Streszczenie

Gry komputerowe z początku hobbystyczne projekty tworzone w środowiskach akademickich, na przestrzeni lat przemieniły się w potężny organizm, pozostawiając w tyle zarówno branżę filmową jak i muzyczną. Według analityków wartość globalnego rynku gier komputerowych na rok 2022 wynosiła 184.4 miliarda dolarów [1]. Przez pewien czas w świecie gier nie było miejsca dla indywidualnych deweloperów, jednak olbrzymia przemiana pozwoliła na stworzenie ogólnodostępnych silników takich jak Unity czy Unreal, które umożliwiły praktycznie każdemu na tworzenie własnych produkcji oraz dokładanie się do tej prężnie rozwijającej się branży.

W poniższej pracy zaprezentowana zostanie gra komputerowa 2D, wykonana przy użyciu silnika Unity. Gra należy do gatunku 'Rougelike' [2]. Składa się z różnego rodzaju poziomów, przeciwników i przeszkód. Celem gry jest przejście wszystkich etapów przez gracza przy jednoczesnym pokonywaniu rywali oraz gromadzeniu potrzebnych zasobów.

W pracy wyróżniono sześć rozdziałów. Każdy z nich opisuje inne zagadnienie dotyczące poszczególnych etapów projektowania i tworzenia gry. Pierwsze dwa rozdziały opisują teorię, która posłużyła do tworzenia projektu. Przegląd literatury jest ogólnym spojrzeniem na gatunek wybranej gry wraz z narzędziami dostępnymi na rynku do ich tworzenia. Kolejny rozdział poświęcony jest szczegółowym elementom projektu. W czwartym rozdziale został opisany proces implementacji technik oraz mechanizmów w środowisku Unity. W podsumowaniu rozwinięto temat pozytywnego wpływu produkcji na umiejętności użytkownika oprogramowania, sfery dzieła które mogą ulec w przyszłości rozwojowi oraz ogólne wnioski wynikłe z procesu twórczego produkcji.

Słowa kluczowe: gra komputerowa, gra wideo, rougelike, dungeon crawl, 2D, unity

Abstract

Computer games, initially hobbyist projects created in academic environments, have transformed over the years into a powerful entity, surpassing both the film and music industries. According to analysts, the global market value of computer games reached $184.4 billion in 2022 [1]. For a while, there was no room for individual developers in the gaming world. However, a significant transformation allowed for the creation of accessible engines such as Unity and Unreal, which practically enabled anyone to create their own productions and contribute to this rapidly developing industry.

This project presents a 2D computer game created using the Unity engine. The game belongs to the 'Rougelike' genre, consisting of various levels, enemies, and obstacles [2]. The objective of the game is for the player to progress through all stages while overcoming rivals and collecting necessary resources.

The thesis consists of six chapters, each addressing different aspects of the game design process. The first two chapters present the theory that was used to create the project. The literature review provides a general overview of the chosen game genre along with the tools available on the market for their creation. The next chapter is dedicated to detailed project elements. The fourth chapter describes the implementation process of techniques and mechanisms within the Unity environment. In summary, the topic of the positive impact of production on software user skills was elaborated upon, as well as the areas of work that may undergo future development and the general conclusions resulting from the creative production process.

Keywords: computer game, video game, rougelike, dungeon crawl, 2D, unity

Spis treści

1 Wstęp **1**
1.1 Motywacja . . . 1
1.2 Cele i zakres . . . 1

2 Przegląd literatury **2**
2.1 Przegląd gier 2D Rougelike . . . 2
2.2 Przegląd Unity Game Engine . . . 7
2.3 Przegląd narzędzi i technik tworzenia gier . . . 10
2.3.1 Silniki gier . . . 10
2.3.2 Języki programowania . . . 10
2.3.3 Techniki graficzne . . . 10
2.3.4 Dźwięk i muzyka . . . 11

3 Projekt gry **12**
3.1 Ogólnie o mechanice . . . 12
3.2 Analiza pętli zachowań . . . 12
3.3 Struktura poziomu . . . 13
3.4 Omówienie pokoi . . . 14
3.5 Przedmioty . . . 14
3.6 Interfejs i doświadczenie użytkownika . . . 14
3.7 Oprawa graficzna . . . 16

4 Produkcja **19**
4.1 Konfigurowanie środowiska Unity . . . 19
4.2 Generowanie poziomów . . . 19
4.2.1 Algorytm generacji . . . 19
4.2.2 Generator lochów . . . 19
4.2.3 Generator pokoi . . . 21
4.3 Wdrażanie kontroli gracza i poruszania się . . . 22
4.4 Projektowanie i wdrażanie zasobów gry . . . 28
4.5 Wdrażanie sztucznej inteligencji w grze i zachowania wrogów . . . 31
4.5.1 Algorytm wędrowca . . . 31
4.5.2 Wybór nowego punktu docelowego . . . 33
4.5.3 Wróg dystansowy . . . 33

5 Podsumowanie **35**
5.1 Umiejętności, które rozwija produkcja . . . 35
5.2 Przyszłość i potencjał projektu . . . 35
5.3 Wnioski . . . 36

1 Wstęp

Gry wideo od dawna stanowią niezwykle popularną formę rozrywki, oferującą użytkownikom możliwość przeniesienia się do wirtualnego świata pełnego przygód i emocji. Jednym z najciekawszych gatunków gier wideo jest *roguelike*, w którym gracze wcielają się w role bohatera, eksplorują losowo generowane poziomy i stawiają czoła nieprzewidywalnym wyzwaniom. W niniejszej pracy licencjackiej skupiono się na stworzeniu gry 2D z gatunku *roguelike*.

1.1 Motywacja

Tworzenie gier komputerowych jest niezwykle fascynującym i dynamicznym procesem, który wymaga zarówno kreatywności, jak i technicznej precyzji. Jednak motywacja do stworzenia gry 2D z gatunku *roguelike* opiera się na wielu czynnikach. Przede wszystkim, fenomen *roguelike* polega na generowaniu losowych poziomów, co sprawia, że każda rozgrywka jest unikalna i niepowtarzalna. Ta nieprzewidywalność dostarcza graczom nieskończonej ilości możliwości i wyzwań, co stanowi główną przyczynę ich zaangażowania i zainteresowania.

1.2 Cele i zakres

Celem tej pracy było zgłębienie różnych elementów gry 2D z gatunku *rougelike*, skupiając się głównie na dwóch kluczowych obszarach: generowaniu poziomów oraz sztucznej inteligencji przeciwników. Oba te elementy mają istotny wpływ na doświadczenie graczy oraz dynamikę rozgrywki.

Ponadto, zamiarem było stworzenie satysfakcjonującego modelu rozgrywki zapewniającego angażującą rozrywkę dla gracza wraz ze zróżnicowanym oraz pseudolosowym światem umożliwiającym zachowanie świeżości podczas kolejnych uruchomień gry.

Zakres tej pracy licencjackiej obejmuje projektowanie, implementację oraz analizę wybranych elementów gry 2D z gatunku *roguelike*. Praca skupia się na generowaniu poziomów, sztucznej inteligencji przeciwników oraz innych istotnych aspektach rozgrywki. Do zakresu nie należy przygotowanie *sprite'ów* [3].

2 Przegląd literatury

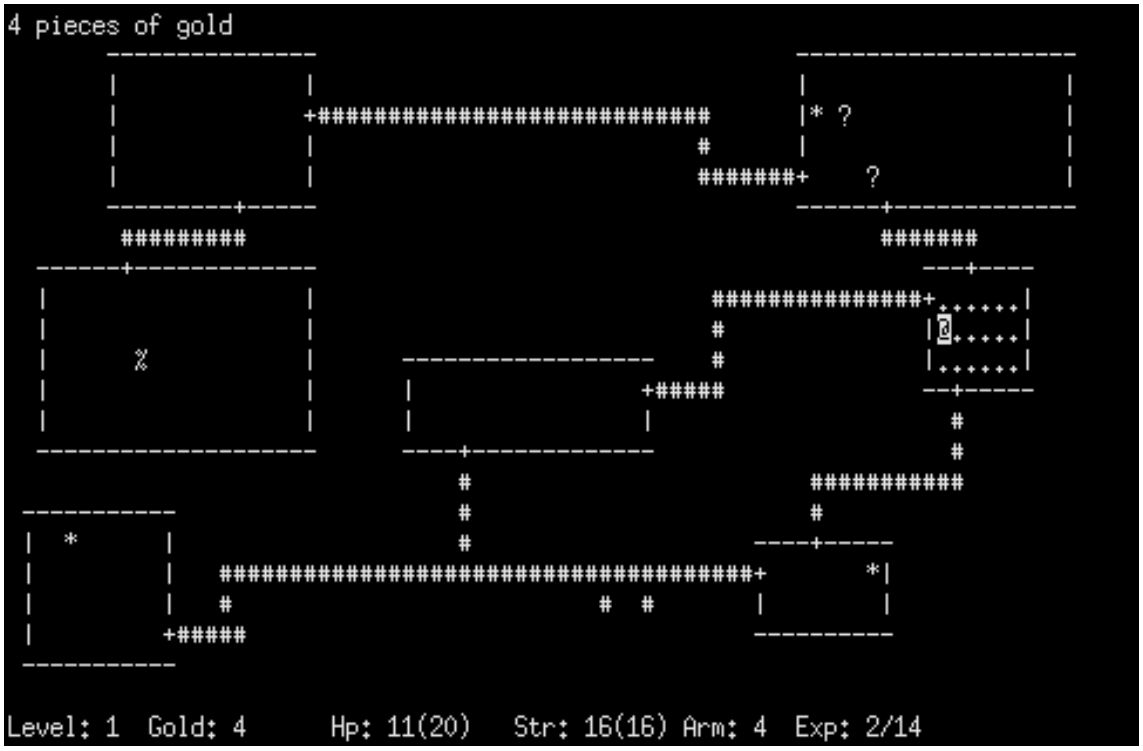
Rozdział ten jest wprowadzeniem w tematykę projektu oraz zawiera rozszerzony opis wybranego gatunku gry. Dodatkowo dokładniej opisano silnik Unity, jego podstawowe elementy, a także przedstawiono inne narzędzia oraz techniki tworzenia gier.

2.1 Przegląd gier 2D Rougelike

Rougelike to rodzaj gier polegających na zwiedzaniu lochów, walce z przeciwnikami i pokonywaniu różnych czekających na gracza przeszkód oraz zbieraniu skarbów znajdujących się w lochu [2].

Gry tego typu cechują się zwiedzaniem lochów (*dungeon crawl*), proceduralnym generowaniem poziomów, permanentną śmiercią gracza w razie starty wszystkich punktów zdrowia oraz poruszaniem się gracza po jednolitej siatce. Większość gier z rodzaju *Rougelike* opiera się na narracji *fantasy*, odzwierciedlającej ich wpływ z gier fabularnych takich jak *'Dungeons & Dragons'* [4].

Pierwsze gry komputerowe *Rougelike* były oparte na *ASCII* i działały w terminalu lub emulatorze terminala. Gry te zostały spopularyzowane wśród studentów i programistów komputerowych w latach 80-tych i 90-tych co doprowadziło do powstania setek różnych wariantów. Nazwa wzięła się od gry pod tytułem *Rouge* utworzonej w roku 1980, jest uważana za prekursora i imiennika gatunku.



Rysunek 1: *Rouge* rok 1980

Z powodu narastającej popularności i wariancji gier *rougelike*, niektórzy gracze oraz deweloperzy postanowili uściślić cechy charakterystyczne tego gatunku. Podczas Międzynarodowej Konferencji Rozwoju Roguelike (*International Roguelike Development Conference*) w 2008 roku, która odbywała się w Berlinie (Niemcy), gracze oraz deweloperzy postanowili ustanowić nową definicję składającą się z ośmiu cech znaną jako *'Interpretacja berlińska'* [5].

Główne cechy według powyższej interpretacji:

1. **Losowo generowane środowisko**
   Środowisko gry jest generowane losowo, co zwiększa grywalność. Położenie przedmiotów i potworów jest również losowe.

2. **Permadeath**
   W momencie śmierci gracza, gra rozpoczyna się od pierwszego poziomu. Istnieje możliwość zapisu gry, jednak po załadowaniu plik zostaje usunięty.

3. **Turn-based**
   Gra jest turowa. Każde polecenie odpowiada pojedynczej akcji. Gracz ma nieograniczony czas by podjąć akcję.

4. **Grid-based**
   Świat jest reprezentowany na siatce kafelków. Gracz i potwory zajmują jedno miejsce na siatce.

5. **Non-modal**
   Każda akcja jest dostępna dla gracza w dowolnym momencie gry. Ruch, walka i inne akcje dostępne są w tym samym trybie gry.

6. **Złożoność**
   Gra jest na tyle złożona, że pozwala osiągnąć dany cel na różne sposoby. Można to osiągnąć dostarczając odpowiednią ilość interakcji potwór/przedmiot i przedmiot/przedmiot.

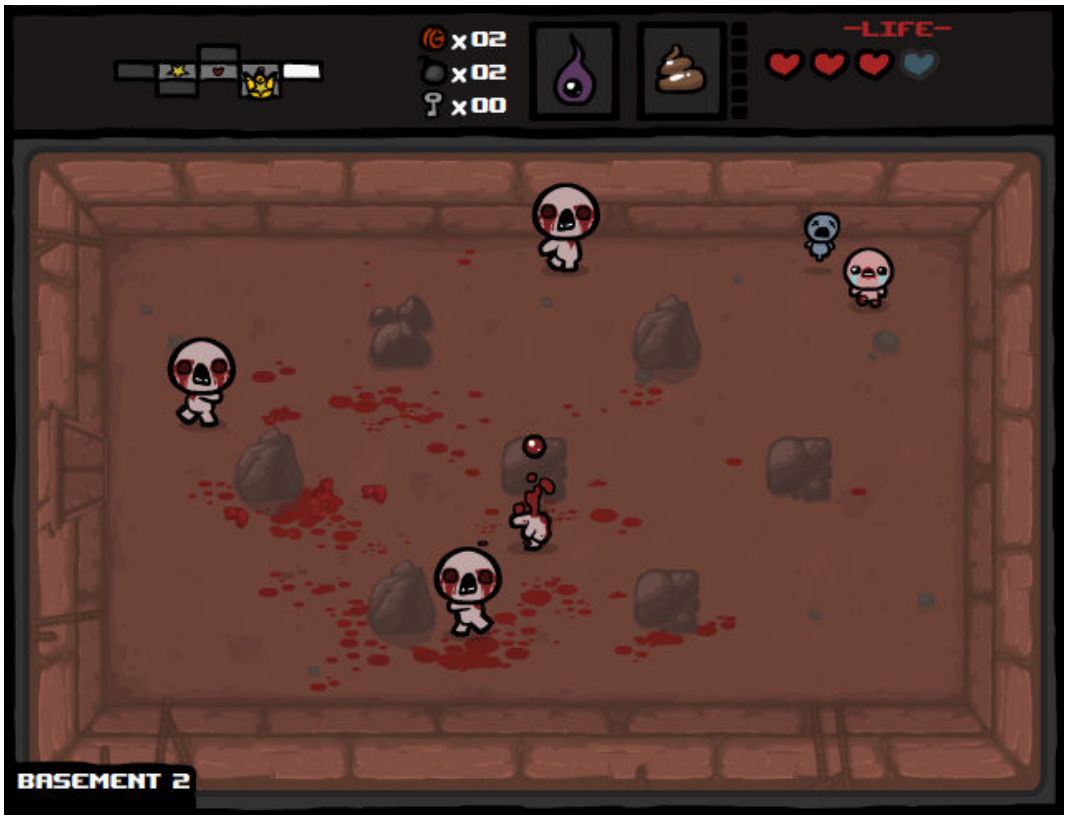
7. **Zarządzanie zasobami**
   Należy odpowiednio zarządzać swoimi zasobami (np. jedzenie, eliksiry, pieniądze) oraz znaleźć dla nich odpowiednie zastosowanie.

8. **Hack n' slash**
   Pomimo wielu dodatkowych elementów gry, zabijanie dużej ilości potworów jest bardzo ważnym czynnikiem składającym się na gry z rodzaju *rougelike*.

Przykłady gier Rougelike:

1. **The Binding of Isaac**

   Gra autorstwa Edmunda McMillena, wydana w 2011 roku na urządzenia Microsoft Windows. Zyskała uznanie przede wszystkim za proceduralne generowanie poziomów, ciekawą rozgrywkę oraz nietypową oprawę audiowizualną.



Rysunek 2: Zrzut ekranu z rozgrywki *The Binding of Isaac*

Zalety

- Dziesiątki ciekawych przedmiotów
- Losowe generowanie lochów
- Interesująca szata graficzna
- Odpowiednio wyważony poziom trudności
- Wciągająca rozgrywka

Wady

- Brak dokładnego opisu przedmiotów
- Gra nie oferuje oficjalnego wsparcia dla kontrolerów

2. **Spelunky**

Podobnie jak *The Binding of Isaac* gra wykonana przez jednego dewelopera o imieniu Derek Yu. Wydana w 2008 roku na urządzenia firmy Microsoft Windows. Gracz wciela się w role grotołaza, który eksploruje szereg jaskiń walcząc z wrogami, zbierać skarby i unikając pułapek.

Rysunek 3: Zrzut ekranu z rozgrywki *Spelunky*

Zalety

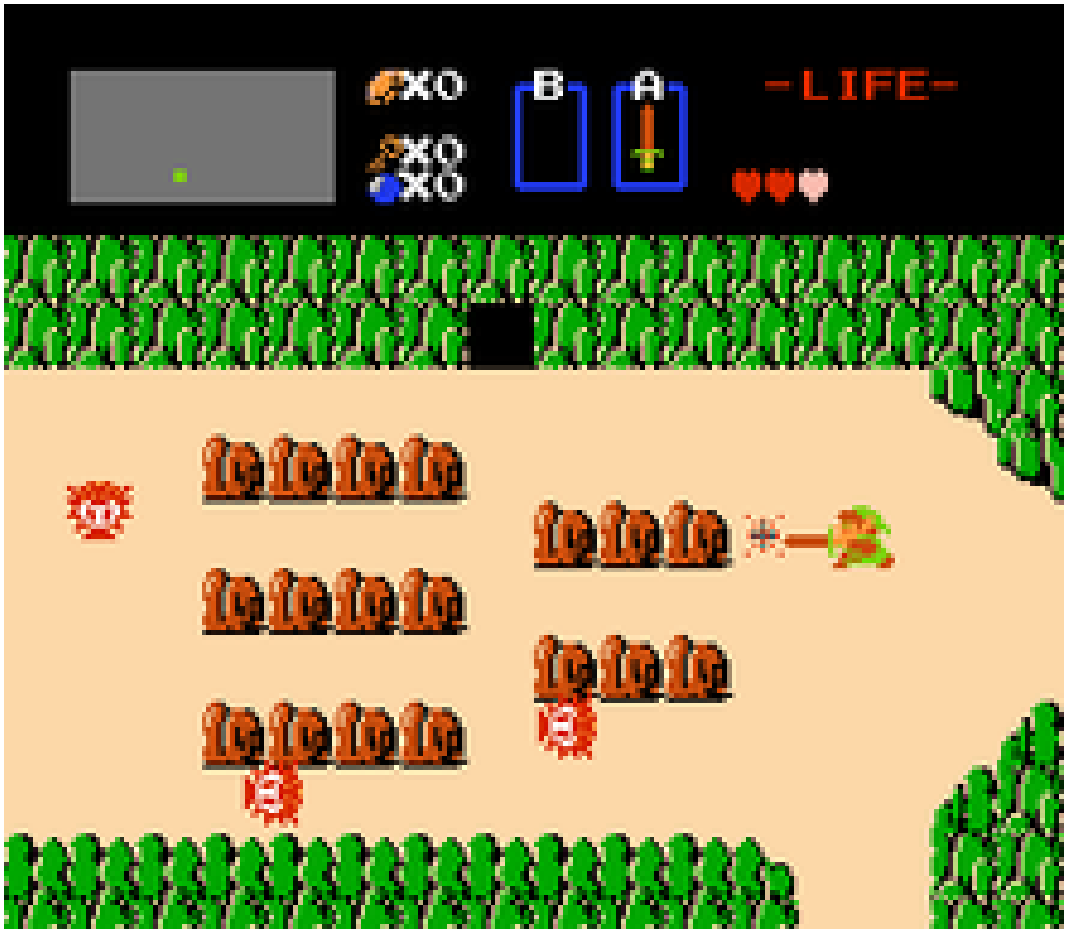
- Satysfakcjonująca eksploracja
- Losowe generowanie jaskiń
- Zadowalający poziom trudności

Wady

- Trudna krzywa nauki

3. **The Legend of Zelda**

Wydana w 1986 roku pierwsza odsłona serii gier *The Legend Of Zelda* jest jedną z tych produkcji, które dały początek całym gatunkom gier. Zrewolucjonizowała podejście do opowiadania historii, budowy poziomów jak i zapoczątkowała wiele stosowanych do dziś mechanik gry. Jest to kamień milowy w historii gier wideo.



Rysunek 4: Zrzut ekranu z rozgrywki *The Legend of Zelda*

Zalety

- Nowatorski jak na tamte czasy otwarty świat
- Poprzez pokonywanie kolejnych lochów i zwiedzanie coraz to odleglejszych zakamarków mapy postać zdobywa ulepszone przedmioty co prowadzi do poczucia naturalnego rozwoju
- Liczne ukryte przejścia, pomieszczenia nadające bogactwa przedstawionemu światu

Wady

- Gra jest znana z bycia prawie niemożliwej do przejścia bez uprzedniego zaznajomienia się z jakimkolwiek poradnikiem

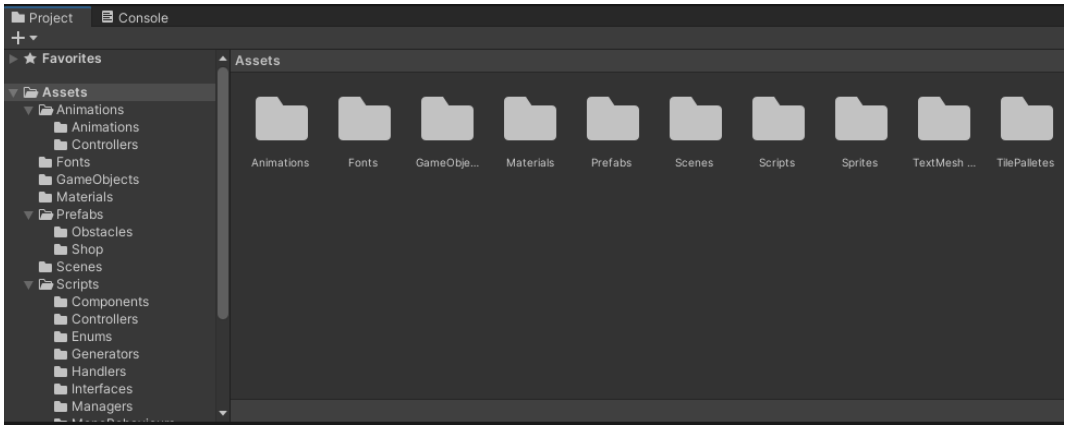
2.2 Przegląd Unity Game Engine

Unity (powszechnie znany jako Unity3D) jest silnikiem gier oraz zintegrowanym środowiskiem programistycznym (IDE) do tworzenia mediów interaktywnych, w szczególności gier wideo. Jak ujął to dyrektor generalny David Helgason Unity 'to zestaw narzędzi do tworzenia gier, jest to technologia, która realizuje grafikę, audio, fizykę, interakcje oraz połączenie sieciowe' [6].

Pierwsza wersja Unity (1.0.0) została stworzona przez współpracowników: Davida Helgason, Joachima Ante i Nicholasa Francis w Danii [6]. Pierwszy produkt wprowadzono na rynek 6 czerwca 2005 roku [7]. Celem było stworzenie niedrogiego silnika gry z profesjonalnymi narzędziami dla twórców gier amatorskich, przy jednoczesnej 'demokratyzacji branży tworzenia gier'. Ta trójka została zainspirowana łatwym przepływem pracy, oraz interfejsem typu 'przeciągnij i upuść' produktu Apple Final Cut Pro [6]. Pierwotnie wydany Unity był dostępny wyłącznie dla systemu Mac OS X, a programiści mogli wdrażać swoje dzieła tylko na kilku platformach. Obecna wersja (w chwili pisania tekstu 2022.3.0) jest obsługiwana zarówno w systemach Windows, Linux jak i Mac OS X i oferuje co najmniej tuzin platform docelowych.

Edytor Unity składa się z wielu okien podrzędnych. Najczęściej używane to: Przeglądarka Projektu, Inspektor, Widok Gry, Widok Sceny i Hierarchia [8].

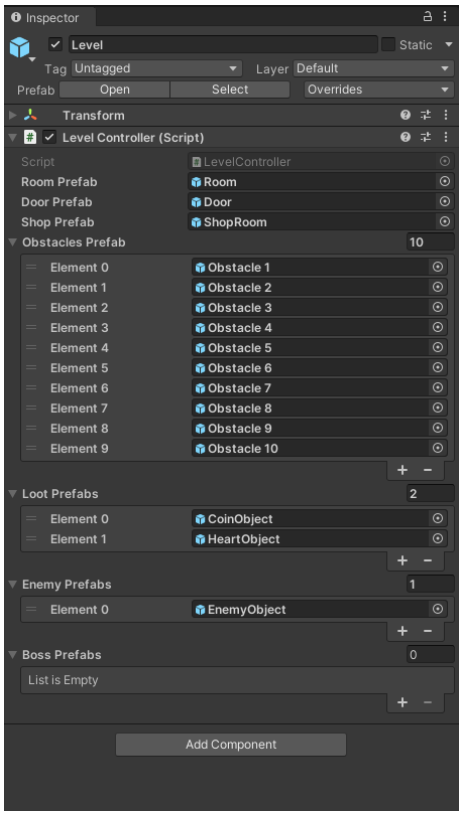
1. **Przeglądarka Projektu**



Rysunek 5: Zrzut ekranu Przeglądarki projektu Unity

Przeglądarka Projektu to okno zawierające wszystkie zasoby, które zostały zaimportowane do Unity i są dostępne dla użytkownika. Układ jest niemal identyczny do Finder w systemie Mac OS X lub Explorer w Microsoft Windows. Dzięki temu programiści czują się bardziej komfortowo i znajomo korzystając, z interfejsu podobnego do tych, których używają na co dzień.

2. **Inspektor**



Rysunek 6: Zrzut ekranu Inspektora Unity

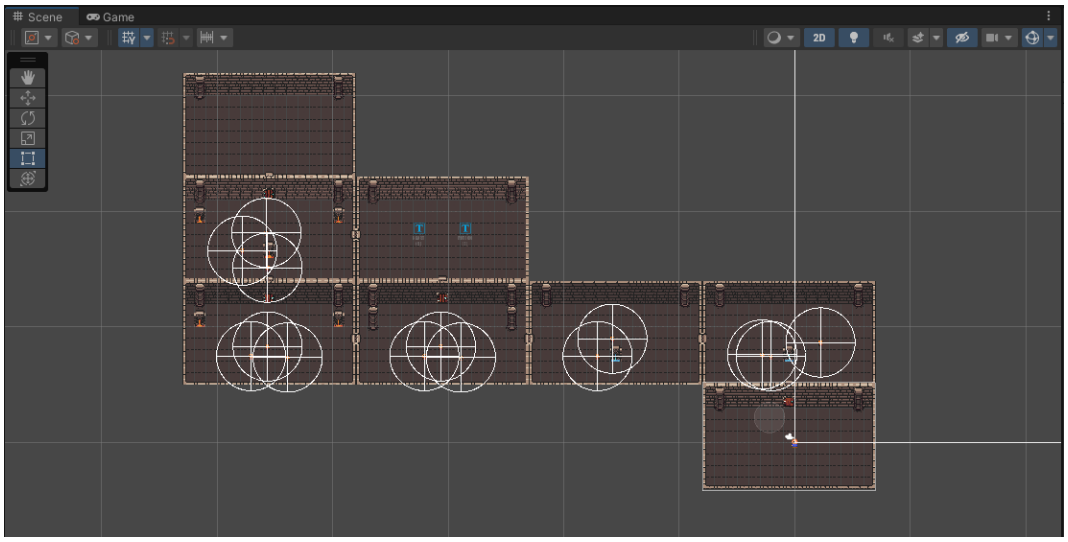
Inspektor zapewnia wgląd do szczegółów każdego obiektu gry *GameObject* oraz umożliwia ich modyfikację. W tym miejscu programista może dostosować różne wartości dotyczące gry. Inspektor zawiera wszystkie Komponenty, które są dołączone do obiektu (np. Skrypt, Fizykę, Zderzacze, Dźwięk). W tym miejscu można również przypisywać i mutować zmienne udostępniane ze skryptów. Inspektor to potężne narzędzie pozwalające zajrzeć do wnętrza obiektu.

3. **Widok Gry**

Rysunek 7: Zrzut ekranu Widoku Gry Unity

Widok Gry oferuje użytkownikowi podgląd *WYSIWYG* (*what you see is what you get*), który pozwala zobaczyć jak będzie wyglądała gra bez konieczności jej zbudowania.

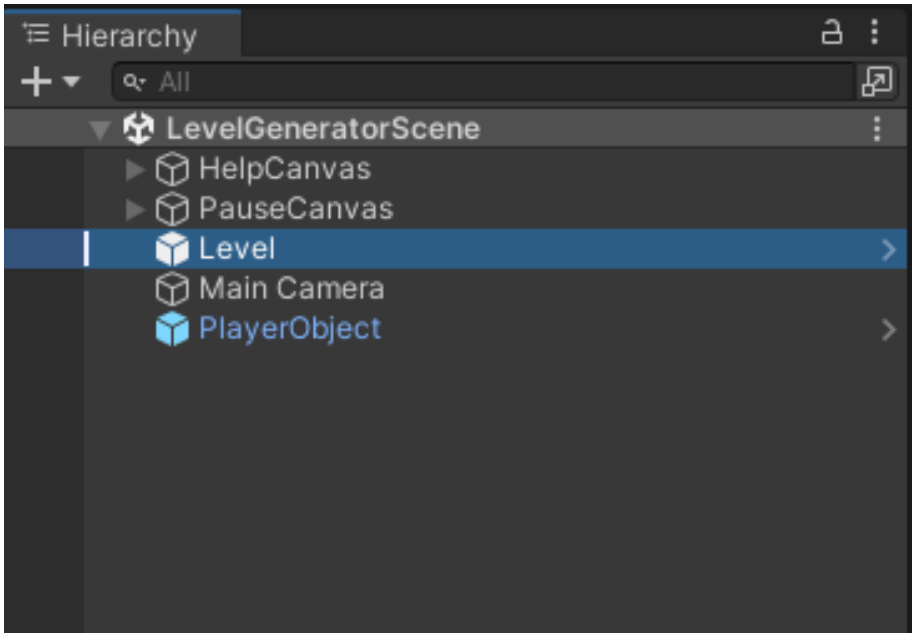
4. **Widok Sceny**



Rysunek 8: Zrzut ekranu Widoku Sceny Unity

Widok Sceny to miejsce, w którym gra jest tworzona. Deweloper może przeciągać i upuszczać zasoby z Przeglądarki Projektu. Elementy sterujące obiektami 2D / 3D pozwalają użytkownikowi umieszczać obiekty w idealnej pozycji, co do piksela.

5. **Hierarchia**



Rysunek 9: Zrzut ekranu Hierarchii Unity

Okno Hierarchii zawiera listę wszystkich obiektów znajdujących się w bieżącej scenie. Ta lista jest automatycznie aktualizowana, gdy obiekt jest wprowadzony na scenę. W tym oknie przeciągając obiekt na inny obiekt, programista może utworzyć hierarchie rodzic-dziecko.

2.3 Przegląd narzędzi i technik tworzenia gier

W dzisiejszych czasach istnieją różnorodne narzędzia oraz techniki używane w procesie tworzenia gier. Jest wiele czynników, które należy wziąć pod uwagę, poniżej przedstawiono kilka z nich.

2.3.1 Silniki gier

- **Unreal Engine**
  Silnik gier komputerowych wyprodukowany w 1998 roku, głównie wykorzystywany w gatunkach gier FPS (*first-person shooter*). [9] Oferuje imponujące efekty wizualne, silne narzędzia do edycji oraz wsparcie dla różnych platform.

- **CryEngine**
  Znany z niesamowitej grafiki i realistycznego oświetlenia. Wymaga większej znajomości technicznej, ale oferuje zaawansowane funkcje.

- **Godot**
  Silnik gier Godot jest narzędziem *open-source* do tworzenia gier 2D i 3D. Zapewnia wiele funkcji i narzędzi, które pomagają w procesie tworzenia gier. Uważany za największego konkurenta Unity Engine.

2.3.2 Języki programowania

- **C#**
  Język programowania używany w silniku Unity. Jest stosunkowo łatwy do nauczenia i pozwala na tworzenie zarówno gier 2D, jak i 3D.

- **C++**
  Potężny język programowania, szeroko stosowany w branży gier. Pozwala na tworzenie wydajnych i zaawansowanych gier, ale wymaga dużej wiedzy programistycznej.

- **Python**
  Popularny język programowania, który może być używany do tworzenia gier. Jest prosty w nauce i posiada wiele bibliotek dedykowanych tworzeniu gier.

2.3.3 Techniki graficzne

- **Renderowanie 2D i 3D**
  Renderowanie 2D odnosi się do procesu wyświetlania grafiki dwuwymiarowej, podczas gdy renderowanie 3D odnosi się do generowania trójwymiarowych scen. W renderowaniu 2D wykorzystuje się przede wszystkim rasteryzację, a w renderowaniu 3D na przykład rzucanie promieni (*ray tracing*), cieniowanie czy mapowanie tekstur, aby stworzyć wrażenie trójwymiarowości.

- **Oświetlenie**
  Oświetlenie jest kluczowym elementem w grafice komputerowej. W grach stosuje się różne techniki oświetlenia, takie jak dynamiczne oświetlenie punktowe, światła kierunkowe, oświetlenie ambientowe i wiele innych.

- **Animacja**
  Techniki animacji obejmują animację klatka kluczowa (*keyframe animation*), w której animacje są tworzone przez definiowanie kluczowych klatek czy też animację szkieletową. Stosuje się techniki takie jak blendowanie animacji, fizyka postaci i mocowanie, aby nadać postacią płynne i realistyczne ruchy.

2.3.4 Dźwięk i muzyka

- **Dźwięk efektów specjalnych**
  W grach dźwięk efektów specjalnych ma na celu wzmocnienie atmosfery i interakcji gracza z otoczeniem. Techniki te obejmują zastosowanie dźwięków środowiskowych, dźwięków oddziałujących z fizyką (np. odgłosy eksplozji, strzałów), dźwięków czynności postaci (np. skoki, otwieranie drzwi) oraz dźwięków interfejsu użytkownika.

- **Muzyka**
  Muzyka odgrywa ważną rolę w tworzeniu atmosfery i nastroju w grach. Skomponowana ścieżka dźwiękowa może dostosowywać się do akcji w grze, akcentować emocje i podkreślać ważne momenty.

- **Dźwięk przestrzenny**
  Odzwierciedla pozycję i odległość źródła dźwięku wirtualnego w grze, co pomaga w kreowaniu bardziej immersyjnego środowiska dźwiękowego.

3 Projekt gry

Projektowanie gier jest procesem składającym się z następujących czynności [10]:

- Wyobrażenia sobie gry,
- Zdefiniowania sposobu, w jaki działa,
- Opisania elementów, które składają się na grę (pojęciowych, funkcjonalnych, artystycznych i innych).

3.1 Ogólnie o mechanice

Przedstawiana w pracy produkcja jest przedstawicielem gatunku rougelike (wyżej wyjaśnione) co za tym idzie implementuje zasady charakterystyczne dla tegoż typu gier. Ma ona wyraźnie nakreśloną pętle zachowań gracza, czyli schemat powtarzalnych czynności, które mają wzbudzić w graczu satysfakcję z ich wykonywania [11].

Ogólny zarys rozgrywki rysuje się następująco: po rozpoczęciu nowej gry zaczyna się poziom pierwszy, inicjalizuje i generuje się nowy loch (z ang. *dungeon*). Loch składa się z licznych pokoi, które pełnią różne funkcje. Ich liczba zależy od numeru poziomu na którym się aktualnie znajduje gracz. Celem gracza jest przemierzenie wygenerowanego lochu oraz pokonanie głównego przeciwnika, tak zwanego *bossa* (nazywanie głównego przeciwnika poziomu *bossem* jest tradycją w świecie gier, [12]).

Na drodze do celu stoją liczne wyzwania. Pokoje, które składają się na loch są zaludnione przez słabszych, lecz liczniejszych od głównego antagonisty, przeciwników. Siła napotkanych stworów, jak i wiele innych elementów rozgrywki dyktowana jest przez numer aktualnego poziomu. To rozwiązanie sprawia, że wraz z postępem gracz będzie napotykał coraz to silniejszy opór, co przeciwdziała monotonii oraz podnosi satysfakcję z kończenia coraz to trudniejszych poziomów.

Oprócz utrudnienia w postaci pomniejszych wrogów, na każdym poziomie generuje się jedno pomieszczenie w którym gracz może nabyć, za zdobytą walutę, przedmioty które ułatwią mu dalszą eksplorację.

Po pokonaniu głównego przeciwnika gracz przechodzi do kolejnego poziomu. Następuje inkrementacja numeru poziomu wraz z ponowną generacją lochu, wszystkich pokoi itd. Zaczynana jest przez gracza świeża rozgrywka z nowej pozycji startowej.

Zgodnie z założeniami gatunku gra nie ma teoretycznego końca; w praktyce jednak mówimy tu o poziomie, którego numer będzie maksymalną wartością typu całkowitego *int*. Koniec gry następuje gdy gracz zostanie pokonany przez *bossa* lub dowolnego z potworów znajdujących się w pokojach między startem a destynacją użytkownika. Respektując zasady produkcji *rougelike*, następuje wtedy definitywny koniec rozgrywki. Graczowi pozostaje rozpocząć nową grę albo zakończyć działanie programu.

3.2 Analiza pętli zachowań

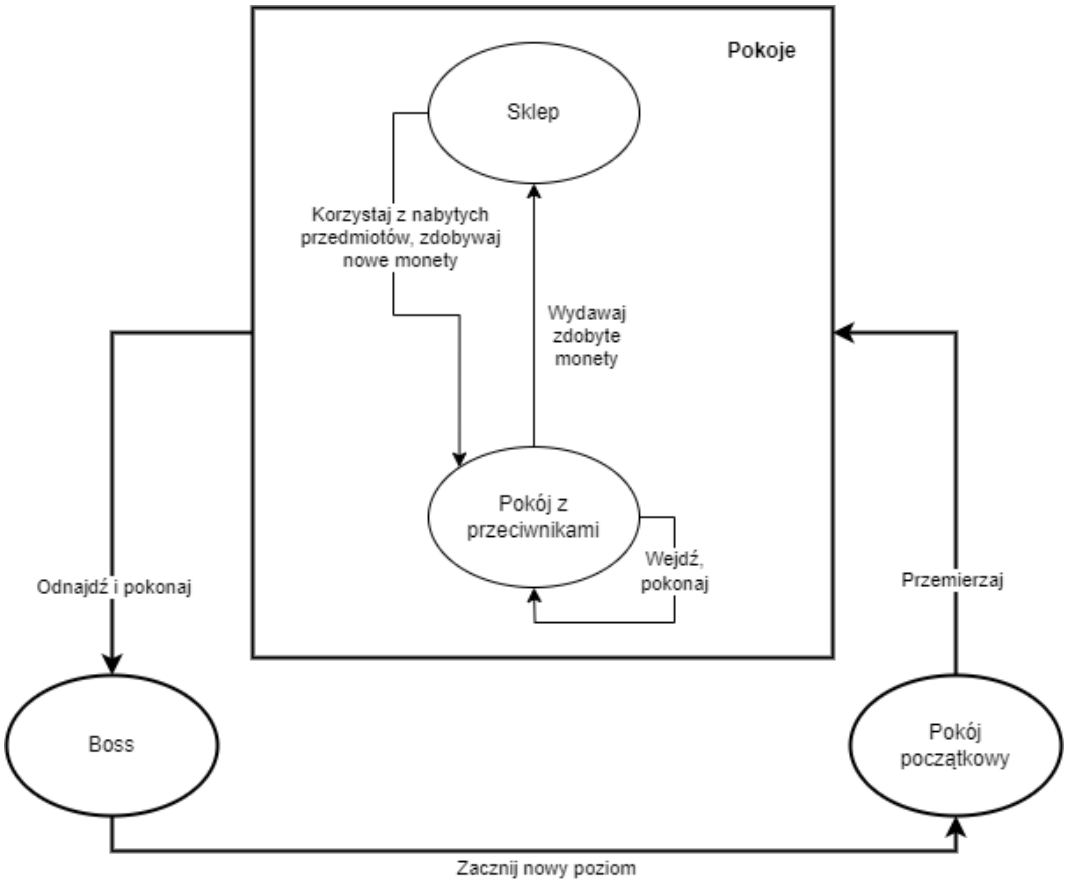
Przez całość rozgrywki gracz wykonuje jasno zdefiniowane oraz łatwe do opanowania czynności. Wyzwanie stanowi coraz to większy loch oraz wytrzymalsi i groźniejsi przeciwnicy. Prostota w pojęciu mechanik oraz cykliczna natura nadaje produkcji duży potencjał na jej wielokrotne rozgrywanie przez użytkownika [13].

Z kolei proste zasady rozgrywki, w połączeniu z coraz to trudniejszym wyzwaniem sprawiają, że gracz szybko opanowuje podstawy gry, lecz trudno osiąga w niej mistrzostwo. Przykładem innego tego typu gry są na przykład szachy. Powiązane z tą cechą jest tak zwane Prawo Bushnella [14], które jest de facto aforyzmem Nolana Bushnella, założyciela Atari, z którego wywodzi się właśnie koncept *'easy to learn, hard to master'*.

W grze możemy wyróżnić kilka pętli zachowań gracza [11]. Pętlami określamy zbiór czynności z określoną kolejnością ich wykonywania, z których każda z nich prowadzi do kolejnej, a finalnie po ostatniej z nich następuje początkowa czynność i cykl się powtarza. Pętle zachowań jakie składają się na niniejszą produkcję, to jedna główna pętla i dwie poboczne.

Pętlą główną nazywamy czynności, które gracz wykonuje aby osiągnąć cel; pokonanie głównego przeciwnika. Po osiągnięciu celu pętla prowadzi do nowego poziomu, w którym to gracz będzie wykonywał identyczny czynności by pokonać nowego bossa.

Pobocznymi z nich są pętle związane z pokonywaniem przeciwników w pojedynczych pokojach co prowadzi do kolejnego pomieszczenia z nowymi przeciwnikami. Następna pętla jest związana ze zdobywaniem monet z przeciwników w celu nabycia przedmiotów, które później służą jako ułatwienie przy zdobywaniu zasobów. Poniższy schemat obrazuje występujące w produkcji pętle:



Rysunek 10: Schemat pętli zachowań w grze

3.3 Struktura poziomu

Na każdy z poziomów składa się loch wypełniony połączonymi ze sobą pomieszczeniami. Liczba pomieszczeń zależy od numeru aktualnego poziomu i opisuje ją poniższy wzór:

$n = \lfloor p * 1.25 + r + 5 \rfloor$, gdzie
$n -$ liczba pokoi
$p -$ numer poziomu
$r -$ losowa liczba całkowita z przedziału $[0, 3]$

Zarówno zawartość pomieszczeń, ich typ jak i ogólny kształt lochu są randomizowane. Generacja proceduralna układu pokoi jest charakterystyczna dla gier typu *rougelike*. Takie podejście do budowania nowych poziomów zapewnia, że użytkownik ma za każdym razem częściowo unikatowe doświadczenie jednocześnie nie ma potrzeby uczenia się mechanik od nowa.

3.4 Omówienie pokoi

Na każdy loch składa się określona liczba pokoi. Wyróżniamy następujące 4 rodzaje pokoi:

1. Pokój startowy
2. Pokój z bossem
3. Pokój ze sklepem
4. Pokój z potworami

Każdy z pokoi, po za pokojami z potworami, występuje w lochu tylko raz. Pokój startowy jest jedynie pozycją startową i nie zawiera żadnych szczególnych elementów ani przeciwników.

Pokój z bossem jest celem rozgrywki, a pokonanie w nim znajdującego się przeciwnika umożliwia progresję na następny poziom.

Sklep z kolei pozwala graczowi na zakup przedmiotów które mają na celu ułatwienie mu osiągnięcia celu. Zawsze dostępny jest w nim zakup punktów zdrowia oraz losowo wybranej przy generacji poziomu mikstury, której użytkownik może użyć w dowolnym momencie swojej rozgrywki, aby zwiększyć jedną ze statystyk swojego bohatera.

Na pokoje z potworami składają się potwory, które zamieszkują ten typ pomieszczeń. Nagroda jaka pojawia się po ich pokonaniu oraz przeszkody, które nadają pokojowi unikalnego wnętrza. Rodzaj potworów, ich liczba, przeszkody jak i nagroda są losowane i ustalane podczas generacji poziomu. Po pokonaniu potworów z pokoju gracz jest nagradzany wylosowaną nagrodą, może być to albo serduszko albo moneta. Ich funkcje zostaną omówione poniżej.

Podczas nieobecności gracza w pokoju następuje odliczanie, po którym część potworów się odrodzi. Podnosi to poziom trudności i zapewnia, że gracz nie będzie nigdy pewny, że dany pokój jest już bezpieczny. Po ponownym pokonaniu przeciwników nagroda nie jest przyznawana.

3.5 Przedmioty

W grze występuje kilka rodzajów przedmiotów - serduszka, monety oraz eliksiry. Serduszka po podniesieniu nie trafiają, jak pozostałe rzeczy, do ekwipunku lecz odnawiają ubytki w poziomie życia gracza. Monety służą do dokonywania zakupów w pokoju ze sklepem. Mikstury z kolei są przedmiotami, które gracz może spożyć w czasie swojej przygody. Mają one za zadanie zwiększyć prędkość poruszania się gracza. Ich efekt jest tymczasowy przez co korzystanie z nich stanowi element planowania taktycznego.

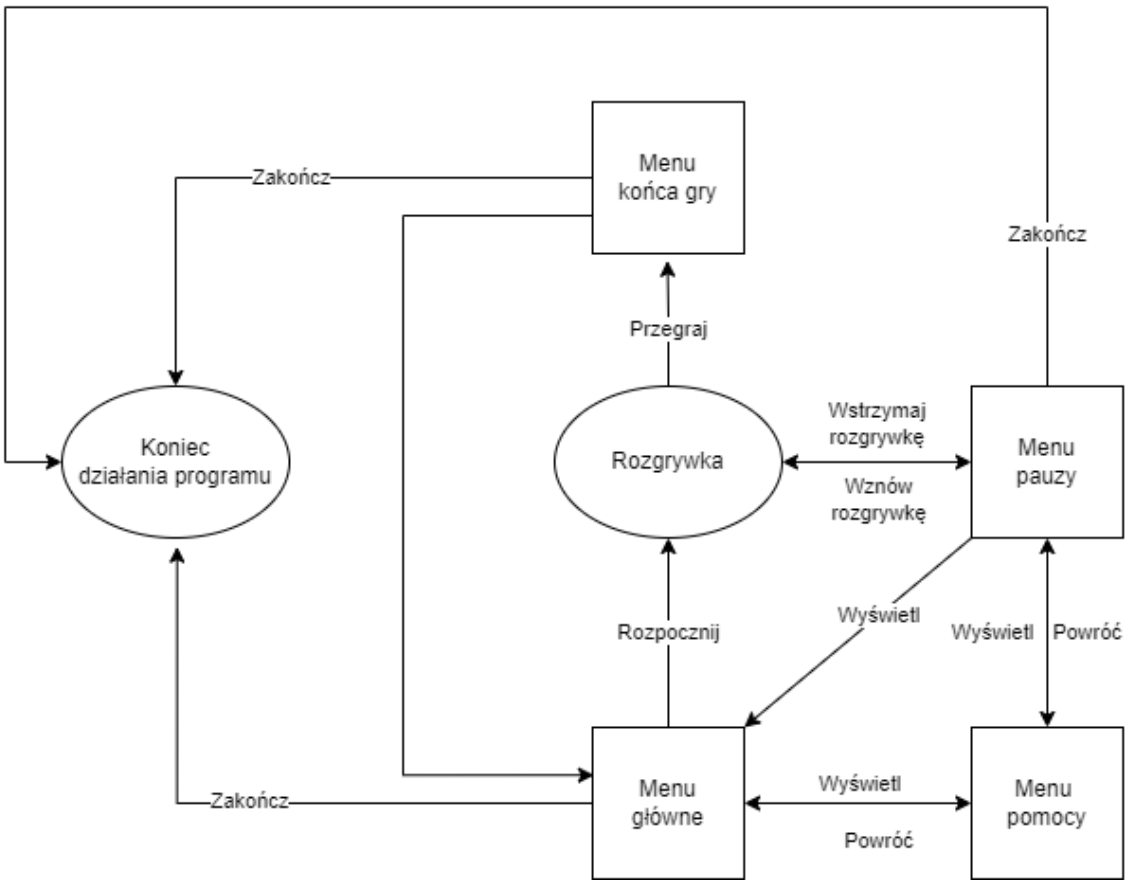
3.6 Interfejs i doświadczenie użytkownika

Interfejs gracza podzielono na widoczny ten podczas rozgrywki oraz panele menu. Podczas rozgrywki na ekranie jest widoczna informacja na temat stanu życia bohatera oraz wszelkie przedmioty przez niego zgromadzone.

Panele menu z kolei mają na celu zapewnić użytkownikowi przejrzysty sposób na poruszanie się po programie oraz dostarczyć potrzebnych informacji. Menu główne, które pojawia się zaraz po uruchomieniu oprogramowania stanowi punkt wejścia do gry. Z jego poziomu gracz ma opcję rozpocząć nową grę, zakończyć program lub wyświetlić menu pomocy.

Panel pauzy pojawia się po naciśnięciu przez użytkownika odpowiedniego klawisza podczas gry. Rozgrywka wtedy się wstrzymuje zapewniając użytkownikowi możliwość zrobienia sobie przerwy. Z poziomu pauzy użytkownik ma również możliwość wyświetlenia menu pomocy.

Menu pomocy ma na celu dostarczenie informacji na temat sterowania oraz ogólnych zasad produkcji. Ostatnim z paneli menu jest panel końca gry (zwany z ang. *game over*) wyświetlający się po tym jak gracz przegra. Poniższy schemat obrazuje występujący schemat menu:



Rysunek 11: Schemat paneli menu

3.7 Oprawa graficzna

Oprawa graficzna produkcji ma na celu nawiązywać do klasycznych gier z lat 80. Cechuje ją styl retro, czyli styl imitujący estetyzm dawnych rozwiązań jednocześnie nie ograniczony tymi samymi barierami sprzętowymi jak w latach, z których się wywodzi.

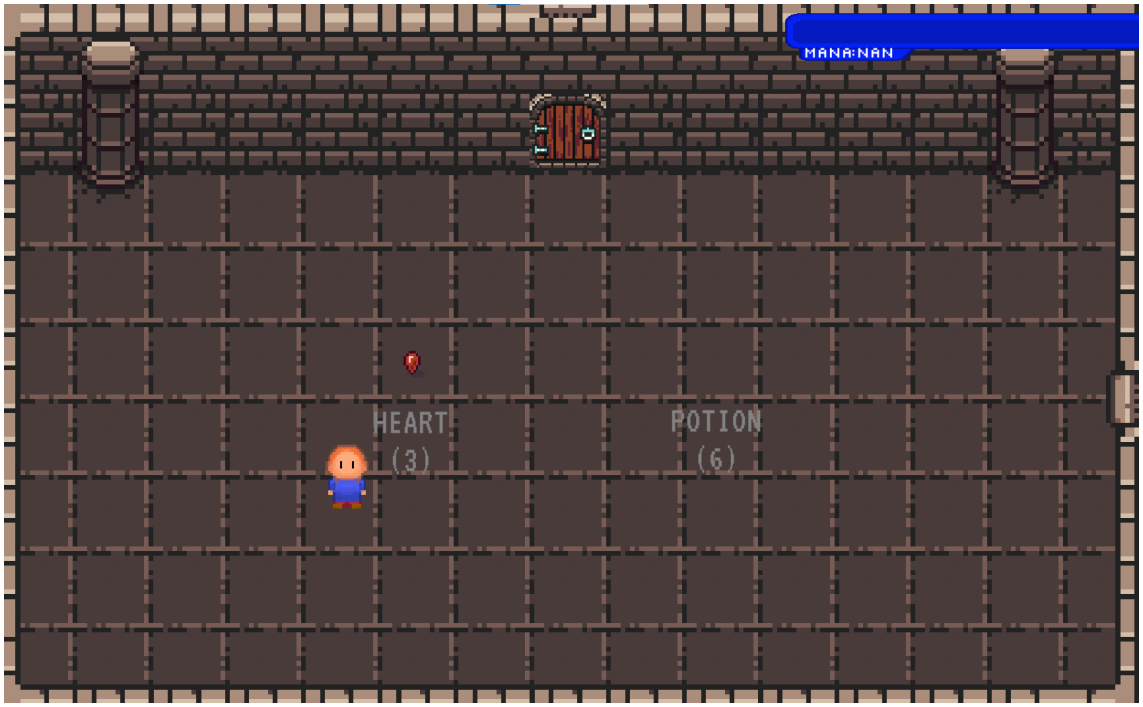
Na grafikę składa się oprawa systemu menu, pasku zdrowia i ekwipunku oraz elementów wypełniających loch. Panele menu zostały przygotowane samodzielnie w darmowym programie do grafiki 2D - Figma. Oprawa graficzna pokoi, które budują loch składają się z połączonych ze sobą *sprite'ów*. Zostały one podzielone na odpowiednie warstwy które zapewniają rozróżnienie między elementami tła, ścianami itd. Następnie utworzone zostały z nich prefabrykaty w celu łatwego wykorzystania przy generacji poziomu.

Do projektu wykorzystano darmowe zasoby graficzne znajdujące się pod adresem *https://0x72.itch.io/dungeontileset-ii*. Licencja *CC0* umożliwia twórcom zrzeczenia się praw autorskich w ich utworach, a tym samym umieszcza je w domenie publicznej, tak aby inny mogli swobodnie na nich budować, ulepszać i ponownie wykorzystywać do jakichkolwiek celów [15].

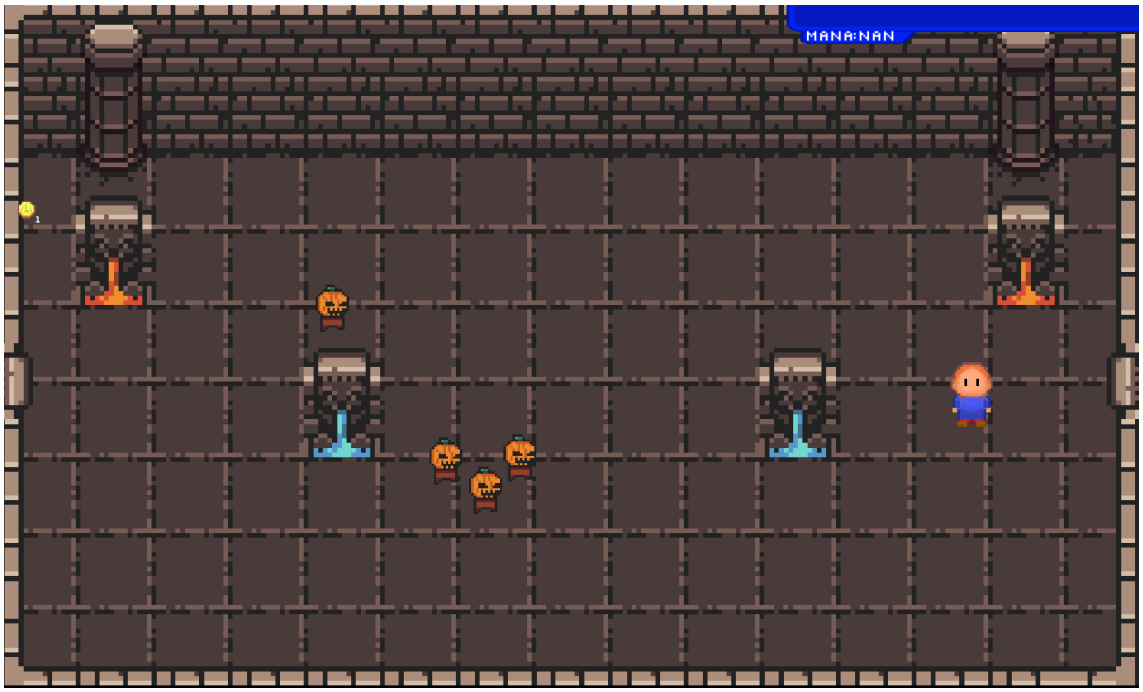
Rysunek 12: Zrzut ekranu głównego menu gry

Rysunek 13: Zrzut ekranu instrukcji obsługi

Rysunek 14: Zrzut ekranu końca gry



Rysunek 15: Zrzut ekranu sklepu



Rysunek 16: Zrzut ekranu gracza i przeciwnika krótkiego zasięgu

4 Produkcja gry

4.1 Konfigurowanie środowiska Unity

Przed rozpoczęciem pracy nad projektem należało skonfigurować Unity. W tym celu pobrano aplikację Unity Hub. Jest to samodzielna aplikacja, która usprawnia nawigację, pobieranie i zarządzanie projektami i instalacjami Unity. [16] Umożliwia pobranie niezbędnych komponentów do tworzenia gier, na przykład komponent *Microsoft Visual Studio 2019*, który w pełni synchronizuje Unity z środowiskiem programowania. Tutaj warto wspomnieć, że począwszy od wersji programu Unity 2018.1, domyślnym edytorem kodu jest właśnie program *Microsoft Visual Studio*. Oprócz tego dodano moduły *Windows*, *Android*, *iOS* i *WebGL*, które umożliwiają tworzenie gry na różnych platformach.

W grze skorzystano z wersji Unity 2021.3.21 wydanej w marcu roku 2023.

4.2 Generowanie poziomów

4.2.1 Algorytm generacji

Mapa jak i elementy, które ją wypełniają są, zgodnie z zasadami gatunku *rougelike*, generowane proceduralnie. W projekcie można wyodrębnić dwa osobne generatory, czyli klasy których przeznaczeniem jest pseudolosowe utworzenie świata rozgrywki: generator lochów oraz generator pokoi.

4.2.2 Generator lochów

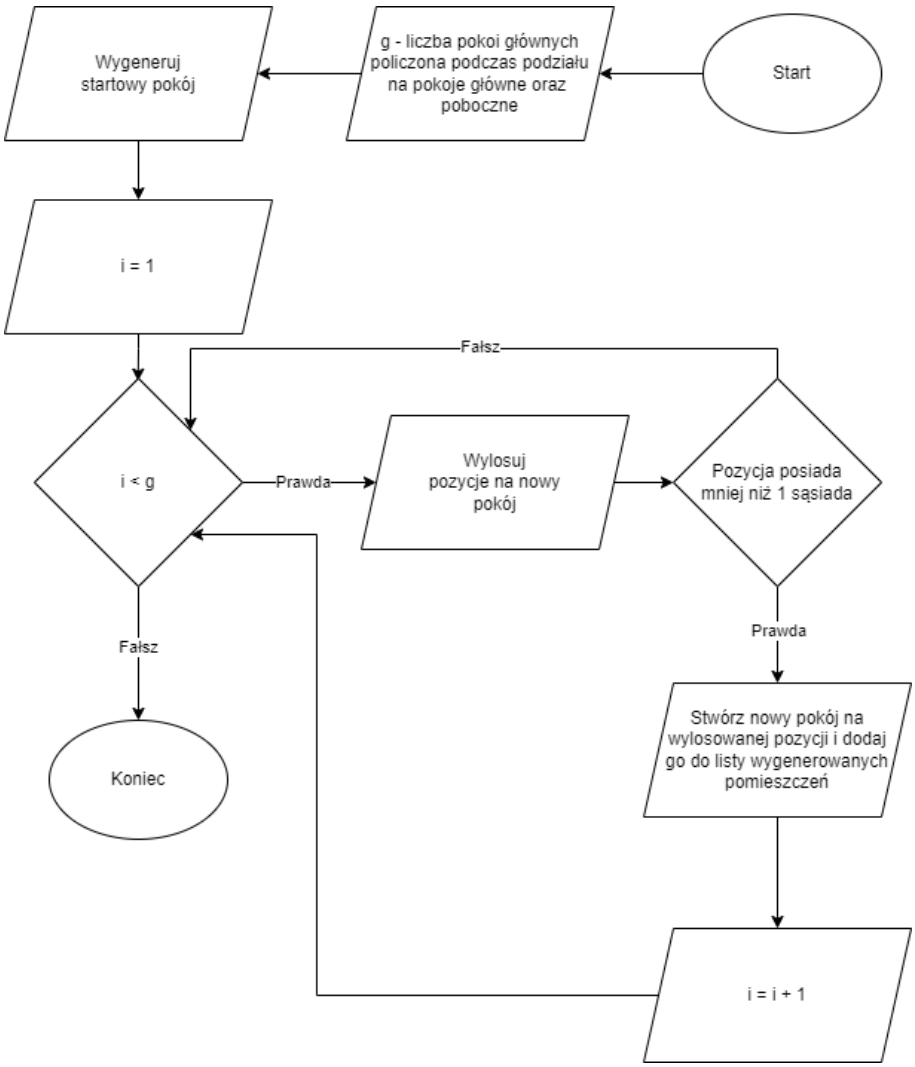
Celem generatora lochu jest wygenerowanie ogólnego rozkładu pomieszczeń na danym poziomie. Jak zostało wyżej wspomniane ilość pokoi zależy od numeru aktualnego poziomu. Generator otrzymuje tę liczbę i dokonuje podziału na pokoje główne i pokoje poboczne. Podział ten określa poniższy wzór:

$$p = \lfloor n * 0.2 \rfloor$$
$$g = n - p, \text{ gdzie}$$
$$p - \text{liczba pokoi pobocznych}$$
$$g - \text{liczba pokoi głównych}$$
$$n - \text{łączna liczba pokoi w lochu}$$

Pokoje główne tworzą ścieżkę prostą w ogólnym układzie pokoi i są generowane w pierwszej fazie generacji lochu. Generacja zaczyna się od utworzenia pierwszego pokoju. Jest to pomieszczenie startowe, w którym gracz rozpoczyna przygodę. Do pokoju startowego przypisuje się współrzędne $x = 0, y = 0$, a każda kolejna pozycja następnego pokoju jest analogicznie przypisana do unikatowej współrzędnej układu kartezjańskiego. Warto wspomnieć, że poszczególne wartości x i y są liczbami całkowitymi.

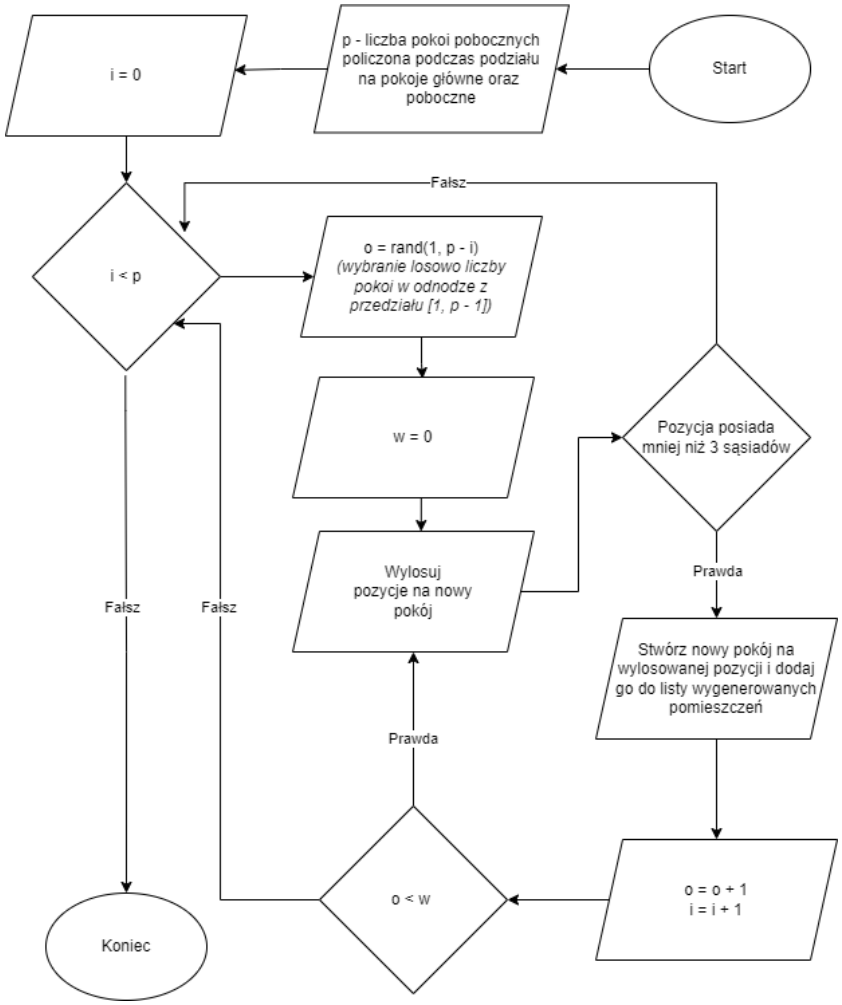
Po wygenerowaniu pierwszego z pomieszczeń następuje wejście do pętli, która będzie się wykonywać aż pojawi się wymagana liczba pokoi głównych. W pętli pierwszym krokiem jest losowe wybranie miejsca sąsiadującego jako potencjalna lokalizacja następnego pokoju. Przez sąsiadujące miejsce rozumiemy miejsce w układzie współrzędnych, które różni się wartością o jeden na współrzędnej x lub współrzędnej y. Inaczej rzecz ujmując odległość następnego pokoju musi wynosić 1 od poprzedniego pomieszczenia. Wyklucza to generowanie pokoi po skosie.

Po tym jak zostanie wylosowana pozycja w jednym z czterech kierunków następuje sprawdzenie ilości sąsiadujących pokoi z wybraną pozycją. Jeżeli nowy pokój miałby sąsiadować z więcej niż jednym pokojem nie jest on w ogóle tworzony i liczba pokoi się nie zmienia. Takie działanie ma zapobiec sytuacji, w której generator natrafiłby na ślepy zaułek i nie miałby jak wybrać pozycji na następny pokój oraz zapewnia prosty i przejrzysty kształt głównej ścieżki w lochu. W przeciwnym przypadku jest tworzony pokój na wylosowanej pozycji, liczba wygenerowanych pokoi zwiększa się a wygenerowany pokój jest przypisywany do zmiennej przechowującej ostatnio stworzone pomieszczenie. Pętla trwa do momentu aż zostanie wygenerowane tyle pokoi ile obliczono w fazie podziału na pokoje główne oraz poboczne. Ostatniemu pokojowi, który zostanie wygenerowany przypisuje się typ *boss* i to w nim się będzie znajdował główny przeciwnik. Powyższy algorytm obrazuje niżej podany schemat:



Rysunek 17: Schemat generowania lochu

Po wygenerowaniu głównych pokoi zaczyna się generacja pokoi pobocznych. Założeniem jest aby od korytarza, powstałego z wygenerowanych głównych pokoi, odchodziły liczne odnogi tworzące ślepe zaułki w celu zwiększenia różnorodności i trudności. Generacja przebiega podobnie z tym, że jako punkt startowy wybiera się jeden z wygenerowanych wcześniej pokoi, a ilość pomieszczeń w odnodze jest losowana z pozostałej do wygenerowania ilości pokoi pobocznych. Jest on wybierany losowo z listy utworzonych pomieszczeń. Od niego zaczyna się generacja nowej odnogi, podobnie jak poprzednio wybiera się potencjalną pozycję na nowy pokój po czym sprawdza się z iloma pomieszczeniami sąsiadowałoby nowe pomieszczenie. Jednak w przypadku pokoi pobocznych dopuszczalna wartość sąsiadów wynosi 3. Jest to podyktowane tym, że w przypadku gdy nowo utworzone pomieszczenie nie będzie miało szans na wygenerowanie żadnego nowe pokoju, to proces generacji tej odnogi zostanie przerwany i zacznie się proces generacji kolejnej odnogi. Podobnie zresztą w momencie gdy w danej odnodze wygeneruje się liczba pokoi, którą wylosowano na początku generacji odnogi. Proces generacji kolejnej odnogi przebiega analogicznie. Wpierw zostaje wybrany pokój, z którego zacznie się generacja oraz ilość pokoi znajdujących się w odnodze i następują te same kroki co poprzednio. Cały proces generacji kończy się wraz z wygenerowaniem tylu pokoi ile zostało wyznaczonych pokoi pobocznych podczas podziału. Ostatni wygenerowany pokój staje się pomieszczeniem, w którym będzie znajdował się sklep. Ten algorytm dobrze obrazuje poniższy schemat:



Rysunek 18: Schemat generowania lochu

4.2.3 Generator pokoi

Kolejnym z generatorów jest generator pokoi. Jego zadaniem jest zapewnienie zróżnicowanego wypełnienia wygenerowanych pomieszczeń. Wypełnienie zależy od typu pokoju i ma na celu zapewnić świeżość rozgrywki. Na początku w każdym typie pokoi są generowane drzwi. Drzwi umożliwiają graczowi przemieszczanie się między kolejnymi pokojami. W momencie gdy gracz naciśnie przycisk akcji znajdując się przy drzwiach, skrypt przypisany drzwi przeniesie najpierw kamerę aby obejmowała następny pokój a potem gracza.

Potem następuje generacja przeszkód w pokojach. Przeszkody są generowane jedynie w pokojach, w których znajdują się przeciwnicy inni od *bossa*. Generacja polega na umieszczeniu po środku pokoju losowo wybranego prefabrykatu z listy prefabrykatów przeszkód. Następnie w pomieszczeniach tego typu są generowani przeciwnicy. Ponownie losowo wybierany jest przeciwnik z listy dostępnych przeciwników wraz z losowo wybraną nagrodą. Randomizowana jest również liczba przeciwników, którzy zostaną dodani do pokoju. Statystyki definiujące zagrożenie jakie będą stanowili wrogowie także podlega drobnej randomizacji, na której wpływ ma numer aktualnego poziomu. Po za tymi pokojami generowane jest wypełnienie dla pokoju ze sklepem oraz pokoju z *bossem*. W pokoju z głównym przeciwnikiem zachodzi jedynie trywialne losowanie bossa z listy oraz modyfikacja jego statystyk względem numeru poziomu. Z kolei w sklepie generator wybiera jaki drugi przedmiot będzie dostępny do kupienia - pierwszym zawsze jest serduszko zapewniające zdrowie. Modyfikuje również ich ceny w zależności od aktualnego poziomu.

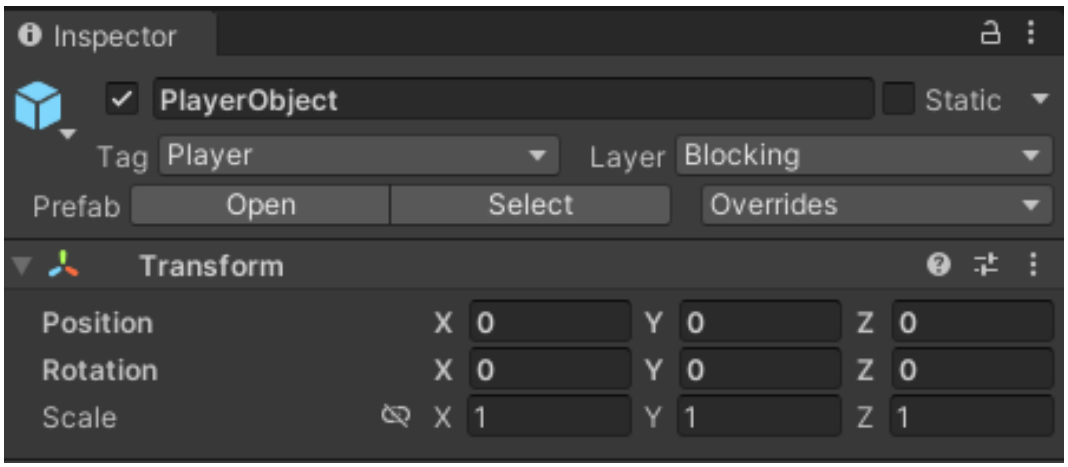
4.3 Wdrażanie kontroli gracza i poruszania się

Obiekt gracza podobnie jak każdy obiekt *GameObject* w Unity opiera się na wzorcu projektowym Jednostka-Komponent. Dodając komponenty do obiektu, implementuje się żądane funkcjonalności.

Gracz składa się z sześciu komponentów mających wpływ na jego kontrole: *Transform*, *Sprite Renderer*, *Animator*, *Box Collider 2D*, *Rigidbody 2D* oraz *Movement Controller* (Skrypt). Poniżej szczegółowo omówiono każdy z nich, co powinno czytającemu dać pełne zrozumienie nad sposobem kontroli i poruszania się gracza.

1. **Transform**

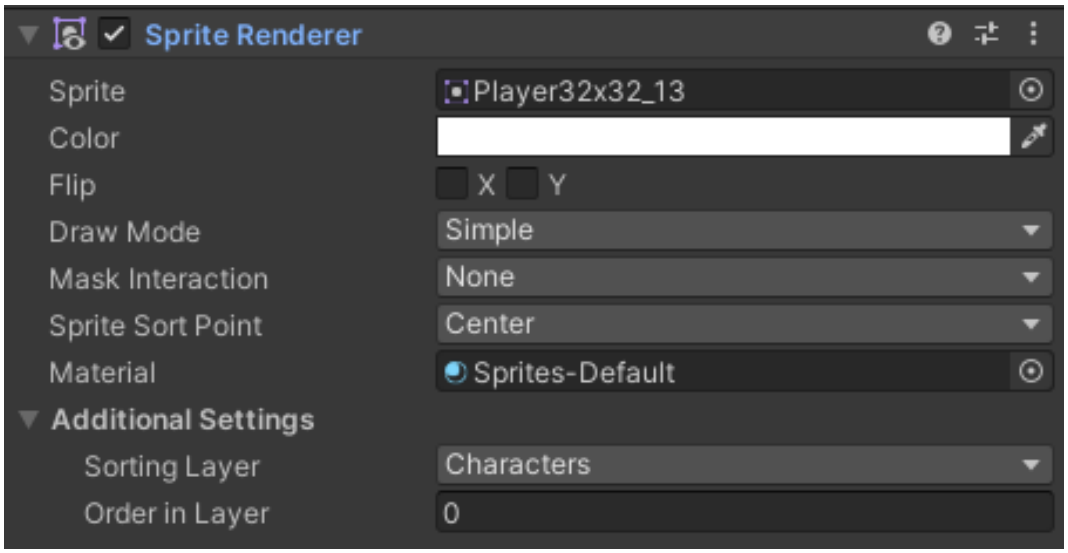
   Komponentem wspólnym dla wszystkich obiektów *GameObject* jest *Transform* (przekształcenie), który określa położenie, orientację i skalę obiektu umieszczonego na scenie [17]. Ten komponent jest używany do przemieszczania postaci gracza.



Rysunek 19: Zrzut ekranu komponentu *Transform*

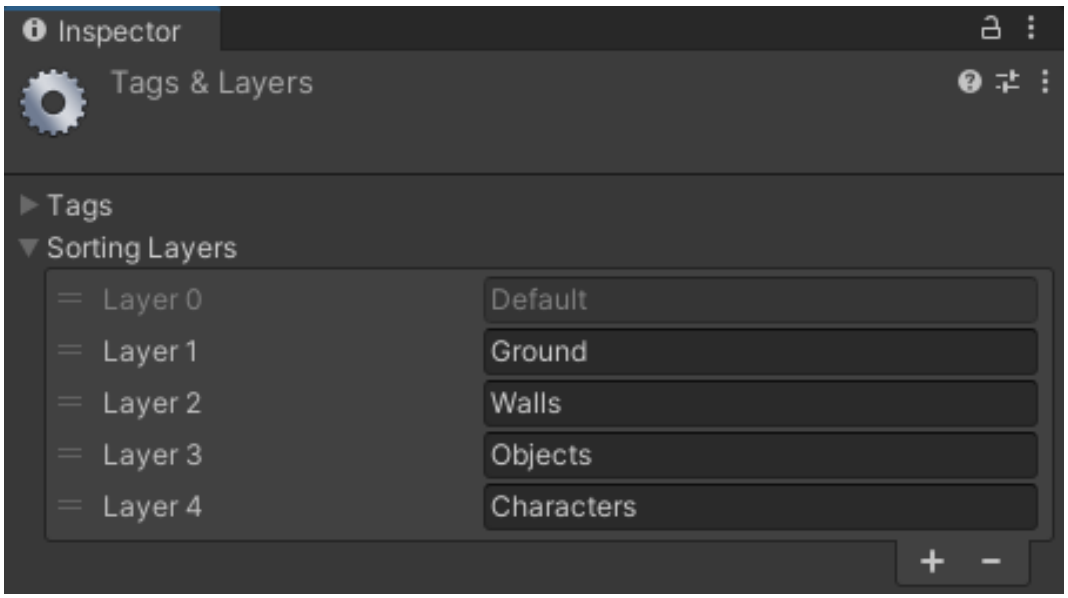
2. **Sprite Renderer**

Sprite Renderer to komponent, bez którego użytkownik nie wiedziałby gdzie jest bohater. Komponent ten odpowiada za wyświetlenie obrazu, dzięki czemu na ekranie jest widoczna postać gracza. Na Rysunku 20 poniżej należy zwrócić uwagę na dwa miejsca, w polu *Sprite* (dwuwymiarowy obraz) umieszczono domyślny obraz gracza (na którym stoi w miejscu), zaś w polu *Sorting Layer* (warstwa sortująca) ustawiono warstwę *Characters* (postacie).



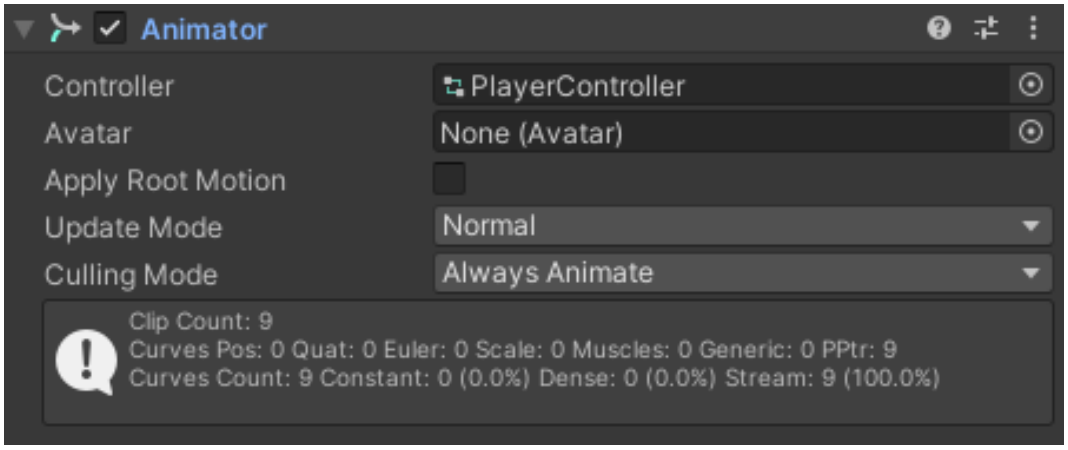
Rysunek 20: Zrzut ekranu komponentu *Sprite Renderer*

Warstwa sortująca określa kolejność, w jakiej silnik Unity ma rysować dwuwymiarowe duszki na ekranie. Na Rysunku 21 warstwa *Characters* znajduje się na samym końcu listy, co oznacza że postacie będą wyświetlane na tle innych obiektów, na przykład ziemi.

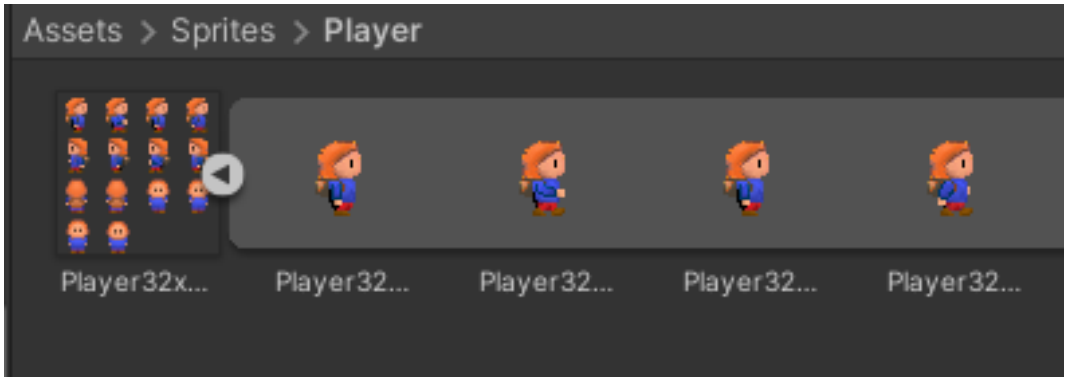


Rysunek 21: Zrzut ekranu Warstw sortujących

3. **Animator**
   Zadaniem *Animatora* jest odtwarzanie animacji za pomocą kontrolera. Szybkie wyświetlanie sekwencji duszków wywołuje wrażenie ruchu, na przykład chodzenia czy walki. Kontroler służy do odtwarzania i kontrolowania animacji.



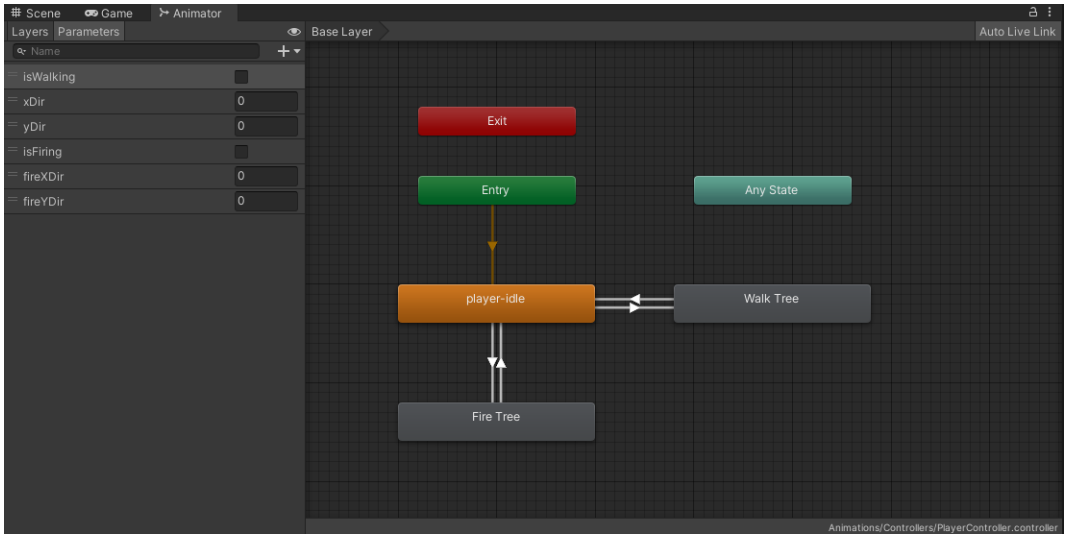
Rysunek 22: Zrzut ekranu komponentu Animatora



Rysunek 23: Zrzut ekranu duszków składających się na animację chodzenia w prawo

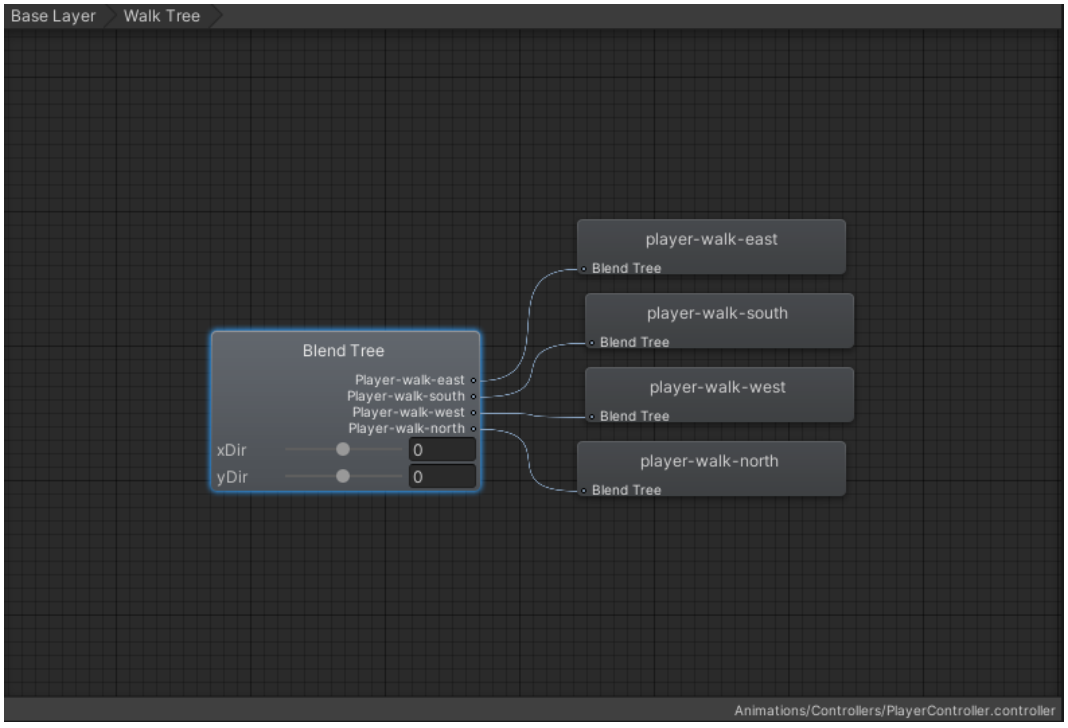
Rysunek 24 przedstawia wnętrze Kontrolera Gracza, czyli maszynę stanów. Maszyna ta decyduje o tym, które klipy animacyjne mają być odtwarzane w zależności od stanu obiektu gracza. [17] Maszyna bohatera gry składa się z dwóch drzew mieszających *Blend Tree*. Jedno odpowiada za animację poruszania się, a drugie za animację strzelania z procy.

Na dodatek maszyna zawiera różne parametry, którymi można zarządzać przejścia pomiędzy stanami. W zależności od wartości parametrów gracz może zmienić kierunek chodzenia albo stan, na przykład z chodzenia na strzelanie.



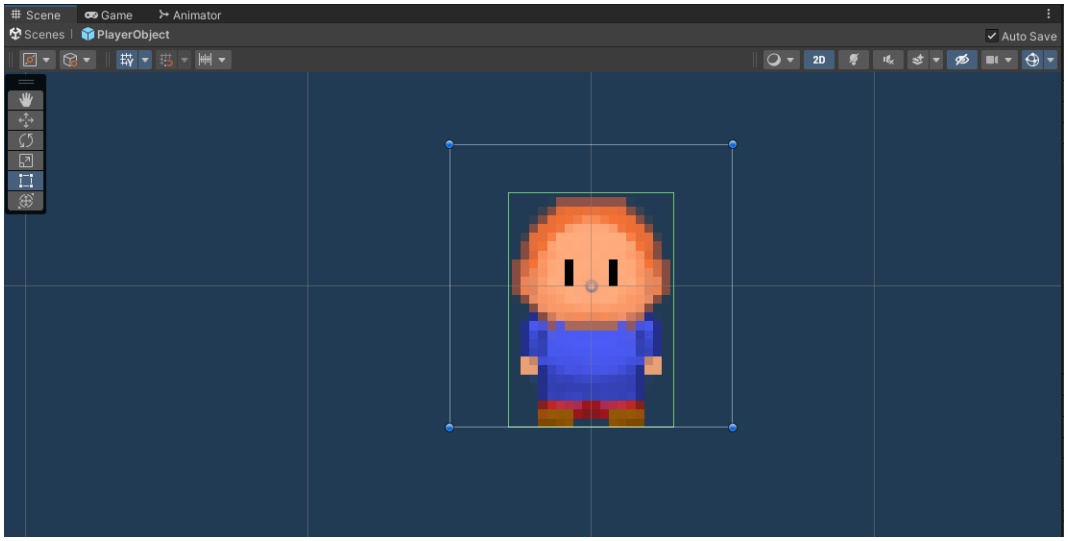
Rysunek 24: Zrzut ekranu Maszyny stanów kontrolera gracza

Wnętrze drzewa mieszającego chodzenia zawiera animację poruszania się (na północ, południe, wschód i zachód). Aby użyć drzewa mieszającego, należy napisać skrypt, który zostanie omówiony w kolejnych punktach.



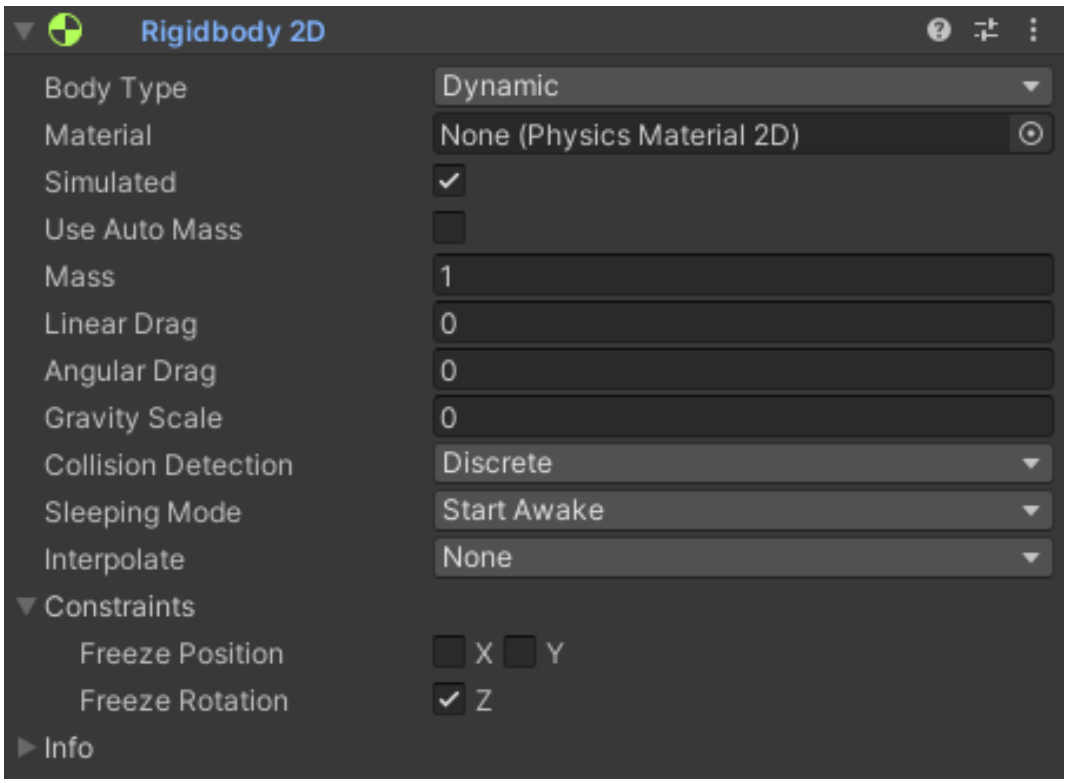
Rysunek 25: Zrzut ekranu wnętrza drzewa chodzenia

4. **Box Collider 2D**
   Chodząc po świecie pełnym różnych obiektów, ścian i przeciwników gracz musi wiedzieć czym jest kolizja. W tym celu wykorzystano komponent *Box Collider 2D*. Zderzacze mogą mieć dowolny kształt, jednak ze względów obliczeniowych zwykle nie nadaje się obiektowi ściśle określonego kształtu. Na Rysunku 26 zielona linia wokół bohatera reprezentuje ów zderzacz.



Rysunek 26: Zrzut ekranu zderzacza gracza

5. **Rigidbody 2D**
   Kolejnym kluczowym komponentem obiektu gracza jest *Rigidbody 2D*. Wykorzystywany przez silnik Unity do interakcji z obiektem *GameObject*. Komponent służy na przykład do wywierania siły grawitacji na dany obiekt.[17] Rysunek 27 przedstawia funkcje komponentu Rigidbody 2D. Na potrzeby gry ustawiono *Body Type* (typ ciała) na *Dynamic* (dynamiczne), dzięki temu obiekt gracza będzie wchodził w interakcje i kolidował z innymi obiektami w grze.

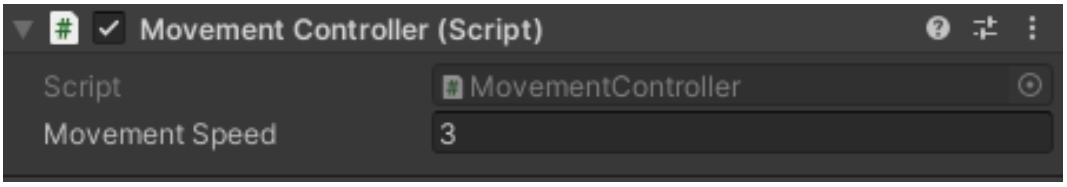


Rysunek 27: Zrzut ekranu komponentu *Rigidbody 2D*

6. **Movement Controller (Skrypt)**
Komponent *Movement Controller* odpowiada za dwie rzeczy, przesuwa obiekt gracza oraz aktualizuje stan gracza. Skrypt odwołuje się do komponentu Rigidbody 2D obiektu gracza i przy pomocy interfejsu *Input* pobiera dane z klawiatury, dzięki czemu gracz porusza się po świecie.

```
private void MoveCharacter()
{
    movement.x = Input.GetAxisRaw("Horizontal");
    movement.y = Input.GetAxisRaw("Vertical");
    movement.Normalize();
    rb2D.velocity = movement * movementSpeed;
}
```

Zmienne publiczne pojawiają się w panelu *Inspector* we właściwościach obiektu *GameObject*, do którego przypisano skrypt.



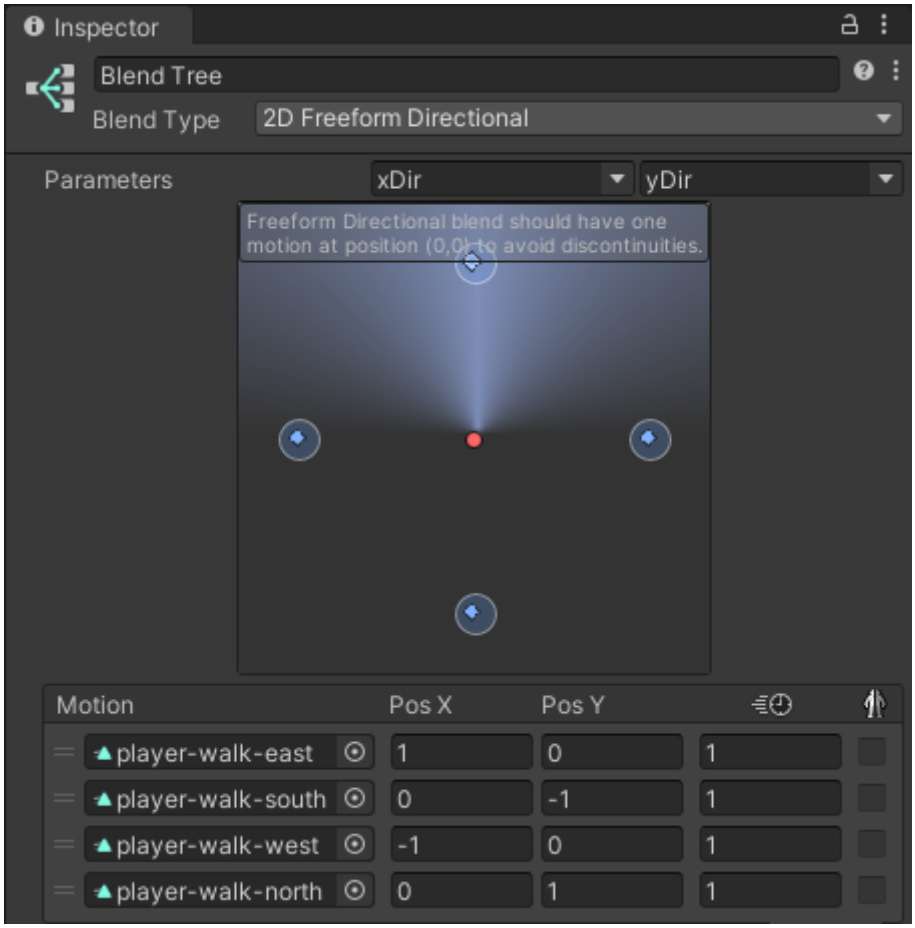
Rysunek 28: Zrzut ekranu komponentu *Movement Controller*

Dodatkowo w każdej klatce uruchamia funkcję aktualizująca stan i przy pomocy komponentu Animator przechodzi między stanami gracza, jak było pokazane na Rysunku 25. Jeżeli gracz nie porusza się, parametr *isWalking* ustawiany jest na fałsz, co powoduje przełączenie na stan stania. W innym przypadku oznacza to, że gracz się porusza i stan ponownie ulega zmianie.

```
private void UpdateState()
{
    if (Mathf.Approximately(movement.x, 0) &&
        Mathf.Approximately(movement.y, 0))
    {
        animator.SetBool("isWalking", false);
    }
    else
    {
        animator.SetBool("isWalking", true);
    }

    animator.SetFloat("xDir", movement.x);
    animator.SetFloat("yDir", movement.y);
}
```

W zależności od kierunku, w którym porusza się gracz, maszyna stanów przechodzi między animacjami chodzenia w różnych kierunkach. Najlepiej ukazuje to tabela na Rysunku 29. Parametry *xDir* i *yDir* odpowiadają za przejścia animacji.

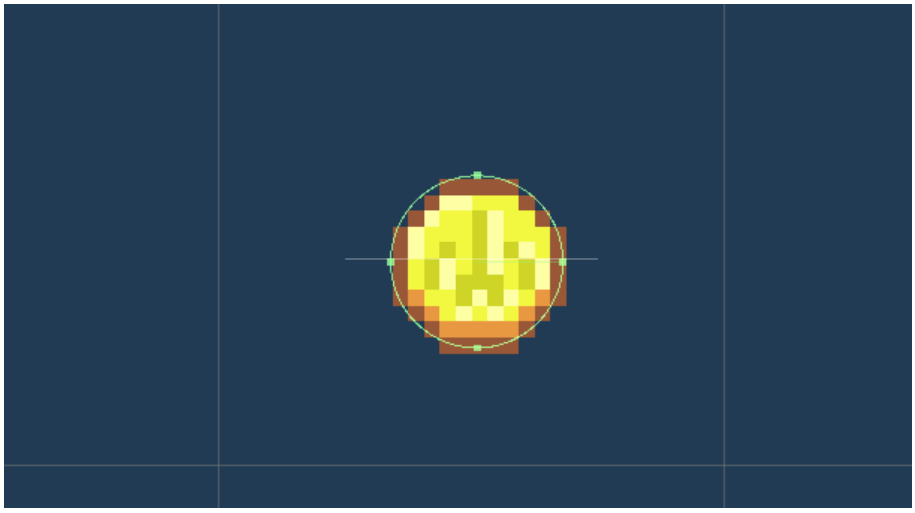


Rysunek 29: Zrzut ekranu stanów chodzenia

4.4 Projektowanie i wdrażanie zasobów gry

W czasie eksploracji lochów gracz może natrafić na różne przedmioty. Niektóre z nich gracz może przechować i skorzystać z nich w dalszej rozgrywce. Inne zużywa od razu po ich zebraniu. Zasoby to kolejne obiekty *GameObject*. Każdy z nich posiada pięć tych samych komponentów: *Transform*, *Sprite Renderer*, *Animator*, *Circle Collider 2D* oraz skrypt *Consumable*.

Komponent *Transform* umożliwia ustalić położenie zasobów. *Sprite Renderer* wyświetla grafikę, za to *Animator* wprawia w ruch przedmioty. Każdy z obiektów posiada wspólną animację kręcenia się wokół własnej osi. Zderzacz *Circle Collider 2D* wraz z włączoną opcją *Is Trigger* (wyzwalacz) jest wykorzystywany przez gracza, kiedy ten chce zebrać dany przedmiot.

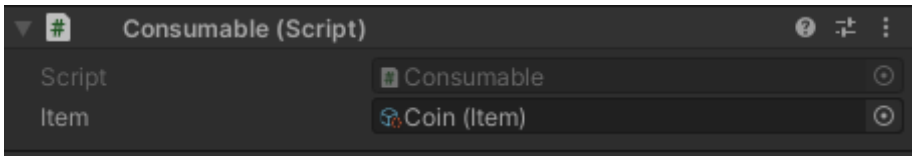


Rysunek 30: Zrzut ekranu zderzacza monety

Najważniejszym elementem każdego zasobu jest skrypt *Consumable* (zużywalne). Przechowuje on referencję do obiektu programowalnego *Item*, który jest kluczowym elementem dla tych obiektów. Pojawia się pytanie, czym jest obiekt programowalny w silniku Unity? Tego rodzaju obiekt można traktować jako zasobnik współdzielonych danych. [17]

Obiekt programowalny posiada dwie ważne cechy:

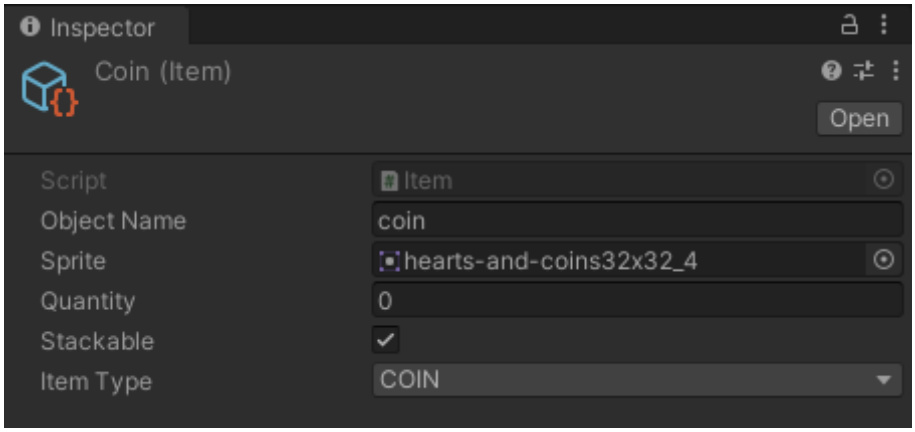
- Zajmuje mniej pamięci w komputerze, ponieważ w grze korzysta się z referencji do jednej instancji obiektu. Dzięki temu nie tworzy się licznych kopii wszystkich właściwości obiektu.
- Pozwala korzystać z predefiniowanych, zewnętrznych danych.



Rysunek 31: Zrzut ekranu komponentu *Consumable*

Aby lepiej zrozumieć pierwszą cechę na Rysunku 32 pokazano wnętrze klasy obiektu programowalnego *Item*, przeznaczonego do przechowywania danych o obiektach, które będzie mógł zbierać gracz. Jeżeli byłyby tworzone kolejne instancje prefabrykatu np. monety, za każdym razem powielane byłyby jej właściwości. W efekcie gra zajmowałaby dużo pamięci.

Jeśli prefabrykat monety będzie zawierał programowalny obiekt, wtedy każda instancja będzie zawierała referencję do jednych i tych samych danych. W efekcie nawet jeżeli zostanie utworzonych wiele instancji prefabrykatu monety, ilość pamięci nie zmieni się.



Rysunek 32: Zrzut ekranu obiektu programowalnego monety

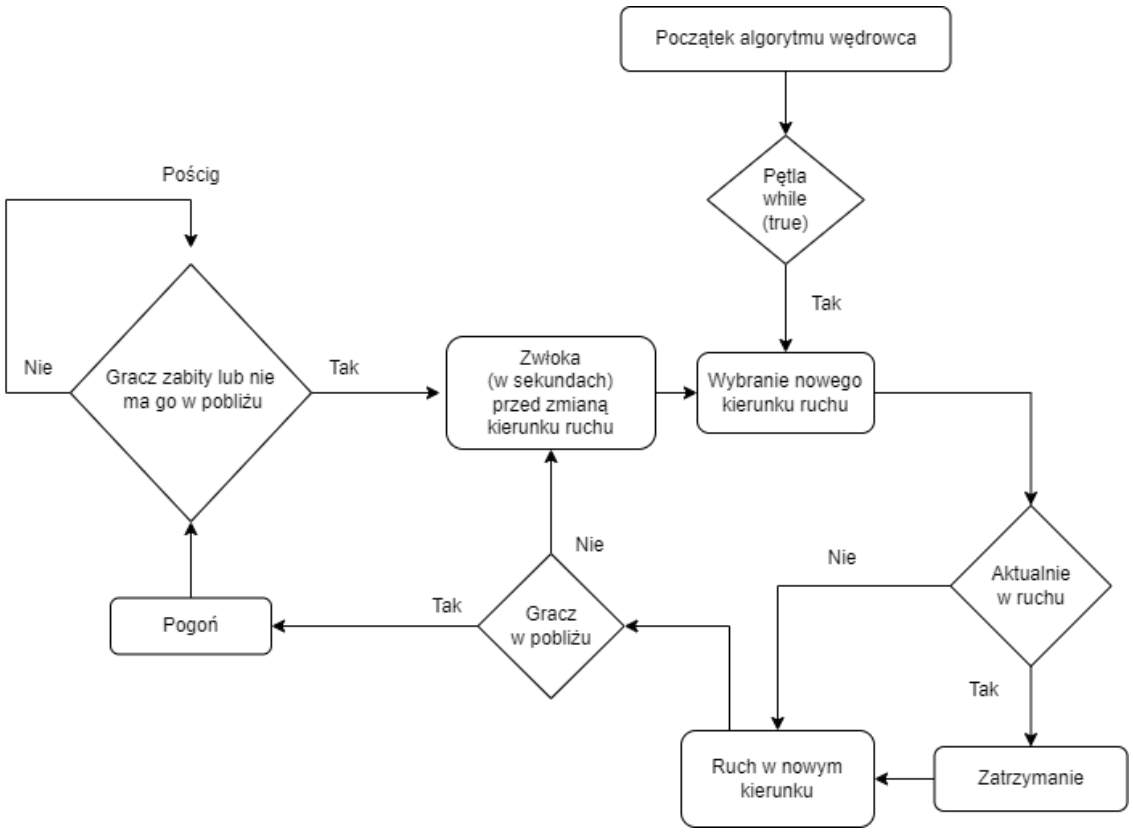
Klasa obiektu programowalnego (*Item*) dziedziczy cechy klasy *ScriptableObject*. Takiego obiektu nie można przypisać do obiektu *GameObject*. Zamiast tego w klasie pochodnej od *MonoBehaviour* (*MonoBehaviour* jest klasą bazową, z której wywodzi się każdy skrypt silnika Unity. [18]), jak *Consumable* definiuje się referencję do obiektu programowalnego.

4.5 Wdrażanie sztucznej inteligencji w grze i zachowania wrogów

Gra oferuje kilka rodzajów przeciwników. Każdy z nich posiada swoje cechy specjalne. Niektórzy walczą wręcz inni z dystansu. Wrogowie różnią się nie tylko wyglądem, ale też siłą, prędkością i wytrzymałością na ataki gracza. W grze zaimplementowano sztuczną inteligencję do sterowania przeciwników ścigających postać gracza.

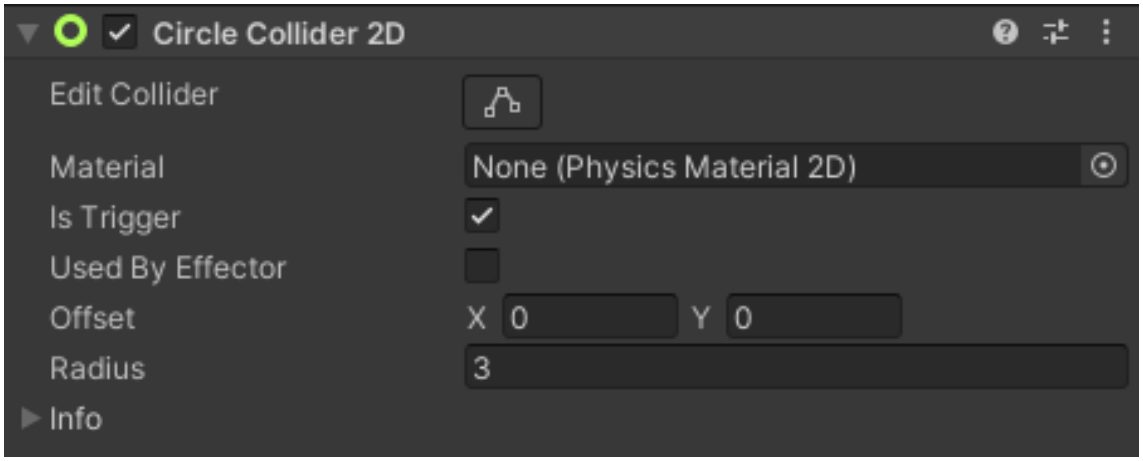
4.5.1 Algorytm wędrowca

Algorytm ten jest wspólny dla wszystkich przeciwników. Wykorzystuje koprocedury i steruje losowymi ruchami przeciwnika na scenie. Kiedy gracz zbliży się do przeciwnika, ten zacznie go ścigać. Pościg trwa dopóki gracz nie ucieknie daleko, zabije przeciwnika lub sam zostanie zabity. Rysunek 33 przedstawia schemat algorytmu wędrowca.

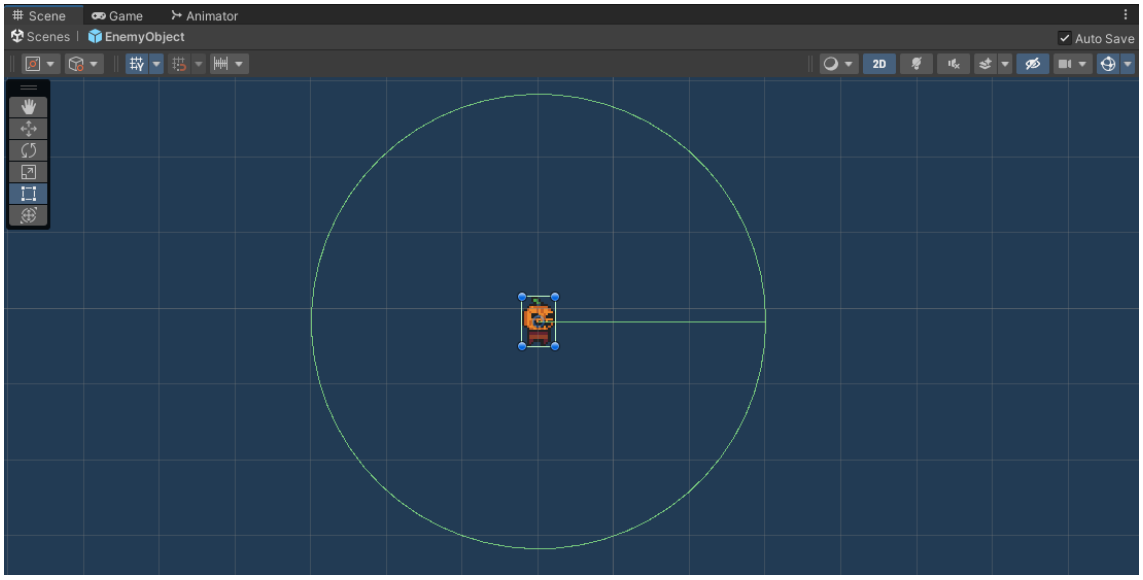


Rysunek 33: Schemat algorytmu wędrowca

Każdy przeciwnik posiada zderzacz *Circle Collider 2D*, który określa jego 'zasięg wzroku'. Oznacza to, że gdy gracz przekroczy linię zderzacza, przeciwnik 'zobaczy' gracza. Jednak żeby mechanizm działał poprawnie, należy zaznaczyć opcję *Is Trigger* w komponencie zderzacza. Teraz komponent działa jako wyzwalacz. Na dodatek średnica zderzacza jest zaimplementowana jako zmienna publiczna, co daje możliwość konfiguracji pola widzenia.



Rysunek 34: Zrzut ekranu komponentu *Circle Collider 2D*



Rysunek 35: Zrzut ekranu przeciwnika ze zderzaczem

Cały algorytm jest zaimplementowany w skrypcie *Wander*, który jest przypisany do obiektu przeciwnika. Komponent daje nam możliwość zmiany prędkości zarówno ścigania gracza jak i wędrowania oraz częstotliwości z jaką przeciwnik zmienia kierunek.

Rysunek 36: Zrzut ekranu komponentu *Wander* (Skrypt)

4.5.2 Wybór nowego punktu docelowego

Warto zwrócić uwagę na metodę *ChooseNewEndpoint()*. Zadaniem metody jest znalezienie nowego punktu docelowego dla przeciwnika. Najpierw następuje losowanie wartości z przedziału od 0 do 360. Wartość ta zostaje dodana do bieżącego kąta kierunku ruchu. Metoda *Mathf.Repeat()* modyfikuje bieżący kąt tak, aby jego wartość znajdywała się w przedziale od 0 do 360. Ostatecznie liczba stopni jest zmieniana na wektor i dodawana do właściwości *endPostion* czyli nowego punktu docelowego.

```
void ChooseNewEndpoint()
{
    if (RmController.IsPlayerPresent)
    {
            currentAngle += Random.Range(0, 360);
            currentAngle = Mathf.Repeat(currentAngle, 360);
            endPosition += Vector3FromAngle(currentAngle);
    }
}
```

4.5.3 Wróg dystansowy

Istnieje jeszcze jeden rodzaj sztucznej inteligencji w projekcie. Dotyczy on przeciwników z możliwością ataku z dalekiego zasięgu. Taki przeciwnik posiada kilka dodatkowych właściwości jak, dystans do strzelania (jeżeli gracz będzie w odpowiedniej odległości od przeciwnika ten zacznie do niego strzelać), częstotliwość z jaką strzela oraz cel, czyli głównego bohatera gry. Oprócz tego do takiego wroga przypisany jest specjalny obiekt amunicji.

Algorytm wroga dystansowego w każdej klatce szuka komponentu *Transform* gracza, wykorzystując do tego *Tag* (znacznik) przypisany do obiektu gracza:

```
private void GetTarget()
{
    if (GameObject.FindGameObjectWithTag("Player"))
    {
       target = GameObject.FindGameObjectWithTag("Player").transform;
    }
}
```

Jeżeli przeciwnik namierzył swój cel i gracz znajduje się w odpowiednim dystansie do strzelania dla wroga, uruchamiana jest metoda *Shoot()*. Jej zadaniem jest wystrzelenie pocisku w miejsce gdzie stał gracz w chwili uruchomienia metody. Wpierw funkcja inicjalizuje obiekty pocisku przeciwnika. Następnie oblicza kierunek, w którym ma być wystrzelony pocisk i strzela. Na Rysunku 37 widać dwa żółte pociski wystrzelone w kierunku gracza.

Rysunek 37: Zrzut ekranu ukazujący mechanizm strzelania przeciwnika

5 Podsumowanie

5.1 Umiejętności, które rozwija produkcja

Można wyróżnić parę umiejętności gracza, do których rozwoju może się przyczynić opisywana w tej pracy gra. Jednym z nich jest pamięć mięśniowa połączona ze zwinnością. Gracz musi reagować szybko i pewnie na pojawiające się zagrożenia, a coraz to bardziej narastający poziom trudności stymuluje ciągłe rozwijanie się w tym zakresie.

Kolejno można wymienić umiejętność strategicznego planowania. Jest ona rozwijana podczas podejmowania decyzji przykładowo w sklepie. Użytkownik ma do wyboru dwa przedmioty, musi ocenić który z nich bardziej mu pomoże. Może również zaryzykować i nic nie kupić w celu zaoszczędzenia pieniędzy na następny poziom, gdzie nabycie pomocy może się okazać bardziej potrzebne.

Ostatecznie, również umiejętność taktycznego myślenia jest stymulowana. Podczas starć z mniejszymi przeciwnikami jak i głównym antagonistą kluczowa do zwycięstwa będzie odpowiednia taktyka jaką gracz obierze w starciu z tymi przeciwnościami.

5.2 Przyszłość i potencjał projektu

Gra ma wielki potencjał rozwoju ze względu na otwartą budowę oraz proceduralne generowanie zawartości mapy. Sama rozgrywka stanowi bardzo wciągający format, do którego przekonane jest wielu graczy na całym świecie, co udowadnia mnogość produkcji *rougelike* na rynku jak i wszelakich pokrewnych gatunków. Prostota zasad oraz szerokie pole na którym gracz może rozwijać swoje umiejętności w grze stanowi ogromny atut produkcji. W przyszłości projekt byłby rozwijany więc na następujących płaszczyznach:

1. **Konwersja na inne platformy**
   Projekt zostałby przekonwertowany na systemy operacyjne komputerów desktopowych inne niż Windows oraz również na systemy mobilne. Dzięki specyfikacji narzędzia Unity jest to znacząco ułatwione, gdyż najczęściej wystarczy zmienić docelową platformę podczas fazy budowania projektu.

2. **Rozbudowanie mechanik**
   Rozbudowanie mechanik polegałoby na dodaniu kolejnych mechanik walki oraz eksploracji. Możliwe by było dodanie pobocznych ataków jak i bardziej złożonego systemu przeciwników. Rozwojowi również uległyby przedmioty, zarówno jeżeli chodzi o różnorodność jak i o ich wpływ na rozgrywkę. Rozszerzono by także listę przeciwników. Poszerzona zostałaby o kolejne typy, zarówno podstawowych wrogów jak i bossów.

3. **Generacja poziomów**
   Generacje poziomów można byłoby rozszerzyć o zróżnicowane rozmiary i kształty pokoi. Wymagałoby to rozwinięcia algorytmu generowania, jednakże kod sam w sobie uwzględnia rozmiar pokoju jako zmienną co ułatwia realizację tego celu rozwojowego.

4. **Poboczne funkcjonalności**
   Grę da się również rozwinąć pod względem takich funkcjonalności jak zapis i odczyt rozgrywki. System serializacji danych mógłby stanowić funkcjonalne rozwinięcie formuły, jednak stanowiłby małą rewolucję jeżeli chodzi o zasady gatunkowe, która jednak stawia na ciągłość rozgrywki.

5. **Fabuła**
   Rozwinięcie produkcji o fabułę stanowi jasny kierunek w którym może się potoczyć ewolucja gry. Zawarcie w poziomach elementów otoczenia, opisów czy innych szczątkowych informacji na temat świata przedstawionego mogło by stanowić motor napędowy dla graczy którzy by chcieli poznać tajemnice które budują świat gry.

6. **Dźwięk oraz muzyka**
   Ostatecznie należy wspomnieć o dodaniu efektów dźwiękowych jak i muzyki która budowałaby zarówno klimat podczas przemierzania złowieszczych pokoi jak i uczestniczyła w budowaniu świata przedstawionego.

5.3 Wnioski

Wynikiem przeprowadzonych prac jest gra komputerowa 2D z gatunku *rougelike*. W ramach pracy przeprowadzono kompleksowy proces projektowania i implementacji gry. Wykonana produkcja stanowi spójną całość i każdy z zaplanowanych elementów został zaimplementowany według założeń. Udało się utworzyć odporny na błędy algorytm generacji poziomu jak i sztuczną inteligencję przeciwników. Wszystkie składowe współgrają ze sobą tworząc wciągającą, niemonotonną i kompletną produkcję, która stanowi sama w sobie pełnoprawne dzieło jak i może być punktem wyjścia do bardziej rozbudowanego tworu z przyszłością na wejście i zaistnienie w rozległym świecie gier wideo.

Bibliografia

[1] Global Games Market Report New Zoo. 2022. URL: https://newzoo.com/globalgamesreport.

[2] David L Craddock. *Dungeon Hacks: How NetHack, Angband, and Other Rougelikes Changed the Course of Video Games*. CRC Press, 2021.

[3] URL: https://en.wikipedia.org/wiki/Sprite_(computer_graphics).

[4] Gary Gygax i Dave Arneson. *dungeons & dragons*. T. 19. Tactical Studies Rules Lake Geneva, WI, 1974.

[5] International Roguelike Development Conference. 2008. URL: https://www.roguebasin.com/index.php?title=Berlin_Interpretation.

[6] Jon Brodkin. "How Unity3D Became a Game-Development Beast". W: *Dostupno: https://insights. dice. com/2013/06/03/how-unity3d-become-a-game-development-beast* (2013).

[7] David Helgason. URL: http://forum.unity3d.com/threads/56-Unity-1-0-isshipping.

[8] Unity Technologies. URL: https://unity.com/products/unity-platform.

[9] Andrew Sanders. *An introduction to Unreal engine 4*. CRC Press, 2016.

[10] Ernest Adams. "Projektowanie gier". W: *Podstawy. Helion, Gliwice* (2011).

[11] URL: https://en.wikipedia.org/wiki/Compulsion%5C_loop.

[12] URL: https://en.wikipedia.org/wiki/Boss_(video_games).

[13] URL: https://en.wikipedia.org/wiki/Replay_value.

[14] URL: https://en.wikipedia.org/wiki/Bushnell%5C%27s_Law.

[15] URL: https://creativecommons.org/share-your-work/public-domain/cc0/.

[16] Unity Technologies. URL: https://docs.unity3d.com/hub/manual/index.html.

[17] Jared Halpern i Halpern. *Developing 2D Games with Unity*. Springer, 2019.

[18] Unity Technologies. URL: https://docs.unity3d.com/ScriptReference/MonoBehaviour.html.