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 2017

Praca przygotowana zespołowo przez:

1. Maciej Wygoda
172407
który jest autorem:
ktore rodzialy + strony

2. Marcin Szadkowski
wpisz swoj numer albumu
który jest autorem:
ktore rodzialy + strony

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, gamedev, 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 najlepiej świadczy fakt, iż pod względem wygenerowanych przychodów prześcignęła już branże filmową oraz muzyczną [7]. 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. W związku z tym, że stworzenie gry 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. Ponadto motywację stanowiły chęć zgłębienia technologii UE4, podjęcia technicznego wyzwania, jakie stawia zaprogramowanie gry wideo oraz pasja do gier. 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.

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 aczkolwiek obecnie najpopularniejsze silniki posiadają 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. Każdy popularny silnik dystrybuowany 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][5]

2.2 Funkcje silnika Unreal Engine 4

UE4 swoją popularność zawdzięcza między innymi otwartemu źródłu, co w pewnym stopniu umożliwia producentom gier dostosowanie silnika do własnych potrzeb na przykład poprzez programowanie narzędzi dla mniej technicznych członków zespołu czy modyfikacje w działaniu silnika. 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. [8][9]

Oprócz podstawowych funkcji takich jak renderowanie grafiki czy obsługa sterowania do dyspozycji oddane zostały między innymi [2]:

Symulacja fizyki: UE4 korzysta z silnika fizyki *PhysX 3.3* dzięki czemu wiarygodnie symuluje kolizje obiektów i inne oddziaływania fizyczne. Producenci mają również możliwość modyfikowania panujących zasad celem lepszego przedstawienia własnej wizji.

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.

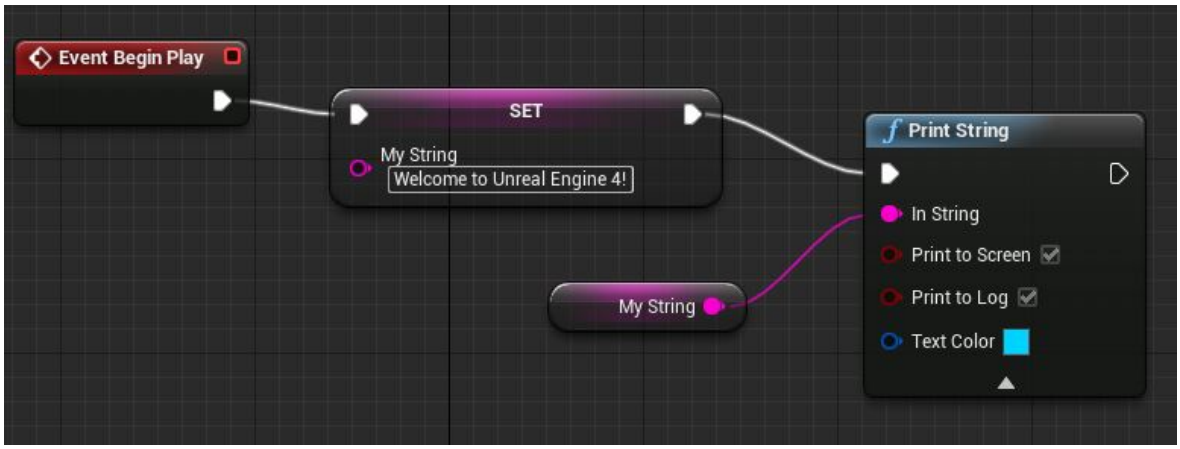
Sequencer: Jest to narzędzie służące do produkcji przerywników filmowych. Jego obsługa przypomina pracę z oprogramowaniem do montażu filmów i modelowania 3D. Edytor ten uwzględnia elementy takie jak oś czasu, ujęcia kamery, szkielety i animacje obiektów.

Networking: UE4 powstał z myślą o rozgrywce sieciowej, wobec tego dużo uwagi poświęcono stworzeniu 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 opierają się o aktualizacje właściwości obiektów i zdalne wywołania procedur *(ang. remote procedure calls - RPC)*. W obu przypadkach wykorzystywany jest protokół *UDP*, który jest zawodny, ale generuje o wiele mniejszy narzut na sieć niż w przypadku *TCP*. UE4 ma zaimplementowany własny system zapewniający częściową niezawodność komunikacji. [6]

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, 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. Do pewnego stopnia można o nim myśleć jak o farbie. Należy jednak uwzględnić, że materiał poza kolorem czy teksturą definiuje również 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 graficznego programowania rozgrywki oparty o węzły reprezentujące elementy takie jak funkcje, klasy czy zmienne, które po połączeniu stanowią pewną logikę (rys. 1). Za pomocą blueprintów można (często w krótszym czasie) osiągnąć podobny efekt co za pomocą kodu przy czym dla osób mniej technicznych są one o wiele prostsze w użyciu. Ponadto mogą dziedziczyć klasy napisane kodem, a programiści mają możliwość rozwijania blueprintów poprzez programowanie kolejnych węzłów do użycia przez resztę zespołu. Jest możliwe napisanie kompletnej gry bez nawet jednej linii kodu, a jedynie z użyciem blueprintów. Nie oznacza to jednak, że kod stał się bezużyteczny. W każdym wypadku kod *C++* jest wydajniejszy od blueprintów (to znaczy szybciej się wykonuje, co bezpośrednio wpływa na liczbę generowanych klatek na sekundę), w wielu przypadkach jest czytelniejszy (na przykład obliczenia w pętli) i łatwiejszy w utrzymaniu (na przykład kontrola wersji przy pracy zespołowej).[3]



Rysunek 1. Przykład blueprinta

Oba te podejścia są wykorzystywane w profesjonalnym środowisku, a kluczem do sukcesu jest ich umiejętne połączenie, co zaprezentowano w dalszej części tej pracy.

2.3 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).[4]

Actor: Jest to bazowa klasa dla każego obiektu, który można umieścić w scenie. Obiekty te nie muszą mieć fizycznej reprezentacji. Często zawierają dodatkowe komponenty (*ActorComponents*) określające między innymi w jaki sposób obiekt się porusza czy jak jest renderowany. Oprócz tego *aktor* posiada obsługę replikacji właściwości i wywołań funkcji przez sieć.

Pawn: Jest to *aktor*, 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 wykorzystywany przy animacjach 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 dla gracza na przykład jego obecny wynik, stan zdrowia sterowanej przez niego postaci czy tak zwana minimapa.

Camera: decyduje o perspektywie, z której grający obserwuje scenę.
Jest elementem PlayerControllera.

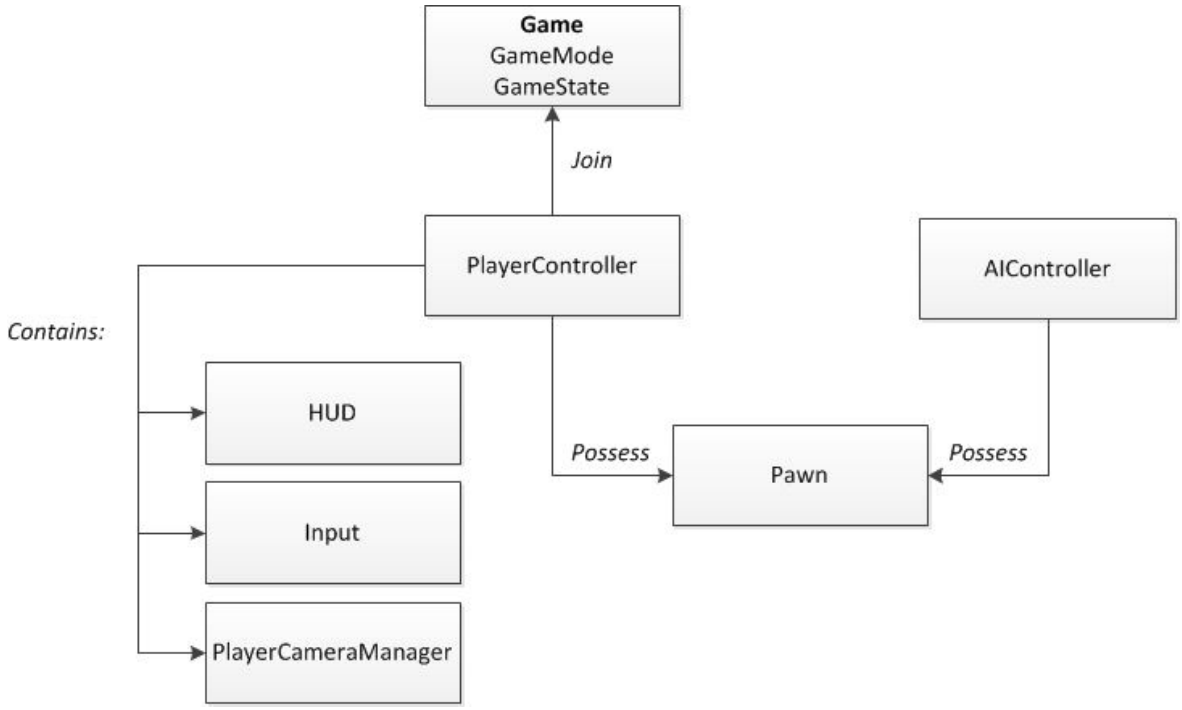
GameMode: zawiera informacje takie jak zasady gry czy warunki zwycięstwa. Nie powinien zawierać żadnych informacji potrzebnych klientom, ponieważ istnieje jedynie na serwerze. Decyduje również o tym, który *GameState* i *PlayerState* zostanie wykorzystany. Wybór *GameMode-a* zależy od wczytywanego poziomu.

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 poziomów nie ma wpływu na *GameInstance*, dzięki czemu klasa ta umożliwia przenoszenie informacji między poziomami.

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



Rysunek 2. Diagram architektury rozgrywki

3 Przykładowy przebieg rozgrywki

1. Gracz 1 uruchamia grę po raz pierwszy, wobec czego jest przenoszony do menu opcji w celu wybrania ustawień. 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 2 uruchamiał już grę wcześniej, wobec czego posiada plik z zapisem. Po uruchomieniu przenoszony jest prosto do menu głównego.

3. Gracz 1 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 2 będzie klientem. W menu głównym gracz klika przycisk *Find game*, a następnie podaje adres IP Gracza 1 i zatwierdza przyciskiem *Connect*. Po chwili łączy się z serwerem i otrzymuje kontrolę nad swoją postacią.

5. Obaj gracze prowadzą rozgrywkę, 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.

4 Opis implementacji gry „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.

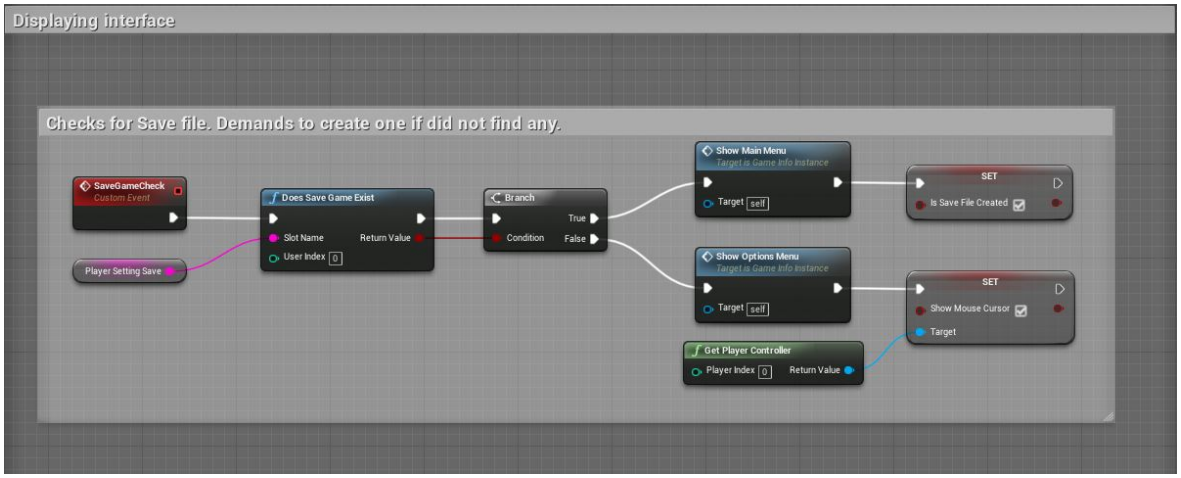
4.1.1 GameInfoInstance

Klasa *GameInfoInstance* dziedziczy po klasie *GameInstance* (opisanej w podrozdziale 2.3), zawiera podstawowe informacje wykorzystywane w innych obszarach gry (na przykład dostępne zdolności i klasy postaci wykorzystywane w systemie opisanym w podrozdziale 4.4.3) oraz odpowiada za wyświetlanie *widgetów* interfejsu (podrozdział 4.2) i tworzenie oraz dołączanie do istniejących rozgrywek sieciowych (podrozdział 4.5).

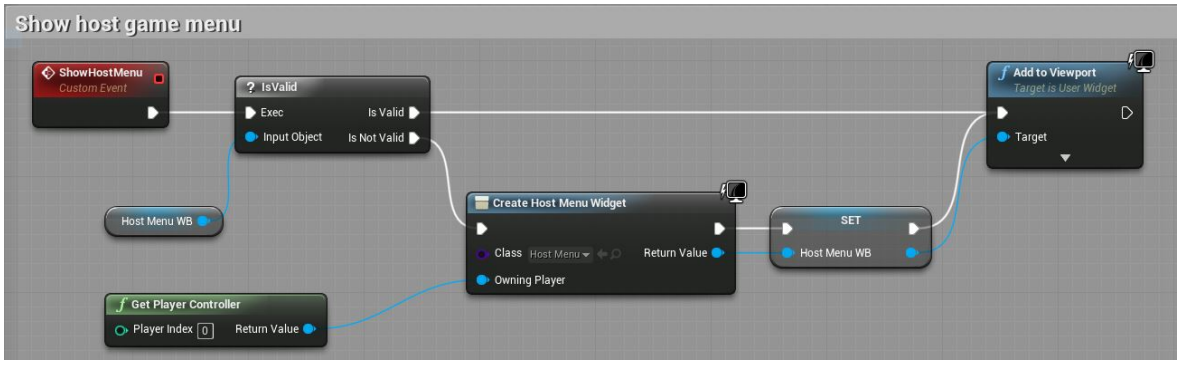
4.2 Interfejs

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. 3, 4).
Z menu głównego grający ma możliwość przejścia do menu tworzenia lub dołączania do gry (rozdział 4.5), menu opcji (rozdział 4.4.3) oraz zamknięcia gry.



Rysunek 3. Funkcja wywołująca wyświetlenie *widgetu* interfejsu wybranego na podstawie istnienia pliku zapisu stanu



Rysunek 4. 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 zgodnych z wyborami grającego).

4.4 Postacie i zdolności

4.4.1 BaseCharacter

4.4.2 Skill

4.4.3 Wybór postaci i zdolności

4.5 Networking – tworzenie oraz dołączanie do rozgrywki sieciowej

4.6 Animacje

5 Kolejny rozdział

6 Bibliografia

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

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

[3] *Blueprints Visual Scripting*, https://docs.unrealengine.com/latest/INT/Engine/Blueprints/index.html (dostęp 30.12.2017)

[4] *Gameplay Framework Quick Reference*, https://docs.unrealengine.com/latest/INT/Gameplay/Framework/QuickReference/index.html (dostęp 31.12.2017)

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

[6] *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)

[7] 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)

[8] *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)

[9] *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)

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

. . . . . .
(czytelny podpis autora pracy)