



POLSKO-JAPOŃSKA 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

Warszawa, 27 maja 2016

Spis treści

1 Wprowadzenie	4
1.1 Rozszerzona rzeczywistość	6
2 Cel pracy	7
3 Implementacje	8
3.1 Implementacja w technologii hologramu	8
3.2 Implementacja w technologii mappingu 3d	9
4 Wykorzystane technologie	10
4.1 Unity	10
4.1.1 Dlaczego Unity	10
4.1.2 Alternatywne rozwiązania	10
4.1.3 Wybór języka programowania	10
4.1.4 Unity IDE	11
4.1.5 Licencja i koszty	11
4.2 Android	11
4.2.1 Android Studio	12
4.2.2 Zależności	12
4.3 Komunikacja sieciowa	12
4.3.1 UNET	12
4.3.2 HTTP (SOAP, REST)	12
4.3.3 TCP	13
4.3.4 UDP	13
5 Aplikacja główna	14
5.1 Świat gry	14
5.2 Aktorzy	14
5.3 Kamery	15
5.4 Prefabrykaty	15
5.4.1 Kamery	15
5.4.2 Światło	17
5.4.3 Network	17
5.4.4 Replikator	18
5.5 Logika biznesowa	20
5.6 Serwer komunikacyjny	20

6 Aplikacja mobilna - kontroler	20
6.1 Przyciski	20
6.2 Informacje tekstowe	22
6.3 Komunikacja sieciowa	22
7 Środowisko uruchomieniowe	22
7.1 Serwer	22
7.2 Aplikacja mobilna - kontroler	22
8 Implementacja na platformie Google Cardboard	23
8.1 ArToolkit	24
8.2 Markery	24
8.3 Przykład implementacji	26
8.4 Napotkane problemy i ograniczenia	27
9 Dalszy rozwój	28

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.

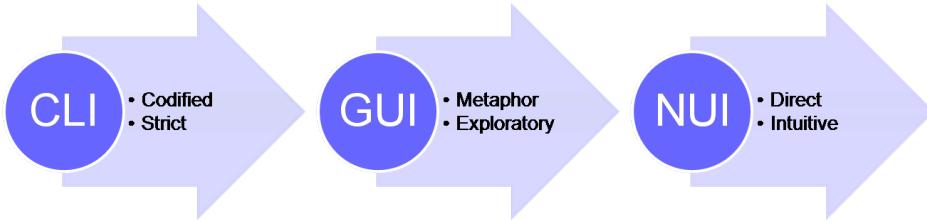
Przez ostatnie kilka lat narasta trend poszukiwania innych metod dostępu do danych, zwłaszcza multimedialnych. Obecnie wiele przedsiębiorstw prowadzi badania nad nowymi, bardziej naturalnymi dla ludzi interfejsami, które nie wymagałyby użycia standarodwych(sztucznych) urządzeń wejścia typu klawiatura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

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>

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 odpowiadają 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.

Meta2, Wonderbook - opisać przykłady

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>



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 budowania gier oraz interaktywnych animacji prezentowanej za pomocą rozszerzonej rzeczywistości sterowanej za pomocą zdalnego kontrolera. Praca zawiera przykłady różnych implementacji w technologiach z rodzin AR.

Przykłady zastosowanie zestawu aplikacji:

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

3 Implementacje

Rozdział ukazujący różne podejścia do stworzenia implementacji sceny w w technologii rozszerzonej rzeczywistości. Jako środowisko badawcze wybrano salę - labolatorium znajdująca się w PJATK. Salę tę wybrano ponieważ znajduje się w podpiwniczeniu, co za tym idzie ilość światła dziennego jest znikoma, lecz wystarczająca do zastosowania urządzeń projekcyjnych w ciągu dnia. Dodatkowo w sali znajduje się rura cieplownicza 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

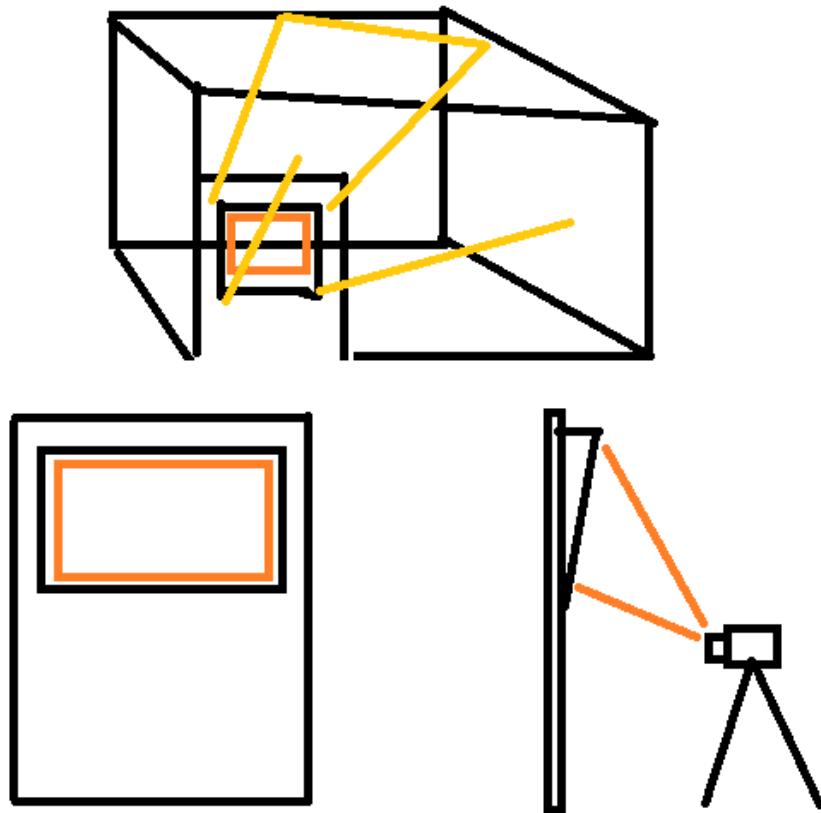
Główym założeniem było stworzenie minigry opartej na grze z początku lat dziewięćdziesiątych o nazwie Lemmingi¹¹

Tutaj opisać logikę gry

3.1 Implementacja w technologii hologramu

¹¹<https://en.wikipedia.org/wiki/Lemmings>

Tutaj opisać próby i przemyślenia na temat hologramu na drzwiach.



Rysunek 4: Poglądowy schemat działania
źródło: własne

3.2 Implementacja w technologii mappingu 3d

Tutaj opisać próby i przemyślenia

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. 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 kołopoty ze środowiskiem MonoDevelop spowodowały decyzje o przenesieniu 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 mobilne(systemy: iOS, Android, Windows Phone, Tizen itp¹³). Jednakże Unity w wersji Personal Edition nie pozwala na uruchomienie warstwy sieciowej na urządzeniach mobilnych.

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

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

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.

4.2.1 Android Studio

opisac

4.2.2 Zależności

1. ButterKnife

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 oznaczało to 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

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

niezbyt dobrze sprawdza się przy małych, lecz częstych połączeniach pomiędzy klientem, a serwerem. Duży narzut czasowy może spowodować transformację 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.

4.3.3 TCP

Opisać, że TCP jest ogólnie lepsze - socket, ale to jest nadal połączeniowy, więc lepiej by było udp

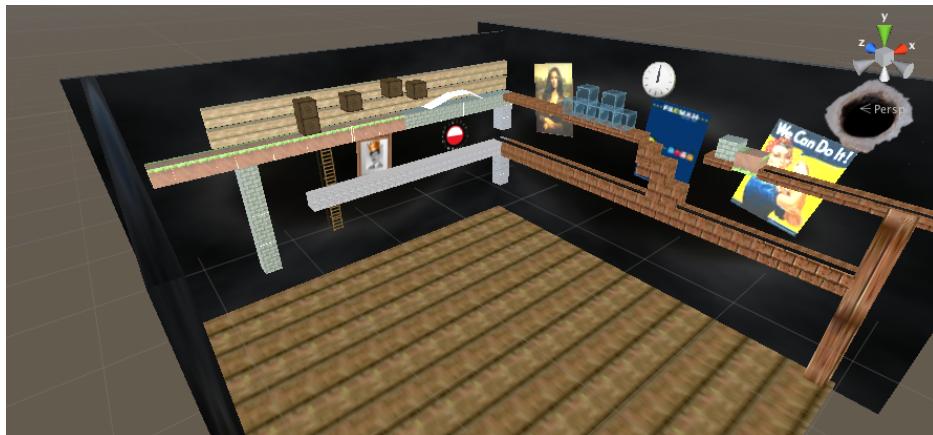
4.3.4 UDP

opisać, ze to najlepsze rozwiązanie - bezpołazeniowe

5 Aplikacja główna

tutaj będą makiety

5.1 Świat gry



Rysunek 5: Model w środowisku Unity
źródło: własne

5.2 Aktorzy



Tablica 2: Właściwości aktora

```
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 Kamery

Platforma Unity wspiera do 8 wirtualnych kamer¹⁵. Każda z tych kamer może być prezentowana na oddzielnym fizycznym ekranie.

5.4 Prefabrykaty

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

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

5.4.1 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 wyświetlane 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.

¹⁵<http://docs.unity3d.com/Manual/MultiDisplay.html>

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 skonfigurowanej 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.

5.4.2 Światło

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

5.4.3 Network

Jest to prefabrykat abstrakcyjny (nie posiada graficznej reprezentacji w trójwymiarowym modelu). Służy on jako trigger uruchomienowy do skryptu obsługującego połączenia sieciowe.

Opisać logikę, network manager itd.

5.4.4 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" , [REDACTED]
            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 3: Uruchomienie replikatora

```

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

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

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

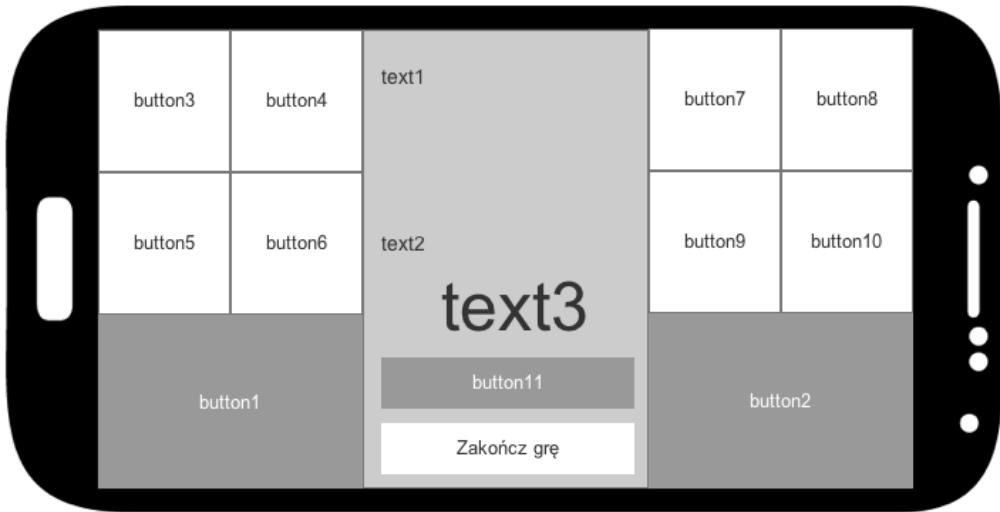
```

Listing 4: Implementacja akcji
Opisać pozostałe prefabrykaty

5.5 Logika biznesowa

5.6 Serwer komunikacyjny

6 Aplikacja mobilna - kontroler



Rysunek 8: Makieta - układ przycisków

Głównym 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.

6.1 Przyciski

Każdy przycisk może zostać skonfigurowany poprzez ustawienie tekstu. Dodatkowo można zablokować przycisk podczas gdy nie jest on potrzebny w danym czasie.

Aby ułatwić pracę nad aplikacją stworzono pole wyliczalne (ENUM) zawierającą wszystkie przyciski.

```
public enum ButtonEnum {
    BTN1, BTN2, BTN3, BTN4, BTN5, BTN6, BTN7, BTN8,
    BTN9, BTN10, BTN11
}
```

Listing 5: Enum z przyciskami

Jednakże aby powiązać elementy „Button” z warstwy widoku (definicja XML) na kod przycisku należy stworzyć listę elementów. Będzie ona służyła do wyszukiwania przycisków oraz zmiany ich właściwości. Dodatkowo do każdego przycisku należy dodać obsługę zdarzeń. Po kliknięciu wysyłany będzie odpowiedni komunikat.

```
protected HashMap<ButtonEnum, Button> buttonsMap =  
    new HashMap<ButtonEnum, Button>();  
  
protected void mapButton(int btnId, final  
    ButtonEnum btnName) {  
    final Button button = (Button)  
        findViewById(btnId);  
    buttonsMap.put(btnName, button);  
  
    button.setOnClickListener(new  
        View.OnClickListener() {  
            public void onClick(View v) {  
                sendToServer("button_" + btnName);  
            }  
        });  
}
```

Listing 6: Metoda obsługująca przycisk

```
@Override  
protected void onCreate(Bundle savedInstanceState)  
{  
    super.onCreate(savedInstanceState);  
  
    this.mapButton(R.id.button1, ButtonEnum.BTN1);  
    this.mapButton(R.id.button2, ButtonEnum.BTN2);  
}
```

}

Listing 7: Przykład mapowania przycisku

6.2 Informacje tekstowe

Pola tekstowe mogą mieć ustawioną dowolną treść w dowolnym czasie.

6.3 Komunikacja sieciowa

tutaj opisać implementację 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 połaczenie dwóch zewnętrznych ekranów - projektorów. 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 Implementacja na platformie Google Cardboard

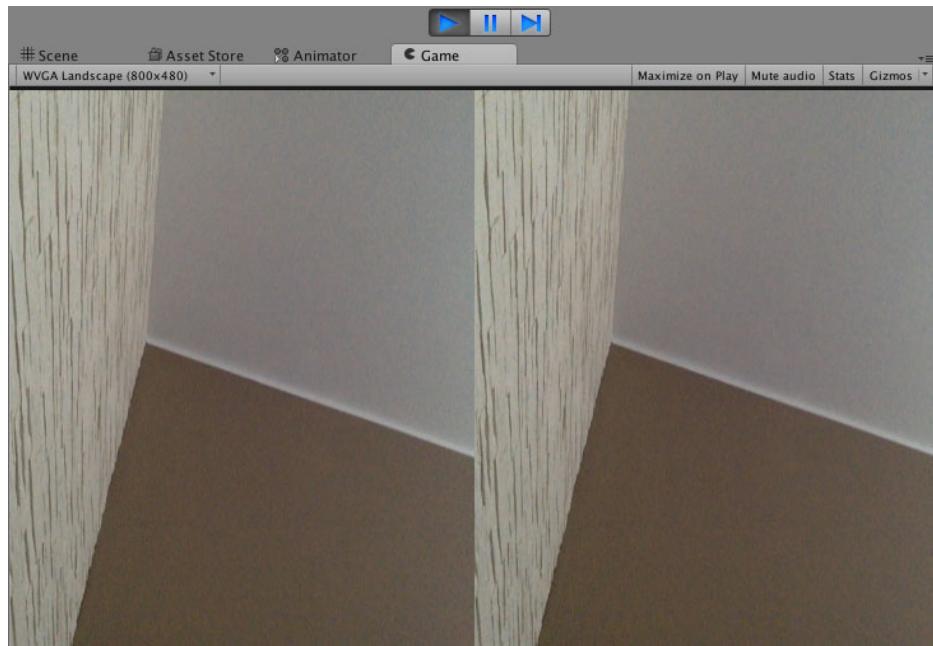
Innym przykładem implementacji projektu w rozszerzonej rzeczywistości może być platforma Google Cardboard¹⁷. Są to niskobudżetowe okulary stworzone przez firmę Google do wyświetlania wirtualnej rzeczywistości. Kartonowe okulary powstawały w celu wyświetlania obrazu stereoskopowego. Posiadają one miejsce 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 platformę Cardboard przeznaczoną pierwotnie tylko do wirtualnej rzeczywistości do stworzenia aplikacji wykorzystującą augmented reality.



Rysunek 9: 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).

¹⁷<https://vr.google.com/cardboard/index.html>



Rysunek 10: 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 [uzupełić rozdział](#). Do platformy Unity doinstalowano zewnętrzny komponent ARToolkit¹⁸. Jest to biblioteka wydana przez University of Washington¹⁹, lecz obecnie upostępniona jest na licencji GNU. Kod źródłowy jest otwarty i rozwijany przez środowisko Open Source²⁰.

8.2 Markery

Za pomocą tej biblioteki możliwe jest wykrywanie w obrazie z kamery markerów, czyli specjalnie przygotowanych czarno-białych obrazków (w najowej implementacji - wydrukowanych na kartkach), oraz nakładanie w ich miejsce

¹⁸<http://artoolkit.org/>

¹⁹<https://www.hitl.washington.edu/artoolkit/>

²⁰<https://github.com/artoolkit>

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 korekcję ułożenia wirtualnych modeli względem ich realnych odpowiedników.



Rysunek 11: Przykład przygotowanego obrazka do rozpoznawania - marker

Szablony markerów można wykonywać we własnym zakresie. Aby zaimportować nowe obrazki do biblioteki ArToolkit należy przygotować specjalny plik binarny reprezentujący model markera.²¹.

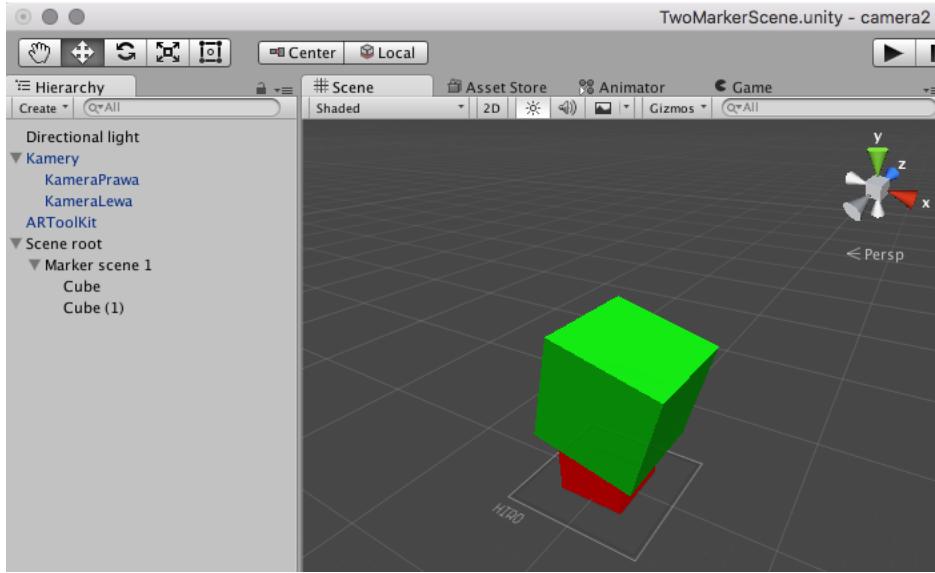
²¹<http://bit.ly/1YU199f>



Rysunek 12: Przykład zastosowania markera w ARToolkit
źródło <http://arblog.inglobetechnologies.com/?p=421>

8.3 Przykład implementacji

Opisać prefabrykaty



Rysunek 13: Podstawowa implementacja
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 telefona z zewnętrznym kontrolerem typu PAD²².

²²<https://pl.wikipedia.org/wiki/Gamepad>

9 Dalszy rozwój

Opisać

Spis rysunków

1	Ewolucja interfejsów użytkownika	5
2	Wizualizacja Microsoft HoloLens	7
3	Labolatorium PJATK	8
4	Poglądowy schemat działania	9
5	Model w środowisku Unity	14
6	Ujęcie wirtualnych kamer	16
7	Konfiguracja replikatora	18
8	Makieta - układ przycisków	20
9	Google Cardboard w wersji do samodzielnego złożenia	23
10	Ujęcie z kamery aparatu w obrazie stereoskopowym - scena w Unity	24
11	Przykład przygotowanego obrazka do rozpoznawania - marker	25
12	Przykład zastosowania markera w ARToolkit	26
13	Podstawowa implementacja	27

Spis tabel

1	Porównanie silników gier	10
2	Właściwości aktora	14

Listings

1	Do Poprawy	14
2	Prosta aktywacja ekranów	17
3	Uruchomienie replikatora	19
4	Implementacja akcji	20
5	Enum z przyciskami	21
6	Metoda obsługująca przycisk	21
7	Przykład mapowania przycisku	22

Literatura

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