



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

tutaj będzie tytuł

Spis treści

1 Wprowadzenie	4
1.1 Rozszerzona rzeczywistość	5
2 Cel pracy	7
3 Implementacje	8
3.1 Implementacja w technologii hologramu	9
3.2 Implementacja w technologii mappingu 3d	10
4 Wykorzystane technologie	12
4.1 Unity	12
4.1.1 Dlaczego Unity	12
4.1.2 Alternatywne rozwiązania	12
4.1.3 Wybór języka programowania	12
4.1.4 Unity IDE	13
4.1.5 Licencja i koszty	13
4.2 Android	13
4.3 Komunikacja sieciowa	14
4.3.1 UNET	14
4.3.2 HTTP (SOAP, REST)	14
4.3.3 WebSocket	15
5 Aplikacja główna	16
5.1 Świat gry	16
5.2 Aktorzy	16
5.3 Prefabrykaty	17
5.3.1 Kamery	17
5.3.2 Światło	19
5.4 SocketIO	19
5.4.1 Network	20
5.4.2 Budowanie zapytania	22
5.4.3 Replikator	22
5.5 Logika biznesowa	24

5.6	Serwer komunikacyjny	24
6	Aplikacja mobilna - kontroler	27
6.1	Przyciski	28
6.2	Użycie stylów	30
6.3	Informacje tekstowe	32
6.4	Komunikacja sieciowa	32
7	Środowisko uruchomieniowe	32
7.1	Serwer	32
7.2	Aplikacja mobilna - kontroler	32
8	Inne implementacje - Google Cardboard	33
8.1	ArToolkit	34
8.2	Markery	34
8.3	Przykład implementacji	36
8.4	Napotkane problemy i ograniczenia	37
9	Inne implementacje - Project Tango	38
9.1	Wady i zalety	39
10	Wnioski i dalszy rozwój	40

1 Wprowadzenie

Ludzie od lat przyzwyczaili się korzystać z elektroniki oraz internetu na standarodowych 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.

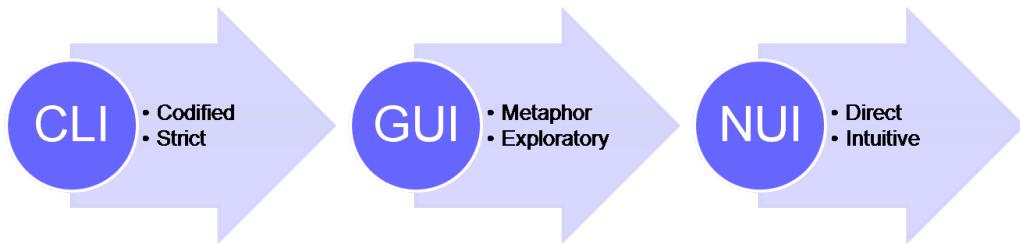
Przez ostatnie kilka lat narasta trend poszukiwania innych metod dostępu do danych, zwłaszcza multimedialnych. Obecnie wiele przedsiębiorstw prowadzi badania nad nowymi, bardziej naturalnymi dla ludzi interfejsami, które nie wymagałyby użycia standarodwzych(sztucznych) urządzeń wejścia typu klawiatura 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-na-pokladzie.aspx>



Rysunek 1: Ewolucja interfejsów użytkownika

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

Wiele nowych urządzeń próbuje implementować sterowanie intefejsem użytkownika za pomocą gestów (np. Microsoft Kinect⁴), czy też za pomocą myśli (np. Emotiv⁵). 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 pojawią się sale kinowe pozwalające na projekcję filmów trójwymiarowych wraz z dodatkowymi elementami takimi jak: drganie foteli, wiatr, dym, woda⁶. Jednakże w obecnej chwili taki format rozrywki jest dość drogi, gdyż wymaga specjalnie przygotowanej sali kinowej oraz okularów, które pozwalają tworzyć iluzję przestrzenną.

1.1 Rozszerzona rzeczywistość

Coraz bardziej popularne staje się pojęcie rozszerzonej rzeczywistości (ang. augmented reality). Jest to zbiór różnych technologii pozwalającej łączyć świat rze-

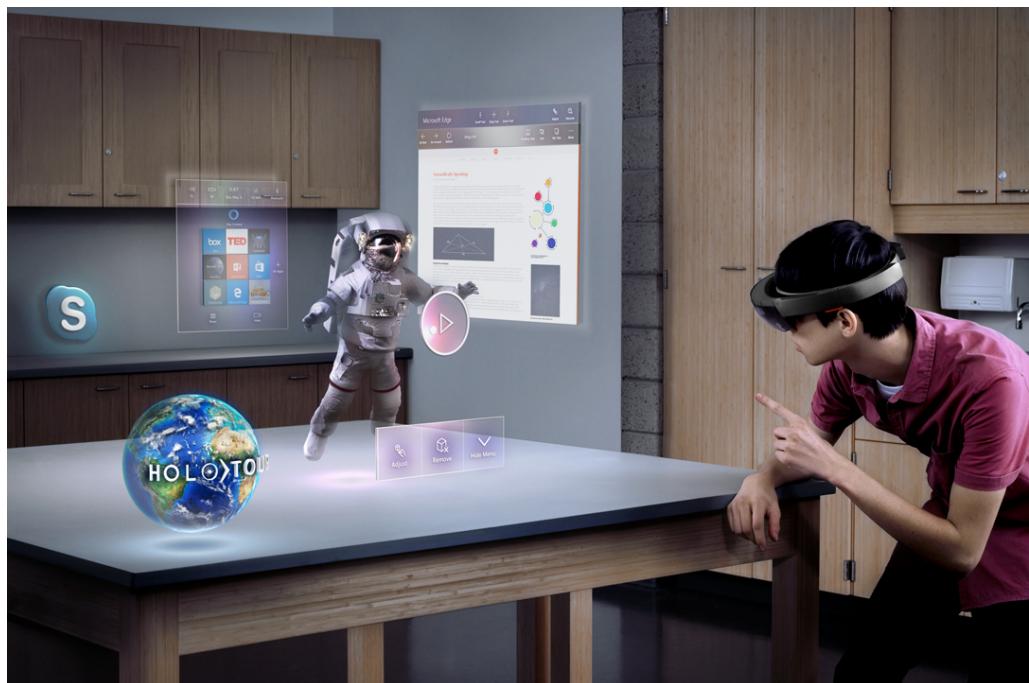
⁴<http://www.xbox.com/pl-PL/xbox-one/accessories/kinect-for-xbox-one>

⁵<http://emotiv.com>

⁶<http://cinema-city.pl/4dx-info>

czywisty 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.”⁷. Stworzył on interaktywne instalacje takie jak Glowflow⁸, Metaplay⁹ oraz Videoplac¹⁰.



Rysunek 2: Wizualizacja Microsoft HoloLens

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

⁷<http://www.techsty.art.pl/hipertekst/cyberprzestrzen/krueger.htm>

⁸<http://dada.compart-bremen.de/item/artwork/1347>

⁹<http://dada.compart-bremen.de/item/artwork/1348>

¹⁰<http://dada.compart-bremen.de/item/artwork/1346>

2 Cel pracy

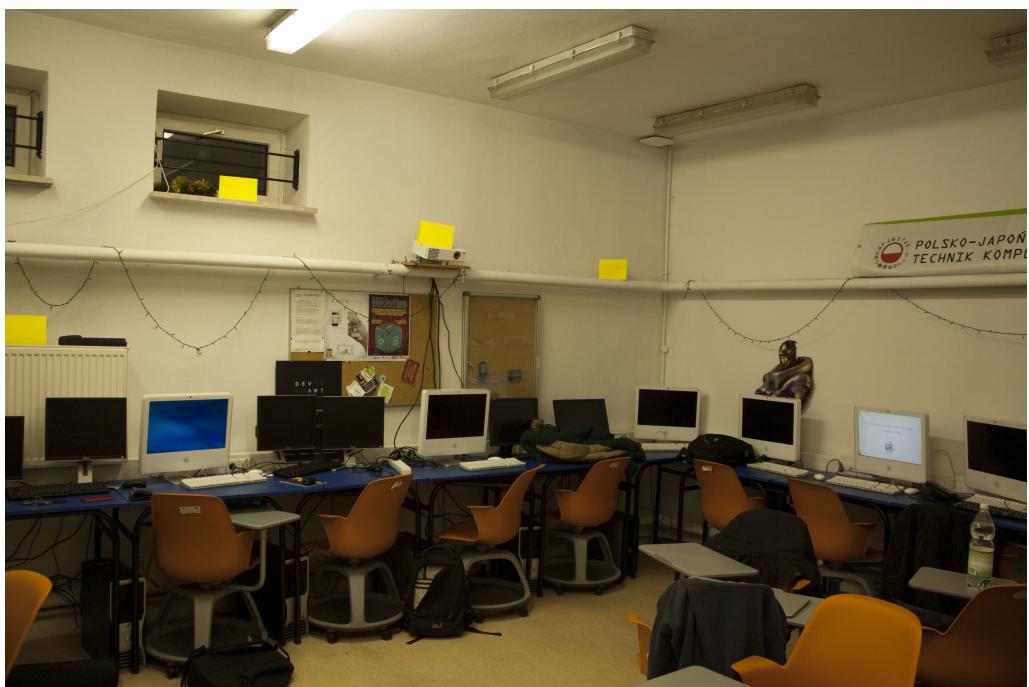
Celem niniejszej pracy jest stworzenie platformy do budowania gier oraz interaktywnych animacji prezentowanej za pomocą rozszerzonej rzeczywistości sterowanej za pomocą zdalnego kontrolera. Praca zawiera przykłady różnych implementacji w technologiach z rodziny AR.

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

źródło własne

Głównym założeniem było stworzenie minigry opartej na grze z początku lat dwudziestolecia o nazwie Lemmingi¹¹. 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.

¹¹<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 labolatorium. Dzięki takiem 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.

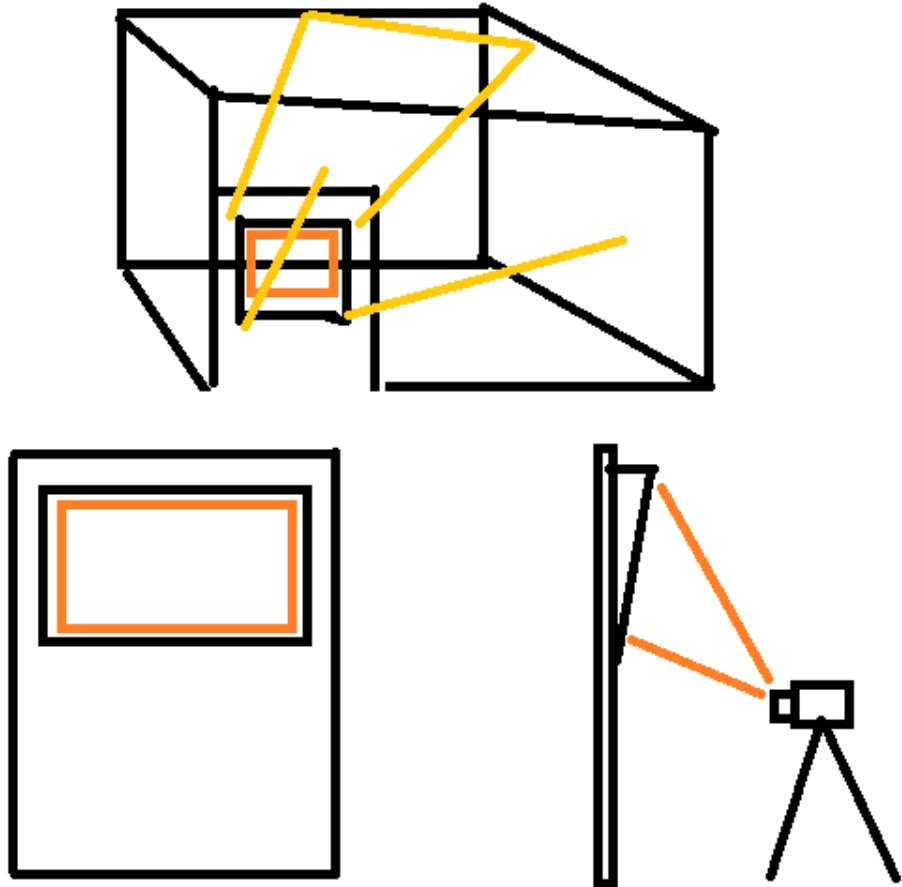
Podczas eksperymentów próbowało wiele różnych folii oraz kilka ustawień projektorów.

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.
- **dalej**

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) należałoby spoglądać przez szybę pod jednym wskazanym kątem. Co za tym idzie prawidłowy odbiór instalacji byłby 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

źródło: własne

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 iewnętrznych), 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 uniewidaczać 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.

Tutaj opisać dwa ekrany

Tutaj opisać próby i przemyślenia + wstawić obrazek

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 Unity, co za tym idzie możliwe jest łatwie 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

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 ów język 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 kołopaty 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 subskryencyjnym (75 dolarów za miesiąc)¹². 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

¹²<https://store.unity3d.com/subscribe>

mobilne(systemy: iOS, Android, Windows Phone, Tizen itp¹³). Jednakże Unity w 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.¹⁴ 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 wysyającym 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 oznaczone było użycie Bluetooth, jednak ograniczyły 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 transformacja danych do oraz z formatu JSON lub XML. Jednakże dużą zaletą wspomnianych protokołów jest prostota implementacji

¹³<https://unity3d.com/unity/multiplatform>

¹⁴<https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=1>

w większości języków programowania, gdyż są już gotowe komponenty.

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.io¹⁵. 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.

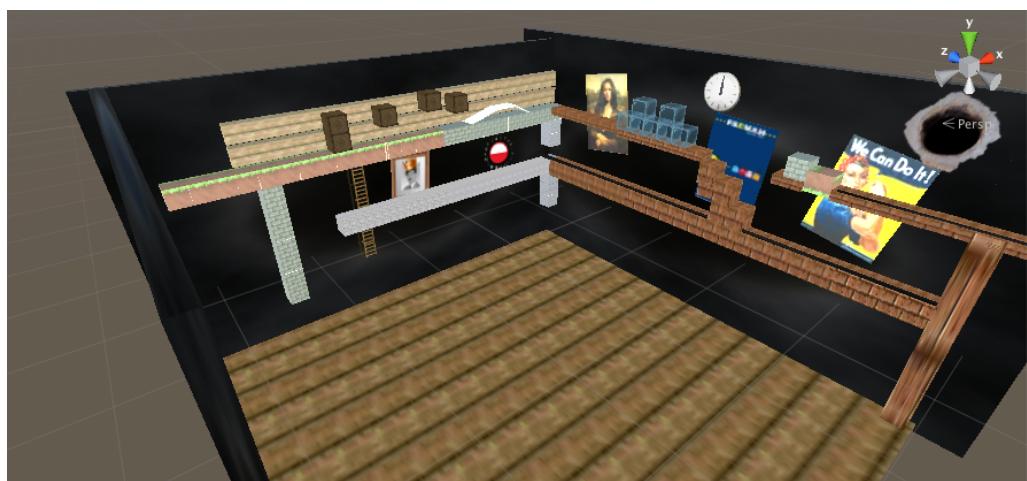
¹⁵<http://socket.io/>

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.
tutaj będą makiety

5.1 Świat gry

Opisać



Rysunek 5: Model w środowisku Unity

źródło: własne

5.2 Aktorzy

Opisać



Tablica 2: Właściwości aktora

Opisać

```
public void SendInfo() {  
    Network . SendMessage( "hasax_ "+this . hasAx) ;
```

```
    Network . SendMessage( "hassh_ "+this . hasSh ) ;  
    Network . SendMessage( "hasdrabina_ "+this . drabina ) ;  
    Network . SendMessage( "isMove_ "+this . isMove ) ;  
}
```

Listing 1: Do Poprawy

5.3 Prefabrykaty

W Unity możliwe jest używanie prefabrykatów (Prefabs¹⁶). Są obiekty lub grupy obiektów, które służą do wielokrotnego wykorzystywania. W Projekcie założono, że wszystkie reużwalne komponenty (dziedziczone pomiędzy scenami) będą prefabrykaciami.

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 wyświetlana 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 labolatorium) z wirtualną rzeczywistością.

¹⁶<http://docs.unity3d.com/Manual/Prefabs.html>



Rysunek 6: 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 skopilowanej 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 2: 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 pośredniom 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 świata. Jest to element bez zwannej 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. Umiejscowienie jest dowolne, gdyż jest to prefabrykat abstrakcyjny (nie posiada graficznej reprezentacji w trójwymiarowym modelu). W prefabrykacie wywołano skrypt SocketIOComponent, który odpowiada za inicjalizację 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 3: Inicjalizacja skryptu

Na początku wyszukiwani 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 przymuje 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. Wszystkie 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 4: 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 5: 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. wcisnięcie 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 6: Przykładowe zapytanie do gra - kontroler

Dokończyć opisywać logikę, network manager itd.

5.4.3 Replikator

Opisać replikator



Rysunek 7: 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 7: Uruchomienie replikatora

```

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

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

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

```

Listing 8: Implementacja akcji

Opisać pozostałe prefabrykaty

5.5 Logika biznesowa

Opisać

5.6 Serwer komunikacyjny

Jako technologię do obsługi połączeń zastosowano NodeJS. Obecnie jedyną zależnością zewnętrzną jest biblioteka Socket.IO¹⁷. 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" ,
```

¹⁷<https://www.npmjs.com/package/socket.io>

```

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

```

Listing 9: 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");
        socket.join(CONTROLLER);
    }
});

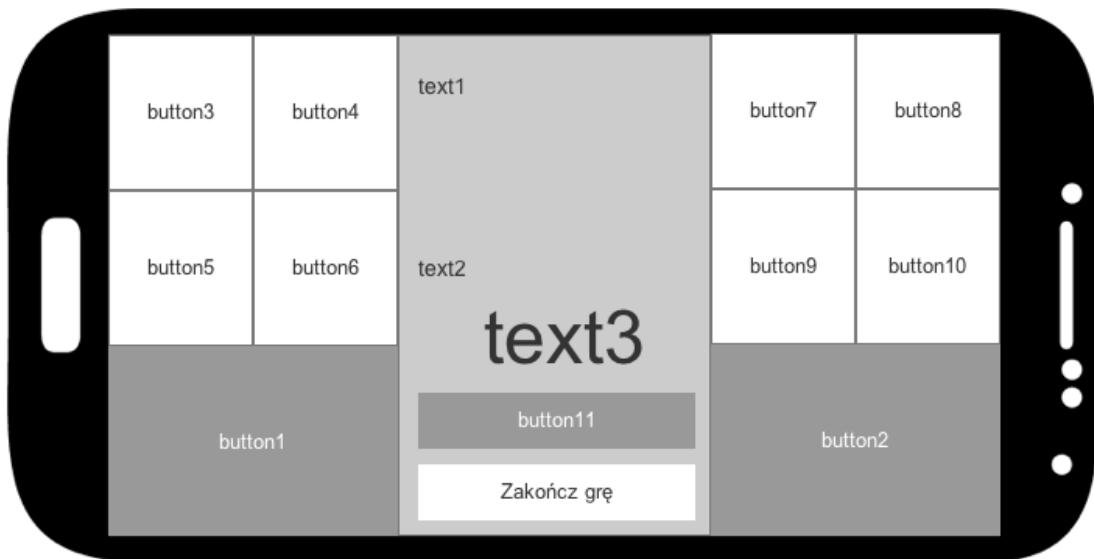
socket.on('register_game', function () {
    if (io.sockets.adapter.rooms[GAME]) {
        console.log("register game");
        socket.join(GAME);
    }
});
}

```

Listing 10: Grupowanie połączeń

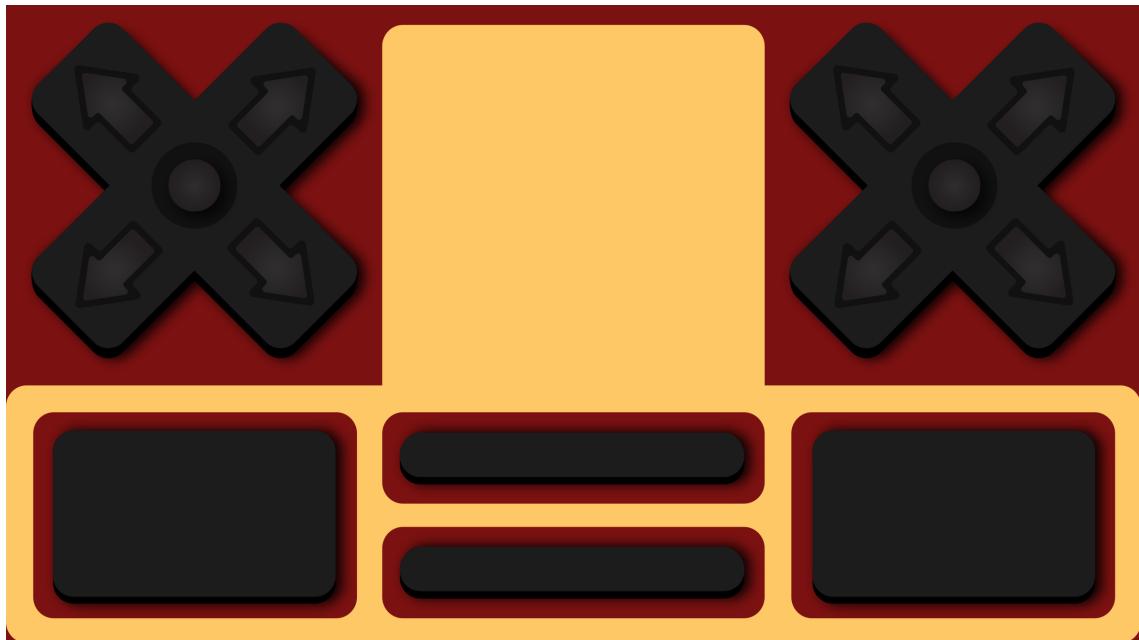
Dalej opisać przykład propagacji

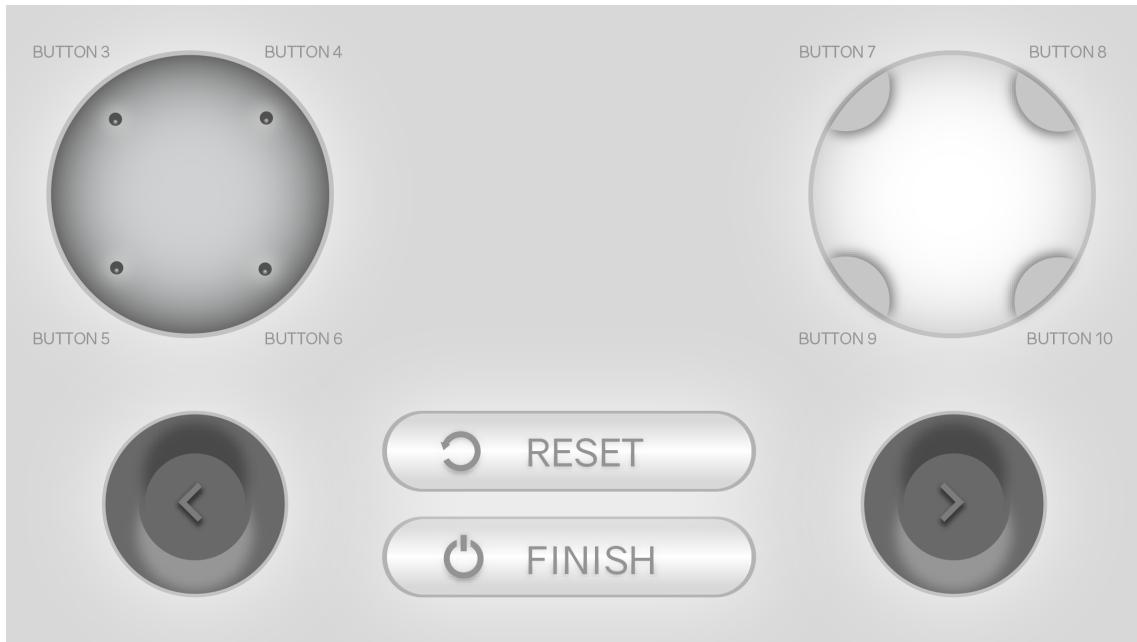
6 Aplikacja mobilna - kontroler



Rysunek 8: 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 9: 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 11: 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 12: 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 następna 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 ) {
            try {
                new Runner(). execute( getName() );
            } catch ( Exception e ) {
                Toast . makeText( getBaseContext() ,
```

```

        R.string .SendError ,
        Toast.LENGTH_SHORT) .show () ;
    }
}
}) ;
}

```

Listing 13: Konfiguracja w klasie CustomButton

opisac co tu sie dzieje

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 zostatnie użyta technologia zmiany tła za pomocą metody backgroundResource. Różne wyglądy przycisku mogą być definiowane jako selectory (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="#9999"
                  android:angle="270" />
    </shape>
</item>
</selector>
```

Listing 14: Przykład zdefiniowanego tła

Wymaganiem była jednoczenna 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 15: Zmiana tła dla wszystkich przycisków

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

Listing 16: Przykład wywołania zmiany tła przycisków.

6.3 Informacje tekstowe

Pola tekstowe mogą mieć ustawioną dowolną treść w dowolnym czasie.

6.4 Komunikacja sieciowa

tutaj opisać implementacje warstwy komunikacyjnej

7 Środowisko uruchomieniowe

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

7.1 Serwer

Serwer został uruchomiony na komputerze przenośnym(laptop) posiadającym kartę graficzną umożliwiającą połączenie dwóch zewnętrznych ekranów - projektów. Pierwszy z nich został połączony za pomocą złącza cyfrowego HDMI, natomiast drugi łączem DVI.

7.2 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 opracyjnej 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 Cardboard¹⁸. Są to niskobudżetowe okulary stworzone przez firmę Google do wyświetlania wirtualnej rzeczywistości. Kartonowe okulary powstawały 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 założona przez obudowę. Dzięki temu można użyć platformy Cardboard przeznaczoną pierwotnie tylko do wirtualnej rzeczywistości do stworzenia aplikacji wykorzystującej augmented reality.



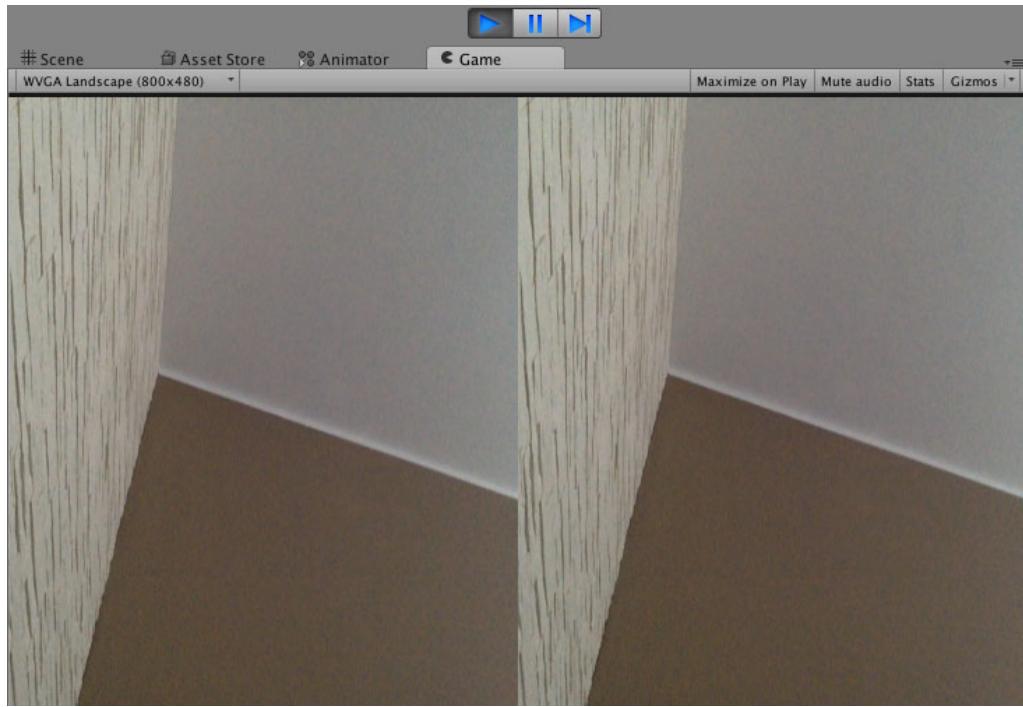
Rysunek 10: Google Cardboard w wersji do samodzielnego złożenia

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).

¹⁸<https://vr.google.com/cardboard/index.html>



Rysunek 11: 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 rodzaju pierwszego [uzupelić rozdział](#). Do platformy Unity doinstalowano zewnętrzny komponent ARToolKit¹⁹. Jest to biblioteka wydana przez University of Washington²⁰, lecz obecnie upostępniona jest na licencji GNU. Kod źródłowy jest otwarty i rozwijany przez środowisko Open Source²¹.

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 pod jakim kątem padania oraz w jakiej odległości od urządzenia znajduje się marker. Umiejscowienie tagu analizowane jest w czasie rzeczywistym, co zapewni ciągłą

¹⁹<http://artoolkit.org/>

²⁰<https://www.hitl.washington.edu/artoolkit/>

²¹<https://github.com/artoolkit>

korekcję ułożenia wirtualnych modeli względem ich realnych odpowiedników.



Rysunek 12: 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.²².

²²<http://bit.ly/1YU199f>

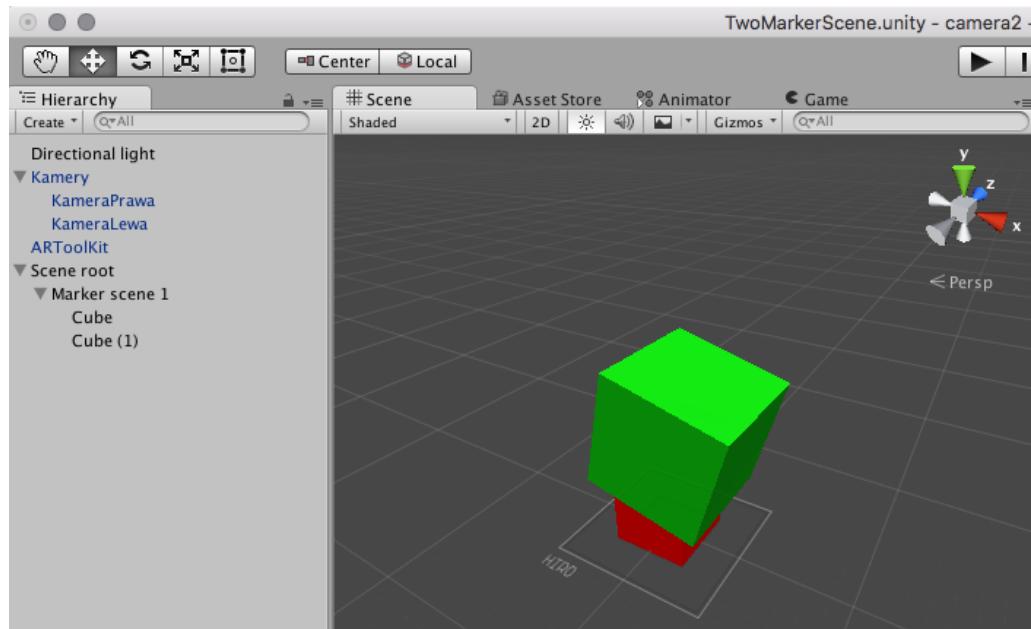


Rysunek 13: Przykład zastosowania markera w ARToolkit

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

8.3 Przykład implementacji

Opisać



Rysunek 14: Podstawowa implemntacja

źródło własne

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 należałoby rozważyć połączenie telefonu z zewnętrznym kontrolerem typu PAD²³.

²³<https://pl.wikipedia.org/wiki/Gamepad>

9 Inne implementacje - Project Tango

Kolejnym przykładem implementacji może być platforma Google Project Tango²⁴. Jest to platforma rozszerzonej rzeczywistości zapoczątkowana przez Johny'ego Lee (współtwórcy między innymi Microsoft Kinect²⁵) 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 Unity²⁶.

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

²⁴<https://get.google.com/tango/>

²⁵[https://en.wikipedia.org/wiki/Johnny_Lee_\(computer_scientist\)](https://en.wikipedia.org/wiki/Johnny_Lee_(computer_scientist))

²⁶<https://developers.google.com/tango/apis/unity/>



Rysunek 15: 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

Napisać o tym, ze Microsoft otworzył platforme hololens, wiec w przyszlosci będą nowe urządzenia

Spis rysunków

1	Ewolucja interfejsów użytkownika	5
2	Wizualizacja Microsoft HoloLens	6
3	Labolatorium PJATK	8
4	Poglądowy schemat działania	10
5	Model w środowisku Unity	16
6	Ujęcie wirtualnych kamer	18
7	Konfiguracja replikatora	22
8	Makieta - układ przycisków	27
9	Przykładowa wizualizacja kontrolera	28
10	Google Cardboard w wersji do samodzielnego złożenia	33
11	Ujęcie z kamery aparatu w obrazie stereoskopowym - scena w Unity	34
12	Przykład przygotowanego obrazka do rozpoznawania - marker	35
13	Przykład zastosowania markera w ARToolkit	36
14	Podstawowa implementacja	36
15	Prototyp urządzenia	39

Spis tablic

1	Porównanie silników gier	12
2	Właściwości aktora	16

Listings

1	Do Poprawy	17
2	Prosta aktywacja ekranów	19
3	Inicjalizacja skryptu	20
4	Definicja przycisków.	21
5	Inicjalizacja komponentów	21
6	Przykładowe zapytanie do gry - kontroler	22
7	Uruchomienie replikatora	23
8	Implementacja akcji	24

9	Definicja zależności w projekcie	25
10	Grupowanie połączeń	26
11	Przykład zdefiniowanego przycisku	28
12	Definicja atrybutu o nazwie name	29
13	Konfiguracja w klasie CustomButton	30
14	Przykład zdefiniowanego tła	31
15	Zmiana tła dla wszystkich przycisków	31
16	Przykład wywołania zmiany tła przycisków.	31

Literatura

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