

Państwowa Wyższa Szkoła Informatyki i Przedsiębiorczości w Łomży

Kierunek Informatyka

Pracownia specjalistyczna, prowadzący Janusz Rafałko

Projekt Zespołowy II

Specjalność: Systemy oprogramowania

Studia stacjonarne I stopnia semestr VI

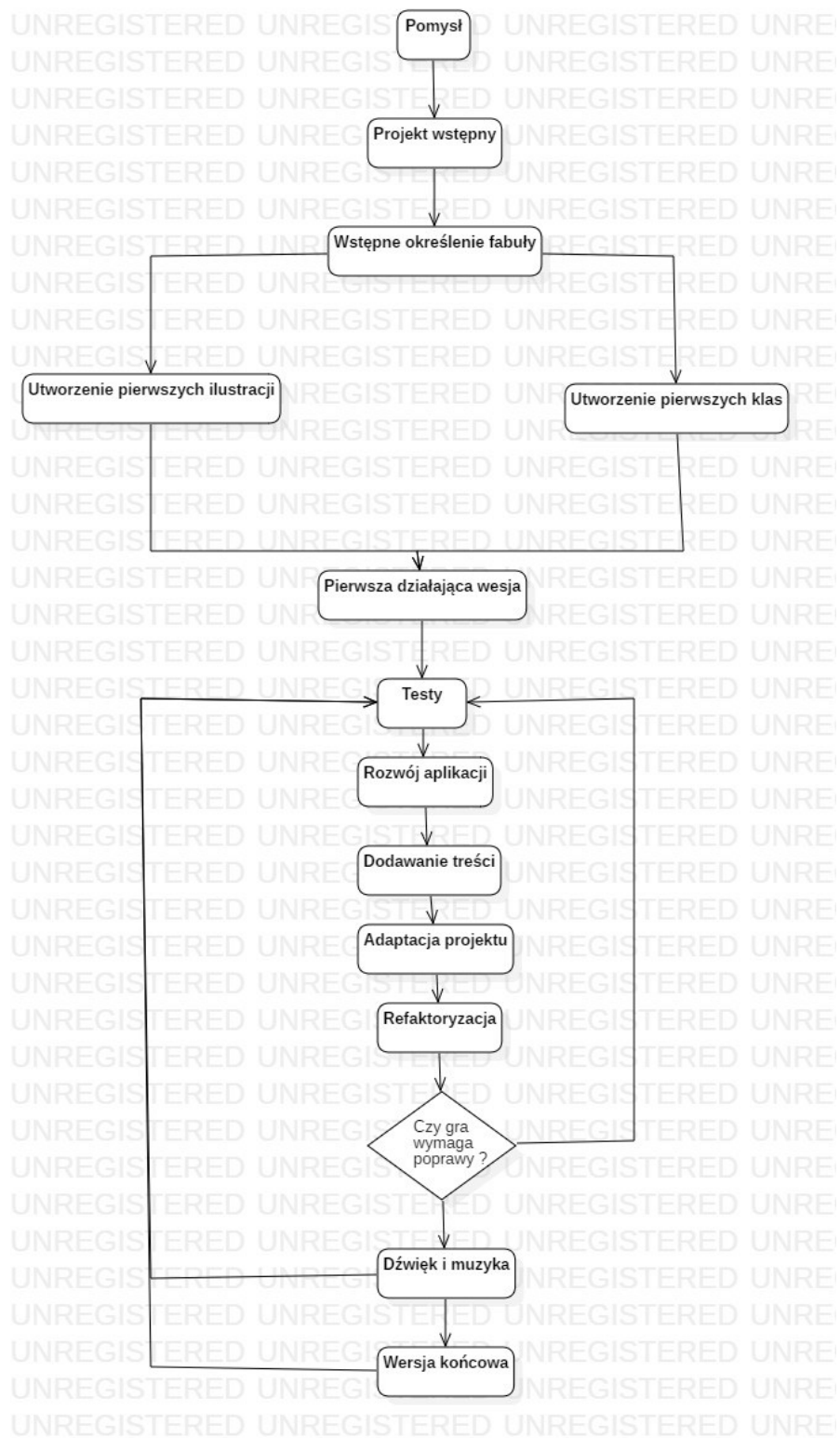


Autorzy: Krzysztof Sobieski, Maciej Surawski, Piotr Ołtarzewski, Sebastian Majewski, Edyta Zakrzewska

Temat: Gra komputerowa

1. Architektura

Architektura aplikacji napisanej w języku obiektowym. Przed przystąpieniem do implementacji powinny istnieć jasne zarysy końcowego systemu, abstrahując od tego czy będzie to gra komputerowa, program użytkowy czy aplikacja biznesowa. Architektura aplikacji zaprojektowana jest jednak w ten sposób, aby każda z klas stanowiła oddzielny byt odpowiedzialny za własną część logiki programu. Dzięki temu luźnemu podejściu do kwestii ustrukturalizowania obiektów, dowolny moduł może korzystać z usług innego, wyłącznie poprzez globalne odwołanie się do jego metod, co znacznie ogranicza ilość zależności w samym kodzie jak i niebywale zwiększa jego czytelność



Rysunek 2. Architektura aplikacji

2. Układ katalogów

Assets - w tym folderze mieszczą się pliki i foldery odpowiedzialne za zawartość w grze.

Animations - mieszczą się tu wszystkie pliki niezbędne do prawidłowego odtworzenia animacji gracza oraz napisów końcowych w grze.

Prefabs - miejsce gdzie są przechowywane wszystkie gotowe obiekty z gry, które są wielokrotnie wykorzystywane.

Scenes - miejsce składowania wszystkich ekranów używanych w czasie rozgrywki jak i głównej klasy aplikacji.

Scripts - miejsce składowania wszystkich skryptów odpowiedzialnych za działanie gry.

Sprites - miejsce składowania wszystkich grafik występujących w grze. Występują one w formacie .png.

TextMeshPro - Zawiera pliki gotowego assetu, który służy polepszeniu wyglądu menu.

Texts - zawiera pliki czcionek do tekstów w menu i intro.

3. Implementacja

Gry wideo, w przeciwieństwie do większości aplikacji użytkowych, są tworem multimedialnym wciągającymi swych użytkowników w świat rozbudowanych animacji, obrazów i dźwięków. Wszystkie te elementy działają w skomplikowany sposób, aby mieć lepszy obraz na to jak działa gra oraz jej elementy poniżej zostaną zaprezentowane kody źródłowe przykładowych klas, grafiki ich krótki opis.

Poniżej przedstawiono klasy odpowiadające za działanie menu, opcji oraz wyświetlanie i działanie napisów początkowych oraz końcowych.

```
public class IntroManager : MonoBehaviour
{
    public float speed = 15;
    float timer;

    void Update()
    {
        timer += Time.deltaTime;
        Vector3 position = transform.position;
        Vector3 localUp = transform.TransformDirection(0, 1, 0);
        position += localUp * speed * Time.deltaTime;
        transform.position = position;
        if (timer > 85)
        {
            SceneManager.LoadScene("TestingScen");
        }
        if(Input.GetMouseButtonDown(0) && timer > 2)
        {
            SceneManager.LoadScene("TestingScen");}}}

```

Listing 1. Klasa IntroManager

Klasa IntroManager - odpowiada głównie za przewijanie się napisów w intrze. Posiada również timer aby określić maksymalny czas wyświetlania sceny

```
public class MenuUIManager : MonoBehaviour
{
    public AudioSource scare;
    public void Play()
    {
        SceneManager.LoadScene("Intro");
    }
    public void Options()
    {
        SceneManager.LoadScene("Options");
    }
    public void Quit()
    {
        Application.Quit();
    }
    public void ToMenu()
    {
        StartCoroutine(licznik());
    }
    IEnumerator licznik()
    {
        scare.volume = 1;
        scare.Play();
        yield return new WaitForSeconds(1);
        SceneManager.LoadScene("Menu");
    }
}
```

Listing 2.Klasa MenuUIManager

Klasa MenuUIManager - zawiera metody wykorzystywane w menu głównym do przechodzenia do scen menu i intra oraz wyjścia z gry.

W grze są dostępne także trzy inne klasy, które zostały krótko opisane, lecz ich kod nie znalazł się w dokumentacji są to :

CreditsManager - zawiera metodę, która pozwala pominąć napisy końcowe.

GameSettings - zawiera pola dla możliwych opcji.

SettingsManager - odpowiada za kontrolę opcji dostępnych w grze: pełny ekran, rozdzielczość i głośność. Dzięki tej klasie możemy zapisać plik konfiguracji na swoim komputerze co oznacza, że raz zapisane ustawienia będą wczytywane przy kolejnych uruchomieniach gry aż do ich ponownej zmiany.

Poniżej przedstawiono foldery klasy odpowiadające za działanie gry, interakcji, przeciwników, gracza oraz UI.

Folder Klas PlayerScripts - zawiera klasy odpowiadające za sterowanie, statystyki i umiejętności gracza. Zawiera ona w sobie takie klasy jak: AbilityScripts, AttackScripts, Player, PlayerStats.

Poniżej zostanie przedstawiony kawałek kodu z klasy Player który odpowiada za podstawowe ataki gracza. W przeciwieństwie do poprzednio przedstawionych klas, które odpowiadają za Menu, Intro oraz Outro, te klasy są bardziej rozbudowane i bardziej skomplikowane.

```
private void PlayerAttack()
{
    if ((Input.GetMouseButton(0) && timer > TimeBetweenBullets && !
EventSystem.current.IsPointerOverGameObject()) || (Input.GetMouseButtonDown(0) &&
timer > TimeBetweenBullets && !EventSystem.current.IsPointerOverGameObject()))
    {
        timer = 0f;
        var pos = Input.mousePosition;
        pos.z = transform.position.z - Camera.main.transform.position.z;
        pos = Camera.main.ScreenToWorldPoint(pos);
        var rotation = Quaternion.FromToRotation(Vector3.up, pos - transform.p
osition);
        var bullet = (GameObject)Instantiate(
            playerStats.SpellList[0],
            transform.position,
            rotation);
        bullet.transform.position = this.transform.position;
        bullet.GetComponent<Rigidbody2D>().velocity = bullet.transform.up * 8;
        Destroy(bullet, 2.0f);}}}
```

Listing 3. Wycinek kodu z klasy Player

Folder Klas EnemiesScripts - zawiera klasy odpowiadające za zachowania i umiejętności przeciwników. Zawiera on w sobie takie klasy jak: EnemyAttack, EnemyScript, MageBossScript, Mele, PurpleBlast, Range.

Poniżej zostanie przedstawiony kawałek kodu z klasy Mele. Odpowiada on za poruszanie się przeciwnika.

```
void Update()
{
    timer += Time.deltaTime;
    if (Vector2.Distance(transform.position, target.position) > 1)
    {
        transform.position = Vector2.MoveTowards(transform.position, target.positi
on, speed * Time.deltaTime);
        if (timer > 15f)
        {
            timer2 += Time.deltaTime;
            if (timer2 > 5f)
            {
                timer = 0f; }
            transform.position = Vector2.MoveTowards(transform.position, target
.position, speed * 4 * Time.deltaTime);
        }
    }
    else
    {
        transform.position = this.transform.position;}}}
```

Listing 4. Wycinek kodu z klasy Mele

Folder Klas InventoryScripts - zawiera klasy odpowiadające za działanie ekwipunku. Zawiera on w sobie takie klasy jak: BagScript, InventoryScript, Loot, LootBoxScript, LootTable, SlotScript.

Poniżej zostanie przedstawiony kawałek kodu z klasy InventoryScript. Odpowiada on za sprawdzanie czy w plecaku jest miejsce dla przedmiotu.

```
private bool PlaceInEmpty(Item item)
{
    foreach (Bag bag in bags)
    {
        if(bag.bagScript.AddItem(item))
        {
            OnItemCountChanged(item);
            return true;
        }
    }
    return false;}
```

Listing 5.Wycinek kodu z klasy InventoryScript

Folder Klas Shop - zawiera klasy odpowiadające za działanie sklepu i elementów UI powiązanych sklepem. Zawiera on w sobie takie klasy jak: Shop, ShopButton, ShopItem, ShopWindows.

Poniżej zostanie przedstawiony kawałek kodu z klasy ShopWindow. Odpowiada on za tworzenie listy przedmiotów. Dla każdej listy page przypada lista przedmiotów.

```
public void CreatePages(ShopItem[] items)
{pages.Clear();
    List<ShopItem> page = new List<ShopItem>();
    for (int i = 0; i < items.Length; i++)
    {
        page.Add(items[i]);
        if (page.Count == 12 || i == items.Length - 1)
        {
            pages.Add(page);
            page = new List<ShopItem>();}}
    AddItems();}
```

Listing 6.Wycinek kodu z klasy ShopWindow

Metoda AddItems korzysta z CreatePages i tworzy strony z przedmiotami dostępnymi w sklepie.

```
public void AddItems()
{
    pageNumber.text = pageIndex + 1 + "/" + pages.Count;
    prevButton.SetActive(pageIndex > 0);
    nextButton.SetActive(pages.Count > 1 && pageIndex < pages.Count - 1);
    if (pages.Count > 0)
    {
        for (int i = 0; i < pages[pageIndex].Count; i++)
        {
            if (pages[pageIndex][i] != null)
            {
                shopButton[i].AddItem(pages[pageIndex][i]);
            }
        }
    }
}
```

Listing 7.Wycinek kodu z klasy ShopWindow

Klasa GameManager - odpowiada za mechaniki gry i zarządzanie przeciwnikami. Poniżej zostanie przedstawiony kawałek kodu z klasy GameManager. Odpowiada on za tworzenie lochu, po którym porusza się gracz.

```

public void MakeDungeon()
{
    int i = 1;
    List<GameObject> CurrentLevelList = new List<GameObject>();
    for (int j = 1; j <= LevelSize; j++)
    {
        int newLevel = Random.Range(0, LevelListNormal.Count);
        CurrentLevelList.Add(LevelListNormal[newLevel]);
    }
    int newLevelBoss = Random.Range(0, LevelListBoss.Count);
    CurrentLevelList.Add(LevelListBoss[newLevelBoss]);
    foreach (GameObject map in CurrentLevelList)
    {
        var room = (GameObject)Instantiate(
            map,
            new Vector3(i * 20, 0, 0),
            this.gameObject.transform.rotation);

        room.GetComponentInChildren<TeleportBackward>().backward = LevelTeleportList[i - 1];
        room.GetComponentInChildren<TeleportBackward>().currentLevel = i;
        room.GetComponentInChildren<TeleportForward>().forward = LevelTeleportList[i + 1];
        room.GetComponentInChildren<TeleportForward>().currentLevel = i;
        room.transform.position = (Vector2)LevelTeleportList[i];
        i++;
        maps.Add(room);
    }
}

```

Listing 8. Wycinek kodu z klasy GameManager

Folder Klas UIScripts - zawiera klasy odpowiadające za działanie pozostałych elementów UI. Zawiera on w sobie takie klasy jak: ActionButton, BagButton, HandScript, LootButtonScript, lootWindowScript, ObservableStack, StackHealth, UIManager.

Poniżej zostanie zaprezentowany kawałek kodu z klasy UIManager odpowiedzialny za aktualizowanie ilości przedmiotów.

```

public void UpdateStackSize(IClicable clickable)
{
    //jesli stack jest wiekszy niz 1 to pokazuje i ustawia ulosc stakow
    if(clickable.ThisCount>1)
    {
        clickable.ThisStackText.text = clickable.ThisCount.ToString();
        clickable.ThisStackText.color = Color.white;
        clickable.ThisIcon.color = Color.white;
    }
    //jesli stack jest mniejszy niz 2 ukrywa stack size
    else
    {
        clickable.ThisStackText.color = new Color(0, 0, 0, 0);
        clickable.ThisIcon.color = Color.white;}
    //jesli 0 ukrywa ikone
    if(clickable.ThisCount==0)
    {
        clickable.ThisIcon.color = new Color(0, 0, 0, 0);
        clickable.ThisStackText.color = new Color(0, 0, 0, 0);}}

```

Listing 8. Wycinek kodu z klasy UIManager

W grze są dostępne także inne foldery klasy, które zostały krótko opisane , lecz ich kod nie znalazł się w dokumentacji. Są to:

Folder Interfaces - zawiera interfejsy określające interakcje z obiektami. Zawiera on w sobie takie klasy jak: IAddOperation, IBaseOperation, IListOperation, IOperationFactory, IremoveOperation, ISearchOperation.

Folder Klas MapScripts - zawiera klasy odpowiadające za działanie map. Zawiera on w sobie takie klasy jak: BoosRoom, LobbyTeleport, Map, TeleportBackward, TeleportForward.

Folder Klas QuestScripts - zawiera klasy odpowiadające za działanie zadań i elementów UI powiązanych z questami. QGivenScript, Quest, QuestGiver, QuestLog, QuestScript, QuestWindow.

Klasa SoundManager - klasa ta odpowiada za dźwięki występujące w grze.

Poniżej przedstawiono przykładowe grafiki użyte w grze.

Grafika głównego bohatera oraz jego kolejne klatki które zostały użyte do animacji ruchu oraz ataku głównego bohatera. Do animacji użyto wbudowanego w Unity animatora. Postać została stworzona w programie Asprite .



Rysunek 3. Klatki ruchu głównego bohatera

Grafika lobby oraz elementów parę elementów wchodzących w jej skład. Do stworzenia jej wykorzystano program Asprite. Przedstawione zostały: podłóże lobby, namioty (sklep, czarny rynek, i namiot od zadań), dodatkowe elementy wizualne które polepszają odbiór wizualny lobby.



Rysunek 4. Przykładowe grafiki z gry

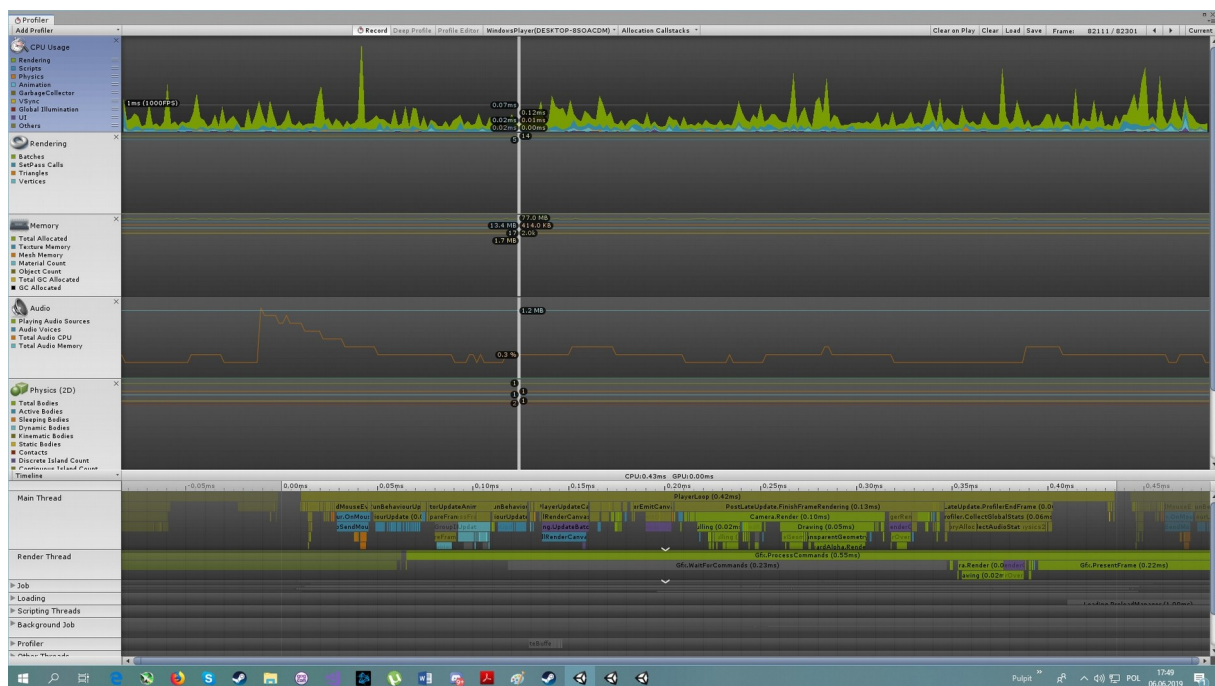
4. Testy

- Działanie i wydajność

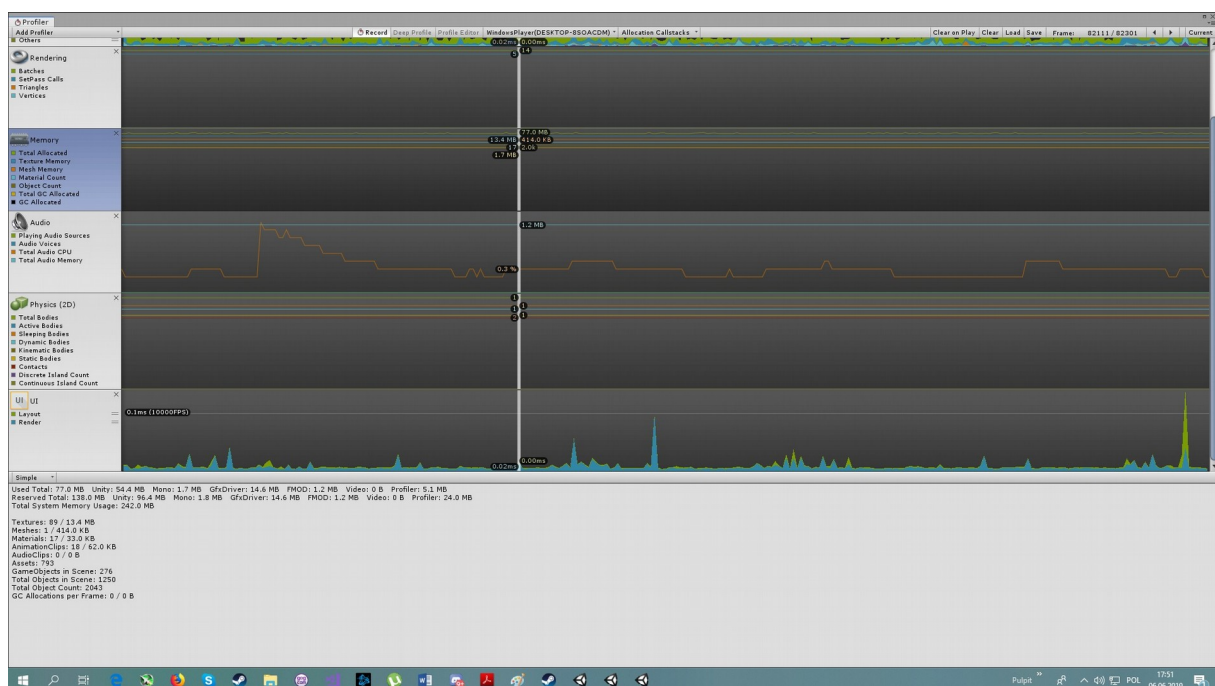
Przedstawiony zrzut pokazuje zużycie karty graficznej, pamięci, procesora przez naszą grę w wysokiej rozdzielczości. Widać że gra jest mało wymagająca i nie potrzebuje dużej ilości zasobów do pracy.

Menedżer zadań					
Plik Opcje Widok					
Procesy Wydajność Historia aplikacji Uruchamianie Użytkownicy Szczegóły Usługi					
Nazwa	Stan	Procesor...	Pamięć	Dysk	Sieć
> Game (2)		12%	70%	0%	0%
> GitHubDesktop (3)		3,2%	75,4 MB	0,1 MB/s	0 Mb/s
System		0%	103,4 MB	0,1 MB/s	0 Mb/s
> Google Chrome (34)		0,6%	0,1 MB	0,1 MB/s	0 Mb/s
> Host usługi: Wstępne ład...		0%	1 389,1 MB	0,1 MB/s	0 Mb/s
> Antimalware Service Exe...		0%	1,1 MB	0 MB/s	0 Mb/s
Registry		0%	100,9 MB	0 MB/s	0 Mb/s
> Host usługi: Dziennik zd...		0%	0,8 MB	0 MB/s	0 Mb/s
> Host usługi: Informacje ...		0%	5,3 MB	0 MB/s	0 Mb/s
> Host usługi: Usługi krypt...		0%	0,6 MB	0 MB/s	0 Mb/s
> Host usługi: program ur...		0%	1,8 MB	0 MB/s	0 Mb/s
> Host usługi: Usługa profi...		0%	8,1 MB	0 MB/s	0 Mb/s
> Host usługi: Usługa klaw...		0%	1,0 MB	0 MB/s	0 Mb/s
Menedżer zadań		0%	0,5 MB	0 MB/s	0 Mb/s
> Aplikacja logowania syst...		1,5%	16,5 MB	0 MB/s	0 Mb/s
Menedżer okien pulpitu		0%	0,7 MB	0 MB/s	0 Mb/s
> Host usługi: Usług syste...		2,0%	17,5 MB	0 MB/s	0 Mb/s
		0%	0,6 MB	0 MB/s	0 Mb/s

Rysunek 5. Zużycie procesów przez grę



Rysunek 6. Zrzut z Profiler Unity



Rysunek 7. Zrzut z Profiler Unity

Do dokładniejszego zbadania użyto narzędzia Profiler które jest w zawarte w unity. Dzięki niemu mamy szczegółowe informacje o tym jak działa nasza gra. Jeśli gra ma problemy takie jak niska szybkość odtwarzania lub wysokie zużycie pamięci, Profiler może nam pokazać co powoduje problem i pomóc nam je naprawić.

Gra działa bardzo dobrze oraz płynnie. Podczas testowania gry nie zauważono żeby gra zatrzymywała swe działanie z nieznanego powodu.

- Poruszanie

Poruszanie w grze działa tak jak było zamierzone. Wszystkie animacje działają prawidłowo. Nie zauważono żadnych nieprawidłowości związanych z poruszaniem się głównego bohatera czy przeciwników. Wszystkie skrypty działają prawidłowo.

- Interakcje z obiektami oraz postaciami

Interakcje z obiektami oraz postaciami działają tak jak było zamierzone. Nie zauważono żadnych nieprawidłowości związanych z interakcjami. Skrypty działają prawidłowo.

- Walka

Walka z przeciwnikami działa tak jak było zamierzone. Nie zauważono żadnych nieprawidłowości związanych z walką. Skrypty działają prawidłowo.

- Generowanie lochu

Skrypt generujący lochy działał bardzo dobrze. Nie zauważono błędów z wygenerowaniem nowego lochu. Generowanie odbywało się w bardzo szybkim tempie co nie wpłynęło na negatywnie na rozgrywkę.

5. Instrukcja użytkowania

- Sterowanie

W - ruch do góry

S - ruch do dołu

A – ruch w lewo

D - ruch w prawo

LPM – strzał, kupno przedmiotu, przenoszenie ekwipunku

PPM – Interakcja

C – karta postaci

E – przejście do następnego lochu

1, 2 ,3 ,4 – szybkie użycie

- Menu



Rysunek 8. Wygląd Meu

Zrzut przedstawiający wygląd menu. Mamy trzy możliwości:

1. Rozpocząć grę za pomocą „Graj”,
2. Wejście w opcje za pomocą przycisku „Opcje”, dzięki czemu będziemy mieli możliwość dostosowania gry pod swoje potrzeby.



Rysunek 8. Wygląd Opcji

Zrzut przedstawiający wybór opcji w menu głównym. W opcjach mamy możliwość zmienienia rozdzielczości ekranu , ustawienia opcji czy pełnego ekranu , mamy także możliwość zmiany głośności dźwięku w grze.

3. Zamknięcie gry

- Lobby



Rysunek 9. Lobby gry

Zrzut przedstawiający wygląd początkowej lokacji w której gracz spędzi część swojego czasu.

Mamy możliwość odwiedzenia :

- I. Dwóch sklepów (czerwony oraz niebieski namiot) – możemy w nich zakupić przedmioty oraz ulepszenia do naszej postaci.
- II. Przyjęcia różnych zadań, w tym zadanie na zwrot pieniędzy (fioletowy namiot)
- III. Loch – Miejsce gdzie gracz będzie walczył z potworami oraz zdobywał złoto i przedmioty.

Na zrzucie można też zauważyć:

- I. Pasek zdrowia informujący gracza o aktualnym stanie zdrowia.
- II. Pasek szybkich akcji, w którym możemy umieścić umiejętności specjalne oraz przedmioty
- III. Aktualny stan posiadanych pieniędzy
- IV. Plecaki, w których przechowujemy potrzebne przedmioty i ulepszenia.

- Namioty



Rysunek 10. Wygląd Sklepu

Zrzut przedstawia sklep z przedmiotami. Mamy możliwość kupna dodatkowego plecaka oraz mikstury leczącej gracza.



Rysunek 11. Wygląd Czarnego rynku

Zrzut przedstawia sklep gdzie możemy kupić ulepszenia do postaci. Mamy możliwość kupna ulepszenia zwiększającego szybkość ataku, zwiększenie obrażeń, zwiększenie maksymalnego poziomu zdrowia i zwiększenie szybkości poruszania się .



Rysunek 12. Wygląd namiotu Zadań

Zrzut przedstawia namiot gdzie możemy oddać i przyjąć różne zadania. Dostępne też jest zadanie na zwrot określonej ilości pieniędzy dzięki któremu będziemy mogli ukończyć grę.

- Loch



Rysunek 13. Wygląd lochu i walki

Zrzut przedstawiający walkę z przeciwnikami w lochu. Przeciwnicy posiadają własne paski zdrowia które są widoczne nad nimi. Po zadaniu im obrażeń przez gracza ich paski maleją, gdy dojdą do zera znikają.



Rysunek 14. Wygląd nagrody za pokonanie przeciwników

Zrzut przedstawiający nagrodę za oczyszczenie pokoju z przeciwników. Nagrody są generowane losowo. Widać także otwarty plecak w którym są przechowywane przedmioty, umiejętności i ulepszenia.

6. Podsumowanie

Cele niezrealizowane:

- Brak możliwości sprzedawania przedmiotów
- Czarny rynek nie pojawia się co pewien okres, nie sprzedaje on karty wyższych jakości po niższych cenach. Jest on dostępny zawsze i posiada tylko ulepszenia i umiejętności.
- Złoto i przedmioty nie wypadają z przeciwników. Przedmioty oraz złoto pojawiają się po pokonaniu wszystkich przeciwników.

Cele zrealizowane:

- Wszystkie prócz niezrealizowanych

Wnioski

Pomimo faktu, że prace nad projektem rozpoczęły się w marcu 2019 roku i planowane były na okres prawie trzech miesięcy, w momencie pisania tych słów (*czerwiec tego samego roku*) gra nadal jest udoskonalana i poszerzana o nowe możliwości i wątki fabularne. Dzieje się tak, gdyż tworzenie gier potrafi być czynnością niezwykle wciągającą i nierzadko o wiele bardziej atrakcyjną od samego grania, a pokusa udoskonalania swojego dzieła często bywa zbyt

wielka, aby podjąć decyzję o wyznaczeniu ostatecznej daty premiery. Bywa niekiedy, że apetyt rośnie w trakcie jedzenia, a potrzeba perfekcjonizmu góruje nad zdrowym rozsądkiem. Wymagania odnośnie nas oraz naszej pracy powinny zostać ustalone znacznie wcześniej niż było to w naszym przypadku. Najlepiej zrobić to w trakcie opracowywania koncepcji gry i mocno się tego trzymać, pomimo że z czasem okazuje się, że możliwe jest znacznie więcej niż było to z początku zakładane. Jeżeli nie określi się w porę swoich oczekiwań, nie sprecyzuje odbiorcy końcowego i zamierzonego efektu, cały projekt może zapaść się w nicość pod naporem naszych własnych ambicji.

Praca w zespole przebiegała bardzo dobrze. Każdy wywiązywał się z powierzonych mu zadań. Pomimo dzielącej odległości każdego członka zespołu nie napotkaliśmy problemów z komunikacją. Bardzo dużą rolę w komunikacji, synchronizacji oraz współdzieleniu plików odegrały dwie strony: GitHub oraz Trello, a także Discord, który umożliwił lepszą komunikację.

Dalszy rozwój

Obecnie gra jest niemal kompletna pod względem programistycznym. Niemniej jednak, niektóre elementy gry jak np. fabuła, grafika, dźwięki, UI wymagają jeszcze doszlifowania i poświęcenia im nieco więcej czasu. Planowane jest też dodanie nowych przeciwników, zadań i lokacji, dzięki którym gra będzie wyglądała lepiej i miała więcej zawartości. Kolejnym celem jest jak najszybsze ukończenie projektu, aby późnym latem roku 2019 projekt mógł zostać uznany za zakończony.