

W A R S Z A W S K A
W Y Ż S Z A S Z K O Ł A I N F O R M A T Y K I

P R A C A D Y P L O M O W A
S T U D I A P I E R W S Z E G O S T O P N I A

Weronika Bednarska

Numer albumu 8819

**Analiza, projekt i implementacja wybranego algorytmu
sztucznej inteligencji w grze komputerowej z wykorzystaniem
silnika Unity.**

Promotor:

mgr inż. Waldemar Ptasznik-Kisieliński

Praca spełnia wymagania stawiane pracom dyplomowym na studiach pierwszego stopnia.

W A R S Z A W A 2022

Spis treści

Wykorzystane skróty i oznaczenia:	2
1. Wstęp	3
1.1 Cel pracy	3
1.2 Ograniczenia i problemy	3
2. Analiza dziedziny	4
2.1 Gry komputerowe	4
2.2 Gry typu FPS	5
2.3 Silnik Unity3D	6
2.4 Zastosowanie AI w grach	7
2.4.1 Automat skończony	8
2.4.2 Drzewo behawioralne	10
3. Analiza systemu	12
3.1 Ogólne założenia projektu	13
3.2 Specyfikacja wymagań	14
3.2.1 Wymagania funkcjonalne	14
3.2.2 Wymagania pozafunkcjonalne	17
3.3 Modelowanie danych	18
3.4 Modelowanie przypadków użycia	18
3.4.2 Logowanie użytkownika	20
3.4.3 Walka gracza z przeciwnikami typu Arachnid i Mutant Podstawowy	20
3.4.4 Walka gracza z przeciwnikiem typu Mutant Kryształowy	22
3.4.5 Mutant Kryształowy – leczenie	23
3.4.6 Zakończenie rozgrywki	24
4. Wybór metodyki	25
5. Projekt	25
5.1 Narzędzia	26
5.2 Projekt bazy	27
5.3 Wykorzystanie funkcjonalności silnika Unity	28
5.3.1 Asset Store oraz Package Manager	28
5.3.2 Sprite Editor	29
5.3.3 Animator Controller	30
5.3.4 System nawigacji oraz Pathfinding	32
5.3.5 Skrypty	33
5.4 PandaBT	34

5.4.1 Wykorzystanie drzew behawioralnych	34
5.5 Mechanika gry	39
5.5.1 Przeciwnicy	39
5.5.2 Gracz oraz system walki	42
5.5.3 Środowisko	42
5.6 Interfejs użytkownika	43
5.6.1 Ekran logowania i rejestracji.....	43
5.6.2 Menu główne	45
5.6.3 Interfejs wyświetlany w trakcie gry	46
5.7 Układ katalogów	49
6. Implementacja	50
6.1 Wykorzystane assety	50
6.2 Stworzenie obsługi gracza i prototypowanie	51
6.3 Implementacja przeciwników	54
6.4 Implementacja z wykorzystaniem drzewa behawioralnego	56
6.5 Świat gry oraz system misji.....	63
7. Testowanie	65
8. Kierunki dalszego rozwoju	65
9. Podsumowanie	66
WYKAZ LITERATURY	67

Wykorzystane skróty i oznaczenia:

Termin	Znaczenie
AI	Sztuczna inteligencja (ang. Artificial intelligence)
2D	Dwuwymiarowy (ang. two-dimensional)
3D	Trójwymiarowy (ang. three-dimensional)
FSM	Automat skończony (ang. finite state machine)
NavMesh	Siatka nawigacji modelu (ang. navigation mesh)
FSM	Automat Stanów Skończonych (ang. Finite State Machine)
Prefab	Prefabrykat

1. Wstęp

Rozdział ten jest wprowadzeniem do tematyki gier komputerowych wykorzystujących algorytmy AI.

1.1 Cel pracy

Celem pracy jest zarówno zaprezentowanie procesu implementacji gry 3D przy użyciu silnika Unity dla systemu operacyjnego Windows, jak i przedstawienie użytych algorytmów sztucznej inteligencji. Praca ta demonstruje wybrane elementy wykorzystanych narzędzi oraz prezentuje sposoby ich użycia.

Algorytmy wykorzystywane w nowoczesnym rozwoju gier obejmują uczenie maszynowe, odpowiednio dostosowane reakcje i podejmowanie decyzji. Krzywa uczenia algorytmów AI jest bardzo szeroka, jednak główny temat będzie obejmował stany maszynowe oraz drzewo zachowań, ukazując przy tym jak wpływają one na rozwój gry.

1.2 Ograniczenia i problemy

Aby stworzyć grę wykonaną przez jedną osobę w ograniczonym przedziale czasowym należy wprowadzić pewne limitacje. Biorąc pod uwagę fakt, że projekt ma posiadać różne obszary tj. wizualnie zadawalające efekty graficzne, sztuczna inteligencja, dźwięk, rozbudowane funkcje rozgrywki, postanowiono używać głównie bezpłatnych modeli ze źródeł zewnętrznych, zamiast modeli projektowanych na własną rękę. Gotowe modele będą jedynie poprawiane i dostosowywane do potrzeb gry. Ogranicza to ilość modeli, które posiadamy do dyspozycji oraz spójność wszystkich elementów nie zostaje zachowana w scenie gry.

Podczas tworzenia gier na swojej drodze napotykamy także pewne problemy. Można podzielić je na kilka części:

Silnik gry. Jak ustrukturyzować kod, aby był on otwarty na zmiany w przyszłości?

Grafika. Jak skutecznie zwizualizować efekty graficzne? Podczas tworzenia gry pojawia się konflikt pomiędzy realizmem w wizualizacji, a wizualizacjami będącymi przyjemniejszymi dla oka użytkownika oraz ograniczonymi możliwościami zaprojektowania animacji.

Sztuczna inteligencja i funkcjonalność. Jak odpowiednio wykorzystać AI w grze? W jaki sposób dobrać do interesujących nas funkcji metodę ich reprezentacji?

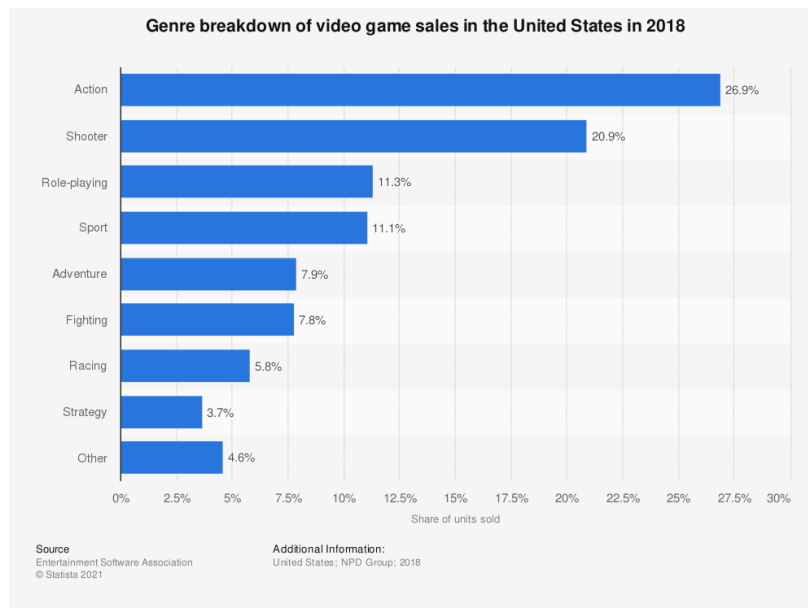
2. Analiza dziedziny

Rozdział przedstawia zarówno wprowadzenie do gier komputerowych, jak i wprowadzenie do gier typu FPS oraz zawiera zestawienie dostępnych silników do gier.

2.1 Gry komputerowe

Gry komputerowe stanowią oprogramowanie bądź formę programu umożliwiającego użytkownikowi wykonywanie zaprojektowanych przez jego autorów celów. Twórcy gier starają się dostarczyć graczom rozrywki lub narzędzi do zdobywania wiedzy i poszerzania horyzontów. Zawartość często imituje gry tradycyjne lub odpowiada za różnego rodzaju symulacje, wykorzystując do tego formę audiowizualną. Decyzje użytkownika stają się kluczowe w przebiegu rozgrywki, budując indywidualną formę narracji przy zapewnieniu interaktywności.

W dzisiejszych czasach gry stały się nieodłączną częścią sektora rozrywki. Coraz większa liczba gier obecnie staje się darmowa, umożliwiając tym samym ich znaczne rozpowszechnienie.



Rys. 2.1 Udział poszczególnych gatunków gier w sprzedaży wyrażony w procentach (źródło: Statistica)

2.2 Gry typu FPS

Gra FPS jest strzelanką pierwszoosobową z wykorzystaniem broni palnej. Gracz porusza się po trójwymiarowym świecie, patrząc oczami głównego bohatera (tj. perspektywa z pierwszej osoby), choć niektóre wykorzystują kamerę podążającą za postacią (TPS) – posiadają one podobną rozgrywkę i również nawiązują do gier akcji, dlatego są ogólnie uznane za ten sam gatunek. Gry FPS należą do jednych z najpopularniejszych gatunków gier komputerowych dla różnych platform. Ze względu na wymagania oraz moc obliczeniową potrzebną do stworzenia trójwymiarowego środowiska, strzelanki pierwszoosobowe są często uznawane za postęp technologiczny zarówno w konsolach, jak i komputerach osobistych. Poza celem rozrywkowym, gry tego typu znalazły zastosowanie w różnych rodzajach symulacjach i szkoleniach. Jako przykład może posłużyć gra taktyczna stworzona przez Armię Stanów Zjednoczonych „America’s Army”, której celem było zapewnienie realistycznej symulacji dla rekrutów, aby mieli oni okazję zapoznania się z procedurami armii. Grę także zaprojektowano tak, aby miała zachęcić potencjalnych kandydatów do zaciągnięcia się do armii. Gry typu

FPS często znajdują zastosowania jako podmiot badawczy ze względu na znaczący wpływ na gracza oraz jego doświadczenia.

2.3 Silnik Unity3D

Ogólnie silnik gry stanowi najważniejszą i podstawową część kodu wykorzystywaną w grze, połączonej razem z zintegrowanym środowiskiem programistycznym. Ma na celu zapewnienie interakcji pomiędzy zarówno użytkownikiem i grą, a także jej poszczególnymi elementami.

Unity jest silnikiem dla gier zarówno trójwymiarowych jak i dwuwymiarowych, wraz z zintegrowanym środowiskiem deweloperskim stworzonym przez Unity Technologies. Stosowany jest do tworzenia gier, animacji lub wizualizacji na różne platformy tj. komputery, konsole, urządzenia mobilne czy przeglądarki internetowe.

Silnik ten stanowi doskonały punkt wejściowy do tworzenia i projektowania własnych gier. Darmowa wersja udostępnia użytkownikom wygodny interfejs oraz stwarza możliwości nauki i rozwoju. To co wyróżnia ten silnik to możliwość wprowadzenia nawet osoby początkującej do zaawansowanego świata gier.

Tabela 2.1

Porównanie Unity wraz z dostępnymi już silnikami (źródło własne).

Nazwa	Zalety	Wady
Unity 3D	<ul style="list-style-type: none">- darmowy do użytku personalnego- łatwy, bezpośredni dostęp do Asset Store- efektywny przy renderowaniu 2D i 3D- umożliwia wykorzystanie języka C#, który posiada lepszą krzywą uczenia niż np. C++- dostępny dla większości platform	<ul style="list-style-type: none">- zakup niektórych produktów z Asset Store może okazać się kosztowny- brak wolnego dostępu do kodu źródłowego- wolniejsze renderowanie

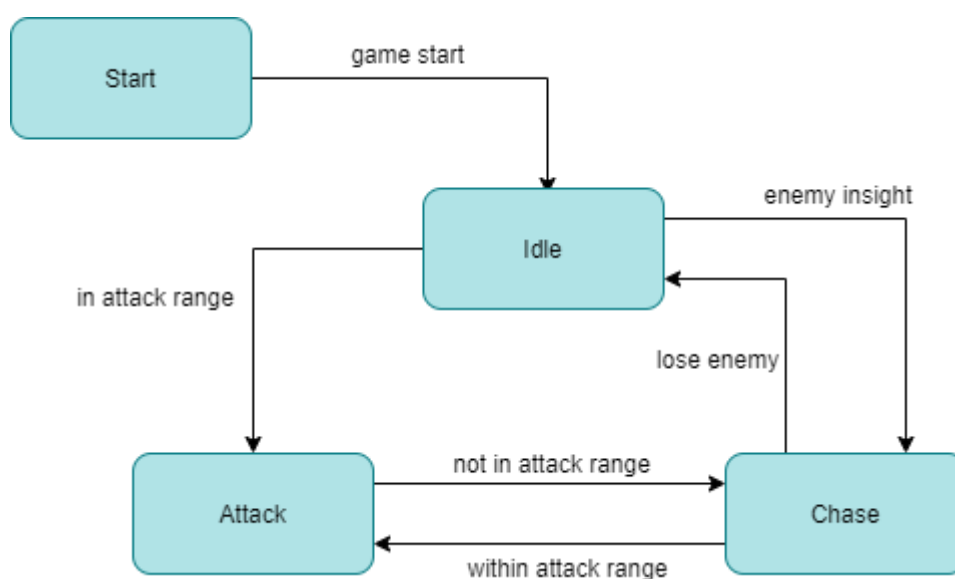
	- większa popularność, bardziej aktywne fora	
Unreal Engine	- najwyższy jakość graficzna - dobry do prototypowania - oprogramowanie typu open-source - wykorzystuje mniej zasobów i pamięci - dobry dostęp do wizualnego debuggowania, wydajniejsza technologia renderowania - oferuje dotacje, może być darmowy dla studentów	- wymagana dobra znajomość C++ - nie posiada bogatej dokumentacji - mniej materiałów do nauki w porównaniu do innych silników - mniej bogaty Asset Store - odpowiedni dla dużych projektów z dużym budżetem
GameMaker	- łatwy w użytkowaniu, bardzo elastyczny i wygodny w eksportowaniu gry na większość platform -nie wymaga wyższych umiejętności programistycznych	- jedynie dla programowania gier w 2D - brak wersji darmowej - wykorzystuje swój własny język GML - nieodpowiedni dla rozbudowanych projektów
Godot	- darmowe użycie i wykorzystanie, licencja MIT - wygodny i łatwy w użyciu interfejs użytkownika przez co jest doskonały dla osób początkujących	- nie wspiera konsol - brak bezpośrednich narzędzi graficznych - wspiera jedynie gry 2D - mniejsza społeczność

2.4 Zastosowanie AI w grach

Sztuczna inteligencja nawiązuje do algorytmów symulujących odpowiedniego rodzaju zachowania, będących elastycznymi i jednocześnie zdolnymi do adaptacji w zależności od sytuacji. Kojarzona jest nie tylko z informatyką, ale także z dziedzinami takimi jak fizyka czy matematyka ze względu na jej szerokie zastosowanie. Na samym początku rozwoju całego przemysłu gier sztuczna inteligencja nie stanowiła ich podstawowej części rozwoju, a opierała się jedynie na kilku prostych zasadach i regułach. Jednym z najczęściej wykorzystywanych narzędzi stał się automat maszynowy, który polegał na angażowanie przeciwników lub gracza w określony rodzaj

działania. W zależności od rodzaju rozgrywki, w danym momencie dla postaci będą dostępne działania tj. chodzenie, atakowanie, ściganie. Pierwsze gry posiadały liczne problemy związane z wykorzystaniem AI np. problemy z odnajdywaniem najkrótszej ścieżki i podejmowaniem odpowiednich decyzji. Jednymi z przyczynami takich problemów są trudności z rozszerzaniem automatu maszynowego, mimo tego, że jest on wyjątkowo prosty do zaprojektowania. Trudność polega na rozbudowaniu automatu oraz w związku z dodaniem nowych stanów koniecznością rozważenia i wprowadzenia zmian do przejść i stanów już istniejących. Aby poradzić sobie z tym problem, do narzędzi wykorzystujących AI wprowadzono drzewo zachowań posiadające strukturę hierarchiczną oraz posługujące się zahermetyzowaną logiką. Cechowało się także wykorzystaniem skalowalnych rozwiązań do dodawania nowych funkcjonalności oraz stanów w logice gry.

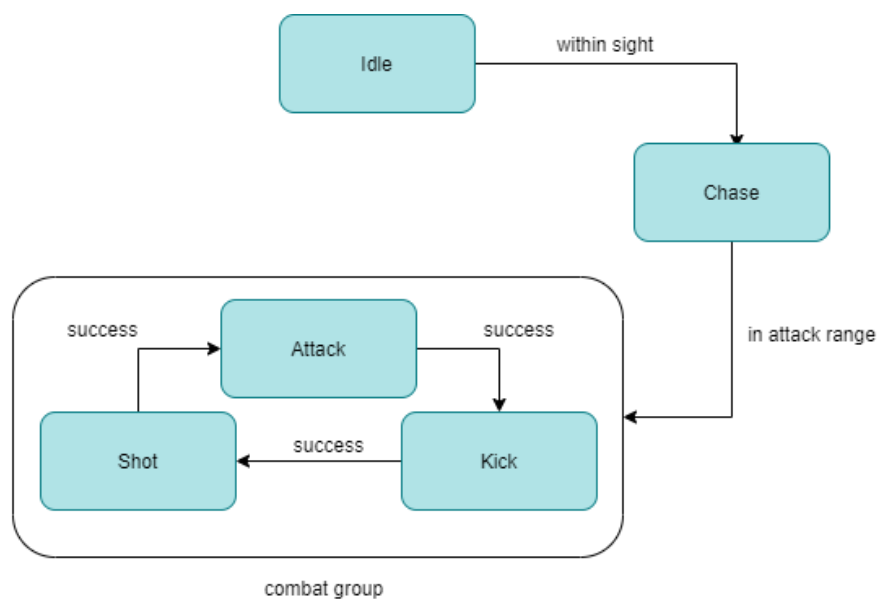
2.4.1 Automat skończony



Rys 2.2: Przykładowy diagram automatu skończonego (źródło własne)

Jedną z najpopularniejszych metod sztucznej inteligencji jest realizacja za pomocą Maszyny Stanów Skończonych (ang. Finite-State Machine – FSM). Pozostałe

określenia to: automat skończony lub automat o skończonej liczbie stanów. Działanie polega na wykorzystaniu dwóch komponentów – stanu i przejścia. Automat musi posiadać określoną, skończoną liczbę stanów, w których może się znajdować, a pomiędzy nimi znajdują się przejścia, które do aktywacji wymagają spełnienia określonych warunków. To określone narzędzie można mniej formalnie rozumieć jako mózg wroga, a każdy stan jako zadanie, które chce skończyć. Rysunek 2.1 pokazuje przykładowe cztery stany: start, bezczynność, ściganie i atak.



Rys 2.3: Przykładowy diagram automatu skończonego hierarchicznego (źródło własne)

Automat skończony posiada swoją ulepszoną wersję zwaną hierarchicznym automatem skończonym. Różnicą w stosunku do klasycznej wersji jest możliwość grupowania określonych stanów. Cała taka grupa stanów może dzięki temu dzielić między sobą to samo przejście do innych grup, zapewniając tym samym ponowne wykorzystanie określonych stanów.

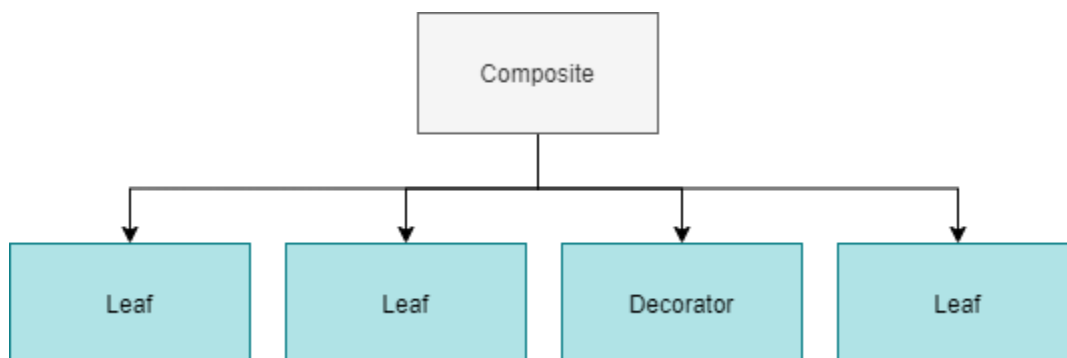
2.4.2 Drzewo behawioralne

Wraz z procesem rozwoju gier dla lepszych efektów i bardziej rozbudowanych zachowań powstało drzewo behawioralne (drzewo zachowań). Podobne jest ono do hierarchicznego automatu maszynowego, jednak zasadniczą różnicą jest powstanie drzewa z pojedynczym blokiem zadań, a nie grupą stanów.

Do stworzenia drzewa behawioralnego niezbędne jest wykorzystanie określonych węzłów, różniących się funkcjonalnością. Hierarchicznie ułożone węzły kontrolują przepływ podejmowania decyzji. Uruchamiane są one od góry do dołu. Schodząc coraz niżej możemy podzielić drzewo na trzy kategorie tj.: kompozyt, dekorator i liść. Drzewa mogą być niezwykle rozbudowane, istnieje możliwość rozbicia drzewa na inne poddrzewa oraz łączenia drzew ze sobą za pomocą węzłów. Pozwala to na programistyczne zaprojektowanie bibliotek zachowań ułatwiających pracę przy sztucznej inteligencji oraz stworzenia jak najbardziej przekonujących zachowań. W tym przypadku projektowanie staje się elastyczne – przykładowo możemy zacząć od podstawowego drzewa zachowań, a następnie rozbudowywać je za pomocą tworzenia gałęzi według ich priorytetowości, a także mamy możliwość zastosowania dodatkowych gałęzi jako alternatywnego rozwiązania w przypadku niepowodzenia określonego zachowania. Pomimo, że drzewa składają się z kilku różnych rodzajów węzłów, istnieje kilka podstawowych funkcji wspólnych dla każdego typu węzła w drzewie. Każdy węzeł może zwrócić określony status swojego działania. W zależności od implementacji, złożoności drzewa oraz potrzeb istnieje możliwość określenia ilości statusów, jednak podstawowe trzy zawarte w każdym węźle to:

- Powodzenie (ang. Success): informuje węzeł nadrzędny o powodzeniu.
- Niepowodzenie (ang. Failure): informuje węzeł nadrzędny o porażce
- Działanie (ang. Running): żaden z powyższych stanów nie został określony, a węzeł nadal działa. Węzeł będzie uruchomiony ponownie przy następnym przejściu przez drzewo i ponownie będzie miał możliwość odniesienia powodzenia, niepowodzenia lub kontynuowania działania.

Występowanie fazy działania pozwala na utrzymanie określonego stanu przez dłuższy czas w grze.



Rys 2.4: Przykładowy podział drzewa zachowań (źródło własne)

Kategorie drzewa behawioralnego:

- Kompozyt lub inaczej węzeł złożony lub węzeł sekwencji: leży u podstawy całego drzewa. Składa się zazwyczaj z kilku węzłów podrzędnych tj.: węzeł równoległy, sekwencji, selektora. Opowiada on za uruchamianie zadań podrzędnych w odpowiedniej kolejności (możliwość przetwarzania węzłów podrzędnych w kolejności od pierwszego do ostatniego lub losowej w zależności od węzła). Przekazuje on jeden z dostępnych statusów swojemu nadrzędnemu węzłowi, zdeterminowany na podstawie powodzeń lub niepowodzeń węzłów podrzędnych. Węzły podrzędne w czasie przetwarzania będą stale zwracać status działania do węzła.

W przypadku węzła złożonego najczęściej wykorzystywana jest technika uruchamiania każdego węzła podrzędnego po kolei. Węzeł ten nazywany jest Sekwencją.

- Liść: jest elementem leżącym najniżej. Pomimo tego, że jest ostatnim węzłem jego funkcje są zazwyczaj najbardziej rozbudowane i mają największy wpływ na rozgrywkę oraz mechanikę całej gry. Jest on końcowym poleceniem. Nie posiada węzłów podrzędnych, a jedynie nadrzędne.
- Dekorator lub węzeł dekoratora: pełni funkcję pośredniczącą. Często posiada węzeł podrzędny, który może jednak wystąpić tylko jeden raz. W zależności od

swojego typu może odpowiadać za funkcje tj.: odwracanie – w tym przypadku wynik otrzymany z węzła podrzędnego zostanie przekształcony, wyłączanie z obiegu węzła podrzędnego, powtórzenie przetwarzania węzła podrzędnego.

W przypadku węzła dekoratora najczęściej wykorzystywana jest technika odwracania. Węzeł ten nazywany jest Inwerterem.

Powyższe węzły wykorzystywane są dla bardziej złożonych zachowań sztucznej inteligencji. W kontekście programowania oraz tworzenia kodu gry liście można przyrównać do ostatecznych funkcji oraz działania programu, natomiast węzły dekoratora lub węzły złożone można analogicznie porównać do funkcji warunkowych (np. if oraz while). Wraz z rozbudową programu liście mogą posiadać wyjątkowo minimalną ilość kodu, co pozwoli na przejrzyste budowanie drzewa oraz utworzenie skomplikowanych zachowań sztucznej inteligencji. Liście dodatkowo mogą zostać sparametryzowane co pozwoli na jeszcze szersze ich zastosowanie, a ponadto mogą wywołać inne drzewo zachowań, przekazując już istniejący kontekst danych do kolejnego drzewa. Więcej na temat drzew w rozdziale 5.4. PandaBT.

3. Analiza systemu

Ten rozdział ukazuje dokładny opis gry, założenia ogólne projektu, model zachowań systemu wraz z modelowaniem przypadków użycia i modelowaniem danych.

Jak wynika z rys.1.1 jednym z gatunków gier cieszącym się największym zainteresowaniem jest FPS. Projekt został zakwalifikowany jako podgatunek strzelanki 3D – strzelanka pierwszoosobowa (FPS). Do zrealizowania go został wybrany ten rodzaj gry ze względu na jego popularność oraz możliwość wpasowania się w zapotrzebowanie na rynku. Istotne znaczenie ma także możliwość zrealizowania, zaprojektowania i implementacji projektu przez jedną osobę mając na uwadze pewne ograniczenia czasowe.

Obecne rozwiązania dostępne na rynku nie wykorzystują w pełni potencjału możliwości sztucznej inteligencji, przez co możliwości ich rozwoju stają się ograniczone, a sama fabuła gier i ich rozgrywka przewidywalna. Wśród gier czasu

rzeczywistego pojawiały się także liczne problemy jak np.: wykorzystanie zbyt wielu obiektów w grze, problemy ze znalezieniem najkrótszej ścieżki, brak wszystkich informacji czy też źle zaprojektowane drzewa zachowań. Przykładowo gra strategiczna Herzog Zwei miała problemy z odnalezieniem odpowiednich ścieżek i naruszyła potrójne stany maszynowe odpowiadające za kontrolę jednostek, natomiast w grze Dune II przeciwnik atakował jedynie bazę gracza, jeśli była ona zwrócona w jego stronę. Późniejsze gry z gatunku wykazywały bardziej wyrafinowaną sztuczną inteligencję, która ciągle jest na etapie rozwoju.

Głównym celem oprócz opracowania gry była nauka i skupienie się na obszarze sztucznej inteligencji. Projekt został w całości opracowany w środowisku Unity3D, którego najważniejszymi zaletami są cechy tj.: posiadanie rozbudowanej dokumentacji technicznej oraz forum społeczności - co zapewnia duży zakres i różnorodność materiałów naukowych. Dostępne są także darmowe wtyczki i zestawy assetów oraz wsparcie rozwoju środowiska 3D mającego zapewniać lepszy odbiór gry od strony użytkownika, a także platforma Unity3D umożliwia nam zbudowanie gry na najpopularniejsze platformy dostępne na rynku.

3.1 Ogólne założenia projektu

Podstawowym założeniem projektu jest stworzenie strzelanki 3D, w której gracz wciela się w rolę głównego bohatera i stara się walczyć o przetrwanie. Gra została napisana w języku C# i opracowana w Unity. Dodatkowo użyta została wtyczka PandaBT, odpowiadająca za wykorzystanie drzewa zachowań, które stanie się podstawą wykorzystania sztucznej inteligencji.

Rozgrywka oparta jest całkowicie na akcji, bez wykorzystania strategii lub elementów fabularnych. Celem jest przejście przez mapę z jednego punktu do drugiego w jak najkrótszym czasie oraz nie zostać przy tym zabitym przez wroga stworzenia. Gra ze względu na swoją prostotę umożliwia łatwą rozbudowę oraz wprowadzenie dalszych modyfikacji.

Wersja w tym projekcie zapewnia jeden poziom, dalsze są planowane wraz z rozbudową gry w przyszłości. Ponadto, gra nie oferuje graczowi punktów kontrolnych, co zmusza go do przejścia przez świat bez całkowitej utraty życia.

Zapewnienie środowiska 3D wraz z połączeniem doświadczania przez gracza świata oczami głównego bohatera ma na celu dostarczenia odpowiedniego poziomu adrenaliny.

Umiejętności i zadania postaci polegają na poruszaniu się we wszystkich kierunkach, skakaniu, kucaniu, używaniu broni wraz z możliwością wymiany, zbieraniu amunicji oraz baterii do latarki.

Domyślnie gracz na początku posiada 100 jednostek życia, które zmniejszają się wraz z atakiem wrogów, którzy mogą gryźć, lecz nie mają możliwości użycia broni lub innych narzędzi. Gdy zdrowie spadnie do zera lub poniżej, gracz umiera.

3.2 Specyfikacja wymagań

Poniższe wymagania przedstawia przewidywania dotyczące niezbędnych funkcjonalności systemu. Wymagania można podzielić na funkcjonalne i pozafunkcjonalne. Funkcjonalne dotyczą działania i zachowań programu, a pozafunkcjonalne określają jego cechy jak np. wydajność lub kompatybilność, mogą mieć jednocześnie większe znaczenie przy wyborze architektury systemu. Wymagania funkcjonalne dodatkowo zostały podzielone na kategorie takie jak m.in.: gracz, interfejs i przeciwnicy.

3.2.1 Wymagania funkcjonalne

Tabela 3.1

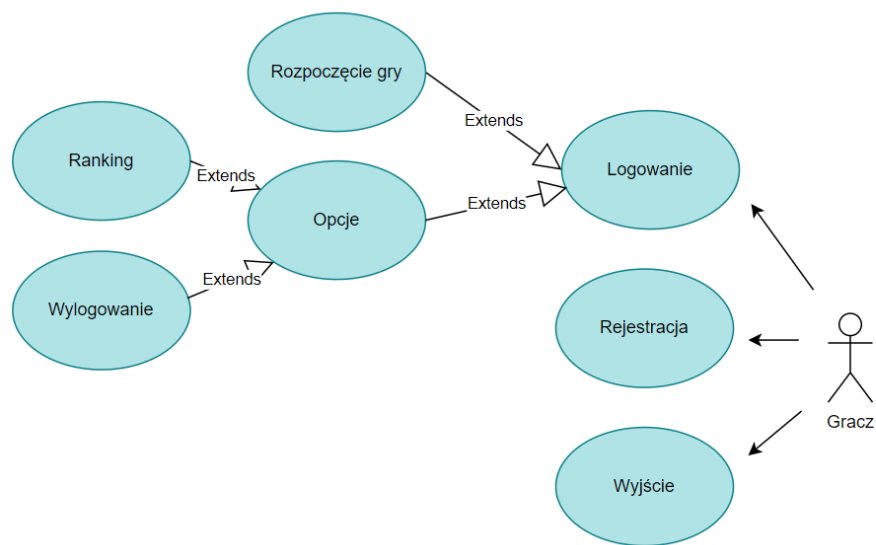
Wymagania funkcjonalne systemu (źródło własne).

Kategoria	Wymagania systemu
-----------	-------------------

Interfejs	<ul style="list-style-type: none"> • Wyświetlanie interfejsu użytkownika w odpowiednim momencie • Zapewnienie okna logowania do systemu • Zapewnienie możliwości rejestracji nowego użytkownika • Zapewnienie możliwości rozpoczynania rozgrywki • Zapewnienie możliwości opuszczenia systemu • Zapewnienie możliwości zatrzymania rozgrywki • Informowanie użytkownika o ilości punktów zdrowia, amunicji oraz czasu rozgrywki w jej trakcie
Gracz	<ul style="list-style-type: none"> • Zapewnienie możliwości poruszania się w trójwymiarowym układzie współrzędnych. • Zapewnienie możliwości strzelania w kierunku obrotu przy użyciu broni. • Zapewnienie interakcji z otoczeniem (kolizji). • Zapewnienie punktów zdrowia. • Umożliwienie graczowi wymiany broni. • W momencie, gdy zdrowie gracza jest równe zero lub poniżej zera system powinien zatrzymać rozgrywkę i przywrócić okno dialogowe.
Amunicja	<ul style="list-style-type: none"> • Zapewnienie interakcji zarówno z graczem, przeciwnikiem jak i z otoczeniem.
Przeciwnik	<ul style="list-style-type: none"> • Zapewnienie możliwości poruszania się w trójwymiarowym układzie współrzędnych oraz pojawiania się w dostępnych polach w świecie. • Zapewnienie punktów zdrowia. • Zapewnienie przeciwnikowi zakresu pościgu gracza i odpowiedniej reakcji w zależności od jego położenia. • Umożliwienie przeciwnikowi atakowania gracza, gdy ten znajduje się obok niego.

3.2.1.1 Obsługa UI

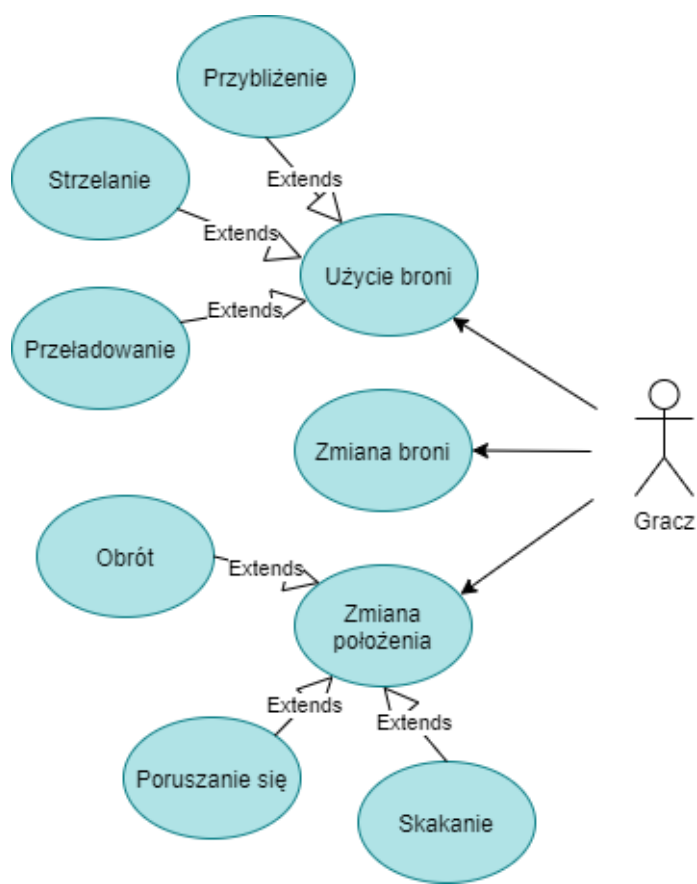
W momencie, gdy gra zostanie włączona przez użytkownika, dostępne dla niego będą opcje z włączonego interfejsu użytkownika tj.: rejestracja – w przypadku, kiedy nie posiada on jeszcze konta lub logowanie. Po zalogowaniu się, będzie istniała możliwość rozpoczęcia rozgrywki.



Rys.3.1 Diagram obsługi UI

3.2.1.2 Przeprowadzenie rozgrywki

Kiedy użytkownik zdecyduje się na rozpoczęcie gry, będzie mógł korzystać z funkcjonalności tj. użycie lub zmiana broni oraz będzie miał możliwość zmiany swojego położenia. W momencie, gdy gracz przejdzie przez całą mapę lub gdy liczba punktów gracza osiągnie zero lub mniej, rozgrywka zakończy się.



Rys.3.2 Diagram obsługi rozgrywki przez użytkownika

3.2.2 Wymagania pozafunkcjonalne

Tabela 3.2

Wymagania funkcjonalne systemu (źródło własne).

Kategoria	Wymagania systemu
Projekt	<ul style="list-style-type: none"> • Gra musi mieć możliwość uruchomienia na systemie Windows 7 lub nowszym. • Gra musi spełniać kryteria wiekowe PEGI 16. • Gra musi zostać utworzona z wykorzystaniem technologii 3D.

	<ul style="list-style-type: none"> • Gra musi zostać utworzona przy użyciu silnika Unity. • W grze musi zostać zastosowany widok z pierwszej osoby. • Rozgrywka będzie przeprowadzana tylko w trybie online. • Gra powinna być obsługiwana za pomocą klawiatury lub myszy. • Zapewnienie płynności rozgrywki na możliwie wielu urządzeniach.
Bezpieczeństwo	<ul style="list-style-type: none"> • Szyfrowanie prywatnych danych użytkowników.
Dostępność	<ul style="list-style-type: none"> • Możliwość zalogowania powinna być dostępna przez cały czas, siedem dni w tygodniu (wyjątkiem mogą być prace konserwacyjne).

3.3 Modelowanie danych

Projekt będzie posiadał bazę danych opartą na przechowywaniu danych użytkowników oraz danych dotyczących rozegranych gier.

Każdy użytkownik powinien posiadać nazwę oraz hasło niezbędne do logowania, a także adres mailowy, położenie użytkownika na trójwymiarowej osi współrzędnych

Wykorzystane w systemie elementy powinny zawierać następujące dane:

- Użytkownicy
 - Nazwa użytkownika
 - Hasło
 - Adres email

3.4 Modelowanie przypadków użycia

W tym podrozdziale zostaną pokazane i opisane scenariusze użycia systemu.

Tabela 3.3

Rejestracja użytkownika w systemie (źródło własne).

Sekcja	Treść
Aktorzy	Użytkownik
Warunki wstępne	Uruchomiony ekran rejestracji
Warunki końcowe	Stworzenie konta użytkownika
Rezultat	Dane użytkownika zostają zarejestrowane do bazy
Scenariusz główny	<ol style="list-style-type: none"> 1. Wyświetlenie formularza logowania. 2. Użytkownik przechodząc z okna logowania do okna rejestracji informuje system o chęci zarejestrowania konta. 3. Użytkownik podaje swoje dane tj.: login, hasło, potwierdzenie hasła, adres email. 4. Użytkownik potwierdza chęć założenia konta klikając w przycisk. 5. System informuje o pomyślnym zarejestrowaniu konta.
Scenariusz wyjątku	<p>Zdarzenie: Błąd lub brak połączenia z siecią lub bazą</p> <ol style="list-style-type: none"> 5.1. System informuje o błędzie połączenia. 6. Następuje przejście do 2 punktu scenariusza głównego.
Scenariusz wyjątku	<p>Zdarzenie: Powtórzenie nazwy użytkownika w bazie</p> <ol style="list-style-type: none"> 5.1. System informuje o powtórzeniu nazwy. 6. Następuje przejście do 2 punktu scenariusza głównego.
Scenariusz wyjątku	<p>Zdarzenie: Wprowadzenie niepoprawnych bądź niepełnych danych</p> <ol style="list-style-type: none"> 5.1. System informuje o błędnie wprowadzonych danych. 6. Następuje przejście do 2 punktu scenariusza głównego.

3.4.2 Logowanie użytkownika

Tabela 3.4

Logowanie użytkownika do systemu (źródło własne).

Sekcja	Treść
Aktorzy	Użytkownik
Warunki wstępne	Uruchomiony ekran logowania
Warunki końcowe	Zalogowanie użytkownika
Rezultat	Dane użytkownika zostają uwierzytelnione i zostaje on zalogowany do gry.
Scenariusz główny	<ol style="list-style-type: none">1. Wyświetlenie formularza logowania.2. Wprowadzenie loginu oraz hasła przez użytkownika.3. Użytkownik potwierdza chęć zalogowania klikając w przycisk.4. Następuje zalogowanie użytkownika i może on rozpocząć rozgrywkę.
Scenariusz wyjątku	Zdarzenie: Błąd lub brak połączenia z siecią lub bazą <ol style="list-style-type: none">4.1. System informuje o błędzie połączenia.5. Następuje przejście do 2 punktu scenariusza głównego.
Scenariusz wyjątku	Zdarzenie: Niepoprawny login bądź hasło <ol style="list-style-type: none">4.1 System informuje o Błędnie wprowadzonych danych.5. Następuje przejście do 2 punktu scenariusza.

3.4.3 Walka gracza z przeciwnikami typu Arachnid i Mutant Podstawowy

Tabela 3.5

Walka gracza z przeciwnikami typu Arachnid i Mutant Podstawowy (źródło własne).

Sekcja	Treść
Aktorzy	Przeciwnik, Gracz
Warunki wstępne	Rozpoczęta rozgrywka, spotkanie przeciwnika i gracza.
Warunki końcowe	Ginie gracz lub przeciwnik.
Rezultat	Przeciwnik i gracz zaprzestają walki.
Scenariusz główny	<ol style="list-style-type: none"> 1. Gracz zostaje dostrzeżony przez przeciwnika. 2. Przeciwnik rozpoczyna pościg za graczem. 3. Przeciwnik dogania gracza. 4. Przeciwnik atakuje gracza atakiem podstawowym. 5. Gracz traci część punktów życia. 6. Gracz równocześnie atakuje przeciwnika. 7. Przeciwnik traci część punktów życia. <p>Powtarzaj punkty 4-7.</p> <ol style="list-style-type: none"> 8. Gracz ginie.
Czynności alternatywne	<p>*a w każdym momencie walki, jeżeli gracz posiada krytyczną ilość punktów życia.</p> <p>I. Przejście do punktu 8 głównego scenariusza.</p> <p>*b w każdym momencie walki, jeżeli przeciwnik posiada krytyczną ilość punktów życia.</p> <p>I. Przeciwnik ginie.</p> <p>*c w każdym momencie trwania scenariusza walki, gracz może rozpocząć ucieczkę.</p> <p>I. Przeciwnik rozpoczyna pościg za graczem.</p> <p>II.1 Przeciwnik podąża za graczem, dopóki nie będzie on w zasięgu jego ataku.</p> <p>II.2 Następuje przejście do punktu 4 głównego scenariusza.</p> <p>1.1 Przeciwnik zostaje dostrzeżony przez gracza.</p> <p>1a.1 Gracz atakuje przeciwnika.</p> <p>1a.2 Następuje przejście do punktu 2 głównego scenariusza.</p> <p>1b.1 Użytkownik ucieka z miejsca, w którym znajduje się przeciwnik.</p>

3.4.4 Walka gracza z przeciwnikiem typu Mutant Kryształowy

Tabela 3.6

Walka gracza z przeciwnikiem typu Mutant Kryształowy (źródło własne).

Sekcja	Treść
Aktorzy	Gracz, Przeciwnik
Warunki wstępne	Rozpoczęta rozgrywka, spotkanie przeciwnika i gracza.
Warunki końcowe	Gracz/Przeciwnik zostaje pokonany lub jedna ze stron ucieka.
Rezultat	Gracz i przeciwnik zaprzestają walk.
Scenariusz główny	<ol style="list-style-type: none"> 1. Gracz zostaje dostrzeżony przez przeciwnika. 2. Przeciwnik woła pobliskie sojusznicze jednostki. 3. Przeciwnik wraz z jednostkami rozpoczyna pościg za graczem. 4. Przeciwnik dogania gracza i atakuje go umiejętnością skoku. 5. Gracz traci część punktów życia. 6. Gracz równocześnie atakuje przeciwnika. 7. Przeciwnik traci część punktów życia. 8. Następnie przeciwnik zadaje graczowi obrażenia atakiem podstawowym na zmianę z atakami specjalnymi. 9. Gracz traci część punktów życia. Powtarzaj punkty 6-9. 10. Gracz ginie.
Czynności alternatywne	<p>*a w każdym momencie walki, jeżeli gracz posiada krytyczną ilość punktów życia. I. Przejście do punktu 10 głównego scenariusza.</p> <p>*b w każdym momencie walki, jeżeli przeciwnik posiada krytyczną ilość punktów życia. I. Przeciwnik ginie.</p> <p>*c w każdym momencie trwania scenariusza walki, gracz może rozpocząć ucieczkę.</p>

	<p>I. Przeciwnik rozpoczyna pościg za graczem.</p> <p>IIa.1 Przeciwnik dogania gracza.</p> <p>IIa.2 Następuje przejście do punktu 4 głównego scenariusza.</p> <p>IIb.1 Gracz ucieka.</p> <p>*d w każdym trwania scenariusza walki, jeśli przeciwnik posiada mniej niż 30 punktów oraz nie posiada sojuszników rozpoczyna ucieczkę.</p> <p>I. Przeciwnik ucieka.</p> <p>II. Gracz dobija przeciwnika.</p> <p>1.1 Przeciwnik zostaje dostrzeżony przez gracza.</p> <p>1a.1 Gracz atakuje przeciwnika.</p> <p>1a.2 Przeciwnik bez jednostek rozpoczyna pościg za użytkownikiem.</p> <p>1a.2 Następuje przejście do punktu 8 głównego scenariusza.</p> <p>1b.1 Użytkownik ucieka z miejsca, w którym znajduje się przeciwnik.</p> <p>2.1 W pobliżu nie ma wystarczającej ilości sojusznicznych jednostek.</p> <p>2.2 Następuje wygenerowanie na mapie nowych sojuszników w pobliżu przeciwnika.</p> <p>2.3 Następuje przejście do punktu 3 głównego scenariusza.</p> <p>3.1 Gracz znajdzie się poza zasięgiem widzenia przeciwnika – ucieczka gracza.</p>
Scenariusz wyjątku	<p>4.1 Przeciwnik lub gracz znajdują się na nierówno położonej siatce nawigacji lub pojawia się między nimi przeszkoda.</p> <p>4.2 Następuje pominięcie umiejętności ataku specjalnego skoku.</p> <p>4.3 Przejście do punktu 5 głównego scenariusza.</p>

3.4.5 Mutant Kryształowy – leczenie

Tabela 3.7

Przebieg leczenia Mutanta Kryształowego (źródło własne).

Sekcja	Treść
Aktorzy	Przeciwnik, Gracz
Warunki wstępne	Rozpoczęta rozgrywka, przeciwnik posiada mniej niż 30 punktów życia oraz nie posiada wystarczającej liczby zmobilizowanych sojusznicznych jednostek.
Warunki końcowe	Przeciwnik posiada maksymalną ilość punktów życia.
Rezultat	Przeciwnik zostaje uleczony.
Scenariusz główny	<ol style="list-style-type: none"> 1. Gracz zadaje obrażenia przeciwnikowi. 2. Liczba punktów życia przeciwnika spada poniżej 30 oraz nie posiada on wystarczająco sojuszników. 3. Przeciwnik sprawdza kąt padania kamery gracza i wybiera punkt w przeciwnym kierunku. 4. Przeciwnik zapamiętuje w którym miejscu widział gracza oraz ile posiadał sojuszników. 5. Przeciwnik biegnie w kierunku punktu. 6. Przeciwnik dociera do punktu leczenia. 7. Przeciwnik leczy się. 8. Uleczony przeciwnik powraca do punktu, w którym ostatni raz widział gracza.
Czynności alternatywne	<p>*a w każdym trwania scenariusza unikania gracza, jeśli przeciwnik posiada krytyczną ilość punktów życia ginie.</p> <p>I. Przeciwnik ginie.</p> <p>5.1 Jeśli przeciwnik posiada ilość punktów życia większą od 0.01, ale mniejszą od 10 rozpoczyna się czołganie przeciwnika.</p> <p>I. Przeciwnik czołga się w kierunku punktu.</p> <p>II. Następuje powrót do 6 punktu głównego scenariusza.</p>

3.4.6 Zakończenie rozgrywki

Tabela 3.8

Zakończenie gry przez użytkownika (źródło własne).

Sekcja	Treść
Aktorzy	Użytkownik
Warunki wstępne	Gracz ginie
Warunki końcowe	Zaktualizowanie bazy danych
Rezultat	Dane o rozgrywce zostają zapisane do bazy danych.
Scenariusz główny	<ol style="list-style-type: none"> 1. Użytkownik dociera do końca mapy lub wychodzi z gry. 2. Informacje zostają przekierowane do bazy danych. 3. Baza danych zaktualizowana o dodatkowy wpis.
Scenariusz wyjątku	<p>Zdarzenie: Błąd lub brak połączenia z siecią lub bazą</p> <ol style="list-style-type: none"> 4.1. System informuje o błędzie połączenia. 5. Następuje przejście do 2 punktu scenariusza głównego.

4. Wybór metodyki

Aby ustrukturyzować model pracy nad projektem stosowane są różnego rodzaju metodyki tj. model spiralny, model wodospadu, model programowania ekstremalnego itd. Zdecydowałam się na model kaskadowy, ponieważ w przypadku tworzenia gry bardzo pomocne było zidentyfikowanie podstawowych faz rozwoju oprogramowania już na początku i uporządkowanie procesu jego tworzenia. Dzięki temu już na samym wstępie możliwe było dokładniejsze zaplanowanie wykonania, a na późniejszym etapie prac ułatwiło zarządzaniem wykonania.

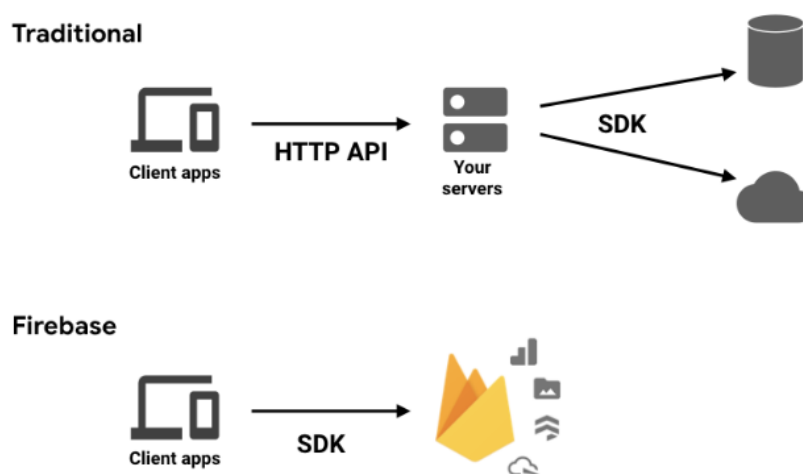
5. Projekt

Ten rozdział zawiera opis stworzonej mechaniki gry oraz architektury obiektowej wraz z użytymi narzędziami podczas procesu tworzenia gry.

5.1 Narzędzia

Nawiązując do wcześniejszych założeń projekt będzie tworzony w metodyce zwinnej Agile, natomiast kod programu zostanie napisany w języku C# z wykorzystaniem środowiska Unity3D oraz Microsoft Visual Studio Community. To zintegrowane środowisko programistyczne zostało użyte ze względu na fakt, że jest darmowe, łatwo dostępne, posiada łatwy w obsłudze edytor kodu w pełni kompatybilny i przystosowany do pracy wraz z Unity3D.

Do przechowywania danych wykorzystana zostanie serwis Firebase zawierający bazę danych typu NoSQL. Firebase jest narzędziem od firmy Google, pozwalającą na tworzenie aplikacji na platformy tj.: Andorid, IOS czy aplikacje sieciowe. Umożliwia analizę aplikacji, odnajdywanie błędów, obsługę chmury plików, monitorowanie aplikacji oraz jej stabilności i jakości, pomaga osiągnąć określone cele biznesowe. Mamy także możliwość wygenerowania określonych raportów i zapewnienie hostingu aplikacji. W projekcie zostanie wykorzystana również możliwość przeprowadzenia autentykacji użytkownika w postaci logowania za pomocą loginu i hasła. Firebase wykorzystuje pakiety SDK, które wchodzą w interakcję z tymi usługami bezpośrednio, w tym przypadku nie jest konieczne ustanawianie oprogramowania pośredniczącego między aplikacją, a usługą.



Rys.5.1: Działanie Firebase źródło: <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0>

5.2 Projekt bazy

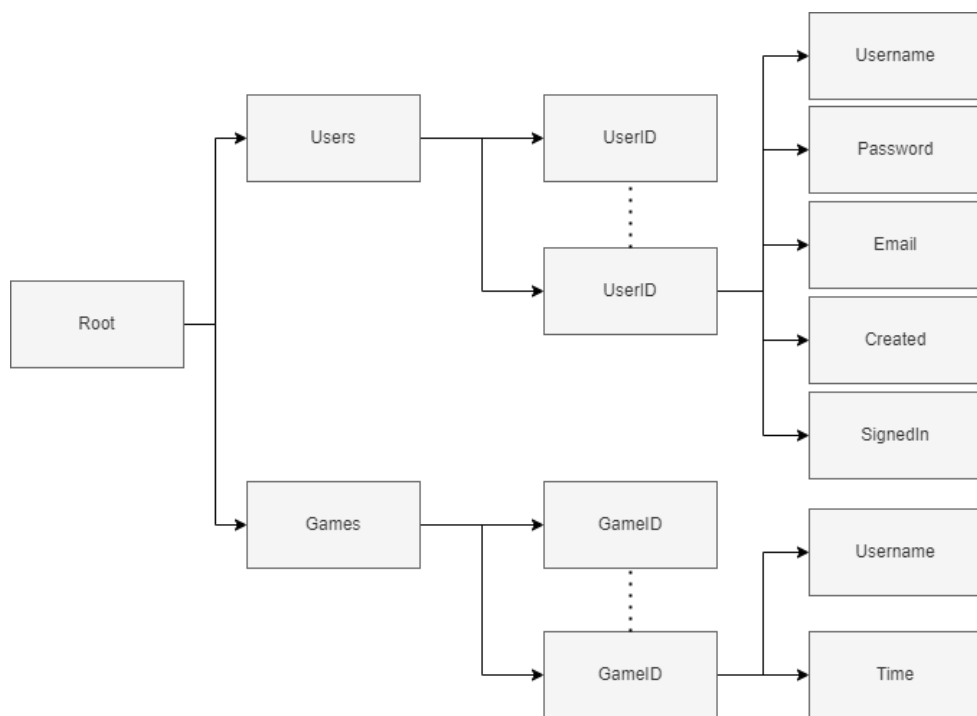
Projekt będzie posiadał bazę danych opartą na przechowywaniu danych użytkowników oraz danych dotyczących rozegranych gier.

Do utworzenia bazy danych niezbędne będą: tabela zawierająca wszystkich użytkowników wraz z ich nazwami, hasłem oraz emailem, a także tabela zawierająca dane rozgrywki w celu wyświetlenia trzech najkrótszych czasów tj.: id rozgrywki, całkowity czas rozgrywki, nazwa użytkownika.

users		games	
id	int	id	int
username	varchar	username	varchar
password	varchar	time	varchar
email	varchar		

Tabela 5.2.1 i 5.2.2: Wymagane tabele do utworzenia nierelacyjnej bazy danych (źródło własne).

W związku z zastosowaniem nierelacyjnej bazy danych konieczne jest zamodelowanie bazy na wzór struktury drzewiastej, wraz z wykorzystaniem danych połączonych na zasadzie klucz-wartość. Podstawą w takim drzewie stanowi korzeń, a kolejne dane są jego odgałęzieniami.



Rys.5.3 Model danych (źródło własne).

5.3 Wykorzystanie funkcjonalności silnika Unity

Wykorzystanie pełni możliwości Unity pozwala nam na projektowanie, stworzenie i dodanie wielu obiektów oraz funkcjonalności bez konieczności korzystania z zewnętrznych programów.

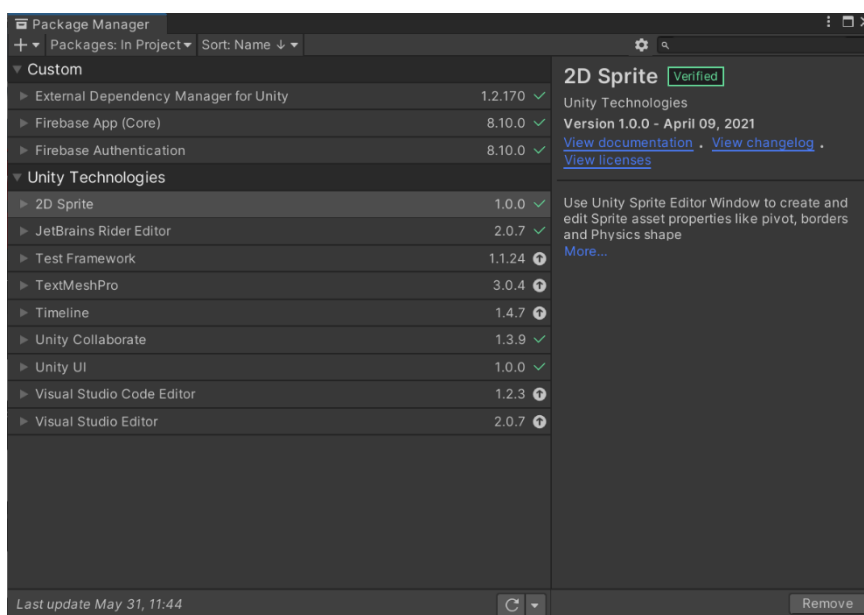
5.3.1 Asset Store oraz Package Manager

Unity Asset Store jest stroną umożliwiającą dodanie pakietów do wbudowanego w Unity Package Managera. Członkowie społeczności mają możliwość dodania stworzonych przez siebie zasobów, które mogą być w wersji komercyjnej lub darmowej.

W związku z szeroką gamą zasobów tj. tekstury, modele, animacje czy rozszerzenia, wiele zasobów wykorzystanych w projekcie pochodzić będzie z Asset

Store, udostępnionym użytkownikom na licencji wolnego, jak i własnościowego oprogramowania. Pozwala to na zaoszczędzenie czasu podczas projektowania. Część pobranych obiektów będzie podlegała obróbce na potrzeby gry w zewnętrznym oprogramowaniu Blender, natomiast pliki graficzne edytowane zostaną w edytorze 2D Sprite.

Dzięki menadżerowi pakietów możemy łatwo dokonywać operacji na całych dostępnych pakietach lub części ich plików – dodawać, instalować, odinstalowywać, wyłączać lub zmieniać dostępne wersje i aktualizować pliki.

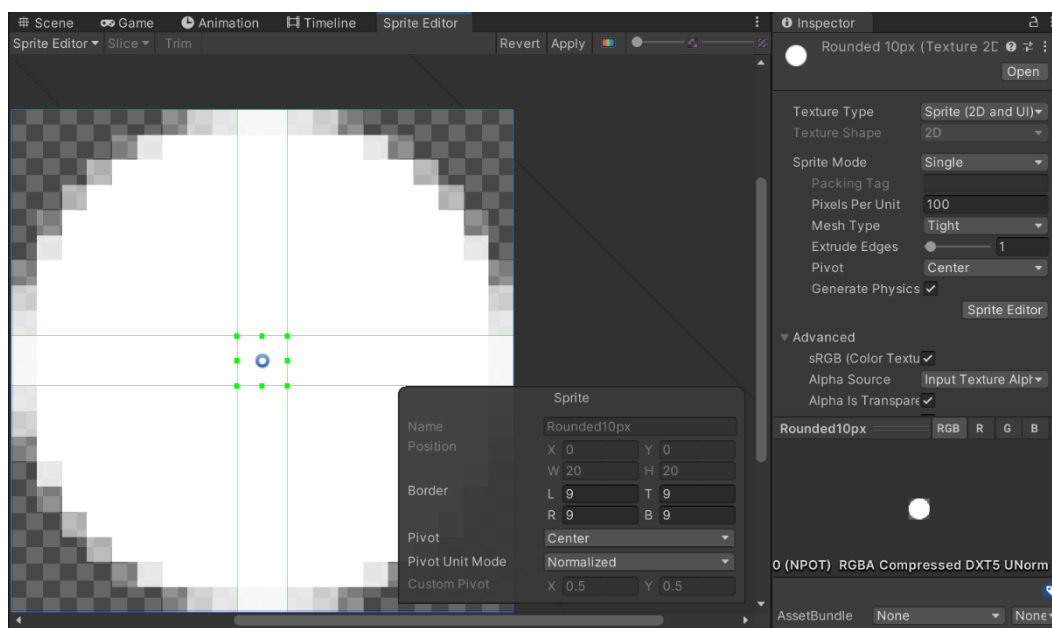


Rys.5.4 Okno Package Manager'a (źródło własne).

5.3.2 Sprite Editor

Package Manager umożliwia dodanie wtyczki takiej jak 2D Sprite. Generuje ona okno odpowiedzialne za edycję elementów graficznych. Pozwala na edytowanie osi obrazu, tekstury, kolorów, filtrów, formatów oraz na łączenie kilku elementów w jeden lub rozdzielenie ich. Narzędzie to jest niezwykle przydatne podczas projektowania gier 2D lub w przypadku obecnego projektu gry 3D do stworzenia interfejsu użytkownika.

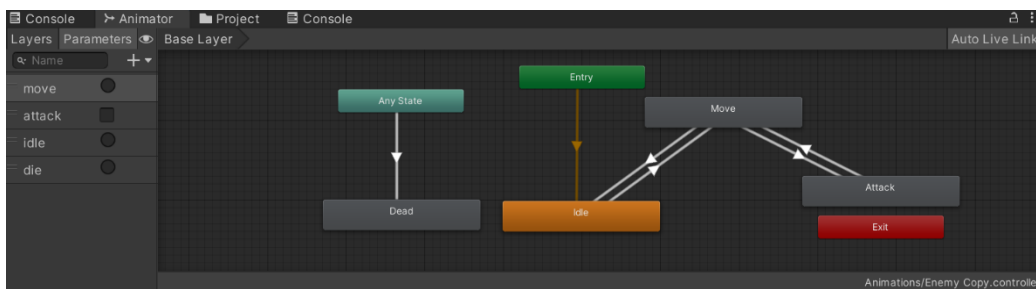
Rysunek 5.5 pokazuje przykładową edycję krawędzi pola guzika wykorzystanego w projekcie.



Rys.5.5 Okno Sprite Editor (źródło własne).

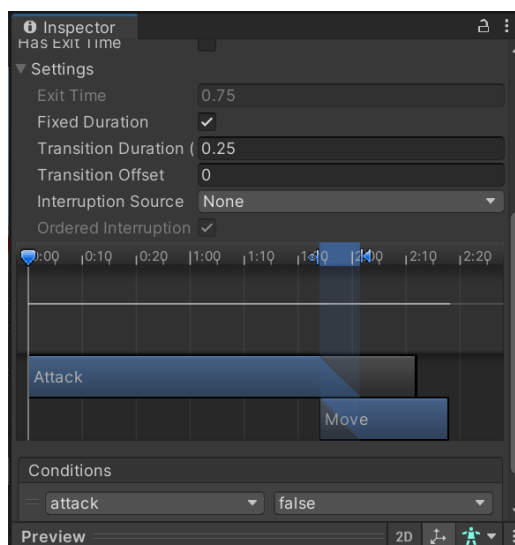
5.3.3 Animator Controller

Unity wykorzystuje swoje własne okno do tworzenia animacji. Działa ono na zasadzie automatu skończonego. Animator Controller jest zasobem Unity, który kontroluje przebieg animacji obiektu. Rysunek 5.6 przedstawia przykładowy najprostszy kontroler, w którym stany są ze sobą połączone za pomocą strzałek. Stany są reprezentacją animacji. Zielony stan Wejścia (ang. Entry) wskazują na pierwszą animację po załadowaniu sceny, następne przejścia są możliwe dzięki parametrom, a obsługa całego kontrolera odbywa się w kodzie. Istnieje możliwość stworzenia 4 rodzajów parametrów w zależności od potrzeb i pełnionych funkcji – float, int, bool lub trigger. Podłączenie do niebieskiego pola „Any State” – umożliwia przejście do wybranego stanu z każdego innego bez uwzględniania pozostałych przejść i parametrów oraz poprzez zignorowanie bieżącego stanu.



Rys.5.6 Okno Animatora – Enemy Controller (źródło własne).

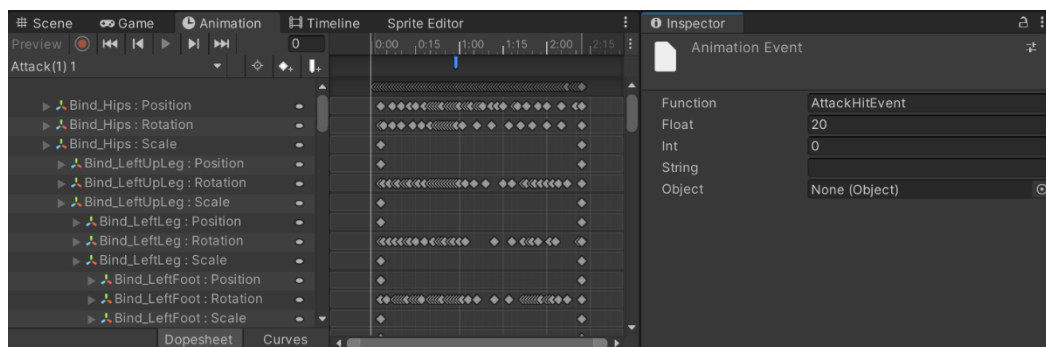
Istnieje możliwość dokładnego kontrolowania i zaprojektowania każdego z przejść z osobna. Czas wyjścia z animacji, jej zapętlenia oraz dodanie warunków przełączania możliwe jest dzięki oknu Inspektora – daje te możliwości stworzenia spersonalizowanych przejść oraz ich płynniejszego działania. Rysunek 5.7 przedstawia stan przejścia pomiędzy animacją zakończenia ataku a rozpoczęciem poruszania się.



Rys.5.7 Okno Inspektora – przejście Attack-Move (źródło własne).

Każda z animacji może zostać dokładnie zaprojektowana poprzez nagranie w oknie Animacji wybranych przejść lub dodanie ich mechanicznie. Mamy możliwości późniejszej edycji, a także dodanie tzw. zdarzeń (ang. events) w konkretnym momencie czasowym, które zostaną później wykorzystane w kodzie. Przykładowo chcemy, aby uderzenie przeciwnika odejmowało graczowi punkty życia jedynie w momencie zderzenia ręki wroga z ciałem gracza, a nie podczas trwania całej animacji uderzenia wraz z zamachem oraz obrotem jego ciała. Rysunek 5.8 przedstawia dokładnie rotację,

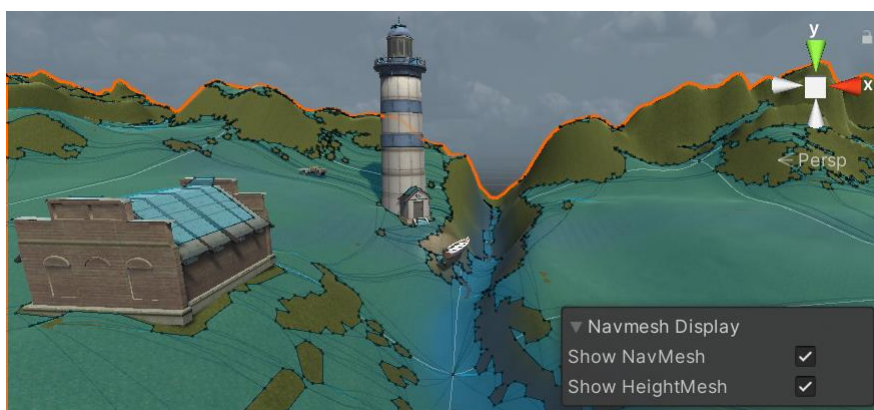
przejście, zmianę rozmiaru i położenia każdego z elementów przecwinika podczas animacji uderzenia. Dodana została także obsługa zdarzenia w konkretnym momencie uderzenia gracza dla bardziej realistycznego oddania ataku.



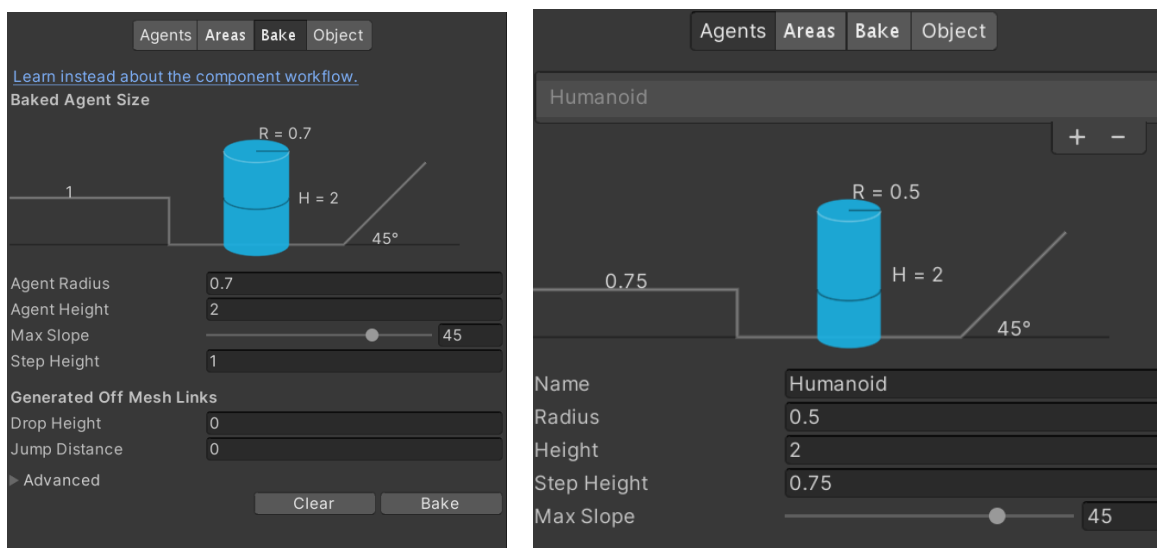
Rys.5.8 Okno Animacji – animacja ataku Mutanta (źródło własne).

5.3.4 System nawigacji oraz Pathfinding

Środowisko Unity umożliwia wyznaczenie i późniejsze wykorzystanie interaktywnej siatki, która służy określaniu obszaru do poruszania się i obszaru, w którym nie jest ono możliwe – jest to tzw. siatka Navmesh. Znajduje ona zastosowanie przy odnajdywaniu ścieżek i nawigacji dla obiektów posiadających sztuczną inteligencję. Drugim komponentem wykorzystywanym przy nawigacji jest Navmesh Agent, który podpięty jest do wybranego obiektu w grze. Zapewnia on możliwość poruszania się na siatce, odnajdywanie celu i unikaniu przeszkód, a także ma za zadanie wskazać obszarów niemożliwych do wyjścia np. obszar znajdujący się za wysoko. Rys.5.9 przedstawia teren, po którym może poruszać się obiekt, natomiast rys.5.10 i rys.5.11 ukazują możliwość personalizacji powierzchni poruszania się w zależności od rodzaju obiektu. W momencie, kiedy system odnajdzie obszar, który jest wystarczająco szeroki, płaski i na odpowiednim poziomie dla wybranego agenta – zostaje on oznaczony jako obszar możliwy do poruszania się.



Rys.5.9 Scena gry - obszar oznaczony jako możliwy do poruszania się (źródło własne).



Rys.5.10 i 5.11 Dostosowywanie powierzchni do agenta (źródło własne).

5.3.5 Skrypty

Do stworzenia gry lub innej aplikacji niezbędne jest wykorzystanie skryptów. Unity umożliwia wygodne podpięcie wybranych skryptów do określonych obiektów, odpowiadając za ich zachowanie w trakcie trwania rozgrywki lub tworzenia efektów graficznych czy audiowizualnych. Każdy nowostworzony skrypt Unity zawiera klasę bazową `MonoBehaviour`, a wraz z nią gotowe już klasy do implementacji tj. `Start()` –

klasa uruchamiająca się raz na samym początku gry, Update() – klasa uruchamiana z każdą klatką gry.

5.4 PandaBT

Jest to rozszerzenie Unity dostępne w Asset Store w wersji płatnej oraz bezpłatnej. Na potrzeby projektu do budowania drzewa zostanie wykorzystana wersja darmowa. Wtyczka ta umożliwia tworzenie drzew behawioralnych dla gry wraz z wykorzystaniem złożonych, skalowalnych i wielokrotnego użytku logik z użyciem funkcji. W robotyce oraz grach wideo często wykorzystywana jest technika sztucznej inteligencji jaką jest drzewo zachowań. Unity nie posiada jednak wbudowanej możliwości tworzenia drzew, dlatego aby projektować jednostki i ich zachowania przy użyciu hierarchicznego drzewa działań niezbędne jest doinstalowanie rozszerzenia. Po zainstalowaniu pakietu istnieje możliwość dodania skryptu Panda Behaviour, który wyświetla stan drzewa behawioralnego bezpośrednio w Inspektorze, dzięki czemu przepływ działania drzewa i przechodzenia przez poszczególne etapy jest widoczny w trakcie działania programu. Ponadto skrypt Panda Behaviour napisany jest w uproszczonym języku i korzysta z funkcji w języku C# zawartych w innych skryptach określonych jako zadania (ang. tasks). Każde zadanie może być wykorzystane więcej niż raz i może przynależeć nie tylko do jednego drzewa.

5.4.1 Wykorzystanie drzew behawioralnych

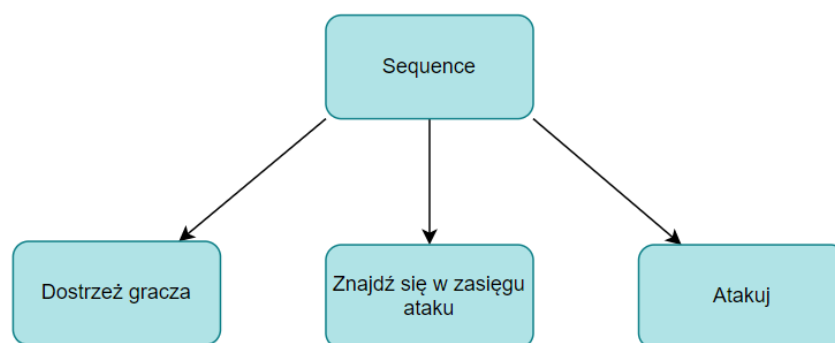
Całe drzewo można uznać za pojedynczą akcję złożoną z podakcji. PandaBT wykorzystuje standardowe trzy główne archetypy węzłów drzewa zachowań tj. Kompozyt, Dekorator i Liść. Zadaniem drzewa jest decydowanie z pomocą węzłów, które zadania należy uruchomić, biorąc pod uwagę obecny stan świata. Węzeł złożony (Kompozyt) jest jednym z najczęściej wykorzystywanych węzłów. Poniżej zostały opisane węzły złożone wraz z ich nazwą używaną w PandaBT:

- Sekwencja (ang. Sequence, PandaBT: sequence): jej zadaniem jest odwiedzenie każdego podrzędnego węzła, zaczynając od pierwszego, a kończąc na ostatnim. Odwiedzenie każdego kolejnego węzła wymaga jednak, aby poprzedni zakończył się powodzeniem. W momencie, w którym każdy z węzłów podrzędnych zwróci status powodzenia, sekwencja zwraca powodzenie węzłowi nadrzędnemu, natomiast jeśli jednak jeden z węzłów podrzędnych zwróci niepowodzenie, wtedy cały węzeł sekwencji zwraca niepowodzenie węzłowi nadrzędnemu.

Istotny jest fakt, że każdy rodzaj węzła posiada wiele zastosowań i nie można przypisać mu jednego. Najbardziej oczywistym zastosowaniem sekwencji jednak jest zdefiniowaniem zestawu lub sekwencji zadań, które muszą być wykonane w całości, aby zwróciły powodzenie.

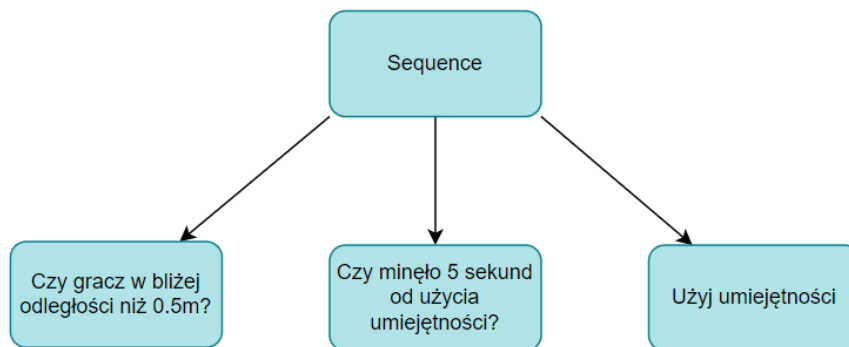
Przykładem może być sekwencja z rys. 5.12: dostrzeżenie gracza -> dotarcie do zasięgu ataku -> atak.

Nie możemy w tym przypadku znaleźć się w zasięgu gracza, jeśli wcześniej go nie zauważyliśmy, ponieważ nie wiemy nawet w jakim kierunku musimy się poruszyć lub rozpoczęcie ataku, jeśli gracz nie jest w jego zasięgu, ponieważ przykładowo napotkaliśmy przeszkody po drodze, droga była zablokowana lub gracz zdążył uciec.



Rys.5.12 Przykładowa sekwencja (źródło własne).

Żeby pokazać możliwość wielu zastosowań do sekwencji w innym kontekście niż tylko lista funkcji i zadań, które muszą się odbyć jedna po drugiej, można posłużyć się przykładem z rys. 5.13.



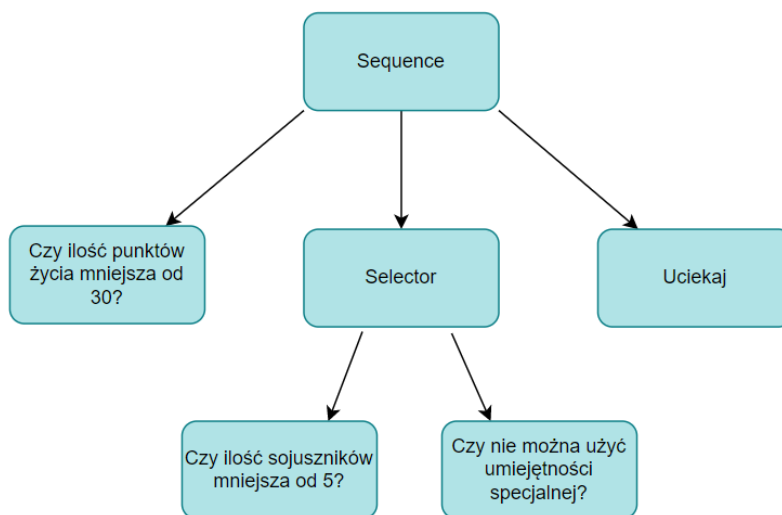
Rys.5.13 Przykładowa sekwencja (źródło własne).

W powyższym przykładzie lista zadań zmienia się na listę testów/warunków koniecznych do spełnienia, aby przejść do następnego węzła. Używanie takich sekwencji pozwala przetestować jeden lub więcej warunków przed wykonaniem akcji. Sprawdzany jest warunek czy gracz nie odszedł w międzyczasie zbyt daleko oraz czy specjalna umiejętność zdążyła zostać naładowana, czyli czy minęło od jej użycia 5 sekund. Dopiero, gdy oba te węzły zwrócą powodzenie, będzie możliwość użycia umiejętności, jednak nawet przy wykonywaniu ostatniego węzła sekwencja będzie mogła zwrócić niepowodzenie, gdy np. przeciwnik nie zdąży jej użyć i w międzyczasie umrze.

Użycie sekwencji można przedstawić jako analogiczne do funkcji bramki AND w obwodach.

- Selektor (ang. Selector, PandaBT: fallback) – jego zadaniem jest odwiedzenie każdego węzła podrzędnego jeden po drugim. W przeciwieństwie do sekwencji, selektor zwróci powodzenie, jeśli choć jeden z jego węzłów podrzędnych zwróci powodzenie i nie będzie wtedy odwiedzał kolejnych węzłów. Selektor zatem przechodzi przez węzły tak długo jak zwracają one niepowodzenie, jeśli natomiast wszystkie jego węzły zwrócą niepowodzenie to i on zwraca niepowodzenie do węzła nadrzędnego. Może służyć także jako instrukcja warunkowa w której sprawdzane jest kilkanaście warunków w poszukiwaniu choć jednego prawdziwego.

Przykładem może być rys.5.14: sprawdzamy czy ilość punktów życia spadła poniżej 30 ->jeśli tak to sprawdzamy czy jedno ze stwierdzeń jest prawdziwe: czy wokół jest mniej niż 5 sojuszników lub czy nie możemy użyć umiejętności specjalnej -> jeśli jedno z nich jest prawdziwe to rozpoczynamy ucieczkę.



Rys.5.14 Przykładowa zastosowanie selektora (źródło własne).

Selektor znajduje różne sposoby działania, ponieważ węzły można ułożyć także w kolejności od najbardziej korzystnych do najmniej korzystnych, co ma wpływ na jakość i rozbudowanie projektowanej przez nas sztucznej inteligencji.

W powyższym przykładzie ważniejszym aspektem niż posiadanie możliwości użycia umiejętności specjalnej jest wsparcie sojuszników, jeśli to się powiedzie wtedy rozpoczynamy ucieczkę, jeśli jednak ilość sojuszników jest większa niż 5, ale mamy na uwadze fakt, że wciąż mamy mniej niż 30 punktów życia to sprawdzamy czy istnieje możliwość zadania silniejszego ataku, czyli użycia umiejętności specjalnej, aby osłabić wroga, jeśli mamy taką możliwość wtedy węzeł zwraca niepowodzenie i cała sekwencja również ma status niepowodzenia – nie rozpoczynamy ucieczki. Dopiero jeśli nie posiadamy w okolicy sojuszników ani nie mamy możliwości użycia umiejętności specjalnej rozpoczynamy ucieczkę.

Użycie selektora można przedstawić jako analogiczne do funkcji bramki OR w obwodach.

- Losowa sekwencja / selektor (ang. Random Sequences / Selectors, PandaBT: random sequence/random selector) – działają identycznie w stosunku do swoich imienników. Ich jedyną różnicą jest kolejność odtwarzania węzłów – odbywa się to sposób losowy. Znajdą one zastosowanie tam, gdzie nie mamy wyraźnie preferowanej kolejności wykonywania pewnych akcji, dzięki temu można nadać pewnego aspektu nieprzewidywalności obiektom ze sztuczną inteligencją. Przykładowo przy projektowaniu atakowania możemy użyć kilku rodzajów ataku o różnej mocy i wykorzystać je tak, aby cała animacja nie wydawała się w żadnym momencie zapętlona.

Istnieją także:

- Węzeł równoległy (ang. Parallel, PandaBT: paralel) – odpowiada za uruchomienie wszystkich węzłów podrzędnych dokładnie w tym samym czasie. Odnosi powodzenie, jeśli wszystkie węzły podrzędne również odniosły powodzenie. Odnosi niepowodzenie na pierwszym węźle podrzędnym, który odniósł niepowodzenie.
- Węzeł „wyścigu”/przebiegu (ang. Race, PandaBT: race) – również odpowiada za uruchomienie wszystkich węzłów podrzędnych dokładnie w tym samym czasie. Odnosi powodzenie przy pierwszym węźle podrzędnym, który również odniósł powodzenie. Odnosi niepowodzenie, jeśli wszystkie jego węzły podrzędne odniosły niepowodzenie.

Rzadziej wykorzystywane są węzły Dekoratora:

- Inwerter (ang. Inverter, PandaBT: not) – odwraca lub neguje wynik węzła podrzędnego. Posiada tylko jeden węzeł podrzędny. Często używany zamiast instrukcji warunkowych.
- Następca (ang. Succeder, PandaBT: mute) – zawsze zwraca powodzenie do węzła nadrzędnego, niezależnie od wyniku węzła podrzędnego. Posiada tylko jeden węzeł podrzędny. Jest przydatny w bardziej rozbudowanych drzewach, gdzie możliwe jest odniesienie niepowodzenia bądź awarii, jednak pożądane jest, aby nie rezygnować z sekwencji następnych działań, w których znajduje się dana gałąź.

- Wzmacniacz/Przekaznik (ang. Repeater, PandaBT: repeat) – będzie przetwarzał swój węzeł podrzędny ponownie za każdym razem, kiedy zwróci on wynik. Przykładowo może uruchamiać on węzeł podrzędny określoną liczbę razy lub znaleźć zastosowanie u podstawy drzewa, aby pracowało ono nieprzerwanie.
- While (PandaBT: while) – posiada dwa węzły podrzędne. Pierwszy węzeł podrzędny jest warunkiem, natomiast drugi jest akcją/działaniem. Powtarza akcję, dopóki warunek jest prawdziwy. Kończy się powodzeniem, jeśli również i akcja zakończy się powodzeniem. Odnosi niepowodzenie, jeśli którykolwiek z obu węzłów również zakończy się niepowodzeniem.

5.5 Mechanika gry

W tym podrozdziale zostanie opisana dokładniej mechanika gry wraz z podziałem na poszczególne funkcje.

5.5.1 Przeciwnicy

Każdy przeciwnik posiada charakterystyczne cechy takie jak ilość punktów życia, zadawane obrażenia, zasięg pościgu gracza oraz szybkość poruszania. Szybkość poruszania oraz zadawane obrażenia zostały wcześniej przypisane do każdego przeciwnika i nie ulegają zwiększeniu wraz z rozwojem rozgrywki. Ograniczenie szybkości poruszania się jest związane z jakością i zachowaniem realizmu wyświetlanych animacji. Przeciwnicy typu Arachnid i Zwykły Mutant, poruszają się jedynie biegając, posiadają jeden rodzaj ataku oraz wykonują atak do momentu śmierci swojej lub śmierci gracza. Natomiast przeciwnik typu Mutant Kryształowy posiada znacznie więcej zachowań.

Stworzenia pojawiające się w rozgrywce to:

- Arachnid – najczęściej pojawiające się stworzenie na mapie. Jest schowany najczęściej w wyższych trawach lub miejscach mocniej zacienionych. Jest

najszybszym przeciwnikiem, ale i najslabszym przeciwnikiem. Może być przyzywany przez najmocniejszego mutanta. AI stworzone z użyciem skończonego automatu stanów.

Zadawane obrażenia: 6

Punkty życia: 50

Prędkość: 6

Zasięg: 20



Rys.5.15 Arachnid (źródło własne).

- Zwyczajny mutant – podstawowy przeciwnik, często patrolujący teren w poszukiwaniu gracza. Generowany po wejściu gracza w strefę niebezpieczeństwa (tzw. „Danger Zone”). AI stworzone z użyciem skończonego automatu stanów.

Zadawane obrażenia: 20

Punkty życia: 100

Prędkość: 3.5

Zasięg: 15



Rys.5.16 Zwykły mutant (źródło własne).

- Mutant kryształowy – najrzadziej pojawiający się przeciwnik, ale i najmocniejszy. Znajdujący się najczęściej w okolicach punktach oznaczonych jako cele misji. Posiada możliwość przyzywania sojusznicznych jednostek – Arachnid. AI stworzone ostatecznie z użyciem drzewa behawioralnego.

Zadawane obrażenia atakiem podstawowym: 20

Zadawane obrażenia doskokiem: 40

Zadawane obrażenia uderzeniem: 25

Zadawane obrażenia dźgnięciem: 30

Punkty życia: 200

Prędkość chodzenia: 1

Prędkość biegania: 3

Prędkość czołgania się: 0.5

Zasięg: 35



Rys.5.17 Mutant kryształowy (źródło własne).

5.5.2 Gracz oraz system walki

Zadaniem gracza jest wypełnienie wszystkich misji w jak najkrótszym czasie i dotarcie do łodzi, która zabierze go z wyspy.

- Gracz jest wyposażony od początku gry w broń palną.
- Istnieje możliwość zatrzymania rozgrywki w dowolnym momencie.
- Dostępne są 3 rodzaje broni – każda z oddzielnym rodzajem amunicji.
- Początkowo po 10 sztuk każdej z amunicji.
- Główny bohater jest stale konfrontowany z różnego rodzaju stworzeniami.
- W zależności od otoczenia i swojego położenia gracz może zostać zaatakowany z każdej możliwej strony.
- Do przejścia całej gry musi wypełnić wszystkie misje.
- Posiada 100 jednostek życia.

5.5.3 Środowisko

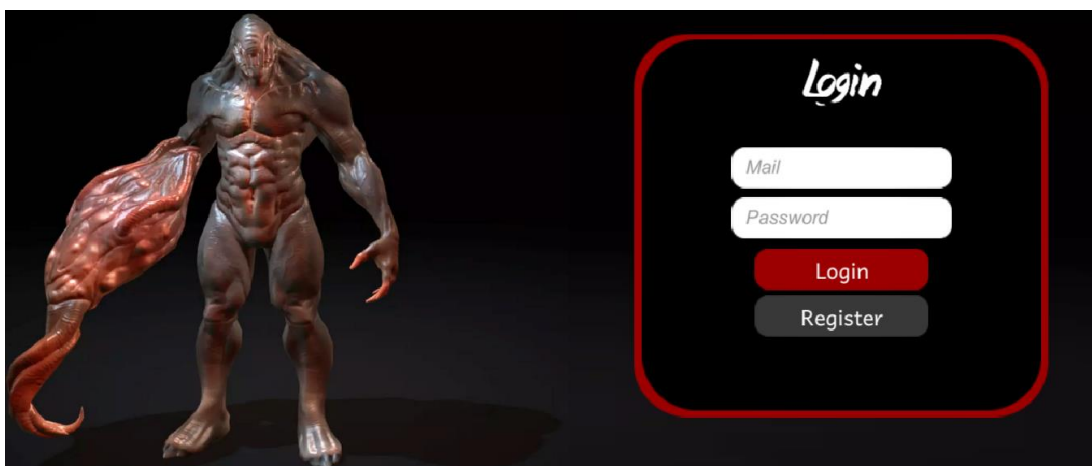
- W otoczeniu rozłożone w pewnych miejscach strefy niebezpieczeństwa (tzw. „Danger Zone”).
- Po wejściu gracza w strefę niebezpieczeństwa z losowego punktu generowani są przez 10 sekund przeciwnicy.
- Na mapie w różnych miejscach rozłożone są naboje pozwalające po zderzeniu z przeciwnikiem na uzupełnienie konkretnego rodzaju amunicji.
- W poszczególnych miejscach na mapie zlokalizowane są punkty umożliwiające leczenie się przeciwników.
- W poszczególnych miejscach na mapie zlokalizowane są punkty kontrolne wybierane przez przeciwników w sposób losowy do patrolowania terenu.
- Punkty, do których trzeba dotrzeć w celu realizacji punktu misji podświetlone są jaśniejszym kolorem.

5.6 Interfejs użytkownika

Gry oparte na silniku Unity składają się ze scen, które zawierają w sobie hierarchię obiektów. W celu wygodniejszej obsługi rozgrywki cały projekt składa się z dwóch scen – w której jedna odpowiada za grę, a druga odpowiada za menu wraz z ekranem rejestracji i logowania. Przy projektowaniu interfejsu położono nacisk na połączenie podobnej kolorystyki i zachowaniu stylu gry wraz z intuicyjnością i prostotą, tak aby całość nie stawała się przytłaczająca dla oka użytkownika.

5.6.1 Ekran logowania i rejestracji

Pierwszą sceną, która ukaże się po włączeniu gry jest menu główne wraz z widocznym ekranem logowania. Unity niestety udostępnia jedynie metody przycisku `OnClick()`, przejścia tabulatorem pomiędzy polami, jak i zatwierdzanie klawiszem `enter` zostaną zaprojektowane samodzielnie. Logowanie jest umożliwione za pośrednictwem adresu e-mail oraz stworzonego wcześniej hasła. Zalogowanie do gry wymaga aktywnego połączenia z Internetem.





Rys.5.18 Ekran logowania (źródło własne).

Weryfikacja poprawnie wprowadzanych danych następuje przy nawiązaniu połączenia z bazą Firebase, w której mamy także istniejący podgląd użytkowników tj. rys.5.19. Po poprawnym zalogowaniu ekran pojawia się zielony komunikat o powodzeniu operacji i po upływie dwóch sekund ekran znika. Przy niepoprawnym wprowadzeniu danych system informuje nas czerwonym komunikatem o błędzie.

Search by email address, phone number, or user UID

Add user

Identifier	Providers	Created ↓	Signed In	User UID
a@aaa.com		May 25, 2022	May 25, 2022	Ki56dLykgOVU7KLkh31rnqkMhI23
test@test.com		May 24, 2022	May 27, 2022	T0ImuW1LTJXPWrhvWTmZyJUcK...

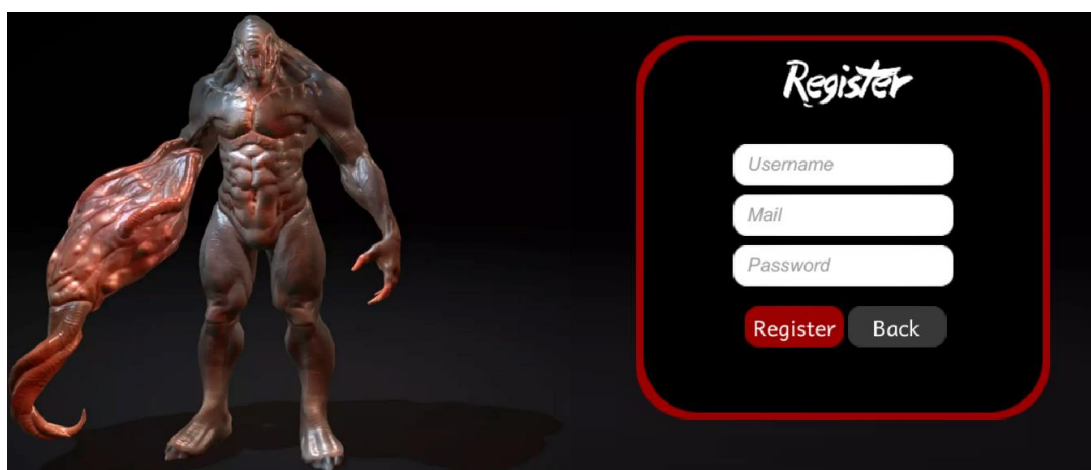
Rows per page:

50

1 – 2 of 2

Rys.5.19 Przykładowe zestawienie użytkowników (źródło własne).

Jeśli użytkownik nie posiada jeszcze konta w systemie, istnieje możliwość utworzenia go przy pomocy przycisku „Register”. Wprowadzone znaki ich długość weryfikowana jest w kodzie. Do rejestracji wymagana jest nazwa użytkownika, adres e-mail oraz hasło. Warunkiem utworzenia konta jest nie istniejący jeszcze adres e-mail i nazwa użytkownika w systemie oraz minimalna liczba znaków hasła musi wynosić 6. Po poprawnym zarejestrowaniu następuje przeniesienie do ekranu logowania. W przypadku błędu pojawia się informacja na dole okna.



Rys.5.20 Okno rejestracji (źródło własne).

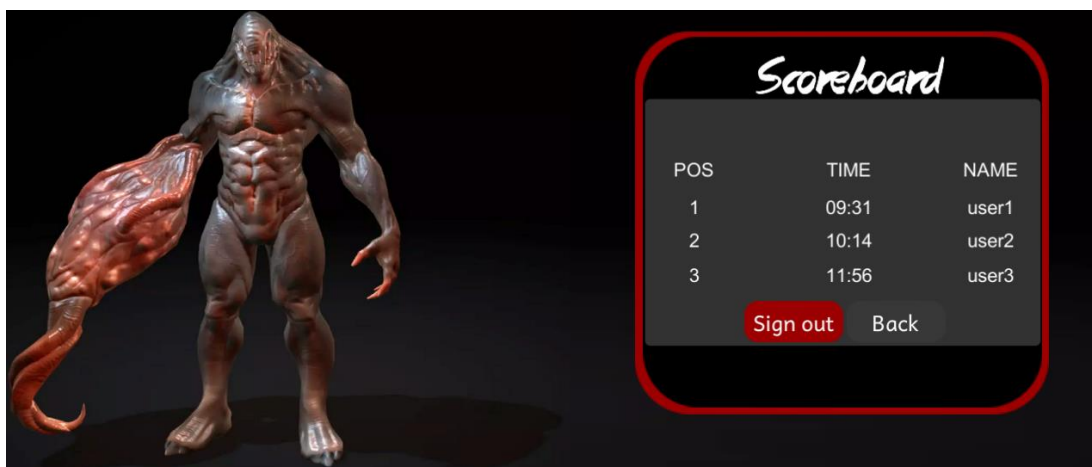
5.6.2 Menu główne

Po poprawnym zalogowaniu następuje przejście do menu głównego. W menu głównym istnieje możliwość rozpoczęcia rozgrywki, wyświetlenie opcji oraz wyjście z aplikacji.



Rys.5.21 Okno menu głównego (źródło własne).

W menu opcji istnieje możliwość wylogowania się z obecnego konta, a dodatkowo mamy możliwość podglądu wyników oraz nazw użytkowników domyślnie trzech najlepszych graczy w postaci tabeli wyników.



Rys.5.21 Okno menu głównego (źródło własne).

5.6.3 Interfejs wyświetlany w trakcie gry

Po uruchomieniu gry gracz zostaje przeniesiony do punktu startowego. W celu jak najlepszej jakości rozgrywki zminimalizowano interfejs widoczny podczas gry, celem zapewnienia graczowi lepszej widoczności. Rysunek 5.22 przedstawia niezbędne elementy zawarte w interfejsie użytkownika podczas trwania rozgrywki.



Rys.5.22 Widok gry (źródło własne).

Aktualny stan zdrowia jest wskaźnikiem punktów życia gracza umieszczony został na górze po lewej stronie. Czas rozgrywki mierzony jest w sekundach i został umieszczony na dole po prawej stronie, natomiast na górze po prawej stronie widoczna jest ilość amunicji, która odświeżana jest podczas zmiany broni. Dodatkowo na środku został umieszczony celownik ułatwiający trafienie w przeciwnika.

Podczas trwania gry gracz może wejść w strefę niebezpieczeństwa (tzw. Danger Zone). Podczas generowania przeciwników w celu skupienia uwagi gracza i przygotowania go na nadchodzącą falę przeciwników wyświetlane jest ostrzeżenie widoczne na rysunku 5.23 i znika po upływie czasu, w którym generowani są przeciwnicy.



Rys.5.23 Ostrzeżenie – wejście w strefę niebezpieczeństwa (źródło własne).

W każdym momencie gry istnieje także możliwość jej zatrzymania – służy do tego klawisz Esc, lub możliwość podglądu misji, które jeszcze musi wypełnić gracz, aby ukończyć grę – służy do tego klawisz M.



Rys.5.24 Menu widoczne po zatrzymaniu gry (źródło własne).



Rys.5.25 Widok misji (źródło własne).








Gra umożliwia podejście przeciwników z każdej strony, zatem aby poinformować gracza, że jest w tym momencie atakowany został nałożony specjalny obraz imitujący krew na kamerę widoczny na rysunku 5.25. Oprócz tego w momencie śmierci pojawia się komunikat, a także możliwość rozpoczęcia rozgrywki od początku lub opuszczenia gry.













Rys.5.26 Widok zakończenia rozgrywki (źródło własne).

5.7 Układ katalogów

Ten podrozdział zawiera opis katalogów niezbędnych do prawidłowego działania gry.

-  Animations – folder zawierający wszystkie kontrolery animacji dla stworzeń zbudowanych z wykorzystaniem skończonego automatu stanów.
-  Assets_Pack – folder zawierający zainstalowane pakiety zewnętrzne.
-  Creatures_Pack – folder zawierający obiekty stworzeń wykorzystanych w projekcie. Każde ze stworzeń posiada własne podfoldery wraz z animacjami, kontrolerami animacji, teksturami, modelami, prefabami oraz shaderami.
-  ExternalDependencyManager – zawiera pliki rozszerzenia obsługujące dowolną wtyczkę Unity. Zarządza wersjami zależności przechodnich.
-  Firebase – zawiera pliki konfigurujące działanie pakietu Firebase. Odpowiada za połączenie z bazą. Posiada także podfoldery z plikami, umożliwiającymi przekonfigurowanie bazy na system IOS lub Android.
-  GooglePackages – folder ten zawiera pliki z rozszerzeniem .tgz odpowiedzialne za działanie pakietów od Google tj. ExternalDependencyManager czy Firebase App.
-  Images – folder ten zawiera wszystkie obrazy wykorzystane przy tworzeniu interfejsu użytkownika.

-  Materials – katalog ten zawiera materiały wykorzystane w świecie gry oraz nałożone na obiekty.
-  Panda.BT.free – folder ten zawiera pliki niezbędne do działania behawioralnych drzew zachowań oraz przykładowe sceny z ich użyciem.
-  Plugins – katalog zawierający wykorzystane wtyczki zewnętrzne posegregowane w odpowiednich podfolderach.
-  Prefabs – katalog zawierający obiekty i pliki prefab z podziałem na elementy świata, interfejs, efekty i postacie.
-  ProBuilder Data – zawiera wszystkie pliki umożliwiające korzystanie z ProBuilder'a. Probuilder umożliwia budowanie i tworzenie całych obiektów bezpośrednio w Unity.
-  Scenes – folder ten zawiera wszystkie sceny wykorzystane w grze.
-  Scripts – katalog zawiera wszystkie skrypty oraz drzewa behawioralne.
-  Standard Assets – jest to pakiet zewnętrzny zawierający obiekty świata zewnętrznego. Nie został on umieszczony w folderze Assets_Pack ze względu na jego częste użycie przy rozbudowie świata.
-  StreamingAssets – folder zawierający pliki json z konfiguracją bazy - niezbędny do przeprowadzania procesu uwierzytelniania.
-  TestsScripts – zawiera skrypty bazy danych, pliki referencyjne oraz testy automatyczne.

6. Implementacja

Ten rozdział zawiera opisy poszczególnych elementów implementacji. Do jej wykonania zostało wykorzystane zintegrowane środowisko Unity w wersji 2020.3.5f. Skrypty były natomiast pisane w języku C# i napisane zostały z wykorzystaniem Microsoft Visual Studio 2019.

6.1 Wykorzystane assety

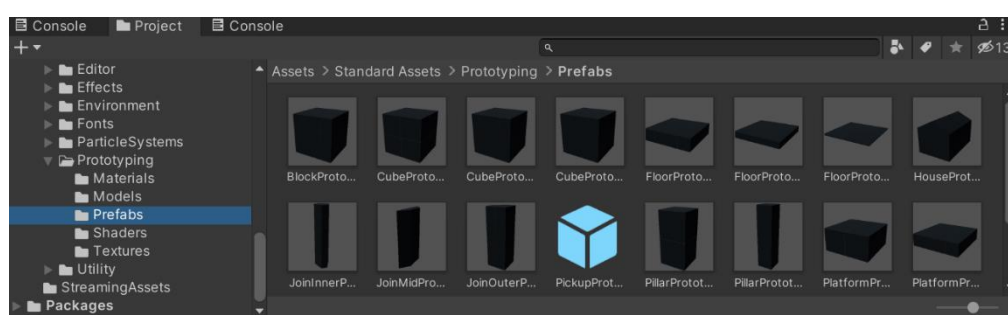
Unity Asset Store zawiera wiele pomocnych i pomagających zaoszczędzić czas pakietów, które zostały wykorzystane w grze. Do nich należą:

- *Standard Assets (for Unity 2018.4)* – asset wyjątkowo pomocny przy prototypowaniu i stworzeniu pierwszych, podstawowych scen.
- *15 Original Wood Texture* – zawiera 15 tekstur imitujących drewno.
- *AK-74 (HDRP)* – zawiera gotowy prefab karabinka oraz gotowy model i teksturę.
- *Ammunition pack (demo)* – zawiera różnego rodzaju amunicje.
- *Autumn Mountain* - zawiera krajobrazy gór.
- *Beautiful Progress Bar Free* – zawiera paski postępu.
- *Guns Pack: Low Poly Guns Collection* – zawiera podstawowe bronie.
- *Flooded Grounds* – posiada gotowe elementy środowiska tj. drzewa, łódzie, samochody, budynki i wiele innych.
- *FREE Rigged Skeleton and Bone Collection* – zawiera zestaw gotowych szkieletów.
- *Grass Flowers Pack Free* – zawiera zestaw traw oraz kwiatów możliwych do implementacji w terenie.
- *Panda BT Free* – pozwala na stworzenie drzew behawioralnych.
- *PBR Creatures (Pack)* – jedyny płatny asset, zawiera gotowy pakiet stworzeń wraz z kilkoma podstawowymi animacjami.
- *Rust Key* – zawiera gotowy obiekt klucza.

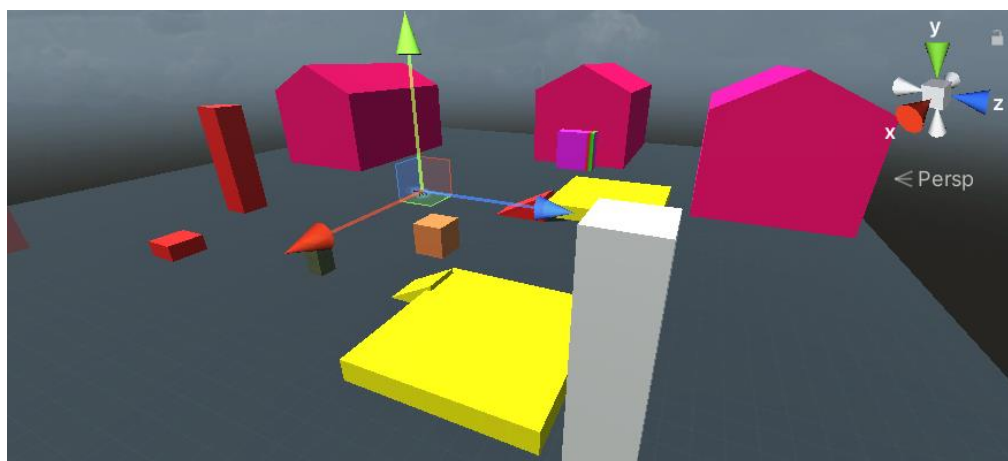
6.2 Stworzenie obsługi gracza i prototypowanie

Stworzenie pierwszoosobowej perspektywy gracza było możliwe dzięki wykorzystaniu kontrolera dostępnego w pakiecie Standard Assets, razem z kontrolerem dostępny jest także skrypt pozwalający na edycję podstawowych parametrów gracza tj. prędkość poruszania się w przód/tył, ustawienia kamery, jej intensywności itd. Poruszanie gracza jest możliwe dzięki wykorzystaniu klawiatury. Za ruchy typu: przód, tył, prawo, lewo odpowiadają strzałki, natomiast za skoki – przycisk spacji.

Jedną z najlepszych i wizualnych technik analitycznych jest prototypowanie. Pomaga ono uszczegółowić oraz dopracować potrzeby i wymagania, a także zoptymalizować doświadczenia użytkownika systemu w celu uzyskania jak najbardziej zadowalającego produktu końcowego. Istnieje wiele metod prototypowania, jednak najszybszym sposobem, który spełni wymagania będzie wykorzystanie gotowych obiektów zawartych w Standard Assets. W późniejszym etapie projektowania wykorzystany został ProBuilder – jest on narzędziem służącym do projektowania i modelowania poziomów 3D, wraz z możliwością tworzenia prostych geometrycznych obiektów.



Rys.6.1 Dostępne obiekty do prototypowania (źródło własne).



Rys.6.2 Obiekty prototypowania użyte w scenie (źródło własne).

Na potrzebę stworzenia ciekawszej, bardziej rozbudowanej gry graczowi zostały przydzielone trzy rodzaje broni – Assault Rifle, Shotgun oraz Pistol. Do każdej z nich zostały zaprojektowane oddzielne amunicje analogicznie wymienione w kolejności – Rockets, Shells, Bullets. Typy broni różnią się od siebie czasem pomiędzy kolejnymi

wystrzałami oraz mocą zadawanych obrażeń. Zmiana broni możliwa jest z wykorzystaniem klawiszy numerycznych lub przewijaniem rolki myszy. Każda broń posiada ograniczoną liczbę pocisków, które mogą być uzupełniane przez gracza podczas podniesienia paczek z amunicją. Istnieją trzy rodzaje paczek – każda posiada jeden rodzaj amunicji, zwiększając jej zapas o konkretną ilość.



Rys.6.3 i 6.4 Obiekt karabinka i odpowiadająca mu amunicja.

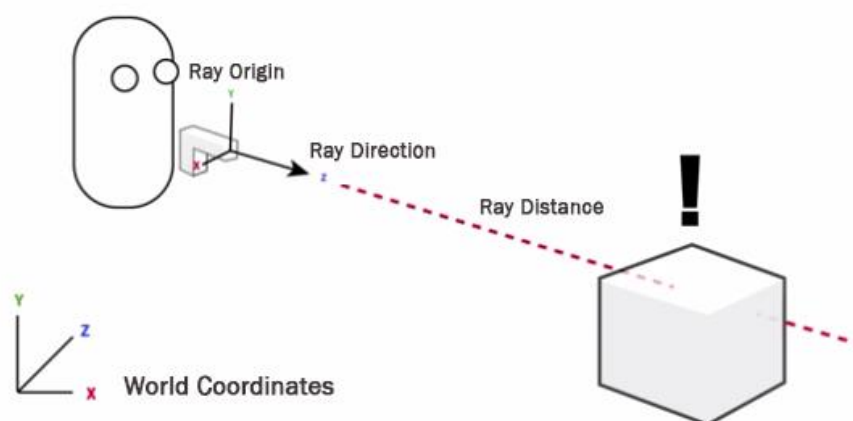


Rys.6.5 i 6.6 Obiekt strzelby i odpowiadająca mu amunicja.



Rys.6.7 i 6.8 Obiekt pistoletu i odpowiadająca mu amunicja.

Podczas strzelania został wykorzystany Raycasting. Jest to proces generowania niewidzialnego promienia z konkretnego punktu, w pewnym określonym kierunku, w celu sprawdzenia i wykrycia innych obiektów znajdujących się na jego drodze. Umożliwia to dostosowanie według potrzeb zasięgu strzału każdej z broni.



Rys.6.9 Raycasting – wizualizacja (źródło: <https://medium.com/@miguel.araujo/raycast-what-the-hell-is-that-6d36b3c8dd8b>)

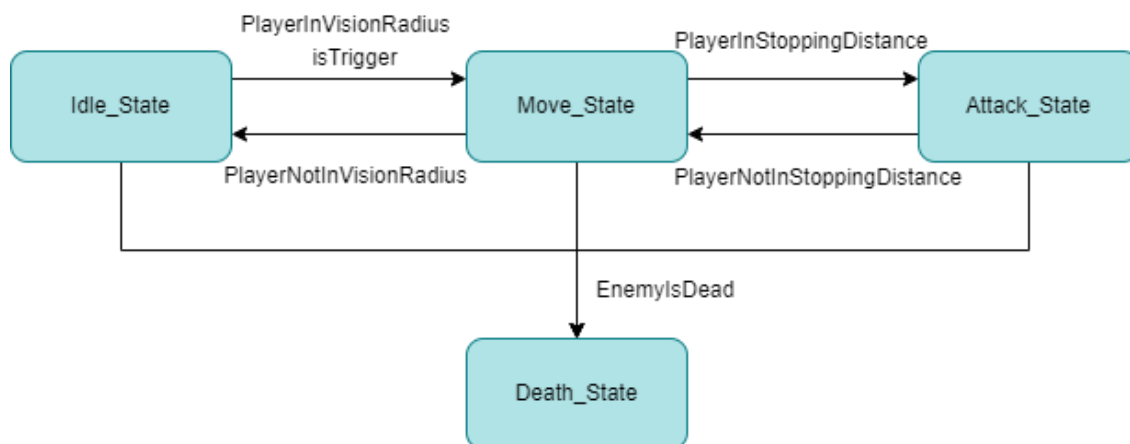
6.3 Implementacja przeciwników

Projektowanie i stworzenie przeciwnika również wymagało wykonania początkowo prototypowania – został dodany obiekt kapsuły wraz z podłączonym agentem siatki nawigacyjnej oraz ustawieniem jej celu na gracza. Aby działanie agenta było możliwe konieczne było dodanie wcześniej siatki nawigacji oraz wyznaczenie jej parametrów. Każdy z przeciwników umieszczanych na scenie wyposażony jest w zakres zasięgu widzialności gracza, tak, aby dopiero po znalezieniu się w nim gracz rozpoczynał się pościg. Dla ułatwienia pracy została zaprojektowana metoda `OnDrawGizmosSelected()` pozwalająca na zwizualizowanie na scenie zasięgu widzenia w postaci sfery.

Jedynym z możliwych warunków rozpoczęcia pościgu gracza jest jego wejście w zasięg widzenia. Oprócz tego każdy przeciwnik może zostać sprowokowany przez atak gracza, jeśli tylko jego zasięg widzenia jest mniejszy od zasięgu strzału gracza.

Do stworzenia zachowań dwóch rodzajów przeciwników typu Arachnid i Zwykły Mutant został wykorzystany automat stanów skończonych, a aktualizowanie ich stanów odbywa się poklatkowo w funkcji Update(). Oba rodzaje przeciwników zostały zaprojektowane w taki sposób, aby najprościej pokazać działanie automatu stanów skończonych, które najlepiej sprawdza się właśnie przy nieskomplikowanych zachowaniach oraz takich, które nie będą wymagać dużego rozbudowania w przyszłości. Jest to dobry wybór, ponieważ tego rodzaju zachowanie jest proste do zrozumienia, implementacji oraz wdrożenia. Ich cykl zachowań ogranicza się do 4 rodzajów stanów: atakowanie, stan bezczynności, bieganie, stan śmierci.

Rysunek 6.10 pokazuje maszynę stanów dla przeciwnika rodzaju Arachnid. Każdy stan posiada możliwość przejścia do innego stanu i powrotu do niego z wyjątkiem stanu śmierci, który jest już nieodwracalny, natomiast może on nastąpić z dowolnego stanu. Również animacja stanu śmierci wykonuje się tylko raz. Po przejściu w ten stan obiekt wyłączany jest z siatki nawigacji i pozostaje w tym samym miejscu, w którym poziom jego zdrowia osiągnął wartość 0.



Rys.6.10 Skończony automat stanów przeciwnika Arachnid.

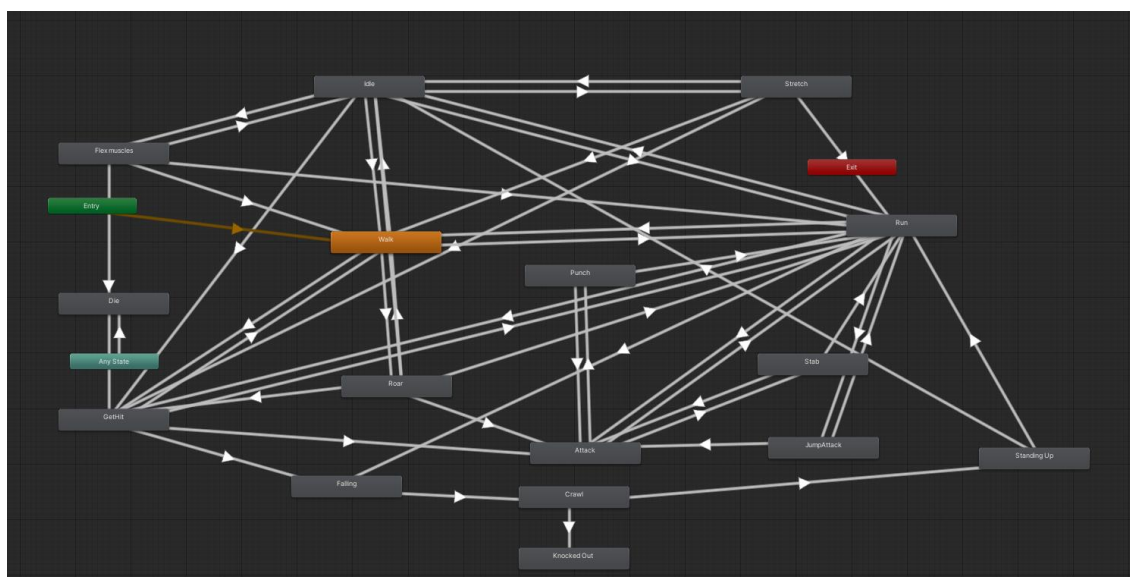
- Stan bezczynności (Idle_State): jest stanem domyślnym, wyświetla powtarzającą się animację oddychania przeciwnika. Może zostać zakończony przez wykrycie przeciwnika lub śmierć przeciwnika.

- Stan ruchu (Move_State): wyświetla powtarzającą się animację biegania. Następuje po stanie bezczynności, kiedy gracz znajdzie się w zasięgu widoczności lub kiedy zostanie sprowokowany, czyli zostaną mu zadane obrażenia ze strony gracza. Może nastąpić także po stanie ataku, kiedy gracz oddali się za bardzo od przeciwnika. W tym stanie przeciwnik będzie ścigał cel, dopóki nie dotrze do gracza. Zakończony jest w momencie dotarcia do przeciwnika i znalezienia się go w zasięgu ataku, który jest równy dystansowi zatrzymania ustawionego agenta siatki lub w momencie śmierci przeciwnika.
- Stan ataku (Attack_State): wyświetla powtarzającą się animację ataku. Następuje w momencie znalezienia się przeciwnika w zasięgu zatrzymania od gracza. Kończy się w momencie kiedy gracz oddala się od przeciwnika i przekształca się w stan pościgu lub kiedy przeciwnik umiera.
- Stan śmierci: może być nazwany inaczej każdym stanem (ang. any state), ponieważ każdy inny stan posiada przejście do niego i podczas jego trwania wszystkie inne stany są wyciszane.

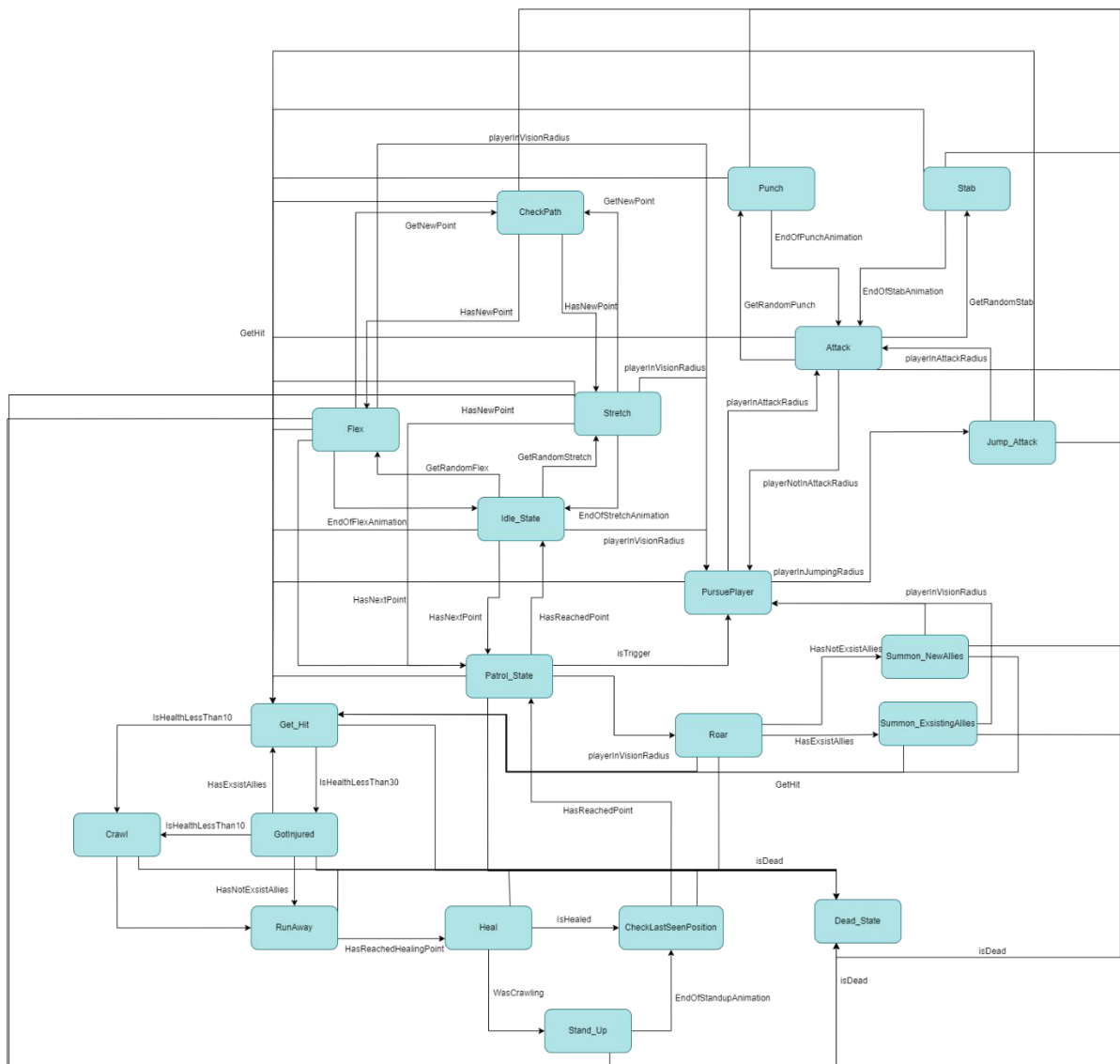
Automat stanów skończonych Zwykłego Mutanta działa w sposób analogiczny. Wraz z rozpoczęciem działania cyklu, który rozpoczyna się w momencie startu gry lub w momencie zainicjowania obiektu przeciwnika w scenie, rozpoczyna się pierwszy stan bezczynności. Później w zależności od tego czy przeciwnik jest w stanie zobaczyć gracza podejmowane są dalsze akcje, jeśli tak to wywoływana jest metoda agenta siatki SetDestination() i rozpoczyna się pogoń gracza, jeśli nie to w przypadku Arachnida następuje przejście do stanu bezczynności i oczekiwanie na gracza, a w przypadku Zwykłego Mutanta następuje przeszukiwanie środowiska w celu odnalezienia punktów do patrolowania. W przypadku odnalezienia punktów na mapie Zwykły Mutant przechodzi do funkcji Patrol() i porusza się po losowo wybranych punktach w celu odnalezienia gracza. Po wejściu gracza w zakres widzenia stworzenia lub sprowokowanie go poprzez strzał rozpoczyna się pościg. Jednorazowo sprowokowany przeciwnik goni cel, aż do momentu śmierci swojej lub gracza. Każdy obiekt gry będący agentem siatki nawigacji posiada dystans zatrzymania - gracz jest atakowany, jeśli znajduje się w jego zakresie.

6.4 Implementacja z wykorzystaniem drzewa behawioralnego

Ostatnie stworzenie zostało najbardziej rozbudowane, aby pokazać przydatność drzewa behawioralnego. Walka z tym przeciwnikiem powinna stanowić wyzwanie dla gracza, dlatego tuż przed fizyczną lokalizacją każdego obiektu wymaganego do odnalezienia do wypełnienia misji zlokalizowany został Mutant Kryształowy. Początkowy zestaw zachowań Mutanta Kryształowego miał być identyczny jak w pierwszych dwóch przeciwników, dlatego pierwsza i skrócona jego implementacja została wykonana za pomocą automatu stanów skończonych. Z czasem natomiast pojawiły się problemy – przy dodawaniu każdego ze stanów trzeba było uwzględnić każdy inny. Proces ten był nie tylko czasochłonny, ale także niezwykle skomplikowany do realizacji. Już na samym początku pojawiał się problem z zaprojektowaniem odpowiedniego przełączania się animacji ze względu na ilość pełnionych funkcji rysunek 6.11 przedstawia wygląd okna animacji Mutanta Kryształowego. Późniejsze problemy wynikały ze wskazania kolejności wykonywania stanów oraz warunków przełączania pomiędzy nimi w przypadku kiedy kilka miało wartość prawdziwą. Maszynę stanów widoczną na rysunku 6.12 należało również zsynchronizować z animacjami.



Rys.6.11 Okno animacji Mutanta Kryształowego



Rys.6.12 Skończony automat stanów przeciwnika Mutant Kryształowy.

W związku z trudnościami implementacyjnymi zrezygnowano z takiego podejścia i zastąpiono je drzewem behawioralnym, aby jednak móc korzystać z zaprojektowanych już klas, część niezbędnych metod oraz funkcji tj. np. komponent EnemyHealth – odpowiadający za punkty życia przeciwnika, została przeniesiona do innych skryptów możliwych do wykorzystania ponownie w przypadku innych postaci. Drzewo behawioralne ułatwiło rozbudowanie aktualnych stanów o dodatkowe oraz pozwoliło na utworzenie bardziej skomplikowanych zachowań i gałęzi, a także umożliwiło usunięcie zawartości funkcji Update() i rozbicie jej na poszczególne zadania

(ang. Tasks). Za obsługę tych zadań stało się odpowiedzialne drzewo zachowań widoczne na rysunku 6.11.

```
tree("Root")
  parallel
    repeat mute tree("BeAlive")
    repeat mute tree("Die")

tree("BeAlive")
  fallback
    tree("AvoidPlayer")
    while not tree("IsThreatened")
      fallback
        tree("DoTasks")
```

```

tree("DoTasks")
  fallback
    while
      sequence
        not tree("IsThreatened")
        not PlayerInVisionRadius
        not PlayerInAttackRadius
      fallback
        while
          not CheckPath
          Patrol
        while
          CheckPath
          sequence
            Rest
            random
              Flex
              Stretch
            Wait 5.0
            BackToPatrol
    while
      sequence
        not tree("IsThreatened")
        PlayerInVisionRadius
        not PlayerInAttackRadius
      sequence
        SawPlayer
        FindAllies
        parallel
          PursuePlayer
          sequence
            CanJumpToPlayer
            JumpToPlayer
    while
      sequence
        not tree("IsThreatened")
        PlayerInVisionRadius
        PlayerInAttackRadius
      sequence
        sequence
          not hasPreviouslyAttack
          BasicAttack
          random
            sequence
              not hasPreviouslyStab
              StabPlayer
            sequence
              not hasPreviouslyPunch
              PunchPlayer

```

```

tree("AvoidPlayer")
  sequence
    while
      fallback
        tree("GotInjured")
        tree("IsThreatened")
      sequence
        SetDestination_Random
        IsDirectionSafe
        MoveToDestination
        while
          not isHealed
            sequence
              Heal

tree("GotInjured")
  sequence
    IsHealthLessThan(10.0)
    IsHealthGreaterThan(0.01)
    Crawl

tree("IsThreatened")
  sequence
    IsHealthLessThan(30.0)
    mute LastSeenPosition
    not HasAllies

tree("Die")
  sequence
    IsHealthLessThan(0.01)
    Die

```

Rys.6.13 Drzewo behawioralne Mutanta Kryształowego.

Opis działania drzewa:

- Drzewo „Root” – jest drzewem wykonującym się jako pierwsze, powtarzającym nieustannie dwa inne drzewa – „BeAlive” oraz „Die”.
- Drzewo „BeAlive” – można w tym przypadku powiedzieć, że głównym zadaniem istoty jest przetrwanie, dlatego dopiero w momencie, w którym nie jest on zagrożony przechodzi do wykonywania swoich zadań – drzewa „DoTasks”.
- Drzewo „DoTasks” – sprawdzane są nieustannie warunki czy stworzenie jest zagrożone, czy nie widzi gracza, czy nie został zaatakowany przez gracza oraz czy gracz nie jest w zasięgu ataku – dopiero wtedy mutant przechodzi do patrolowania terenu. Patrolowanie odbywa się po wyznaczonych wcześniej

punktach, które są przez niego losowo wybierane. Po dojściu do wybranego punktu następuje kolejne losowanie, w trakcie którego obiekt przechodzi do stanu beczynności i losowo wybierane jest przeciąganie się lub rozciąganie mięśni. Po wybraniu punktu następuje wyjście ze stanu beczynności po odczekaniu 5 sekund i powrót do przemieszczania się do wybranego punktu.

Jeśli gracz znajdzie się w zasięgu wzroku przeciwnika następują kolejno po sobie funkcje takie jak dostrzeżenie gracza podczas której gracz obierany jest jako cel oraz przyzwanie sojuszników. Funkcja FindAllies() odpowiada za zwerbowanie do wspólnego ataku przeciwników Arachnid znajdujących się w pobliżu lub jeśli ich liczba jest mniejsza od 5 generowanie nowych na mapie – powoduje to atakującą gracza falę przeciwników. Mutant Kryształowy pamięta także jaka ilość sojuszników została ostatecznie przywołana. Po przyzwaniu sojusznicznych jednostek następuje pogon za graczem. Jeśli w trakcie jej trwania mutant znajdzie się w odpowiedniej odległości do skoku – mutant wykona go skacząc na przeciwnika i zadając podwójne obrażenia w stosunku do podstawowych. Po tym ataku nastąpi przejście do ataku podstawowego, który jest wykonywany losowo na zmianę wraz z atakiem typu dźgnięcie i uderzeniem, gdzie sprawdzane są warunki czy minęła odpowiednia ilość czasu od ostatniego uderzenia.

Jeśli mutant zostanie zaatakowany zanim dostrzeże gracza, nie przyzywa on sojusznicznych oddziałów, a od razu przechodzi do pogoni za graczem. Reszta kroków następuje analogicznie do podanych powyżej.

- Drzewo „isThreatened” – jest drzewem, które sprawdza w sekwencji warunki tj.: czy ilość punktów życia jest mniejsza od 30, następnie zapamiętuje pozycję gracza i sprawdza czy nie posiada sojuszników. Gdy podane warunki są spełnione drzewo zwróci powodzenie.
- Drzewo „AvoidPlayer” – następuje w momencie, kiedy stworzenie jest zagrożone, czyli spełnione jest drzewo „isThreatened”. Mutant porzuca wszelkie aktualne czynności i oblicza kąt padania wzroku (w tym wypadku kamery) przeciwnika. Dzięki temu jest w stanie wybrać na mapie jeden z utworzonych wcześniej ukrytych punktów służących do leczenia. Stworzenie wybiera najbliższy punkt leżący poza wzorkiem wzrokiem gracza i obiera go jako swój cel poruszania się. W trakcie drogi może on nadal przyjmować obrażenia i zmienić sposób poruszania z biegania w czołganie w przypadku krytycznego

poziomu punktów życia. Po dotarciu do wyznaczonego wcześniej celu mutant leczy się. Kiedy wszystkie jego punkty życia zostaną przywrócone, stworzenie powraca do patrolowania terenu, jednak pierwszym punktem staje się ostatnia pozycja w jakiej widział gracza.

- Drzewo „GotInjured” – sprawdza czy przeciwnik ma mniej punktów życia od 10 i większą od 0.01, jeśli te warunki są spełnione poruszanie stworzenia zmienia się w czołganie.
- Drzewo „Die” – sprawdza czy ilość punktów życia jest mniejsza od 0.01, jeśli tak mutant ginie.

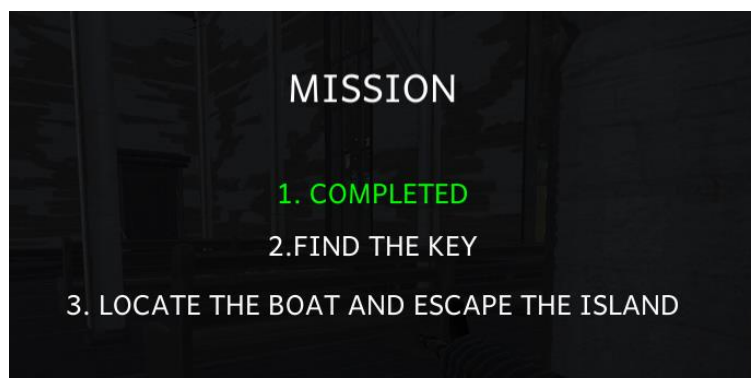
6.5 Świat gry oraz system misji

Środowisko w grze składa się z wyspy osadzonej na terenie wodnym. W terenie wyznaczone zostały ścieżki, które mają na celu ułatwienie graczowi poruszanie się po mapie, a także dojście do punktów kluczowych w misji. Granice poruszania się gracza zostały wyznaczone przez wzniesienia terenu, które zostały przekształcone w teren górzysty. Na mapie w celu poprawy jakości rozgrywki zostały umieszczone różnego rodzaju budynki np. kościół, szklarnia, latarnia morska, cmentarz, willa, budynki mieszkalne. Oprócz wspomnianych wcześniej przeciwników i skrzynek z amunicją dodatkowo w terenie zostały rozmieszczone inne obiekty np. drzewa, krzewy i innego typu roślinność, jaskinie, skały, mosty, samochody, łodzie, studnie, obiekty wyposażenia domów. Rysunek 6.14 przedstawia mapę gry wraz z rozkładem obiektów.



Rys.6.14 Mapa gry.

Zadaniem gracza jest zrealizowanie trzech misji tj.: odnalezienie kościoła, odszukanie w nim klucza do łodzi oraz znalezienie samej łodzi. W każdym momencie w grze istnieje możliwość podglądu misji oraz jej statusu. Zrealizowane podpunkty zostają odznaczone kolorem zielonym. Rysunek 6.15 przedstawia podgląd misji po zrealizowaniu pierwszego podpunktu. Gra kończy się w momencie, gdy gracz wykona wszystkie misje.



Rys.6.15 Podgląd misji po zrealizowaniu pierwszego podpunktu.

Dla ułatwienia każdy obiekt wymagany do odnalezienia w misji jest na mapie podświetlony – indywidualnie każdy z obiektów posiada swój własny obiekt oświetlenia. Rysunek 6.15 pokazuje przykładowy oświetlony obiekt.



Rys.6.14 Oświetlona łódź – obiekt niezbędny do odnalezienia przez gracza.

7. Testowanie

Wraz z rozwojem projektu zwiększa się liczba plików i czynności koniecznych do zweryfikowania ich działania. Pojawiają się dodatkowe skrypty, metody czy klasy i w tym przypadku do ułatwienia pracy i oszczędzania czasu przydatne okazują się narzędzia pomagające wykryć na wczesnym etapie błędy. Aby sprawdzić, czy wszystko działa zgodnie z oczekiwaniami można posłużyć się pomocą narzędzi do testowania. Do łatwej i szybkiej identyfikacji błędów Unity udostępnia specjalny pakiet pomagający przeprowadzić użytkownikowi testy automatyczne. Jest to Unity Test Framework, który pozwala na testowanie kodu zarówno w trybie gry jak i edycji. W trakcie trwania prac najważniejsze elementy zostały przetestowane w przy pomocy testów jednostkowych.

8. Kierunki dalszego rozwoju

Ze względu na bezproblemową i szybką zmianę FSM na drzewo behawioralne w dalszym procesie rozwoju zakłada się nie tylko dodanie nowych przeciwników, ale także modyfikację i rozbudowanie już istniejących. Zastosowanie drzewa behawioralnego umożliwia rozbudowanie każdego z nich o nowe stany. Jednym z planów na przyszłość jest dodanie systemu poziomów trudności w zależności od rozbudowania przeciwników. Dodatkowo planowane jest dodanie systemu punktów – przeliczane byłyby rodzaje pokonanych przeciwników wraz z całkowitym czasem rozgrywki oraz przechowywanie ich w bazie i wyświetlanie w tabeli wyników. Przechowywanie tego typu danych pozwoliłoby także na wyświetlanie statystyk dla każdego z graczy. Systematyczne wprowadzanie nowych obiektów, misji, poziomów i przeciwników umożliwiłoby przyciągnięcie uwagi nowych graczy. Dzięki zbudowaniu gry z zastosowaniem silnika Unity istnieje również możliwość rozszerzenia gry na inne platformy.

9. Podsumowanie

Celem pracy było wykonanie analizy, projektu i implementacji gry, który został zrealizowany. Na początku wybrane zostały narzędzia, które były wykorzystane podczas tworzenia gry, następnie opracowany został projekt bazy oraz przedstawione zostały najważniejsze funkcjonalności Unity, które były pomocne przy stworzeniu gry, opisana również została w tym punkcie mechanika gry oraz zaprojektowany został interfejs użytkownika. Później miała miejsce faza implementacji, w której oprócz gracza powstały konkretne rodzaje przeciwników wraz ze swoimi unikatowymi zestawami zachowań połączone z szeregiem rodzaju animacji. Wykorzystane zostało zarówno drzewo behawioralne jak i FSM. Wynikiem przeprowadzonych prac jest zatem gra, spełniająca wszystkie założone funkcjonalności. Projekt wykazał także kilka zalet drzew zachowań w stosunku do automatu stanów skończonych są one przede wszystkim bardziej elastyczne, wydajne, pozwalają na zaprojektowanie znacznie bardziej rozbudowanych zachowań oraz są bardziej przychylne wprowadzaniu zmian.

Do wdrożenia drzewa zachowań niezbędna była jednak wtyczka PandaBT. Pomimo spełniania swojej funkcjonalności wtyczka nadal nie jest dostarczana wraz z

Unity, przez co jej działanie można uznać w pewnym sensie za ograniczone. Po dłuższym czasie nad projektem i dokładniejszym zgłębieniu analizy problemu większość znalezionych przeze mnie materiałów dotyczących drzew zachowań przedstawiona była z wykorzystaniem silnika Unreal Engine. Prawdopodobnie decydując się kolejny raz na projekt z wykorzystaniem drzew behawioralnych moim wyborem byłby silnik posiadający lepsze wsparcie dla konkretnie tego rodzaju metod wykorzystania sztucznej inteligencji.

WYKAZ LITERATURY

1. Michele Colledanchise i Petter Ögren, *Behavior Trees in Robotics and AI: An Introduction*, Taylor & Francis Ltd, 2020.
2. Wolfgang Ertel, *Introduction to Artificial Intelligence*, Springer, Ravensburg-Weingarten University of Applied Sciences, Germany, 29.01.2018.
3. Francesco Sapio, *Hands-On Artificial Intelligence with Unreal Engine*, Packt Publishing, 2019.
4. Dr. Davide Aversa, Aung Sithu Kyaw, Clifford Peters, *Unity Artificial Intelligence Programming*, Packt Publishing, 2018.