Szkoła Główna Gospodarstwa Wiejskiego
w Warszawie
Wydział Zastosowań Informatyki i Matematyki

Maciej Wygoda
172407

Implementacja sieciowej gry wideo z wykorzystaniem silnika Unreal Engine 4

Implementation of an online video game using Unreal Engine 4

Praca dyplomowa inżynierska
na kierunku Informatyka

Praca wykonana pod kierunkiem
dr. Bartłomieja Kubicy
Wydział Zastosowań Informatyki i Matematyki
Katedra Zastosowań Informatyki

Warszawa, 2018

Praca przygotowana zespołowo przez:

1. Maciej Wygoda
172407
który jest autorem:
rozdziału 1 ze strony 8, rozdziału 2 ze stron 9-14, rozdziału 3 ze stron 13-14, podrozdziałów 4.1, 4.2, 4.3 ze stron 15-17, podrozdziałów 4.4.3, 4.4.4 ze strony 18, podrozdziałów 4.4.6, 4.4.7, 4.5 ze strony 20, podrozdziału 4.7 ze strony 23

2. Marcin Szadkowski
167607
który jest autorem:
rozdziału 1 ze strony 9, rozdziału 3 ze stron 15-16, podrozdziału 4.2 ze strony 17, podrozdziału 4.4.1 ze strony 19, podrozdziałów 4.4.2, 4.4.3 i 4.4.4 ze strony 20, podrozdziału 4.4.5 ze strony 21, podrozdziału 4.4.6 ze strony 22, podrozdziału 4.5 ze strony 22, podrozdziału 4.6 ze stron 23-26, rodziału 7 ze strony 29

Oświadczenie promotora pracy

Oświadczam, że wskazane przez autora rozdziały pracy dyplomowej przygotowanej zespołowo zostały przygotowane pod moim kierunkiem i stwierdzam, że spełniają one warunki do przedstawienia tej pracy w postępowaniu o nadanie tytułu zawodowego.

Data ..... Podpis promotora pracy .....

Oświadczenie autora pracy

Świadom odpowiedzialności prawnej, w tym odpowiedzialności karnej za złożenie fałszywego oświadczenia, oświadczam, że wskazane przeze mnie rozdziały pracy dyplomowej przygotowanej zespołowo zostały napisane przeze mnie samodzielnie i nie zawierają treści uzyskanych w sposób niezgodny z obowiązującymi przepisami prawa, w szczególności z ustawą z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. Nr 90 poz. 631 z późn. zm.)

Oświadczam, że przedstawiona praca nie była wcześniej podstawą żadnej procedury związanej z nadaniem dyplomu lub uzyskaniem tytułu zawodowego.

Oświadczam, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną. Przyjmuję do wiadomości, że praca dyplomowa poddana zostanie procedurze antyplagiatowej.

Data ..... Podpis autora pracy .....

Streszczenie

Implementacja sieciowej gry wideo z wykorzystaniem silnika Unreal Engine 4
Niniejsza praca jest opisem implementacji sieciowej gry wideo pod tytułem „thesis_1“ z wykorzystaniem silnika Unreal Engine 4. Zawiera opis silnika, prezentuje architekturę gry oraz zastosowane rozwiązania.

Słowa kluczowe – Unreal Engine 4, tworzenie gier wideo, gra wideo

Summary

Implementation of an online video game using Unreal Engine 4
This study is a description of an implementation of an online video game „thesis_1“ using Unreal Engine 4. It describes the engine and presents the game's architecture and applied solutions.

Keywords – Unreal Engine 4, game development, gamedev, video game

Spis treści

1 Wstęp

Gry wideo stanowią rozrywkę dla coraz szerszego grona odbiorców, a sama branża nieustannie rośnie, o czym świadczy fakt, iż pod względem wygenerowanych przychodów prześcignęła już branżę filmową oraz muzyczną [16]. Gry coraz częściej postrzegane są jako nowoczesne medium przekazu oraz forma wyrazu artystycznego i poruszają tematy dotychczas zarezerwowane dla literatury i kinematografii.

Tworzenie gier wideo (*ang. game development*) to obszerne zagadnienie łączące w sobie wiele dziedzin. Od strony technicznej są to między innymi grafika komputerowa, inżynieria oprogramowania, programowanie komputerów, bezpieczeństwo komputerowe, matematyka[2]. W związku z tym, że tworzenie gier to proces długi i skomplikowany, istnieje wiele narzędzi wspierających go, a jednym z najpopularniejszych jest silnik *Unreal Engine 4* (zwany dalej "UE4").

1.1 Cel i zakres pracy

Głównym celem niniejszej pracy jest rozwój wiedzy o procesie tworzenia gier wideo oraz technologii UE4. Na całą pracę składa się zaprojektowanie i zaimplementowanie gry z użyciem UE4 oraz podstawowy opis silnika i implementacji gry.

Uwagę skupiono między innymi na poznawaniu działania i efektywnym wykorzystywaniu technologii UE4 oraz oprogramowania do modelowania i animacji Blender oraz zdobyciu doświadczenia w pracy zespołowej. Praca ta może z powodzeniem służyć za przykład i drogowskaz dla osób chcących napisać własną grę.

1.2 Motywacja

Motywację stanowiły chęć zgłębienia technologii UE4, podjęcia technicznego wyzwania, jakie stawia zaprogramowanie gry wideo oraz pasja do gier.

2 Unreal Engine 4

2.1 Czym jest silnik gry?

Przez pojęcie „silnik gry“ rozumie się zbiór funkcji i narzędzi (*ang. framework*) wspierający tworzenie gier. Musi on oferować przede wszystkim renderowanie grafiki, dźwięku i obsługę sterowania. Współczesne silniki posiadają jednak znacznie więcej funkcji, a są to między innymi obsługa sieci, symulacja fizyki, edytory shaderów i efektów cząsteczkowych, produkcja przerywników filmowych oraz obsługa wielu platform na przykład komputerów, konsol czy urządzeń mobilnych takich jak smartfony. Większość silników dystrybuowana jest wraz z edytorem będacym graficznym interfejsem między programistą, a funkcjami silnika. Wykorzystanie jednego silnika do stworzenia wielu różnych gier znacząco skraca okres produkcji i stanowi powszechną w branży praktykę[1][13].

2.2 Język programowania

Jednym z problemów dziedziny tworzenia gier jest zebranie standardów technicznych, które zostały by przyjęte przez całą branżę. Rozwój tej gałęzi informatyki jest napędzany przez duże firmy, które wprowadzają na rynek nowe technologie. Utrudnia to dobranie narzędzi, które można by wykorzystywać przez długi okres czasu[2]. Ogólnodostępne standardy są ważne aby małe studia mogły obok dużych korporacji dokładać nowe rozwiązania. Z tych powodów do tworzenia w silniku UE4 wykorzystywany jest *C++*, który pod wieloma względami jest doskonałym narzędziem. Korzystanie z niego jest darmowe, przez co nawet małe studia mogą pozwolić sobie na jego użycie. Oprócz dostępności, narzędzia do tworzenia gier powinny łączyć w sobie wydajność oraz umożliwiać szybkie tworzenie projektów. Wszystkie te cechy łączy w sobie język *C++*[3].

2.3 Funkcje silnika Unreal Engine 4

UE4 swoją popularność zawdzięcza między innymi otwartemu źródłu, co w pewnym stopniu umożliwia programistom dostosowanie silnika do potrzeb gry na przykład poprzez modyfikacje w działaniu silnika lub tworzenie narzędzi dla reszty zespołu. Ponadto UE4 jest w stanie renderować grafikę bardzo zbliżoną do fotorealizmu, co w połączeniu z szeroką gamą oferowanych funkcji sprawia, że poza grami korzysta się z niego na przykład do produkcji spotów i aplikacji reklamowych [17][18].

Oprócz podstawowych funkcji takich jak renderowanie grafiki czy obsługa sterowania do dyspozycji oddane zostały między innymi [4]:

Symulacja fizyki: UE4 korzysta z silnika fizyki *PhysX 3.3* dzięki czemu wiarygodnie symuluje kolizje obiektów i inne oddziaływania fizyczne. Ponadto edytor umożliwia modyfikację panujących zasad, co pozwala na lepsze przedstawienie wizji autorów gry.

Edytor interfejsu użytkownika: Interfejs stanowi istotny element w komunikacji między grą, a grającym. UE4 zapewnia rozbudowany edytor pozwalający na tworzenie między innymi takich elementów interfejsu jak *HUD (head-up display)* czy menu.

Drzewa behawioralne: Stanowią one podstawę sztucznej inteligencji w UE4 i pozwalają na zaprogramowanie zachowania postaci sterowanych przez komputer w zależności od odbieranych przez nie bodźców i stanu sceny.

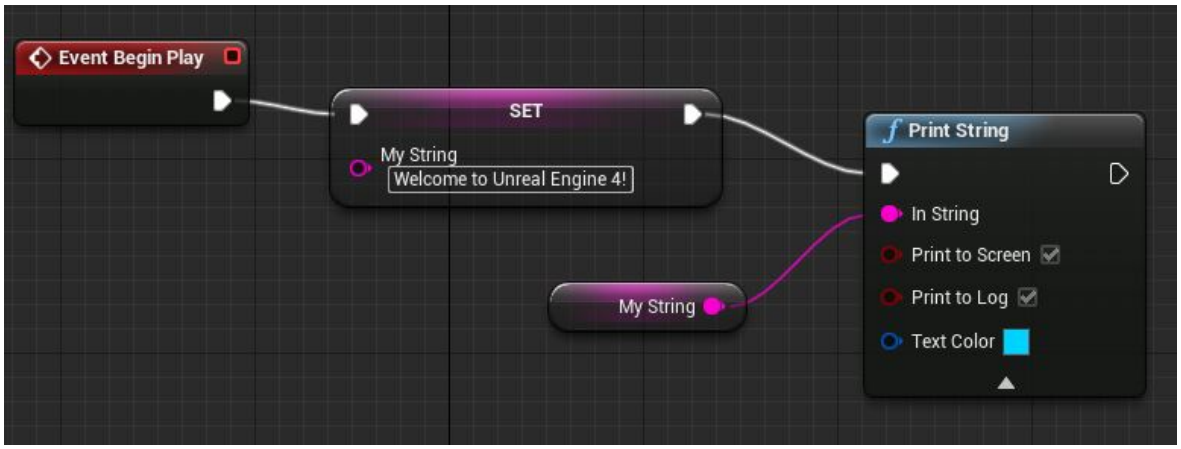
Sequencer: Jest to narzędzie służące do produkcji przerywników filmowych. Jego obsługa przypomina pracę z oprogramowaniem do montażu filmów i modelowania 3D. Edytor ten uwzględnia elementy takie jak oś czasu, ujęcia kamery, szkielety i animacje obiektów.

Networking: UE4 powstał z myślą o rozgrywce sieciowej. Wobec tego dużo uwagi poświęcono utworzeniu odpowiedniej abstrakcji, ułatwiającej programowanie komunikacji sieciowej. Komunikacja ta wykorzystuje model klient-serwer, co oznacza, że istnieje jeden serwer z autorytatywną instancją świata oraz wielu klientów, których światy są aktualizowane na podstawie tego, co dzieje się na serwerze. Aktualizacje te mogą być zrealizowane dwojako: jako aktualizacje właściwości poszczególnych obiektów lub za pomocą wywołania zdalnej procedury *(ang. remote procedure calls – RPC)*. W obu przypadkach wykorzystywany jest protokół *UDP*, który jest zawodny, ale generuje o wiele mniejszy narzut komunikacyjny, niż konkurujący z nim *TCP*. UE4 ma zaimplementowany własny system, zapewniający częściową niezawodność komunikacji [15].

Analiza wydajności: Osiągnięcie iluzji ruchomego obrazu wymaga wygenerowania przynajmniej 15 klatek na sekundę (*ang. frames per second - FPS*), obecnie na konsolach do gier pożądaną wartością jest przynajmniej 30FPS, na komputerach 60FPS[10], a w tytułach esportowych nawet dwa razy więcej. Miara ta jest odzwierciedleniem płynności obrazu i wydajności gry, a na wydajność składają się stopień skomplikowania scen i obliczeń oraz optymalizacja. UE4 zapewnia narzędzia do profilowania, dzięki którym łatwiejsze staje się zidentyfikowanie obszarów wymagających optymalizacji.

Edytor materiałów: W konwencji UE4 materiał to zbiór informacji o wizualnej stronie obiektu. Definiuje on kolor, teksturę, rodzaj powierzchni obiektu (na przykład metal, drewno), przezroczystość i inne cechy, które mogą mieć wpływ na zachowanie światła padającego na dany obiekt. UE4 posiada rozbudowany edytor materiałów opierający się na programowaniu graficznym (*ang. visual scripting*).

Blueprints Visual Scripting: Jest to system programowania rozgrywki, wykorzystujący graficzną reprezentację elementów takich, jak funkcje, klasy czy zmienne, które po połączeniu stanowią pewną logikę (rys. 1)[6]. Za pomocą blueprintów można (często w krótszym czasie) osiągnąć efekt podobny, jak za pomocą kodu, przy czym dla osób bez doświadczenia w programowaniu są one o wiele prostsze w użyciu. Ponadto mogą dziedziczyć po klasach, napisanych za pomocą kodu, a programiści mają możliwość



Rysunek 1. Blueprint

rozwijania blueprintów, poprzez programowanie kolejnych węzłów do użycia przez resztę zespołu. Blueprinty umożliwiają nawet opracowanie kompletnej gry bez użycia kodu. Nie oznacza to jednak, że kod stał się bezużyteczny. Kod napisany w *C++* jest wydajniejszy od blueprintów (to znaczy szybciej się wykonuje, co bezpośrednio wpływa na liczbę generowanych klatek na sekundę). W wielu przypadkach jest też czytelniejszy (na przykład obliczenia w pętli) i łatwiejszy w utrzymaniu (na przykład kontrola wersji przy pracy zespołowej).
Oba te podejścia są wykorzystywane w profesjonalnym środowisku, a kluczem do sukcesu jest ich umiejętne połączenie, co zaprezentowano w dalszej części tej pracy.

2.4 Konwencja i architektura rozgrywki

Styl rozgrywki zależy od gry i wizji jej autora, jednak istnieją pewne cechy wspólne widoczne w niemal każdym tytule. Jest to na przykład sterowanie za pomocą urządzeń wejścia czy pewien zbiór zasad gry. Z tego powodu w UE4 przyjęto zaprezentowaną poniżej konwencję dotyczącą rozgrywki (rys. 2)[11].

Level: W terminologii UE4 *level* jest zbiorem wszystkich obiektów, które gracz widzi lub może wejść z nimi w interakcje. Stanowi on reprezentację obszaru, w którym toczy się gra.

Actor: Jest to bazowa klasa dla każego obiektu, który można umieścić w *levelu*. Obiekty te nie muszą mieć fizycznej reprezentacji. Często zawierają dodatkowe komponenty (*ActorComponents*) określające na przykład sposób porusza się obiektu. Oprócz tego *actor* posiada obsługę replikacji właściwości i wywołań funkcji przez sieć.

Pawn: Jest to *actor*, który może być kontrolowany przez gracza lub komputer. Stanowi ich fizyczną reprezentację w grach, które tego wymagają.

Character: Jest to humanoidalny *pawn* rozszerzony o następujące komponenty:

SkeletalMeshComponent definiujący geometrię postaci w przypadku animacji szkieletowych,

CapsuleComponent wykorzystywany przy kolizjach z innymi obiektami,

CharacterMovementComponent opisujący ludzkie ruchy takie jak chodzenie, bieganie czy pływanie oraz właściwości związane z ruchem na przykład prędkość chodzenia czy wpływ grawitacji.

Controller: jest to *aktor*, który po przejęciu *pawna* sprawuje nad nim kontrolę. Wyróżniamy dwa rodzaje: *PlayerController*, który stanowi interfejs między grającym, a sterowaną przez niego postacią (reprezentuje wolę gracza) oraz *AIController*, który decyduje o zachowaniach postaci na podstawie zaprogramowanych wcześniej drzew behawioralnych.

PlayerController danego gracza w przypadku rozgrywki sieciowej występuje w dwóch instancjach: po jednej na serwerze oraz urządzeniu grającego, co należy brać pod uwagę podczas programowania tego elementu.

HUD (*ang. head-up display*): podręczne informacje wyświetlane na ekranie gracza na przykład jego obecny wynik, stany zdrowia widocznych postaci czy tak zwana minimapa.

Camera: decyduje o perspektywie, z której grający obserwuje *level*.

GameMode: zawiera informacje takie jak zasady gry czy warunki zwycięstwa. Nie powinien zawierać żadnych informacji potrzebnych klientom, ponieważ istnieje jedynie na serwerze. Decyduje również o tym, który *GameState* i *PlayerState* zostanie wykorzystany. Wybór *GameMode-a* zależy od wczytywanego *levelu*.

GameState: zawiera informacje o obecnym stanie rozgrywki na przykład czy mecz już się rozpoczął, wykonane misje, wyniki, listę graczy. *GameState* istnieje zarówno na serwerze jak i u klientów oraz jest replikowalny.

PlayerState: zawiera informacje o uczestniku rozgrywki na przykład jego imię, wynik czy zespół, do którego należy. Zarówno serwer jak i wszyscy klienci posiadają kopie *PlayerState-ów* dotyczących każdego grającego (co nie ma miejsca w przypadku *PlayerControllerów* – każdy klient wie jedynie o swoim *PlayerControllerze*).

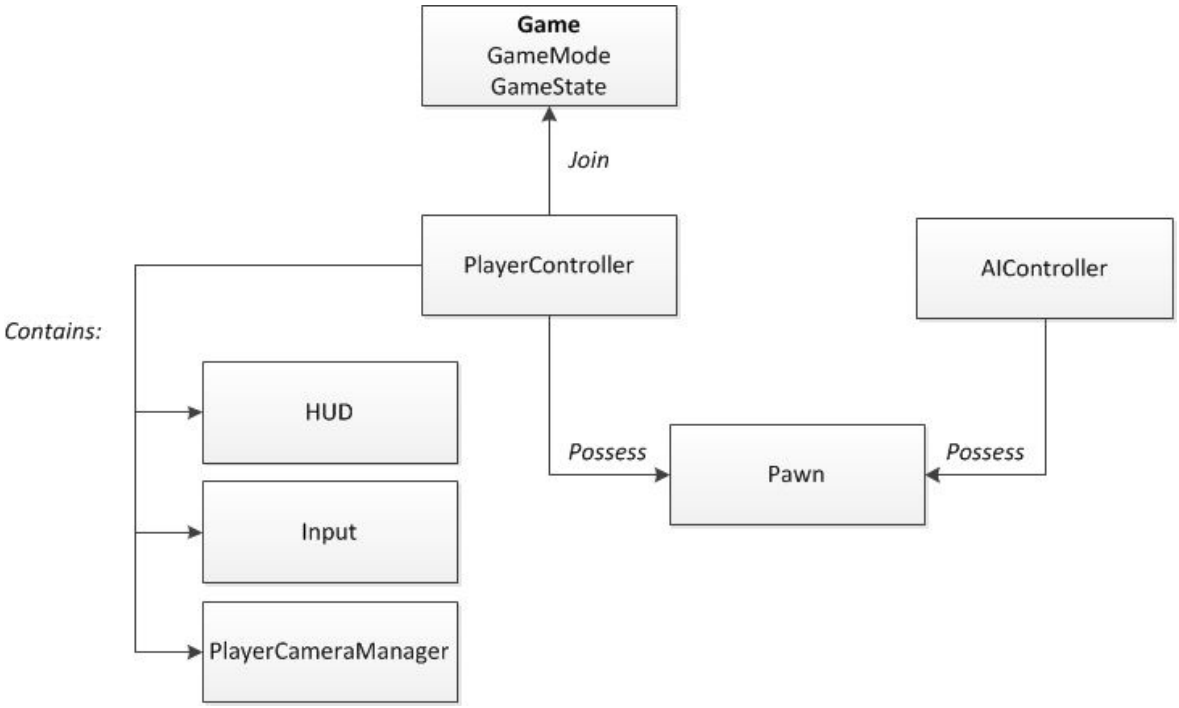
GameInstance: zawiera informacje o danej instancji gry. Istnieje jeden obiekt tej klasy na każdą uruchomioną grę i pozostaje on do dyspozycji aż do jej wyłączenia. Wczytywanie *leveli* nie ma wpływu na *GameInstance*, dzięki czemu klasa ta umożliwia przenoszenie informacji między *levelami*.

Stosowanie się do tej konwencji zaprezentowano w rozdziale 4 skupiającym się na implementacji gry „thesis_1“.

2.5 Porównanie z innymi silnikami

Bezpośrednim konkurentem dla UE4 jest silnik *Unity*. Jego główne różnice w stosunku do UE4 to:

- Pełniejsza i precyzyjniejsza dokumentacja.
- Większa społeczność.
- Lepsze narzędzia do produkcji gier 2D.



Rysunek 2. Diagram architektury rozgrywki

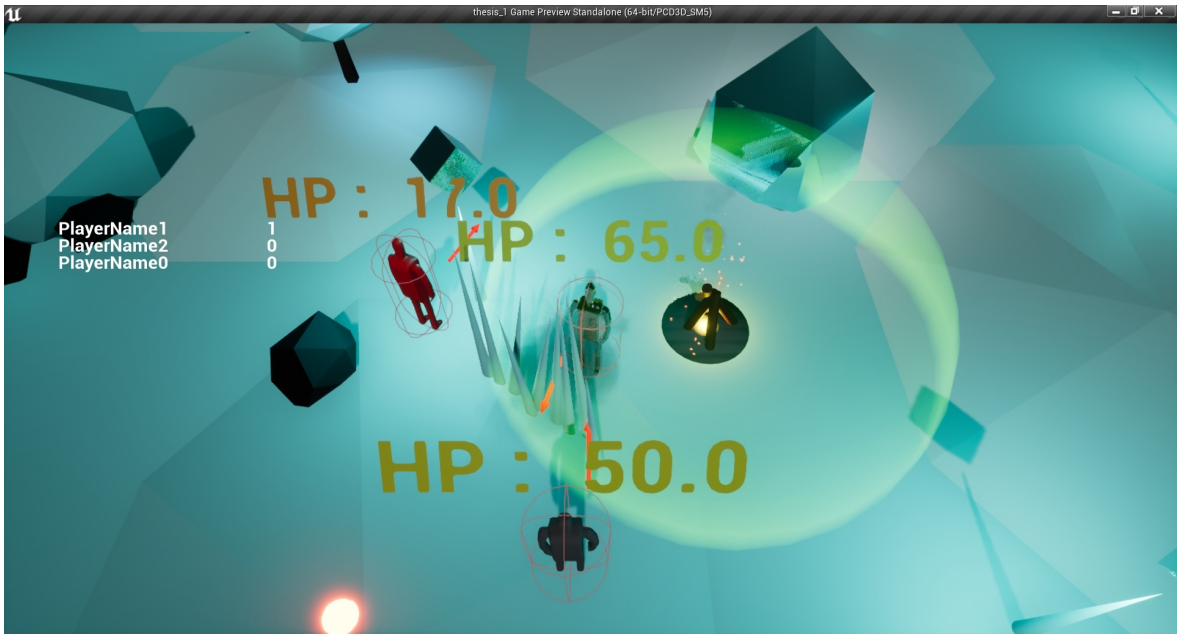
- Bogatszy *AssetStore* czyli serwis, w którym użytkownicy mogą sprzedawać swoje assety (na przykład modele, dźwięki, animacje, gotowe systemy).
- Język programowania *C#*, co jest jednym z powodów niższej wydajności gier utworzonych w *Unity*.
- Brak otwartego źródła.
- Brak wielu funkcji w domyślnej konfiguracji silnika. Funkcje takie jak *visual scripting* czy drzewa behawioralne są stworzone przez społeczność, a za dostęp do nich trzeba zapłacić.
- Rzadziej używany w wysokobudżetowych projektach.

Inną możliwością jest opracowanie własnego silnika od podstaw. W takim przypadku należy zadbać o podstawowe funkcje wymienione w podrozdziale 2.1. Wówczas przydatne są biblioteki *C++*: *SFML* oraz *SDL*. Osiągnięcie efektu porównywalnego do UE4 nawet dla dużego zespołu doświadczonych programistów będzie zadaniem czasochłonnym i kosztownym. Na to rozwiązanie decydują się głównie duże studia deweloperskie, które planują wykorzystać silnik do wielu projektów, a popularne silniki nie spełniają ich wymagań.

3 Przykładowy przebieg rozgrywki w „thesis_1“

1. Gracz pierwszy uruchamia grę po raz pierwszy, wobec czego jest przenoszony do menu opcji w celu wybrania ustawień (rys. 3). Wpisuje swoje imię w odpowiednim polu, wybiera klasę postaci, zdolności, obrazek oraz czy będzie korzystał z klawiatury i myszki czy z kontrolera do gier (pada). Klika przycisk *Accept*, w katalogu gry powstaje plik z zapisem ustawień, gracz przenoszony jest do menu głównego.
2. Gracz drugi uruchamiał już grę wcześniej, wobec czego posiada plik z zapisem. Po uruchomieniu przenoszony jest do menu głównego.
3. Gracz pierwszy w menu głównym klika przycisk *Host game*, a następnie ustala nazwę serwera, maksymalną liczbę graczy i metodę połączenia (Internet lub LAN). Aplikacja tego gracza będzie jednocześnie serwerem i klientem. Gracz zatwierdza przyciskiem *Accept*, po czym wczytuje mu się poziom, otrzymuje kontrolę nad swoją postacią, a serwer jest gotowy do przyjmowania klientów.
4. Aplikacja Gracza drugi będzie klientem. W menu głównym gracz klika przycisk *Find game*, a następnie podaje adres IP Gracza pierwszego i zatwierdza przyciskiem *Connect*. Po chwili łączy się z serwerem i otrzymuje kontrolę nad swoją postacią.
5. Obaj gracze prowadzą rozgrywkę(rys. 4), mogą chodzić, skakać, unikać oraz używać wybranych wcześniej zdolności. Sterowanie zależy od ustawień (mysz i klawiatura lub pad). Atakowanie postaci przeciwnika odbiera jej punkty zdrowia. Po utraceniu wszystkich punktów zdrowia dany gracz zostaje wyłączony z rozgrywki na siedem sekund, a wynik jego rywala jest zwiększany o jeden punkt. Obaj gracze widzą na swoich ekranach tablicę wyników aktualizowaną na bieżąco. Po upływie 7 sekund czasu gracz z powrotem otrzymuje kontrolę nad swoją postacią i gra jest kontynuowana.
6. Gracze kończą rozgrywkę przyciskiem *X* w prawym górnym rogu okna. Gracz o najwyższym wyniku zostaje zwycięzcą.

Rysunek 3. Menu opcji



Rysunek 4. Rozgrywka

4 Opis implementacji gry „thesis_1“

Opis dotyczy projektu dostępnego w publicznym repozytorium pod adresem https://github.com/mszadko/thesis_1.

4.1 Uruchomienie gry i podstawowe funkcje

Wywołanie pliku wykonywalnego z grą (*thesis_1.exe*) powoduje uruchomienie silnika, utworzenie obiektu klasy *GameInfoInstance* (opisanej w podrozdziale 4.1.1), a następnie wczytanie początkowego poziomu *MainMenu*, który wywołuje funkcję *SaveGameCheck* z *GameInfoInstance* sprawdzającą czy istnieje plik z zapisem gry (system opisany w podrozdziale 4.3) i wyświetlający interfejs użytkownika. Na tym etapie grający otrzymuję kontrolę nad dalszym przebiegiem programu. Sprowadza się ona do wywołań funkcji z *GameInfoInstance* poprzez interfejs.

4.1.1 GameInfoInstance

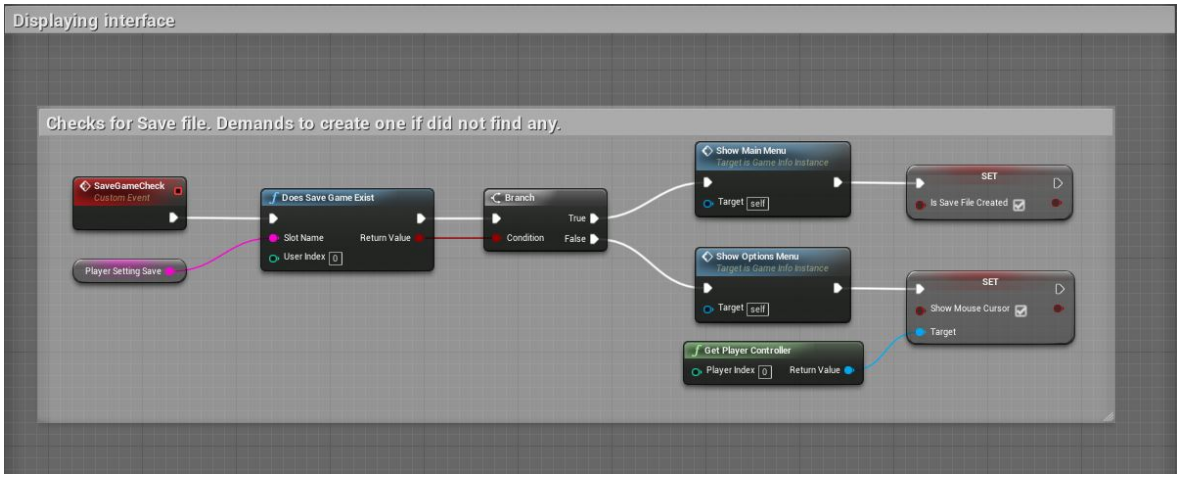
Klasa *GameInfoInstance* dziedziczy po klasie *GameInstance* (opisanej w podrozdziale 2.4), zawiera podstawowe informacje wykorzystywane w innych obszarach gry (na przykład dostępne zdolności i klasy postaci wykorzystywane w systemie opisanym w podrozdziale 4.4.7) oraz odpowiada za wyświetlanie *widgetów* interfejsu (podrozdział 4.2) i tworzenie oraz dołączanie do istniejących rozgrywek sieciowych (podrozdział 4.5). Klasa została zaimplementowana za pomocą blueprintów.

4.2 Interfejs

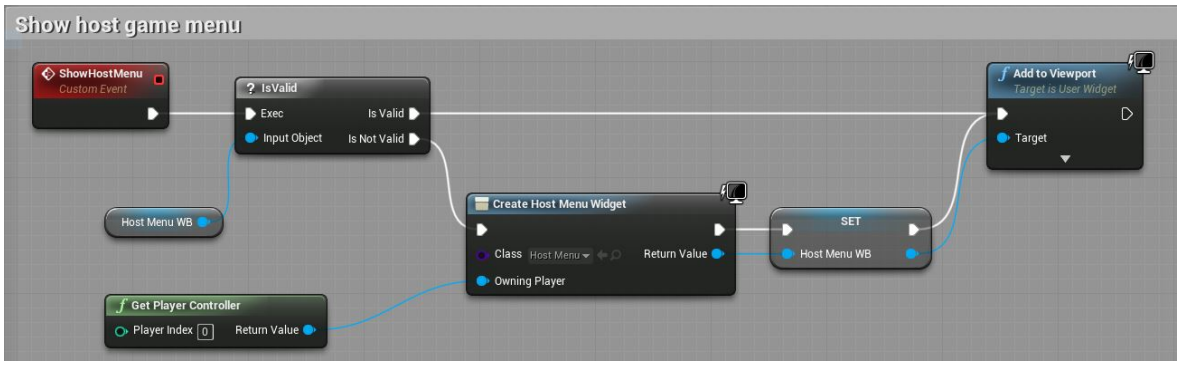
Podstawę interfejsu w UE4 stanowią *widgety* czyli elementy posiadające pewną logikę, które można wyświetlać w oknie użytkownika w warstwie interfejsu. Są to na przykład przyciski, listy, suwaki, pola tekstowe i tym podobne. Za pomocą edytora *widgetów* z podstawowych *widgetów* buduje się bardziej skomplikowane (na przykład menu główne lub tablica wyników), z których konstruuje się docelowy interfejs.

W przypadku gry „thesis_1“ *widgety* wyświetlane są poprzez wywoływanie odpowiednich funkcji z *GameInfoInstance* (rys. 5, 6).

Z menu głównego grający ma możliwość przejścia do menu tworzenia lub dołączania do gry (rozdział 4.5), menu opcji (rozdział 4.4.7) oraz zamknięcia gry. Wszystkie *widgety* zaimplementowane zostały za pomocą blueprintów.



Rysunek 5. Funkcja wywołująca wyświetlenie *widgetu* interfejsu wybranego na podstawie istnienia pliku zapisu stanu



Rysunek 6. Funkcja wyświetlająca *widget* interfejsu hostowania gry

4.3 System zapisu stanu gry

W grze „thesis_1“ zaimplementowano system zapisu gry, który umożliwia grającemu zachowanie swoich ustawień do kolejnego uruchomienia lub przeniesienia ich na inny komputer. System ten korzysta z klasy *PlayerSaveGame*, funkcji *SaveGameToSlot* i *LoadGameFromSlot* oraz struktury *PlayerInfo*. Obiekty klasy *PlayerSaveGame* są serializowane i deserializowane z pliku "thesis_1/Saved/SaveGames/PlayerSettingsSave.sav" oraz posiadają w sobie strukturę *PlayerInfo*, w której przechowywane są ustawienia grającego: imię, obrazek, klasa postaci, rodzaj sterowania oraz zdolności.
Po uruchomieniu gry wywoływana jest funkcja *SaveGameCheck* z klasy *GameInfoInstance*, która jest odpowiedzialna za sprawdzenie, czy istnieje plik "thesis_1/Saved/SaveGames/PlayerSettingsSave.sav". Jeżeli nie istnieje, grający przenoszony jest do menu opcji, w którym wybiera ustawienia i powoduje stworzenie pliku z zapisem. Jeżeli plik ten istnieje, grający przenoszony jest do menu głównego, a dane z pliku wczytywane są przy wejściu do menu opcji (w celu wyświetlenia zapisanych ustawień) lub dołączeniu do rozgrywki (w celu utworzenia postaci oraz zdolności wybranych przez grającego).

4.4 Postacie i zdolności

Gra składa się wielu nieodzownych elementów, jednym z istotniejszych jest niewątpliwie postać którą gracz kontroluje.

4.4.1 Pawn, a PlayerController

Podczas tworzenia postaci pojawia się wiele pytań i problemów do rozwiązania. Jednym z nich jest sposób zbierania danych od gracza. UE4 pozwala na obsługę danych wejściowych w klasach *Pawn* lub w *PlayerController*. Rzeczą wartą rozważenia jest sposób w jaki rozdzielimy obsługę wejścia pomiędzy tymi klasami[7]. W prostych przypadkach możliwa jest obsługa zebranych informacji o woli gracza całkowicie w klasie *Pawn*. Jednak gdy sterowanie staje się skomplikowane, jego obsługa w *PlayerControllerze* jest nieunikniona. *PlayerController* pozwala na przykład na sterowanie wieloma *pawnami* na jednej maszynie czy dynamiczne przełączanie się pomiędzy postaciami. Kolejną zaletą *PlayerControllera* jest to, że jedna jego instancja jest przypisana do gracza przez całą rozgrywkę, a kontrolowane przez niego obiekty klasy *Pawn* mogą się zmieniać. Na przykład podczas odradzania naszej postaci utworzony zostaje nowy obiekt klasy *Pawn*, a kontroler pozostaje ten sam, więc dane, których nie chcemy stracić, takie jak zdobyte punkty czy złoto powinny być składową *PlayerControllera*.

W grze „thesis_1“ jako klasy bazowej do reprezentacji gracza użyto klasy *Character*, a dane wejściowe są całkowicie obsługiwane przez *PlayerController*.

4.4.2 Nazewnictwo klas C++

Utworzone klasy pochodne mają nazwy poprzedzone przedrostkiem *Base*. Konwencja ta odzwierciedla fakt, iż klasy *C++* zaimplementowane przez programistę są następnie wykorzystywane jako klasy bazowe blueprintów postaci. Wszystkie podstawowe umiejętności jak poruszanie się, skakanie czy obracanie się postaci zostały zaimplementowane za pomocą języka *C++*, a następnie odziedziczone w blueprintach. Zyskano dzięki temu warstwę abstrakcji oddzielającą niezbędne linie kodu od dodatkowych zdolności postaci oraz łatwość zmian indywidualnych cech postaci za pomocą edytora blueprintów.

4.4.3 BaseCharacter

W celu rozwinięcia podstawowych możliwości klasy *Character* udostępnionej przez silnik utworzono pochodną klasę *BaseCharacter*. Głównym celem było pozwolenie graczom na dowolne dostosowanie zdolności postaci. Problem ten został rozwiązany za pomocą tablicy obiektów *Skill*, które mają metody *OnPress* oraz *OnRelease*, wykonywane, gdy gracz – odpowiednio – naciska lub zwalnia odpowiedni przycisk. Dodane zostały również zmienne, które są używne przez silnik do odgrywania odpowiednich animacji takich jak animacja uniku czy animacja rzucania czaru oraz funkcje, które zmieniają wartości tych zmiennych na serwerze, gdy gracz na maszynie klienckiej wciśnie odpowiednie przyciski.

4.4.4 BasePlayerController i sterowanie

Klasa *BasePlayerController* jest opracowanym na potrzeby projektu rozszerzeniem klasy *PlayerController*. Jest ona odpowiedzialna między innymi za obsługę danych wejściowych, zebranych od gracza. Gracz ma do wyboru dwa sposoby kontrolowania swojej postaci. W zależności od ustawień w odpowiednim oknie interfejsu użytkownika gracz może korzystać z klawiatury oraz myszy lub z kontrolera gier. Sterowanie odbywa się za pomocą:

Klawiatury i myszy Postać można poruszać za pomocą klawiszy W, A, S oraz D i obrócona jest zawsze w kierunku kursora myszy. Klawisz F odpowiada za unik, a *spacja* za skok. Zdolności uruchamiane są przyciskami 1, 2, 3 i 4.

Kontrolera Xbox One Drążki kontrolera odpowiadają za ruch i kierunek postaci. Przycisk A odpowiada za unik, a B za skok. Zdolności uruchamiane są przyciskami LT, LB, RT i RB.

Uruchamianie zdolności działa dwuetapowo. Podczas wciśnięcia klawisza odgrywana jest animacja oraz wykonywany jest kod metody *OnPress* z klasy *Skill*, który może wykonać całą logikę zdolności lub przygotować naszą postać do momentu gdy gracz zwolni przycisk. Następnie wykona się metoda *OnRelease*, która może pozostać pusta, dokończyć wykonanie zdolności lub przywrócić gracza do stanu sprzed kliknięcia przycisku.

4.4.5 BaseCharacterMovementComponent

CharacterMovementComponent jest komponentem klasy *Character*, odpowiedzialnym za przemieszczanie się postaci, który automatycznie obsługuje komunikację sieciową. Komponent ten umożliwia między innymi predykcję, replikację i korekcję ruchu. Działa on w następujący sposób [8]. Co klatkę wywoływana jest metoda *TickComponent*, gdzie obliczane są zmiany przyspieszenia oraz rotacji postaci. Następnie, w zależności od tego, czy program wykonywany jest na kliencie czy na serwerze, wywoływana jest metoda *PerformMovement* lub *ReplicateMoveToServer*. *ReplicateMoveToServer* dodaje ruch do listy oczekujących ruchów, następnie wywołuje metodę *PerformMovement* i replikuje ruch na serwer za pomocą zdalnie wywoływanej metody *ServerMove*. *ServerMove* przesuwa postać na serwerze w odpowiednie miejsce, a następnie oblicza dystans pomiędzy pozycją postaci na serwerze i na maszynie klienckiej. Jeżeli dystans jest większy niż dopuszczalny, serwer zdalnie wywołuje na kliencie metodę *ClientAdjustPosition*, która przesuwa postać w odpowiednie miejsce. Jeżeli dane korekcyjne dotrą do klienta, a jego pozycja zostanie poprawiona, metoda *ClientAdjustPosition* wykona ponownie wszystkie ruchy, które zostały dodane na listę ruchów oczekujących po ruchu, który został poprawiony przez serwer.

Dane o aktorze, takie, jak jego pozycja, prędkość czy rotacja, są replikowane na inne maszyny za pomocą standardowego mechanizmu replikacji. Oznacza to, że maszyny nie dostają potrzebnych im informacji co klatkę, co twórcy silnika rozwiązali za pomocą systemów predykcji i wygładzania. Systemy te wypełniają luki w danych spowodowane rozbieżnością między częstotliwościami renderowania klatek i replikowania danych. Predykcja polega na przesuwaniu obiektu klasy *Character* zgodnie z ostatnio otrzymanymi danymi na temat ruchu postaci. Gdy predykcja odbiega od rzeczywistych zmian, które zaszły na innych maszynach, UE4 nakłada na pozycję stopniowe korekty w celu uniknięcia nagłej jej zmiany.

Z powodu tych systemów wszystkie zdolności postaci, modyfikujące jej położenie, powinny zostać zaimplementowane poprzez rozszerzenie klasy *CharacterMovementComponent*. W celu dodania możliwości uniku, polegającego na gwałtownym odskoku postaci w danym kierunku, opracowano klasę *BaseCharacterMovementComponent*. Implementacja polegała na nadpisaniu metod klasy bazowej [14]. TickComponent oprócz swojego pierwotnego zadania sprawdza czy gracz wciska przycisk odpowiedzialny za unik. W takim przypadku dodawane są symulowane dane wejściowe, które naśladują poruszanie się gracza w odpowiednim kierunku. W celu zapewnienia szybszego poruszania się postaci podczas uniku, nadpisano metody *GetMaxSpeed* oraz *GetMaxAcceleration* tak, aby w zależności od zmiennej wskazującej na to czy postać wykonuje unik, zwracały różne wartości. Ponadto potrzebne było rozszerzenie klas używanych do predykcji i zapisywania ruchu na listę oczekujących w taki sposób, by ewentualna poprawa ruchu spowodowana opóźnieniami transmisji danych uwzględniła kliknięcie przez gracza przycisku uniku.

4.4.6 Skill

Jedną z głównych cech gry „thesis_1“ jest możliwość wyboru czterech z wielu dostępnych zdolności do używania w trakcie rozgrywki. W tym celu powstała klasa *Skill* reprezentująca zdolność. Klasa ta powstała za pomocą kodu *C++* z myślą o rozszerzaniu jej blueprintami, co umożliwiają znaczniki *Blueprintable* oraz *BlueprintNativeEvent* użyte w nagłówku klasy. Każda klasa dziedzicząca po *Skill* reprezentuje jedną zdolność i nadpisuje bazowe funkcje *OnPress* oraz *OnRelease* logiką danej zdolności. Decyduje też o tym, które klasy postaci mogą używać danej zdolności.

4.4.7 Wybór postaci i zdolności

Grający ma możliwość wyboru postaci oraz zdolności w menu opcji, które pobiera informacje o klasach postaci i zdolnościach z *GameInfoInstance*. Aby zapobiec sytuacji, w której wybrana została zdolność przeznaczona dla innej postaci niż obecnie wybrana, interfejs aktualizuje listę dostępnych zdolności przy każdej zmianie klasy postaci. Po zaakceptowaniu zmian tablica zdolności trafia do *PlayerInfo* oraz pliku z zapisem gry. Przy dołączeniu do rozgrywki informacje te trafiają do *PlayerControllera* za pomocą funkcji *LoadPlayerInfo*, a stamtąd do *Charactera* poprzez funkcję *ABaseCharacter::LoadSkills()* przy każdym jego odrodzeniu.

4.5 Networking – tworzenie oraz dołączanie do rozgrywki sieciowej

„thesis_1“ jest grą sieciową i do kompletnej rozgrywki wymaga przynajmniej dwóch komputerów, z których jeden pełni rolę serwera, a pozostałe są klientami. W UE4 serwer może działać w jednym z dwóch trybów sieci [9]:

NM_DedicatedServer: wymaga stworzenia oddzielnego oprogramowania serwera, które pozbawione jest między innymi elementów związanych z grafiką, dźwiękami czy wejściem. Dzięki temu więcej zasobów komputera wykorzystywanych jest na obliczenia związane z obsługą podłączonych klientów. Gracze do podłączenia i prowadzenia rozgrywki korzystają z oprogramowania klienckiego. Nie jest możliwe granie z poziomu serwera. To podejście stosowane jest w przypadku gier ze skomplikowaną, wymagającą komunikacją sieciową.

NM_ListenServer: zarówno serwer i klienci korzystają z dokładnie tego samego oprogramowania, a komputer będący serwerem jest jednocześnie klientem. Wydajność jest niższa niż w przypadku serwera dedykowanego w zamian za możliwość prowadzenia rozgrywki z poziomu serwera.

W „thesis_1“ wykorzystano tryb *NM_ListenServer* ze względu na zapewnianą przez niego łatwość rozpoczęcia rozgrywki i niskie wymagania sieciowe gry. W implementacji sie-

Rysunek 7. Blueprint, w którym logika zależy od roli

ciowych aspektów rozgrywki wykorzystywano głównie zdalne wywoływanie procedur, co wymaga sprecyzowania, czy mają one być wykonywane na serwerze czy na kliencie, sprawdzania roli instancji gry to znaczy czy obecnie wykonujemy kod na serwerze czy na kliencie (rys. 7) oraz oznaczania *actorów* do replikacji czyli przesyłania informacji o nich między serwerem, a klientami.

Utworzenie serwera odbywa się poprzez menu *Host game*, a jego widoczność zależy od konfiguracji sieciowej gracza i jego dostawcy Internetu. Gra „thesis_1“ korzysta z portu 7777 (domyślnego w UE4).

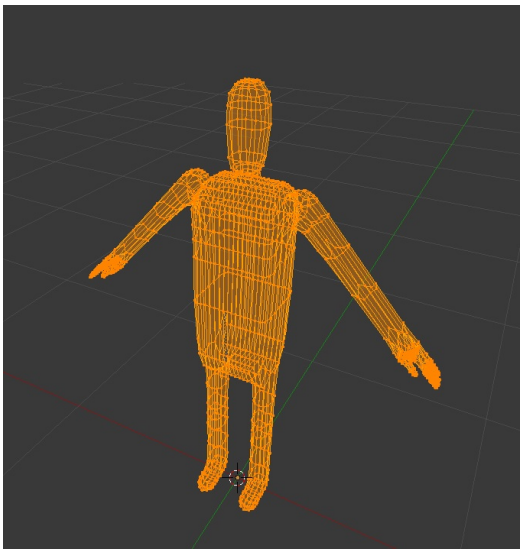
Łączenie z serwerem inicjowane jest w menu *Find game* poprzez automatyczne wyszukanie gry w sieci lokalnej lub podanie adresu IP serwera w Internecie lub sieci lokalnej. Następnie na serwerze wykonywana jest funkcja *OnPostLogin* (rys. 7), która inicjuje wczytanie danych z pliku łączącego się gracza, pobiera je od niego i na ich podstawie przydziela mu postać.

4.6 Modele i Animacje

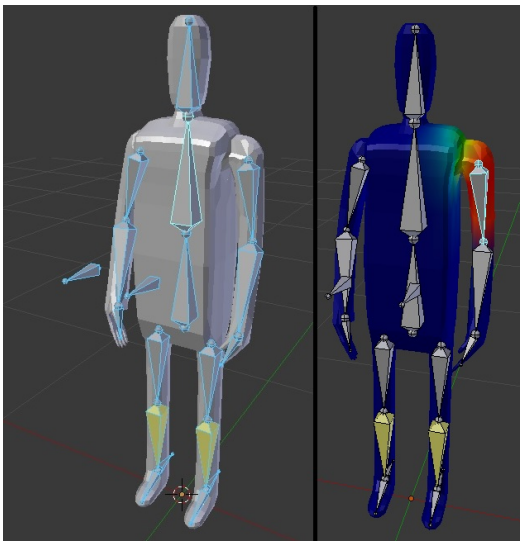
Wartym zaznaczenia jest fakt, że głównym celem tworzenia modeli i animacji było zapoznanie się z narzędziami oraz samym procesem tworzenia tej części gier komputerowych. Podczas pracy nad projektem skupiono uwagę na aspektach technicznych. Znaczną część elementów *levelu* udało się stworzyć za pomocą obracania i skalowania gotowych kształtów udostępnionych w UE4. Bardziej skomplikowane kształty wymagały wymodelowania w osobnym oprogramowaniu dedykowanym do tego celu. Do stworzenia modeli i animacji użytych w projekcie wykorzystano oprogramowanie Blender w wersji 2.79.

4.6.1 Modelowanie

Modelowanie w Blenderze polega na przemieszczaniu odpowiednich wierzchołków podstawowych kształtów, takich jak sfera czy sześcian, aż do momentu otrzymania docelowego obiektu. Animowanie tak wymodelowanej siatki obiektu (rys. 8) jest praktycznie niemożliwe.



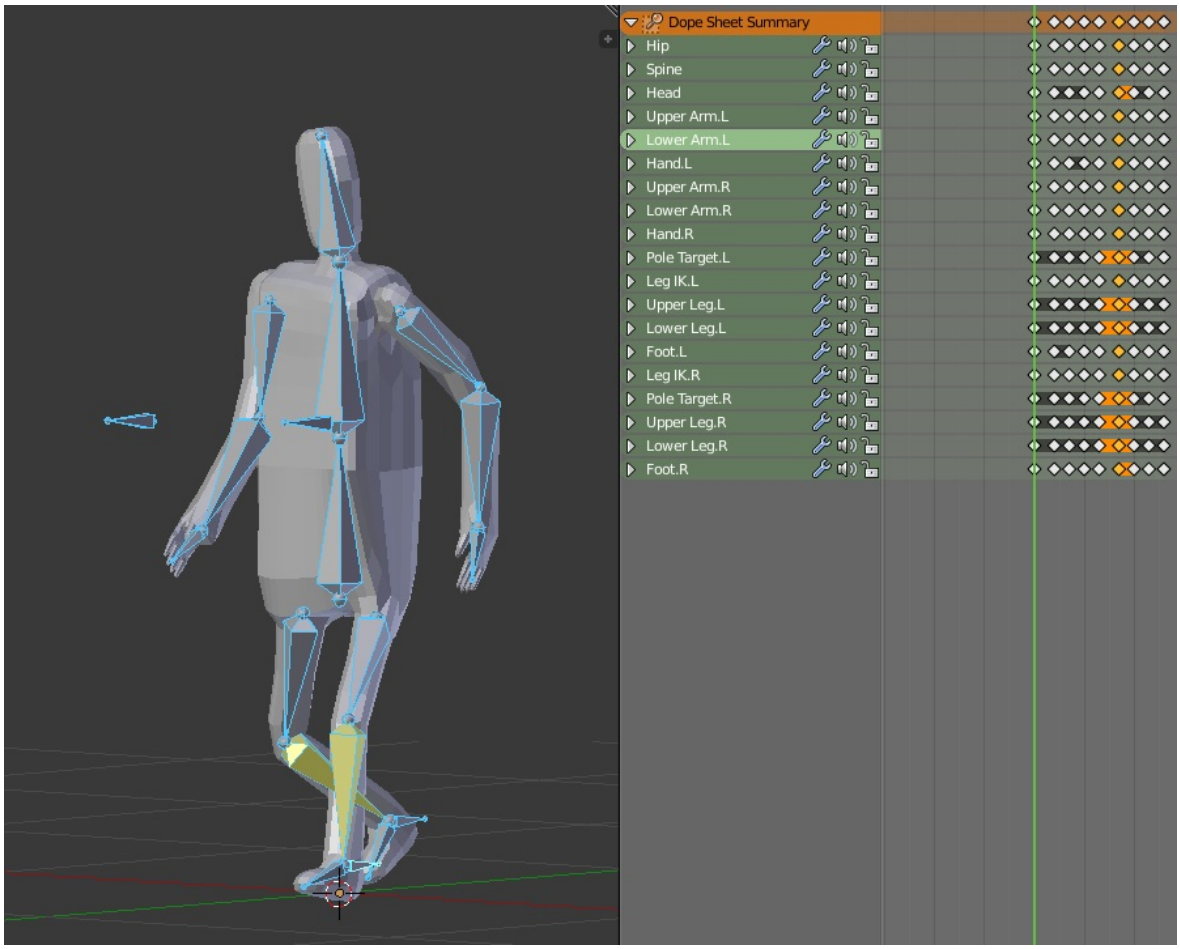
Rysunek 8. Model postaci wykorzystany w grze



Rysunek 9. Kości oraz stawy użyte w modelu (po lewej) oraz mapa wag dla jednej z kości (po prawej)

4.6.2 Rigging

Następnym krokiem jest proces nazwany *rigging*[19]. *Rigging* polega na wyposażeniu modelu w kości, stawy oraz stworzeniu mapy wag informującej o tym jak dane kości mają wpływać na kształt modelu (rys. 9). Umożliwia to animowanie obiektu.



Rysunek 10. Jedna z klatek animacji chodzenia

4.6.3 Animacje

Proces tworzenia animacji polega na ustawianiu kości w odpowiednich pozach i zapisywaniu ich zmian w rotacji oraz położeniu (rys. 10). W animacjach nawet podstawowych ruchów pojawia się wiele problemów do rozwiązania. Przykładem może być animacja skoku, w której problematyczny jest zmienny czas przebywania postaci w powietrzu. Animacja zeskakiwania z wysokiej przeszkody powinna trwać dłużej niż animacja wskakiwania na przeszkodę. Problem ten został rozwiązany za pomocą rozbicia sekwencji skoku na 3 oddzielne animacje – wybicie się, przebywanie w powietrzu oraz lądowanie. Ostatnia klatka wybicia jest jednocześnie pierwszą i ostatnią klatką przebywania w powietrzu oraz pierwszą klatką lądowania. Pozwoliło to na zapętlone odgrywanie animacji przebywania w powietrzu z płynnymi przejściami pomiędzy wybiciem, a lądowaniem.

4.7 Udźwiękowienie

Odtwarzanie dźwięków w grze obsłużono dwoma funkcjami. W przypadku muzyki i dźwięku towarzyszącego zdobyciu punktu jest to funkcja *PlaySound2D*, która odtwarza dźwięk u tych klientów, u których została wywołana. Dźwięki związane ze zdolnościami i ogniskiem

odtwarzane są za pomocą funkcji *SpawnSoundAtLocation*, dzięki czemu słyszą je jedynie gracze będący blisko źródła dźwięku.

Użyto następujących dźwięków pochodzących ze strony internetowej freesound.org:

- „06673 electronic flipper bonus.wav“ autorstwa Robinhood76 (licencja CC BY-NC)
- „avl03.wav“ autorstwa blaukreuz (licencja CC 0)
- „shatter a wine glass“ autorstwa edhutschek (licencja CC 0)
- „jumping“ autorstwa fins (licencja CC 0)

5 Dalszy rozwój gry

W kolejnych podrozdziałach zaprezentowano możliwości dalszego rozwoju gry. Poprzez implementację nowych klas postaci, zdolności oraz postaci sterowanych przez komputer można urozmaicić rozgrywkę, a nawet całkowicie zmienić jej charakter z rywalizacji między graczami na współpracę.

5.1 Klasy postaci i zdolności

Klasy *Skill* oraz *BaseCharacter* zostały opracowane z myślą o ułatwieniu dalszego rozwoju gry. Utworzenie nowej postaci, różniącej się od dotychczasowych na przykład liczbą punktów zdrowia czy prędkością, sprowadza się do odziedziczenia klasy bazowej i zmiany wartości odpowiednich zmiennych. Utworzenie zdolności różniącej się na przykład liczbą zadawanych obrażeń czy czasem, który należy odczekać przed ponownym użyciem zdolności, to również kwestia zmiany pojedyńczych zmiennych. W przypadku implementacji zdolności o zupełnie nowej logice za wzór służą zdolności dostępne w finalnej wersji gry.

Nowo opracowane postaci i zdolności muszą zostać umieszczone w – odpowiednio – słowniku *CharacterClasses* lub tablicy *Skills* w blueprincie *GameInfoInstance*, aby były widoczne w grze.

5.2 Sztuczna inteligencja

Możliwe jest również wykorzystanie dostępnego w UE4 mechanizmu drzew behawioralnych w celu zaimplementowania postaci sterowanych przez komputer. Sprowadza się to do utworzenia następujących elementów [12]:

- Nowa postać dziedzicząca po *BaseCharacter*
- *NavMesh* określający, w które miejsca postać może się przemieścić
- Drzewo behawioralne zwracające konkretne zachowanie na podstawie danych wejściowych
- *Blackboard* będący zbiorem danych dla drzewa behawioralnego
- *AIController* będący odpowiednikiem *PlayerControllera* i spinający wyżej wymienione elementy. *AIController* decyduje o wykorzystywanym *Blackboardzie* i drzewie behawioralnym oraz inicjuje dane zachowanie w postaci.

6 Napotkane problemy

Niejednokrotnie trudności sprawiała niedokładna dokumentacja silnika lub nawet jej brak. W takich sytuacjach pomocne okazywały się rady społeczności skupionej wokół UE4 oraz otwarte źródło silnika.

Problem stanowiło również korzystanie z systemu kontroli wersji Git, ponieważ nie jest on przystosowany do obsługi plików innych niż tekstowe. Skutkiem jest utrudnione rozwiązywanie ewentualnych konfliktów powstałych w plikach związanych na przykład z blueprintami czy modelami. Problem ten będzie dotkliwszy dla większych zespołów.

7 Podsumowanie i wnioski

Tworzenie gier nie należy do zadań łatwych oraz szybkich. Podczas pracy nad projektem pojawiło się wiele dodatkowych problemów, przez co lista założonych na początku opcji gry musiała zostać skrócona. Jednym z nich był wysoki próg wejścia UE4. Przed rozpoczęciem pracy nad docelowym projektem, stworzonych zostało kilka mniejszych gier wzorowanych na przykładowych projektach z dokumentacji[5], które miały na celu zaznajomienie autorów z konwencjami, możliwościami silnika oraz systemem sieciowym. Czas tworzenia wydłużało dodatkowo scalanie podsystemów utworzonych przez poszczególnych autorów gry. Gry wideo są skomplikowanymi programami komputerowymi z duża liczbą połączeń pomiędzy danymi elementami gry. System zapisu jest na przykład zależny od wybranych opcji, który z koleji jest powiązany z klasą postaci. Ze względu na dużą liczbę powiązań pomiędzy różnymi elementami projektu, nawet najmniejsze zmiany mogły spowodować powstawanie różnorakich błędów, przez co dużą część czasu poświecono na proces testowania każdego nowo dodanego elementu.

Silnik UE4 umożliwił stworzenie prostej sieciowej gry w stosunkowo niedługim czasie. Gra była zaprojektowana z myślą o jej dalszym rozwoju co wraz ze znajomością silnika UE4 pozwala na sprawne dodawanie kolejnych elementów rozgrywki. Osiągniety efekt nie byłby możliwy bez wykorzystania profesjonalnych narzędzi takich jak UE4 czy Blender.

8 Bibliografia

[1] Joanna Lee, *Learning Unreal Engine Game Development*, Packt Publishing, 2016

[2] Mark DeLoura, *Perełki Programowania Gier, Vademecum profesjonalisty, Tom I, strony 17-21*. Helion, 2002

[3] Stephen Prata, *Język C++, Szkoła programowania, wydanie VI, strony 21-37*. Helion, 2013

[4] *Engine Features*. `https://docs.unrealengine.com/latest/INT/Engine/index.html` (dostęp 30.12.2017)

[5] *UE4 Documentation Tutorials*. `https://docs.unrealengine.com/latest/INT/Videos/` (dostęp 14.01.2018)

[6] *UE4 Documentation Blueprints Visual Scripting*. `https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html` (dostęp 30.12.2017)

[7] *UE4 Documentation Player Controller*. `https://docs.unrealengine.com/latest/INT/Gameplay/Framework/Controller/PlayerController/` (dostęp 3.01.2018)

[8] *UE4 Documentation Character Movement Component*. `https://docs.unrealengine.com/latest/INT/Gameplay/Networking/CharacterMovementComponent/index.html` (dostęp 3.01.2018)

[9] *UE4 Documentation Networking Overview*. `https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Overview/` (dostęp 05.01.2018)

[10] *UE4 Documentation Performance and Profiling*. `https://docs.unrealengine.com/latest/INT/Engine/Performance/` (dostęp 3.01.2018)

[11] *UE4 Documentation Gameplay Framework Quick Reference*. `https://docs.unrealengine.com/latest/INT/Gameplay/Framework/QuickReference/index.html` (dostęp 31.12.2017)

[12] *UE4 Documentation Behavior Tree Quick Start Guide*. `https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/QuickStart/index.html` (dostęp 14.01.2018)

[13] *„Game engine“, Wikipedia*. `https://en.wikipedia.org/wiki/Game_engine` (dostęp 29.12.2017)

[14] *UE4 Wiki - Authoritative Networked Character Movement*. https://wiki.unrealengine.com/Authoritative_Networked_Character_Movement (dostęp 3.01.2018)

[15] *Everything you ever wanted to know about replication (but were afraid to ask)*. https://wiki.beyondunreal.com/Everything_you_ever_wanted_to_know_about_replication_%28but_were_afraid_to_ask%29#Function_call_replication_-_Sending_messages_between_server_and_client (dostęp 30.12.2017)

[16] Trevir Nath, *Investing in Video Games: This Industry Pulls In More Revenue Than Movies, Music*. http://www.nasdaq.com/article/investing-in-video-games-this-industry-pulls-in-more-revenue-than-movies-music-cm634585 (dostęp 29.12.2017)

[17] *The Human Race – An Inside Look at the Technology Behind the Groundbreaking Real-Time Film from Epic Games, The Mill and Chevrolet*. https://www.unrealengine.com/en-US/showcase/the-human-race-an-inside-look-at-the-technology-behind-the-groundbreaking-real-time-film-from-epic-games-the-mill-and-chevrolet (dostęp 30.12.2017)

[18] *Virtual Reality- Into the magic*. http://www.ikea.com/ms/en_US/this-is-ikea/ikea-highlights/Virtual-reality/index.html (dostęp 30.12.2017)

[19] Justin Slick, *What is Rigging? Preparing a 3D Model For Animation* . https://www.lifewire.com/what-is-rigging-2095 (dostęp 3.01.2018)

Wyrażam zgodę na udostępnienie mojej pracy w czytelniach Biblioteki SGGW w tym w Archiwum Prac Dyplomowych SGGW.

. . . . . .

(czytelny podpis autora pracy)