



POLITECHNIKA WARSZAWSKA  
WYDZIAŁ MATEMATYKI I NAUK INFORMACYJNYCH



PRACA DYPLOMOWA INŻYNIERSKA  
NA KIERUNKU INFORMATYKA

**WIELOOSOBOWA GRA 3D NA PLATFORMY MOBILNE  
W ARCHITEKTURZE Klient-SERWER**

MULTIPLAYER 3D MOBILE GAME IN CLIENT-SERVER  
ARCHITECTURE

AUTORZY:  
DOMINIKA BODZON  
EMILIA SZYMAŃSKA

PROMOTOR:  
DR INŻ. PAWEŁ KOTOWSKI

WARSZAWA, LUTY 2015

.....  
podpis promotora

.....  
podpisy autorów

# Streszczenie

Niniejsza praca dyplomowa miała na celu stworzenie mobilnej gry trójwymiarowej opartej na architekturze klient-serwer. W rezultacie powstała gra Sculpic, która jest wersją popularnej gry kalambury opartą na grafice trójwymiarowej oraz powiązany z nią serwer. Gracze, zamiast rysować na płaszczyźnie 2D mają do dyspozycji trójwymiarowe bryły, w których mogą dowolnie rzeźbić. Osoby próbujące odgadnąć hasło mogą oglądać aktualny stan sceny ze wszystkich stron.

Sculpic jest aplikacją na urządzenia z systemem operacyjnym Android. Została ona wykonana przy użyciu silnika do tworzenia gier Unity3D, wykorzystującego napisane przez nas skrypty w języku C#.

Serwer obsługujący grę został umiejscowiony na maszynie wirtualnej stworzonej na platformie Azure. Jako baza danych zostało użyte MongoDB. Baza danych przechowuje informacje o użytkownikach (m.in. login, hash hasła, aktualny ranking) oraz listę dostępnych fraz do zgadywania podczas rozgrywki. Komunikacja z nią jest możliwa poprzez serwis WCF oparty na architekturze SOA (Service Oriented Architecture) przy pomocy zapytań RESTowych. Dodatkowo, na maszynie wirtualnej działa usługa, która na żądanie gracza uruchamia aplikację służącą jako pokój, który jest miejscem rozgrywki.

Podczas tworzenia aplikacji największych problemów przysporzyło nam:

- tworzenie pokoi, które byłyby dostępne dla wszystkich użytkowników niezależnie od sieci, w której się znajdują,
- przesyłanie aktualnego stanu brył na scenie od gracza rysującego do graczy zgadujących,
- wywoływanie zapytań RESTowych z aplikacji stworzonej w Unity3D.

Sculpic będzie dostępny w sklepie Google Play na platformie Android.



# Abstract

This engineering thesis was aimed to create a 3-dimensional mobile game based on server-client architecture. As a result Sculpic and game server was created. The game is based on widely-known Pictionary but with 3-dimensional graphics. Players instead of drawing on a 2-dimensional plane can use a variety of solids by placing them in 3-dimensional space and sculpting. Players trying to guess a phrase being shown can inspect the scene from any possible angle.

Sculpic is an application for devices with Android operating system. It's been built with game engine Unity 3D using our C# scripts.

The game server has been placed in virtual machine created in Azure platform. As a database we have used MongoDB. The database keeps user information (e.g. username, password hash, current ranking) and a list of available phrases. The communication with database is possible thanks to WCF service based on SOA (Service Oriented Architecture) by RESTful requests. What is more, the virtual machine contains a service which launches a room hosting application on player demand.

While creating the application we have come by some major problems:

- hosting rooms available for all players no matter what network they are in,
- sending current solid state from drawing player scene to the guessing players scenes,
- sending RESTful requests from Unity3D application.

Sculpic will soon be available from Google Play store on Android.



# Spis treści

<b>Streszczenie</b>	<b>3</b>
<b>Abstract</b>	<b>5</b>
<b>1. Wstęp</b>	<b>9</b>
1.1. Cel projektu . . . . .	9
1.2. Motywacja . . . . .	9
1.3. Zakres projektu . . . . .	10
1.3.1. Gra . . . . .	10
1.3.2. Serwer . . . . .	10
1.4. Podział pracy . . . . .	10
<b>2. Specyfikacja wymagań</b>	<b>13</b>
2.1. User stories . . . . .	13
2.1.1. Jako gracz... . . . . .	13
2.1.2. Jako właściciel pokoju... . . . . .	14
2.1.3. Jako rysujący... . . . . .	14
2.1.4. Jako zgadujący... . . . . .	15
<b>3. Technologie</b>	<b>17</b>
3.1. WCF . . . . .	17
3.2. Unity . . . . .	17
3.2.1. Master Server . . . . .	18
3.3. MongoDB . . . . .	18
3.4. JSON . . . . .	18
3.5. Azure . . . . .	19
<b>4. Architektura projektu</b>	<b>21</b>
4.1. Opis komponentów . . . . .	21
4.1.1. Sculpic . . . . .	22
4.1.2. Azure - maszyna wirtualna . . . . .	22
4.2. Dziedzina problemu . . . . .	23
4.3. Struktura bazy danych . . . . .	24
4.4. Wzorce projektowe . . . . .	25
4.4.1. Singleton . . . . .	25
4.4.2. Repozytorium . . . . .	25
4.4.3. SOA . . . . .	26
<b>5. Implementacja</b>	<b>27</b>
5.1. Projekt w Unity . . . . .	27

---

5.1.1. Graficzny interfejs użytkownika . . . . .	27
5.1.2. Rozgrywka . . . . .	27
5.1.3. Dotyk . . . . .	30
5.1.4. Punktacja . . . . .	30
5.1.5. Komunikacja . . . . .	30
5.1.6. Pokój . . . . .	31
5.2. Serwer bazodanowy . . . . .	31
5.2.1. Dostęp do bazy danych . . . . .	32
5.2.2. Serwer WCF . . . . .	32
5.2.3. Server Host . . . . .	35
<b>6. Opis aplikacji</b>	<b>37</b>
6.1. Ekran logowania . . . . .	37
6.2. Ekran rejestracji . . . . .	38
6.3. Ekran wyboru pokoju . . . . .	39
6.4. Ekran gracza zgadującego . . . . .	41
6.5. Ekran gracza rysującego . . . . .	42
6.6. Ekran rankingu . . . . .	43
<b>7. Testy</b>	<b>45</b>
7.1. Testy jednostkowe . . . . .	45
7.2. Testy integracyjne . . . . .	45
<b>8. Podsumowanie i wnioski</b>	<b>49</b>
8.1. Napotkane problemy . . . . .	49
8.1.1. Przesyłanie brył . . . . .	49
8.1.2. Pokoje . . . . .	50
8.1.3. Wywoływanie zapytań RESTowych . . . . .	50
8.2. Wnioski . . . . .	51
<b>Literatura</b>	<b>53</b>

# Rozdział 1

## Wstęp

### 1.1. Cel projektu

Niniejsza praca dyplomowa miała na celu stworzenie trójwymiarowej mobilnej gry wieloosobowej z towarzyszącą jej architekturą klient-serwer. W ramach projektu zaimplementowana została gra na system operacyjny Android oraz serwer, który ją obsługuje.

Gra została nazwana Sculpic. Jest ona bardzo podobna do popularnej gry Kalambury – z tą różnicą, że rozgrywka odbywa się w przestrzeni trójwymiarowej. Aplikacja jest w angielskiej wersji językowej.

Serwer został umieszczony na maszynie wirtualnej znajdującej się na platformie Azure. Głównymi jego elementami jest baza danych MongoDB, serwis WCF do komunikacji z bazą danych oraz usługa tworząca pokoje.

Praca ta zawiera szczegółowy opis wszystkich komponentów projektu, ich wzajemnych zależności, procesu ich powstawania oraz uzasadnienia podjętych przez nas decyzji dotyczących problemów podczas implementacji.

### 1.2. Motywacja

Pomysł na grę został zainspirowany projektem, który stworzyliśmy podczas zajęć na uczelni w ramach przedmiotu Laboratorium CAD/CAM. Był to Rzeźbiarz figur w 3D wykorzystujący Unity3D. Użytkownik mógł dowolnie modyfikować figury (robiąc wklęśności i wypukłości o wybranym rozmiarze i ostrości) przy pomocy kliknięć myszki w jej obrębie.

Po zapoznaniu się z aplikacjami dostępnymi na platformy mobilne dostrzegliśmy niszę – nigdzie nie było dostępnych kalambur w 3D.

W związku z powyższym, postanowiliśmy wykorzystać doświadczenie zdobyte przy wcześniejszym projekcie i połączyć Kalambury z Rzeźbiarzem dając graczom nowe doświadczenie w postaci gry, której zasady są bardzo proste i wszystkim znane. Nazwa gry Sculpic wynika z połączenia słów „sculptor” (z ang. rzeźbiarz) oraz „Pictionary” (z ang. Kalambury).

## 1.3. Zakres projektu

### 1.3.1. Gra

#### Opis rozgrywki

Gra polega na zgadywaniu haseł przedstawionych przez gracza rysującego przy użyciu dostępnych figur i wykonanego na nich rzeźbienia (modyfikacji). Graczom, po dołączeniu do jednego z pokojów, pojawia się ekran gracza zgadującego. Gdy w grze znajdują się co najmniej dwaj gracze i co najmniej jeden z nich zaznaczy pole *I want to draw* (czyli dołączył do kolejki graczy rysujących), pierwszy gracz z kolejki rysujących uzyskuje możliwość rysowania. Po narysowaniu wylosowanej frazy, gracz rysujący przesyła do reszty graczy swoje dzieło. Gracze zgadujący mają stały dostęp do czatu, przy użyciu którego mogą przez cały czas trwania rozgrywki się ze sobą komunikować oraz wpisywać swoje propozycje zgadywanego hasła.

Jeżeli podczas ograniczonego czasu trwania jednej rundy (5 minut), któryś z graczy odgadnie hasło – on oraz gracz rysujący dostają odpowiednie ilości punktów (zależne od ilości czasu, który upłynął od początku rysowania). W przeciwnym wypadku, po upłynięciu czasu, gracz rysujący dostaje punkty karne. Następnie, niezależnie od tego, czy hasło zostało odgadnięte, czy też nie, losowana jest kolejna fraza oraz prawo do rysowania dostaje kolejny gracz z kolejki do rysowania.

Gra toczy się aż któryś z graczy nie osiągnie maksymalnej liczby punktów. W tym momencie gracza z największą liczbą punktów uznaje się za zwycięzcę gry. Odpowiednio do miejsc, które gracze zajęli w punktacji podczas gry, zmieniany jest ich ogólny ranking.

### 1.3.2. Serwer

Integrację aplikacji klienckich Sculpica zapewnia Master Server, który zajmuje się zestawianiem połączeń pomiędzy graczami oraz serwer bazodanowy w technologii WCF. Opis tych komponentów i ich architekury znajduje się w dalszych rozdziałach.

## 1.4. Podział pracy

Praca nad pracą inżynierską została podzielona w następujący sposób:

### 1. Dominika Bodzon

- implementacja sieciowej rozgrywki oraz modelowania w czasie rzeczywistym
- obsługa dotyku podczas rozgrywki
- implementacja logiki rankingu

### 2. Emilia Szymańska

- konfiguracja maszyny wirtualnej i komponentów na niej umieszczonych
- implementacja serwera bazodanowego oraz modułu klienta do komunikowania się z nim
- stworzenie graficznego interfejsu użytkownika

Oprócz tego każda z nas napisała testy jednostkowe do funkcjonalności, które implementowałyśmy w Web Service i opisała pracę, którą wykonała nad projektem w niniejszej pracy inżynierskiej.



## Rozdział 2

# Specyfikacja wymagań

### 2.1. User stories

Aktorzy: gracz, rysujący, zgadujący, właściciel pokoju

#### 2.1.1. Jako gracz...

	chce...	aby...
1	mieć możliwość zarejestrowania konta	mieć wyłączny dostęp do swoich osiągnięć w grze
2	mieć możliwość zalogowania się na swoje konto	grać pod swoim nickiem i mieć dostęp do swoich osiągnięć
3	by moje konto było zapamiętane w grze	nie musieć się logować przy każdym uruchomieniu gry
4	mieć możliwość założenia pokoju do gry	móc samemu wybrać jego ustawienia
5	mieć możliwość dołączenia do wybranego przeze mnie pokoju	rozpocząć grę w pokoju, w którym są moi znajomi
6	mieć możliwość dołączenia do losowego pokoju	móc nie podejmować decyzji o wyborze pokoju
7	mieć możliwość zmienienia ustawień dźwięku	dopasować je do moich preferencji
8	mieć możliwość obejrzenia rankingu	porównać swoje osiągnięcia z innymi graczami
9	mieć możliwość sprawdzenia, kto jest twórcami gry	móc znaleźć inne projekty tych osób
10	mieć możliwość wyjścia z gry	zakończyć grę
11	słyszeć dźwięki podczas gry	być informowanym o zdarzeniach w Grze

### 2.1.2. Jako właściciel pokoju...

	chcę...	aby...
1	mieć możliwość ustalenia limitu graczy w pokoju	uniknąć sytuacji w której zbyt długo czeka się na swój ruch
2	mieć możliwość założenia hasła do pokoju	mieli do niego dostęp jedynie gracze, którym podam hasło
3	mieć możliwość rozpoczęcia rozgrywki w Pokoju	zacząć grę
4	mieć możliwość nadania Pokojowi nazwy	był rozpoznawalny pośród innych

### 2.1.3. Jako rysujący...

	chcę...	aby...
1	poznać hasło, które mam rysować na początku rysowania	wiedzieć, co rysować
2	mieć możliwość tworzenia brył na scenie	mieć kształt początkowy
3	mieć możliwość wybierania kształtu brył, które tworzę	tworzyć bardziej zróżnicowane konstrukcje
4	mieć możliwość wybierania koloru brył, które tworzę	tworzyć bardziej zróżnicowane konstrukcje
5	mieć możliwość przesuwania brył po scenie	tworzyć bardziej zróżnicowane konstrukcje
6	mieć możliwość wyczyszczenia sceny	móc zacząć rysować od początku
7	mieć możliwość obracania Sceny	móc obejrzeć rysunek z każdej strony
8	mieć możliwość tworzenia wkładnięć w bryłach	tworzyć bardziej zróżnicowane konstrukcje
9	mieć możliwość tworzenia wznieśnień na bryłach	tworzyć bardziej zróżnicowane konstrukcje
10	mieć możliwość dostosowywania promienia i ostrości tworzonych wkładnięć i wznieśnień	tworzyć bardziej zróżnicowane konstrukcje
11	widzieć czat	wiedzieć, czy mój rysunek naprowadza zgadujących na dobry trop
12	zakończyć rysowania tak szybko jak ktoś zgadnie hasło	runda mogła się skończyć
13	mieć ograniczony czas rysowania	nie rysować w nieskończoność

**2.1.4. Jako zgadujący...**

	<b>chce...</b>	<b>aby...</b>
1	mieć możliwość obracania sceny niezależnie od rysującego	móc się przyjrzeć rysowanym bryłom
2	mieć możliwość pisania swoich odpowiedzi na chacie	serwer mógł określić, czy pasują one do zgadywanego hasła
3	widzieć chat	poznać odpowiedzi innych graczy
4	widzieć punkty swoje i innych graczy znajdujących się w pokoju	był wiedzieć jakie są moje szanse na wygraną



## Rozdział 3

# Technologie

### 3.1. WCF

Windows Communication Foundation jest to framework służący do tworzenia aplikacji zoorientowanych na usługi. Przy jego pomocy możliwe jest asynchroniczne przesyłanie danych w postaci wiadomości z jednego punktu dostępowego do drugiego. Adresatem wiadomości może być aplikacja kliencka bądź inny serwis. Możliwe są różne sposoby przesyłu wiadomości – najprostszym jest model zapytanie → odpowiedź, który jest wykorzystywany w naszym serwerze. WCF pozwala też na łatwe stworzenie architektury RESTowej poprzez umożliwienie tworzenia szablonów zapytań oraz definiowania routingu.

Na WCFie oparty jest serwer bazodanowy i w tym zastosowaniu sprawdza się bardzo dobrze, działa stabilnie. Koniecznie było jednak użycie innego rozwiązania do obsługi wiadomości przesyłanych między graczami podczas gry (takich jak na przykład zmiana kształtu bryły), gdyż narzucałyby wydajność aplikacji.

Serwer ma postać prostej aplikacji konsolowej, która hostuje odpowiednie usługi (tzw. self-hosted service).

### 3.2. Unity

Unity jest rozbudowanym środowiskiem do tworzenia gier w 2D i 3D stworzonym przez Unity Technologies. Łączy ono w sobie intuicyjny zestaw narzędzi oraz silnik gry. Umożliwia produkowanie aplikacji na strony internetowe, komputery stacjonarne, konsole i urządzenia mobilne.

Gry stworzone w Unity z łatwością można portować na różne platformy jednocześnie. To był dla nas ważny argument, gdy wybierałyśmy silnik gry, ponieważ Sculpic został stworzony z myślą o platformach mobilnych. Z powodów sprzętowych i kosztowych zdedydowałyśmy się ostatecznie testować naszą grę na platformie Android.

Głównym elementem gier stworzonych w Unity są skrypty, które zawierają całą logikę gry. Unity umożliwia ich pisanie w kilku językach (UnityScript, JavaScript, C# i Boo). My zdecydowałyśmy się na język C#, ponieważ bardzo dobrze go znamy. Dało to nam również możliwość wykorzystania Visual Studio jako środowiska programistycznego.

W pracy nad naszą aplikacją wykorzystałyśmy Unity w wersji 4.6.

### 3.2.1. Master Server

Master Server jest aplikacją stworzoną przez Unity Technologies, by ułatwić pracę programistom tworzącym w Unity gry sieciowe. Jest to serwer, który zajmuje się ruchem sieciowym pomiędzy aplikacjami klienckimi. Server ten pozwala użytkownikowi założyć pokój gry, a następnie zajmuje się graczami, którzy chcą do niego dołączyć. Zestawia on połączenia pomiędzy poszczególnymi aplikacjami, by mogły się one ze sobą wymieniać informacjami.

Zdecydowałyśmy się wykorzystać Master Server w naszym projekcie, ponieważ został on napisany specjalnie na potrzeby silnika Unity.

## 3.3. MongoDB

MongoDB jest to dokumentowa baza danych z rodziny NoSQL, charakteryzująca się obiektowym charakterem danych, wysoką skalowalnością i wieloplatformowością.

Decyzja o wykorzystaniu właśnie tej bazy do przechowywania danych użytkowników była podyktowana głównie tym, że nasze modele bazodanowe pasują do dokumentowego schematu – wszystkie dane potrzebne w danym momencie można uzyskać poprzez pobranie rekordu z jednej kolekcji. Dodatkowo, cenna jest również możliwość zmiany modelu bez potrzeby czyszczania całej bazy, a także prostota jej uruchomienia i utrzymania. Jako, iż projekt nie zakładał funkcjonalności, które wymagałyby obsłużenia transakcji bazodanowych, atomowość dostarczona przez MongoDB jest w zupełności wystarczająca. Ponadto jest to dobra okazja do rozwinięcia umiejętności i nauki obsługi bazy danych innego rodzaju niż tradycyjny SQL (Structured Query Language).

Do obsługi zapytań do bazy danych z poziomu serwera wykorzystany jest oficjalny sterownik w języku C# – „Mongo C# Driver” dostępny jako paczka na platformie NuGet.

Do przeglądania zawartości bazy danych używany jest program Robomongo, który jest projektem open-source (<http://robomongo.org/>).

## 3.4. JSON

JavaScript Object Notation jest to lekki format wymiany danych, charakteryzujący się łatwością w czytaniu i pisaniu przez ludzi oraz parsowaniu i generowaniu przez systemy informatyczne. Jest to format tekstowy, uniwersalny względem języka programowania, wykorzystujący konwencje języka JavaScript. Ma strukturę słownikową, dane ułożone są w parach klucz-wartość. W ogólności można do niego zapisać obiekty o dowolnej strukturze (m.in. złożone obiekty, tablice). Jest to format danych coraz bardziej wypierający popularnego format XML (Extensible Markup Language) głównie z powodu mniejszego rozmiaru pliku wyjściowego. W naszym projekcie JSON jest obecny w dwóch miejscach. W serwerze bazodanowym jako format argumentów przesyłanych do usługi oraz postać odpowiedzi zwracanej przez serwer. Ponadto baza danych MongoDB składa się dane na dysku w formacie BSON (Binary JSON) - binarnym odpowiednikiem formatu JSON, przez co zapytania do bazy również mają identyczną strukturę.

W skryptach Unity do serializacji i deserializacji plików o formacie JSON wykorzystywana jest biblioteka JsonFx (<http://www.jsonfx.net/>) udostępniana na licencji open-source.

### 3.5. Azure

Microsoft Azure jest platformą chmurową stworzoną przez firmę Microsoft, która służy do budowania, wdrażania oraz zarządzania aplikacjami i serwisami poprzez globalną sieć centrów danych. Charakteryzuje się łatwością obsługi oraz dużym stopniem zintegrowania z innymi produktami tej firmy (na przykład możliwość publikowania aplikacji webowej za pomocą funkcji wbudowanych w program Visual Studio).

Funkcjonalnością platformy Azure, która została wykorzystana w naszym projekcie jest możliwość prostego tworzenia maszyn wirtualnych z systemem operacyjnym Windows Server. Posłużyła ona za miejsce działania serwera bazodanowego, aplikacji Master Server oraz procesu bazy danych. Dzięki temu usługi są dostępne globalnie i przez całą dobę. Obsługa maszyny wirtualnej odbywa się poprzez połączenie przez Pulpit zdalny. Aby maszyna mogła pełnić rolę serwera, do jej zapory sieciowej musiały zostać dodane wyjątki na kilka portów, między innymi port dostępu do serwera bazodanowego oraz aplikacji MasterServer.

Wirtualna maszyna służy również za miejsce uruchomienia pokojów gry, z których każdy jest oddzielną instancją projektu Unity, której punktem startowym jest specjalna scena o nazwie RoomHoster. Każdy z nich musi działać na innym porcie, ponieważ w przeciwnym przypadku proces nasłuchujący dostawałby komunikaty przeznaczone dla różnych pokojów. Powoduje to konieczność odblokowania dużego zakresu portów w zaporze sieciowej systemu.

Aby serwisy były w pełni dostępne z punktu widzenia świata zewnętrznego, należy również ustawić obsługę przekierowania portów w portalu do zarządzania platformy Azure. Z poziomu portalu do zarządzania maszynami wirtualnymi możliwe jest jedynie ręczne, pojedyncze dodawanie portów, zatem dodanie dużej ich ilości zajmowałoby sporo czasu. W związku z tym napisany został skrypt, który po uruchomieniu przez Azure PowerShell pozwolił na automatyczne otwarcie 100 kolejnych portów. Ze względów finansowych konfiguracja maszyny jest dość średnio wydajna (1 rdzeń procesora, niecałe 2 GB pamięci RAM), co ogranicza ilość działających procesów hostujących pokoje.

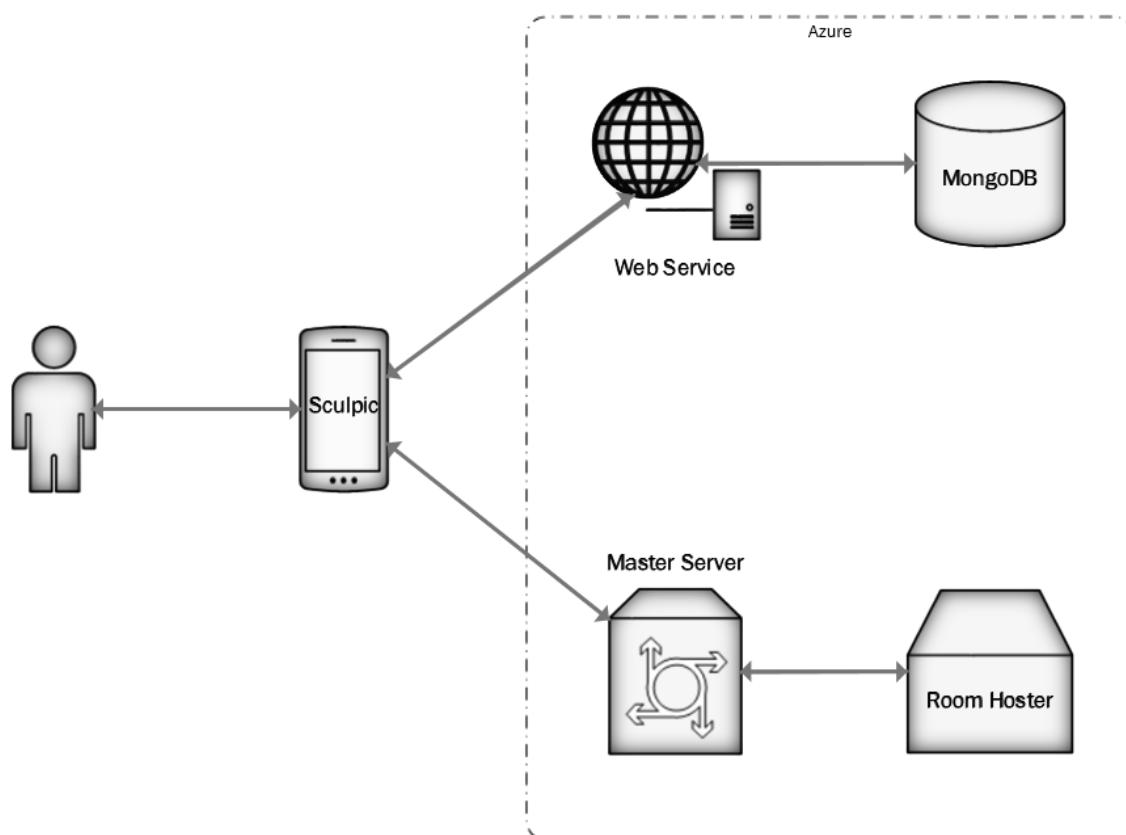


## Rozdział 4

# Architektura projektu

### 4.1. Opis komponentów

Rysunek 4.1 przedstawia ogólny zarys architektury projektu. Szczegółowe elementy zostały opisane poniżej.



Rysunek 4.1: Architektura projektu.

#### 4.1.1. Sculpic

Sculpic jest aplikacją mobilną na platformę Android stworzoną z wykorzystaniem Unity. Z punktu widzenia użytkownika jest to najistotniejsza część projektu, gdyż jedynie z nią użytkownik odbywa interakcję. Łączy się ona z Master Serverem w celu założenia pokoju bądź dołączenia do już istniejącego. Dzięki jego udziałowi pobiera wszelkie informacje o stanie gry z aplikacji hostującej pokój.

#### 4.1.2. Azure - maszyna wirtualna

Na platformie Azure istnieje maszyna wirtualna, na której znajdują się wszystkie elementy związane z serwerową stroną gry.

#### Master Server

Wykorzystujemy aplikację Master Server do obsługi:

- utworzenia pokoju,
- dołączenie gracza do pokoju,
- przesyłania informacji pomiędzy graczami, m.in:
  - zmian zachodzących w obiektach obecnych w grze
  - wiadomości chatu
  - punktów
  - aktualnego rysującego
  - zgadywanego hasła

#### Room Hoster

Room Hoster jest aplikacją, która uruchamia się na maszynie wirtualnej, gdy gracz utworzy pokój. Działa ona dopóki gra w owym pokoju nie zostanie zakończona. Przy pomocy Master Servera komunikuje się z graczami podłączonymi do pokoju, który reprezentuje. Bez Room Hostera niemożliwe jest przeprowadzenie rozgrywki, ponieważ aplikacja ta zarządza głównymi zdarzeniami w grze:

1. rozpoczyna grę
2. utrzymuje kolejkę graczy chcących rysować i wyznacza kolejnego rysującego
3. rozsyła punkty
4. kończy grę
5. uaktualnia ranking

### Web Service

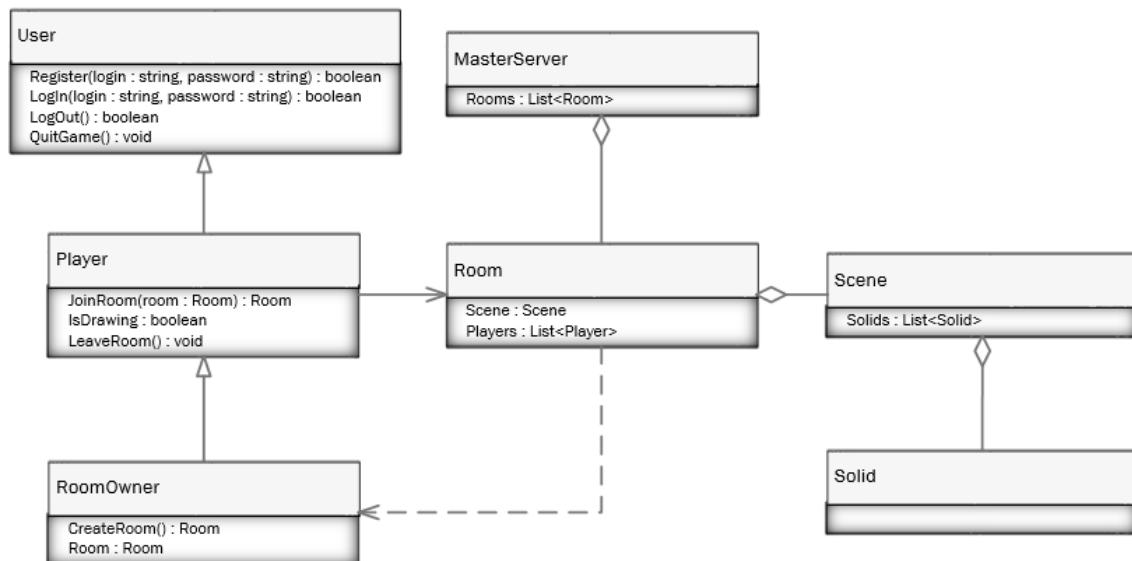
Web Service jest serwisem napisanym w konwencji architektury REST przy wykorzystaniu technologii WCF. Sculpic komunikuje się z nim poprzez proste zapytania. Serwis ten jest odpowiedzialny za komunikację z bazą danych, czyli:

1. przetwarzanie informacji o użytkowniku (tworzenie użytkownika, logowanie, utrzymywanie rankingu)
2. utrzymywanie listy dostępnych haseł
3. losowanie hasła dla rysującego gracza

### MongoDB

MongoDB jest bazą danych, którą wykorzystujemy w naszym projekcie. Przechowuje ona wszystkie informacje o użytkownikach korzystających z aplikacji Sculpic oraz bazę fraz dostępnych w grze. Komunikacja z nią odbywa się przez Web Service.

## 4.2. Dziedzina problemu



Rysunek 4.2: Dziedzina problemu.

Głównymi składnikami dziedziny są:

- użytkownik – osoba korzystająca z aplikacji Sculpic, która może:
  - zarejestrować swoje konto w grze,
  - zalogować się na swoje konto w grze,
  - występować w poniższych rolach:
    - \* gracz – osoba biorąca udział w grze; może:

- założyć pokój (staje się wtedy również Właścicielem Pokoju),
  - dołączyć do istniejącego Pokoju,
  - grać w Pokoju, do którego dołączyła,
  - wyjść z Pokoju,
  - wyjść z gry;
- \* właściciel pokoju – gracz, który założył pokój;
- Master Server – aplikacja udostępniająca m.in. następujące funkcje:
    - zarejestrowanie Pokoju,
    - dołączenie do Pokoju,
    - wyjście z Pokoju;
  - pokój – instancja gry zainicjowana przez Właściciela Pokoju,
  - scena – przestrzeń w ramach Pokoju na której odbywa się gra,
  - bryła – obiekt, który Gracz może umieścić na Scenie podczas gry.

Rysunek 4.2 przedstawia zależności pomiędzy elementami dziedziny.

### 4.3. Struktura bazy danych

W projekcie wykorzystana jest dokumentowa baza danych, która charakteryzuje się brakiem relacji między poszczególnymi kolekcjami. Dlatego też dane nie są łączone poprzez klucze główne ani obce, obiekt zawarty w jednym rekordzie stanowi kompletną całość. Dzięki temu pobieranie danych jest dużo wydajniejsze, gdyż bazy dokumentowe nie wspierają tradycyjnych operacji typu JOIN – musiałoby się to odbywać poprzez ręcznie wyszukiwanie rekordów w różnych kolekcjach.

Baza danych projektu składa się z dwóch kolekcji:

- Users – zawiera dane logowania użytkowników a także informację o ich rankingu,
- Phrases – pełni funkcję spisu haseł losowanych do odgadnięcia w grze, model zawiera również pole wskazujące na ilość wylosowań danej frazy oraz losową liczbę służącą do optymalizacji procesu wyboru hasła.

Do wydajnego działania tych kolekcji wystarczą tylko dwa indeksy na pojedynczych polach (Single Field Index). W kolekcji Users jest to pole UserId, natomiast w kolekcji Phrases – RandomNumber.

Warto zaznaczyć, że baza MongoDB pozwala na umieszczanie w kolekcjach obiektów dowolnych typów. Dbanie o spójność i zgodność obiektów w kolekcji leży w kwestii programisty, w naszym projekcie zapewnione jest to przez system repozytoriów.

## 4.4. Wzorce projektowe

Aby zapewnić czystość i przejrzystość kodu oraz logiczną strukturę aplikacji, zdecydowałyśmy się na wykorzystanie kilku znany i sprawdzonych wzorców projektowych, których zastosowanie znacznie wspomogło proces planowania architektury.

### 4.4.1. Singleton

Wzorzec singleton polega na ograniczeniu ilości istniejących instancji danej klasy do jednej oraz na zapewnieniu dostępu do owego obiektu.

W naszym projekcie wzorzec ten został wykorzystany w klasie Player znajdującej się w grze Sculpic. Z danej instancji gry może korzystać jeden gracz w danym momencie. Jego aktualny stan jest przetrzymywany właśnie w obiekcie klasy Player i musi być dostępny z każdego miejsca w kodzie gry.

### 4.4.2. Repozytorium

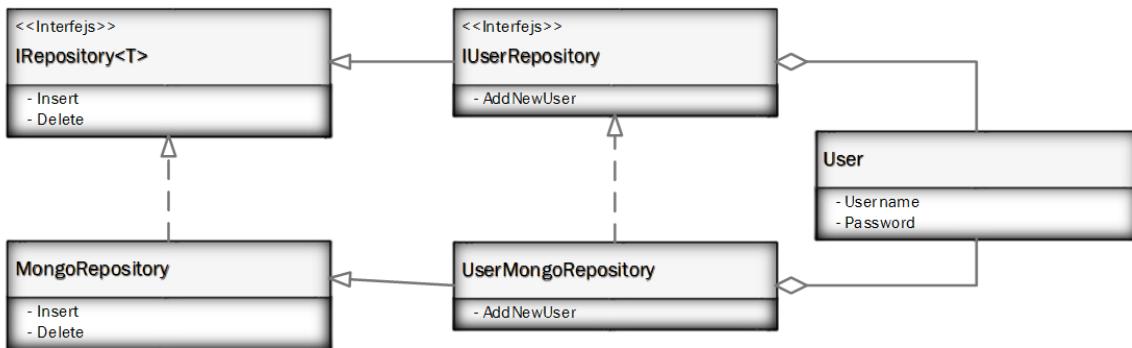
Jest to wzorzec, który ma zastosowanie wszędzie tam, gdzie zachodzi potrzeba odwołania się do danych pobieranych z zewnętrznego źródła takiego jak np. baza danych bądź SharePoint. Polega on na wydzieleniu obsługi operacji związanych z pobieraniem, zapisywaniem i aktualizowaniem każdego modelu do oddzielnej klasy. Repozytorium może zajmować się również mapowaniem modelu bazowego na model biznesowy – jest pośrednikiem między źródłem danych a logiką aplikacji, co ułatwia utrzymanie oraz testowanie funkcjonalności oraz zapewnia przejrzystość architektury. Wzorzec ten ponadto znacznie ogranicza redundancję kodu a także uniezależnia implementację logiki biznesowej aplikacji od użycia konkretnego rodzaju bazy danych.

Najczęstszym zarzutem stawianym temu wzorcowi projektowemu, a także powodem, dla którego często jest on nazywany "antywzorcem" programowania jest twierdzenie, iż repozytorium łamie zasadę jednej odpowiedzialności klasy, jednak dobrze zaprojektowane repozytorium będzie zawierać jedynie operacje bazodanowe, niezależne od logiki aplikacji, co wydaje się być spójną odpowiedzialnością.

W naszym projekcie implementacja tego wzorca odbywa się za pomocą następujących komponentów:

1. Generyczny interfejs IRepository<T> - określa podstawowe operacje bazodanowe możliwe do wykonania na każdym modelu,
2. Generyczna klasa bazowa MongoRepository<T>, implementująca powyższy interfejs – zawiera implementację operacji interfejsu za pomocą sterownika do obsługi bazy Mongo,
3. Interfejsy dla repozytoriów zawierających dodatkowe metody dla każdego typu modelu, dziedziczące po IRepository<T>,
4. Klasy repozytoriów zawierające implementację dodatkowych metod dla każdego modelu, dziedziczące po klasie MongoRepository oraz implementujące interfejs dla konkretnego modelu

Warto zaznaczyć fakt, iż architektura ta jest bardzo uniwersalna i może służyć do obsługi dowolnej aplikacji korzystającej z bazy danych MongoDB.



Rysunek 4.3: Przykład zastosowania wzorca Repozytorium.

#### 4.4.3. SOA

Architektura zorientowana na usługi (Service Oriented Architecture) składa się z dobrze zdefiniowanych serwisów, z których każdy jest niezależnym systemem, akceptującym zapytania oraz zwracającym odpowiedzi za pomocą łatwo dostępnych i jasno określonych interfejsów. Każdy z serwisów może być zmieniany bądź rozbudowywany niezależnie od reszty komponentów wchodzących w skład systemu, co pomaga w łatwym dopasowywaniu się do zmieniających się wymagań biznesowych.

W odróżnieniu od tradycyjnych sposobów tworzenia architektury, które zakładają ścisłe związki oraz współpracę elementów, SOA składa się z luźno powiązanych usług, które wspólnie pozwalają uzyskać pożądane rezultaty. Dodatkowo, implementacja serwisów jest ukryta z punktu widzenia aplikacji klienckiej, zatem wszelkie jej modyfikacje (takie jak optymalizacja bądź dostosowanie do nowych wymagań) nie będą powodowały konieczności wprowadzania zmian w kodzie klienta tak długo, jak nie zmieni się sposób dostępu do usługi.

Kolejną cechą charakterystyczną dla architektury SOA jest również podejście oparte na otwartych i uniwersalnych standardach. Dla przykładu, w naszej implementacji parametry zapytań oraz odpowiedzi przesyłane są w formacie JSON, który może być łatwo obsłużony w niemal każdym współczesnym języku programowania. Zaletą takiego podejścia jest możliwość wykorzystania tych samych usług z poziomu różnych platform, np. aplikacji webowej, aplikacji mobilnej na system Android bądź skryptu używanego przez Unity3D.

Założenia SOA spełnia w naszym systemie serwer bazodanowy. Dzięki temu jest on na tyle uniwersalny, że z jego pomocą możliwa byłaby obsługa grup użytkowników z wielu różnych gier.

## Rozdział 5

# Implementacja

### 5.1. Projekt w Unity

#### 5.1.1. Graficzny interfejs użytkownika

W momencie rozpoczęcia prac nad aplikacją Unity w wersji 4.6 było jeszcze w fazie testów beta. Zdecydowałyśmy się jednak na jego użycie, gdyż wprowadzony został w nim bardzo istotny element – nowy silnik do tworzenia interfejsu użytkownika. Tworzenie elementów w starym silniku wymagało deklarowania ich bezpośrednio w kodzie, bez możliwości podglądu widoku zanim projekt nie został skompilowany i uruchomiony. Dodatkowo wszelką obsługę zmiany rozdzielczości i skalowania elementów również należało zaimplementować we własnym zakresie. A nie jest to proste, szczególnie w przypadku aplikacji tworzonych na urządzenia mobilne, które posiadają bardzo zróżnicowane rozdzielczości ekranów.

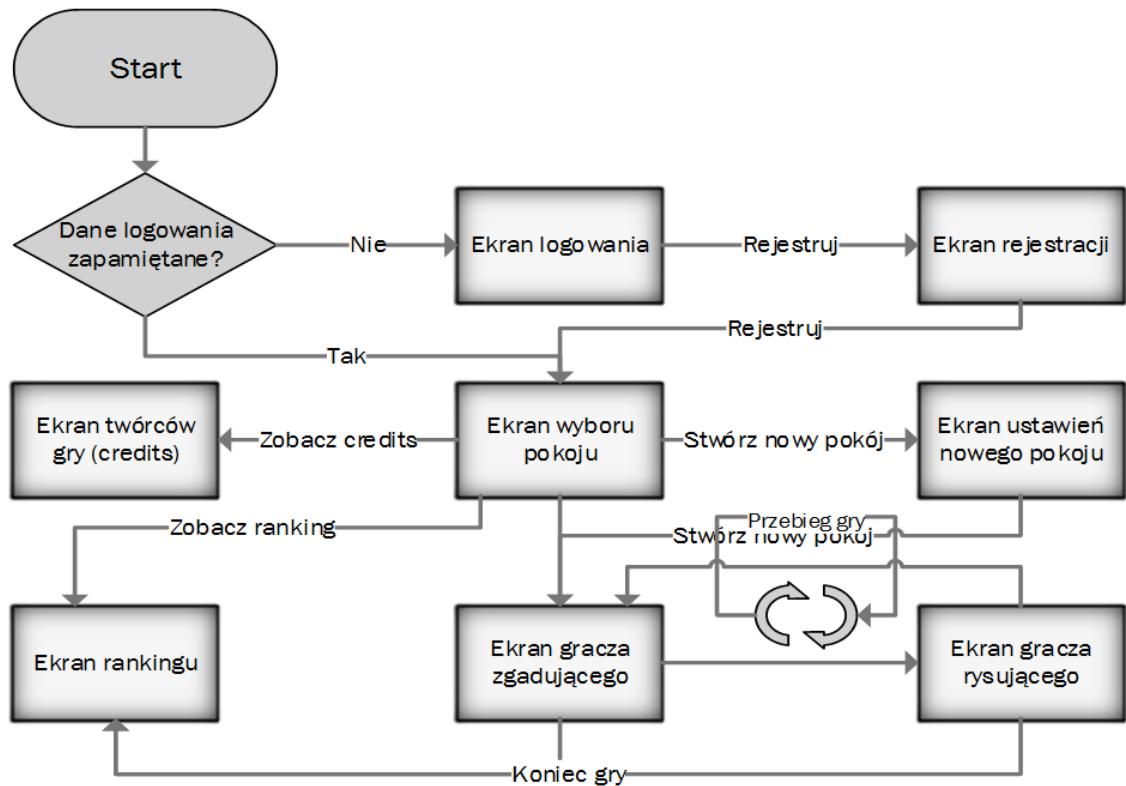
W nowej wersji silnika sposób tworzenia graficznego interfejsu użytkownika przypomina bardziej ten znany z takich technologii jak na przykład WPF (Windows Presentation Foundation) czy WinForms – możliwe jest manualne ustawianie i zmiana rozmiaru elementów. Dodatkowo, każdy element ma przypisane tzw. punkty zakotwiczenia (ang. anchors), które służą do ustalenia wymiarów i pozycji relatywnie do rozmiaru elementu nadzawanego. Znacznie ułatwiło to tworzenie naszej aplikacji. Rysunek 5.1 przedstawia przejścia między ekranami w Sculpicu.

#### 5.1.2. Rozgrywka

Po włączeniu klienta gry i zalogowaniu się, użytkownik może dołączyć do pokoju lub założyć nowy. Gdy użytkownik dołączy do pokoju, do wszystkich graczy znajdujących się w danym pokoju rozsyłana jest informacja o nowej osobie w pokoju. U każdego z nich aktualizowana jest lista aktualnie podłączonych graczy. Aplikacja ładuje wtedy ekran gracza zgadującego.

By rozgrywka się rozpoczęła muszą zostać spełnione 2 warunki:

1. w pokoju muszą się znajdować co najmniej 2 osoby,
2. co najmniej jedna osoba musi się zapisać do kolejki osób rysujących poprzez zaznaczenie opcji *I want to draw*.



Rysunek 5.1: Przejścia między ekranami.

Sposób, w jaki odbywa się rozgrywka jest widoczny na rysunku 5.2. Wyróżniamy podczas niej kilka ważnych elementów opisanych poniżej.

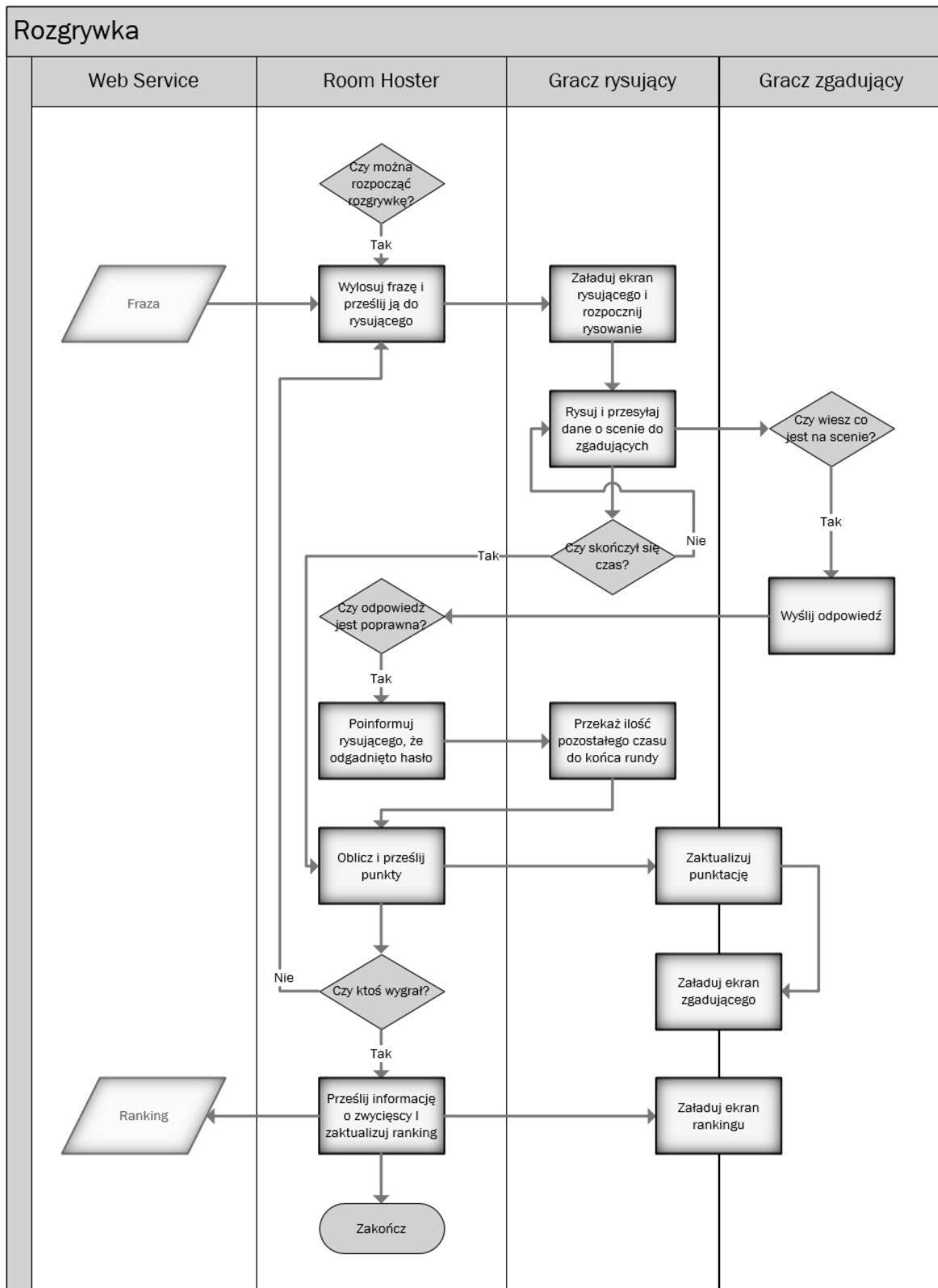
### Obsługa kolejki graczy chcących rysować

Odbiera się to w następujący sposób:

1. Z kolejki graczy chcących rysować zdejmowany jest pierwszy gracz.
2. Wysyłana jest do niego fraza, którą ma rysować.
3. Po zakończeniu rysowania gracz jest ponownie umieszczany w kolejce graczy chcących rysować.

### Hasło do zgadnięcia

Fraza jest losowana, gdy wiadomo już, że gra może się rozpocząć. Odbiera się to poprzez wywołanie zapytania z Web Service'u. Następnie hasło jest wysyłane do gracza, który ma w tej rundzie rysować. Od tego momentu, wszystkie wiadomości w chacie są sprawdzane po stronie Room Hostera pod kątem zgodności z rysowanym hasłem. Gdy wynik jest pozytywny, ogłoszany jest zwycięsca rundy.



Rysunek 5.2: Diagram przedstawiający rozgrywkę.

## Czas

Podczas rundy gracz ma ograniczony czas, w którym może rysować. Gdy ten czas się skończy, gracz otrzymuje ujemne punkty i wybierana jest kolejna osoba do rysowania (procedura jest taka sama jak przy zgadnięciu, z tą różnicą, że nikt nie zdobywa dodatkowych punktów).

## Koniec gry

Podczas całej rozgrywki Room Hoster po zakończeniu każdej rundy rozsyła graczom punkty, o jakie powinni zaktualizować swoją lokalną punktację. Po stronie Room Hostera ta punktacja jest aktualizowana w taki sam sposób. Dzięki temu możliwe jest śledzenie, czy któryś z graczy nie osiągnął odpowiedniej ilości punktów, by zakończyć grę. Tym samym zapewniamy, że niemożliwe jest oszukanie gry w kwestii punktów, ponieważ są one niezależnie liczone przez aplikację znajdującą się na serwerze. Gdy pułap zostanie osiągnięty, Room Hoster informuje o końcu gry wszystkich graczy, a sam wysyła do Web Service'u informację o aktualizacji rankingów graczy, którzy brali udział w danej rozgrywce.

### 5.1.3. Dotyk

W grze zaimplementowana została obsługa dotyku na urządzeniach mobilnych. Jest ona wykorzystywana do:

1. obracania kamery wokół środka sceny (dostępne dla rysującego i zgadującego),
2. przesuwania obiektów po scenie (dostępne dla rysującego).

### 5.1.4. Punktacja

Po wygraniu każdej rundy graczom przydzielane są punkty w zależności od ilości czasu, który upłynął od rozpoczęcia rysowania, odpowiednio:

- do 1 minuty – maksymalna ilość punktów,
- od 1 minut do 2 minut –  $4/5$  maksymalnej ilości punktów,
- od 2 minut do 3 minut –  $3/5$  maksymalnej ilości punktów,
- od 3 minut do 4 minut –  $2/5$  maksymalnej ilości punktów,
- od 4 minut do 5 minut –  $1/5$  maksymalnej ilości punktów,

Maksymalna ilość punktów wynosi:

- 20 dla gracza zgadującego,
- 30 dla gracza rysującego.

### 5.1.5. Komunikacja

Najpowszechniejszymi sposobami komunikacji w naszym projekcie są zapytania REST oraz zdalne wywołania procedur (RPC - Remote Procedure Call) zapewnione przez Unity i Master

Server. Komunikacja z serwerem bazodanowym zostanie opisana w kolejnym podrozdziale, teraz opiszemy komunikację przy pomocy RPC.

### Remote Procedure Calls

Unity daje możliwość oznaczenia wybranych metod atrybutem [RPC]. Dzięki temu możemy wywoływać taką metodę na odpowiadającym obiekcie w innej aplikacji klienckiej. Istnieje kilka warunków, które należy spełnić, by uzyskać oczekiwany efekt.

1. Obiekt deklarujący metody RPC musi mieć podłączony do siebie komponent NetworkView (z przestrzeni nazw UnityEngine).
2. Metoda może mieć argumenty jedynie określonych typów (szczegóły znajdują się w dokumentacji Unity).
3. Wszystkie obiekty (ten, który wywołuje oraz te, na których metoda jest wywoływana) muszą mieć wywoływaną metodę zadeklarowaną. Nawet jeżeli miałaby ona nie mieć wnętrza.

### Master Server

Działanie RPC zapewnione jest za pośrednictwem Master Servera. Unity Technologies udostępniają instancję tej aplikacji użytku programistów Unity. Jednak o wiele wygodniej, co też czynimy w naszym projekcie, jest wykorzystać dostępny kod źródłowy MasterServera dostarczony przez Unity Technologies, by zbudować i włączyć własną instancję owej aplikacji. Ma to wiele zalet, co najważniejsze – posiadamy władzę nad jej czasem działania i nie jesteśmy zależni od dostępności bardzo istotnego w naszym projekcie komponentu.

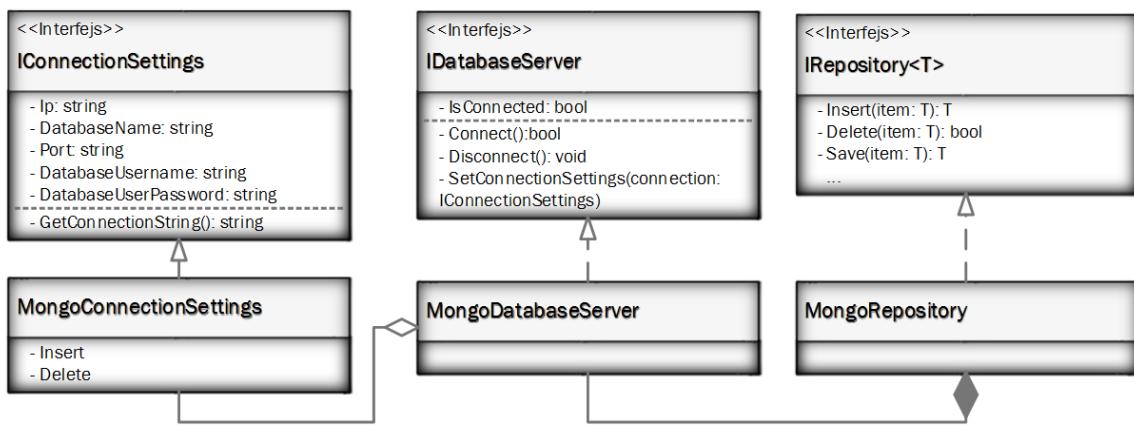
#### 5.1.6. Pokój

Pokój jest reprezentowany przez aplikację Room Hoster. Zawiera ona całą, wcześniej opisaną funkcjonalność. Najważniejszym skryptem w niej umieszczonym jest Room. Znajdujące się tam metody mają za zadanie czuwać nad prawidłowym działaniem komponentów pokoju – kolejką graczy chcących rysować, początkiem i końcem rundy oraz gry, przydzielaniem punktów po każdej rundzie i aktualizacją rankingu.

## 5.2. Serwer bazodanowy

Na serwer komunikujący się z bazą danych składają się trzy projekty:

1. Database – biblioteka do łączenia się z bazą danych, wraz z implementacją dla bazy MongoDB,
2. WcfServer – zawiera implementację serwisów oraz repozytoriów,
3. ServerHost – aplikacja konsolowa hostująca usługi WCF.



Rysunek 5.3: Schemat biblioteki do dostępu do bazy danych.

### 5.2.1. Dostęp do bazy danych

Biblioteka przedstawiona na rysunku 5.3 zawiera podstawowe interfejsy niezbędne do komunikacji z bazą danych, takie jak:

- **IConnectionSettings** – model ustawień bazy danych, zawiera informacje dotyczące serwera: IP, port, nazwa bazy danych, użytkownik i hasło; definiuje również metodę `GetConnectionString`;
- **IDatabaseServer** – zawiera podstawowe metody niezbędne przy współpracy z bazą danych umożliwiające połączenie, rozłączenie, zmianę ustawień oraz sprawdzenie, czy połączenie jest aktywne;
- **IRepository<T>** - definiuje operacje możliwe do przeprowadzenia na danych, takie jak wstawienie rekordu bądź wywołanie zapytania.

Dzięki użyciu w aplikacji interfejsów zamiast konkretnej implementacji możliwe jest łatwe dodawanie obsługi nowego rodzaju bazy danych. Ich użycie jest również szczególnie przydatne w przypadku, gdy aplikacja korzysta z kontenera IoC. W bibliotece znajduje się również implementacja wyżej wymienionych interfejsów przy użyciu sterownika Mongo C# Driver.

### 5.2.2. Serwer WCF

W tym projekcie znajduje się implementacja serwisów wykorzystywanych przez klienta gry do zarządzania użytkownikami, losowania haseł oraz tworzenia pokojów. Zawiera on też dodatkowe metody potrzebne do komunikacji z bazą danych w postaci repozytoriów. Usługi stworzone są w architekturze REST, zapytania wywoływanie są metodą GET, formatem odpowiedzi jest JSON.

#### Zarządzanie użytkownikami

Usługi umożliwiające zarządzanie użytkownikami znajdują się w klasie `UserService`, z którą powiązany jest model `User` oraz repozytorium `UserRepository`. Do najważniejszych jej metod należą:

- LoginUser – umożliwia zalogowanie użytkownika, na podstawie podanego loginu i hasła zwarcane są dane użytkownika; ponadto w bazie zapisywana jest data ostatniego logowania;
- AddNewUser – służy do rejestracji nowych użytkowników;
- PingService – metoda nie przyjmuje żadnych argumentów i zwraca jedynie informację o powodzeniu nawiązania połączenia z serwisem; wywoływana jest przy starcie aplikacji klienckiej w celu rozwiązania adresu IP domeny, dzięki czemu kolejne zapytania wykonywane są dużo szybciej.

## Ranking

Pozycja graczy w rankingu Sculpica jest określana podobnie jak w rankingu szachistów – wykorzystałyśmy metodę obliczania relatywnej siły graczy w punktacji elo. Jednak w przypadku gry wieloosobowej potrzebna była modyfikacja tego systemu, gdyż elo uwzględnia jedynie pojedynki dwóch graczy. Dlatego też każda grupowa gra w Sculpica jest uznawana jako wiele gier jeden na jeden (jakby każdy gracz odbył grę z każdym). W ten sposób do rankingu każdego gracza uczestniczącego w grze dodawana jest suma różnic w rankingu uzyskanych przy obliczeniach dla poszczególnych par.

Aktualizacja rankingu odbywa się poprzez wywołanie metody UpdateRanking w UserService. Jako argumenty podaje się dwa ciągi znaków:

1. nazwy użytkowników oddzielone średnikami,
2. odpowiadające użytkownikom punkty zdobyte podczas rozgrywki oddzielone średnikami.

Metoda ta wykorzystuje klasę EloRanking, w której zawarłyśmy logikę obliczania rankingu.

**Obliczanie rankingu** Dla gracza  $X$  o rankingu  $R_X$ , który grał z graczem  $Y$  o rankingu  $R_Y$  najpierw obliczana jest różnica rankingów obu graczy:  $dR = R_Y - R_X$ .

Następnie oczekiwany wynik rozgrywki, który zawiera się w zakresie  $<0,1>$  (gdzie 0 oznacza przegrana gracza  $X$ , 0,5 – remis, a 1 – przegrana gracza  $X$ ):

$$\mu S = \frac{1}{(1 + 10^{(dR/400)})}.$$

Przez  $S$  oznaczamy rzeczywisty wynik rozgrywki (przyjmuje on wartość według przedstawionych powyżej zasad). Wykorzystamy go do obliczenia różnicy między wynikiem oczekiwany a rzeczywistym:  $dS = S - \mu S$ .

Ostatecznie zmiana w rankingu gracza  $X$   $dR_X$  wynosi:  $dR_X = R_Y + (K * dS)$

Za stałą  $K$  przyjęłyśmy liczbę 32. Niezależnie od początkowego rankingu graczy, w Sculpicu stała ta jest taka sama. W ogólnym przypadku można wykorzystać tę stałą do kontrolowania tempa zmian rankingu w zależności od jego wartości.

## Losowanie frazy do odgadnięcia

Serwisem służącym do pobrania hasła z bazy jest PhraseService z metodą DrawPhrase. Hasła w bazie przechowywane są w postaci modelu Phrase, w którym oprócz samego hasła znajdują

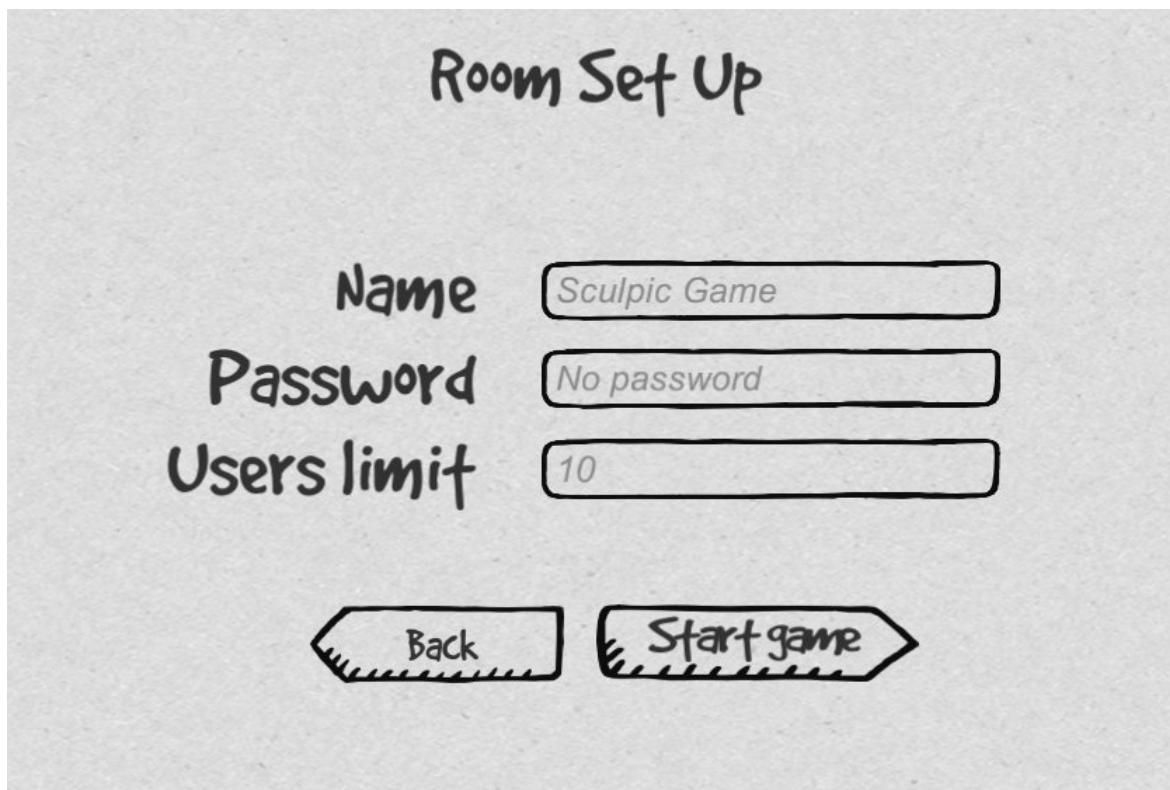
się też takie informacje jak ilość wylosowań oraz losowa liczba, służąca do optymalizacji zapytania losującego hasło z bazy.

Z tym serwisem związane jest repozytorium PhraseService, które zawiera tylko jedną metodę – GetRandomPhrase. Najbardziej oczywistym podejściem do losowania hasła byłoby wylosowanie liczby, a następnie pobranie rekordu znajdującego się na tej pozycji w bazie, jednak wiążałoby się to z koniecznością pobrania dużej ilości danych, co byłoby nieoptymalne. Dlatego też każde hasło ma przypisaną losową liczbę – przy losowaniu hasła losowana jest liczba, następnie rekordy sortowane są rosnąco i wybierane jest pierwsze hasło z wartością pola RandomNumber większym od wylosowanej liczby. Jeśli takie nie istnieje, brane jest hasło z mniejszą wartością.

Zapytanie to wywoływanie jest bezpośrednio na bazie za pomocą zapytania sterownika Mongo i wiąże się z pobraniem tylko jednego rekordu. Konieczność sortowania danych nie jest dużym obciążeniem, gdyż na polu RandomNumber założony jest indeks, dodatkowo zakładana jest rzadka częstotliwość zmian tych danych w bazie.

### Zakładanie nowego pokoju

Do założenia nowego pokoju niezbędne jest wywołanie usługi SetUpNewRoom znajdującej się w klasie RoomService. Wymaga ona podania bazodanowego ID użytkownika zakładającego pokój, nazwy gry oraz limitu osób mogących dołączyć do pokoju. Opcjonalnie można również podać hasło dostępu do pokoju – w przypadku, gdy nie jest ono zdefiniowane, usługa otrzymuje zamiast niego komunikat „nopassword”.



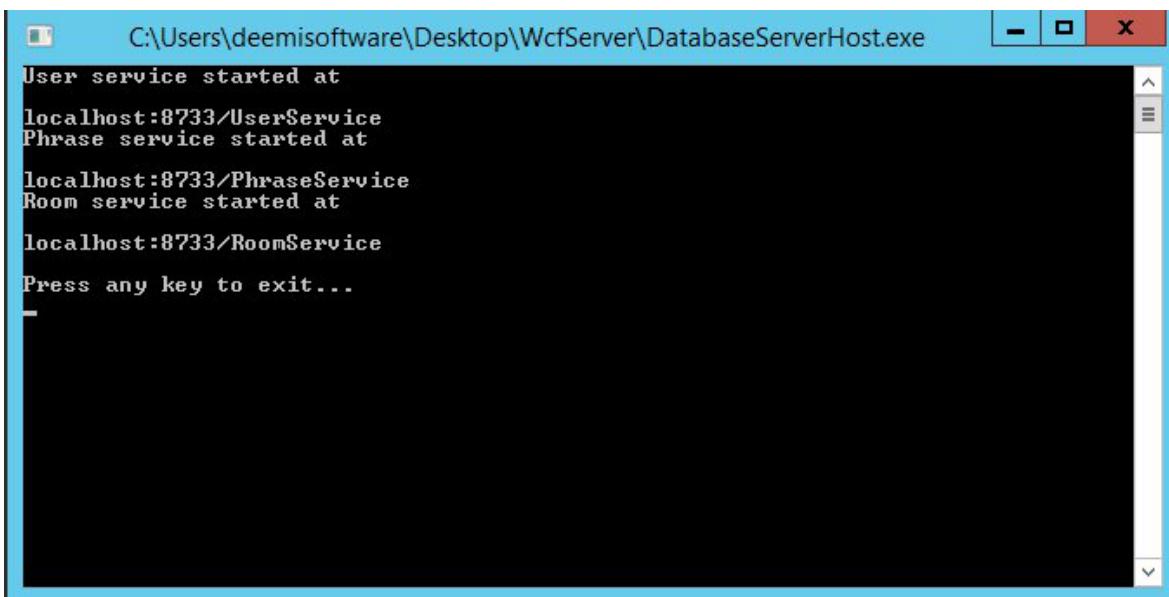
Rysunek 5.4: Ekran ustawień nowego pokoju.

Logika zarządzania pokojami zawarta jest w klasie RoomManager. Zawiera ona listę aktualnie uruchomionych pokojów a jej główną metodą jest metoda CreateNewRoom. Jako iż nie jest możliwe dodanie własnych argumentów uruchomienia programu stworzonego w Unity, przekazanie parametrów pokoju musiało zostać obsłużone w inny sposób:

1. Tworzony jest nowy katalog, który będzie katalogiem roboczym nowego procesu Unity.
2. W katalogu do pliku XML zapisywane są ustawienia nowego pokoju.
3. Uruchamiany jest proces RoomHoster, który jest wersją aplikacji klienckiej zbudowanej ze sceną RoomHoster. Program ten na starcie odczytuje konfigurację, ustala pierwszy wolny port i automatycznie rozpoczyna hostowanie pokoju.

### 5.2.3. Server Host

Aplikacja konsolowa służąca do hostowania serwisów jest bardzo prosta – przy starcie uruchamiane są hosty poszczególnych serwisów, naciśnięcie dowolnego klawisza kończy działanie programu. Najważniejszym elementem aplikacji jest plik konfiguracyjny App.config, który definiuje punkty dostępne dla serwisów. Typem wiązania używanym przez usługi jest „WebHttpBinding” z racji tego, że są one konsumowane za pomocą zapytań HTTP (architektura REST), a nie przy użyciu wiadomości SOAP.



```
C:\Users\deemisoftware\Desktop\WcfServer\DatabaseServerHost.exe
User service started at
localhost:8733/UserService
Phrase service started at
localhost:8733/PhraseService
Room service started at
localhost:8733/RoomService
Press any key to exit...
```

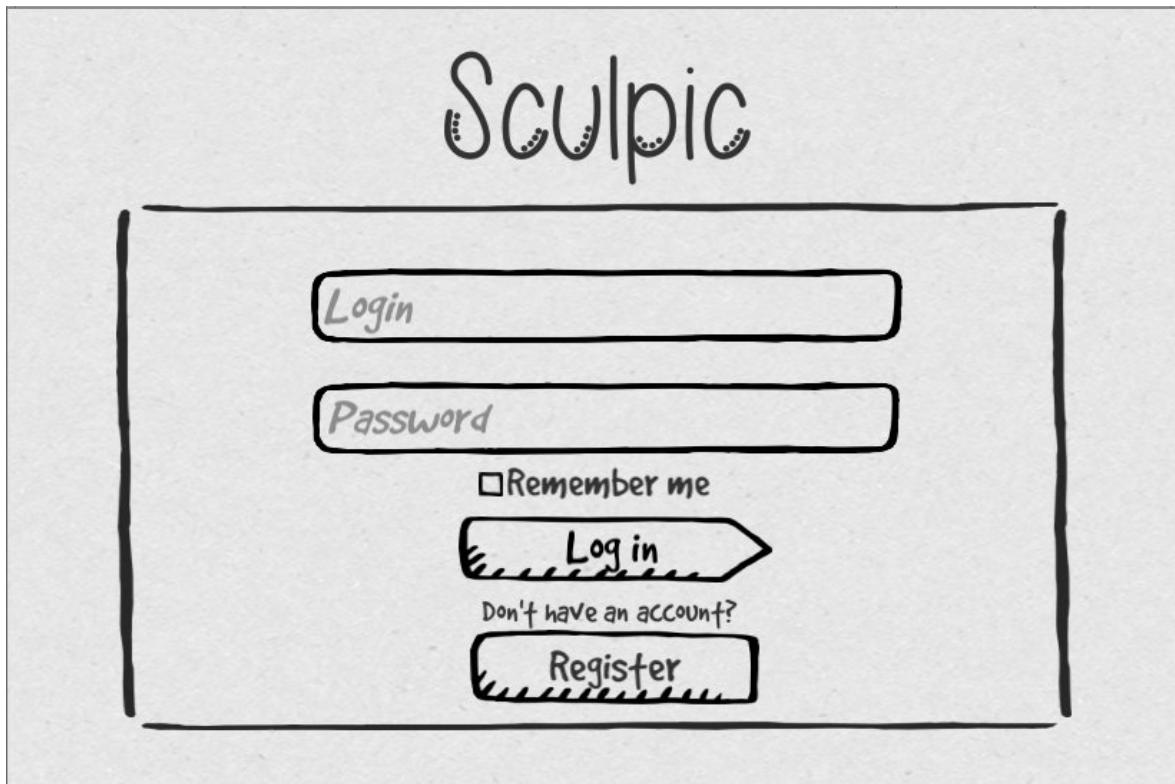
Rysunek 5.5: Wygląd uruchomionej aplikacji hostującej serwer bazodanowy.



## Rozdział 6

### Opis aplikacji

#### 6.1. Ekran logowania



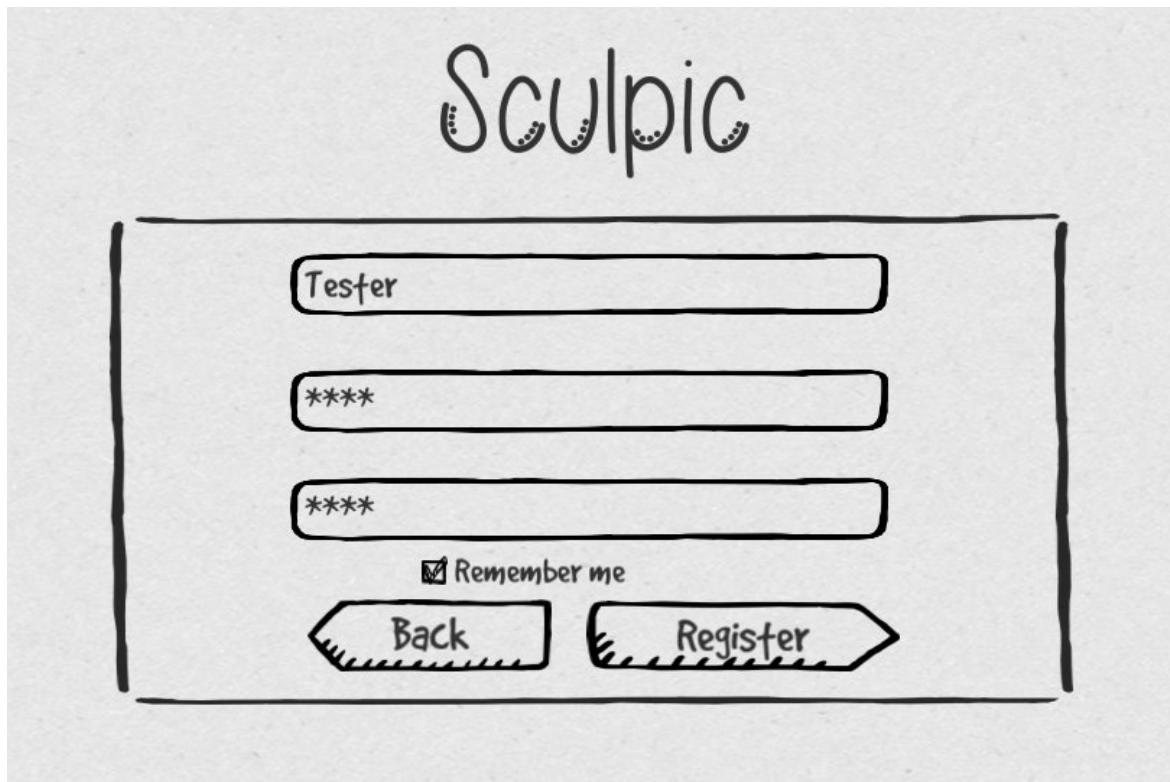
Rysunek 6.1: Ekran logowania.

Ekran ten umożliwia zalogowanie się za pomocą danych do konta stworzonego wcześniej, bądź przejście do ekranu rejestracji. Pola tekstowe akceptują alfanumeryczne dane wejściowe, ponadto przy kliknięciu przycisku logowania następuje walidacja pól oraz wyświetlenie odpowiednich komunikatów w przypadku jej niepowodzenia. Możliwe jest także odznamowanie pola „Zapamiętaj mnie”, co sprawi, że przy ponownym uruchomieniu aplikacji nastąpi próba automatycznego logowania zapisanymi danymi.



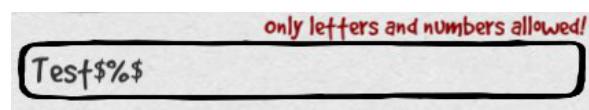
Rysunek 6.2: Walidacja pustego pola.

## 6.2. Ekran rejestracji

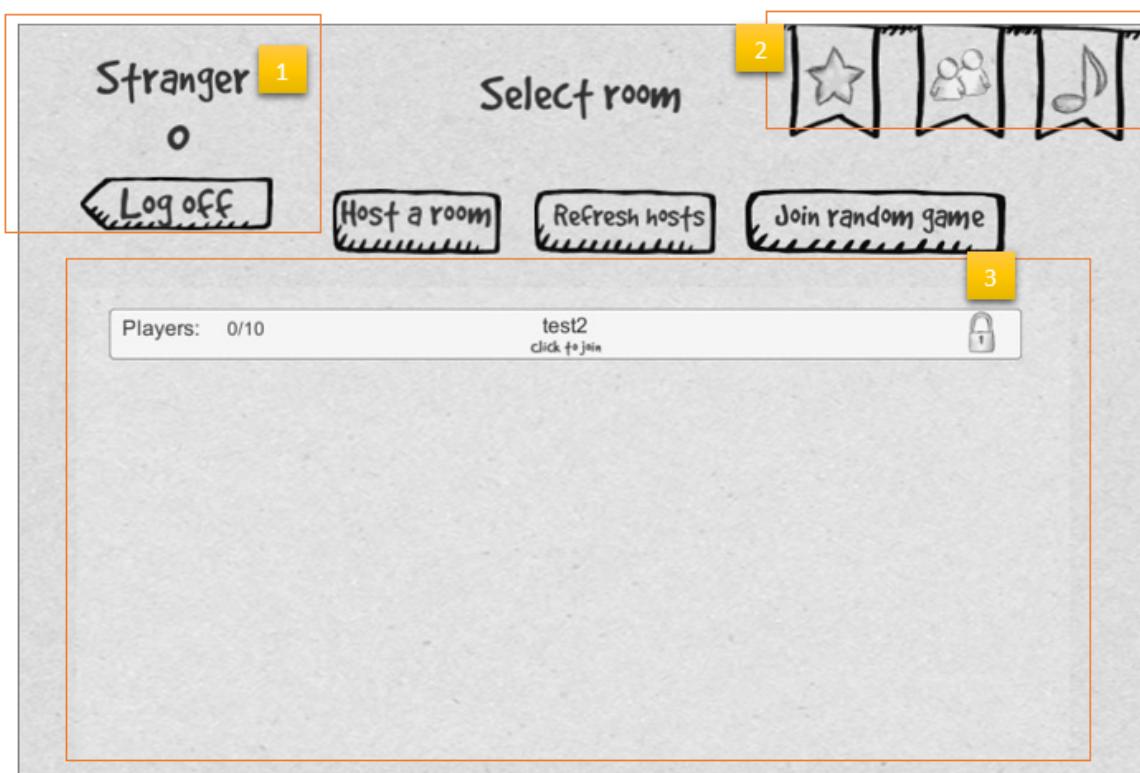


Rysunek 6.3: Ekran rejestracji.

Ekran ten umożliwia założenie nowego konta poprzez podanie loginu oraz hasła. Kliknięcie przycisku „Zarejestruj” powoduje rejestrację, logowanie oraz przejście do ekranu wyboru pokoju. Reszta elementów zachowuje się tak jak na ekranie logowania, dochodzi jedynie dodatkowa walidacja – login musi składać się od 2 do 15 znaków, może zawierać jedynie litery oraz cyfry, hasła muszą do siebie pasować. W przypadku błędnie wprowadzanych danych wyświetlane są odpowiednie komunikaty.



Rysunek 6.4: Walidacja znaków specjalnych.



Rysunek 6.5: Ekran wyboru pokoju.

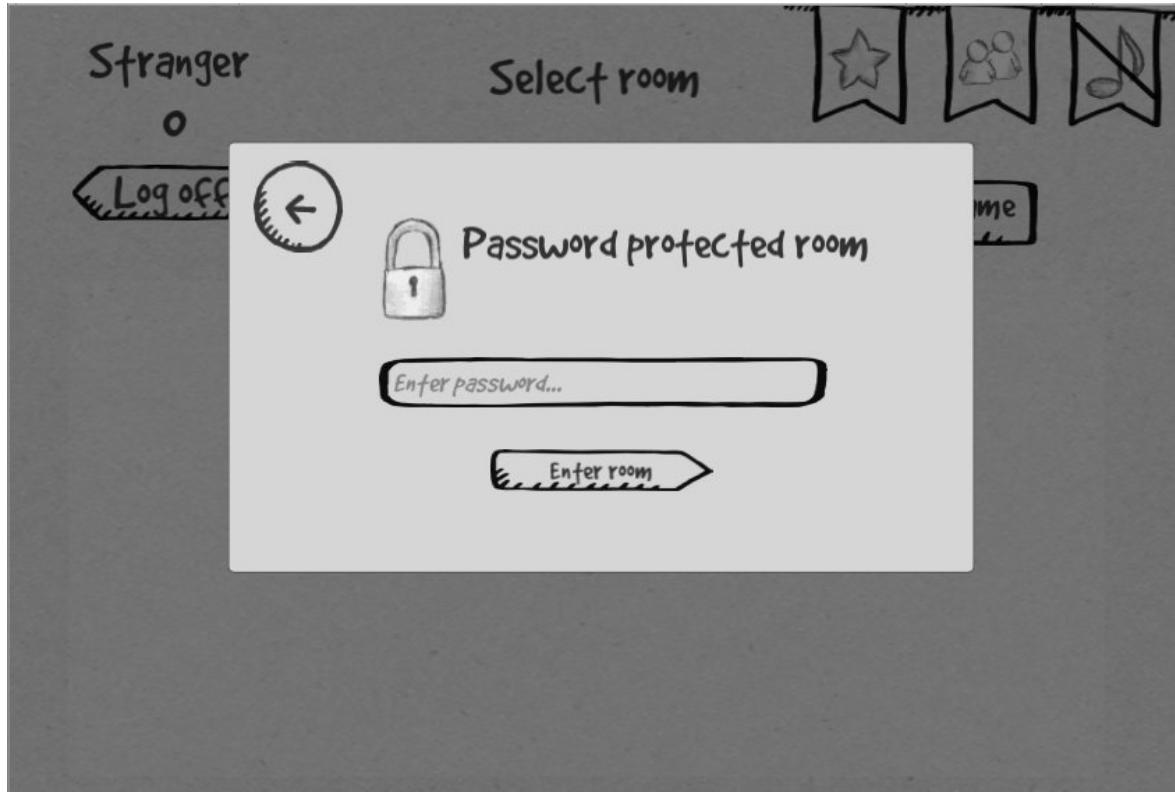
### 6.3. Ekran wyboru pokoju

Jest to główny ekran aplikacji. Składa się on z trzech obszarów:

1. Panel użytkownika – w nim wyświetlane są podstawowe informacje o użytkownikowi, takie jak nazwa oraz ilość punktów rankingowych, zawiera on też przycisk umożliwiający wylogowanie (wyczyszczenie zapisanych danych dostępowych do konta oraz przejście do ekranu logowania).
2. Mini menu – zawiera przycisk przejścia do ekranu rankingu, przycisk prowadzący do ekranu zawierającego informacje o twórcach aplikacji oraz przycisk pozwalający włączyć lub wyłączyć dźwięki gry.
3. Lista dostępnych pokojów – przyjmuje postać przewijanej listy złożonej z przycisków odpowiadającym pokojom, do których użytkownik może dołączyć. Kafelek pokoju zawiera podstawowe informacje takie jak liczba znajdujących się w nim graczy, limit graczy oraz nazwa. Ikonka kłódki znajdująca się po prawej stronie elementu informuje o tym, że dostęp do pokoju chroniony jest hasłem, w przeciwnym przypadku nie jest wyświetlana. Kliknięcie w przycisk powoduje połączenie się z pokojem oraz wejście do gry.

Dodatkowo na ekranie znajduje się jeszcze przycisk pozwalający odświeżyć listę, przycisk umożliwiający przejście do ekranu tworzenia nowego pokoju oraz przycisk dołączenia do losowego pokoju niezabezpieczonego hasłem.

W przypadku próby wejścia do pokoju z ustawionym hasłem wyświetlone jest okno przedstawione na rysunku 6.6.

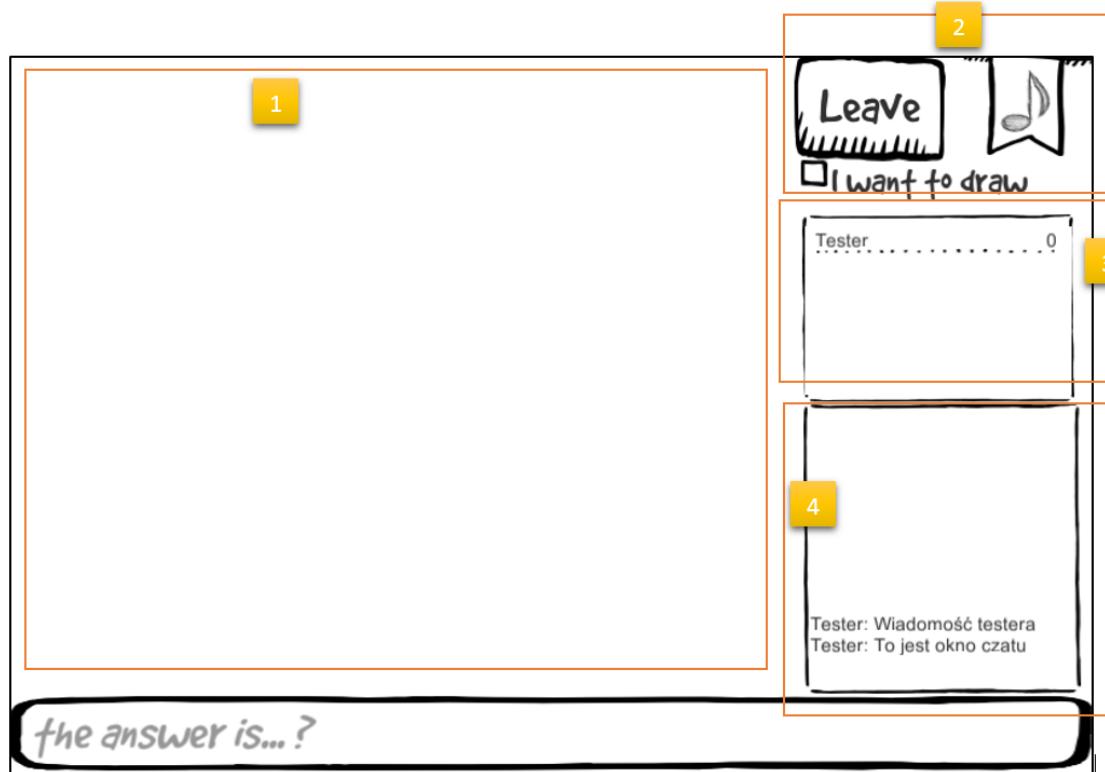


Rysunek 6.6: Okno umożliwiające wpisanie hasła.



Rysunek 6.7: Komunikat informujący o niepoprawnym haśle.

## 6.4. Ekran gracza zgadującego

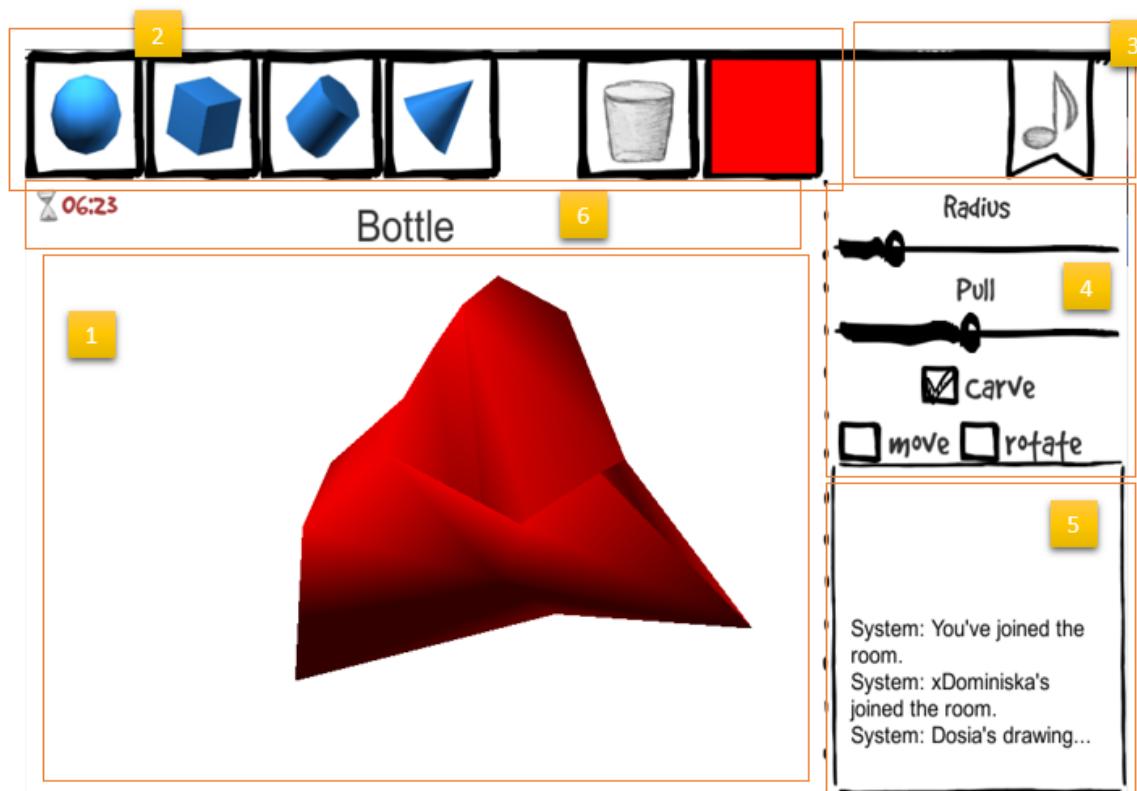


Rysunek 6.8: Ekran gracza zgadującego.

Ten ekran widzą gracze, który aktualnie nie rysują. Złożony jest z następujących komponentów:

1. Obszar gry – w tym miejscu jest wyświetlany obraz aktualnie stworzony przez rysującego. Każdy gracz może niezależnie obracać kamerą na swoim ekranie tak, by dało się przyjrzeć figurom z każdej strony.
2. Mini menu – zawiera przycisk umożliwiający opuszczenie gry, wyłączenie/wyłączenie dźwięku a także pole wyboru pozwalające na dołączenie bądź opuszczenie kolejki rysujących.
3. Lista graczy – przewijana lista wyświetlająca nazwę oraz aktualną ilość punktów w grze każdego gracza w pokoju, posortowana malejąco według punktacji.
4. Okno czatu – przewijana lista wiadomości, które użytkownicy wymieniają między sobą. Wyświetlane są tam też komunikaty systemu takie jak informacje o dołączających do pokoju graczech czy odgadniętym haśle.

Dodatkowo ekran zawiera pole umożliwiające pisanie wiadomości czatu, a tym samym odgadnięcie hasła.



Rysunek 6.9: Ekran gracza rysującego.

## 6.5. Ekran gracza rysującego

Ekran ten w danym momencie widzi tylko jedna osoba w pokoju – gracz aktualnie rysujący. Jest to najbardziej złożona scena aplikacji. Jej elementy są następujące:

1. Pole modelowania – w tym obszarze pojawiają się nowo dodane figury, które można następnie modelować. Możliwe jest obracanie figur bądź poruszanie kamerą, umożliwiają to pola wyboru z obszaru numer 4.
2. Przybornik – grupa przycisków po lewej stronie tego obszaru służy do wstawiania na scenę nowych figur. Dostępne są modele takie jak kula, sześcian, walec oraz stożek. Przyciski po prawej stronie umożliwiają wyczyszczenie sceny oraz zmianę koloru dodawanej figury.
3. Przycisk włączenia/wyłączenia dźwięku.
4. Ustawienia rysowania – suwaki umożliwiają zmianę parametrów skryptu sculptora – skryptu odpowiadającego za zmianę kształtu bryły. Możliwa jest zmiana zasięgu obszaru, na który wpływa sculptor oraz siły wybrzuszania. Pola wyboru pozwalają na tworzenie wgłębień a także zmianę pozycji oraz obracanie bryły.
5. Okno czatu – statyczne okno czatu. Gracz rysujący nie ma możliwości pisania wiadomości, jednak może podglądać, czy gracze zgadujący są naprowadzani na poprawny trop odgadnięcia hasła.
6. Panel informacyjny – w nim wyświetlany jest stoper odliczający czas pozostały na odgadnięcie hasła oraz samo hasło.

## 6.6. Ekran rankingu



Rysunek 6.10: Ekran rankingu.

Ekran rankingu składa się z przesuwanej listy nazw graczy oraz wartości ich rankingów. Wyświetlane jest 20 najlepszych wyników w kolejności malejącej. Rekordy pobierane są z bazy za pomocą odpowiedniego serwisu.



# Rozdział 7

## Testy

### 7.1. Testy jednostkowe

Projekt serwera testowany jest za pomocą testów jednostkowych z użyciem biblioteki MSTest. Aby testy mogły zostać uruchomione, niezbędne jest uruchomienie bazy danych MongoDB pod lokalnym adresem oraz domyślnym portem (27017). Testowane jest podstawowe połączenie z bazą danych oraz operacje na niej, a także serwisy: logowania, rejestracji, losowania hasła, tworzenia pokoju oraz wyliczania rankingu.

W sumie napisane zostały 44 testy jednostkowe, sprawdzające poprawność działania usług pod kątem nie tylko standardowego wywołania, ale też różnych sytuacji brzegowych, takich jak na przykład niepoprawne argumenty.

Konwencja nazewnictwa testów jest następująca: nazwaMetody\_warunki\_oczekiwanyRezultat.

W sprawnym pisaniu warunków asercji pomogła nam biblioteka FluentAssertions, dostępna jako paczka NuGet (<http://www.fluentassertions.com/>). Przykład jej użycia widać na rysunku 7.2.

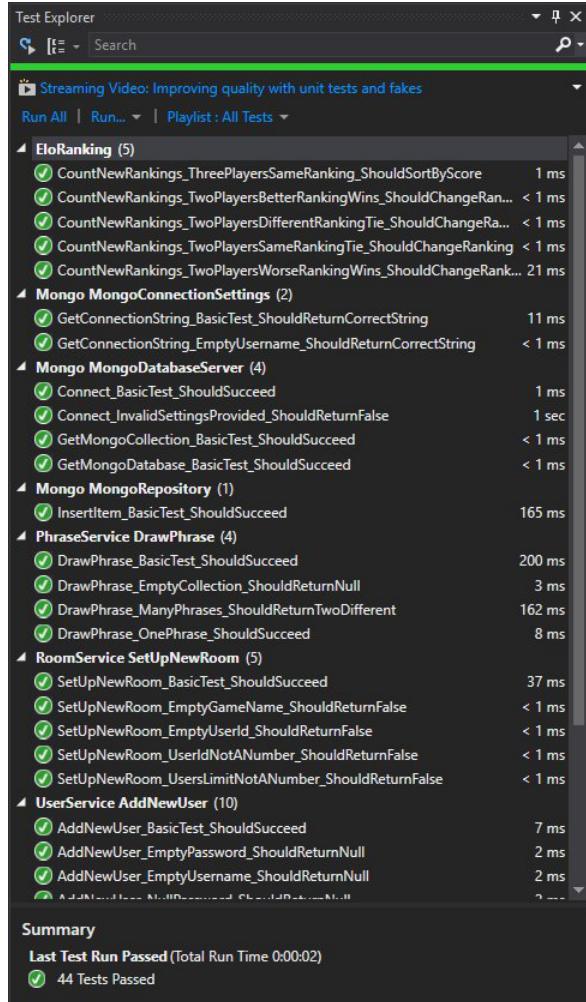
### 7.2. Testy integracyjne

Pełne środowisko testowe składało się z następujących elementów:

- Maszyna wirtualna stworzona na platformie Azure:
  - System operacyjny: Windows Server 2012, x64
  - Ilość rdzeni: 1 x 2,20 GHz
  - Ilosć pamięci RAM: 1,75 GB
  - Adres DNS: sculpicserver.cloudapp.net

z uruchomionymi instancjami MasterServera, serwera WCF oraz uruchamianymi w miarę potrzeb instancjami pokojów (Sculpic Hoster). Uruchomiona jest też baza danych MongoDB dostępna lokalnie pod domyślnym portem.

- Urządzenia mobilne:



Rysunek 7.1: Wynik uruchomienia wszystkich testów jednostkowych.

```
0 | 0 references | Emilia Szymańska, 61 days ago | 1 change
public void LoginUser_BasicTest_ShouldSucceed()
{
    var result = userService.LoginUser(defaultTestUser.Username, defaultTestUser.Password);
    result.Should().NotBeNull();
    result.Username.Should().Be(defaultTestUser.Username);
}
```

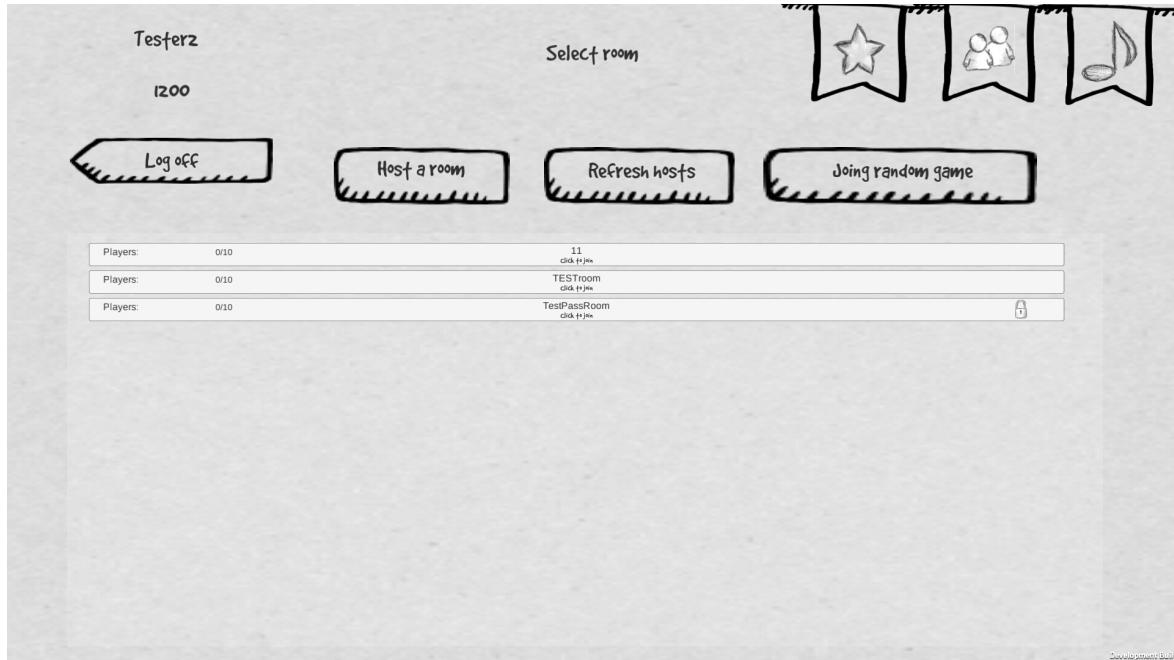
Rysunek 7.2: Przykład asercji z użyciem biblioteki FluentAssertions.

Rodzaj urządzenia	Model	Wielkość ekranu	Rozdzielcość ekranu
Smartfon	Sony Xperia P	4"	540 x 960 px
Smartfon	Samsung Galaxy Note III	5,7"	1080 x 1920 px
Smartfon	Samsung Galaxy S5	5,1"	1080 x 1920 px
Tablet	Samsung Galaxy Tab 2	10,1"	1280 x 800 px

Na wszystkich wyżej wymienionych urządzeniach aplikacja działała poprawnie. Na każdym z nich została dokładnie sprawdzona cała funkcjonalność wraz z jednoczesnym prowadzeniem rozgrywki (w jednym pokoju). Poniżej zaprezentowane są zrzuty ekranu z urządzeń, na których można zaobserwować skalowalność interfejsu użytkownika:



Rysunek 7.3: Zrzut ekranu z telefonu Sony Xperia P.



Rysunek 7.4: Zrzut ekranu z telefonu Galaxy Note III.



Rysunek 7.5: Zrzut ekranu z tabletu Galaxy Tab 2.

## Rozdział 8

# Podsumowanie i wnioski

## 8.1. Napotkane problemy

### 8.1.1. Przesyłanie brył

Funkcjonalność, na której opiera się cała gra była najtrudniejszą do wykonania. Należało przesyłać aktualny stan brył ze sceny gracza rysującego na sceny graczy oglądających.

Pierwszym pomysłem było przesyłanie wszystkich informacji o każdej bryle. W każdej klatce przesyłałyśmy rozmieszczenie wierzchołków, informacje o trójkątach oraz wektorach normalnych. Przy używanych przez nas siatkach brył, które miały ponad 500 wierzchołków oraz prawie 800 trójkątów oznaczało to przesyłanie ogromnych porcji danych. Przesłanie takiej siatki w sieci lokalnej trwało około 7 sekund. Był to czas niedopuszczalny przy grze, która miała przesyłać obraz w czasie zbliżonym do realnego nie wspominając o zacinającym się interfejsie graficznym.

Najprostszym rozwiązaniem wydawałoby się umieścić przesyłanie w oddzielnym wątku, jednak Unity3D w większości przypadków nie zezwala na użycie wątków. Jednak one nie sprawdziłyby się w naszym przypadku.

W związku z tym, postanowiłyśmy zmniejszyć ilość przesyłanych danych poprzez zmniejszenie ilości wierzchołków w siatkach. Dało to spodziewany efekt – przyspieszyło przesyłanie (do czasu około 1 sekundy na bryłę), jednak interfejs nadal się zacinał i nie dało się w swobodny sposób korzystać w aplikacji.

Postanowiłyśmy więc zmniejszyć częstotliwość przesyłania danych. Udostępniliśmy graczowi rysującemu przycisk aktywujący przesyłanie aktualnej sceny. Znaczaco zniwelowało to problem zacinającego się interfejsu graficznego. Jednak pozostał problem, który towarzyszył nam praktycznie od samego początku – dane czasami się gubiły i niektóre figury nie aktualizowały swoich kształtów. Jako przyczynę podejrzewamy za duży rozmiar pakietów, które były przesyłane podczas synchronizacji scen.

Ostatecznie wdrożyłyśmy pomysł, co do którego miałyśmy bardzo duże wątpliwości od samego początku. Polega on na tym, że gracz rysujący rozsyła graczom oglądającym lokalizację każdego swojego kliknięcia i aktualizacja wyglądu brył następuje po stronie każdego klienta. Istniało prawdopodobieństwo, że po pewnym czasie wyglądy scen gracza rysującego i gracza oglądającego będą od siebie znaczaco odbiegać, jedna nie zaobserwowałaby niczego takiego. To rozwiązanie również w żaden sposób nie spowalnia obsługi interfejsu użytkownika ani nie

przesyła nadmiar danych przez sieć. Jest ono optymalne do naszych potrzeb i sprawdza się bardzo dobrze.

### 8.1.2. Pokoje

Mechanizm sieciowy dostarczany razem z silnikiem Unity3D działa na zasadzie peer to peer, umożliwiając jednoczesne hostowanie pokoju oraz dołączenie do niego. Niestety o ile rozwiązanie to działało w przypadku, gdy wszystkie urządzenia były w zasięgu jednej podsieci, nie sprawdziło się ono w momencie gdy urządzenie hostujące pokój znajdowało się w innej podsieci niż gracz próbujący dołączyć do pokoju. Pokój był widoczny dla użytkownika, gdyż lista hostów znajduje się na MasterServerze, uruchomionym na maszynie z publicznym IP. Gracz nie mógł jednak połączyć się z takim pokojem, gdyż urządzenie hostujące nie posiadało publicznego adresu IP. W takiej sytuacji zwracany był wyjątek „Receiving NAT punchthrough attempt from target [...] failed”.

Pierwszą próbą rozwiązania tego problemu było użycie dodatkowego programu dostarczanego przez Unity3D – tzw. Facilitatora, który w teorii powinien był pomagać w zestawianiu połączeń peer to peer. Niestety, po wielokrotnych próbach i przy różnych konfiguracjach nie udało nam się doprowadzić takiego zestawu do stanu działającego. Dodatkowo w Internecie znalazłyśmy wiele opisów pisanych przez innych programistów, którzy natknęli się na podobny problem. Wskazywały one na to, iż program ten jest zwyczajnie pełen błędów i jego działanie (bądź nie) jest często losowe.

W związku z tym konieczne stało się uruchamianie pokojów na serwerze posiadającym publiczne IP oraz przekierowane porty pokojów. Aby można było dać użytkownikom możliwość stworzenia własnego pokoju, niezbędne stało się udostępnienie odpowiedniego serwisu, który powoduje uruchomienie procesu aplikacji hostującej pokój na maszynie serwera. Przy okazji tworzenia tej usługi natknęłyśmy się na kolejny problem – nie ma możliwości zdefiniowania własnych argumentów, które można podać aplikacji wywołując ją na przykład z wiersza poleceń. W związku z tym przekazanie parametrów pokoju musi odbywać się poprzez stworzenie nowego folderu dla każdego pokoju, zapisanie do niego konfiguracji w postaci pliku XML oraz uruchomienie procesu hostera z ustawionym danym katalogiem jako roboczy. Program na starcie szuka pliku konfiguracji i wczytuje go ( w przypadku braku danego pliku stosowana jest konfiguracja domyślna), po czym uruchamiane jest hostowanie pokoju.

### 8.1.3. Wywoływanie zapytań RESTowych

W początkowej fazie projektu komunikacja z bazą danych została zaimplementowana za pomocą specjalnej biblioteki wspomagającej obsługę zapytań typu REST. Przy komplikacji projektu na system Android okazało się jednak, że do budowania aplikacji korzystających z przestrzeni nazw System.Net potrzebna jest płatna wersja wtyczki do Unity3D – AndroidPro. Jako iż wyżej wspomniała biblioteka oparta jest na socketach platformy .Net, konieczna była zmiana implementacji.

Jednym wyjściem stało się użycie specjalnej klasy Unity3D – WWW. Służy ona głównie do wyświetlania zawartości stron internetowych w kontrolkach, jednak okazało się, że dobrze obsługuje też zapytania typu GET. Jej użycie jest jednak dość toporne, dodatkowo nie ma możliwości jej użycia w wątku innym niż główny wątek aplikacji, co skutkuje krótkochwilowym zawieszeniem interfejsu użytkownika podczas łączenia się z serwerem.

## 8.2. Wnioski

Mimo licznych problemów udało nam się ostatecznie doprowadzić aplikację do stanu, który można uznać za kompletny. Wszystkie wymagania początkowe zostały spełnione, łącznie z działaniem modelowania bryły w czasie (prawie) rzeczywistym. Gra wygląda schludnie, działa na zdecydowanej większości urządzeń z systemem Android, oraz dobrze skaluje się na ekranach o różnych wymiarach. Teoretycznie Unity3D umożliwia skompilowanie projektu na urządzenia mobilne z innymi systemami operacyjnymi, takimi jak iOS czy Windows Phone, jednak niestety nie dysponujemy odpowiednim sprzętem pozwalającym to przetestować.

Trzeba przyznać, iż jest to projekt eksperymentalny – taka forma obsługi modelowania na scenie nie pozwala na swobodne i łatwe stworzenie kształtów przypominających wiele fraz, które można by było odgadywać. Aby było to bardziej przystępne należałoby dodać kilka różnych sposobów „wybrzuszania” modelu – na przykład umożliwić tworzenie ostrych końców.

Dalsze rozwijanie aplikacji mogłoby również polegać na stworzeniu nowych serwisów, przykładowo obsługujących system przyjaciół, co pozwoliłoby na zapraszanie znajomych do wspólnej gry. W przypadku bardzo dużej ilości graczy konieczne byłoby także stworzenie bardziej rozbudowanej i rozproszonej architektury serwerowej.

Praca nad projektem Sculpic była dla nas świetną okazją na pogłębienie wiedzy o tworzeniu gier sieciowych oraz poszerzyła naszą świadomość o szereg problemów, które można napotkać, gdy rozgrywka wymaga ciągłego przesyłania danych.



# Literatura

- [1] Nishith Pathak, *Pro WCF 4: Practical Microsoft SOA Implementation, Second Edition.* 2011.
- [2] Unity Technologies, <http://docs.unity3d.com/Manual/>. dostęp styczeń 2015.
- [3] Microsoft, <http://azure.microsoft.com/en-us/documentation/>. dostęp styczeń 2015.



Dominika Bodzon Nr albumu 245518  
Emilia Szymańska Nr albumu 245558

Warszawa, 2 lutego 2015

## Oświadczenie

Oświadczamy, że pracę inżynierską pod tytułem „Wieloosobowa gra 3D na platformy mobilne w architekturze klient-serwer”, której promotorem jest dr inż. Paweł Kotowski wykonałyśmy samodzielnie, co poświadczamy własnoręcznymi podpisami.

.....  
.....  
Dominika Bodzon  
Emilia Szymańska