



POLSKO-JAPONSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra Sieci Komputerowych

Sieci Urządzeń Mobilnych

Kamil Warpechowski

Nr albumu 10709

Praca inżynierska

Promotor:

dr inż. Michał Tomaszewski

Platforma do tworzenia gier oraz animacji interaktywnych w
środowisku rozszerzonej rzeczywistości

Spis treści

1 Wprowadzenie	4
1.1 Rozszerzona rzeczywistość	6
2 Cel pracy	7
3 Implementacje	8
3.1 Implementacja w technologii hologramu	9
3.2 Implementacja w technologii mappingu 3d	11
4 Wykorzystane technologie	13
4.1 Unity	13
4.1.1 Dlaczego Unity	13
4.1.2 Alternatywne rozwiązania	13
4.1.3 Wybór języka programowania	13
4.1.4 Unity IDE	14
4.1.5 Licencja i koszty	14
4.2 Android	14
4.3 Komunikacja sieciowa	15
4.3.1 UNET	15
4.3.2 HTTP (SOAP, REST)	15
4.3.3 WebSocket	16
5 Aplikacja główna	17
5.1 Świat gry	17
5.2 Aktorzy	17
5.3 Prefabrykaty	21
5.3.1 Kamery	21
5.3.2 Światło	23
5.4 SocketIO	23
5.4.1 Network	24
5.4.2 Budowanie zapytania	26
5.4.3 Replikator	26
5.5 Serwer komunikacyjny	29

6 Aplikacja mobilna - kontroler	31
6.1 Przyciski	32
6.2 Użycie stylu	34
6.3 Komunikacja sieciowa	36
7 Środowisko uruchomieniowe	38
7.1 Aplikacja główna - Unity	38
7.2 Serwer komunikatów	39
7.3 Aplikacja mobilna - kontroler	39
8 Inne implementacje - Google Cardboard	40
8.1 ArToolkit	41
8.2 Markery	41
8.3 Przykład implementacji	43
8.4 Napotkane problemy i ograniczenia	44
9 Inne implementacje - Project Tango	45
9.1 Wady i zalety	46
10 Wnioski i dalszy rozwój	47

1 Wprowadzenie

Ludzie od lat przyzwyczaili się korzystać z elektroniki oraz internetu na standardowych urządzeniach elektronicznych. Na początku lat dziewięćdziesiątych do domów zaczęły trafiać komputery stacjonarne. Najpierw z modemami DSL¹, a następnie ze stałymi łączami światłowodowymi. Na przestrzeni lat korzystanie z ekranu w połączeniu z klawiaturą i myszą stało się dla ludzi naturalne.

Przez ostatnią dekadę na rynku pojawiły się interfejsy dotykowe. Popularność smartfonów, a następnie tabletów oraz urządzeń typu Wearables² spowodowało, że coraz bardziej sporadycznie korzystamy z standardowej fizycznej klawiatury.

Ekrany dotykowe pojawiły się nie tylko na urządzeniach telekomunikacyjnych, lecz także jako monitory w komputerach pokładowych samochodów oraz instalowane są w zagłówkach w samolotach jako multimedialne centrum rozrywki ³.

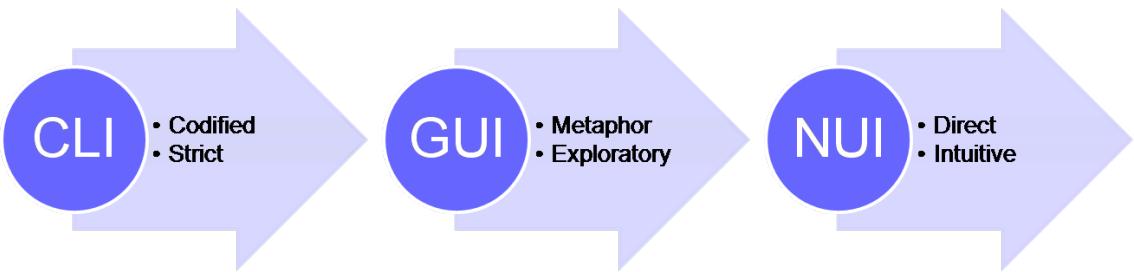
Przez ostatnie kilka lat narasta trend poszukiwania innych metod dostępu do danych, zwłaszcza multimedialnych. Obecnie wiele przedsiębiorstw prowadzi badania nad nowymi, bardziej naturalnymi dla ludzi interfejsami, które nie wymagałyby użycia standardowych (sztucznych) urządzeń wejścia typu klawiatura czy myszka komputerowa.

Na przestrzeni lat interfejsy urządzeń elektronicznych ewoluowały pierwotnie z aplikacji sterowanych za pomocą wiersza polecań poprzez programy z graficznym interfejsem użytkownika (Graphic User Interface), aż do obecnie coraz bardziej popularnej i rozwijanej grupy interfejsów „naturalnych” (Natural User Interface).

¹Digital Subscriber Line – technologia cyfrowego szerokopasmowego dostępu do internetu.[

²<https://pl.wikipedia.org/wiki/Wearables>

³<http://www.komputerswiat.pl/nowosci/wydarzenia/2012/28/boeing-z-androidem-napokladzie.aspx>



Rysunek 1: Ewolucja interfejsów użytkownika

źródło: https://en.wikipedia.org/wiki/Natural_user_interface

Wiele nowych urządzeń próbuje implementować sterowanie interfejsem użytkownika za pomocą gestów (np. Microsoft Kinect⁴ lub sensor Leap Motion⁵), czy też za pomocą myśli (np. Emotiv⁶). Na rynku widać duże zainteresowanie nową formą kontroli urządzeniami, zwłaszcza tymi bardziej naturalnymi dla człowieka. Jednakże obecnie są to głównie eksperymenty nowej technologii. Nie ma na rynku obecnie wypracowanego popularnego standardu dostępu w grupie NUI.

W branży filmowej oraz gier wideo narasta trend używania nowych technologii do rozszerzania doznań jakie otrzymuje odbiorca. W kinach odbywają się coraz częściej projekcje filmów stworzonych w technologii trójwymiarowej. Natomiast w ostatnim czasie pojawiają się sale kinowe pozwalające na projekcję filmów trójwymiarowych wraz z dodatkowymi elementami takimi jak: drganie foteli, wiatr, dym, woda⁷. Jednakże w obecnej chwili taki format rozrywki jest dość drogi, gdyż wymaga szczególnie przygotowanej sali kinowej oraz okularów, które pozwalają tworzyć iluzję przestrzenną.

⁴<http://www.xbox.com/pl-PL/xbox-one/accessories/kinect-for-xbox-one>

⁵<https://www.leapmotion.com/>

⁶<http://emotiv.com>

⁷<http://cinema-city.pl/4dx-info>

1.1 Rozszerzona rzeczywistość

Coraz bardziej popularne staje się pojęcie rozszerzonej rzeczywistości (ang. augmented reality). Jest to zbiór różnych technologii pozwalającej łączyć świat rzeczywisty z wirtualnym. Jest to jeszcze mało popularny sposób interakcji, lecz w ostatnim dziesięcioleciu rozwój (zarówno urządzeń jak i specjalistycznego oprogramowania) jest bardzo dynamiczny.

Pierwsze próby w tej dziedzinie odbywały się jeszcze w latach sześćdziesiątych amerykański naukowiec oraz artysta Myron Krueger prowadził badania nad wirtualną oraz rozszerzoną rzeczywistością. Jest on twórcą pojęcia środowiska responsywnego. „Jest to środowisko w którym działania użytkownika i odpowiada na nie w sposób przemyślany poprzez złożony system środków wizualnych i akustycznych, oraz dostosowuje się do powstałych w ten sposób nowych warunków środowiska.”⁸. Stworzył on interaktywne instalacje takie jak Glowflow⁹, Metaplay¹⁰ oraz Videoplac¹¹.



⁸<http://www.techsty.art.pl/hipertekst/cyberprzestrzen/krueger.htm>

⁹<http://dada.compart-bremen.de/item/artwork/1347>

¹⁰<http://dada.compart-bremen.de/item/artwork/1348>

¹¹<http://dada.compart-bremen.de/item/artwork/1346>

Rysunek 2: Wizualizacja Microsoft HoloLens

źródło: <https://www.microsoft.com/microsoft-hololens/en-us/why-hololens>

2 Cel pracy

Celem niniejszej pracy jest stworzenie platformy do tworzenia gier oraz interaktywnych animacji prezentowanej za pomocą rozszerzonej rzeczywistości sterowanej za pomocą zdalnego kontrolera. Praca zawiera przykłady różnych implementacji w technologiach z rodziny AR.

Przykłady zastosowanie zestawu aplikacji:

- Prezentacje przestrzeni architektonicznych
- Rozrywka (np. gry)
- Reklama między innymi w miejscach użyteczności publicznych (np. centra handlowe)

3 Implementacje

Rozdział prezentuje różne podejścia do stworzenia implementacji przykładowej sceny w technologii rozszerzonej rzeczywistości. Jako środowisko badawcze wybrano salę - labolatorium znajdującą się w Polsko-Japońskiej Akademii Technik Komputerowych. Salę tę wybrano, ponieważ znajduje się w podpiwniczeniu, co za tym idzie ilość światła dziennego jest znikoma. Pozwala to na bardzo łatwe stosowanie urządzeń projekcyjnych w ciągu dnia. Dodatkowo w sali znajduje się rura ciepłownicza poprowadzona po dwóch ścianach. Umiejscowienie tego elementu pozwala go wykorzystać do stworzenia wirtualnej sceny (np. animacja fal wody w środku rury).



Rysunek 3: Labolatorium PJATK

Główym założeniem było stworzenie minigry opartej na grze z początku lat dwudziestolecia minionego o nazwie Lemmingi¹². Pierwotnie gra została stworzona w 1991 roku na platformę Amiga.

Celem gry jest doprowadzenie grupy aktorów (tytułowych Lemmingów) do wyjścia (mety). Aktorzy automatycznie idą w jedną stronę. Każdemu z nich można włączyć jedną lub więcej umiejętności (np. umiejętność kopania, swobodnego spadania - spadochron). Aktorzy generowani są automatycznie w określonej sekwencji czasu.

¹²<https://en.wikipedia.org/wiki/Lemmings>

3.1 Implementacja w technologii hologramu

Pierwotnym założeniem było stworzenie projektu za pomocą hologramu. Planowano wykorzystanie złudzenia optycznego stworzonego za pomocą światła na pół-przezroczystej płaszczyźnie znajdującej się przed oczami odbiorcy. Analogiczną konsepcję prezentowało w przeszłości urządzenie Google Glass, a obecnie np. Microsoft Hololens lub Meta2.

Jednakże zamiast urządzenia nakładanego na głowę planowano użyć przezroczystą szybę znajdującą się na drzwiach wejściowych do laboratorium. Dzięki takiemu rozwiązaniu odbiór instalacji odbywałby się bez dodatkowych urządzeń, co za tym idzie interakcja byłaby bardziej naturalna. Na szybie umieszczona zostałaby warstwa półprzezroczystej folii na której prezentowany byłby obraz za pomocą projektora multimedialnego ustawionego pod kątem około 30 stopni w góre w kierunku szyby. Projektor znajdowałby się na statywie w środku sali - technologia projekcja tylnej.

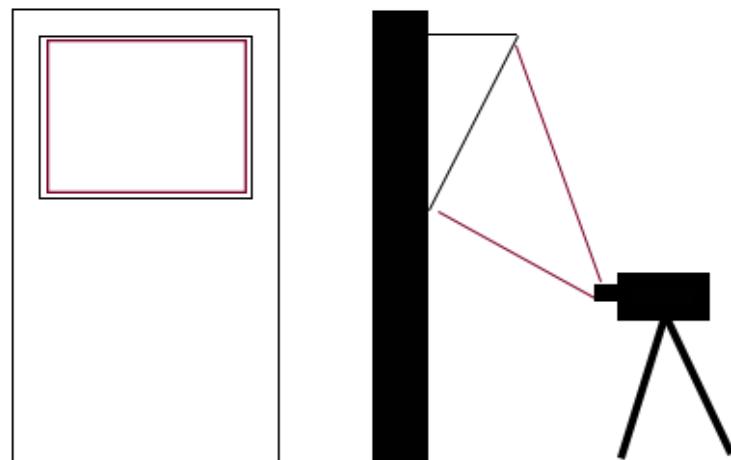
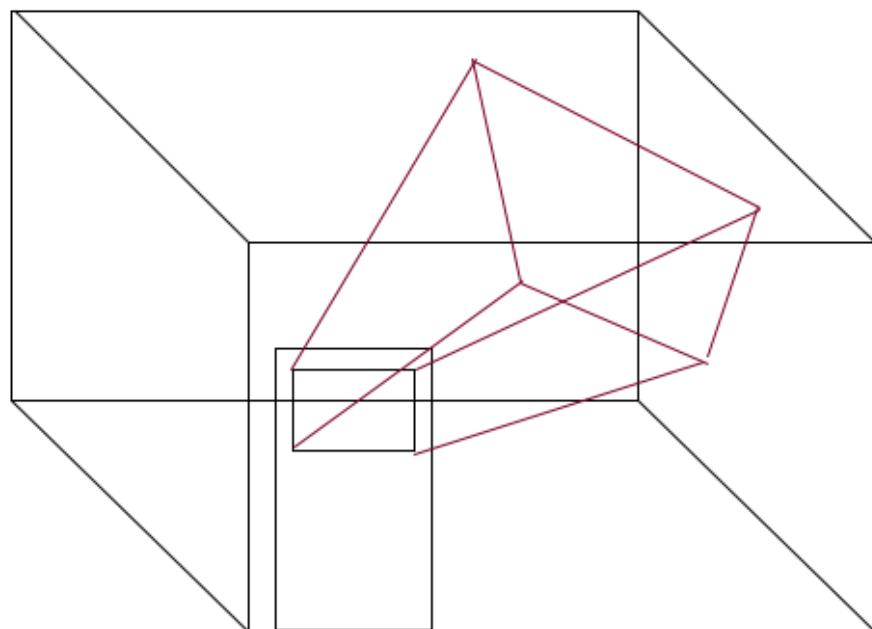
Podczas eksperymentów próbowało wiele różnych folii oraz kilka ustawień projektorów.

Zaobserwowano:

- Umiejscowienie projektora pod kątem względem szyby powoduje bardziej naturalny obraz. Nie widać wtedy głównego strumienia światła z projektora (strumień światła nie jest prowadzony w linii prostej do odbiorcy), co za tym idzie nie ma odczucia oślepienia.
- Użycie specjalnej folii do projekcji tylnej powoduje najlepszy odbiór. Inne folie przepuszczają zbyt małą ilość światła, bądź obraz z projektora był mało ostry.
- Użycie projektora w technologii LED okazało się lepsze, niż zastosowanie standardowego projektora multimedialnego. Mała odległość pomiędzy drzwiami, a urządzeniem pozwalała na użycie projektora z małą ilością lumenów.

Jednakże zrezygnowano z powyższego pomysłu, ponieważ napotkano problem nakładania obrazu z elementami znajdującymi się w sali. Aby punkty wirtualne z ich rzeczywistymi odpowiednikami się nakładały i tworzyły spójny obraz (augmented reality) należało spoglądać przez szybę pod jednym wskazanym kątem. Co za

tym idzie prawidłowy odbiór instalacji byłyby zaburzony przez takie czynniki jak np. wzrost odbiorcy, kierunek wzroku czy też odległość od szyby. Urządzenia takie jak Microsoft Hololens niwelują ten problem poprzez stałe umiejscowienie półprzezroczystego ekranu w małej odległości od oka.



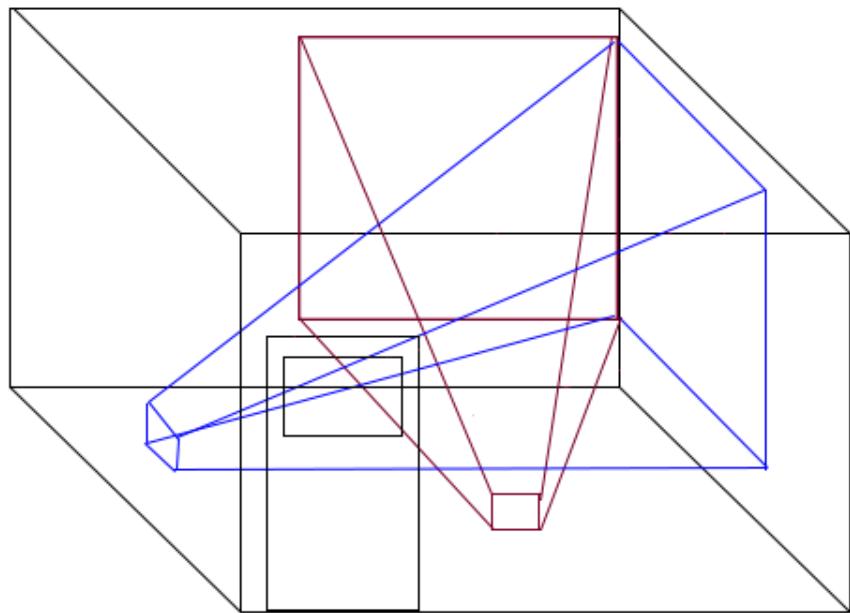
Rysunek 4: Poglądowy schemat działania hologramu

3.2 Implementacja w technologii mappingu 3d

Kolejnym pomysłem było zastosowanie video mappingu 3d. Jest to technologia często spotykana w branży rozrywkowej (jako tło sceny koncertowej) oraz do prezentowania przestrzeni architektonicznych (zarówno wewnętrznych jak i zewnętrznych), czy też pojazdów jak i innych mniejszych przedmiotów.

Technologia polega na oświetlaniu rzeczywistego elementu źródłem światła z projektora multimedialnego. Najlepsze efekty można uzyskać w zaciemnionych pomieszczeniach oraz przy lampach z dużą ilością lumenów. Dzięki temu za pomocą kolorów można pokazywać lub uniewidoczyć elementy przy wykorzystaniu koloru czarnego. Podczas projekcji ciemnego koloru ilość światła z projektora jest bardzo znikoma, co za tym idzie powstaje złudzenie, że element oświetlony światłem czarnym jest niewidoczny. Przy realizacji większości instalacji takiego typu stosowany jest wcześniej przygotowany obraz wideo. Instalacje nie są interaktywne. Jednakże dzięki temu nawet zaawansowane animacje trójwymiarowe obciążają sprzęt komputerowy tylko podczas renderowania (zamiany obiektów trójwymiarowych na strumień video). Wprowadzenie elementów interaktywnych (np. sterowanie grą) mocno obciąża kartę graficzną komputera.

Założono, że projekcja odbywać będzie się na dwóch prostopadłych do siebie ścianach. Takie podejście wymaga dwóch projektorów. Jednakże obraz z nich musi być synchronizowany, ponieważ elementy interaktywne (np. postacie) będą przemieszczały się z jednej ściany na drugą. Zastosowanie dwóch komputerów wymagałoby stworzenia protokołu komunikacyjnego. Założono, iż zastosowany będzie jeden komputer z wydajną kartą graficzną, która obsłuży dwa źródła wyjścia.



Rysunek 5: Poglądowy schemat działania - mapping

źródło: własne

Zaobserwowano:

- Efekty wizualne są odpowiednie, jednakże zauważalny jest brak głębi obrazu.
Wszystkie elementy są dwuwymiarowe.
- Bardzo ważnym aspektem jest jakoś lampy projektora i jego umiejscowienie.
Nawet drobne przesunięcie projektora w stosunku do ściany może zaburzyć odbiór instalacji.

4 Wykorzystane technologie

4.1 Unity

Unity jest obecnie najpopularniejszą platformą do tworzenia gier (zarówno trójwymiarowych jak i dwuwymiarowych) na wiele platform sprzętowych. Silnik ten korzysta z API Direct3D (na urządzeniach Windows) oraz OpenGL.

4.1.1 Dlaczego Unity

Najnowsza wersja posiada natywne wsparcie do rozszerzonej oraz wirtualnej rzeczywistości. Dodatkowo większość projektów budowanych w oparciu o ideologię AR dostarcza Software Development Kit właśnie dla Unit, co za tym idzie możliwe jest łatwe porównanie różnych implementacji. Narzędzie te posiada prosty, ergonomiczny interfejs co ułatwia pracę.

Bardzo pomocnym dodatkiem do narzędzia jest „Assets Store”. Jest to wirtualny sklep z komponentami do tworzenia gry. W projekcie zastosowałem tekstury i obiekty 3D pochodzące z tego źródła.

Dodatkowo silnik ten wspiera import modeli trójwymiarowych w bardzo dużej ilości specjalistycznych rozszerzeń plików.

4.1.2 Alternatywne rozwiązania

	Unity	Unreal Engine
Wsparcie języków programowania	C#, JavaScript, Boo	c++
Obsługa wielu ekranów	Tak	Nie
Wsparcie dla Google Cardboard	Tak	Nie
Popularność/społeczność	duża	znikoma

Tablica 1: Porównanie silników gier

4.1.3 Wybór języka programowania

Środowisko Unity wspiera obsługę skryptów (animacje oraz logika biznesowa) w kilku językach programowania: C#, UnityScript (zmodyfikowana wersja JavaScript) oraz w przeszłości Boo. Podjęto decyzję, by w projekcie użyć język C#, gdyż jest

najbardziej stabilny, posiada najbardziej rozbudowaną dokumentację oraz jest to najpopularniejszy język w specjalistycznej literaturze. Dodatkowym udogodnieniem jest to, iż język posiada wiele wbudowanych klas (np. do obsługi połączeń TCP) oraz niezliczoną ilość zewnętrznych bibliotek.

4.1.4 Unity IDE

Środowisko Unity jest multiplatformowe. Aplikacje można używać na dowolnym systemie operacyjnym. Jednakże edycja skryptów odbywa się za pomocą zewnętrznego narzędzia. W systemie Mac OS X jest to MonoDevelop, natomiast w systemie Windows jest to VisualStudio w wersji Community. Opisywana aplikacja początkowo była tworzona na systemie Mac OS X, jednakże kołopody ze środowiskiem MonoDevelop spowodowały decyzje o przeniesieniu środowiska na system Windows. Subiektywnie mogę stwierdzić, że stabilność oraz komfort pracy jest dużo lepszy w systemie Windows. Dodatkową alternatywą dla MonoDevelop może okazać się Visual Studio Code. Jest to prosty multiplatformowy edytor posiadający obsługę języka C# .

4.1.5 Licencja i koszty

Unity jest zamkniętym, licencjonowanym oprogramowaniem. Darmowa wersja (Personal Edition) pozwala na nielimitowane użycie, jednakże jest to okrojona edycja. Szersze informacje o ograniczeniach wersji Personal zawarte są w kolejnych rozdziałach. Licencja pozwala na komercyjne użycie przy limicie zarobków na poziomie stu tysięcy dolarów. Komercyjna wersja (Professional Edition) jest płatna w modelu subskrypcyjnym (75 dolarów za miesiąc)¹³. Na potrzeby opisywanego projektu zastosowano Unity w wersji Personal Edition.

4.2 Android

Naturalnym wyborem technologii przy tworzeniu aplikacji na urządzenie sterujące byłoby Unity, gdyż te środowisko pozwala na komplikacje kodu na urządzenia mobilne(systemy: iOS, Android, Windows Phone, Tizen itp¹⁴). Jednakże Unity w

¹³<https://store.unity3d.com/subscribe>

¹⁴<https://unity3d.com/unity/multiplatform>

wersji Personal Edition nie pozwala na uruchomienie warstwy sieciowej na urządzeniach mobilnych.

Na potrzeby implementacji przykładowego urządzenia sterującego wybrano platformę Android, gdyż ma ona największy udział w rynku.¹⁵ Proces tworzenia aplikacji na tą platformę przebiega w języku Java.

4.3 Komunikacja sieciowa

Największym wyzwaniem było stworzenie dwukierunkowego protokołu komunikacyjnego pomiędzy serwerem (aplikacja napisana w środowisku Unity) oraz dowolnym kontrolerem lub w przyszłości innym urządzeniem wysyłającym dane do aplikacji. Podstawowym założeniem było to iż, kontrolerem gry może być standardowy telefon komórkowy. Dodatkowo w przyszłości planowana jest rozbudowa o zdalne sterowanie za pomocą przeglądarki internetowej. Pierwotnie rozważane było użycie Bluetooth, jednak ograniczyłoby to zdalne sterowanie. Podjęto decyzję projektową o użyciu połączenia sieciowego. Rozważano następujące protokoły:

4.3.1 UNET

Unity wspiera natywną obsługę multiplayer - UNET, jednakże jest to zamknięty protokół. Komunikacja możliwa jest tylko pomiędzy aplikacjami stworzonymi w tym środowisku.

4.3.2 HTTP (SOAP, REST)

Komunikacja za pomocą HTTP (protokoły komunikacyjne takie jak np. SOAP, REST) są bardzo często spotykane. Jest to standard aplikacji internetowych. HTTP nadaje się do przesyłu dużych wolumenów danych, jednak niezbyt dobrze sprawdza się przy małych, lecz częstych połączeniach pomiędzy klientem, a serwerem. Duży narzut czasowy może spowodować transformacje danych do oraz z formatu JSON lub XML. Jednakże dużą zaletą wspomnianych protokołów jest prostota implementacji w większości języków programowania, gdyż są już gotowe komponenty.

¹⁵<https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10>

4.3.3 WebSocket

Zdecydowano, że najlepszym rozwiązaniem będzie użycie protokołu komunikacyjnego TCP, a dokładniej technologii WebSocket, która pozwala na dwustronną komunikację na zasadzie klient-serwer. Jest to protokół połączeniowy. W celu ułatwienia implementacji wybrano bibliotekę socket.io¹⁶. Główną jej zaletą jest możliwość nasłuchiwanego i emitowania danych do wszystkich odbiorców z poszczególnej grupy. Pozwala to na stworzenie platformy, która będzie mogła w przyszłości obsługiwać wiele instancji gry lub kilka kontrolerów jednocześnie. Dodatkowym udogodnieniem jest to, iż popularność tej biblioteki sprawiła, że posiada ona wiele implementacji w różnych językach programowania. Jako Middleware - pośrednik pomiędzy grą, a kontrolerem wybrano technologię Node.js. Zbudowano lekki szkielet, obsługujący połączenia wielokierunkowe. Dzięki wydzieleniu Middleware żadna z instancji gry lub kontrolera nie pełni roli serwera co powoduje wzrost wydajności danej platformy. Dodatkowo w przyszłości można rozbudować serwer o możliwość przechowywania zapisanego stanu gry lub np. ilości punktów zebranych przez użytkownika.

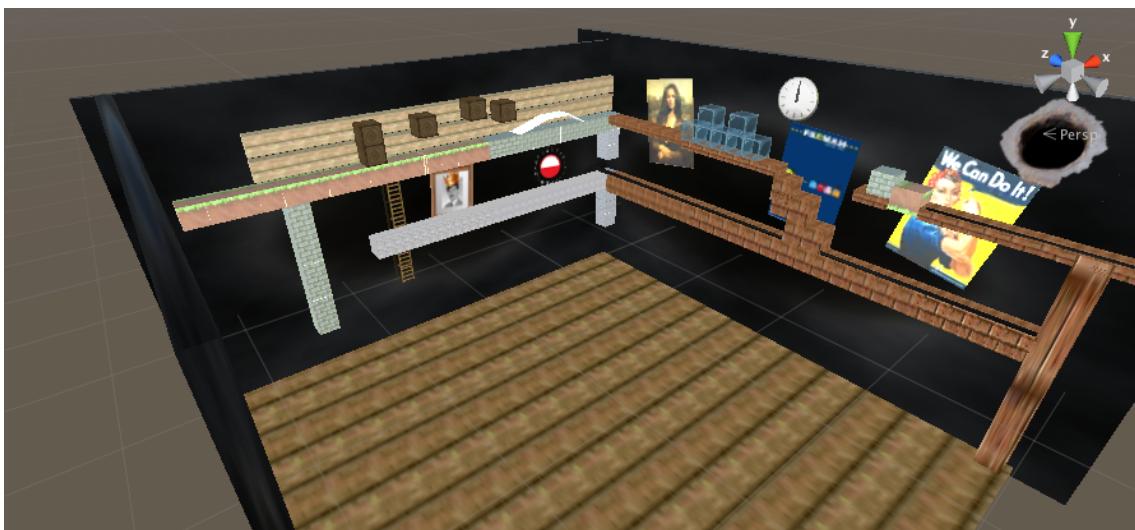
¹⁶<http://socket.io/>

5 Aplikacja główna

Głównym założeniem aplikacji w Unity jest stworzenie wirtualnego świata składającego się z wizualizacji dwóch ścian oraz elementów znajdujących się na nich. Po wyznaczonych elementach poruszać się będą aktorzy, czyli postacie gry.

5.1 Świat gry

Na wizualizacji dwóch ścian znajdują się przeszkody, czyli wirtualne platformy zbudowane z prostopadłościanów. Część z nich nakłada się na rzeczywiste elementy - na przykład rura ciepłownicza. Na platformach się elementy aktywne, które reagują na interakcje z aktorem. Aby pokonać przeszkodę, będzie musiał on wykonać jedną z dostępnych w danej chwili akcji (np. rozbicie szklanego elementu czy też wykopanie).



Rysunek 6: Model w środowisku Unity

źródło: własne

5.2 Aktorzy

Aktor to element interaktywny w grze. Podczas jej inicjalizacji aktor (prefabrykat z gotowym trójwymiarowym modelem) zaczyna się poruszać w jednym kierunku. Zadaniem gracza (za pomocą kontrolera) jest nadanie jednej z wielu umiejętności, tak aby aktor mógł przejść z miejsca startu do końca planszy.

Zaimplementowane możliwości aktora to:

- wykopanie elementu

- spadochron - swobodne spadanie
- rozbicie za pomocą kilofa elementu
- umiejętność skakania
- umiejętność zatrzymania się i ponownego swobodnego przechodzenia
- reset - powrót do pozycji startowej

```
public static Vector3 lemmingPosition = new Vector3  

(1205.9F, 103.4F, -4996.9F);  

private static Vector3 jumpSize = new Vector3 (0, 200F, 0F);
```

Listing 1: Konfiguracja pozycji początkowej i siły skoku

```
public void SendInfo () {  

    Network . SendMessage ( "hasax_ "+this . hasAx) ;  

    Network . SendMessage ( "hassh_ "+this . hasSh) ;  

    Network . SendMessage ( "hasdrabina_ "+this . drabina) ;  

    Network . SendMessage ( "isMove_ "+this . isMove) ;  

}
```

Listing 2: Przykład wysyłania komunikatów do serwera

Podczas tworzenia nowej instancji klasy Aktora zapisywana ona jest do listy statycznej wszystkich dostępnych na daną chwilę obiektów (podczas usunięcia z planszy element również znika z listy). Ma to na celu możliwość implementacji w kontrolerze możliwości wyboru aktywnego aktora.

```
public static void GetNext () {  

    Debug . Log ( "get next" );  

if (Lemming2 . activeEl == null) {  

        Lemming2 . activeEl = Lemming2 . lista [ 0 ];  

} else {
```

```

    int i = Lemming2.lista.IndexOf(Lemming2.activeEl);
    if (i < Lemming2.lista.Count - 1) {
        Lemming2.activeEl = Lemming2.lista [i + 1];
    } else {
        Lemming2.activeEl = Lemming2.lista [0];
    }
}

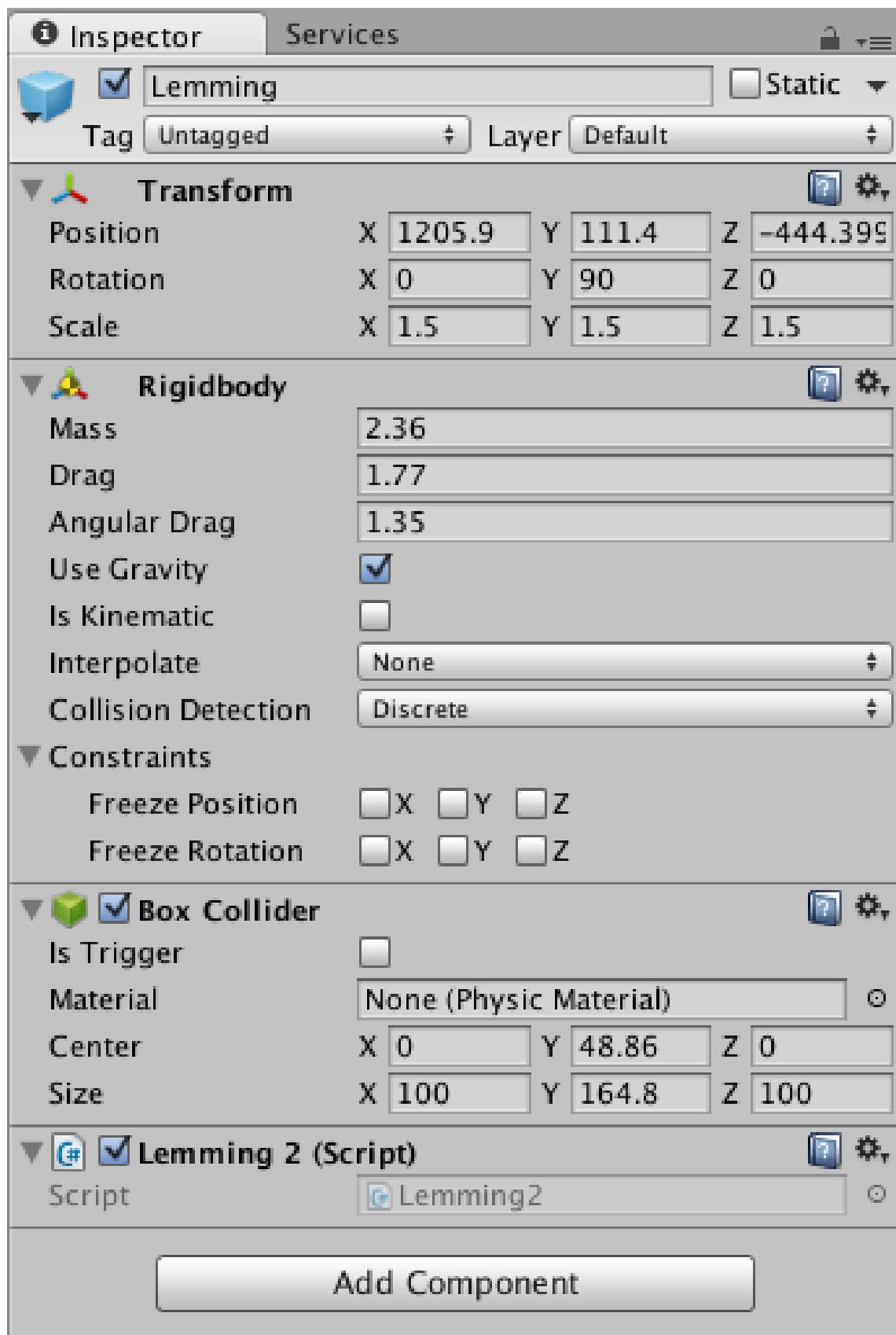
Lemming2.activeEl.SendInfo ();
}

```

Listing 3: Wybór aktywnego aktora.

Każdy z aktorów posiada detekcję kolizji z innymi elementami za pomocą nadpisania wbudowanych metod w środowisku Unity. Są to: OnTriggerEnter, OnTriggerExit, OnCollisionEnter, OnCollisionExit. Kolizja od triggera, różni się tym, iż metody OnCollision współpracują tylko z elementami, które posiadają zaimplementowaną fizykę (np. spadający element platformy), natomiast trigger to można być dowolny element, nie posiadający „ciała” (np. ściany).

W celu uproszczenia detekcji kolizji z innymi elementami wprowadzono czarne prostopadłościany emitujące rzeczywiste ściany. Są one koloru czarnego, dzięki temu projektor emitujący obraz w tych miejscach nie wyśle światła widzialnego. Natomiast dzięki takiemu zabiegowi możliwe jest zamodelowanie w środowisku unity kolizji z rzeczywistymi elementami.



Rysunek 7: Konfiguracja aktora

źródło: własne

5.3 Prefabrykaty

W Unity możliwe jest używanie prefabrykatów (Prefabs¹⁷). Są obiekty lub grupy obiektów, które służą do wielokrotnego wykorzystywania. W Projekcie założono, że wszystkie reużwalne komponenty (dziedziczone pomiędzy scenami) będą prefabrykatami.

Dodatkowo aktorzy gry (generowane dynamicznie) są również prefabrykatami. Instancje aktora są tworzone podczas działania aplikacji.

5.3.1 Kamery

Ważnym elementem gry są wirtualne kamery. To z nich renderowane jest ujęcie, czyli obraz gry. W projekcie zastosowano dwie kamery. Podczas konfiguracji uruchomieniowej należy ustawić by każda z kamer była wyświetiana na oddzielnym źródle obrazu (monitor, projektor). Dzięki temu projekt można uruchomić na dwóch prostopadłych ścianach.

Jako tło sceny wybrano jednolity kolor czarny, ponieważ ten kolor nie jest prezentowany podczas projekcji. Światło z projektora jest w tym miejscu znikome, wręcz niewidoczne. Stosując taki prosty zabieg można łączyć elementy rzeczywiste (np. rura czy inne elementy stałe znajdujące się w laboratorium) z wirtualną rzeczywistością.

¹⁷<http://docs.unity3d.com/Manual/Prefabs.html>



Rysunek 8: Ujęcie wirtualnych kamer

źródło: własne

Środowisko Unity domyślnie nie ma włączonej opcji wspierania wielu kamer jednocześnie. Do opisywanego prefakbrykatu należy dodać abstrakcyjny GameObject z poniższym prostym skryptem, który przy uruchomieniu skompilowanej gry sprawdza dostępność sprzętową ekranów.

```
using UnityEngine;  
using System.Collections;
```

```

public class DisplayScript : MonoBehaviour
{
    void Start()
    {
        Debug.Log( "displays connected: " +
            Display.displays.Length);
        if (Display.displays.Length > 1)
            Display.displays[1].Activate();
        if (Display.displays.Length > 2)
            Display.displays[2].Activate();
    }
}

```

Listing 4: Prosta aktywacja ekranów

Podczas testów uruchomieniowych przy dwóch kamerach występował problem z wydajnością karty graficznej. Zwłaszcza gdy do komputera podłączano dwa zewnętrzne ekranы po złączach cyfrowych (np. HDMI, DisplayPort, DVI). Finalnie problem rozwiązyano wydajniejszym komputerem, jednakże pośrednim rozwiązaniem było użycie portu VGA, który jest mniej obciążający dla karty graficznej.

5.3.2 Światło

Kolejny prefabrykat stworzono by zachować spójność w oświetleniu trójwymiarowej sceny. Służy on do zgrupowania wszystkich źródeł wirtualnego światła. Jest to element bez zawartej logiki biznesowej. Stworzono go w celu zachowania porządku w projekcie.

5.4 SocketIO

Jest to prefabrykat dostarczony jako komponent implementacji Socket.io w bibliotece Asset Store. Jest on udostępniony na licencji Open Source. Musi być on umiejscowiony w każdej scenie, która korzysta z połączenia sieciowego. Umiejscow-

wienie jest dowolne, gdyż jest to prefabrykat abstrakcyjny (nie posiada graficznej reprezentacji w trójwymiarowym modelu). W prefabrykacie wywołano skrypty SocketIOComponent, który odpowiada za inicjalizacje komunikacji sieciowej.

5.4.1 Network

Jest to kolejny prefabrykat abstrakcyjny, który służy do uruchomienia klasy Network odpowiadającej za implementacje metod służących do dwustronnej komunikacji. Prefabrykat ten jest nierozerwalnie złączony z SocketIO, gdyż bezpośrednio korzysta z metod dostarczonych przez tą bibliotekę.

```
private SocketIOComponent socket;

// Use this for initialization
void Start () {
    GameObject go = GameObject.Find( "SocketIO" );
    socket = go.GetComponent<SocketIOComponent>();

    socket.On( "open" , InitGame );
    socket.On( "button" , Button );
}
```

Listing 5: Inicjalizacja skryptu

Na początku wyszukiwany jest obiekt gry o określonej nazwie, a następnie pobierany komponent, czyli obiekt klasy. Warto pamiętać, że połączenie inicjalizowane jest już przy uruchomieniu, więc nie ma potrzeby „ręcznego” zestawiania warstwy sieciowej.

Metoda On w klasie SocketIOComponent odpowiada za nasłuchiwanie serwera. Jako pierwszy parametr przyjmuje ciąg znaków określający nazwę metody. Natomiast drugi to referencja do metody, która wywoła się podczas wywołania akcji o nazwie wynikającej z pierwszego parametru.

Na potrzeby łatwiejszego zarządzania zdarzeniami i uniknięcia błędów w nazwach przycisków stworzono enum, zawierający nazwy obsługiwanych przycisków. Wszyst-

kie metody odpowiadające za komunikację, a zwłaszcza za zarządzanie stanami oraz nazwami przycisków powinny przyjmować w parametrze opisywany atrybut wyliczalny.

```
using System;

namespace AssemblyCSharp
{
    public enum ButtonEnum
    {
        BUTTON1, BUTTON2, BUTTON3, BUTTON4,
        BUTTON5, BUTTON6, BUTTON7, BUTTON8,
        BUTTON9, BUTTON10
    }
}
```

Listing 6: Definicja przycisków.

Metoda zarejestrowana pod nazwą „open” wywoła się przy udanym zestawieniu połączenia. Jest to najlepsze miejsce do wstępnej konfiguracji planszy gry oraz przycisków w kontrolerze.

```
public void InitGame(SocketIOEvent e)
{
    socket.Emit("register_game");
    EnableButton(ButtonEnum.BUTTON1);
    SetText(ButtonEnum.BUTTON1, "Nazwa 1");
    DisableButton(ButtonEnum.BUTTON2);
}
```

Listing 7: Inicjalizacja komponentów

Powyższy przykład inicjalizacji pokazuje emitowanie akcji do serwera o nazwie „register_game”. Służy on do zarejestrowania gry na serwerze. Od tego czasu wszystkie akcje z kontrolera (np. wciśnięcie przycisku) będzie emitowane do gry. Dodatkowo ukazano wstępna konfigurację przycisków: Uaktywnienie przycisku pierwszego, ustawienie określonej nazwy oraz wyłączenie przycisku drugiego.

5.4.2 Budowanie zapytania

```
public void SetText(ButtonEnum btn, string text) {  
    Dictionary<string, string> dic = new  
        Dictionary<string, string> ();  
    dic.Add ("name", btn.ToString());  
    dic.Add ("text", text);  
    socket.Emit ("set_text", new JSONObject (dic));  
}
```

Listing 8: Przykładowe zapytanie do gra - kontroler

5.4.3 Replikator

Replikator to komponent do tworzenia nowych instancji aktora. Ma na celu cykliczne dodawanie kolejnych obiektów aktora w miejsce startowe określone w jego klasie. Dodatkowo uruchamia on nasłuchiwanie akcji przycisków płynących od kontrolera i zamienia je na wywołania metod na obecnie wybranym elemencie - aktywnym aktorze.



Rysunek 9: Konfiguracja replikatora

```
...
public void Run () {
    StartCoroutine( Runner() ) ;

    NetworkManager . StartListening ( "button_left" , Left ) ;
    NetworkManager . StartListening ( "button_right" ,
        Right ) ;
    NetworkManager . StartListening ( "button_kilof" ,
        Kilof ) ;
    NetworkManager . StartListening ( "button_lopata" ,
        Lopata ) ;
    NetworkManager . StartListening ( "button_jump" , Jump ) ;
    NetworkManager . StartListening ( "button_spadochron" ,
        Drabina ) ;
    NetworkManager . StartListening ( "button_rotate" ,
        Rotate ) ;
    NetworkManager . StartListening ( "button_startstop" ,
        Startstop ) ;
```

```

        NetworkManager.StartListening("button_reset",
            Reset);
    }

IEnumerator Runner() {
    while (lemmingCount < LemmingSize) {
        Create();
        lemmingCount += 1;
        yield return new WaitForSeconds
            (secoundLimit);
    }
}
...

```

Listing 9: Uruchomienie replikatora

```

...
void Left () {
    Lemming2.GetPrev ();
}

void Right () {
    Lemming2.GetNext ();
}

void Kilof () {
    if (Lemming2.activeEl) {
        Lemming2.activeEl.ToggleKilof ();
    }
}
...

```

Listing 10: Implementacja akcji

5.5 Serwer komunikacyjny

Jako technologię do obsługi połączeń zastosowano NodeJS. Obecnie jedyną zależnością zewnętrzną jest biblioteka Socket.IO¹⁸. Jako manager zależności wybrano NPM. Jest on domyślnie dołączony do oprogramowania NodeJS. Aby pobrać zależności należy uruchomić polecenie „npm install” w katalogu projektu.

```
{  
  "name": "server",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "author": "Kamil Warpechowski",  
  "license": "GNU",  
  "devDependencies": {  
    "socket.io": "^1.4.6"  
  }  
}
```

Listing 11: Definicja zależności w projekcie

Głównym założeniem części serwerowej jest propagowanie wiadomości na linii kontroler - gra. Jednakże dzięki zastosowaniu Socket.IO możliwa jest obsługa wielu urządzeń jednocześnie. (np. dwóch kontrolerów). Dlatego też zastosowano mechanizm pokoi. Jest to grupowanie unikalnych identyfikatorów połączeń, co pozwala wysyłać komunikaty jednocześnie do wszystkich kontrolerów lub gier. Na potrzeby projektu wprowadzono limity (jedna gra, dwa kontrolery), jednakże w przyszłości jest możliwość dalszej rozbudowy.

```
socket.on('register_controller', function () {  
  if (io.sockets.adapter.rooms[CONTROLLER].length < 1) {  
    console.log("register controller");  
  }  
});
```

¹⁸<https://www.npmjs.com/package/socket.io>

```
        socket . join (CONTROLLER) ;  
    }  
});  
  
socket . on ( ' register _ game ' , function () {  
    if ( io . sockets . adapter . rooms [GAME] ) {  
        console . log ( " register game " );  
        socket . join (GAME);  
    }  
});  
}
```

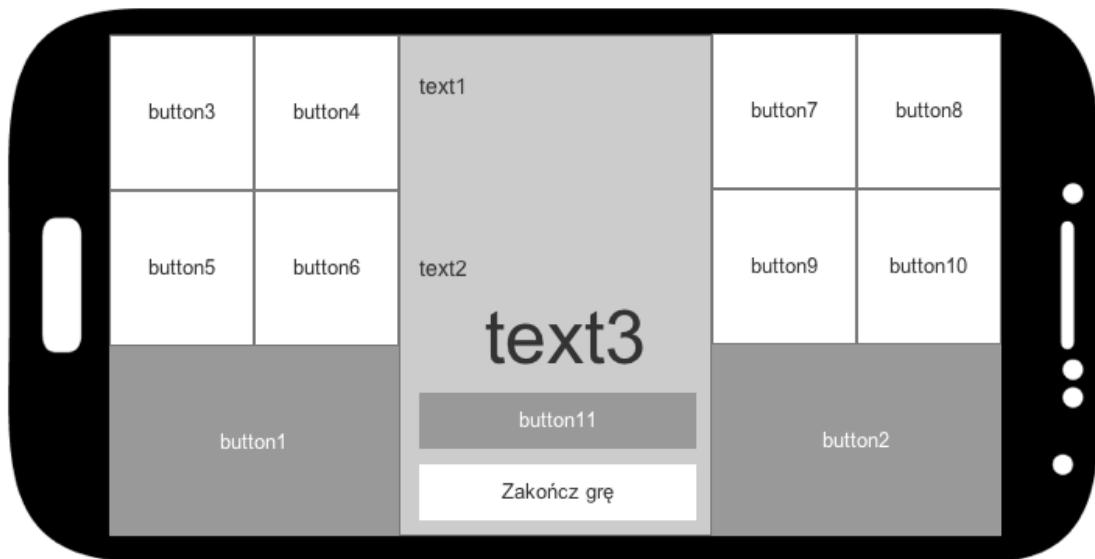
Listing 12: Grupowanie połączeń

```
socket . on ( ' enable _ button ' , function ( data ) {  
    console . log ( " enable _ button " , data );  
    io . to (CONTROLLER) . emit ( " enable _ button " , data );  
});
```

Listing 13: Przykład propagacji komunikatu

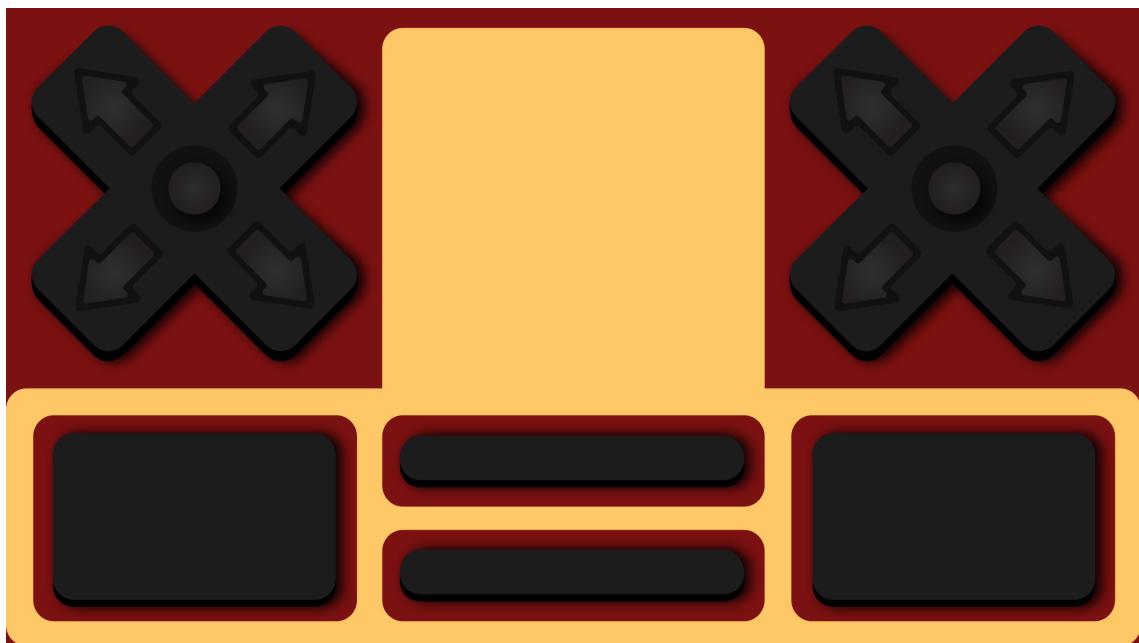
W przyszłości możliwe jest przechowywanie stanu gry na poziomie serwera (każdy kontroler może przesyłać unikalny identyfikator urządzenia).

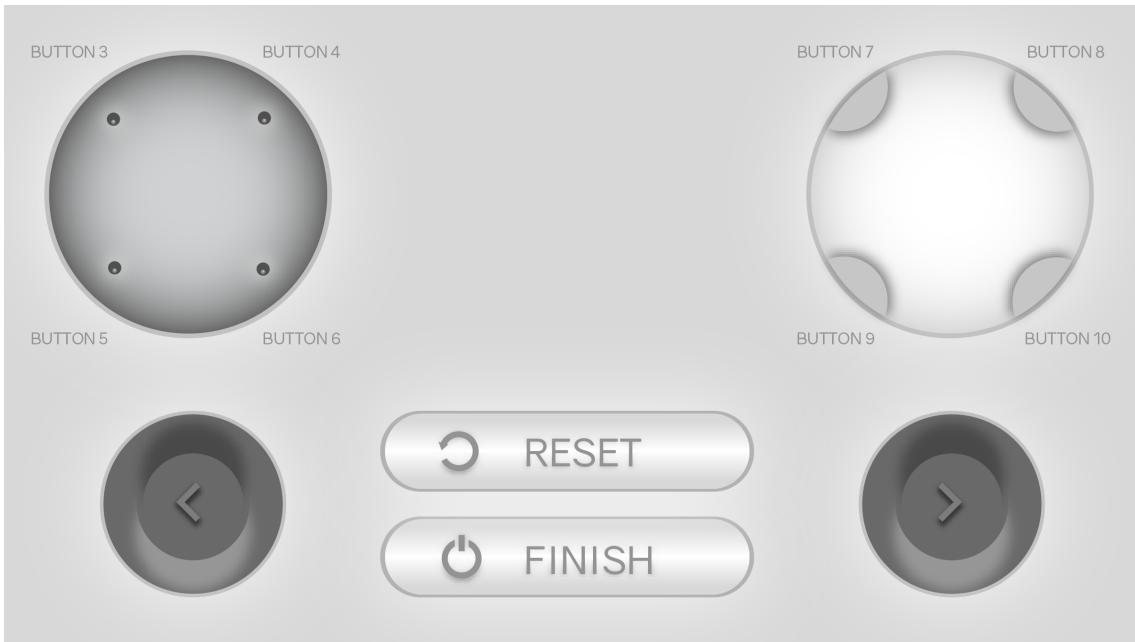
6 Aplikacja mobilna - kontroler



Rysunek 10: Makieta - układ przycisków

Główym założeniem było stworzenie uniwersalnego kontrolera przygotowanego pod dowolny rodzaj gry, bądź innej wizualizacji stworzonej w środowisku Unity. Podczas uruchomienia kontrolera serwer wysyła statusy przycisków oraz pól tekstowych.





Rysunek 11: Przykładowa wizualizacja kontrolera

6.1 Przyciski

Każdy przycisk może zostać skonfigurowany poprzez ustawienie tekstu. Dodatkowo można zablokować przycisk podczas gdy nie jest on potrzebny w danym czasie. Jednym z głównych założeń architektonicznych był rozdzielenie warstwy widoku od logiki biznesowej. Ustalono, że stworzenie nowego przycisku odbywać się będzie wymagało tylko dodania definicji w warstwie widoku (plik layout w formacie XML).

```
<pl.pjatk.remotecontroller.CustomButton  
    app:name="button2"  
    android:layout_gravity="center_horizontal"  
    android:layout_height="wrap_content"  
    android:layout_width="wrap_content"  
/>
```

Listing 14: Przykład zdefiniowanego przycisku

Na potrzeby realizacji powyższego założenia stworzono klasę CustomButton, która jest rozszerzeniem (dziedziczy bezpośrednio) klasy Button znajdującej się w pakiecie android.widget.

Każdy z przycisków musi posiadać własną nazwę kodową, gdyż serwer podczas komunikacji sieciowej identyfikuje przycisk poprzez unikalny klucz. Domyślnie w środowisku Android każdy komponent wizualny może posiadać swoje Id, jednakże jest one reprezentowane poprzez liczbę typu Integer. Dla ułatwienia dalszego rozwoju aplikacji postanowiono stworzyć nowy atrybut. Ich definicje umieszcza się w formacie XML w pliku attrs.xml

```
<resources>
    <declare-styleable name="CustomButton">
        <attr name="name" format="string" />
    </declare-styleable>
</resources>
```

Listing 15: Definicja atrybutu o nazwie name

W konstruktorze poza domyślnymi wywołaniami klasy bazowej Button zapisywana jest wartość atrybutu name do zmiennej o tej nazwie oraz następuje wstępna konfiguracja przycisku.

```
private static HashMap<String, CustomButton> buttons = new
HashMap<String, CustomButton>();

private void setUp() {
    buttons.put(getName(), this);
    setText(getName());

    setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            ServerCommunication.click_button(getName());
        }
    });
}
```

```
}
```

Listing 16: Konfiguracja w klasie CustomButton

Dzięki metodzie setUp konfigurowany jest przycisk. Ustawiana jest nazwa przycisku (zmiana możliwa za pomocą aplikacji Unity), przycisk jest zapisywany do listy wszystkich dostępnych przycisków (dzięki temu konfiguracja ilości przycisków odbywa się tylko w jednym miejscu - na poziomie warstwy widoku) oraz uruchamiany jest listener. Przy każdym kliknięciu wywala się metoda w singletonie SerwerCommunication.

6.2 Użycie stylu

Dodatkowym wymaganiem było umożliwienie szybkiej zmiany wyglądu przycisków w trakcie działania aplikacji. Użycie natywnych stylów niestety nie jest możliwe bez ponownego renderowania widoku. Aby zaoszczędzić czas i moc obliczeniową postanowiono, iż zasadniczo zmiany tła za pomocą metody backgroundResource. Różne wyglądy przycisku mogą być definiowane jako selektory (zewnętrzne pliki xml w katalogu drawable). Ważne, by plik był odpowiednio nazwany, gdyż po nazwie następuje wyszukiwanie schematu podczas zmiany wyglądu.

```
<?xml version="1.0" encoding="utf-8"?>
<selector
    xmlns:android="http://schemas.android.com/apk/res/android
">
    <item android:state_pressed="true" >
        <shape>
            <gradient
                android:startColor="#bf1d00"
                android:endColor="#801300"
                android:angle="270" />
            <corners android:radius="10dp" />
            <stroke
```

```

        android:width="1dp"
        android:color="#71c2eb" />
    </shape>
</item>
<item>
    <shape
xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
        <gradient android:startColor="#FFFFFF"
            android:endColor="#999"
            android:angle="270" />
    </shape>
</item>
</selector>
```

Listing 17: Przykład zdefiniowanego tła

Wymaganiem była jednoczesna zmiana wyglądu wszystkich przycisków. Rozwiązano to za pomocą metody statycznej, która iteruje po wszystkich przyciskach i wywołuje metodę `backgroundResource`.

```

public static void setLayout(int i) {
    for (CustomButton btn : buttons.values()) {
        btn.setBackgroundResource(i);
    }
}
```

Listing 18: Zmiana tła dla wszystkich przycisków

```
CustomButton.setLayout(R.drawable.dark);
```

Listing 19: Przykład wywołania zmiany tła przycisków.

6.3 Komunikacja sieciowa

```
public class ServerCommunication {  
    private static Socket mSocket = null;  
    private static AppCompatActivity activity = null;  
  
    public static void start(AppCompatActivity  
        mainActivity) {  
        activity = mainActivity;  
        ServerCommunication.start();  
    }  
  
    private static void start() {  
        if(mSocket == null) {  
            Log.i("socket", "createServer");  
            try {  
                mSocket =  
                    IO.socket("http://192.168.0.12:5555");  
                mSocket.connect();  
                ServerCommunication.listenDisableButton();  
                ServerCommunication.listenEnableButton();  
                ServerCommunication.listenSetText();  
                mSocket.emit("register_controller");  
            } catch (URISyntaxException e) {  
            }  
        }  
    }  
  
    public static void click_button(String buttonName) {  
        ServerCommunication.start();  
        mSocket.emit("click", buttonName);  
    }  
}
```

```

        }

public static void listenDisableButton() {
    ServerCommunication.start();
    mSocket.on("disable_button", new Emitter.Listener()
    {
        @Override
        public void call(Object... args) {
            Log.i("socket", "disable_button");
            try {
                JSONObject mainObject = new
                    JSONObject(args[0].toString());
                final String name =
                    mainObject.getString("name");
                activity.runOnUiThread(new Runnable() {
                    public void run() {
                        CustomButton.disableButton(name);
                    }
                });
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    });
}
...

```

Listing 20: Komunikacja sieciowa

Klasa ServerCommunication to prosta implementacja biblioteki Socket.IO. Posiada ona metody statyczne, służące do nasłuchiwanego akcji przychodzących z serwera lub emitowania komunikatów do aplikacji Unity. Każde wywołanie sprawdza na początku czy połączenie socket jest aktywne, jeśli nie, to tworzone jest nowe połą-

czenie i uruchomiane domyślne nasłuchiwanie na akcje. Ważne jest to, iż wszystkie nasłuchiwanie tworzone są w nowych wątkach i wykonanie akcji aktualizacji graficznego interfejsu bezpośrednio nie jest możliwe. Należy wtedy utworzyć nowy obiekt klasy Runnable i uruchomić go za pomocą metody activity.runOnUiThread. Do obsługi bardziej złożonych komunikatów JSON używana jest standardowa klasa Java - JSONObject.

```
...
} catch (Exception e) {
    Context context = activity.getApplicationContext();
    CharSequence text = "Communication error";
    int duration = Toast.LENGTH_SHORT;

    Toast toast = Toast.makeText(context, text, duration);
    toast.show();
}
...
```

Listing 21: Przykład obsługi wyjątków

Jako graficzną reprezentację potencjalnych błędów (wyjątków w kodzie) można zastosować komunikaty pojawiające się poprzez zastosowanie klasy Toast dostępnej w Android SDK.

7 Środowisko uruchomieniowe

Powyżej opisana aplikacja została uruchomiona testowo w laboratorium na Polsko-Japońskiej Akademii Technik Komputerowych.

7.1 Aplikacja główna - Unity

Aplikacja została uruchomiona na komputerze przenośnym(laptop) posiadającym kartę graficzną umożliwiającą podłączenie dwóch zewnętrznych ekranów - projektów. Pierwszy z nich został połączony za pomocą złącza cyfrowego HDMI, natomiast

drugi łączem DVI.

7.2 Serwer komunikatów

Serwer został uruchomiony na tym samym urządzeniu co aplikacja główna. Do uruchomienia niezbędne było zainstalowanie Node.JS wraz z menadżerem zależności - NPM.

7.3 Aplikacja mobilna - kontroler

Kontroler został uruchomiony na urządzeniu Xiaomi Mi4c wyposażonym w system Android w wersji 5.1. Urządzenie sprawdziło się jako kontroler, gdyż posiada ekran o rozmiarze 5 cali. Ilość pamięci operacyjnej była wystarczająca. Aplikacja zużywała tylko około 2-4% pamięci RAM.

8 Inne implementacje - Google Cardboard

Innym przykładem implementacji projektu w rozszerzonej rzeczywistości może być platforma Google Cardboard¹⁹. Są to niskobudżetowe okulary stworzone przez firmę Google do wyświetlania wirtualnej rzeczywistości. Kartonowe okulary powstały w celu wyświetlania obrazu stereoskopowego. Posiadają one miejsce do umieszczenia dowolnego telefonu komórkowego typu smartphone. Do projektu wybrano wersję, która pozwala na umieszczenie telefonu w sposób taki, iż tylna kamera nie jest zasłonięta przez obudowę. Dzięki temu można użyć platformę Cardboard przeznaczoną pierwotnie tylko do wirtualnej rzeczywistości do stworzenia aplikacji wykorzystującej augmented reality.



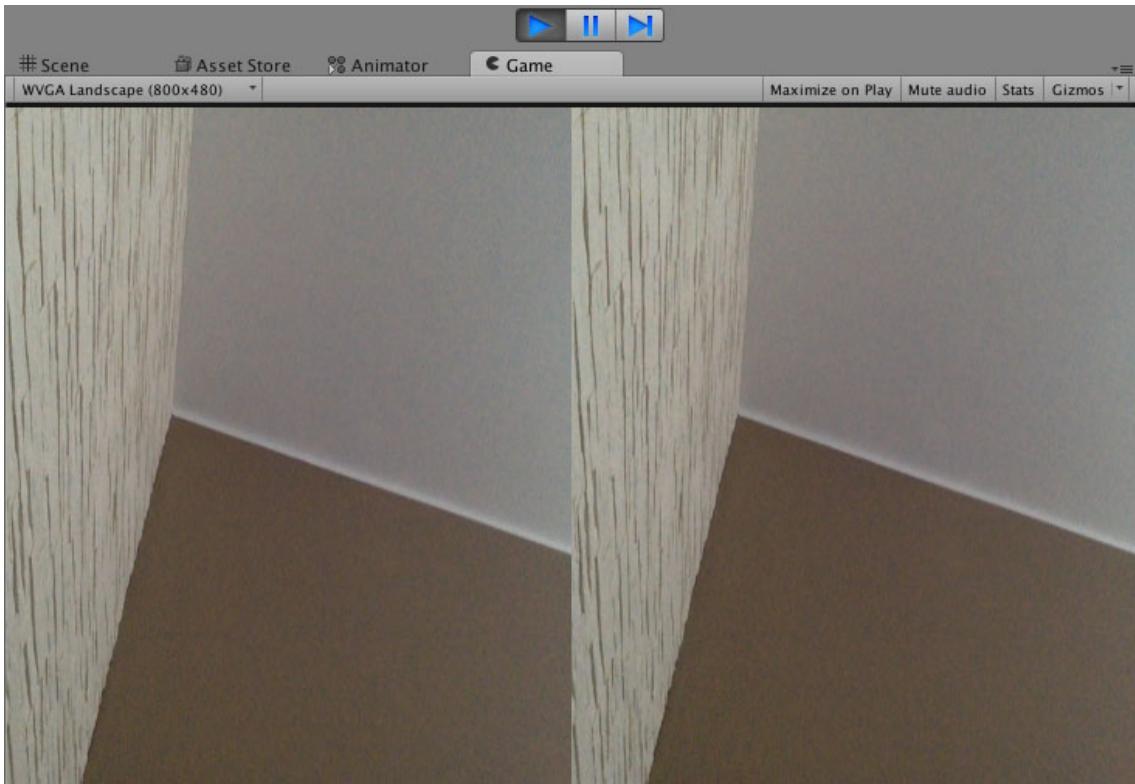
Rysunek 12: Google Cardboard w wersji do samodzielnego złożenia

źródło:

<http://gizmodo.com/turn-your-android-into-a-virtual-reality-headset-with-g-1596026538>

Dzięki użyciu kamery użytkownik widzi obraz znajdujący się przed nim. Aby obraz był stereoskopowy należało stworzyć aplikację, która wyświetlić strumień danych z kamery dzieląc go na dwa obrazy (kolejno dla lewego oraz prawego oka).

¹⁹<https://vr.google.com/cardboard/index.html>



Rysunek 13: Ujęcie z kamery aparatu w obrazie stereoskopowym - scena w Unity
źródło własne

8.1 ArToolkit

W celu wyświetlenia wirtualnych obiektów na obrazie z kamery rozbudowano aplikację z porzednich rozdziałów. Do platformy Unity doinstalowano zewnętrzny komponent ARToolKit²⁰. Jest to biblioteka wydana przez University of Washington²¹, lecz obecnie upostępniona jest na licencji GNU. Kod źródłowy jest otwarty i rozwijany przez środowisko Open Source²².

8.2 Markery

Za pomocą tej biblioteki możliwe jest wykrywanie w obrazie z kamery markerów, czyli specjalnie przygotowanych czarno-białych obrazków (w naiwiej implementacji - wydrukowanych na kartkach), oraz nakładanie w ich miejsce trójwymiarowych modeli lub całych scen. Dzięki bibliotece ArToolkit możliwe jest diagnozowanie

²⁰<http://artoolkit.org/>

²¹<https://www.hitl.washington.edu/artoolkit/>

²²<https://github.com/artoolkit>

pod jakim kątem padania oraz w jakiej odległości od urządzenia znajduje się marker. Umiejscowienie tagu analizowane jest w czasie rzeczywistym, co zapewni ciągłą korekcję ułożenia wirtualnych modeli względem ich realnych odpowiedników.



Rysunek 14: Przykład przygotowanego obrazka do rozpoznawania - marker

Szablony markerów można wykonywać we własnym zakresie. Aby zimportować nowe obrazki do biblioteki ArToolkit należy przygotować specjalny plik binarny reprezentujący model markera.²³.

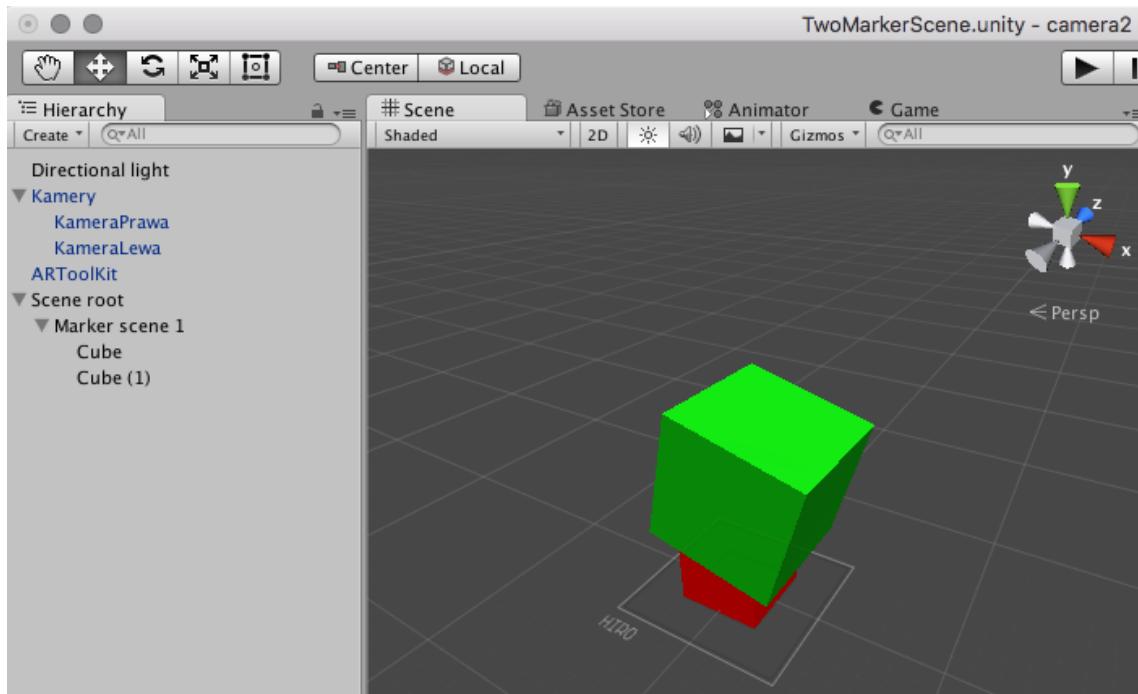
²³<http://bit.ly/1YU199f>



Rysunek 15: Przykład zastosowania markera w ARToolkit

źródło <http://arblog.inglobetechnologies.com/?p=421>

8.3 Przykład implementacji



Rysunek 16: Podstawowa implementacja
źródło własne

Implementacja na platformie Unity przy użyciu ArToolkit jest zasadniczo prosta. Przy użyciu wcześniej przygotowanych kodów markerów (domyślnie dostępne są dwa) i zastosowaniu prefabrykatów w trójwymiarowej przestrzeni można wskazać miejsce od którego będzie odliczana odległość do pozostałych elementów. np. Jeżeli bryła ustawniona zostanie w odległości 5cm od prefabrykatu to podczas analizy obrazu i wykrycia rzeczywistego markera ta bryła zostanie umiejscowiona na ekranie w tej samej odległości.

8.4 Napotkane problemy i ograniczenia

1. Proces renderowania musi odbywać się na telefonie komórkowym, gdyż obraz kamery jest ciągle analizowany. Przy dużych scenach stworzonych w Unity moc obliczeniowa urządzenia jest niewystarczająca.
2. Analizowanie pozycji markera przy nieustannie włączonej kamerze powoduje dużą drenację baterii urządzenia. Czas pracy na baterii jest mocno ograniczony
3. Unity w wersji Personal (darmowej) skompilowanej pod platformę mobilną (np. Android) nie udostępnia obsługi sieci (np. za pomocą połączenia TCP). Nie pozwala to na połączenie z zewnętrznym kontrolerem.
4. Sterowanie za pomocą kontrolera bez fizycznych przycisków z założonymi okularami Google Cardboard jest bardzo uciążliwe. W przyszłości należałyby rozważyć połączenie telefonu z zewnętrznym kontrolerem typu PAD²⁴.

²⁴<https://pl.wikipedia.org/wiki/Gamepad>

9 Inne implementacje - Project Tango

Kolejnym przykładem implementacji może być platforma Google Project Tango²⁵. Jest to platforma rozszerzonej rzeczywistości zapoczątkowana przez Johny'ego Lee (współtwórcy między innymi Microsoft Kinect²⁶) w 2014.

Idea projektu jest bardzo podobna jak przykład zaprezentowany w poprzednim rozdziale. Jednakże Project Tango to również podzespoły sprzętowe. Twórcy zastosowali specjalne kamery do pomiaru głębi oraz analizy ruchu (technologia podczerwieni). Kamery te korzystają z technologii Intel Real Sense. Dzięki temu urządzenie potrafi analizować obraz kamery i mapować go na trójwymiarowy obraz. Z dokładnością do milimetra urządzenie jest w stanie określić wymiary realnych elementów znajdujących się przed kamerą. Dzięki temu nie ma potrzeb używania zbędnych fizycznych markerów do określenia miejsca w którym znajduje się odbiorca z urządzeniem.

Firma Google zaprezentowała projekt w 2014 roku wraz z dwoma urządzeniami testowymi (The Yellowstone tablet, The Peanut phone). Jednakże te urządzenia nie trafiły nigdy na rynek komercyjny. Dopiero w 2016 roku firma Lenovo zaprezentowała pierwszy masowo produkowany telefon obsługujący Project Tango - Lenovo Phab2 Pro.

Projekt pod początku udostępnia developerom możliwość tworzenie aplikacji za pomocą API do języków Java oraz C. Dodatkowo udostępniona jest SDK (Software Development Kit) wraz z obszerną dokumentacją do platformy Unity²⁷.

Jedynym środowiskiem uruchomieniowym dostępnym na obecną chwilę jest Android, chociaż Google zapowiada możliwość w przyszłości uruchomienia w środowisku Windows 10.

²⁵<https://get.google.com/tango/>

²⁶[https://en.wikipedia.org/wiki/Johnny_Lee_\(computer_scientist\)](https://en.wikipedia.org/wiki/Johnny_Lee_(computer_scientist))

²⁷<https://developers.google.com/tango/apis/unity/>



Rysunek 17: Prototyp urządzenia

Google zaprezentowało również okulary do wirtualnej rzeczywistości. Jednakże był to tylko prototyp. Obecnie na rynku nie ma urządzenia dostosowanego do tego celu. Jedyną możliwością byłoby użycie Lenovo Phab Pro wraz z Google Cardboard.

9.1 Wady i zalety

1. Project Tango jest stworzony jako kooperacja dedykowanego sprzętu oraz specjalistycznego oprogramowania, co za tym idzie wydajność urządzeń powinna być duża.
2. Dużą wadą jest to, iż na rynku dopiero pojawił się pierwszy smartphone z obsługą projektu. Technologia wydaje się być na początkowej fazie rozwoju.
3. Dzięki tej technologii można zrezygnować z markerów opisanych w poprzednim rozdziale. Odczucia użytkowników powinny być bardziej intuicyjne.

10 Wnioski i dalszy rozwój

Obserwując rynek nowych technologii można zauważyc duże zainteresowanie zarówno rozszerzoną jak i wirtualną rzeczywistością. Producenci eksperymentują z różnymi urządzeniami, jednakże większość ich jest w fazie testów i nie jest dostępna dla potencjalnych konsumentów. Wiele z firm próbuje tworzyć własne standardy, jednakże żaden z nich nie zyskał jeszcze dużej popularności. Wśród twórców oprogramowania interesujących się wirtualną rzeczywistością narasta trend stosowania oprogramowania Unity jako platformy do programowania. Trend ten wykorzystują przedsiębiorstwa produkujące urządzenie udostępniając specjalne Software Development Kit dla Unity. Warto wspomnieć, iż Microsoft planuje udostępnić know-how innym, mniejszym firmom²⁸. Spowoduje to duży rozrost rynku.

Oczami programisty można stwierdzić, że stopa wejścia na nowy rynek jakim jest oprogramowanie urządzeń AR i VR jest dość prosty. Dzięki połączeniu kilku technologii w krótkim czasie można stworzyć nowoczesne narzędzie, które może rewolucjonizować rynek.

²⁸<https://www.engadget.com/2016/06/01/microsoft-opens-the-hololens-platform-to-other-companies/>

Spis rysunków

1	Ewolucja interfejsów użytkownika	5
2	Wizualizacja Microsoft HoloLens	6
3	Labolatorium PJATK	8
4	Poglądowy schemat działania hologramu	10
5	Poglądowy schemat działania - mapping	12
6	Model w środowisku Unity	17
7	Konfiguracja aktora	20
8	Ujęcie wirtualnych kamer	22
9	Konfiguracja replikatora	27
10	Makieta - układ przycisków	31
11	Przykładowa wizualizacja kontrolera	32
12	Google Cardboard w wersji do samodzielnego złożenia	40
13	Ujęcie z kamery aparatu w obrazie stereoskopowym - scena w Unity	41
14	Przykład przygotowanego obrazka do rozpoznawania - marker	42
15	Przykład zastosowania markera w ARToolkit	43
16	Podstawowa implementacja	43
17	Prototyp urządzenia	46

Spis tablic

1	Porównanie silników gier	13
---	------------------------------------	----

Listings

1	Konfiguracja pozycji początkowej i siły skoku	18
2	Przykład wysyłania komunikatów do serwera	18
3	Wybór aktywnego aktora.	19
4	Prosta aktywacja ekranów	23
5	Inicjalizacja skryptu	24
6	Definicja przycisków.	25
7	Inicjalizacja komponentów	25

8	Przykładowe zapytanie do gry - kontroler	26
9	Uruchomienie replikatora	28
10	Implementacja akcji	28
11	Definicja zależności w projekcie	29
12	Grupowanie połączeń	30
13	Przykład propagacji komunikatu	30
14	Przykład zdefiniowanego przycisku	32
15	Definicja atrybutu o nazwie name	33
16	Konfiguracja w klasie CustomButton	34
17	Przykład zdefiniowanego tła	35
18	Zmiana tła dla wszystkich przycisków	35
19	Przykład wywołania zmiany tła przycisków.	35
20	Komunikacja sieciowa	37
21	Przykład obsługi wyjątków	38

Literatura

- [1] Building Microservices, Sam Newman , Wydanie 4, 2016