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. Spis treści

Spis treści

2. Wstęp

Tematem projektu została gra komputerowa, ponieważ członkowie grupy mają doświadczenie związane z tworzeniem gier komputerowych.

Celem projektu jest stworzenie gry komputerowej. Opierać się ona będzie na silniku Unity z wykorzystaniem języka c#. Planujemy stworzenie gry należącej do gatunku Dungeon Crawler/Roguelike w rzucie izometrycznym 2D. Zawierać ona będzie elementy grafiki własnej, efekty dźwiękowe oraz skrypty.

3. Opis biznesowy projektu

Celem projektu jest stworzenie gry komputerowej z gatunku Dungeon Crawler/Roguelike. Gracz będzie miał możliwość wielokrotnego przechodzenia gry na różne sposoby. Ze względu na tematykę gry jest ona skierowana do osób powyżej 16 roku życia. Przeznaczona będzie na komputery osobiste z systemem Windows. Wymagania potrzebne do uruchomienia gry będą niewielkie, dzięki czemu będzie można uruchomić ją również na starszych komputerach.

Głównym celem będzie zebranie waluty potrzebnych do spłacenia długu. Odbywać się to będzie poprzez przemierzenie lochów i walkę z przeciwnikami za pomocą kart o różnych właściwościach, które zdobyć będzie można podczas gry oraz przez wymianę przedmiotów. Z poziomu na poziom gra umożliwi ulepszanie przedmiotów. W przerwie pomiędzy poziomami gracz znajduje się w mieście (lobby), w którym może dokonywać ulepszeń przedmiotów oraz dokonywać zakupów.

Gra będzie dystrybuowana cyfrowo za pomocą platformy Steam.

4. Założenia

- Rozgrywka trwa 14 dni podczas której musisz spłacić dług, każdego dnia dług rośnie o naliczone odsetki
- W grze udajemy się na poszukiwania waluty do lochów, każde wejście do lochu zabiera 1 dzień
- Jak zbierzesz odpowiednią sumę waluty to wygrywasz, w przeciwnym wypadku przegrywasz
- Możliwość ulepszania postaci oraz umiejętności
- W grze będzie istniała waluta, za pomocą której możemy kupować przedmioty, uzbieranie określonej wartości umożliwi ukończenie gry
- Z potworów będzie wypadała waluta i przedmioty, które można wymienić na walutę
- Gra będzie posiadała muzykę oraz dźwięki
- W grze będzie występowało miasto, które będzie działało jak lobby. Będziemy mogli w nim wykonać następujące czynności:

  - odwiedzić sklep, w którym będziemy mogli zakupić i sprzedać przedmioty

  - udać się do lochu

  - spłacić haracz, czyli wydać określoną kwotę żeby ukończyć grę

  - wykonać różne zadania, które pomogą w spłaceniu długu

- miasto będzie posiadało czarny rynek, który będzie się pojawiał co jakiś okres – sprzedawał będzie on karty wyższych jakości po niższych cenach niż w sklepie.

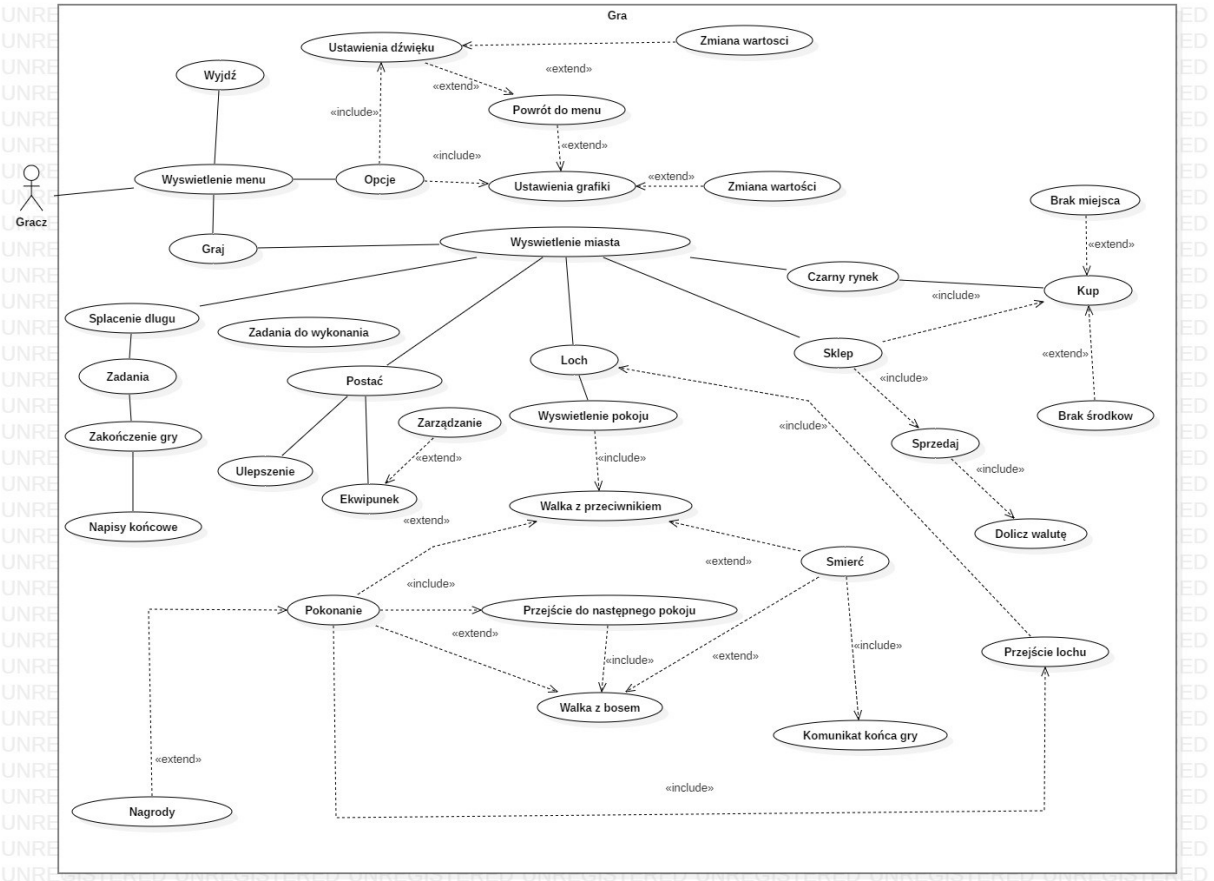
- Gra będzie opierała się na kartach o różnych właściwościach (pasywne, użytkowe dające różne efekty)
- Gracz zaczyna grę z podstawowymi umiejętnościami (po jednej umiejętności każdego typu)
- Ze skrzynek wypadają karty i przedmioty, które można sprzedać w sklepie
- Loch będzie posiadał wiele pokoi które będziemy przemierzać
- Boss na koniec lochu
- Postać będzie posiadała ekwipunek
- Gra będzie posiadać fabułę

5. Funkcje

- Gra ma zapewniać rozrywkę graczowi.

6. Wymagania funkcjonalne

Na poniższym rysunku przedstawiono zbiór przypadków użycia gry dla gracza za pomocą diagramu UML



Rysunek 1. Diagram przypadków użycia

6.1 Wymagania funkcjonalne gry przedstawione w postaci przypadków użycia.

Tabela 1. Przypadki użycia dla Gracza.

Nazwa	Opis	Odpowiedź systemu
Wyświetlanie menu	Gracz widzi menu, z którego może wybrać następujące opcje : graj, opcje, wyjdź	Wyświetlenie odpowiedniej sceny oraz reakcja na wybrane opcje. W zależności od wybranej opcji następuje przejście do niej.
Przycisk Graj	Użytkownik klika graj dzięki czemu rozpoczyna nową grę	Przejście z menu do „lobby" czyli miasta. Załadowanie dodatkowych obiektów do interakcji.
Przycisk Wyjdź	Użytkownik klika wyjdź przez co zamyka grę.	Zakończenie działania gry.
Przycisk Opcje	Wyświetlenie dodatkowych opcji służących do zmiany dźwięku oraz grafiki	Przejście do nowej sceny. Załadowanie opcji, które umożliwią zmianę głośności oraz grafiki
Loch	Użytkownik klika na loch. Wyświetlony zostaje pokój gdzie będzie walczyć z przeciwnikiem. Po pokonaniu przeciwnika gracz przenosi się do kolejnego. Po ukończeniu wszystkich gracz przenosi się do miasta	Przejście do nowej sceny. Załadowanie i wyświetlenie przeciwników oraz innych obiektów. Odblokowanie kolejnych interakcji. Powtarzanie wyświetlania kolejnych pokoi aż do ostatniego. Po przejściu ostatniego powrót do lobby.
Walka z przeciwnikiem	Użytkownik walczy z przeciwnikiem. Oboje mają swoje statystyki. Mogą zadawać sobie obrażenia . Gracz używa kart do ranienia przeciwnika. Wygrania umożliwia dalsze zwiedzanie lochu, śmierć kończy grę.	Liczenie zadanych obrażeń oraz monitorowanie zdrowia. Sprawdzanie warunku zwycięstwa lub porażki. Wyświetlenie odpowiedniego komunikatu na koniec walki.
Nagrody	Gracz podnosi walutę , przedmioty, karty. Przeniesienie ich do ekwipunku.	Przedmioty zostają przypisane do ekwipunku gracza. Wyświetlenie odpowiednich ikon.
Sklep	Gracz odwiedza sklep gdzie może sprzedać lub kupić przedmioty, karty. W przypadku sprzedaży doliczona zostaje waluta. Przedmiot znika z ekwipunku. W przypadku kupna zostaje odliczona waluta. Przedmiot pojawia się w ekwipunku.	Przejście do nowej sceny . W przypadku kupna przypisanie nowych przedmiotów kart do ekwipunku gracza. Odjęcie waluty. Możliwość wyświetlenie komunikatu o braku miejsca. W przypadku sprzedaży usunięcie informacji o przedmiotach, kartach z ekwipunku gracza. Dodanie waluty.
Czarny rynek	Gracz odwiedza czarny gdzie może kupić przedmioty, karty.	Przejście do nowej sceny. W przypadku kupna przypisanie

	W przypadku kupna zostaje odliczona waluta. Przedmiot pojawia się w ekwipunku.	nowych przedmiotów kart do ekwipunku gracza. Odjęcie waluty. Możliwość wyświetlenie komunikatu o braku miejsca.
Postać	Wyświetlone zostaną szczegóły postaci. Możliwość wybrania ulepszenia postaci oraz ekwipunku. Możliwość ulepszania kart potrzebnych do walki.	Pobranie, zmiana i wyświetlenie statystyk. Wyświetlenie przedmiotów w ekwipunku oraz zarządzanie nimi.
Spłacanie długu	Gracz spłaca dług w ratach lub za jednym razem. Dług rośnie codziennie. Spłacenie długu umożliwia graczowi zakończenie gry.	Zmiana wartości długu w zależności od podjętych działań użytkownika. Osiągnięcie konkretnej wartości umożliwia zakończenie gry. Przejście do kolejnej sceny. Wyświetlenie napisów końcowych.
Zadania	Gracz wykonuje różne zadania w celu zdobycia złota, dzięki któremu będzie mógł spłacić dług i ukończyć grę.	Wygenerowanie zadań, w których będzie trzeba spełnić konkretny warunek. Po spełnieniu warunku dodana zostanie kwota do aktualnego stanu złota gracza.

6.2 Historia użytkownika

Poniżej opisane zostały wymagania funkcjonalne w formie user stories.

1.Jako gracz chcę zmienić głośność gry żeby dostosować głośność gry do swoich potrzeb.

Opis: Przejście do odpowiedniej sceny. Zmiana wartości głośności.

2.Jako gracz chcę zmienić grafikę oraz rozdzielczość gry żeby dostosować je do swojego monitora oraz komputera

Opis: Przejście do odpowiedniej sceny. Zmiana wartości grafiki.

3.Jako gracz chcę rozpocząć nową grę aby zagrać, ponieważ się nudzę.

Opis: Przejście do odpowiedniej sceny. Wyświetlenie lobby wraz z jego zawartością. Umożliwienie użytkownikowi dokonywania nowych interakcji.

4.Jako gracz chcę udać się do lochu aby zdobyć walutę oraz przedmioty, ponieważ jest to potrzebne do przejścia gry.

Opis: Przejście do odpowiedniej sceny. Wyświetlenie przeciwników oraz obiektów, z którymi może dojść do interakcji bądź kolizji.

5.Jako gracz chcę pokonać przeciwnika żeby ukończyć loch i przeżyć.

Opis: Walka z przeciwnikiem. Umożliwienie użycia kart. Monitorowanie zdrowia gracza i potwora .Monitorowanie pozostałych statystyk oraz wartości obrażeń.

6.Jako gracz chcę odwiedzić sklep żeby sprzedać przedmioty.

Opis: Przejście do nowej sceny. Umożliwienie sprzedaży. Doliczenie waluty za każdy sprzedany przedmiot. Usunięcie przedmiotu lub karty z ekwipunku.

6.Jako gracz chcę odwiedzić sklep żeby kupić przedmioty.

Opis: Przejście do nowej sceny. Umożliwienie kupna. Odjęcie waluty za każdy kupiony przedmiotu. Dodanie go do ekwipunku. Jeśli jest on pełny opcja kupna zablokowana.

6.Jako gracz chcę odwiedzić czarny rynek żeby kupić wyjątkowe przedmioty.

Opis: Przejście do nowej sceny. Umożliwienie kupna. Odjęcie waluty za każdy kupiony przedmiot lub kartę. Dodanie go do ekwipunku. Jeśli jest on pełny opcja kupna zablokowana.

6.Jako gracz chcę ulepszyć postać aby stała się silniejsza.

Opis. Zwiększanie statystyk postaci.

7.Jako gracz chcę ulepszyć karty by dawały lepsze bonusy lub miały lepsze efekty użytkowe.

Opis: Ulepszenie wymaga odpowiedniej ilości tej samej karty. Po ulepszenie daję ona lepszy efekt.

8. Jako gracz chcę zarządzać swoim ekwipunkiem aby nie panował tam chaos.

Opis: Dowolna możliwość przenoszenia przedmiotów w ekwipunku.

9. Jako gracz chcę wykonać zadania, aby przyśpieszyć ukończenie gry i otrzymać dodatkowe złoto.

Opis: Wykonanie tej czynności powoduje otrzymanie nagrody w postaci złota.

10. Jako gracz chcę spłacić dług aby ukończyć grę.

Opis: Należy wykonać tę czynność aby ukończyć grę. Wydanie odpowiedniej ilości waluty .

7. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne zostały poddane oraz rozdzielone na cztery podstawowe obszary wymagań.

Tabela 1.1. Wymagania niefunkcjonalne dla gry

Obszar wymagań	Nr	Opis
Użyteczność	1	Wszystkie funkcjonalności gry dostępne dla użytkownika muszą działać.
	2	Wszystkie funkcjonalności gry dostępne dla użytkownika muszą mieścić się na pojedynczym ekranie przy rozdzielczości 1280x720.
Niezawodność	3	Wszystkie błędy gry muszą być sygnalizowane czytelnym dla użytkownika komunikatem. Niedopuszczalne jest nagłe zakończenie gry z powodu błędu.

Wydajność	4	Gra powinna osiągać wartość co najmniej 30 klatek na sekundę.
	5	Gra nie powinna się uruchamiać dłużej niż 5 min.
	6	Gra nie powinna wykorzystywać w 100 % zasobów komputera.
Utrzymanie	7	Gra jest przeznaczona na komputery z systemem operacyjnym Windows 7 lub nowszy.
	8	Potrzebna jest określona pojemność dysku do przechowywania gry.
	9	Gra wymaga DirectX

8. Metodyka

Model kaskadowy (waterfall) - polega on na wykonywaniu podstawowych czynności jako odrębnych faz projektowych, w porządku jeden po drugim. Każda czynność to kolejny schodek (kaskada):

1. Planowanie systemu (w tym specyfikacja wymagań)
2. Analiza systemu (w tym analiza wymagań i studium wykonalności)
3. Projekt systemu (poszczególnych struktur itp.)
4. Implementacja (wytworzenie kodu)
5. Testowanie (poszczególnych elementów systemu oraz elementów połączonych w całość)
6. Wdrożenie i pielęgnacja powstałego systemu.

Jeśli któraś z faz zwróci niesatysfakcjonujący produkt cofamy się wykonując kolejne iteracje aż do momentu kiedy otrzymamy satysfakcjonujący produkt na końcu schodków.

Metodyka ta została wybrana ponieważ najbardziej pasuje do naszego projektu.

9. Technologia

Wykorzystane technologie:

- Język programowania – C#
- Framework – Microsoft .NET Framework

Wykorzystane biblioteki:

- Sytem
- UnityEngine

Środowisko:

- Silnik Unity

Wykorzystane programy:

- Asprite
- Visual Studio
- Unity

Wykorzystane assety:

- TextMesh Pro – autor „Unity Technologies”
- Ultimate game music collection – autor „John Leonard French”
- Universal Sound FX – autor „Imphenzia”

10. Podział pracy

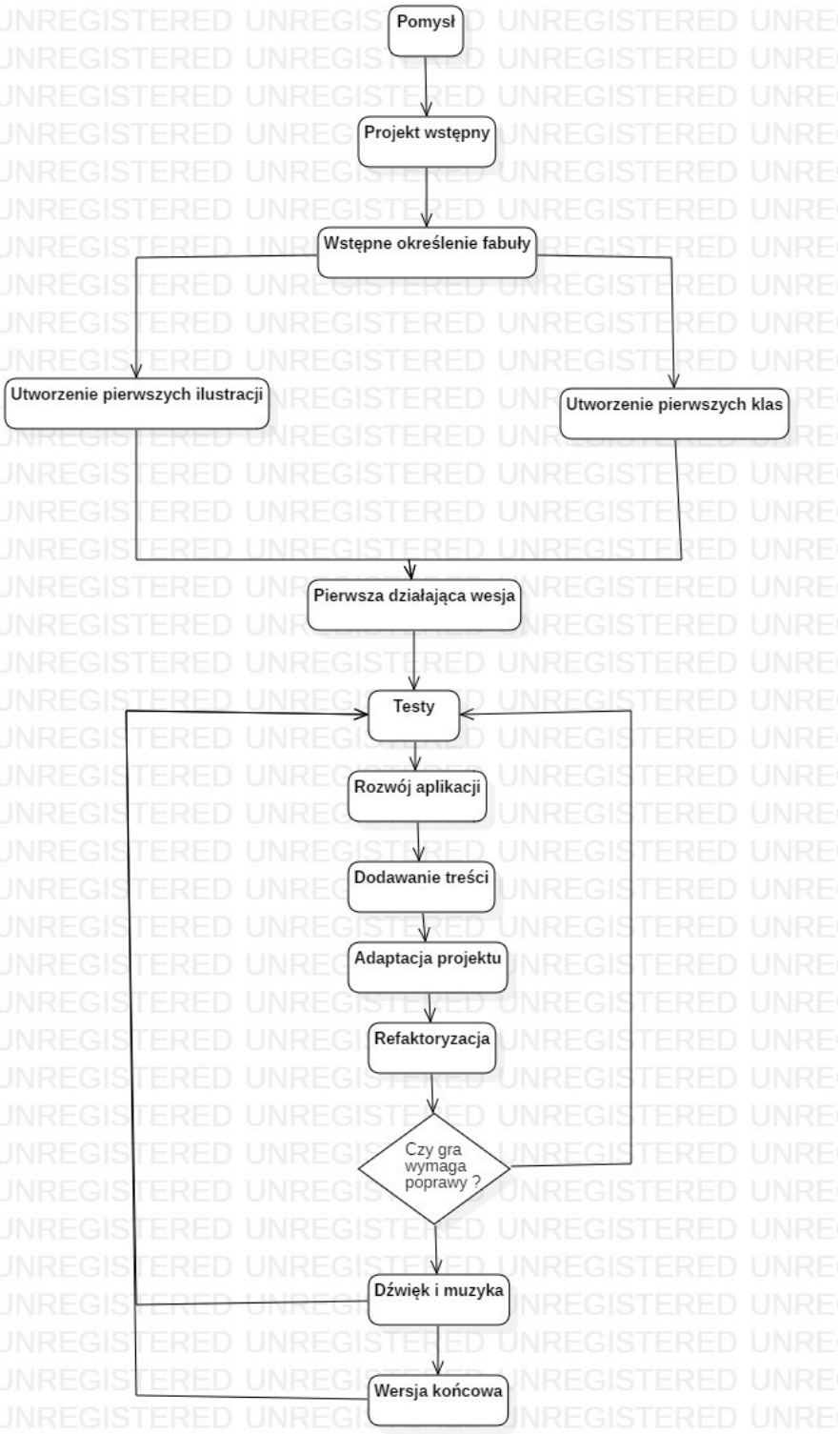
- Pisanie kodu – Krzysztof Sobieski, Sebastian Majewski
- Grafika – Edyta Zakrzewska, Maciej Surawski
- Dokumentacja – Piotr Ołtarzewski
- Testowanie – Wszyscy członkowie grupy

11. Harmonogram

Deadline	Nazwa elementu projektu
9.04.2019	Zaprojektowanie sterowania mechanikami gry
16.04.2019	Stworzenie grafiki miasta i postaci
30.04.2019	Zaimplementowanie sterowania postaci
07.05.2019	Stworzenie grafiki lochów, bossów oraz innych elementów
21.05.2019	Zaprogramowanie mechanik bossów
4.06.2019	Scalenie elementów gry
11.06.2019	Przetestowanie i naprawianie błędów, dokumentacja techniczna projektu
18.06.2019	Gotowy projekt

12. 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

13. 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.

14. Implementacja

Gry wideo, w przeciwieństwie do większości aplikacji użytkowych, są tworami 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 PlayerAtack()
    {
        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 EmemiesScripts - zawiera klasy odpowiadające za zachowania i umiejętności przeciwników. Zawiera on w sobie takie klasy jak: EnemyAtack, 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 = LevelTelepo
rtList[i - 1];
            room.GetComponentInChildren<TeleportBackward>().currentLevel = i;
            room.GetComponentInChildren<TeleportForward>().forward = LevelTeleport
List[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 clicable)
    {
        //jesli stack jest wiekszy niz 1 to pokazyje i ustawia ulosc stakow
        if(clicable.ThisCount>1)
        {
            clicable.ThisStackText.text = clicable.ThisCount.ToString();
            clicable.ThisStackText.color = Color.white;
            clicable.ThisIcon.color = Color.white;
        }
        //jesli stack jest mniejszy niz 2 ukrywa stack size
        else
        {
            clicable.ThisStackText.color = new Color(0, 0, 0, 0);
            clicable.ThisIcon.color = Color.white;}
        //jesli 0 ukrywa ikone
```

```
        if(clicable.ThisCount==0)
        {
            clicable.ThisIcon.color = new Color(0, 0, 0, 0);
            clicable.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 określaające 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.

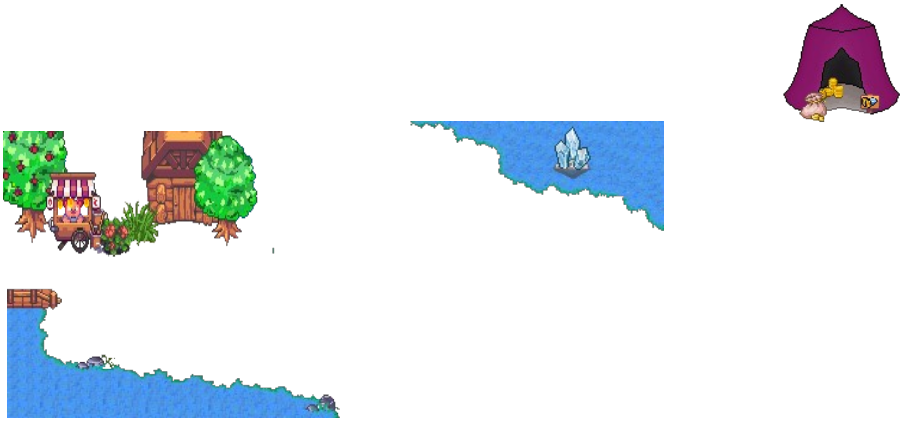
Klasa SoundManager - klasa ta odpowiada za dźwięki występujące w grze.

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: podłoże lobby, namioty (sklep, czarny rynek, i namiot od zadań), dodatkowe elementy wizualne które polepszają odbiór wizualny lobby.

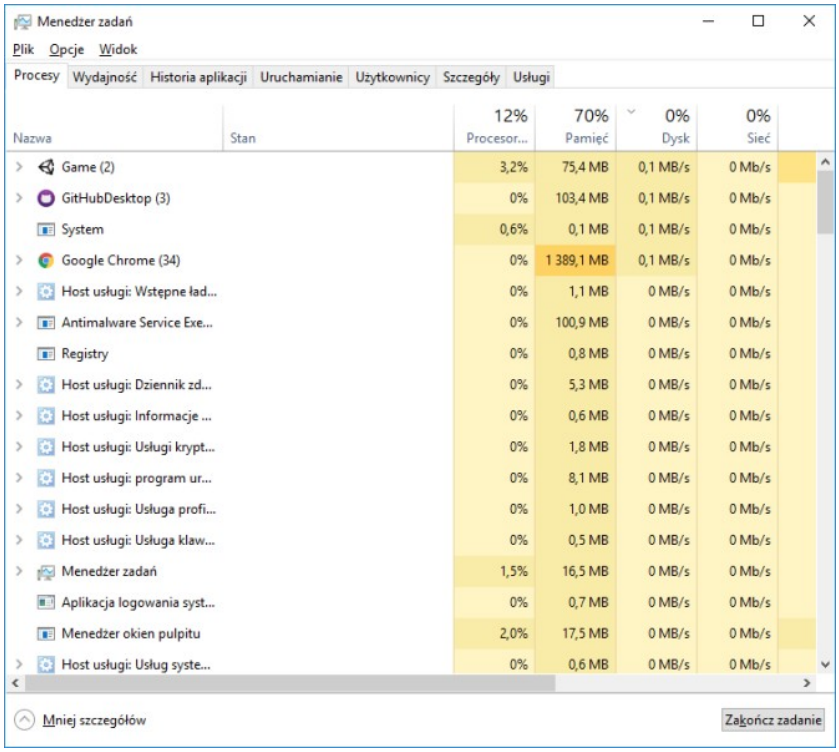


Rysunek 4. Przykładowe grafiki z gry

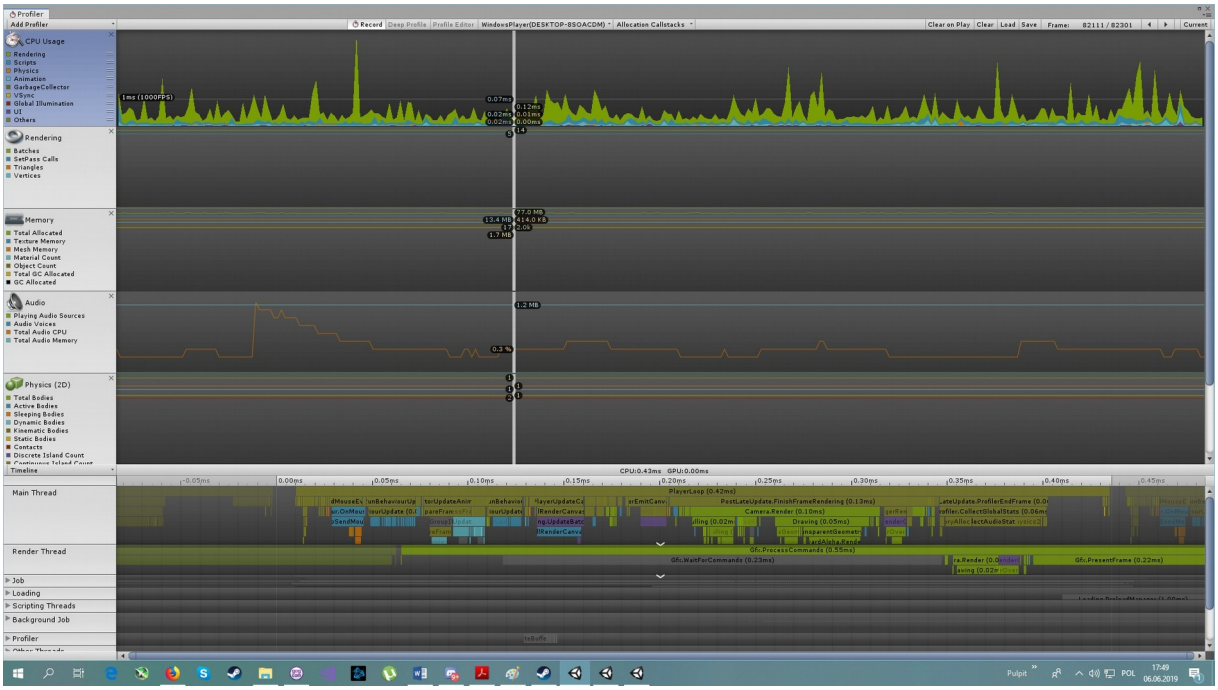
15. Testy

- Działanie i wydajność

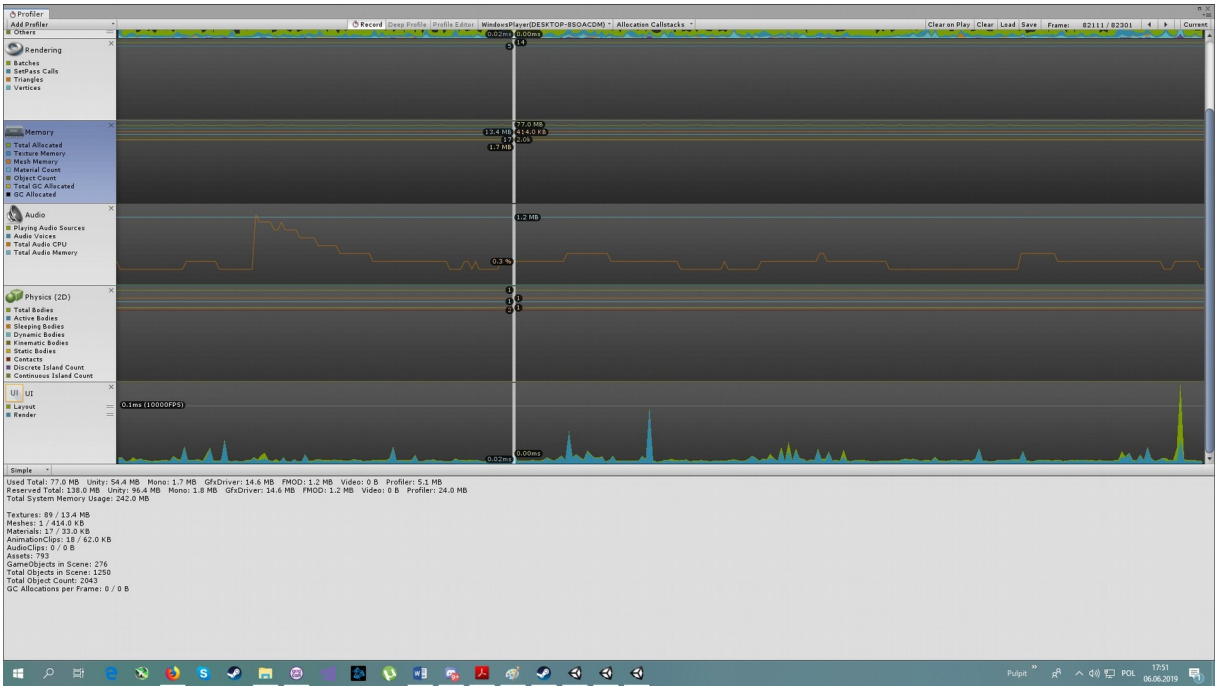
Przedstawiony zrzut pokazuje zużycie karty graficznej, pamięci, procesora przez naszą grę w wysokiej rozdzielczości. Widać że gra jest mało wymagająca i nie potrzebuje dużej ilości zasobów do pracy.



Rysunek 5. Zużycie procesów przez grę



Rysunek 6. Zrzut z Profilera Unity



Rysunek 7. Zrzut z Profilera 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.

16. 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 przetrzymujemy 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.

17. 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.