Politechnika Wrocławska

Wydział Informatyki i Zarządzania
kierunek studiów: Informatyka

Praca dyplomowa - inżynierska

Gra z wykorzystaniem wirtualnej rzeczywistości

Marek Derkowski Aureliusz

słowa kluczowe:
Rzeczywistość wirtualna
HTC Vive
Wave defense

krótkie streszczenie:
Celem pracy jest wykonanie projektu i implementacja aplikacji rozrywkowej, tj. gry komputerowej z wykorzystaniem gogli wirtualnej rzeczywistości – HTC Vive. W pracy przedstawiono konkurencyjne oprogramowanie, użyte w projekcie narzędzia oraz najbardziej interesujące aspekty szczegółów implementacyjnych.

opiekun pracy dyplomowej	.....	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

Ostateczna ocena za pracę dyplomową			
Przewodniczący Komisji egzaminu dyplomowego	.....	.....	.....
	Tytuł/stopień naukowy/imię i nazwisko	*ocena*	*podpis*

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**
a) kategorii A (akta wieczyste)
b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)
** niepotrzebne skreślić*

pieczątka wydziałowa

Wrocław 2017

Dla moich rodziców i Wiki…za ciągłe motywowanie.

Streszczenie

Przedmiotem wykonanej pracy jest implementacja gry z wykorzystaniem wirtualnej rzeczywistości. W tym celu została przeprowadzona analiza rynku gier wirtualnej rzeczywistości, a następnie implementacja gry wykorzystującej VR przy użyciu wybranych technologii – HTC Vive oraz Unreal Engine 4 - i narzędzi – Visual Studio, czy 3ds Max. W ramach pracy zostały sprawdzone istniejące rozwiązania, jak również silnik gier, pod kątem użycia ich w tworzonej aplikacji. Praca zawiera również skrócony opis historii wirtualnej rzeczywistości, od początku powstania technologii do dnia dzisiejszego. Dodatkowo przedstawiona została ewolucja gier typu Wave Defense i jej duchowego spadkobiercy - Tower Defense. Można w niej znaleźć również wnioski i komentarze dotyczące perspektyw rozwoju technologii wirtualnej rzeczywistości w przyszłości, jak i implementowanej aplikacji. Konkluzje zawierają również sposób na wielopłaszczyznowy rozwój projektu. Podczas pisania pracy następowały modyfikacje pierwotnego projektu związane z testowaniem różnych rozwiązań, w celu wyboru optymalnego dla założonego projektu. Związane z tym były pewne komplikacje, z których te, dotyczące opisywanego projektu, zostały rozwiązane. Nie do pokonania okazały się natomiast ograniczenia sprzętowe, których nie wyeliminowała lepsza implementacja. Wszystkie mechaniki gry zostały doprowadzone do poziomu, który pozwala przedstawić prototypowy projekt potencjalnemu odbiorcy.

Abstract

The subject of this work was the implementation of game using virtual reality. For this purpose, an analysis of the virtual reality game market was carried out, followed by the implementation of the game using VR with selected technologies – HTC Vive and Unreal Engine 4 – and other tools - Visual Studio and 3ds Max. As part of the work, existing solutions and game engines, that allow creating virtual reality application have been checked. This paper also has a shortened history of virtual reality from the beginning of technology to present day. Additionally, the evolution of Wave Defense games and its spiritual heir – Tower defense was described. In the work, you can also find conclusions and comments about the prospects of virtual reality in the future, as well as the implemented application. The deductions also include a way for multi-faceted development of the project in the future. Writing the thesis was accompanied by modifications of the project related to testing of other solutions, in order to be able to choose the best for implemented software. At the same time, there were some complications and those related to the described project were solved, however, hardware limitations cannot be solved through better implementation. All game mechanics have been brought to the level where they present stable prototype project that can be released to the public one day.

Spis treści

Wstęp

Obecna dekada mija pod kątem rozwoju technologii związanej z wirtualną rzeczywistością. Trend ten z dużym prawdopodobieństwem będzie utrzymywał się jeszcze przez lata. Jest to możliwe dzięki coraz większej liczbie firm tworzących własne urządzenia i środkom, jakie na to przeznaczają. Wraz z rozwojem sprzętu, przychodzą również innowacyjne pomysły, jego wykorzystania. Pozwala to na interakcję z drugą osobą na niespotykaną dotąd skalę i choć na dzień dzisiejszy ich wymagania są bardzo wysokie – z każdą chwilą na rynku pojawiają się nowe systemy VR oraz procesory, czy karty graficzne, które z większą wydajnością potrafią je obsługiwać.

Celem pracy jest w pierwszej kolejności zapoznanie z historią technologii wirtualnej rzeczywistości, jak również analiza istniejących na rynku rozwiązań, oraz ich dostępność dla użytkownika. Następnie, wykorzystując uzyskaną wiedzę, zaprojektowanie oprogramowania, które będzie stanowiło jego komplementarną część oraz taką jego późniejszą fragmentaryczną implementacje, która pozwoli uzyskać w pełni działający prototyp. Jego przetestowanie pozwoli ocenić, czy produkt posiada odpowiednią wartość rynkową. Dzięki takiej wycenie będzie istniała możliwość kontynuowania produkcji w celu jego opublikowania. Prototyp zostanie wykonany przy użyciu systemu wirtualnej rzeczywistości – HTC Vive oraz framework Unreal Engine 4.

Zakres pracy obejmuje wykonanie aplikacji działającej na systemach operacyjnych Windows, na urządzeniach spełniających minimalne wymagania związane z technologią i narzucone przez system Vive. Projekt ma za zadanie umożliwić komfortowe korzystanie z wirtualnego łuku wraz z realistycznie odwzorowaną symulacją poruszania się strzały i ułatwieniu strzelania poprzez zainstalowanie holograficznych przyrządów celowniczych. Zadaniem gracza będzie obrona wyznaczonego obiektu na mapie przed nacierającymi falami przeciwników. Gra będzie wykonana z trójwymiarową grafiką stylizowaną pod względem kolorystycznym o zmniejszonej ilości renderowanych trójkątów (tak zwany styl low poly), w celu zmniejszenia wymagań sprzętowych.

Do utworzenia gry zostanie wykorzystane następujące oprogramowanie i sprzęt:

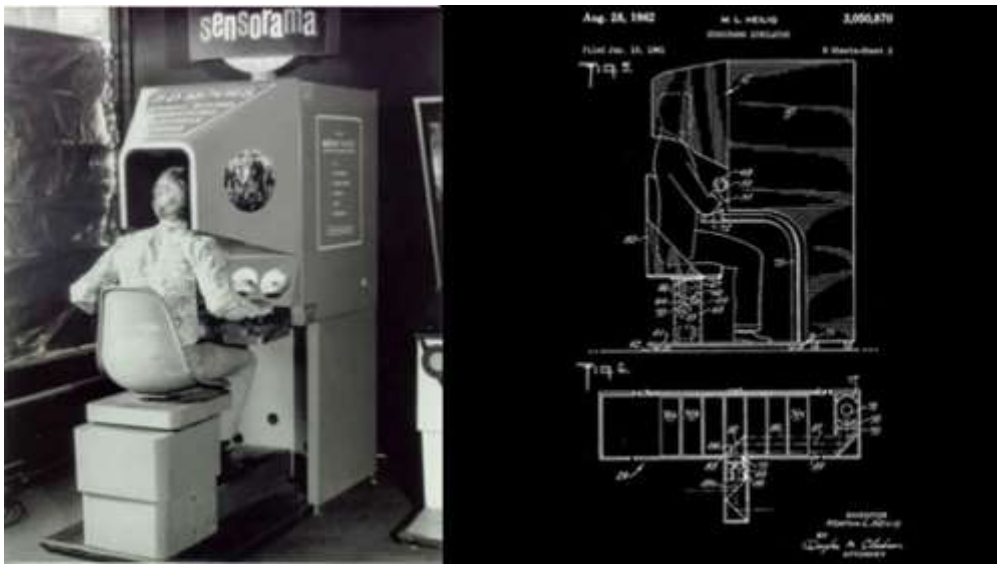
- Komputer stacjonarny:
    - Intel Xeon E3-1231 v3
    - 16 GB DDR3
    - Nvidia GeForce GTX 780
- Unreal Engine 4
- 3ds Max 2016
- Visual Studio 2015
- HTC Vive

Poniższa praca jest kompletnym opisem wykonanego projektu. Składa się z trzech rozdziałów, wstępu, zakończenia, wykazu literatury. W rozdziale pierwszym przedstawiono tematykę projektu wraz z historią wirtualnej rzeczywistości oraz dodatkowymi informacjami mającymi na celu uzasadnienia wyboru takiej tematyki projektu. Najważniejszą częścią pracy są rozdziały: drugi i trzeci. Rozdział drugi został poświęcony realizacji założeń teoretycznych, zgodnie z którymi została wykonana implementacja. W tym rozdziale znajdują się również istniejące już rozwiązania, które zostały dogłębnie przeanalizowane w celu zastosowania możliwie najlepszych w tworzonym projekcie. Natomiast w rozdziale trzecim omówiono proces implementacji projektu, dotykając zagadnień i problemów, które wystąpiły w trakcie jego realizacji. W zakończeniu podsumowano wykonaną pracę, jak również zawarto możliwe dalsze kierunki rozwiązań w tym zakresie. Praca kończy się wykazem cytowanej literatury i spisem rysunków.

1. Tematyka projektu

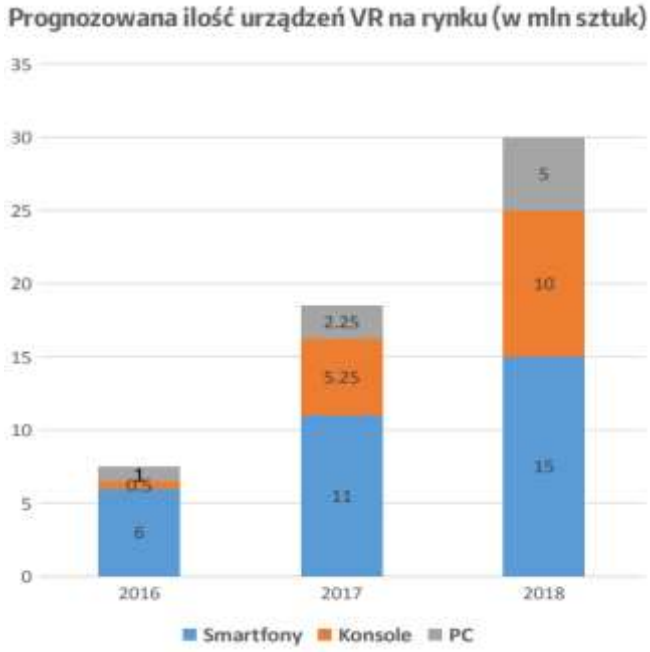
1.1. Wirtualna rzeczywistość

Początek „wirtualnej rzeczywistości" miał miejsce jeszcze w latach sześćdziesiątych ubiegłego wieku, kiedy to Myron W. Krueger we współpracy z Dan Sandinem, Jerry Erdmanem i Richardem Venezky utworzyli na Uniwersytecie Wisconsin-Madison projekt Glowflow. Była to instalacja składająca się z zaciemnionego pokoju, w którym znajdowały się przezroczyste rury. Dzięki użyciu fluorescencyjnych cząsteczek, zmieniano kolor zainstalowanych rur. Akompaniowały temu dźwięki syntezatora Mooga. Osoba znajdująca się w pomieszczeniu mogła się swobodnie poruszać, a jej ruchy były przetwarzane przez komputer i odpowiednie algorytmy, które modyfikowały dźwięki i światło w pokoju. Równolegle w innym miejscu powstawał prototyp tego, co dziś można nazwać goglami wirtualnej rzeczywistości. Miało to miejsce w roku 1962. Sensorama (rysunek 1), wykonana przez Mortona Heilig, była dużą i skomplikowaną maszyną, która zapewniała nie tylko wizję, ale również podmuch i wibrujące siedzenie. Maszyna odtwarzała nagrany film w perspektywie pierwszej osoby i w pierwotnej wersji umożliwiała „przejażdżkę" motocyklem po Nowym Jorku. Na definicję pojęcia Wirtualna Rzeczywistość, należało czekać do lat osiemdziesiątych, kiedy to Jaron Lanier spisał jej treść – „Virtual Reality is the use of computer technology to create the effect of an interactive three-dimensional world in which the objects have a sense of spatial presence."[1] (tłum. Rzeczywistość wirtualna jest użyciem technologii komputerowej do tworzenia efektu interaktywnego, trójwymiarowego świata, w którym znajdujące się obiekty sprawiają wrażenie przestrzennych). Opracował on również w pełni działający sprzęt składający się z gogli i rękawicy. W kolejnych latach testowano różne rozwiązania, jednakże kolejne porażki spowodowały wycofanie się wielkich firm z rynku VR. Do łask wróciła w roku 2012, kiedy to Palmer Luckey wykorzystał swoje umiejętności w celu wykonania nowej generacji gogli – Oculus. W krótkim czasie od realizacji prototypu i wejściu na portal kickstarter otrzymał 2,5 mln, czyli niemal o 1000% więcej niż zakładał. Później w krótkim czasie został multimilionerem, gdy Facebook wykupił firmę za 2,5 mld dolarów. Jak pisze Paweł Olszewski „każda większa firma chce uszczknąć trochę z nowego, obiecującego sektora" [2]. Konkurencja zatem nie spała – Sony wkrótce ogłosiło Project Morpheus, który miał być goglami współpracującymi z Playstation 4, a Valve nawiązało współpracę z HTC nad systemem Vive, który w danym momencie posiadał najbardziej rozbudowany sprzęt. W kolejnych latach, kolejne nowatorskie systemy ogłaszały swoje projekty, łącznie z tym, iż na kolejne lata są zapowiadane już gogle o rozdzielczości nawet 8 krotnie większej.



Rysunek 1 Projekt i wykonany prototyp Sensoramy (źródło:[2])

Wirtualna rzeczywistość, która jest dziś dostępna publicznie i powoli trafia pod strzechy, stanowi relatywnie nowy rynek. W związku z powyższym powstała nisza, której deweloperzy nie zdołali jeszcze zapełnić. Daje to wiele możliwości nowym studiom i twórcom nie tylko gier komputerowych, ale również oprogramowania użytkowego zaprojektowanego specjalnie na użytek firm, czy nawet w celach medycznych i wojskowych. Szerokie zastosowanie technologii dodatkowo rozpowszechniło sprzedaż, dzięki czemu zostało sprzedanych już ponad 1,5 miliona urządzeń stacjonarnych (rysunek 2), tj. HTC Vive, Oculus, czy Playstation VR i nawet sześć milionów urządzeń mobilnych (dane według raportu wykonanego przez studio The Farm 51). Użytkownicy poszukują multimedialnych rozrywek z użyciem technologii VR.



Rysunek 2 Prognozowana ilość urządzeń VR na rynku w latach 2016-2018 (źródło: [3])

Aby wyjść im naprzeciw Oculus sponsoruje i zleca projekty zewnętrznym firmom, takim jak Epic Games, w celu zrealizowania gier, które sprostają oczekiwaniom stawianym przez graczy. W ten sposób powstały gry *Wilson's Heart* czy *Robo Recall*. Wyróżniają się na rynku nie tylko ciekawą oprawą graficzną, ale również interesującym sposobem prowadzenia samej

rozgrywki. Pierwsza z wymienionych to psychologiczny horror, który został umiejscowiony w latach czterdziestych ubiegłego wieku. Gracz wciela się w rolę tytułowego Wilsona, który musi odkryć jak i dlaczego znalazł się w szpitalu oraz jak z niego uciec. Gra posiada interesujący sposób lokomocji. Stara się również w pełni wykorzystywać możliwości technologii, stawiając gracza w sytuacjach, które umożliwiają wykorzystanie kontrolerów ruchowych w interesujący sposób. Jest to gra, w której mogą odnaleźć się nawet osoby nie mające styczności z grami – czego przykładem może być Conan O'Brien – gospodarz programu o tym samym tytule grający w grę Wilson's Heart (rysunek 3) [4].

Rysunek 3 Gracz wcielający się w postać Wilsona w grze "Wilson's Heart" (źródło: [4])

Na chwilę obecną, oprócz kilku wyjątkowych produkcji, wiele skupia się i oferuje podobne rozwiązania – znajdując się w jednym miejscu, strzelać do nacierających przeciwników korzystając z jednej broni. Umożliwienie graczowi korzystania z kontrolerów ruchowych spowodowało, że urządzenie jest wprost przygotowane do tego, by pozwolić graczowi wcielenie się w obrońcę pewnego obszaru.

1.2. Gra

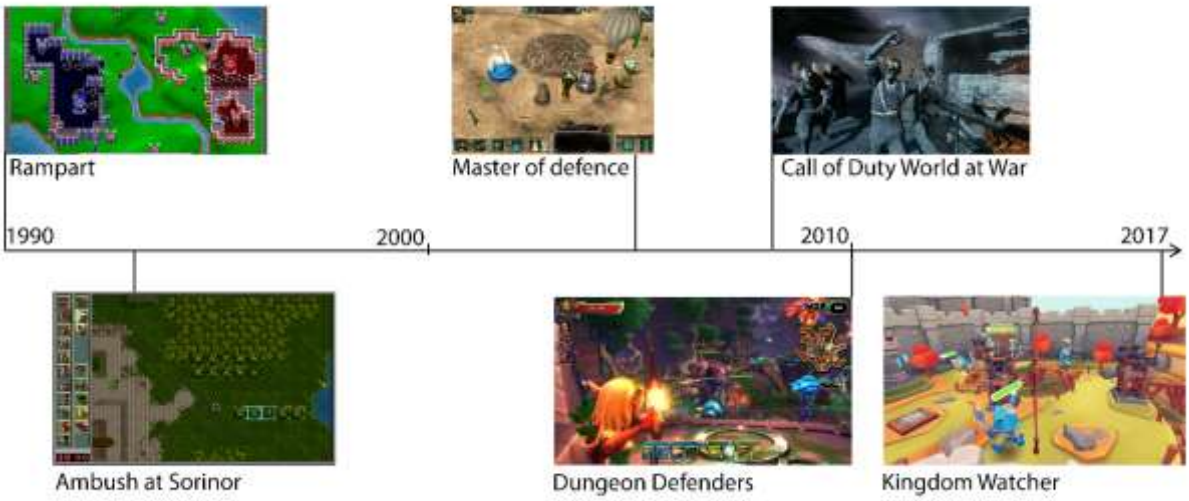
Zgodnie z definicją słownika języka polskiego PWN, gra to: „zabawa towarzyska prowadzona według pewnych zasad" [7], „przedmioty służące do grania" [7] oraz „zmienność postrzeganych wzrokiem szczegółów tworzących harmonijną całość" [7]. Z wielu innych definicji te najbardziej odwzorowują istotę, czym jest projekt będący grą w wirtualnej rzeczywistości. Nie jest to wystarczająca definicja, gdyż program posiada znamiona gry komputerowej, tj. - „gra rozgrywana na ekranie komputera; też: program komputerowy umożliwiający tę grę" [8]. Już w tak krótkiej i niewiele wnoszącej definicji, istnieje widoczny zarys tego, czym jest ten projekt. Podział gier, odbywa się nie tylko na płaszczyźnie elektronicznej, ale również w kontekście docelowej platformy. Definicja, która w przypadku tej pracy pasuje najlepiej, to ta opisująca rzeczywistość wirtualną „sztuczna, trójwymiarowa przestrzeń, w której zaczyna funkcjonować grający w grę komputerową". Jest to definicja zbliżona do tej podanej przez Jarona Laniera. Jej brzmienie jest jednak zbyt ogólne. Już sama gra komputerowa jest interaktywnym programem o bardzo zróżnicowanych możliwościach, czy przeznaczeniu. Korzysta z najnowszych możliwości współczesnych komputerów i nowinek technicznych, w celu jak najciekawszej prezencji. Pierwsze gry, z perspektywy

czasu, nie są już niczym interesującym, a ich implementacja przy użyciu odpowiednich narzędzi, nie zajęłaby więcej niż godziny. Takie odbicie piłeczki, żeby gracz nie zdołał zareagować i stracił punkt (zasady znane z tenisa stołowego), czyli „Tennis for Two" – będący prekursorem gier komputerowych, a rozgrywany na bazie oscyloskopu. Jak widać, ewolucja gier się nie zatrzymuje. Wczorajszy człowiek grał na automatach w „Pong", a dziś zakładamy gogle rzeczywistości wirtualnej, by cieszyć się pełnią immersyjnej rozrywki.

Przyczyna wybrania takiego tematu, jako pracy inżynierskiej jest prozaicznie prosta. Nieodkryty jeszcze potencjał wirtualnej rzeczywistości i jego wykorzystanie w dziedzinie rozrywki. Jest to również duże wyzwanie, gdyż istniejące projekty gier należą do jednych z najbardziej złożonych projektów informatycznych wykorzystywanych każdego dnia. Ich rozwój przyczynia się również do rozwoju innych dziedzin komputerowych, takich jak grafika, sztuczna inteligencja, sieci internetowe, jak również tworzone są nowe algorytmy. Proces wykonania gry nie jest prosty. Jej wykonanie wymaga często zespołów składających się z setek osób. Jest to proces kreatywny, a wkład mają nie tylko programiści, ale również graficy, projektanci, scenarzyści, czy muzycy. Skończone produkcje nierzadko posiadają oprócz interesującej oprawy graficznej, czy wciągających wątków fabularnych - tematy trudne dla społeczeństwa, dzięki czemu zmuszają do głębokiej refleksji. Jest to jedna z niewielu dziedzin, która łączy w sobie tak zróżnicowane dziedziny – od matematyki, przez informatykę, na sztuce kończąc. Rosnące grono odbiorców przyczynia się do coraz większego wzrostu rynku rozrywki i wciąga kolejne pokolenia do kreowanych wirtualnych światów. Już teraz świat zaczyna łaskawiej spoglądać na gry, organizując turnieje e-sportu, opierające się wyłącznie na rozrywkach w gry. Wirtualna rzeczywistość również może zaistnieć na tej arenie – już teraz odbywają się turnieje e-sportu w VR.

1.3. Gra typu „Wave defence"

Projekt, którego dotyczy niniejsza praca należy do gatunku Wave defense. W ogólnej definicji gatunek ten wyewoluował z gatunku Tower Defense, będącym podgatunkiem gier strategicznych czasu rzeczywistego, gdzie celem gracza jest obrona wyznaczonych terenów przez uniemożliwienie dokonania ataku. Do tego celu użytkownik może wykorzystać różne przeszkody i oręże. Pierwsze gry tego gatunku powstawały już w latach dziewięćdziesiątych ubiegłego wieku, kiedy Atari wydało grę Ramparts. Powyższy gatunek pojawiał się i znikał w ciągu trzech dekad swojego istnienia, jednakże rozkwit nastał dopiero w roku 2007, głównie za sprawą wzrostu znaczenia rynku mobilnego. Twórcy gier tego typu wprowadzali różne mniej, lub bardziej innowacyjne zmiany do swoich produkcji, czyniąc je wyjątkowymi na tle poprzedników. Kluczowym dla historii gier stał się moment, kiedy elementy gatunku wyciągnięto poza strategie. Twórcy zauważyli, że w ten sposób można osiągnąć dłuższy czas rozgrywki, mniejszym nakładem pracy i kosztów. Podejście do tematu było zróżnicowane. Jedni wykorzystali wyłącznie elementy, w celu urozmaicenia trwania, czy też sztucznego wydłużenia, kampanii, inni postawili na wykonanie gier skupiających się wyłącznie na aspekcie obrony przed kolejnymi falami przeciwników. Te drugie dodatkowo można podzielić na dwie osobne kategorie – obrona obiektu, lub postaci, w którą wciela się gracz. Na rysunku 6 zostały przedstawione jedne z najważniejszych produkcji gatunku.



Rysunek 4 Historia gatunku i jego ewolucja od lat dziewięćdziesiątych do dnia dzisiejszego

Na uwagę zasługuje przede wszystkim projekt Call of Duty: World at War, wykonany przez studio Treyarch w roku 2008. Jako jedni z pierwszych zrealizowali tryb gry, który w całości opierał się na odpieraniu kolejnych fal przeciwników w grze typu FPP (pierwszoosobowa strzelanka). Wydana dwa lata później gra Dungeon Defenders charakteryzowała się zmianą podejścia do tematu Tower Defense oferując trzecioosobową grę akcji z możliwością budowania wież. Lata 2016 i 2017 przyniosły nową świeżość w postaci wirtualnej rzeczywistości, dzięki czemu odczucia graczy tego trzydziestoletniego gatunku całkowicie się zmieniły.

1.4. Projekt Medieval Archer

Opisany w poniższej pracy projekt – Medieval Archer – skłania się do klasyki gier typu Wave Defense i nie będzie wyznaczać nowych standardów gatunku, a jedynie rozwijać istniejące rozwiązania. Jest to podyktowane w dużej mierze ograniczeniami w postaci czasu i zasobów ludzkich. Gry w dzisiejszych czasach wykonywane są przez grupy od kilku do kilkuset osób przez okres kilku lat. Zatem aby wykonać kompletny projekt w tak krótkim czasie, został on ograniczony w stosunku do początkowej idei. Dzięki temu, warstwa rozgrywki mogła zostać dopracowana, by zapewnić jak najlepsze doświadczenia w wirtualnej rzeczywistości. Pozwoliło to również na dopracowanie graficzne poziomu wraz z udźwiękowieniem.

Głównym założeniem projektu jest sprawdzenie zręczności graczy poprzez strzelanie do nadchodzących fal przeciwników. Kilka ścieżek, z których mogą nadchodzić przeciwnicy i ich losowe pojawianie się, powoduje, że każda rozgrywka jest na swój sposób unikalna. Rosnący poziom trudności wynikający z coraz większej ilości przeciwników w każdej fali sprawia, iż gracz musi coraz celniej strzelać, w przeciwnym wypadku nie będzie w stanie obronić swojego zamku. Aby ułatwić graczowi walkę z przeciwnikiem, na mapie zostały rozmieszczone pułapki, które po uruchomieniu zadają bardzo duże obrażenia przeciwnikom.

Rysunek 5 Zrzut ekranu z gry Medieval Archer

Rysunek 5 pokazuje projekt w trakcie działania. Warto zwrócić na zauważalne sylwetki nacierających przeciwników oraz rozbudowaną warstwę oprawy graficznej. Ponadto gra stawia na większą immersję rezygnując z interfejsu użytkownika, a wszystkie potrzebne reakcje są przedstawiane za pomocą efektów audio-wizualnych.

2. Projekt

2.1. Założenia ogólne

U podstaw założeń realizowanego tematu było wykonanie prostej gry, która będzie mogła być w łatwy sposób modyfikowana w przyszłości. Uzyskana prostota nie jest wynikiem zmniejszenia ilości elementów składowych, a ich odpowiedniej alokacji w projekcie. Przejrzysty kod, enkapsulacja, czy minimalizacja skomplikowanych relacji między wykonanymi fragmentami pozwala na większą czytelność w przyszłości. Umiejętność łatwej modyfikacji w dużej mierze ma pomóc w tworzeniu dodatkowych poziomów. Zastosowanie wszelkich starań do odpowiedniego projektowania gry umożliwi nie tylko proste tworzenie nowych poziomów, ale również korzystanie z innych wizualnie broni, czy też dodanie dodatkowych przeciwników i dynamicznych elementów, które mogą ich zranić.

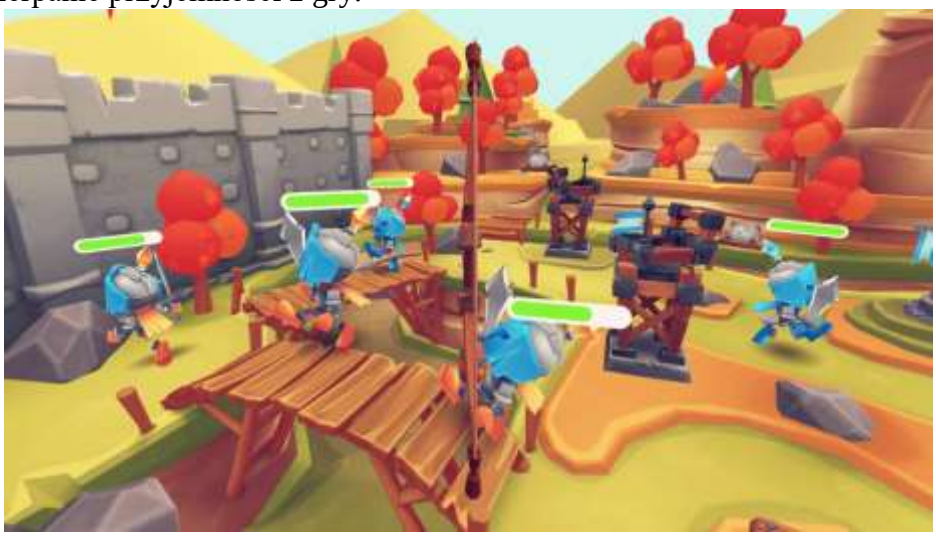
Przed rozpoczęciem projektowania zostały spisane kluczowe idee, które były precyzowane jeszcze przed tworzeniem bazowego prototypu. Wyznaczały one kolejne etapy sprawdzania projektu, dzięki czemu ułatwiło to śledzenie postępu w kolejnych fragmentach projektu . Wyżej wymienione wymagania można zestawić w następującą listę:

- Oddzielenie projektu od jakichkolwiek danych wejściowych. Rozwijając myśl – program może korzystać wyłącznie z plików znajdujących się wewnątrz projektu. Pozwoli to na uniknięcie problemów z referencjami do zmieniających się plików.
- Zmienne parametry dotyczące projektu, powinny być możliwie łatwo dostępne, a jeśli istnieje taka możliwość, zebrane w jednym miejscu.
- Tworzone stałe powinny mieć uzasadnienie w projekcie i być odpowiednio nazwane.
- Każdy moduł ma za zadanie przeprowadzać tylko i wyłącznie odpowiednie akcje i zadania, do których został napisany.
- Niedopuszczalne jest umożliwienie użytkownikowi modyfikacji w zakresie działania gry.
- Deklarowane nazwy zmiennych, funkcji i innych elementów składowych wyrażane są w języku angielskim. Ma to być konsekwentnie przestrzegane w projekcie.
- Czytelność kodu ponad jego wydajność.

Powyższe zasady stanowią próbę systematyzacji kodu i zminimalizowania ilości popełnianych błędów w trakcie jego realizacji, jak również ułatwienie w dodawaniu nowej zawartości do istniejącego już projektu. Kompetentne korzystanie z wyżej wymienionych przepisów powinno umożliwić ukończenie projektu w określonym czasie. Jedną z najważniejszych zasad jest ta wymieniona na samym końcu – w przeciwnym wypadku, gdy nastąpi potrzeba poprawienia istniejącego kodu, może zaistnieć sytuacja, w której ponowne zrozumienie kodu będzie równie czasochłonne, co jego pierwotne napisanie. Dopóki wydajność projektu pozwala na płynną i satysfakcjonującą grę na sprzęcie obsługujących wirtualną rzeczywistość, dopóty wydajność będzie sprawą drugorzędną. Dlatego też w sytuacjach, gdy będzie istniał wybór między wydajniejszym, a czytelniejszym kodem, zostanie zastosowany czytelniejszy, tak długo, jak nie będzie to wpływało bardzo znacząco na wydajność oprogramowania.

2.2. Podobne pozycje na rynku

Na rynku znajdują się już gotowe produkty, które pozwalają strzelać z łuku. Jednym z nich jest Kingdom Watcher (rysunek 6). Jak nazwa wskazuje, gracz jest strażnikiem królestwa, którego zadaniem jest obrona państwa przed stylizowanymi, zarówno w grafice i proporcjach, żołnierzami. Produkcja nie jest rozbudowana. Gracz może budować, w kilku wybranych przez twórców, miejscach wieżę oraz strzelać z łuku. Plusem są zróżnicowane poziomy, będące czymś naturalnym, gdy tworzy się grę opartą o rozgrywkę znaną z Tower Defense. Jak piszą twórcy, „Kingdom Watcher jest idealną mieszanką tower defense i shoot'em up" [5]. Nie rekompensuje to jednak braku immersyjnego korzystania z łuku, co utrudnia czerpanie przyjemności z gry.



Rysunek 6 Widok z gry Kingdom Watcher (źródło: [5])

Kolejną ciekawą propozycją jest Elven Assassin VR (rysunek 7). Od poprzedniej wyróżnia go zdecydowanie inny styl graficzny – postawili na realizm średniowiecznej fantastyki. Powstająca od ponad roku gra w Gliwicach. W recenzji Petera Grahama, zauważył on, iż „strzelanie w wirtualnej rzeczywistości jest zawsze przyjemne, nieważne czy to bronie, magia, czy łuk i strzały" [6]. Nic zatem dziwnego, że przychylnie spojrzał na ten tytuł. Gra umożliwia teleportację w kilka wybranych miejsc. Podobnie jak w przypadku Kingdom Watcher, gra również skupia się na celnym strzelaniu z łuku. Gra wymaga większej spostrzegawczości względem cukierkowej gry. Powodów jest kilka – postaci nadchodzą z różnych stron, a kolorystycznie potrafią zlać się z tłem. Efekt ten jest potęgowany tym bardziej ze względu na stosunkowo niską rozdzielczość gogli wirtualnej rzeczywistości. Potrzeba strzelania na duże odległości do szybko poruszających postaci jest dodatkowo utrudniona przez brak nawet najprostszych przyrządów celowniczych i problemy z percepcją, znane przy korzystaniu z VR.

Rysunek 7 Widok z gry Elven Assassin VR (źródło: [6])

Ostatnia ze sprawdzonych produkcji – Holopoint (rysunek 8) – jest grą zdecydowanie bardziej dynamiczną niż te przedstawione dotychczas. Poza interesującym stylem graficznym, wyróżnia się na tle pozostałych również sposobem rozgrywki. Zadaniem nie jest obrona obiektu przed nadciągającymi przeciwnikami, a skupia się na atakowaniu przeciwników: unoszących się w powietrzu sześcianów, czy chodzących, holograficznych przeciwników, nim te zaatakują nas. Przeciwnicy mają dodatkowo możliwość strzelania, co powoduje, że gra wymaga większego poruszania się po ograniczonej przestrzeni pokoju. Daje to jednak pewną satysfakcję, gdy unikniemy kolejnego pocisku, a sami poślemy celną strzałę. Rozgrywka zatem sprowadza się do tworzenia wyimaginowanej choreografii w celu unikania kolejnych pocisków i jednoczesnego atakowania nacierających przeciwników.

Rysunek 8 Projekt Holopoint w akcji (źródło: [10])

Po sprawdzeniu powyższych, istniejących już projektów, zostało zanotowane, jakie aspekty można poprawić, na czym należy skupić większą uwagę i z takimi informacjami wzięto się za podjęcie wykonania opisywanego projektu.

2.3. Elementy gry

Przed rozpoczęciem właściwej części opisu projektu gry i jej implementacji, istotne jest zrozumienie najważniejszych elementów związanych z grą Medieval Archer, które będą się pojawiać wielokrotnie w kolejnych fragmentach poniższej pracy.

Gracz: Obiekt posiadający wymagane funkcjonalności. Dzięki dziedziczeniu po zaimplementowanej w silniku klasie postaci, dzięki czemu w przyszłości będzie możliwość zaprogramowania dodatkowych sposobów poruszania się. Do gracza przypisane są takie komponenty jak kontrolery, łuk, kamera i komponent kapsuły stanowiący fizyczną reprezentację gracza (odpowiada za sprawdzanie czy gracz znajduje się na powierzchni stałej, czy w powietrzu).

Kontroler: Komponent reprezentowany jako dłoń. W związku z potrzebnymi funkcjonalnościami i reprezentacją graficzną, jest rozróżniany jako: lewy i prawy. Posiada interfejsy odpowiedzialne za łapanie i puszczanie obiektów. Składa się z obiektów posiadających szkielet - reprezentowany przez hierarchie kości. Jest to wymagane w frameworku Unreal Engine dla przedstawiania animacji.

Łuk: Obiekt posiadający reprezentację graficzną broni dwuręcznej dalekosiężnej w postaci średniowiecznego łuku. Korzysta z interfejsów pochodzące z kontrolerów. Wykorzystuje je do implementacji funkcjonalności związanej z łapaniem, strzelaniem z łuku i jego odkładaniem. Dodatkowo posiada funkcje wywoływania wibracji w kontrolerze po oddaniu strzału i tworzy dodatkowe strzały, gdy gracz zniszczy balon.

Strzała: Komponent posiadający logikę ranienia przeciwnika. Jest reprezentowany przez średniowieczną strzałę, z magicznym zakończeniem w celu ułatwienia jej widoczności po wystrzeleniu. Na grocie jest dodany komponent fizyczny (w kształcie sfery) sprawdzający kolizję z innymi obiektami znajdującymi się na scenie.

Rysunek 9 Graficzna wizualizacja strzały

Cel: Obiekt ustawiany statycznie każdemu kontrolerowi sztucznej inteligencji w celu określenia miejsca, do którego ma podążać i co ma zniszczyć.

Przeciwnik: Obiekt posiadające funkcjonalności związane z implementacją poruszania się, jak również zadawania obrażeń (w stosunku do celu) oraz otrzymywania obrażeń (od strzał i pułapek). Jest sterowany przez kontroler. Zawiera drzewo behawioralne sterujące sztuczną inteligencją. Posiada reprezentację graficzną w postaci zróżnicowanych rycerzy posiadających broń oraz opcjonalnie - tarczę.

Pułapka: Aktor posiadający fizyczną reprezentację w postaci obiektu z możliwością jego destrukcji (Destructible). Zawiera logikę ranienia przeciwników znajdujących się w zadanym obrębie. Ulega zniszczeniu po kolizji ze strzałą gracza.

Rozgrywka: Główna pętla gry, na którą składa się: aktywowanie przeciwników w zwiększającej się liczbie z każdą falą. Oczekiwanie na zniszczenie celu, bądź wszystkich przeciwników i ponowne aktywowanie rycerzy.

Poziom gry: Główna scena, na której rozgrywa się cała gra. Składa się z elementów statycznych i dynamicznych. Posiada też część logiki odpowiedzialną za rozgrywkę na konkretnej instancji mapy. Znajdują się tutaj również pułapki, spawnery, dodatkowe wieże, cel przeciwników, potrzebne informacje dotyczące rozgrywki, w tym statystyki rozgrywki i stan bramy.

Spawner: Obiekt służący za miejsce, w którym mogą być aktywowani przeciwnicy. Na poziomie gry musi znajdować się przynajmniej kilka jego instancji.

Boss: Aktor dziedziczący po Przeciwniku, który posiada zwiększone życie i zadaje większe obrażenia celowi.

Balon: Obiekt pojawiający się po zestrzeleniu przeciwnika. Porusza się z dużą siłą ku górze. Jego zestrzelenie powoduje dodanie odpowiedniej liczby punktów do bramy i aktywowanie potrójnych strzał w łuku.

Wieża: Obiekt reprezentowany przez kamienną wieżę ze znajdującą się na niej czerwoną flagę. Strzelenie w kierunku flagi powoduje zmianę pozycji gracza pomiędzy wieżami.

Rysunek 10 Wieża umożliwiająca zmianę pozycji strzelca - gracza

2.4. Założenia funkcjonalne

Tester sprawdzający wykonane oprogramowanie, a docelowo użytkownik – gracz, największą uwagę zwróci na wymagania funkcjonalne, gdyż to od nich zależy końcowa funkcjonalność projektu i jego możliwości. Istotne jest wykonane wszystkich założonych celów, w przeciwnym wypadku, gracz może poczuć się oszukany ze względu na otrzymanie niekompletnego produktu. W przypadku „Medieval Archer" część wymagań funkcjonalnych została zaprojektowana już w trakcie zaawansowanego tworzenia gry – było to spowodowane metodyką prowadzenia prac. Mimo takiego sposobu kierowania czasem w projekcie – po poprawkach i przemyśleniach, została wykonana lista założeń zawierająca najważniejsze punkty dotyczące opisywanej pracy. Została ona przedstawiona poniżej.

- Gracz dysponuje możliwością strzelania z łuku,
- Gracz dysponuje możliwością celowania przy pomocy holograficznego celownika,
- Siła naciągu cięciwy mająca znaczenie na tor lotu strzały,
- Gracz może uzyskać bonusowy strzał składający się z trzech strzał,
- Gracz może przemieszczać się między istniejącymi wieżami,
- Płynnie poruszający przeciwnicy stanowiący wyzwanie,
- Dwa bazowe rodzaje przeciwników posiadający różną ilość punktów życia i ilość zadawanych obrażeń,
- Możliwość sprawdzania na bieżąco stanu bramy,
- Gracz w dowolnym momencie może sprawdzić, ile fal przetrwał i ile zdobył punktów,
- Gracz otrzymuje widoczną wiadomość po przegraniu rozgrywki.

2.5. Metodyka projektu

Wymyślona koncepcja wraz ze spisanymi założeniami nie są wystarczające, by z powodzeniem ukończyć projekt. Przed rozpoczęciem implementacji, należy utworzyć projekt aplikacji, a z pewnością jego szkic, tak aby umożliwić osobom pracującym nad projektem na podążanie za wspólną ideą. Dodatkowo, w przypadku opisywanego dokumentu, przeprowadzono rozeznanie w zakresie metodyk prowadzenia projektu, w celu zapewnienia możliwie najlepszych efektów, przy stosunkowo niskich nakładach pracy. Zauważono, iż nie istnieje jedno, uniwersalne rozwiązanie, które pozwalałoby na rozwiązanie wszystkich problemów. Jest to uwarunkowane między innymi tym, kim jest użytkownik docelowy, czy też jaki zestaw wymagań został postawiony u początku projektowania.

W przypadku opisywanego tutaj projektu, klientem jest gracz. W związku z powyższym, naturalne wydało się stosowanie metodyki zwinnej. Programowanie zwinne w dużej mierze skupia się na przyrostowym modelu budowania aplikacji i zakłada często wydawane wersje prototypowe, które przechodzą przez testy z udziałem klienta. W projekcie nie było ustalonych sztywnych ram czasowych, co pozwalało na częstsze wprowadzanie zmian i poprawek do wcześniej utworzonych elementów. W wyniku czego, po kilku – tak zwanych przebiegach (z angielskiego sprints) – spisana zostałą pierwotna wersja projektu, na podstawie którego rozpoczęto pracę nad implementacją. Skorzystanie z metodyk agile i częste testowanie aplikacji pozwoliła na spełnienie dodatkowych wymagań, które powstały dopiero w trakcie dogłębnego testowania. Pozwoliło to na przygotowanie projektu pod grupę

docelową, a ponadplanowo również zanotowano pewne powtarzające się scenariusze, czy pomysły na to, jak rozbudować projekt w przyszłości.

2.6. Rozgrywka

Rozgrywka będzie stanowiła swojego rodzaju trybut do czasów średniowiecznych i zarazem gier akcji. Jest to również odpowiedź na pytanie, dlaczego akurat łuk. Umiejętności gracza są stylizowane na tych pochodzących z tego okresu czasowego, z odrobiną magii, w postaci magicznych strzał pojawiających się, gdy zdobędziemy odpowiedni bonus. Zaimplementowany tor lotu strzały stanowi symulację sposobu poruszania strzał w świecie rzeczywistym, z delikatnym podejściem w stronę rozrywkowego podejścia ilością siły, aby gracz nie był zmuszony do trudnego naciągu cięciwy.

Znaczącym problemem wielu gier w wirtualnej rzeczywistości jest wielkość map. Nierzadko duże mapy, z zawiłymi korytarzami i wciąż pojawiający się zagadnienie lokomocji w VR. Jest to szczególnie problematyczne w sytuacji, gdy do znalezienia mamy mały element w otaczającym nas świecie. Medieval Archer eliminuje te problemy poprzez wykonanie niewielkiej areny z teleportacją wyłącznie do określonych miejsc, co eliminuje sytuacje, gdy poprzez teleportację dostaniemy się do miejsca, którego nie potrafimy opuścić. Niewielkie obszary to również dobry sposób na lepsze kierowanie rozgrywką poprzez doprecyzowanie, skąd, gdzie i jaką trasą będą poruszać się postaci widoczne przez gracza.

Miejscem życia bohatera jest mityczny, średniowieczny świat, który został zalany przez siły złego władcy. Różne kultury starają się przejąć kontrolę nad europejskim życiem, tworząc zróżnicowanych przeciwników, których napotka na swojej drodze gracz. Wpływ zła może jednak zostać opóźniony i wyłącznie efektywność gracza może tutaj coś zmienić.

Początek każdej rozgrywki to możliwość przetestowania łuku w formie intuicyjnego samouczka. Ma to również za zadanie wprowadzić gracza do świata, w którym się znalazł. Zrozumie swoje możliwości i nauczy się je wykorzystywać, żeby móc w pełni cieszyć się rozgrywką.

2.7. Mechanika

Opisywany projekt korzysta z jednej z najbardziej pożądanych zasad przy tworzeniu gier: Prosta do zagrania, trudna do opanowania do perfekcji. Pozwala to na ciągłe rozwijanie gracza w istniejących mechanikach, czy uczenie i poznawanie nowych.

Najistotniejszą mechaniką w przypadku tego projektu jest strzelanie z łuku. Zagadnienie to było konsultowane z łucznikiem, w celu zwiększenia immersji trzymania i strzelania z łuku, dzięki temu, zaprojektowana broń miała szanse dorównywać prawdziwej. Drugą rdzenną mechaniką jest wysyłanie coraz większej liczbie przeciwników wraz z progresją w grze. Dostosowanie odpowiednich parametrów, takich jak początkowa liczba, czy liczba, o jaką ma się zwiększać ilość aktywowanych przeciwników. Powoduje to stopniowe zwiększanie poziomu trudności. Za efektywną walkę, gracz jest nagradzany punktami. Ich ilość jest zależna od tego, do której fali gracz doszedł i czy potrafi celnie trafiać w głowę, co jest nagradzane dodatkowymi punktami. Pojawiające się po pokonaniu przeciwnika balony, jeżeli zostaną zestrzelone, również nagradzają gracza punktami, jak również punktami wytrzymałości dla bramy.

Dzięki odpowiedniemu połączeniu interesującego świata gry i mechanik gier, projekt jest w stanie utrzymać gracza w stanie „flow” [11] (rysunek 11), czyli takiego zaangażowania, że czerpiemy przyjemność z gry i nie jesteśmy znudzeni jej prostotą, po krótkim czasie od zapoznania się z produkcją.

Rysunek 11 Wykres pokazujący w jaki sposób powinna zmieniać się trudność umiejętności (źródło:[11])

2.8. Styl graficzny

Spójność jest kluczem do immersji. Jest to zatem jedna z decyzji, które znacząco wpływają na odbiór produktu przez klienta. W przypadku wykonywanego projektu – wybór był tym istotniejszy, gdyż projekt posiada narzucone duże wymagania sprzętowe związane z technologią wirtualnej rzeczywistości. Dlatego też, w wyniku testów wydajnościowych, wybrano grafikę stylizowaną „low-poly" w średniowiecznym świecie. Użyty sposób przedstawienia, pozwala na znaczną optymalizację projektu i odchudzeniu go z wielu nieistotnych elementów, przy równoczesnym zachowaniu przyjemnej dla oka otoczki graficznej.

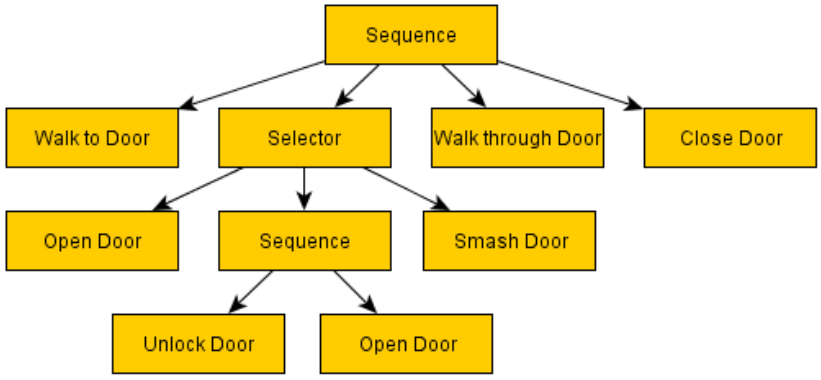

Rysunek 12 Tymczasowy obrazek z wyjaśnieniem grafiki

Użycie stylistyki low-poly wymaga roztropnego wykorzystywania kolorów, aby gracz, nie miał problemu z zauważeniem, co robi i gdzie się znajduje. Konsystencja wykorzystywanych kolorów jest bardzo istotna, zatem należy uważać, by nie użyć tego samego (widocznego dla gracza) koloru, również do innych oznaczeń. Przykładowo, użycie konkretnego koloru

do symbolizowania pewnego typu interakcji spowoduje, że gracz będzie się spodziewał każdorazowo, że wykorzystany kolor będzie służył tylko i wyłącznie do interakcji podobnych i takich samych jak ta pierwsza. Taki sposób uczenia spowodował, że pewne schematy przewijają się we wszystkich grach – przykładem może być powielanie schematu czerwonych beczek, które, co jest bardzo prawdopodobnie, wybuchną po oddaniu w nie strzału. Skorzystanie z takiego stylu graficznego upraszcza sytuację związaną z teksturowaniem obiektów przestrzennych. Zamiast tego, każdy poligon posiada wyłącznie kolor, który jest zawarty w jednej teksturze – bitmapie. Pozwala to dodatkowo zaoszczędzić pamięć i zmniejsza obciążenie procesora.

2.9. Sztuczna inteligencja

Sztuczna inteligencja w grach to duże wyzwanie. Musi nie tylko reagować w czasie rzeczywistości, ale również być zdolna do dynamicznego zmieniania decyzji, gdy sytuacja w trakcie rozgrywki będzie tego wymagać. W przypadku tego projektu, wykorzystano zaimplementowane w silniku drzewa behawioralne.



Rysunek 13 Przykład drzewa behawioralnego (źródło:[13])

Chris Simpson w swoim artykule o drzewach behawioralnych napisał: „Drzewa mogą być bardzo głębokie, z węzłami odwołującymi się do poddrzew, które przeprowadzają konkretne funkcje, pozwalając programiście na tworzenie bibliotek zachowań, które mogą być następnie połączone ze sobą, by uzyskać bardzo przekonujące zachowanie sztucznej inteligencji" [13]. Rozwiązanie wykorzystujące drzewa behawioralne, to bardzo elastyczne rozwiązanie, które pozwala w krótkim czasie przetestować dziesiątki sposobów reakcji, co znacząco przyspiesza proces implementacji przeciwnika. Po utworzeniu prostego drzewa łatwo dodawać dodatkowe węzły, które wprowadzą zróżnicowanie w zachowaniu, a tym samym przyczynią się do ciekawszej rozgrywki.

Kontroler sztucznej inteligencji ma za zadanie w odpowiedni sposób sterować wirtualnymi przeciwnikami. W tym celu będzie posiadał odpowiednią instancję powyższego drzewa behawioralnego (rysunek 13). Dzięki temu, przeciwnik będzie płynne przechodził z punktu, w którym został aktywowany, do punktu, w którym znajduje się brama i rozpocznie jej atak. Następnie, gdy uda mu się ją zniszczyć, przejdzie do środka. Istotne jest, by zaimplementowana sztuczna inteligencja, była wystarczająco generyczna, by znalazła zastosowanie nie tylko na wykonanej w ramach pracy poziomie gry, ale również na każdym innym spełniającym podstawowe warunki, tj. posiadającym miejsce docelowe i wygenerowaną mapę, służącą do wyszukiwania trasy na powierzchni mapy.

3. Implementacja

3.1. Środowisko i użyte narzędzia programistyczne

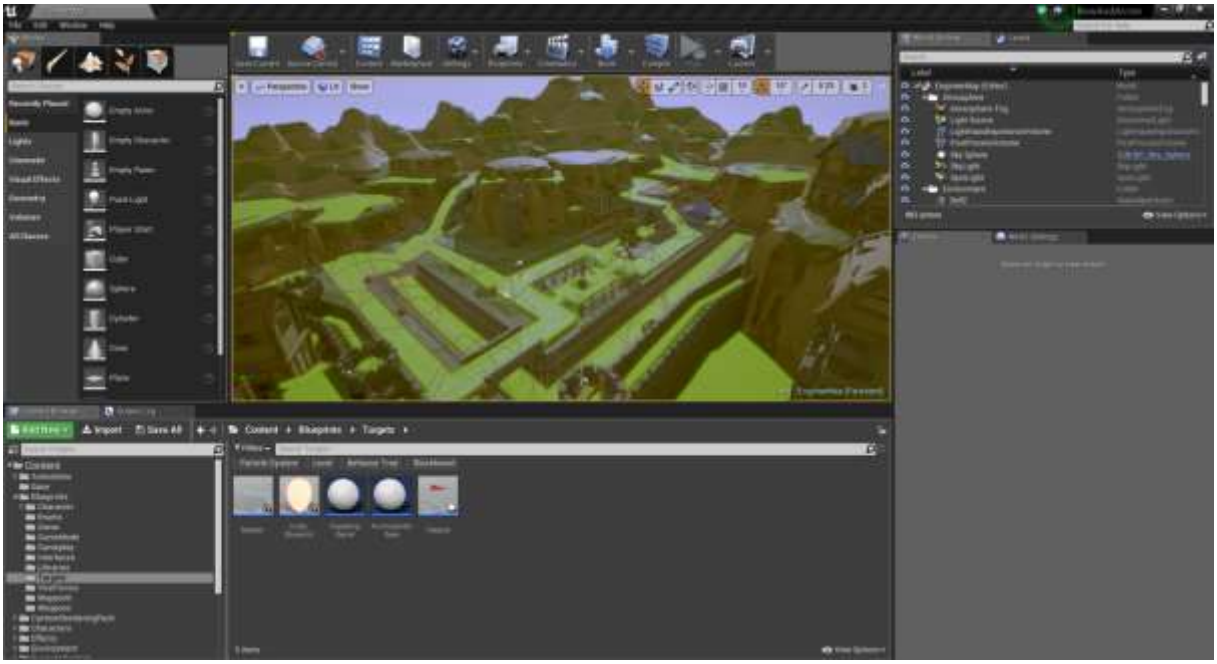
3.1.1. Wprowadzenie

W opozycji do wielu aplikacji użytkowych, gry są wykonane tak, by wciągać użytkowników w świat multimedialny za pomocą rozbudowanych, złożonych scen w dwóch lub trzech wymiarach, korzystając z dobrodziejstw techniki. Aby gracz potrafił wczuć się w otaczający go świat, każdy aspekt projektu musi być spójnie wykonany, w tym: grafika, udźwiękowienie, a co najstotniejsze z punktu widzenia gracza – interesująca i umiejętnie zbudowana rozgrywka. Za zarządzanie całością projektu odpowiada framework - w przypadku tego projektu, jest to Unreal Engine 4. Zestaw narzędzi, jaki oferuje, umożliwia wykonanie opisywanego pomysłu. Silnik został zaprojektowany tak, by umożliwiał nie tylko tworzenie gier, ale również filmów animowanych, czy foto-realistycznych wizualizacji. Tak przygotowany stanowi odpowiedni fundament do wykonywania projektów w wirtualnej rzeczywistości – dla komputerów stacjonarnych, konsoli stacjonarnych, czy urządzeń mobilnych. Firma Epic Games zdaje sobie sprawę, iż rozszerzona i wirtualna rzeczywistość to bardzo dynamicznie rozwijające się technologie. W związku z zainteresowaniem firmy dalszym ich rozwojem, zrozumiałe jest, że już od kilku wersji silnik posiada natywne wsparcie urządzeń, między innymi - HTC Vive. Przetestowali go przy okazji tworzenia własnych produkcji – najpierw Bullet Train, na podstawie którego wykonali prezentację [12], która jest obowiązkową pozycją dla osób interesujących się wirtualną rzeczywistością, następnie Robo Recall.

Jednym z istotnych aspektów wyboru właśnie tego rozwiązania, była możliwość zastosowania Forward Rendereringu, który zmniejsza wymagania dla karty graficznej. Na potrzeby swojego projektu VR rozbudowali możliwości renderujące, tak, by były w stanie obsługiwać dodatkowe opcje dotyczące świateł, MSAA, czy instancjonowane renderowanie – polegające na renderowaniu klatek dla obu oczu jednocześnie, a nie w sekwencji, powoduje to wzrost wydajności zarówno dla karty graficznej, jak i procesora [15] - co pozwala uzyskać satysfakcjonującą jakość wizualną i utrzymać wymagane 90 klatek na sekundę. Silnik domyślnie posiada również optymalizację obiektów graficznych, pozwalając silnikowi renderującemu na odrzucenie pikseli, które nie będą widzialne ze względu na zakrzywienie soczewek gogli [15]. Został on napisany w języku C++ i też z jego wykorzystaniem wytwarza się oprogramowanie. Atutem wyżej przytoczonego silnika, względem innych istniejących propozycji jest natywne narzędzie Blueprints, które pozwala na wizualne programowanie.

3.1.2. Edytor map

Jednym z podstawowych narzędzi silnika Unreal Engine 4, jest edytor map. To przy jego użyciu wykonywane są poziomy, po których porusza się gracz. Równie rozbudowany, co edytor, jest jego interfejs graficzny. W odpowiednich oknach znajdziemy bazowe obiekty, w tym prymitywne obiekty geometryczne, czy efekty graficzne i wolumeny, m.in. poszukujące płaszczyzny, po których mogą się poruszać przeciwnicy. Po uruchomieniu silnika, po prawej stronie znajdują się szczegóły dotyczące wybranego obiektu i lista obiektów rozstawionych na mapie. U dołu umieszczono wykaz obiektów, w tym utworzonych przez siebie aktorów, jak również pliki z materiałami i teksturami. W centralnej części można zobaczyć widok z otworzonej mapy, z możliwością swobodnego poruszania. Opisywane okno może również posłużyć do wyświetlanie gry, jeśli zostanie uruchomiona odpowiednia opcja.



Rysunek 14 Interfejs edytora map w Unreal Engine 4

3.1.3. Blueprints – wizualny język skryptowy

System wizualnego skryptowania – Blueprints, jest kompletnym narzędziem pozwalającym na wykonanie całego projektu bez użycia natywnego C++. Koncept ten jest oparty o interfejs korzystający z węzłów, służących do pisania elementów rozgrywki wewnątrz silnika. Podobnie jak inne języki wizualne, używa on do definiowania klas OO (object-oriented) lub obiektów wewnątrz silnika. Zdefiniowane obiekty przy użyciu tego systemu, są nazywane tak samo jak system, czyli Blueprints. Elastyczność i możliwości systemu pozwalają projektantom używać wielu narzędzi, które zwykle dostępne są tylko programistom. Dodatkowo, można go rozszerzyć o funkcje zapisane w natywnym języku c++, co sprawia, że jedynym ograniczeniem są zasoby, do których ma się dostęp. Jest to kolejny ewolucyjny krok w stronę szybkiego prototypowania projektów. Zastąpił on częściowo wcześniejszy język – UnrealScript i posiada też większość jego możliwości, między innymi: dziedziczenie klas, przechowywanie i modyfikowanie własności obiektów, czy zarządzanie komponentami (obiektami, będącymi dziećmi klasy). Docelowo system został tak zaprojektowany, aby umożliwić ścisłą współpracę między programistami, a designerami. Pierwsi mają za zadanie napisać bazowe klasy, z odpowiednimi polami możliwymi do modyfikacji, a drudzy wykorzystać klasy poprzez dziedziczenie.

System składa się z wielu typów, z których każdy posiada inną funkcjonalność – od tworzenia nowych typów, poprzez skryptowanie wydarzeń, które mają mieć miejsce w trakcie rozgrywki, na tworzeniu interfejsów kończąc.

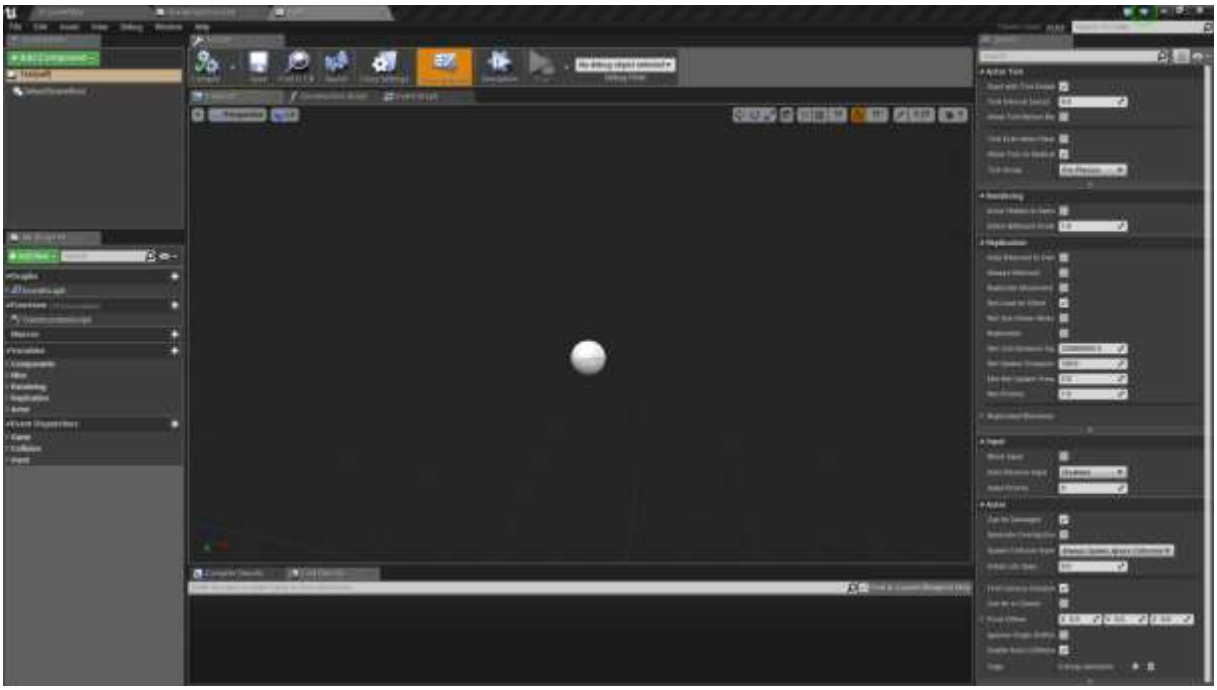
W opisywanym projekcie najczęściej używanym typem jest Blueprint class. Obiekt ten pozwala z łatwością dodać funkcjonalność do istniejących klas. Jak inne – jest tworzony wewnątrz silnika, z wykorzystaniem wizualnego skryptowania, a następnie zapisuje go w projekcie. Służy również do zdefiniowania nowych klas, które mogą zostać umieszczone w postaci instancji na mapie i będą zachowywać się jak każdy inny aktor.

Innym z wykorzystywanych typów jest Level blueprint. Zachowuje się on jak globalny graf poziomu. Każdy poziom posiada własną instancję tego obiektu tworzoną

w momencie kreacji mapy, jednakże nie można utworzyć go w inny sposób. Przechowuje dane dotyczące wydarzeń i instancji aktorów znajdujących się na mapie. Pozwala również na podział mapy i wczytywanie jej w odpowiednich momentach, dla zwiększenia wydajności.

Warto również wspomnieć o strukturze wewnątrz obiektu Blueprint. Najważniejszymi elementami są: Event Graph, Construction Script i Viewport. W zakładce Event Graph można znaleźć węzły grafu, które używają zdarzeń i wywołania różnych funkcji, w celu przeprowadzenia akcji związanych z Blueprintem. Jest wykorzystywane w celu dodania funkcjonalności do wszystkich istniejących instancji klasy. Zgodnie z tym, co napisał Brenden Sewell - „event nawiązuje do pewnych akcji w grze, które zachowują się jak wyzwalacz dla Blueprinta by coś wykonać. Większość tworzonych blueprintów posiada poniższą strukturę: Event (kiedy), warunki (jeśli), akcje (wykonaj).” [14]. Construction Script przechodzi, w momencie utworzenia instancji klasy, po liście komponentów w niej zawartej, aby wykonać odpowiednie operacje inicjalizacyjne. Ostatnią, lecz równie ważną częścią jest Viewport [Rysunek 14] – tutaj jest przedstawiona graficzna reprezentacja tworzonej klasy, gdzie w łatwy sposób można ustawić komponenty w relatywnej pozycji do komponentu będącego korzeniem hierarchii.

Narzędzie posiada również własny debugger. Po wejściu w tryb pracy z jego użyciem, można przyjrzeć się dokładnemu przepływowi sterowania w czasie rzeczywistym i wówczas stosunkowo łatwo wykryć błędy, które zostały popełnione.



Rysunek 15 Interfejs Blueprintu

3.2. Problemy implementacyjne

3.2.1. Wprowadzenie

Proces implementacji składa się z problemów i sposobów ich rozwiązania. Pod tym względem, wykonywanie gry nie różni się niczym od realizacji innych projektów informatycznych. Jak w ich przypadku, tak i tutaj niejednokrotnie należy przeprowadzić długotrwały proces wyszukiwania problemu, a następnie poprawiać tak długo, aż rozwiązanie będzie satysfakcjonujące dla klienta. W przypadku opisywanego projektu, w związku z pewną

losowością, nawet replikacja błędu może być wymagająca. Poza tym nie wszystkie błędy są wyłącznie wynikiem aplikacji, nierzadko wynikają z niewłaściwego działania sprzętu. Gdy takie wystąpią, można jedynie próbować przeciwdziałać istniejącym objawom, gdyż dopóki nie nadejdzie nowa edycja urządzeń – problem będzie występować powszechnie.

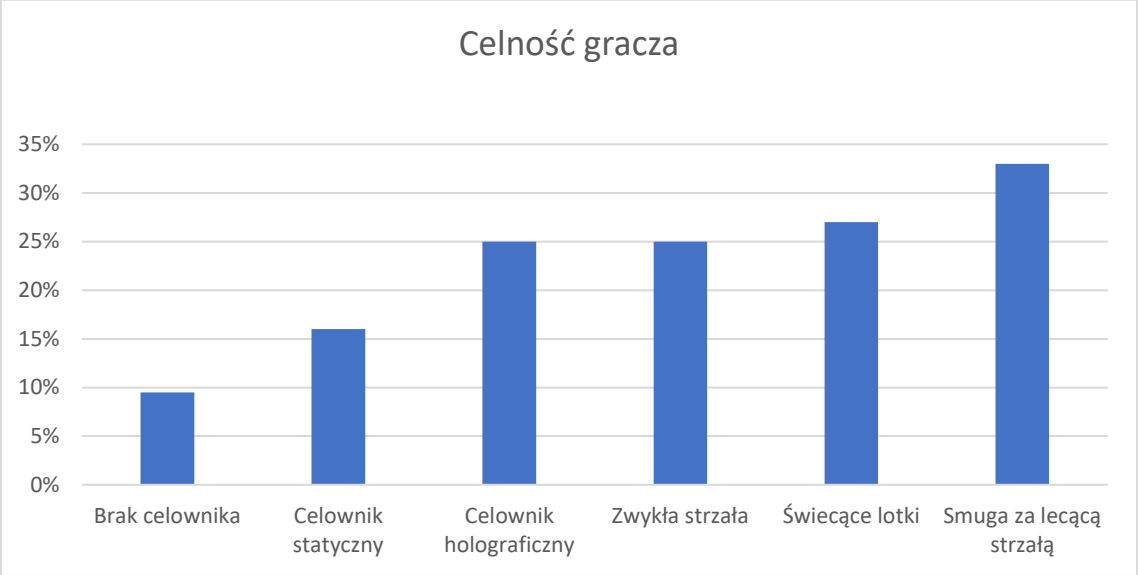
3.2.2. Problemy z celnym strzelaniem

Ze względu na istniejącą niedokładność pozycji kontrolerów, a co ważniejsze – percepcyjne problemy związane z postrzeganiem głębi w wirtualnej rzeczywistości, zadanie gracza związane ze strzelaniem jest utrudnione. Niestety, projekt nie jest w stanie zapobiec występującym problemom, może jedynie poprzez odpowiednie mechaniki ułatwić korzystanie z łuku. W tym celu sprawdzono trzy różne rozwiązania. Test polegał na trafianiu poruszających się celów i obliczaniu stosunku trafień do ilości wszystkich strzałów.

Rysunek 16 Wykorzystane celowniki w projekcie

W pierwszej kolejności przetestowano działanie łuku bez implementacji dodatkowych przyrządów celowniczych. W trakcie testowania okazało się, że problemy ze strzelaniem były na tyle problematyczne, iż cel trafiało mniej niż dziesięć procent wystrzelonych strzał. W związku z powyższym, został dodany statyczny celownik (Rysunek 16, po lewej). Dodanie powyższego nieznacznie ułatwiło strzelanie i poprawiło wynik o 6 punktów procentowych. Dzięki statycznemu celownikowi, łatwiej jest graczowi utrzymać strzałę w odpowiedniej płaszczyźnie. Niestety nie jest to wystarczające, aby uzyskać satysfakcjonujący wynik. Kierując się zasadą – rozgrywka ponad wszystko, został dodany dynamiczny celownik holograficzny, który choć nie pasuje do średniowiecznego stylu, miał za zadanie ułatwić ustawienie strzały w dwóch płaszczyznach, dzięki czemu gracz czuje, że strzała porusza się tam, gdzie ją posłał. Zwiększyło to celność do 25 procent. Kolejnym krokiem była próba zasygnalizowania graczowi, gdzie trafiła wypuszczona strzała. Realistyczne odwzorowanie

połączone z niską rozdzielczością gogli powodowało, iż gracz nie był w stanie zauważyć gdzie poleciała strzała. W związku z powyższym zastosowano kolejne usprawnienia w mechanice strzelania. Pierwszą z nich było dodanie świecących lotek. Miało to na celu ułatwienie graczowi zauważenie post factum miejsca, gdzie wylądowała strzała, tak by mógł skorygować tor lotu następnej. Nieznacznie zwiększyło to celność dalszych strzałów. Kolejnym pomysłem było zaimplementowanie smugi, która ukazuje się za strzałą, dzięki czemu gracz jest w stanie zauważyć cały, paraboliczny tor lotu strzały. Był to dobry pomysł, który dodatkowo zwiększył celność o kilka procent. Poniżej można zauważyć na wykresie (Rysunek 16), kolejne etapy ulepszania mechaniki, w celu zwiększenia celności. Finalnie średnio co trzecia strzała trafia celu, co stanowi wystarczający wynik, by zapewnić graczowi poczucie, że jest dobrym strzelcem.



Rysunek 17 Celność gracza w zależności od typu celownika

3.2.3. Droga przeciwników

Przeciwnicy stanowią ważną część gry. W związku z powyższym istotne jest, aby sposób ich poruszania był możliwie odpowiedni do stylu projektu. Ważne również było, aby spełniony był warunek możliwie dynamicznego sposobu prowadzenia przeciwników. W programie rozważano dwa różne podejścia do tego zagadnienia. Pierwszym z nich było utworzenie odpowiedniej hierarchii punktów – grafu, gdzie postać miała poruszać się od punktu do punktu (rysunek 17).

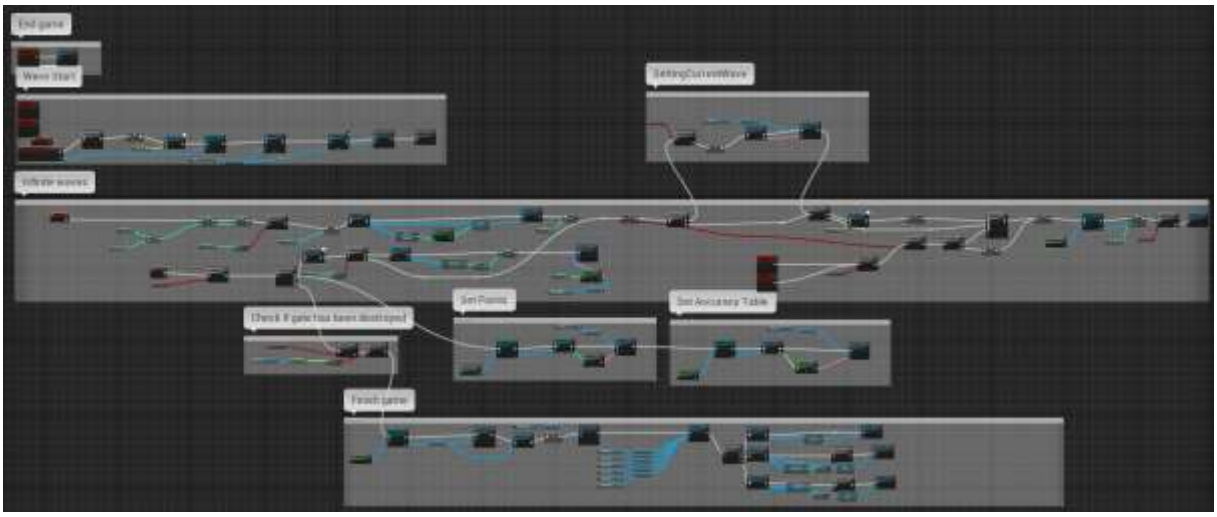
Rysunek 18 Skierowany graf dla trasy przeciwników

Jak widać na rysunku powyżej, utworzenie zestawu punktów (czerwone kropki na rysunku 18), i ustawienie im odpowiedniej kolejności, tak by postaci poruszały się zgodnie z przeznaczeniem wymagało wielu obiektów, które wskazywały, gdzie mają poruszać się po dotarciu do kolejnych węzłów. Jest to czasochłonne i obarczone dużym ryzykiem popełnienia błędu w przypadku zmiany lub dodania kolejnego odcinka, po którym mogą się poruszać przeciwnicy. W związku z powyższym, rozwiązanie to zostało zastąpione innym, prostszym. Każdy przeciwnik otrzymuje jedynie punkt, który ma atakować. Trasę wyznacza sam, korzystając z wbudowanego algorytmu nawigacyjnego. Uprościło to nie tylko implementację, ale również poprawiło płynność poruszania się przeciwników. Sprawiło również, że jest możliwa implementacja mechaniki związanej z odcinaniem drogi – algorytm jej wyszukiwania jest dynamiczny i odnajdzie w czasie rzeczywistym nową, najkrótszą trasą.

3.3. Model i logika gry

3.3.1. Główna pętla aplikacji

Główna pętla całej rozgrywki sprowadza się do poniższego grafu (rysunek 18) odpowiedzialnego za wszystkie, najistotniejsze jej elementy.

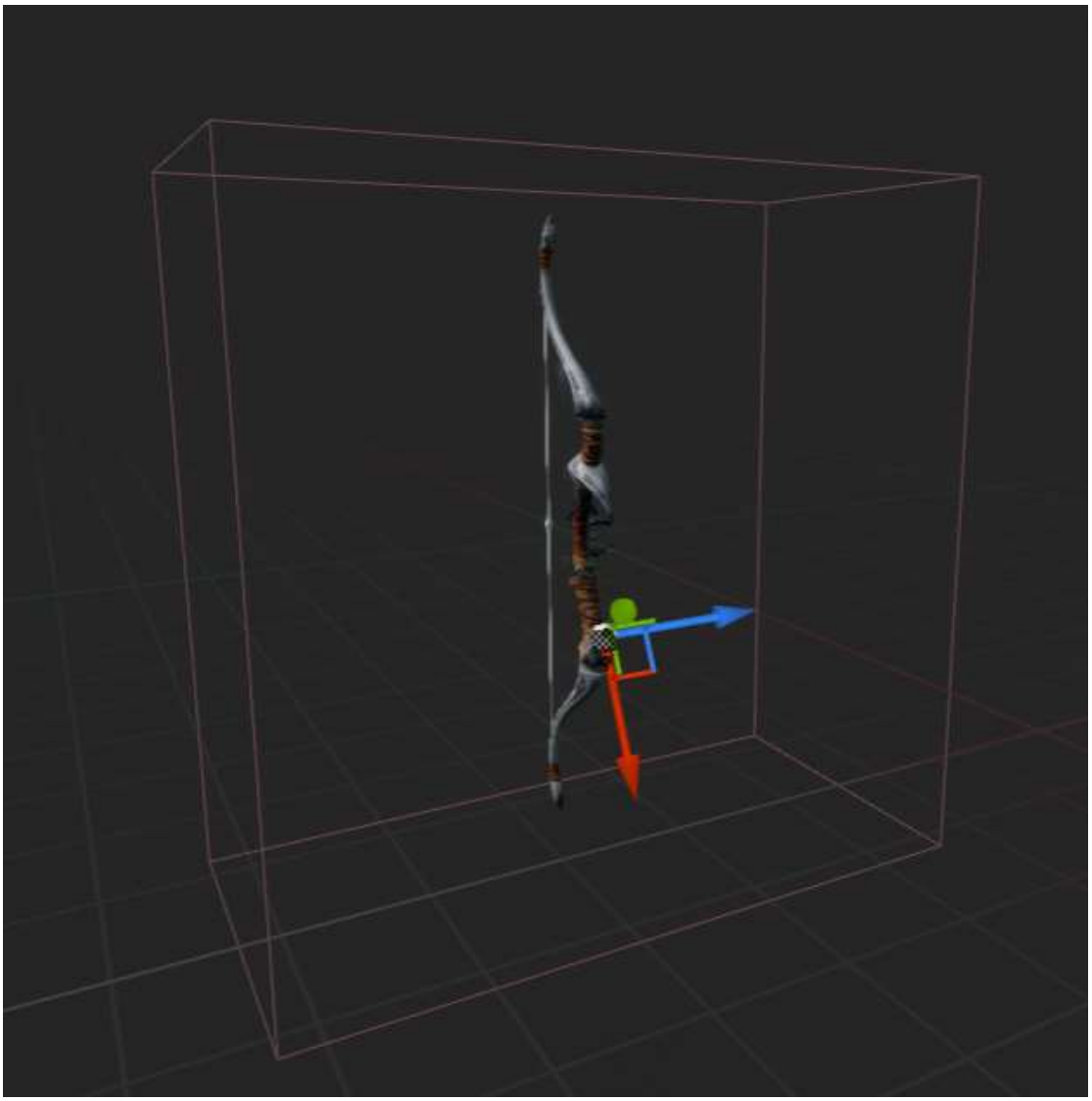


Rysunek 19 Wygląd obiektu odpowiedzialnego za główną pętlę gry

Na powyższym grafie widać wyszczególnione konkretne fragmenty odpowiadające za kolejne etapy gry. Najistotniejszymi z punktu widzenia gracza jest jej rozpoczęcie, pętla i zakończenie. Proces nie wejdzie do rozwinięcia, dopóki nie zostanie aktywowana przez gracza poprzez zestrzelenie odpowiedniego obiektu na mapie. Nim tego nie dokona, może dowolnie strzelać z łuku. Ma to na celu umożliwienie graczowi oswojenie z mechaniką strzelania, nim przyjdzie do trafiania ruchomych celów. Po trafieniu obiektu, którym na zaimplementowanej mapie jest dużych rozmiarów dzwon – gra przechodzi do głównej pętli. Tutaj w pierwszej kolejności do zmiennej zapisywana jest początkowa ilość przeciwników, a następnie w pętli zachodzi aktywowanie przeciwników. Aby tego dokonać, należy wylosować jeden z istniejących obiektów typu spawner i użyciu jego funkcji Spawn Enemy. Teraz, w każdej klatce sprawdzane są dwa istotne warunki – czy przeciwnicy jeszcze żyją i czy brama nie została zniszczona. Jeśli pierwszy z warunków zwróci wartość fałszywą – należy odczekać ustaloną wcześniej wartość czasową, aby dać graczowi możliwość odpoczynku i ponownie rozpocząć proces aktywowania przeciwników. W przypadku zniszczenia bramy, rozgrywka się kończy, co wiąże się z usunięciem z rąk gracza łuku i uruchomieniem efektów specjalnych w postaci ognia na otaczających gracza budynkach.

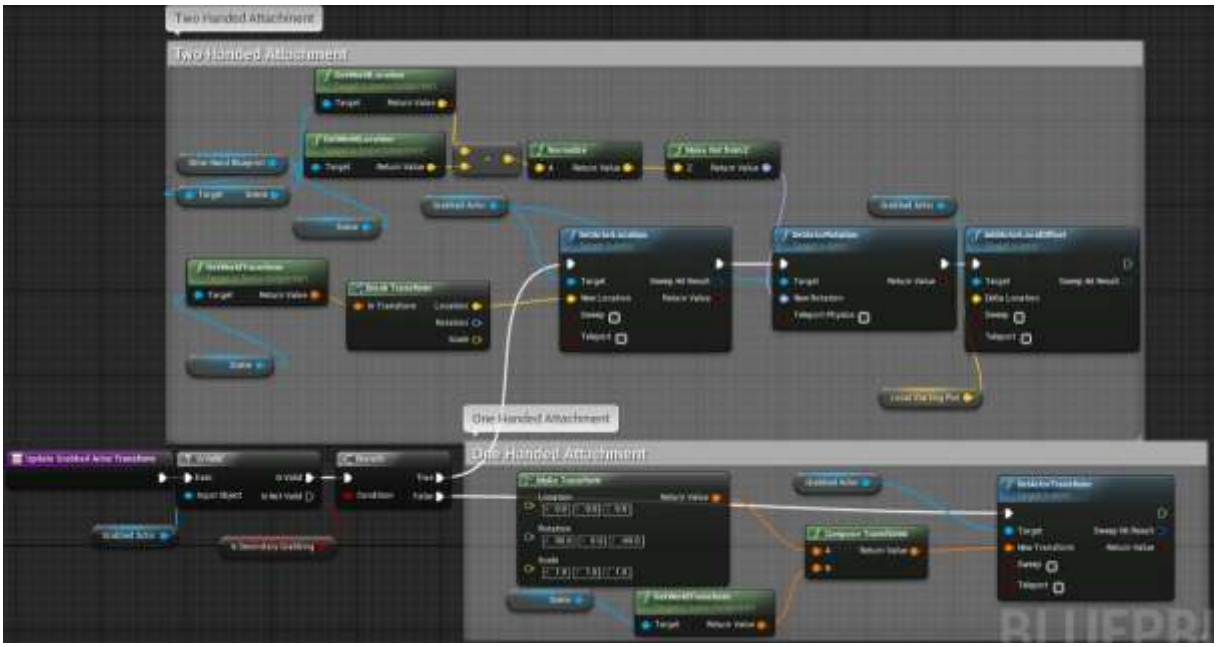
3.3.2. Funkcjonalność broni

Łuk, będący jedynym interaktywnym obiektem, z którym bezpośrednio ma styczność gracz, jest zarazem jedyną, zaimplementowaną bronią, która umożliwia wystrzeliwanie pocisków – w tym przypadku – strzał. Jak widać na rysunku 19, łuk nie składa się wyłącznie z obiektu graficznego, ale również z odpowiednich obiektów, które posiadają różnorakie zastosowania. Istotnym z punktu widzenia gracza jest obiekt, który graficznie został przedstawiony w postaci prostopadłościanu samych krawędzi. Wyznacza on obszar, w obrębie którego gracz jest w stanie złapać łuk. Jego znaczna wielkość nie jest przypadkowa – wynika to ze wspomnianego wcześniej problemu związanego z percepcją. Sam łuk nie jest również symulowany fizycznie – gracze odczuwają dyskomfort, gdy łuk sam znika, by pojawić się na stojaku, lub gdy ten spada na ziemię i występuje problem z jego ponownym pochwyceniem. Ponadto, łuk posiada komponenty, które są wymagane przez jego implementację do poprawnego funkcjonowania, takie jak miejsce złapania cięciwy, czy miejsce, gdzie zostanie oparta nowo utworzona strzała oraz celownik holograficzny.



Rysunek 20 Łuk wraz z reprezentacją graficzną obiektu przechwytującego wydarzenia nachodzenia dwóch obiektów

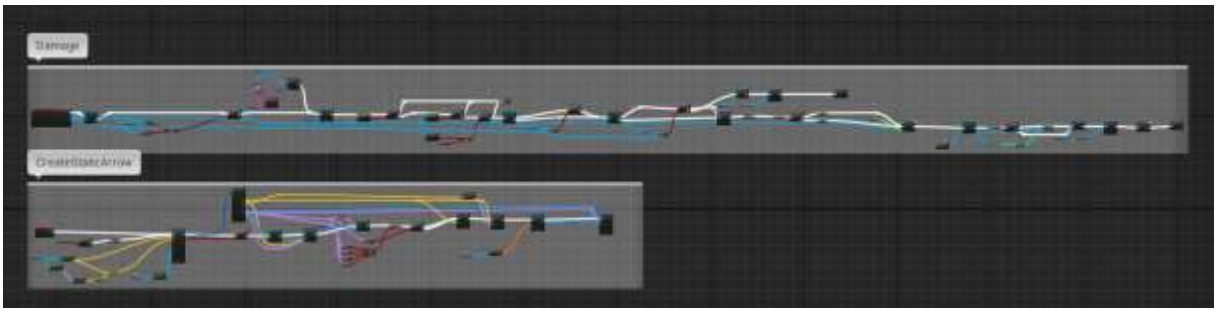
Klasa łuku, wraz z klasą strzały, stanowią rdzeń całego projektu i w związku z powyższym należą do najbardziej rozbudowanych w całym przedsięwzięciu. Naturalne dla gracza, a kluczowe w projekcie funkcjonalności wykorzystują dodatkowy interfejs, by zapewnić komunikację między kontrolerami, a łukiem. Przechwytuje między innymi momenty, gdy łuk jest podnoszony, bądź upuszczany i w odpowiedni sposób reaguje. Jest to tym bardziej istotne, iż złapany łuk musi poruszać się wraz z trzymanym kontrolerem, a częstotliwość odświeżania musi być na poziomie wyświetlanych klatek na sekundę, by gracz nie odczuwał problemów z opóźnieniem. Aby uzyskać satysfakcjonujący wynik, w klasie odpowiedzialnej za kontroler, została zaimplementowana funkcja odświeżająca jego transformację wywoływana w każdej klatce. Jako, że łuk, jest bronią dwuręczną w momencie korzystania z niej, a trzymaną w jednej ręce w stanie spoczynku – należało wprowadzić warunek sprawdzający, w jakim stanie są kontrolery gracza, czy oba trzymają, czy jeden.



Rysunek 21 Aktualizacja pozycji trzymanego elementu

W przypadku trzymania łuku, transformacja trzymanego obiektu jest zdecydowanie prostsza w implementacji, gdyż wymaga jedynie odpowiedniej rotacji względem kontrolera. Dwuręczna implementacja wymaga znormalizowania różnicy wektorów lokalizacji dwóch kontrolerów i na ich podstawie utworzenia wektora rotacji, któremu zostanie poddany łuk. W przypadku puszczenia, należy sprawdzić, która ręka puściła, ponieważ znowu prowadzi to do dwóch różnych rozwiązań – może to puścić broń, bądź wystrzelić strzałę. W pierwszym przypadku, należy sprawdzić, czy w momencie upuszczania broni nie była naciągana strzała, w przeciwnym wypadku może ona pozostać w powietrzu, w drugim zaś należy aktywować komponent służący do poruszania strzały. Ten sam warunek jest sprawdzany przed uruchomieniem funkcjonalności strzału. Nim jednak to nastąpi, należy obliczyć długość od punktu przyłożenia strzały do jej oparcia o cięciwę, gdyż siła nadana pociskowi rośnie wykładniczo w zależności od naprężenia linki. Sprawdzanie, czy cięciwa jest naciągnięta jest wywoływana w każdej klatce, z warunkiem początkowym sprawdzającym, czy powyższa czynność ma miejsce. Gdy warunek jest spełniony, brana jest różnica wektora początkowej pozycji złapanej cięciwy z aktualną pozycją w świecie komponentu, który jest trzymany przez kontroler i zwracana jest jego długość. Co klatkę wywoływane są również funkcjonalności związane z odpowiednim ustawieniem pozycji nowo powstałej strzały. Wystrzelenie powoduje aktywację komponentu Projectile Movement, który posiada również symulację fizyki i pozwala na odpowiednie ustawienie grawitacji i tarcia. Po nadaniu mu prędkości w przestrzeni lokalnej oraz ustawieniu, aby rotacja obiektu była zależna od wektora prędkości, tak by strzała płynnie zmieniała obrót w locie, pocisk leci w wybranym przez gracza kierunku. Po wypuszczeniu strzały, cięciwa wraca do poprzedniego kształtu w sposób możliwie realistyczny. Do tego celu zostało użyte proste fizyczne obliczanie sprężystości ciała, jakim jest cięciwa. Efekt ten jest aktywny cały czas, jednakże widzialny jest dopiero przy znacznym ruchu, takim jak naciągnięcie strzały.

Łuk umożliwia strzelanie trzema strzałami jednocześnie. Jest to zależne od poczynań gracza, tj. bonus zależny od tego, czy graczowi udało się zestrzelić balon, pojawiający się po unieszkodliwieniu przeciwnika. Ich pozycja jest wyliczana w dodatkowej funkcji, gdzie do odpowiedniej transformacji dodawany jest wektor z jedną wartością w osi Z, tak by strzały leciały pod delikatnie innym kątem niż pierwotna.



Rysunek 22 Prezentacja blueprintu z implementacją strzały

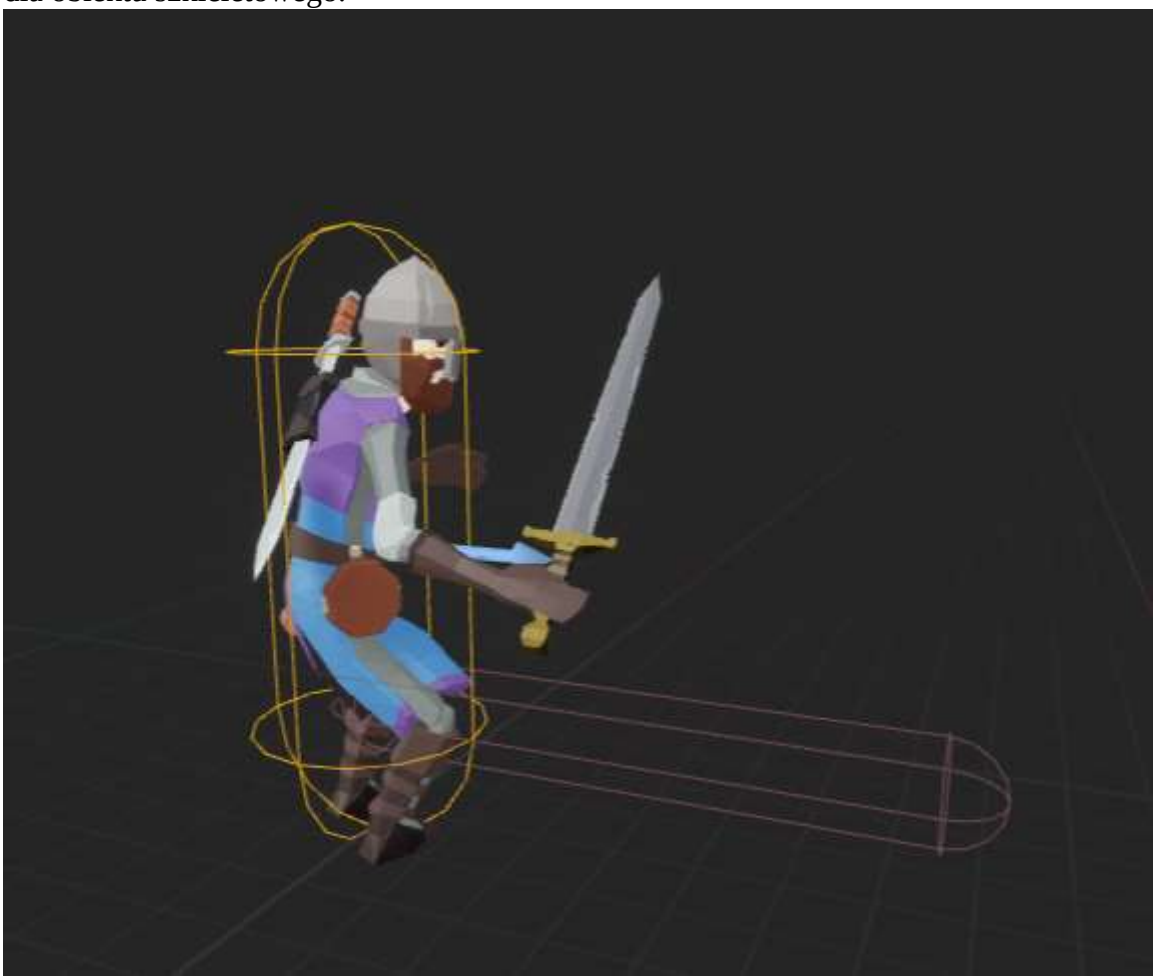
Drugim istotnym obiektem biorącym udział w strzelaniu, jest pocisk – strzała. W trakcie złapania cięciwy przez gracza, tworzona jest instancja klasy strzały, która posiada jej pełną funkcjonalność. W skrajnych przypadkach, była możliwość jej przypadkowego aktywowania poprzez zetknięcie grotu z otaczającym gracza murem przed jej faktycznym wystrzeleniem. Taka możliwość, po jej zaobserwowaniu, została zablokowana, gdyż była błędem utrudniającym grę. Na rysunku 21, można zauważyć, w jaki sposób zostały zaimplementowane dwie najistotniejsze funkcjonalności strzał, czyli zadawanie obrażeń, oraz tworzenie statycznego obiektu w miejscu uderzenia. Przed wywołaniem funkcji z zaimplementowanego interfejsu IDamageable, należy przejść przez szereg warunków, by upewnić się, że aktor, któremu mają zostać zaaplikowane obrażenia, jest odpowiedniego typu, jak również deaktywować możliwość zadawania obrażeń, gdy trafi się w inny przedmiot. Pominięcie tego kroku, może skutkować wielokrotnym zadawaniem obrażeń przez jedną strzałę, która wbiła się w ziemię. Warunki sprawdzają między innymi, czy obiekt posiada Tag „Ground", który został przypisany do każdego statycznego obiektu występującego na poziomie, bądź „Gate" – wpisany w obiekt bramy, jak również sprawdzone zostaje, czy obiekt nie przenika sam siebie, bądź łuku oraz czy implementuje wymagany interfejs. W przypadku spełnienia wszelkich warunków, aktywowana zostaje funkcja pojawienia statycznego aktora, gdyż upewniliśmy się, że dotknięta została instancja odpowiedniej klasy. Tworzenie statycznego obiektu wymaga przeprowadzenia kilku obliczeń. W pierwszej kolejności należy wykonać śladowanie (z ang. trace) z obiektu fizycznego, który stanowi reprezentację fizyczną strzały. Ma on na celu wykrycie miejsca uderzenia. Pozwoli to na dokładne określenie, w jaką część ciała został trafiony przeciwnik i w jaki sposób ma zostać przymocowana do niego strzała. Tutaj też sprawdzane jest, czy strzała trafiła w głowę, co zadaje dodatkowe obrażenia. Jako, że w hierarchii szkieletu postaci, znajdują się również oczy i brwi, one także są brane pod uwagę przy zwiększaniu urazów. W trakcie testów zauważono, że trafienie i zdobycie obrażeń krytycznych jest trudne, dlatego też, do bonusowych obrażeń zaliczone zostało również uderzenie w szyję. Po wykryciu miejsca uderzenia, instancja strzały zostaje przymocowana do trafionego komponentu. W tym momencie przepływ wraca do poprzedniej funkcji i w zależności od tego, w co trafiła strzała, zadaje, za pomocą odpowiedniego interfejsu, obrażenia, jak również zlicza trafienia do statystyk, by po chwili usunąć instancję obiektu.

3.3.3. Funkcjonalność przeciwnika

3.3.3.1. Wprowadzenie

Wykonanie opisywanego projektu, bez implementacji przeciwników, mijałoby się z celem. Przyczyna jest prosta – gatunek wymaga, aby gracz bronił konkretnego obiektu przed przeciwnikami. Idea z jaką działa zaimplementowany wróg jest stosunkowo prosta, aczkolwiek nawet tak prosty przeciwnik wymaga kilku interesujących struktur do poprawnego działania. Na poniższym rysunku (rysunek 22), można zauważyć poszczególne składowe, z których zbudowana jest postać. Oprócz wizerunku jest to oddzielny

miecz trzymany w ręku, poprzez użycie odpowiedniego gniazda, kapsuła biorąca udział w symulacji fizycznej, między innymi z otoczeniem, tak by móc sprawdzić, czy postać może przejść po danym obszarze. Jak również druga kapsuła, która jest wykorzystywana do zadawania obrażeń bramie, gdy przeciwnik do niej dotrze. Postać ta dziedziczy po utworzonej klasie BaseCharacter, która dziedziczy po klasie Character wbudowanej w silnik. Klasa ta wyróżnia się na tle innych między innymi zaimplementowanymi komponentami pozwalającymi na poruszanie. Dodatkowo posiada kapsułę dla postaci i zawiera miejsce dla obiektu szkieletowego.

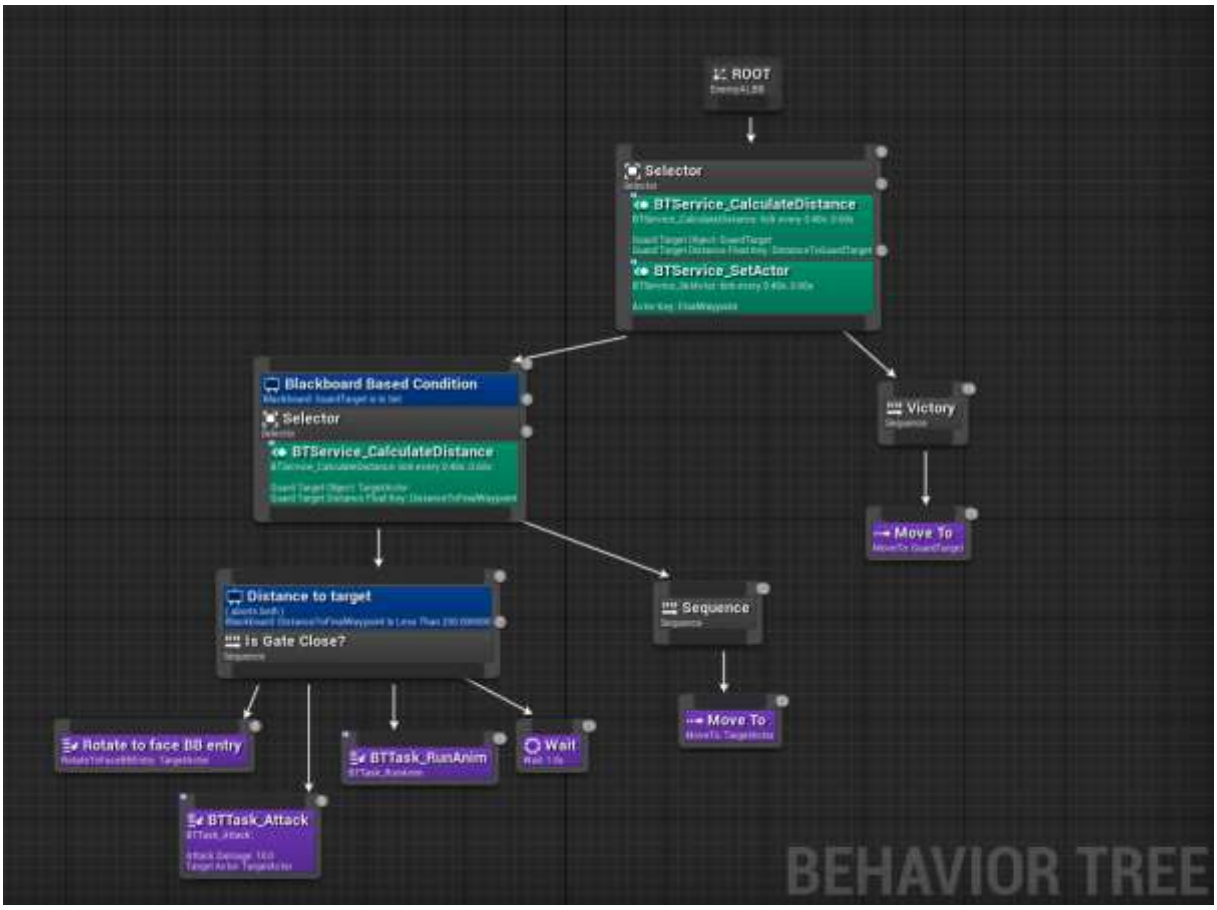


Rysunek 23 Bazowa postać wraz z fizycznymi reprezentacjami

3.3.3.2. Drzewo behawioralne

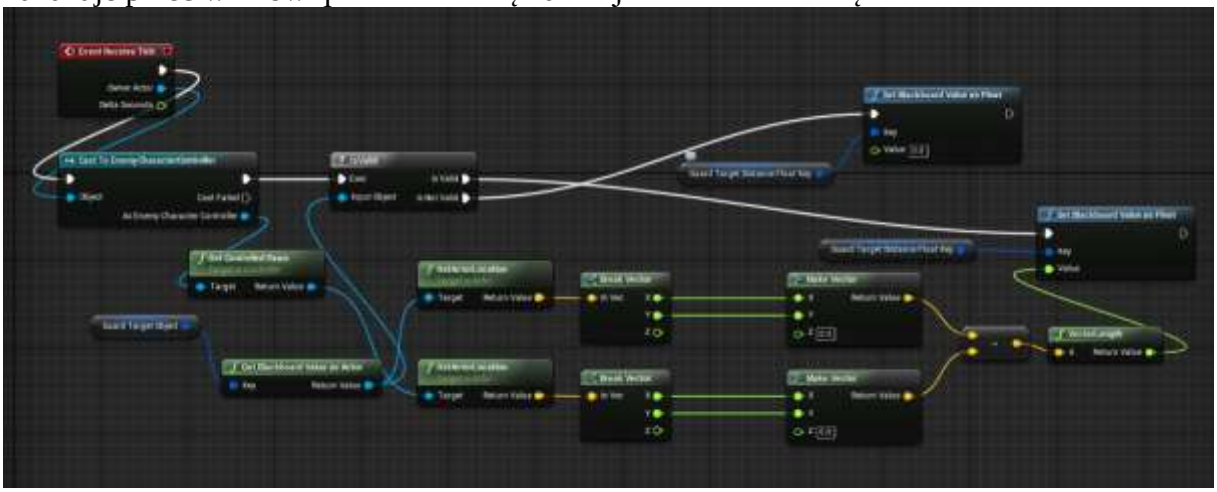
Podstawowym sposobem implementacji sztucznej inteligencji w silniku Unreal Engine 4 są drzewa behawioralne wykorzystujące dane zawarte w odpowiedniej tablicy o strukturze klucz – wartość. Jak zostało to określone przez Brendela Sewell - jest to serce sztucznej inteligencji [14] Rysunek 23 przedstawia drzewo behawioralne, które w kolejnych krokach umożliwia przeciwnikowi dotrzeć do obiektu, który ma być atakowany, obrócenie w jego kierunku i atakowaniu go w pewnych odstępach czasowych tak długo, aż zostanie zniszczony, po czym ma za zadanie przejść dalej.

Tak w skrócie można opisać poniższe drzewo (rysunek 23). Nim jednak będzie w stanie spełniać tak prozaiczne funkcje, należy zaimplementować odpowiednie serwisy. Spełniają one między innymi funkcję ustawiania odpowiednich wartości kluczom w tablicy.



Rysunek 24 Drzewo behawioralne

Nim przeciwnik zrobi swój pierwszy krok, funkcja musi obliczyć odległość od swojej pozycji, do poszukiwanego obiektu (rysunek 24). Po otrzymaniu zgłoszenia, następuje wzięcie lokalizacji kontrolowanej przez komputer postaci i obliczenie różnicy wektorowej z lokalizacją celu. Tak obliczona wartość, umożliwia przejście do kolejnych warunków. W tym samym bloku ustawiany jest również aktor, do którego mają się udać. Teraz następuje sprawdzenie, czy cel jest ustawiony. Ma to na celu wyeliminowanie ewentualnych błędów wykonania. Kolejnym krokiem jest sprawdzenie, czy odległość do punktu docelowego jest krótsza niż 200 jednostek. Dopóki tak nie będzie, postać będzie szła do celu. W przeciwnym razie, będzie wywoływała atak na aktora, który jest celem w odstępach sekundowych. W momencie, kiedy obiekt ten zostanie zniszczony, wywołuje się drugie poddrzewo, które nakazuje przeciwnikowi przemieścić się na miejsce zbiórki wewnątrz murów miasta.



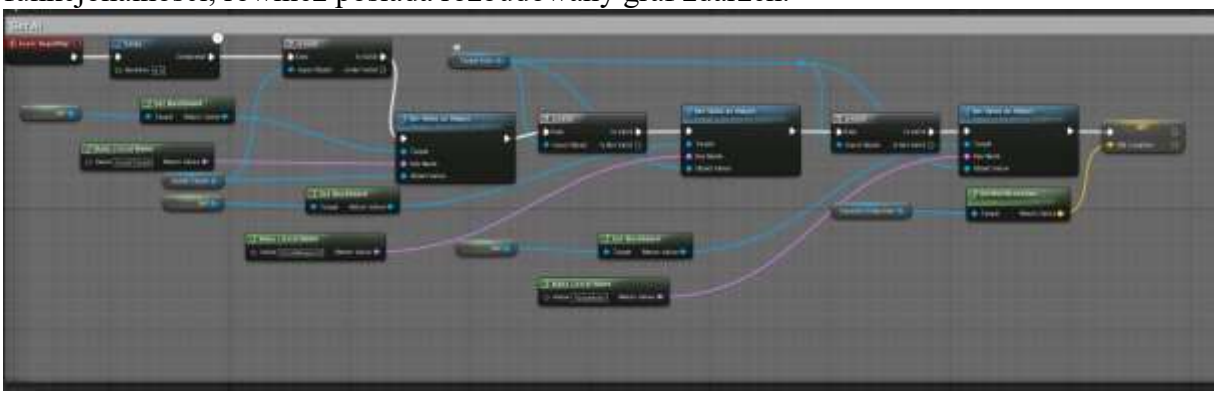
Rysunek 25 Obliczanie odległości między przeciwnikiem, a jego celem

3.3.3.3. Kontroler

Kontroler sztucznej inteligencji jest tworzony do każdej jednostki, która wymaga sterowania. Każda instancja posiada własną tablicę wartości i korzysta z drzewa behawioralnego. Ma za zadanie reagować na sytuacje występujące w środowisku i grze. Wszystko ma odbywać się bez interakcji ludzkiego gracza. Poza spełnianiem odpowiednich funkcji i posiadaniem odziedziczonych komponentów pozwalających postaciom poruszać się po drodze, jego graf ma za zadanie jedynie ustalenie jaki obiekt tablicy będzie wykorzystany i jakie drzewo należy brać pod uwagę w podejmowaniu decyzji odnośnie posiadanego pod kontrolą osobnika.

3.3.3.4. Graf zdarzeń

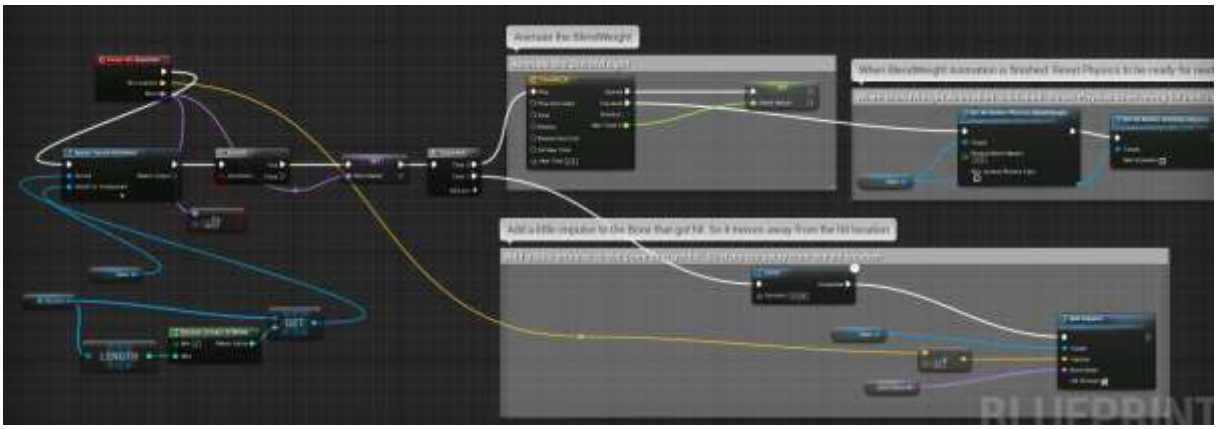
Przeciwnik, pomimo wielu osobnych obiektów, których potrzebuje do poprawnej funkcjonalności, również posiada rozbudowany graf zdarzeń.



Rysunek 26 Przypisywanie bazowych wartości do tablicy wykorzystywanej przez drzewo behawioralne

Choć drzewo i tablica zostały utworzone osobno, należy wpisać pewne wartości bazowe do każdej instancji przeciwnika. W związku z powyższym, zostało to tutaj zrobione. Na rysunku 25 widać, w jaki sposób następuje przypisanie konkretnych wartości odpowiadającym im kluczom.

Została również zaimplementowana dodatkowa fizyczna animacja, która ma symulować uderzenie strzały w przeciwnika. Ma to na celu dać graczowi informację zwrotną, że strzała rzeczywiście trafiła. Dodatkowo sprawia to, że gra jest odbierana jako bardziej realistyczna. Aby działało to poprawnie, musi być odpowiednio ustawiona fizyczna reprezentacja przeciwnika. Błędy w tym elemencie sprawią, że symulacja będzie niepoprawna i pełna artefaktów graficznych. Po poprawnym ustawieniu, należy wykrywać uderzenia i sprawdzać w jaką kość miało miejsce trafienie. Jest to potrzebne, gdyż dzięki temu ustalane jest, w jakie miejsce przeciwnika należy dodać impuls siły i z której strony.

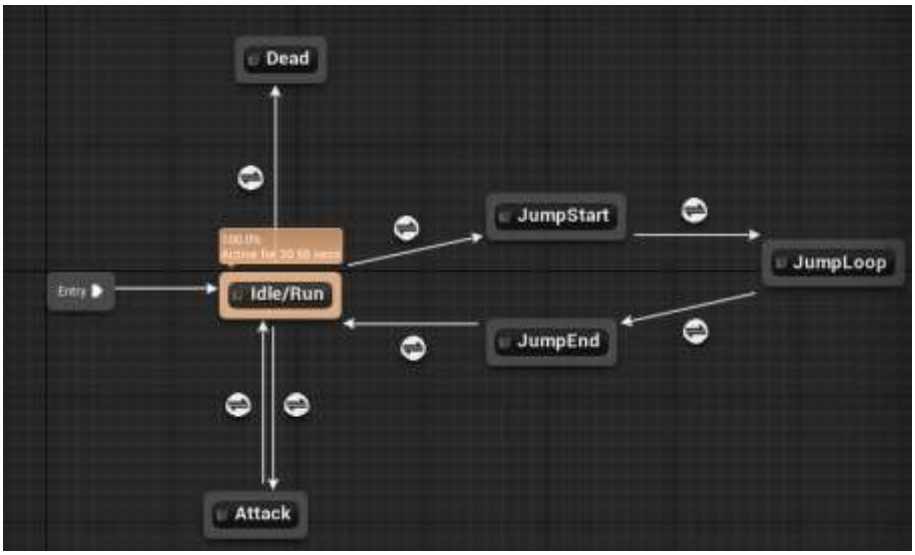


Rysunek 27 Impementacja animacji fizycznej w przeciwniku

Dodatkowo jest tutaj zaimplementowana podstawowa funkcjonalność przeciwników, pozwalająca określić, czy gracz skutecznie unieszkodliwił postać. Korzystając z interfejsu IDamageable, klasa otrzymuje informację, kiedy i z jaką siłą zostało wykonane uderzenie. Następnie obliczane jest ile punktów życia pozostało przeciwnikowi i na tej podstawie sprawdzane jest, czy przeciwnik powinien zostać usunięty z gry. Po jego śmierci jest tworzony bonusowy balon, który daje graczowi szansę na odnowienie punktów wytrzymałości bramy.

3.3.3.5. Maszyna stanów animacji

Omówione aspekty pozwalają na przemieszczanie postaci. Jednak bez utworzonej maszyny stanów animacji (rysunek 27) wraz z ustalonymi animacjami, postać będzie poruszała się w pozie T, w której jest tworzona zdecydowana większość modeli humanoidalnych. Maszyna stanów pozwala na graficzne rozbicie animacji w zestaw wymaganych stanów. Powyższe stany posiadają odpowiednie warunki, kiedy należy przejść z jednego podgrafu do kolejnego i w jaki sposób należy je między sobą mieszać. Pozwala to uprościć klasę blueprintu odpowiedzialną za strukturę, jak również oddzielić animacje od mechanik, co znacząco upraszcza implementację postaci w wielu projektach.



Rysunek 28 Główny graf maszyny stanów animacji przeciwnika

3.4. Animacje szkieletowe

Nim postać zacznie poruszać się w grze, należy przeprowadzić szereg procesów. W pierwszej kolejności trzeba taką postać zrealizować w grafice trójwymiarowej za pomocą zewnętrznego programu. Można do tego celu wykorzystać program 3ds max. Po wykonaniu postaci i odpowiednim przypisaniu materiałów do odpowiednich fragmentów modelu, pozostaje–utworzyć odpowiednią hierarchię kości. W przypadku wykonywania projektów z wykorzystaniem silnika Unreal Engine 4, można posłużyć się istniejącą hierarchią w prototypowej postaci. Pozwoli to zaoszczędzić czas i zapewni, że postać będzie posiadała wszelkie potrzebne kości do współpracy z frameworkiem. Mając już zdefiniowaną hierarchię, trzeba użyć modyfikatora Skin. Ma on na celu zdefiniowanie, w jaki sposób będą reagowały poszczególne wierzchołki modelu na zmianę rotacji i pozycji kości w hierarchii. Proces ten jest długotrwały, gdyż każdy punkt może być modyfikowany przez wiele kości, ograniczeniem jest, by suma wpływów na wierzchołek była równa jeden.

Rysunek 29 Widok postaci w programie 3ds max

Gdy już proces dopasowywania zostanie ukończony, należy wybrać jedną z kilku dróg związaną z procesem animowania. Można spróbować wykonywać je samemu, poprzez tworzenie kolejnych klatek kluczowych i umożliwienie programowi wykonanie przejścia między nimi w postaci klatek pośrednich. Innym rozwiązaniem, jest wykorzystanie źródła, do pobrania gotowych animacji. Choć bez wątpienia nie będą tak dopasowane do projektu, jak własne, jednakże nie zawsze jest czas i doświadczenie by wykonać odpowiednie animacje. Zaimplementowana postać w silniku wraz z hierarchią, prototypowa postać posiada również zestaw animacji, który był pomocny przy implementacji opisywanego projektu. Po zaimportowaniu odpowiednich plików do projektu, otrzymany zostanie poniższy efekt

(rysunek 29), w postaci zaimportowanego modelu wraz z hierarchią kości. Istotne jest, by postać była w pozie T, gdyż taka poza jest najłatwiejsza w dalszej pracy nad obiektami szkieletowymi. Animacje, to nic innego, jak zestaw plików binarnych, które posiadają informacje o transformacjach kości w każdej klatce. Należy uważać, by pierwsza klatka i ostatnia były bardzo do siebie zbliżone, aby uniknąć efektu szarpania przy zapętleniu animacji. Ważnym z punktu widzenia silnika jest, aby animacja posiadała również korzeń (najniżej położony element hierarchii), który nie będzie zmieniał swojego położenia w przestrzeni trójwymiarowej. Będzie on stanowił punkt odniesienia. Warto wiedzieć, że animacje wykonywane są w miejscu, a prędkość postaci jest ustalana w kontrolerze.

Rysunek 30 Reprezentacja graficzna postaci i hierarchii kości

Pomniejsze poprawki można wykonać w edytorze Persona – wewnętrznym edytorze animacji silnika Unreal Engine 4. Posiada on wiele funkcjonalności, od prostych przejść animacji, po tworzenie systemów odwrotnej kinematyki [15].

Tak wykonane animacje następnie są wykorzystywane przy tworzeniu blueprintu animacyjnego, który wybiera jaka animacja będzie odgrywana w danym momencie w oparciu o posiadane informacje na temat wycinka świata.

4. Podsumowanie

4.1. Wnioski

Rezultatem prac implementacyjnych jest w pełni funkcjonalna aplikacja, która umożliwia rozgrywkę dla jednego gracza w wirtualnej rzeczywistości. Po przestudiowaniu możliwych rozwiązań, zostało zanotowane, iż aplikacja ta w aktualnej formie może zostać pokazem technologicznym, z interesującym stylem graficznym. Jednak aby projekt mógł zaistnieć na rynku, wymaga jeszcze wielu godzin pracy. Jest to wynik nie tylko samego czasu potrzebnego na opracowanie nowych mechanik, czy dodatkowych poziomów, ale również potrzeba lepszego poznania tematyki wirtualnej rzeczywistości. Zachowania uczestników gier wykorzystujących VR różnią się od tych, którzy wybierają gry komputerowe. I tak jak niegdyś gracze zachwycali się, gdy możliwa była interakcja z obiektem symulowanym fizycznie, dziś, aby rynek zwrócił uwagę na produkt, musi się on znacząco wyróżniać względem innych, bądź będąc podobnym – przewyższać go dzięki lepszemu wykonaniu. Gracze starają się dotknąć wszystkiego i oczekują na odpowiednią reakcję. Szukają rozwiązań, o których nie myślą twórcy. Gdy posiadają możliwość łapania, starają się złapać każdą jedną możliwą rzecz w obrębie ich rąk.

Przygotowując produkt, który jest tworzony z myślą o graczach, należy myśleć jak gracz. Gra jest pod tym względem znacznie trudniejszym projektem niż inne aplikacje użytkowe, w których przedstawia się użytkownikowi, jakie ma możliwości i ten nie stara się wyjść poza ich ramy. W przypadku gier - gracze często szukają, w jaki sposób mogą ominąć wymyślone przez twórców zagadki, co pozwoli im ukończyć produkcję szybciej. Szukają wymagającej walki, wielu efektów specjalnych, czy niesamowitej oprawy graficznej. I z takimi oczekiwaniami zakładają gogle wirtualnej rzeczywistości. Problem zaczyna się w momencie, kiedy okazuje się, że pewnych mechanik nie można wykorzystać, bo powodują zawroty głowy, a inne wymagają mocy obliczeniowej, której brakuje, gdyż jest już wykorzystywana aby umożliwić płynną rozgrywkę. „Simulator Sickness. W przeciwieństwie do choroby lokomocyjnej, która jest spowodowana fizycznym ruchem, choroba symulatorowa jest produktem ubocznym sposobu, w jaki postrzegają użytkownicy wirtualną rzeczywistość, która różni się od tego, co mówią im ciała.” [15]. Zderzenie z rzeczywistością (wirtualną), stało się dla wielu przykrym doświadczeniem, gdyż nie doznali tego, czego oczekiwali. Zapominają, że ta technologia, dopiero wchodzi w fazę ogólnodostępną i potrzeba wielu lat pracy nad jej udoskonalaniem, zanim powstaną zachwycające gry z użyciem wirtualnej rzeczywistości.

Wirtualna rzeczywistość, to nie tylko oprogramowanie lecz również urządzenia, które ciągle są udoskonalane i pozwalają skutecznie rozwiązywać kolejne problemy. Mimo to, nie można przejść obojętnie obok wad, które dotykają aktualnych produktów. Między innymi występujący konflikt sensoryczny, gdy zmysł wzroku wysyła do mózgu inne informacje niż zmysł ruchu (podobny problem występuje w chorobie lokomocyjnej). Wymusza to inny sposób lokomocji niż poruszanie za pomocą kontrolera, gdyż może to zdezorientować użytkownika. W wyniku testów, producenci wystosowali pewne rady odnośnie tworzenia aplikacji z użyciem ich urządzeń, między innymi: implementowanie nieruchomych punktów odniesienia, tak by mózg mógł zrozumieć, że nie tylko on się nie porusza. Z tych samych przyczyn, radzi się, aby unikać szybkiego przyspieszenia, czy gwałtownej zmiany obszaru. W produkcjach na komputery stacjonarne, popularne jest, aby stosować rozmycie ekranu, które ma powodować zwiększenie dynamiki obrazu. Po kilku testach z użyciem tego efektu w projekcie, okazało się to bardzo nietrafione, podobnie jak z użyciem płytkiej głębi ostrości. W świecie rzeczywistym wzrok akomoduje się tak, aby widzieć ostro obiekty znajdujące się w odpowiedniej odległości. W przypadku gier jest to wykorzystywane, aby zasymulować

działanie oka. Jest to wizualnie bardzo ciekawy efekt, jednakże w przypadku wirtualnej rzeczywistości, należy z niego zrezygnować. Nie mogąc wykrywać ruchów gałek ocznych, nie jesteśmy w stanie stwierdzić, na jaki punkt aktualnie patrzy się gracz, a przypadkowe rozmycie może spowodować bardzo nieprzyjemne w skutkach urazy. We wcześniejszej wersji aplikacji, istniała możliwość bardziej swobodnego poruszania się, jednakże gracz zakładając gogle nie widzi świata rzeczywistego, a co za tym idzie nie jest w stanie ocenić odległości od ścian. Choć była możliwość implementacji półprzezroczystych ścian, które pojawiałyby się, gdy dochodzi się do granicy obszaru grania, zmniejszało to imersję świata i w związku z tym postanowiono pozbyć się tej mechaniki, zamknąć gracza w mniejszym obszarze i skupić się na grze w jednym miejscu. Zastosowanie się do wszelkich zdobytych i doświadczonych rad, umożliwiło to zmaksymalizowanie komfortowego czasu grania.

Największym problemem wirtualnej rzeczywistości pozostaje cena. Zakup urządzeń wirtualnej rzeczywistości to wydatek rzędu kilku tysięcy złotych. Dodatkowo wymagany jest komputer, który będzie w stanie udźwignąć aplikacje i wyświetlać je z częstotliwością ekranów zamontowanych w goglach, co sprawia, że te sprzed trzech lat spełniają zaledwie minimalne wymagania sprzętowe. Skutkuje to dwojako. Z jednej strony posiadacze urządzeń poszukują dobrych produkcji, których jest niewiele, a co za tym idzie można podnieść cenę na produkty, które wychodzą, gdyż jest duża szansa, że i tak zostaną zakupione. Z drugiej natomiast mała liczba osób powoduje, iż nawet dobry produkt może nie trafić do dużego grona odbiorców, bądź też kupujących jest zbyt mało, by pozwoliło to na utrzymanie się z wytworzenia aplikacje wirtualnej rzeczywistości.

4.2. Możliwe kierunki rozwoju

Projekt początkowo zakładał wykonanie gry prostej, która zostanie wykorzystywana wyłącznie w celach promocyjnych wirtualnej rzeczywistości, jednakże już w trakcie pisania projektu, powstało wiele różnych pomysłów, które mogą znacząco rozwinąć produkt, tak by zwiększyć jego szanse na rynku.

Dodanie dodatkowych, zróżnicowanych map będzie naturalnym posunięciem. Pozwoli to na zwiększenie ilości różnorodnych rozgrywek. Jednakże koncepcja ciągłej obrony, tylko w innych miejscach, nie wystarczy, aby zainteresować graczy. Implementacja stacjonarnych przeciwników, którzy będą atakowali z gór przy użyciu broni dalekiego zasięgu, takich jak katapulty jest jednym z pomysłów na zwiększenie zainteresowania. Gracz, aby wyeliminować zagrożenie będzie zmuszony trafić płonącą strzałą w broń przeciwnika. Płonąca strzała to nie jedyny modyfikator, który zostanie wprowadzony w przyszłości do broni gracza. Dodatkowo zostaną wprowadzone strzały o innych efektach. Lodowa, która pozwoli zamrozić przeciwników, wybuchowa, która spowoduje efekt podobny do beczki wypełnionej prochem, raniąc wszystkich wokół, czy trujące, które będą zadawały obrażenia co pewien okres czasu. Aby powyższe dodatki miały znaczenie, należy również znacząco zwiększyć grono przeciwników. Pomijając już zaimplementowanych rycerzy, pomysłem jest dodanie postaci z legend, takich jak smoki, czy ogry. Różni przeciwnicy będą reagować w różne sposoby na atak gracza. Do bestiariusza ma szanse dołączyć również banshee, czy bazyliszek. Zwiększenie różnych przeciwników będzie wymagać różnych podejść od gracza. Odejście w stronę fantastyki pozwoli na wiele interesujących rozwiązań. Rozwinięciu ulegnie również warstwa strategiczna projektu. Gracz nie będzie już ograniczony wyłącznie do strzelania. Aby przetrwać, będzie musiał również rekrutować swoich rycerzy, którzy będą bronić jego zamku. Sterowanie nimi będzie jednak uproszczone i sprowadzać będzie się do wydawania poleceń poprzez wystrzelenie specjalnej strzały w miejsce, w które mają się udać. Pozwoli to nadać większego znaczenia punktom, które są zdobywane za dobrą grę. Dodany zostanie również ranking, który pozwoli graczom konkurować ze sobą.

Po procesie kreatywnym, można zauważyć, że zaimplementowany projekt, to zaledwie prototyp tego, co zostanie wykonane w przyszłości. Duża ilość pomysłów daje szanse projektowi na zaistnienie na rynku. Innowacyjne rozwiązania, wiele dodatkowych elementów pozwolą graczom cieszyć się godzinami spędzonymi w wirtualnej rzeczywistości i stać się nowym Robin Hoodem.

Bibliografia

[1] Jaron Lanier, *Virtual Reality: Definition and Requirements,* https://www.nas.nasa.gov/Software/VWT/vr.html (29.10.2017)

[2] Paweł Olszewski, *Historia wirtualnej rzeczywistości,* http://www.komputerswiat.pl/centrum-wiedzy-konsumenta/gaming/wszystko-o-wirtualnej-rzeczywistosci/historia-wirtualnej-rzeczywistosci.aspx (29.10.2017)

[3] Łukasz Rosiński, *Raport dotyczący sytuacji na rynku VR,* http://thefarm51.com/ripress/Rynek_VR_raport_The_Farm_51.pdf (31.10.2017)

[4] Ben Lang, *Conan O'Brien Plays 'Wilson's Heart' in Latest 'Clueless Gamer' Segment*, https://www.roadtovr.com/conan-obrien-plays-wilsons-heart-latest-clueless-gamer-segment/ (31.10.2017)

[5] Oficjalna strona gry Kingdom Watcher https://www.subdreamstudios.com/kingdom-watcher/ (31.10.2017)

[6] Peter Graham, *Grab Your Bow And Defend The Town In Elven Assassin,* https://www.vrfocus.com/2016/09/grab-your-bow-and-defend-the-town-in-elven-assassin (31.10.2017)

[7] Słownik języka polskiego, hasło*: gra,* https://sjp.pwn.pl/sjp/gra;2462662.html (03.11.2017)

[8] Słownik języka polskiego, hasło: *gra komputerowa,* https://sjp.pwn.pl/sjp/gra-komputerowa;2462665.html (03.11.2017)

[9] Słownik języka polskiego, hasło: *rzeczywistość wirtualna,* https://sjp.pwn.pl/sjp/rzeczywistosc-wirtualna;2518604.html (03.11.2017)

[10] http://store.steampowered.com/app/457960/Holopoint/ (08.11.2017)

[11] Tomek Grochowiak, *Poziom trudności gier komputerowych z perspektywy projektanta*, (08.11.2017)

[12] Donaldson N., Whiting N., *Going Off the Rails: The Making of Bullet Train,* 2016, https://www.gdcvault.com/play/1023648/Going-Off-the-Rails-The (15.11.2017)

[13] Chris Simpson, *Behavior trees for AI: How they work,* https://www.gamasutra.com/blogs/ChrisSimpson/20140717/221339/Behavior_trees_for_AI_How_they_work.php (28.11.2017)

[14] Sevell Brenden, *Blueprints Visual Scripting for Unreal Engine,* Packt Publishing, Birmingham, 2015

[15] Mitch McCaffrey, *Unreal Engine VR Cookbook,* Addison-Wesley, London 2017

Spis ilustracji