". Zamyka okno ekwipunku.