



**POLITECHNIKA
GDAŃSKA**

WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI I INFORMATYKI



Imię i nazwisko studenta: Michał Żyłowski
Nr albumu: 137448
Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Specjalność/profil: -

Imię i nazwisko studenta: ANDRZEJ JOSKOWSKI
Nr albumu: 143245
Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Specjalność/profil: -

Imię i nazwisko studenta: PATRYK URBAN
Nr albumu: 143370
Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Specjalność/profil: -

Imię i nazwisko studenta: ADRIAN PIELECH
Nr albumu: 143313
Studia pierwszego stopnia
Forma studiów: stacjonarne
Kierunek studiów: Informatyka
Specjalność/profil: -

PRACA DYPLOMOWA INŻYNIERSKA

Tytuł pracy w języku polskim: Programowalna sieciowa gra RTS

Tytuł pracy w języku angielskim: A programming RTS online game

Potwierdzenie przyjęcia pracy	
Opiekun pracy	Kierownik Katedry/Zakładu
<i>podpis</i>	<i>podpis</i>
dr Adam Przybyłek	

Data oddania pracy do dziekanatu:



**POLITECHNIKA
GDAŃSKA**

WYDZIAŁ ELEKTRONIKI,
TELEKOMUNIKACJI I INFORMATYKI



OŚWIADCZENIE studenta realizującego pracę dyplomową w ramach programu o podwójnym dyplomowaniu

Imię i nazwisko: Michał Żyłowski
Data i miejsce urodzenia: 02.05.1991, Toruń
Nr albumu: 137448
Wydział: Wydział Elektroniki, Telekomunikacji i Informatyki
Kierunek: informatyka
Poziom studiów: I stopnia - inżynierskie
Forma studiów: stacjonarne

Ja, niżej podpisany(a), wyrażam zgodę/nie wyrażam zgody* na korzystanie z mojej pracy dyplomowej zatytułowanej: Programowalna sieciowa gra RTS do celów naukowych lub dydaktycznych.¹

Gdańsk, dnia

.....
podpis studenta

Świadomy(a) odpowiedzialności karnej z tytułu naruszenia przepisów ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2006 r., nr 90, poz. 631) i konsekwencji dyscyplinarnych określonych w ustawie Prawo o szkolnictwie wyższym (Dz. U. z 2012 r., poz. 572 z późn. zm.),² a także odpowiedzialności cywilno-prawnej oświadczam, że przedkładana praca dyplomowa została opracowana przeze mnie samodzielnie.

Niniejsza praca dyplomowa była wcześniej podstawą urzędowej procedury związanej z nadaniem tytułu zawodowego w ramach programu podwójnego dyplomowania, a zakres pracy jest zgodny z zasadami tego programu.

Wszystkie informacje umieszczone w ww. pracy dyplomowej, uzyskane ze źródeł pisanych i elektronicznych, zostały udokumentowane w wykazie literatury odpowiednimi odnośnikami zgodnie z art. 34 ustawy o prawie autorskim i prawach pokrewnych.

Potwierdzam zgodność niniejszej wersji pracy dyplomowej z załączoną wersją elektroniczną.

Gdańsk, dnia

.....
podpis studenta

Upoważniam Politechnikę Gdańską do umieszczenia ww. pracy dyplomowej w wersji elektronicznej w otwartym, cyfrowym repozytorium instytucjonalnym Politechniki Gdańskiej oraz poddawania jej procesom weryfikacji i ochrony przed przywłaszczaniem jej autorstwa.

Gdańsk, dnia

.....
podpis studenta

*) niepotrzebne skreślić

¹ Zarządzenie Rektora Politechniki Gdańskiej nr 34/2009 z 9 listopada 2009 r., załącznik nr 8 do instrukcji archiwalnej PG.

² Ustawa z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym:

Art. 214 ustęp 4. W razie podejrzenia popełnienia przez studenta czynu podlegającego na przypisaniu sobie autorstwa istotnego fragmentu lub innych elementów cudzego utworu rektor niezwłocznie poleca przeprowadzenie postępowania wyjaśniającego.

Art. 214 ustęp 6. Jeżeli w wyniku postępowania wyjaśniającego zebrany materiał potwierdza popełnienie czynu, o którym mowa w ust. 4, rektor wstrzymuje postępowanie o nadanie tytułu zawodowego do czasu wydania orzeczenia przez komisję dyscyplinarną oraz składa zawiadomienie o popełnieniu przestępstwa.

Streszczenie

Głównym celem poniższej pracy było stworzenie programowalnej sieciowej gry RTS (strategii czasu rzeczywistego) połączonej z webserwisem. Projekt został podzielony na dwa moduły: (1) webserwis, pełniący rolę menu głównego oraz całej infrastruktury zarządzania rozgrywką oraz statystykami; (2) aplikacja gry. Każdy z komponentów odpowiedzialny jest za inną funkcjonalność co pozwoliło nam również na ułatwienie organizacji pracy nad projektem.

Główną funkcjonalnością gry jest możliwość pisania skryptów, które są interpretowane przez obiekty występujące w rozgrywce (np. ruchome jednostki i budynki). Naszym celem było umożliwić graczom mechanizm zautomatyzowania pewnych zachowań w przeciwieństwie do gier zręcznościowych, w których najbardziej liczy się szybkość posługiwania myszą. Gracz miał mieć możliwość zaimplementowania skryptów odpowiadających za proste zachowania jak przejście jednostki na daną pozycję, po zaawansowane reagowanie na stan pola bitwy i wspieranie sprzymierzonych jednostek.

Rozgrywki są prowadzone za pośrednictwem internetu, tak aby gracz zawsze walczył z żywym przeciwnikiem. Zrezygnowaliśmy z wbudowanej sztucznej inteligencji na rzecz przyjemności płynącej z prowadzenia rozgrywki sieciowej.

Zrealizowaliśmy wszystkie założenia, które sobie na początku postaviliśmy. Co prawda nie możemy stwierdzić, że nasza praca dokładnie odzwierciedla nasze plany, ale w aktualnym stadium nadaje się do dalszego rozwoju. Etap ewaluacji (oparty o metodę badawczą "Design Science Research"), dokonany na podstawie zebranych opinii użytkowników korzystających z naszego produktu, dostarczył nam potrzebnej oceny, dzięki której mogliśmy dostrzec błędy w naszym produkcie. Ewaluacja dostarczyła nam również sugestie, co można poprawić oraz ewentualne nowe funkcjonalności. Produkt został przyjęty z entuzjazmem, jednakże użytkownicy wyrazili również nieco zastrzeżeń (np. do intuicyjności interfejsu).

Podział pracy prezentował się następująco:

- Patryk Urban – praca nad komponentem gry i jej koordynacja.
- Andrzej Joskowski – praca nad komponentem webserwisu i jej koordynacja, koordynacja prac nad całym projektem.
- Michał Żyłowski – rozwijanie funkcjonalności w komponencie webserwisu.
- Adrian Pielech – rozwijanie funkcjonalności w komponencie gry.

Wkład w pisanie dokumentacji miał każdy z członków.

Abstract

The main purpose of this project was to create a programmable network RTS game (real time strategy) combined with web service. All the work was divided into two modules. First is a web service, component, which is acting as the main menu and game management infrastructure and collecting statictics. Second is the application of the game. Each of the components is responsible for another functionality which allowed as also to facilitate the organisation of work on the project.

The major feature of the game is the ability to write scripts that are interpreted by the objects that appear in the game (eg. mobile units and building). Our goal is to serve players the mechanism to automate certain behaviours as opposed to arcade games, where the most useful skill is the fast clicking of mouse. The player is able to implement the most basic type of behaviour as moving to the destination, as responding to the state of battlefield and supporting allied units.

Competitions are carried out via the internet, so the player always fights against a live opponent. We gave up the built-in artificial intelligence for the network experiences.

We have completed all the assumptions that we set at the beginning. It is true that we can not say that our work accurately reflects our plans, but in the current stadium is suitable for further development. Evaluation stage (based on "Design Science Research"), which we made on the basis of the opinion of the users of our product, has provided us necessary rating which allowed us to take a different point of view on the errors in our product. The evaluation also provided us suggestions on what can be improved or added. The product has been received with enthusiasm, however users said they had objections (e.g. intuitive interface).

The division of labor presented as follows:

- Patryk Urban – creating and overseeing the work on a component of the main game,
- Andrzej Joskowski – creating and overseeing the work on a component of the web service and workflow of the whole project,
- Michał Żyłowski – developing functionality in the component of the web service,
- Adrian Pielech – developing functionality in a component of the main game.

Each of members took a part in writing documentation.

Spis treści

1. Wstęp	7
1.1. RTS - co to właściwie jest? (A. J.).....	7
1.2. Cel oraz zakres projektu. (A. J.).....	7
1.3. Czym jest nasz produkt? (A. J.)	7
1.4. Kategoria produktu (A. J.)	8
1.5. Metoda badawcza (A. J.)	8
1.6. Podobne rozwiązania. (P. U.)	8
1.7. Co wyróżnia naszą grę? (P. U.)	13
1.8. Struktura pracy. (A. J.)	14
2. Analiza i projekt systemu.....	15
2.1. Dla kogo? (M. Ż.)	15
2.2. Ograniczenia i założenia (M. Ż.)	15
2.2.1. Ograniczenia związane z procesem wytwarzania	15
2.2.2. Ograniczenia związane z produktem	15
2.2.3. Założenia dotyczące produktu i jego użytkownika	16
2.3. Wymagania funkcjonalne i aktorzy (A. J.), (P. U.)	16
2.3.1. Opis aktorów	16
2.3.2. Model przypadków użycia	17
2.3.3. Specyfikacja przypadków użycia	19
2.4. Wymagania niefunkcjonalne (M. Ż.)	26
2.5. Struktura bazy danych (A. J.).....	27
3. Implementacja	31
3.1. Wykorzystane technologie	31
3.1.1. REST API (M. Ż.)	31
3.1.2. Baza danych (A. J.).....	32
3.1.3. Unity (P. U.).....	32
3.1.4. NodeJS (A. J.)	33
3.1.5. AngularJS (M. Ż.).....	34

3.1.6. Obsługa sieciowa w Unity (P. U.)	35
3.1.7. Połączenie webserwisu i Gry na poziomie protokołu (A. P.).....	39
3.1.8. Przegląd istniejących rozwiązań użytych w projekcie (A. J.)	39
3.2. Przygotowanie środowiska deweloperskiego (A. P.)	40
3.2.1. Przygotowanie środowiska deweloperskiego dla webserwisu	40
3.2.2. Przygotowanie środowiska deweloperskiego dla Gry	42
3.3. Element programowania w grze (A. P.)	43
3.4. Wytworzone artefakty (A. P.).....	44
3.5. Napotkane problemy (A. P.), (A. J.), (P. U.), (M. Ż.)	45
4. Ewaluacja	47
4.1. Weryfikacja (A. J.)	47
4.2. Walidacja (A. P.).....	48
4.3. Opinie użytkowników (P. U.)	51
4.4. Kierunki dalszego rozwoju i plany ulepszeń (M. Ż.).....	52
5. Podsumowanie (A. J.)	54
Wykaz literatury	55
Wykaz rysunków.....	56
Wykaz tabel	57
Dodatek A: Raport końcowy	59

Legenda:

(A. J.) – Andrzej Joskowski

(A. P.) – Adrian Pielech

(P. U.) – Patryk Urban

(M. Ż.) – Michał Żyłowski

1. Wstęp

1.1. RTS - co to właściwie jest?

RTS, czyli "Real-Time Strategy" to jeden z gatunków gier komputerowych [9]. Jak sama nazwa wskazuje są to gry czasu rzeczywistego [9], w których zwycięstwo uzależnione jest od przyjętej przez gracza na początku rozgrywki określonej strategii gry. Głównym zadaniem zawodnika jest dowodzenie zasobami oferowanymi przez grę (np. jednostki wojskowe, pracownicy fabryki, mieszkańcy miasta itp.) w taki sposób, aby osiągnąć zamierzony wcześniej cel, który być może doprowadzi do sukcesu, a więc wygranej rundy.

Cała rozgrywka odbywa się w czasie rzeczywistym, dzięki czemu skutki podejmowanych decyzji widoczne są niemal natychmiast [17]. W tym rodzaju rozgrywki ważne jest umiejętne podejmowanie decyzji w jak najkrótszym czasie, gdyż nawet najmniejsza zwłoka może doprowadzić do przegranej.

Przyjęło się, że datą, którą uważa się za powstanie tego typu gatunku gier są lata 80 XX wieku [18]. Na rynku pojawiły się wtedy gry "Cytron Masters", "Stonkers", "The Ancient Art of War". Tytułem, który rozślał gry typu RTS jest gra "Dune II: Battle for Arrakis". Kolejną grą RTS z pewnością znaną współczesnym pokoleniom graczy jest seria "Warcraft", która dostępna jest w wielu wersjach, z grafiką dwuwymiarową jak i trójwymiarową.

1.2. Cel oraz zakres projektu.

Projekt realizowany jest w ramach pracy dyplomowo-inżynierskiej na semestrze siódmym studiów stacjonarnych na kierunku "Informatyka" wydziału Elektroniki, Informatyki i Telekomunikacji Politechniki Gdańskiej.

Celem pracy jest stworzenie sieciowej gry strategicznej czasu rzeczywistego (RTS), w której podstawowym zadaniem graczy jest programowanie zachowań jednostek i budynków.

W ramach projektu dokonamy analizy narzędzi deweloperskich oraz komponentów wielokrotnego użycia dostępnych na licencji otwartego kodu (nasz projekt również jest niekomercyjny i zostanie udostępniony), a także udokumentujemy projekt logiczny i implementację. Postaramy się przedstawić nasze spostrzeżenia związane z użytymi w projekcie technologiami. Po utworzeniu wersji "beta" produktu zbierzemy opinie użytkowników, na których podstawie dokonamy usprawnień, naprawę błędów oraz zaimplementujemy funkcjonalność, o którą poproszą nas nasi użytkownicy.

1.3. Czym jest nasz produkt?

Produktem przedsięwzięcia, któremu stawiliśmy czoła będą dwa komponenty: wspomniana już wcześniej gra strategiczna czasu rzeczywistego (RTS) oraz serwis internetowy, służący do zarządzania rozgrywką oraz zapewniający komunikację platformy dla społeczności związanej z naszym produktem.

Gra jest rozgrywką, w której walczymy z innymi graczami podczas sieciowych starć. Gracz ma do dyspozycji widok 2D z "lotu ptaka". Głównym jego zadaniem ma być pisanie zachowań jednostek i budynków w języku skryptowym. Mapa pokryta jest "mgłą wojny", więc widzimy tylko tyle, ile widzą nasze jednostki, co zmusza nas do patrolowania terenu i wysyłania ekspedycji, aby odkryć położenie wrogów.

Serwis internetowy ma do dyspozycji funkcje takie jak rejestracja graczy, pobieranie gry, wyświetlanie statystyk gry, udostępnienie poradników do gry, tworzenie spersonalizowanej galerii zrzutów ekranu, organizacja pojedynków czyli lobby z wyszukiwaniem przeciwników.

1.4. Kategoria produktu

Podczas formułowania tematu realizowanej przez nas pracy inżynierskiej natknęliśmy się na artykuł dotyczący pewnej kategorii gier zwanej "serious games" [6]. Nazwa przetłumaczona na język polski "poważna gra" jest nieco rozmywająca obraz jej znaczenia, dlatego też pozostaniemy przy nazwie angielskiej.

Większość gier nastawiona jest na grywalizację. Nie inaczej jest z "serious games" lecz sama grywalizacja nie jest ich głównym celem. "Serious games" to gry, które nie mają zapewniać wyłącznie rozrywki, ale mają również pomóc w zdobywaniu i rozwijaniu określonych umiejętności oraz rozwiązywaniu problemów [6]. Najczęściej są to różnego rodzaju gry interaktywne, ale samo pojęcie znane jest od bardzo dawna, kiedy to gry komputerowe nie były jeszcze popularne. Gry wideo stosowane są w celach edukacyjnych [14] czy też w leczeniu chorób, fobii itp. Rozwiązania te bardzo często są skierowane do przedszkolaków pomagając im w nauce pisania, czytania, ortografii i matematyki [13].

W 1970 roku Clark Abt wydał książkę pt. "Serious Games", w której napisał "poważne gry [...] mają wyraźny i starannie obmyślany cel edukacyjny, i ich podstawowym przeznaczeniem nie jest zapewnianie rozrywki" [6], co uważa się na pierwszą definicję tego rodzaju gier. Swoją książkę tworzył bazując przede wszystkim na grach karcianych [6]. W roku 2005 Mike Zyda rozwinął tę definicję mówiąc o "serious games" w następujący sposób: "rywalizacja umysłowa, rozgrywana z komputerem, w oparciu o określone zasady, która wykorzystuje rozrywkę, aby promować i osiągać rządowe lub korporacyjne cele związane z treningiem umiejętności, edukacją, zdrowiem, polityką publiczną i komunikacją strategiczną" [15].

Tworząc naszą grę chcieliśmy, aby nie służyła ona tylko przyjemności, ale przede wszystkim żeby rozwijała naszych graczy. Z tego to powodu zakwalifikowaliśmy ją do kategorii "serious games". Nasz produkt ma na celu zainteresować gracza dziedziną programowania.

1.5. Metoda badawcza

Realizując nasz projekt postępowaliśmy zgodnie z paradygmatem "Design Science Research". "Design Science" jest to metoda badawcza zorientowana na tworzenie i ewaluację innowacyjnych i istotnych artefaktów IT [7]. March i Smith [8] wskazali cztery kategorie artefaktów IT, które mogą zostać stworzone w ramach tej metody badawczej, są to: konstrukcje, modele, metody i instancje. Według livari [16] innowacyjna gra komputerowa może być uznana

za artefakt IT typu instancja. Jako że, nasza gra rozwiązuje istotny problem odnoszący się do społeczeństwa informacyjnego – opanowanie podstawowych umiejętności programistycznych, oraz robi to w sposób innowacyjny – nauka poprzez grę, spełniliśmy wszystkie wymogi zastosowania “Design Science Research”.

Hevner i Chatterjee [7] zaproponowali pewną sekwencję kroków, których realizacja umożliwi skuteczne przeprowadzenie badania “Design Science”. Postanowiliśmy się im przyjrzeć a następnie spróbować je zrealizować. Są to:

- Identyfikowanie problemu i motywacja,
- Definicja celów badania,
- Projekt i rozwój,
- Demonstracja,
- Ewaluacja,
- Komunikacja dla dyfuzji wiedzy w społeczności badaczy.

Ponadto Hevner i Chatterjee [7] podali zestaw wytycznych dla “Design Science Research” [7]. Poniżej przedstawiliśmy je wraz z nawiązaniem do kwestii dotyczących naszego projektu:

- dostarczenie artefaktu – naszym artefaktem jest gra,
- istotność problemu – dziedzina programowania jest bardzo istotnym elementem informatyki, natomiast informatyka jest podstawą rozwoju cywilizacyjnego, nasz produkt pozwala na rozwijanie umiejętności programowania oraz ma na celu zainteresować graczy jego aspektami,
- ewaluacja artefaktu - na podstawie opinii naszych klientów zebranych poprzez ankiety chcielibyśmy dokonać walidacji naszego produktu, co z kolei da nam szansę na ulepszenie naszego oprogramowania. Wspólnie z naszym promotorem przedyskutowana została forma zebrania opinii od naszych klientów. Nie chcielibyśmy, aby uzyskane przez nas informacje były przekłamane z powodu źle skonstruowanej formy ankiety.
- wkład naukowy badań - możliwy do realizacji w kolejnym etapie badań poprzez realizację kontrolowanego eksperymentu mającego zbadać w jakim stopniu granie w naszą grę rozwija umiejętności programistyczne,
- rygor naukowy badań – realizując projekt podążaliśmy metodą badawczą “Design Science”,
- projekt jako proces poszukiwań – wytwarzanie produktu odbywać się będzie iteracyjnie, pozwoli to na ciągłą komunikację z użytkownikami, co z kolei umożliwi dostosowanie oprogramowania do ich potrzeb,
- komunikacja rezultatów – w serwisie będącym jednym z wytworzonych przez nas artefaktów dostępna będzie instrukcja dla nowych graczy wyjaśniająca w jaki sposób należy korzystać z produktu.

1.6. Podobne rozwiązania.

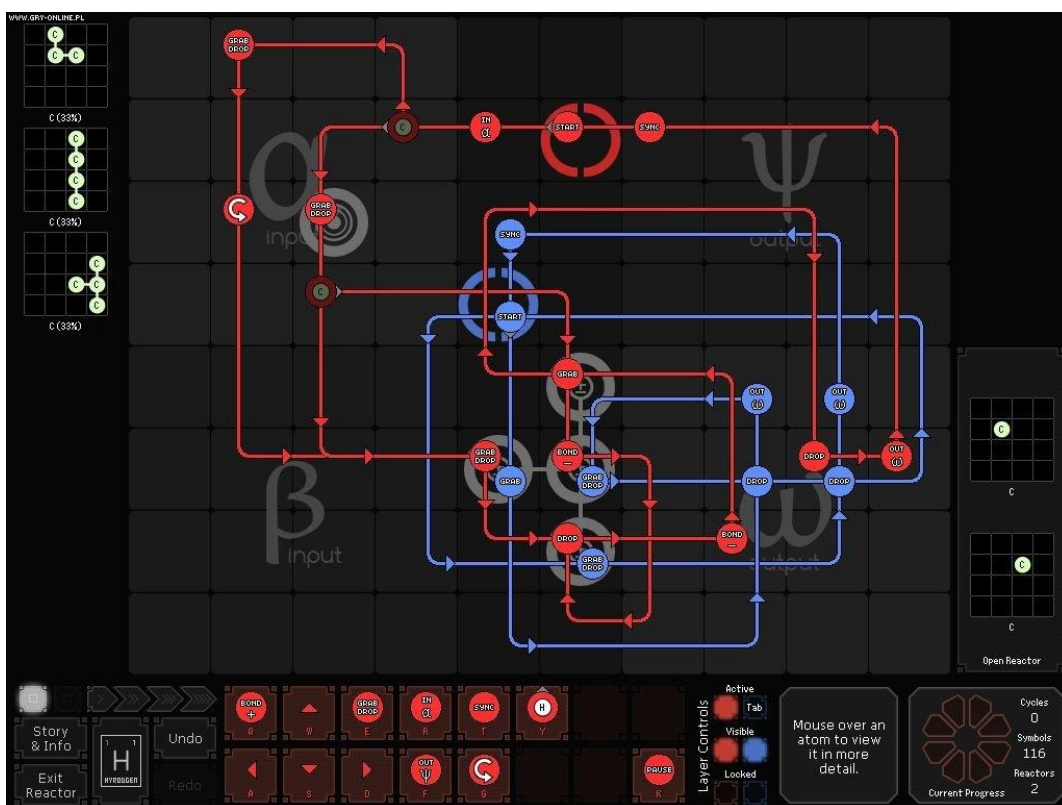
Colobot (zapisywane również CoLoBoT) jest edukacyjną grą strategiczną czasu rzeczywistego z elementami programowania. Wyprodukowana została przez firmę Epsitec i została wydana 3 listopada 2001 roku przez Alsyd. Jej kontynuacja znana pod nazwą Ceebot ukazała się na rynku w 2005 roku. Polska społeczność graczy poprosiła, aby Epsitec zmienił licencję na GNU GPL v3 co pozwoliło na wydawanie kolejnych wersji ulepszonych przez społeczność programistów amatorów.



Rys. 1.1. Grafika pochodzi ze strony <http://gogolev.net/>

Colobot nie wymaga taktyki, wystarczy w odpowiedni sposób prowadzić rozgrywkę. W przeciwieństwie do innych gier RTS tutaj kontrolujemy każdą jednostkę osobno z perspektywy kamery “boga”. Ponieważ sami nie bylibyśmy w stanie obsługiwać wiele jednostek w tym samym czasie, gra posiada własny sposób na programowanie robotów. Jest to specjalny język zwany CBOT, który konstrukcyjnie przypomina trochę połączenie języka Java i C++ (rys. 1.1). Samo programowanie robotów jest podobne do pisania kodu metodą brute-force, co właściwie sprowadza się do mieszania myślenia z improwizowaniem.

SpaceChem (rys. 1.2) to kolejny przykład gry RTS. Przeznaczona ona jest dla użytkowników popularnych “pecetów”. Jest produktem autorstwa niezależnego studia Zachtronics Industries. Zadaniem gracza jest rozwiązanie szeregu zagadek logicznych w celu stworzenia określonych związków chemicznych.



Rys. 1.2. Grafika pochodzi ze strony <http://www.indiegames.com/>

W tej grze łączymy umiejętności analitycznego myślenia z podstawami chemii. Na start wiemy jakie posiadamy dane wejściowe oraz jakich danych wejściowych od nas system oczekuje. Naszym zadaniem jest tak stworzyć algorytm łączenia atomów, aby działał on sprawnie, szybko, żeby nie występowały w nim kolizje oraz aby wykorzystywał on jak najmniej operacji. Wszystko to sprowadza się do poradzenia sobie z szeregiem zagadek logicznych bazujących na automatyce.

Kolejnym przedstawicielem RTS'ów jest Robocode (rys. 1.3). Jest to open source'owa gra programistyczna stworzona przez Mathewa Nelsona. Obecnie rozwijana jest głównie przez Flemminga N. Larsena oraz Pavla Šavarę. Została zaprojektowana jako aplikacja pomagająca w nauce programowania w języku Java.

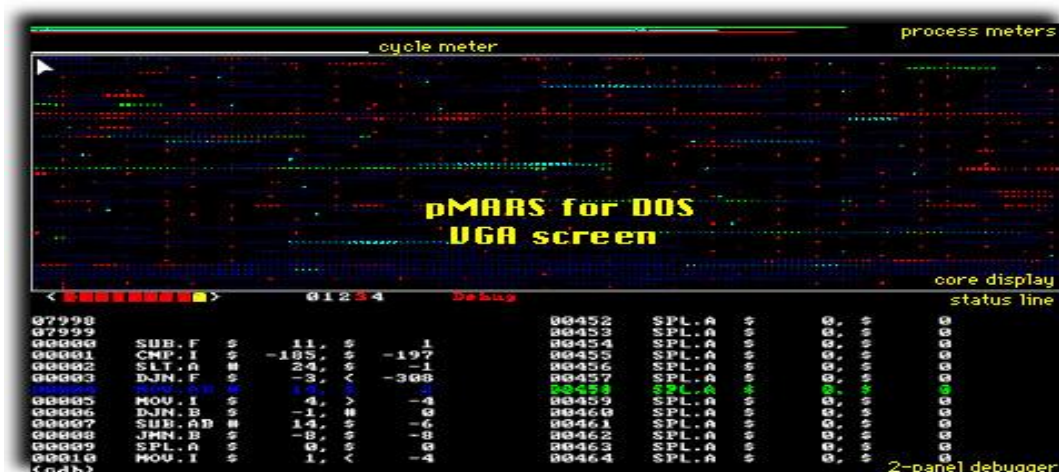
Zadaniem graczy jest napisanie programu dla wirtualnego robota w języku stworzonym na potrzeby gry. Musimy napisać algorytm, który nasz robot będzie wykonywał podczas starć z innymi robotami sterowanymi komputerowo.



Rys. 1.3. Grafika pochodzi ze strony <http://sourceforge.net/projects/robocode/>

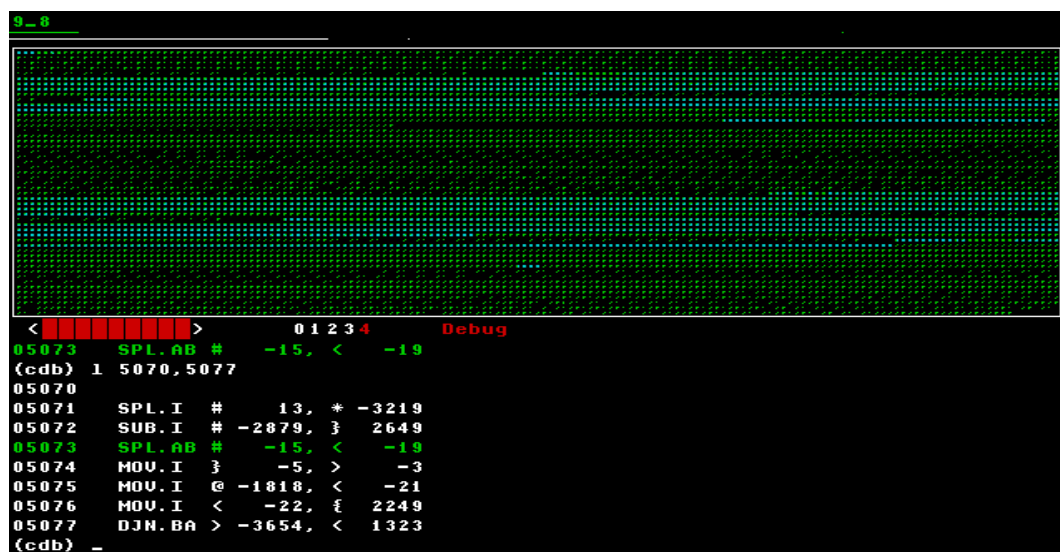
Początkujący gracze skupiają się na unikach w stylu ruch do przodu, do tyłu i obracaniu działku dookoła. Stworzenie prostego robota nie jest trudne, ponieważ wystarczy tylko poświęcić na to tylko kilka linii kodu. Jednakże nie jesteśmy niczym ograniczeni w tworzeniu oprogramowania i możemy implementować skomplikowane mechanizmy uników, czy też uczenia się otoczenia.

Innym przykładem jest Corewars (rys. 1.4), czyli gra komputerowa, która polega na pisaniu programu dla naszego bohatera (core'a). Gra wykorzystuje język podobny do assemblera zwany Redcode. Na wirtualnej arenie spotykają się dwa programy w specjalnie wydzielonej pamięci komputerowej. Jednostki wykonują swoje instrukcje na przemian w turach, jedna instrukcja na turę. Nad przebiegiem całej walki pieczę sprawują program "sędzia". Nasze programy mogą przemieszczać się dowolnie po rdzeniu, dzielić się na niezależne procesy, co może przyspieszyć ekspansję jak i również doprowadzić do samozniszczenia.



Rys. 1.4. Grafika pochodzi ze strony <http://www.koth.org/pmars/>

Zwroty "walka programów" i "zabicie programu" brzmią nieco abstrakcyjnie, ale łatwo je wyjaśnić. Jeden z rozkazów Redcode (DAT) nie może być wykonany - jeśli któryś z procesów próbuje to robić, zostaje zabity (rys. 1.5). Program przegrywa walkę, gdy zginą wszystkie jego procesy. Walka polega na tym, aby zmusić wszystkie procesy przeciwnika do wykonania zabójczej instrukcji tj. np. nadpisanie (skasowanie) jego kodu.



Rys. 1.5. Grafika pochodzi ze strony <http://www.infionline.net/~wtnewton/corewar/evol/>

1.7. Co wyróżnia naszą grę?

Co prawda sam pomysł na programowalną grę nie jest nowy, aczkolwiek w naszym wykonaniu będzie on innowacyjny. Przede wszystkim będzie to czystej postaci RTS, czyli strategia czasu rzeczywistego, która wyglądem nie będzie specjalnie różnić się od innych tego typu produkcji. Główną zmianą będzie fakt, że zachowania jednostek będziemy pisali sami. Każda wyprodukowana jednostka będzie musiała zostać zaopatrzona w swój program, który będzie wykonywać. W przeciwieństwie do innych gier nie jesteśmy tutaj ograniczeni do programowania tylko jednej jednostki, lecz tutaj ilość naszych wirtualnych wojowników zależy

tylko od nas samych oraz od naszego zaplecza ekonomicznego. Musimy nadal pamiętać, że gramy w strategię, a więc budynki również muszą posiadać swój program, który będą realizowały na potrzeby produkowania nowych jednostek i magazynowania surowców.

Kolejną i chyba największą innowacją będzie fakt, że walczyć będziemy z żywymi graczami w pojedynkach sieciowych, a nie z wirtualnymi przeciwnikami komputerowymi. Doda to do naszej gry motyw rywalizacji, myślenia oraz ludzką nieprzewidywalność. Zwycięstwo nie będzie zależeć od tego kto ma lepszy refleks i wykonuje punkty z ustalonej kolejki budowania (jak to robią koreańczycy w Starcraftie), ale od tego jak wydajny jest nasz algorytm i na ile elastyczne jest jego zachowanie w różnych sytuacjach. Uważamy, że czym innym jest nakazać wszystkim jednostkom spotkać się w jednym miejscu i ruszyć grupą do ataku, a co innego wycofywać ranne jednostki i zastępować je nowymi, świeżymi, a następnie wspierać bardziej potrzebujące flanki.

1.8. Struktura pracy.

Na początku naszej pracy przedstawiliśmy ogólny pomysł obrazujący to, co chcemy zrobić w ramach projektu inżynierskiego. Dokonaliśmy przeglądu rynku w celu odnalezienia konkurencyjnego dla nas oprogramowania. Przedstawiliśmy kilka podobnych rozwiązań, które różnie interpretują postawiony sobie przez nas problem.

W kolejnych rozdziałach chcielibyśmy dokonać analizy wymagań oraz przedstawić projekt naszego systemu. Przedstawimy użytkownika naszego produktu, skupimy się na ograniczeniach oraz założeniach. Krótko przedstawimy użyte przez nas technologie wraz z argumentami przemawiającymi za ich użyciem. Opiszemy również proces implementacji oraz napotkane przez nas podczas pracy problemy.

Jednym z punktów naszej pracy będzie weryfikacja naszego produktu. Chcielibyśmy zastanowić się, które z założeń udało nam się zrealizować, ile z nich zmieniło się w trakcie pracy oraz co najważniejsze, chcielibyśmy przedstawić opinie naszych użytkowników dotyczące naszego systemu. Na ich podstawie chcielibyśmy sformułować wnioski, a następnie zastanowić się w jaki sposób moglibyśmy ulepszyć nasze oprogramowanie oraz jakich błędów nie popełniać w przyszłości tworząc inne projekty.

W załączniku do naszej pracy umieścimy krótki raport z wykonanych prac, a w nim więcej technicznych aspektów oraz trochę informacji o organizacji pracy w zespole.

2. Analiza i projekt systemu

2.1. Dla kogo?

Produkt jest skierowany przede wszystkim do dwóch grup odbiorców - do początkujących programistów (również osób, które w ogóle nie mają żadnej wiedzy na temat programowania) jak i do zawodowych programistów. Grupy te bezpośrednio są związane z misją i pomysłem na produkt - "Uczyć programowania poprzez zabawę".

Początkujący programista może rozwijać swoje umiejętności poprzez pisanie wraz z upływem czasu coraz bardziej zaawansowanych skryptów i rozwiązań w swoim projekcie.

Pomysł na zainteresowanie naszą grą zaawansowanych programistów pojawił się w naszej pracy nieco później. Coraz popularniejsze stają się turnieje organizowane dla zawodowych graczy (np. turnieje w "League of Legends" czy też "Counter-Strike"). Niestety rywalizacja w nich w dużej mierze zależy od sprawnego posługiwania się myszą oraz jak najszybszym posługiwaniem się lewym przyciskiem myszy oraz opiera się na wyuczonej i powtarzalnej strategii. Nasz projekt daje szansę bardziej zaawansowanym graczom - programistom - na sprawiedliwe uczestnictwo w takich turniejach. To umiejętności przewidywania i reagowania na zmiany będą tutaj kluczowe. Dodatkowym atutem będą zaawansowane i szybkie techniki programowania. Każda gra będzie inna ze względu na różnorodne sposoby jej prowadzenia przez przeciwników. Ostatecznie to rozumowanie weźmie górę nad wyuczonym, mało kreatywnym "szybkim klikaniem".

2.2. Ograniczenia i założenia

2.2.1. Ograniczenia związane z procesem wytwarzania

Proces wytwarzania opisywanego systemu nakłada na nas kilka kluczowych ograniczeń:

- Czasowe: na przygotowanie systemu począwszy od projektu do końcowej implementacji oraz wdrożenia mieliśmy około 10 miesięcy.
- Ludzkie: w projekcie uczestniczyło czterech programistów z różnymi zainteresowaniami zawodowymi. Projekt wymagał od nich poznania technologii wytwarzania.
- Sprzętowe: projekt jest grą komputerową sprzęt biorący udział w procesie wytwarzania musi być w stanie uruchamiać nowoczesne oprogramowanie i gry komputerowe.

2.2.2. Ograniczenia związane z produktem

W ramach produktu również możemy wskazać kilka ograniczeń. Są to:

- Gra jest sieciowa, a co za tym idzie wymuszamy na użytkowniku posiadanie dostępu do internetu i ograniczamy się wszystkimi narzutami i problemami związanymi z siecią [21].
- Sprzęt komputerowy, na którym będzie uruchamiany nasz system powinien być w stanie obsługiwać współczesne gry komputerowe [21].
- Format danych wejściowych - skrypty użytkownika, które są używane w naszym produkcie będą zapisywane w formacie tekstowym.
- Wymaga się od użytkownika zainstalowania przeglądarki internetowej i umiejętności korzystania ze stron internetowych.
- Wraz z produktem należy dostarczyć odbiorcy podręcznik użytkownika, przykładowe fragmenty kodu skryptów. Należy przygotować tzw. "tutoriale", czyli krótkie materiały szkoleniowe dla nowych użytkowników.
- Produkt będzie wymagał wdrożenia na serwerze www wraz z bazą danych.
- Należy przygotować instalator celem łatwego wdrożenia produktu u użytkownika końcowego [24].

2.2.3. Założenia dotyczące produktu i jego użytkownika

Na potrzeby realizacji projektu potrzebowaliśmy kilku założeń związanych z użytkownikiem końcowym:

- Użytkownik posiada dostęp do internetu oraz sprzęt zdolny do uruchomienia gry.
- Odbiorca końcowy ma minimalne umiejętności programistyczne.
- Zakładamy, że użytkownik końcowy ma chociażby minimalne doświadczenie w poruszaniu się po stronach internetowych oraz prowadzeniu interakcji w grach.

2.3. Wymagania funkcjonalne i aktorzy

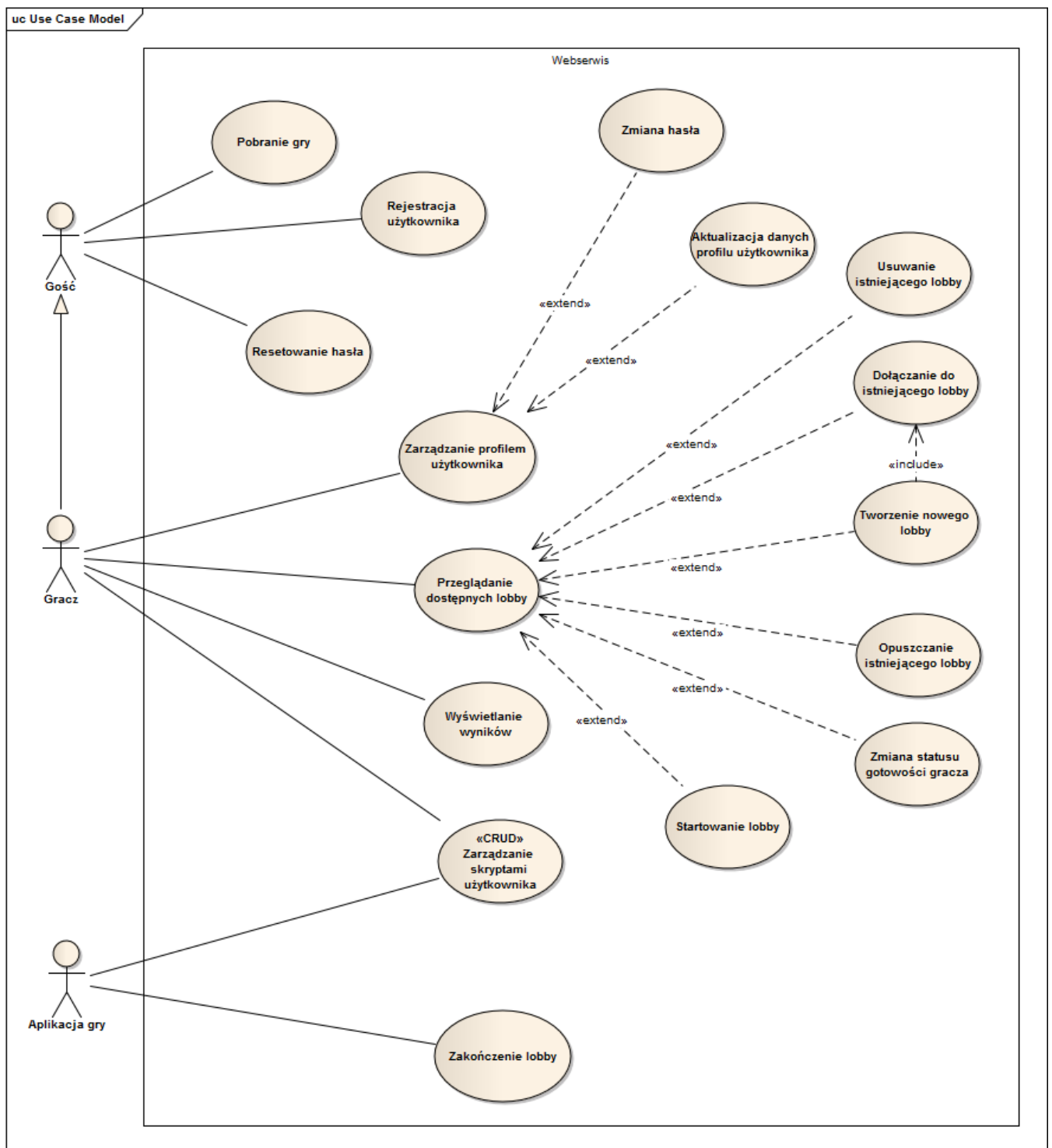
2.3.1. Opis aktorów

W naszym projekcie możemy wyróżnić następujących aktorów:

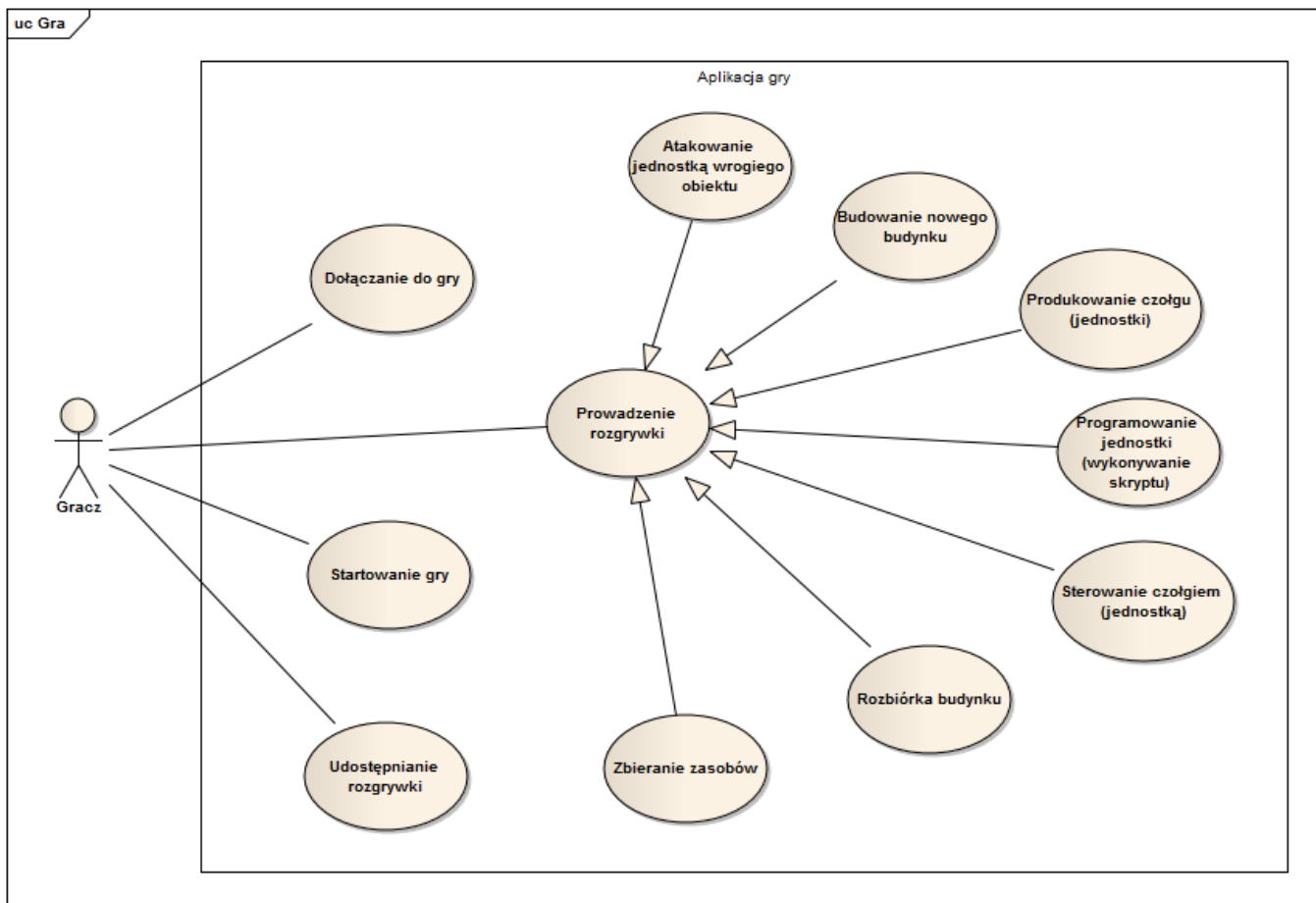
- Gość - osoba, która korzysta z webserwisu, ale nie posiada konta w systemie.
- Gracz - osoba, która posiada konto w systemie i jest zalogowana.
- Aplikacja gry - podsystem korzystający z webserwisu.

Diagram przypadków użycia dla webserwisu jest przedstawiony na rys. 2.1. Osobny diagram przypadków użycia znajduje się na drugim diagramie (rys.2.2), który reprezentuje aplikację gry.

2.3.2. Model przypadków użycia



Rys 2.1. Diagram przypadków użycia dla podsystemu webserwis



Rys 2.2. Diagram przypadków użycia dla podsystemu aplikacji gry

2.3.3. Specyfikacja przypadków użycia

Tabela 2.1: Przypadek użycia "Pobranie gry"

Nazwa	Pobranie gry
Warunki początkowe	Brak
Opis	Gość pobiera grę na dysk twardy swojego komputera.
Sytuacje wyjątkowe	Brak
Warunki końcowe	Gość pobrał folder skompresowany "zip" na dysk twardy swojego komputera.

Tabela 2.2: Przypadek użycia "Rejestracja użytkownika"

Nazwa	Rejestracja użytkownika
Warunki początkowe	Brak
Opis	Gość wybiera opcję "Rejestracja". System w odpowiedzi dostarcza formularz, do którego gość wpisuje swoje dane. Po kliknięciu przycisku "Zarejestruj mnie" konto zostaje utworzone w bazie, a na podany adres email zostaje wysłana wiadomość z linkiem aktywacyjnym.
Sytuacje wyjątkowe	Gość podał dane w niepoprawnym formacie: system udziela podpowiedzi oraz blokuje możliwość rejestracji.
Warunki końcowe	W bazie danych zostaje utworzone nieaktywne konto dla rejestrującego się gościa.

Tabela 2.3: Przypadek użycia "Resetowanie hasła"

Nazwa	Resetowanie hasła
Warunki początkowe	Gość zapomniał hasła potrzebnego do zalogowania się na swoje konto.
Opis	Gość klika na przycisku "Zapomniałeś hasła". W wyświetlonym formularzu podaje adres email, a następnie potwierdza go używając przycisku. System informuje o tym, że na podany adres email została wysłana wiadomość zawierająca link umożliwiający zresetowanie hasła. Gość klika na otrzymany link, ustawia nowe hasło.
Sytuacje wyjątkowe	<ol style="list-style-type: none">1. Niepoprawny format adresu email: komunikat o niepoprawnym adresie email.2. Użytkownik nie istnieje w systemie: komunikat o błędzie.
Warunki końcowe	Gość ustawił dla swojego konta nowe hasło i może zalogować się na swoje konto.

Tabela 2.4: Przypadek użycia "Zmiana hasła"

Nazwa	Zmiana hasła
Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o jego profilu. Korzysta tam z formularza zmiany hasła, którego funkcjonalność oparta jest o podanie aktualnego hasła.
Sytuacje wyjątkowe	Podane hasło jest nieprawidłowe: komunikat o błędzie autoryzacji.
Warunki końcowe	Gracz zmienił hasło dla swojego konta.

Tabela 2.5: Przypadek użycia "Aktualizacja danych profilowych"

Nazwa	Aktualizacja danych profilu użytkownika
Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o jego profilu. Widok prezentujący dane użytkownika jest edytowalny i użytkownik korzysta z niego w celu aktualizacji informacji.
Sytuacje wyjątkowe	Gracz dokonuje aktualizacji adresu email na adres o niepoprawnym formacie: system wyświetla informację o niepoprawnym formacie adresu email.
Warunki końcowe	Gracz zaktualizował dane dla swojego konta.

Tabela 2.6: Przypadek użycia "Zarządzanie profilem użytkownika"

Nazwa	Zarządzanie profilem użytkownika
Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o jego profilu.
Sytuacje wyjątkowe	Brak
Warunki końcowe	Gracz wyświetlił informacje o swoim profilu.

Tabela 2.7: Przypadek użycia "Tworzenie nowego lobby"

Nazwa	Tworzenie nowego lobby
Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich lobby. Wybiera opcję "Utwórz grę". Następnie wypełnia wyświetlony przez system formularz, po czym zatwierdza dane używając przycisku.
Sytuacje wyjątkowe	1. Gracz jest już przypisany do istniejącego lobby: system wyświetla informację o błędzie.

Warunki końcowe	Gracz utworzył nowe lobby, został przekierowany do listy istniejących lobby, na którym widnieje nowo utworzone przez niego lobby, Gracz został automatycznie przypisany do swojego lobby.
------------------------	---

Tabela 2.8: Przypadek użycia "Usuwanie istniejącego lobby"

Nazwa	Usuwanie istniejącego lobby
Warunki początkowe	Gracz jest zalogowany na swoje konto, ma utworzone lobby.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich lobby. Jako właściciel lobby ma możliwość usunięcia go z listy dostępnych lobby.
Sytuacje wyjątkowe	Brak
Warunki końcowe	Gracz usunął istniejące lobby, użytkownicy do niego przypisani mogą dołączyć do innego lobby.

Tabela 2.9: Przypadek użycia "Dołączanie do istniejącego lobby"

Nazwa	Dołączanie do istniejącego lobby
Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich lobby. Używając przycisku "+" może dołączyć do istniejącego lobby.
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Gracz jest już przypisany do innego lobby: system informuje o błędzie. 2. Brak wolnych miejsc: system informuje o błędzie. 3. Lobby chronione hasłem: system wyświetla formularz, do którego należy wpisać hasło dostępowe, następnie system pozwala dołączyć do lobby lub wyświetla informację o błędzie.
Warunki końcowe	Gracz dołączył do istniejącego lobby, może zasignalizować swoją gotowość do rozpoczęcia rozgrywki.

Tabela 2.10: Przypadek użycia "Opuszczanie istniejącego lobby"

Nazwa	Opuszczanie istniejącego lobby
Warunki początkowe	Gracz jest zalogowany na swoje konto, dołączył do lobby (nie jest jego właścicielem).
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich lobby. W widoku aktywnego lobby użytkownik ma do dyspozycji przycisk "opuść".
Sytuacje wyjątkowe	Brak
Warunki końcowe	Gracz opuścił lobby, zwiększyła się liczba dostępnych miejsc.

Tabela 2.11: Przypadek użycia "Przeglądanie dostępnych lobby"

Nazwa	Przeglądanie dostępnych lobby
--------------	-------------------------------

Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich lobby.
Sytuacje wyjątkowe	Brak
Warunki końcowe	Gracz widzi listę dostępnych lobby.

Tabela 2.12: Przypadek użycia "Zmiana statusu gotowości gracza"

Nazwa	Zmiana statusu gotowości gracza
Warunki początkowe	Gracz jest zalogowany na swoje konto, jest przypisany do lobby.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich lobby, w widoku aktywnego lobby ma do dyspozycji przycisk "zmień status".
Sytuacje wyjątkowe	1. Gra się już rozpoczęła: Gracz nie może zmienić statusu
Warunki końcowe	Gracz zmienił status swojej gotowości do gry.

Tabela 2.13: Przypadek użycia "Startowanie lobby"

Nazwa	Startowanie lobby
Warunki początkowe	Gracz jest zalogowany na swoje konto, jest przypisany do lobby, które sam utworzył.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich lobby, w widoku aktywnego lobby ma do dyspozycji przycisk "start".
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Gra się już rozpoczęła: Gracz nie może ponownie wystartować gry, system wyświetli ostrzeżenie. 2. Gra się już zakończyła: użytkownik po naciśnięciu "start" dokona restartu lobby. 3. Wolne miejsca: system wyświetli ostrzeżenie, że do lobby dołączyło zbyt mało graczy.
Warunki końcowe	Gracz wystartował grę.

Tabela 2.14: Przypadek użycia "Zarządzanie skryptami użytkownika"

Nazwa	Zarządzanie skryptami użytkownika
Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się informacje o wszystkich skryptach należących do niego. Ma możliwość utworzenia nowego skryptu, edycji istniejących i ich usuwania.
Sytuacje wyjątkowe	Brak

Warunki końcowe	Gracz wyświetlił listę ze swoimi skryptami.
------------------------	---

Tabela 2.15: Przypadek użycia "Wyświetlanie wyników"

Nazwa	Wyświetlanie wyników
Warunki początkowe	Gracz jest zalogowany na swoje konto.
Opis	Gracz przechodzi do sekcji, w której znajdują się wyniki użytkowników.
Sytuacje wyjątkowe	Brak
Warunki końcowe	Gracz widzi statystyki.

Tabela 2.16: Przypadek użycia "Zakończenie lobby"

Nazwa	Zakończenie lobby.
Warunki początkowe	Wszyscy gracze zakończyli grę.
Opis	Webserwis przyjmuje informacje z wynikami poszczególnych graczy, wraz z otrzymaniem wyniku ostatniego z graczy zmienia status lobby na "finished".
Sytuacje wyjątkowe	Brak
Warunki końcowe	Lobby zostaje zatrzymane.

Tabela 2.17: Przypadek użycia "Udostępnianie rozgrywki."

Nazwa	Udostępnianie rozgrywki.
Warunki początkowe	Gracz będący masterem, rozpoczął grę z poziomu webserwisu
Opis	Rozpoczęło się uruchamianie aplikacji właściwej (gry). Gra bazując na parametrach startowych przechodzi w tryb udostępniania rozgrywki (oczekuje na kolejnych graczy)
Sytuacje wyjątkowe	Gracz nie zaakceptował procesu uruchamiania gry. Gra się nie uruchomiła.
Warunki końcowe	Gra została uruchomiana w trybie udostępniania rozgrywki i oczekuje na kolejnych graczy, aby do niej dołączyli.

Tabela 2.18: Przypadek użycia "Dołączanie do gry"

Nazwa	Dołączanie do gry.
Warunki początkowe	Gracz będący masterem, rozpoczął grę z poziomu webserwisu

Opis	Rozpoczęło się uruchamianie aplikacji właściwej (gry). Gra bazując na parametrach startowych przechodzi w tryb dołączania i próbuje dołączyć do gry udostępnianej przez mastera.
Sytuacje wyjątkowe	Gra nie połączyła się z serwerem. Gracz nie dołączył do gry.
Warunki końcowe	Gra została uruchomiona w trybie dołączania i połączyła się z serwerem. Gracz odnajduje się wśród użytkowników lobby, czekając na pozostałych graczy.

Tabela 2.19: Przypadek użycia "Startowanie gry"

Nazwa	Startowanie gry.
Warunki początkowe	W lobby gry znajduje się gracz master oraz pozostali gracze o statusie "ready"
Opis	W momencie gdy gra wykryje, że ostatni z graczy zmienił swój status na "ready" następuje automatyczne rozpoczęcie gry właściwej.
Sytuacje wyjątkowe	Brak
Warunki końcowe	Wszyscy gracze rozpoczęli grę.

Tabela 2.20: Przypadek użycia "Produkowanie czołgu (jednostki)"

Nazwa	Produkowanie czołgu (jednostki).
Warunki początkowe	Gracz posiada zbudowane "baraki" oraz posiada odpowiednią ilość surowców.
Opis	Gracz klika na swoje "baraki", które następnie zostają zaznaczone. W specjalnym menu z prawej strony, gracz wybiera z listy jednostkę, którą chce wyprodukować. Następnie klika na nią co powoduje zakolejkowanie jej na taśmie produkcyjnej i zredukowanie zasobów o odpowiednią ilość.
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Gracz nie posiada odpowiedniej ilości surowców - jednostka nie zostaje zakolejkowana. 2. Kolejna produkcyjna jest pełna - jednostka nie zostaje zakolejkowana. 3. Gracz utracił "baraki" - produkowana jednostka oraz przeznaczone fundusze zostają utracone.
Warunki końcowe	Jednostka zostaje wyprodukowana przez "baraki" i jest pod kontrolą gracza

Tabela 2.21: Przypadek użycia "Sterowanie czołgiem (jednostką)"

Nazwa	Sterowanie czołgiem (jednostką)
Warunki początkowe	Gracz zaznaczył własną, żywą jednostkę
Opis	Gracz wydaje rozkaz przemieszczenia się za pomocą lewego przycisku myszy. Jednostka udaje się w miejsce wskazywane przez kursor
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Gracz kliknął lewym przyciskiem myszy na pole inne niż ziemia - nastąpiła zmiana zaznaczenia, jednostka nie wykonuje ruchu. 2. Gracz wybrał miejsce niedostępne dla jednostki - jednostka dojedzie do najbliższego dostępnego miejsca względem wybranego celu.
Warunki końcowe	Jednostka rozpoczęła ruch w kierunku wybranego celu

Tabela 2.22: Przypadek użycia "Atakowanie jednostką wrogiego obiektu"

Nazwa	Atakowanie jednostką wrogiego obiektu
Warunki początkowe	Gracz posiada zaznaczony żywy czołg
Opis	Gracz wydaje rozkaz ataku się za pomocą lewego przycisku myszy. Jednostka rozpoczyna atak obiektu wskazywanego przez kursor myszy
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Gracz kliknął lewym przyciskiem myszy na pole ziemia - nastąpiła zmiana rozkazu, jednostka rozpoczyna ruch do wybranego miejsca. 2. Wroga jednostka rozpoczyna ucieczkę - jednostka gracza przerywa atakowanie z powodu utraty celu z obszaru zasięgu
Warunki końcowe	Jednostka rozpoczęła atak wybranego wrogiego celu

Tabela 2.23: Przypadek użycia "Budowanie nowego budynku"

Nazwa	Budowanie nowego budynku
Warunki początkowe	Gracz posiada zaznaczoną jednostkę inżynierską oraz posiada odpowiednią ilość surowców.
Opis	Gracz wybiera interesujący go budynek z menu znajdującego się po prawej stronie ekranu. Klika na wybrany budynek, w tym momencie musi zdecydować, w którym miejscu chciałby postawić nowy budynek. Lokację docelową wskazuje za pomocą kursora myszy i zatwierdza lewym przyciskiem myszy.
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Gracz wybrał miejsce niedozwolone dla budowy budynku - próba zbudowania budynku w tym miejscu zostaje zablokowana. 2. Gracz nacisnął prawy przycisk myszy - tryb stawiania nowego budynku został wyłączony. 3. Gracz nie posiada odpowiedniej ilości surowców - budowa budynku nie została zatwierdzona.
Warunki końcowe	Na wskazanej lokacji pojawiają się fundamenty nowego budynku. Budynek jest gotowy do budowy

Tabela 2.24: Przypadek użycia "Rozbiórka budynku"

Nazwa	Rozbiórka budynku
Warunki początkowe	Gracz posiada zaznaczony własny, żywy budynek
Opis	Gracz wybiera z menu po prawej stronie opcję rozbiórki budynku. Budynek automatycznie zostaje rozebrany w określonym czasie. Budynek znika a część surowców zużytych na jego konstrukcję wraca do gracza.
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Budynek został zniszczony podczas rozbiórki - żadne zasoby nie zostają dodane do konta gracza.
Warunki końcowe	Budynek został zniszczony podczas rozbiórki - żadne zasoby nie zostają dodane do konta gracza.

Tabela 2.25: Przypadek użycia "Zbieranie zasobów"

Nazwa	Zbieranie zasobów
--------------	-------------------

Warunki początkowe	Gracz posiada zaznaczoną jednostkę zbierającą ("harvester") oraz aktywny budynek rafinerii.
Opis	Gracz klika na złoża rudy lewym przyciskiem myszy. Jednostka zbierająca wyrusza w kierunku złoża. Po dotarciu do celu rozpoczyna proces zbierania zasobu. Po zapelnieniu zbiorników rozpoczyna ruch w kierunku najbliższej rafinerii. Po dotarciu do niej, jednostka rozpoczyna proces wyładunku surowca. Po skończeniu rozładunku, jednostka zbierająca wraca do złoża o ile ono jeszcze istnieje i rozpoczyna przedstawiony powyżej proces od nowa.
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Złoże zostało wyczerpane - jednostka przerywa wydobywanie. 2. Budynek rafinerii został zniszczony - jednostka przerywa wydobywanie w momencie zapelnienia zbiorników.
Warunki końcowe	Jednostka zbiera surowce ze złoża do rafinerii, cyklicznie.

Tabela 2.26: Przypadek użycia "Programowanie jednostki (wykonywanie skryptu)"

Nazwa	Programowanie jednostki (wykonywanie skryptu)
Warunki początkowe	Gracz posiada zaznaczoną własną jednostkę, posiada utworzony skrypt.
Opis	Gracz wybiera z menu po prawej stronie opcję "Skryptowe okno". Z rozwijanego menu gracz wybiera nazwę interesującego go skryptu. Następnie klika opcję "Zaprogramuj obiekt"
Sytuacje wyjątkowe	<ol style="list-style-type: none"> 1. Gracz zamknął główne okno skryptowe - proces został przerwany
Warunki końcowe	Wybrana jednostka rozpoczyna wykonywanie skryptu wybranego przez gracza

2.4. Wymagania niefunkcjonalne

Wymagania poza funkcjonalne, które chcemy spełnić w naszym projekcie celem zapewnienia jakości produktu to:

- ochrona - w naszym produkcie będziemy żądać od użytkownika kilku jego danych osobowych oraz hasła. Są to dane wrażliwe, którym szczególnie należy zapewnić poufność,
- przenośność - webserwis powinien działać poprawnie i identycznie na wszystkich przeglądarkach,
- niezawodność - chcemy wyeliminować błędy w działaniu produktu. Nie chcemy aby pojedynek graczy został przerwany lub w wyniku błędnych obliczeń jeden z graczy miał większą szansę na zwycięstwo,
- dostępność - chcemy osiągnąć możliwie wysoką dostępność serwera gry (webserwisu) tak, aby rozgrywkę można było prowadzić w dowolnej chwili,
- wydajność - chcemy osiągnąć zadowalający poziom wydajności zarówno na styku gra → gracz, jak i gra → API serwer. Pierwszy styk można zrealizować niewymagającym poziomem skomplikowania grafiki, optymalnymi algorytmami etc. Drugi optymalizując „endpointy” i liczbę zapytań do możliwie niskiego poziomu,

- interfejs użytkownika - zarówno gra jak i webserwis powinien bazować na doświadczeniach użytkownika z innych gier oraz stron internetowych.

2.5. Struktura bazy danych

Baza danych to niezwykle ważny element naszego systemu. W niej gromadzimy wszystkie niezbędne dane potrzebne dla poprawnego funkcjonowania webserwisu oraz gry. Znajdują się tam informacje o użytkownikach, skryptach itp., ale to nie wszystko. Przechowujemy w niej również tymczasowe tokeny, które służą użytkownikom do uwierzytelnienia się podczas wykonywania akcji takich jak resetowanie hasła czy aktywacja konta.

Istnieje kilka podejść w projektowaniu bazy danych. W naszym przypadku mamy do czynienia z podejściem NoSQL (szczegóły w podrozdziale poświęconym technologii MongoDB (podrozdział 2.6.2). W skrócie mówiąc w podejściu takim nie budujemy modelu relacyjnego lecz przechowujemy osobne kolekcje danych zorientowane na szybki odczyt i zapis [19].

W systemie, który stworzyliśmy, dane podzieliliśmy na pięć oddzielnych kolekcji zróżnicowanych pod względem tematycznym. Poniżej przedstawiamy schemat struktury naszej bazy danych wraz z krótkimi opisami.

Users - kolekcja przechowująca dane dotyczące użytkowników zarejestrowanych w systemie. Struktura pojedynczej krotki prezentuje się następująco (opis pól przedstawia tabela 2.27):

```
{
  "_id": ObjectId("55eb4ebe7d504fde0a55620a"),
  "activated": {true/false},
  "created": {timestamp},
  "email": {string},
  "hobby": {string},
  "location": {string},
  "name": {string},
  "password": {string},
  "surname": {string},
  "username": {string}
}
```

Tabela 2.27: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Users”

Nazwa pola	Opis pola
_id	Identyfikator wygenerowany automatycznie przez silnik bazy danych.
activated	Pole określające czy konto użytkownika zostało aktywowane.
created	Pole zawiera informację o dacie utworzenia konta.
email	Adres email użytkownika.
hobby	Wartość tekstowa opisująca zainteresowania użytkownika.
location	Miejsce zamieszkania użytkownika.
name	Imię.
password	Hasło służące do późniejszego logowania się do systemu.
surname	Nazwisko użytkownika.
username	Login, nazwa użytkownika w systemie.

Tokens - kolekcja przechowująca tokeny uwierzytelniające użytkowników potrzebne w procesie resetowania hasła oraz aktywacji konta (opis pól przedstawia tabela 2.28).

```
{
  "username": {string},
  "token": {string},
  "type": "reset_password/activation",
  "_id": ObjectId("561105e11b83892511b484a5")
}
```

Tabela 2.28: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Tokens”

Nazwa pola	Opis pola
_id	Identyfikator wygenerowany automatycznie przez silnik bazy danych.
username	Pole zawierające nazwę użytkownika, dla którego przeznaczony jest token.
token	Pole zawiera token uwierzytelniający..
type	Informacja o rodzaju tokenu. W systemie wyróżniamy tokeny resetujące hasło użytkownika oraz aktywujące konto.

Scripts - kolekcja przeznaczona do gromadzenia danych dotyczących skryptów użytkowników (opis pól przedstawia tabela 2.29).

```
{
  "name": {string},
  "description": {string},
  "code": {string},
  "owner": {string},
  "codeRevisions":[
    {
      "dateTime": {timestamp},
      "codeLength": {number},
      "code": {string}
    }
  ],
  "_id":ObjectId("56367e65992f13b909814bdb")
}
```

Tabela 2.29: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Scripts”

Nazwa pola	Opis pola
_id	Identyfikator wygenerowany automatycznie przez silnik bazy danych.
name	Nazwa skryptu nadana przez użytkownika.
description	Pole zawierające opis skryptu.
code	Aktualny kod skryptu.
owner	Nazwa użytkownika będącego właścicielem (twórcą) skryptu.
codeRevisions	Lista, której elementami są poszczególne rewizje kodu skryptu (kod, długość skryptu, data edycji). Dzięki tak przetrzymywanych danych mamy możliwość obserwacji progresu skryptów poszczególnych graczy.

Lobbies - kolekcja przechowująca informacje o pokojach (lobby) dostępnych w systemie oraz rozgrywce (opis pól przedstawia tabela 2.30).

```
{
  "_id": ObjectId("561819340a5ac03012eac80a"),
  "countOfMembers": {number},
  "encrypted": {true/false},
  "password": {string}
  "master": {string},
  "name": {string},
  "players":[
    {
      "username": {string},
      "state": {waiting/play/finished},
      "publicIP": {IPv4}
    }
    ...
  ],
  "serverPort": {number},
  "state": {waiting/play/finished}
}
```

Tabela 2.30: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Lobbies”

Nazwa pola	Opis pola
_id	Identyfikator wygenerowany automatycznie przez silnik bazy danych.
countOfMembers	Pole określające pojemność lobby tj. ilość osób, które mogą do niego dołączyć.
encrypted	Informacja o tym, czy lobby jest chronione hasłem dostępowym.
password	Hasło dostępowe do lobby (w przypadku wartości “encrypted” równej “true”).
master	Nazwa użytkownika będącego właścicielem lobby (twórca).
name	Nazwa lobby.
players	Lista, której elementami są informacje dotyczące użytkowników, którzy dołączyli do lobby. Można tam znaleźć informacje takie jak nazwa użytkownika, jego status gotowości oraz adres IP, z którego dołączył się do lobby.
serverPort	Pole zawierające informację, na jakim porcie nasłuchuje serwer gry.
state	Status lobby. Zawiera informację czy lobby jest w stanie oczekiwania, gry czy zakończonej gry.

Scores - kolekcja zawierająca dane statystyczne tj. wyniki poszczególnych graczy (opis pól przedstawia tabela 2.31).

```
{
  "_id":ObjectId("5611095d8f359dc677d598c9"),
  "lastGameScore": {number},
  "lastGameTime": {number},
  "losses": {number},
  "summaryGameScore": {number},
  "summaryGameTime": {number},
  "username": {string},
  "victories": {number}
}
```

Tabela 2.31: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Scores”

Nazwa pola	Opis pola
_id	Identyfikator wygenerowany automatycznie przez silnik bazy danych.
lastGameScore	Pole zawierające informację o tym, jaki wynik uzyskał gracz w ostatniej grze.
lastGameTime	Pole zawierające informację o tym, ile czasu gracz spędził w ostatniej grze.
losts	Ilość przegranych gier.
summaryGameScore	Sumaryczny wynik punktowy.
summaryGameTime	Całkowity czas spędzony przez gracza.
username	Nazwa użytkownika, którego dotyczy statystyka.
victories	Ilość wygranych gier.

3. Implementacja

3.1. Wykorzystane technologie

3.1.1. REST API

Zacznijmy od definicji. Tak więc REST (ang. Representational State Transfer) jest wzorcem narzucającym dobre praktyki tworzenia rozproszonych aplikacji internetowych. REST API jest implementacją wzorca REST z wykorzystaniem protokołu HTTP.

Zasada działania REST API jest bardzo prosta. Serwer po otrzymaniu zapytania w oparciu o zasób wyszukuje odpowiedniej akcji do wykonania. Dokonuje zdefiniowanych przez programistę instrukcji np. zapis do bazy danych, obliczenie wartości etc. Na koniec w zależności od wyniku przeprowadzonych operacji zwraca odpowiedni kod odpowiedzi.

Do dyspozycji programistów oddanych zostało kilka typów żądań. Są to znane z protokołu http zapytania typu.:

- POST - zapis danych, zasobu etc.,
- GET - tylko pobranie danych,
- PUT - edycja istniejących już danych,
- DELETE - żądanie usunięcia zasobu, danych etc.

Twórca systemu REST API musi dostarczyć dokumentację dla klientów, którzy chcą z tego systemu korzystać. Muszą się znaleźć tam informacje określające pod jakim adresem można się spodziewać odpowiednich akcji do wykonania, jakie parametry należy ustawić i przesłać oraz w jaki sposób akcja pod podanym adresem będzie raportowała wynik (kod odpowiedzi).

W systemie REST API zachowane zostały odpowiedzi znane z HTTP - kod 20x będzie oznaczał sukces wykonania, 404 nieznalezienie zasobu a 401 będzie oznaczał brak autoryzacji do wykonania zadania. W każdym projekcie odpowiedzi mogą być poszerzane o kody specyficzne dla danego projektu - np. "Podany login już istnieje" i inne.

Każde żądanie do API wykonywane jest oddzielnie oraz bez pamiętania wyniku poprzedniej operacji (bezstanowo). Wszystko wykonywane jest asynchronicznie i niezależnie od siebie. W przypadku zdefiniowania akcji wymagającej długiego czasu obliczeń, nawet kilkukrotne jej wywołanie nie sprawi przyspieszenia któregośkolwiek wykonania.

Podsumowując, wybraliśmy REST API, gdyż świetnie sprawdza się w projektach, w których tworzy się wiele aplikacji w różnych językach i występuje konieczność wymiany danych. Niemal każdy język ma dostępną bibliotekę do obsługi protokołu HTTP a co za tym idzie jest w stanie korzystać z wystawionego wcześniej REST API. Pozwala to na odrobinę swobody w wyborze technologii, gdyż nie wymusza stosowania zamkniętych na jeden język technologii. Asynchroniczne podejście REST API jest według nas dodatkowym plusem w rozrachunku czy z niego korzystać czy nie. W naszym projekcie musimy wymieniać różne

dane (tylko tekstowe) zarówno między serwerem a grą, jak i serwerem a webserwisem. Zastosowanie REST API pozwoli nam w nieskomplikowany sposób pokryć to wymaganie.

3.1.2. Baza danych

Systemem bazy danych, na który się zdecydowaliśmy jest MongoDB. Jest to otwarty projekt napisany w całości w języku C++ [3]. Ze względu na zasadę działania zapewnia bardzo wysoką wydajność, dzięki czemu cieszy się popularnością w profesjonalnych aplikacjach [19].

NoSQL, bo tak nazywa się to nierelacyjne podejście, zorientowane jest na jak najszybsze działanie operacji odczytu oraz zapisu do bazy danych [19]. Największą zaletą tej technologii jest to, że nie narzuca ona sztywnej struktury, lecz dane można przechowywać w dowolnej postaci [3]. Obsługuje ona wiele typów danych jak np. JSON, pliki graficzne, pliki wideo, tekst oraz wiele innych.

Baza MongoDB w naszym projekcie umożliwia nam wygodne przechowywanie informacji, tzn. że dzięki jej wykorzystaniu nie potrzebujemy dokonywać nadmierowej konwersji danych do obiektów wykorzystywanych w językach programowania, co znacznie upraszcza logikę przetwarzania zapytań do API. Baza obsługiwana jest przez API webserwisu. Komponent ten został napisany w języku JavaScript, którego obiekty przedstawiane są w postaci JSON. MongoDB jest rozwiązaniem wręcz stworzonym dla przechowywania tego typu obiektów.

Pojęcie NoSQL jest rozwiązaniem dedykowanym głównie dla aplikacji internetowych operujących na wielkich ilościach danych [19]. W naszym projekcie nie zależy nam na wyrafinowanym modelu relacyjnym, lecz na dynamice oraz wygodzie użycia. Potrzebujemy zwyczajnych pojemników danych, do których będziemy mogli wkładać dane, a następnie je z nich wyciągać. MongoDB umożliwia tworzenie bardziej lub mniej skomplikowanych zapytań, obsługuje indeksowanie dzięki czemu działa bardzo wydajnie [3].

MongoDB nie ma żadnych problemów z obsługą unicode'a. Można w niej przechowywać wartości zawierające niestandardowe znaki, co ważne jest w przypadku operowania na danych dotyczących użytkowników.

Popularność MongoDB sprawiła, że w sieci dostępnych jest bardzo dużo informacji na temat tej technologii, poradników, typowych problemów. Korzystanie ze znanych powszechnie technologii zawsze ułatwia pracę programistom.

3.1.3. Unity

Unity jest elastyczną i potężną platformą deweloperską dla tworzenia multiplatformowych gier 3D i 2D oraz interaktywnych doświadczeń [21]. Jest to kompleksowy ekosystem dla każdego, kto zamierza budować biznes na tworzeniu treści wysokiej klasy i łączenia najbardziej lojalnych oraz entuzjastycznych graczy czy klientów [23].

Istnieje wiele różnych platform, na które można tworzyć aplikacje w silniku Unity, a ich liczba stale rośnie [21]. Tworzymy naszą aplikację w jednym konkretnym środowisku a Unity zapewnia nam możliwość przeniesienia naszej aplikacji na wszystkie główne technologie mobilne, desktopy czy konsole bez żadnych większych modyfikacji.

Unity wybraliśmy głównie ze względów na ilość komponentów [23], które oferuje nawet w swojej podstawowej wersji. Wykorzystując *Physically-based shading* oraz *Enlighten-powered Real-time Global Illumination* umożliwiło nam stworzenie schludnie wyglądającej aplikacji, która nie odstraszała użytkownika swoją surowością. Oczywiście jest to nasza subiektywna ocena, która zostanie zweryfikowana przez użytkowników podczas etapu ewaluacji. Jak dobrze wiemy, większość ludzi ocenia książkę po okładce, dlatego tworząc aplikację wysokiej jakości mamy większe szanse pozytywne zaskoczenie użytkownika.

Oczywiście sama grafika to nie wszystko, dzięki miksowaniu w czasie rzeczywistym oraz hierarchicznych mikserach i predefiniowanych efektach dostarczamy efekty audio na bardzo wysokim poziomie. Potwierdzają to opinie użytkowników, dostępne w rozdziale 4.3. Łatwość z jaką te komponenty są edytowalne i konfigurowalne, nie tylko przyspieszają cały proces deweloperski [24], ale również wpływają na ogólną ocenę końcową produktu.

Grafika i dźwięk to nie wszystko na co stać Unity. Aby nadać wszystkiemu realnego wydźwięku potrzebna jest fizyka [25]. Komponenty dostarczane z silnikiem takie jak Box2D, z kompleksową gamą efektorów, połączeń oraz *colliders*ów, jak i również cały kombajn fizyczny marki NVIDIA® PhysX® 3.3 daje tam bardzo duże pole do popisu w kwestii implementacji złożonych efektów środowiskowych oraz oddziaływań między obiektami.

Nie można również pominąć faktu, iż Unity jest bardzo dobrze zoptymalizowane [21]. Posiada zaawansowane narzędzia analizowania zużycia pamięci oraz *occlusion culling*, czyli usuwanie powierzchni niewidocznych, oparte na technologii Umbra, która jest wykorzystywana przez wiele światowych firm. Można powiedzieć, że pracujemy na tej samej technologii co ogólnoswiatowe korporacje tworzące gry komputerowe [22].

Najbardziej przemawiającą dla nas zaletą wykorzystania Unity jest to, iż cały kod oraz skrypty piszemy w języku C#, czyli w bardzo popularnym i szeroko znanym języku, z którym każdy z nas miał do czynienia. Doświadczenie wyniesione z pisania programów w języku C++ oraz Java w komfortowy sposób pozwoliło nam bardzo szybko i sprawnie tworzyć elegancki, spójny i wydajny kod. Zachowanie jednolitej struktury na obszarze całego kodu umożliwia przyszłym wolontariuszom oraz nam samym na kontynuowanie pracy, dalsze usprawnienia i rozwijanie kolejnych funkcjonalności.

3.1.4. NodeJS

Język JavaScript jest bardzo popularnym językiem wśród webdeweloperów. Ma bardzo wielu zwolenników jak i wrogów. Ze względu na swój charakter przez niektórych jest wychwalany, a inni go nienawidzą. My uważamy, że język ten jest o wiele lepszym wyborem aniżeli stosowanie PHP, którego popularność z każdym dniem maleje. Przeciwnie ma się sprawa z JavaScript.

W naszym projekcie musieliśmy dokonać wyboru i spośród wielu technologii należało zdecydować, jakiej technologii użyjemy do stworzenia serwera naszego webserwisu. Jak już wytłumaczyliśmy w punkcie 2.6.1, chcieliśmy użyć REST API. Ostatnio bardzo dużo mówi się na temat NodeJS. Jego popularność jest coraz większa. Fakt, że technologia ta oparta jest

całkowicie o JavaScript [1], a także pochlebne opinie programistów na jej temat sprawiły, że zdecydowaliśmy się na użycie NodeJS w naszym projekcie. Jest to projekt otwarty, całkowicie darmowy [1]. Na uwagę zasługuje fakt, że pomimo jego młodego wieku posiada dobrze rozbudowaną dokumentację, co świadczy o tym, że technologia ta jest rozwijana i chętnie używana.

NodeJS jest platformą, która umożliwia uruchomienie kodu JavaScript poza przeglądarką. Dzięki temu w bardzo łatwy sposób możemy obsługiwać np. bazę danych czy system plików [1]. Korzystając z menadżera pakietów (npm) w łatwy sposób możemy doinstalować do naszego projektu dostępne za darmo w sieci komponenty.

NodeJS nie narzuca sztywnej struktury kodu. Umożliwia zorganizowanie projektu w dowolny sposób. W naszym webserwisie wyróżniliśmy osobno część obsługującą żądania na poziomie http, część zawierającą logikę biznesową, osobno elementy obsługi bazy danych czy też walidacji danych. Dzięki takiemu podziałowi kod programu jest czytelniejszy i łatwiejszy do zrozumienia.

Do niedawna popularnym rozwiązaniem wśród aplikacji webowych był język PHP, który królował na rynku przez kilka dobrych lat. Przy jego użyciu powstało bardzo wiele projektów. Twórcy NodeJS tworząc swoją technologię chcieli wyciągnąć wnioski z tego, czego się już nauczyli wcześniej podczas używania PHP. Ich język miał rozwiązać wszystkie te problemy, z którymi borykał się PHP. NodeJS miał stanowić narzędzie umożliwiające tworzenie nowoczesnych aplikacji internetowych, dzięki czemu można go uznać za produkt przemysłany i godny polecenia [20].

NodeJS jest technologią asynchroniczną. Dzięki temu jest bardzo wydajny. Działa bezstanowo, co umożliwia bardzo szybkie przetwarzanie żądań [1]. W naszym projekcie nie przesyłamy zapytań o dużych rozmiarach lecz niewielkie obiekty typu json. NodeJS jest technologią dedykowaną dla serwerów obsługujących bardzo dużo niewielkich rozmiarów zapytań w tym samym czasie. Ten argument również przemawiał za użyciem tej technologii.

3.1.5. AngularJS

AngularJS jest frameworkiem JavaScript pozwalającym na szybkie tworzenie nawet skomplikowanych stron internetowych. Jest zarządzany i rozwijany przez firmę Google. Jest kompromisem pomiędzy pomysłami stosowanymi w JavaScriptcie a modelem MVC (Model - View - Controller). Owym kompromisowym rozwiązaniem jest tzw. model MVW (Model – View – Whatever). Popularność frameworka ciągle rośnie co sprawia, że jego dokumentacja jak i społeczność z każdym dniem się poszerzają.

Wybraliśmy Angulara ze względu na mechanizmy dostępne w tej technologii. Są to między innymi:

- dwukierunkowy binding danych - pozwala na automatyczną synchronizację danych, które mamy po stronie widoku, a naszym kontrolerem,
- obiekt \$scope - jest to obiekt w którym przechowane są dane tworzone przez kontroler. W łatwy sposób możemy się porozumiewać między kontrolerami,

- dyrektywy - możliwość tworzenia nowych tagów html, dodawania atrybutów do istniejących tagów, budowania bloków CSS itp.

3.1.6. Obsługa sieciowa w Unity

Unity w wersji 5.1 wprowadziło nowy, ulepszony system Sieciowy z dużo bardziej elastycznym i potężniejszym systemem niż ten dostarczany w poprzednich wersjach Unity. Posiada nisko poziomowy dostęp przy pomocy klasy "NetworkTransport", która jest cienką warstwą nad podstawowymi gniazdkami (ang. sockets) oraz posiada inne wysoko poziomowe komponenty, które dodają więcej użytecznych funkcjonalności wieloosobowych.

Pisząc naszą aplikację skupiliśmy się na wykorzystaniu wysoko poziomowego API skryptowego (High-level scripting API -> HLAPI). Wykorzystywanie go daje nam dostęp do rozkazów, które pokrywają najpopularniejsze wymagania gier wieloosobowych bez potrzeby martwienia się o implementację niskopoziomową.

HLAPI pozwala nam:

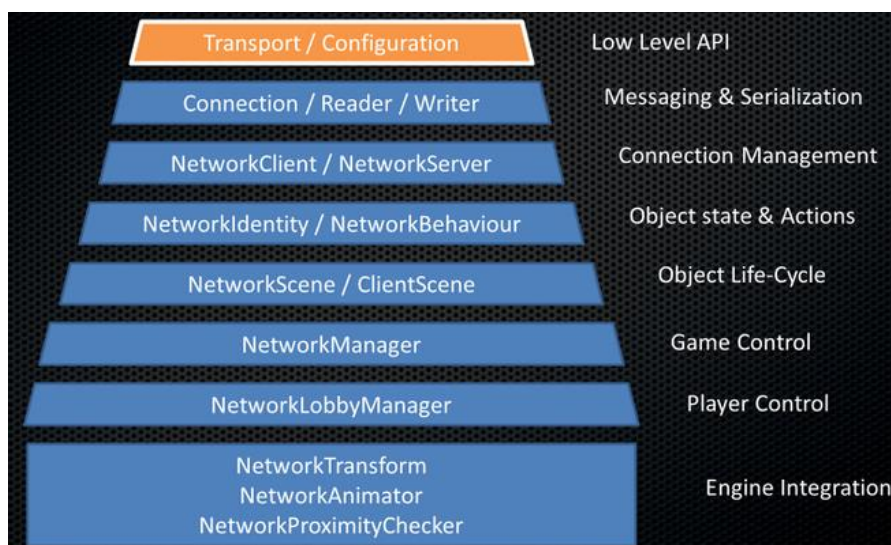
- kontrolować status gry przy pomocy "Network Manager",
- tworzyć samo-hostujących klientów, czyli klient sam sobie hostuje rozgrywkę,
- serializować dane za pomocą serializera ogólnego przeznaczenia,
- wysyłać i odbierać wiadomości sieciowe,
- wysyłać sieciowe rozkazy (commands) przez klientów do serwera,
- wywoływać zdalnie procedury (RPC) z serwera na klientach,
- wysyłać sieciowe zdarzenia (events) z serwera do klientów.

Obsługa sieciowa w Unity jest zintegrowana z edytorem. Pozwala nam to pracować z komponentami i wizualnymi pomocami podczas tworzenia sieciowej gry.

W skład owych komponentów wchodzi:

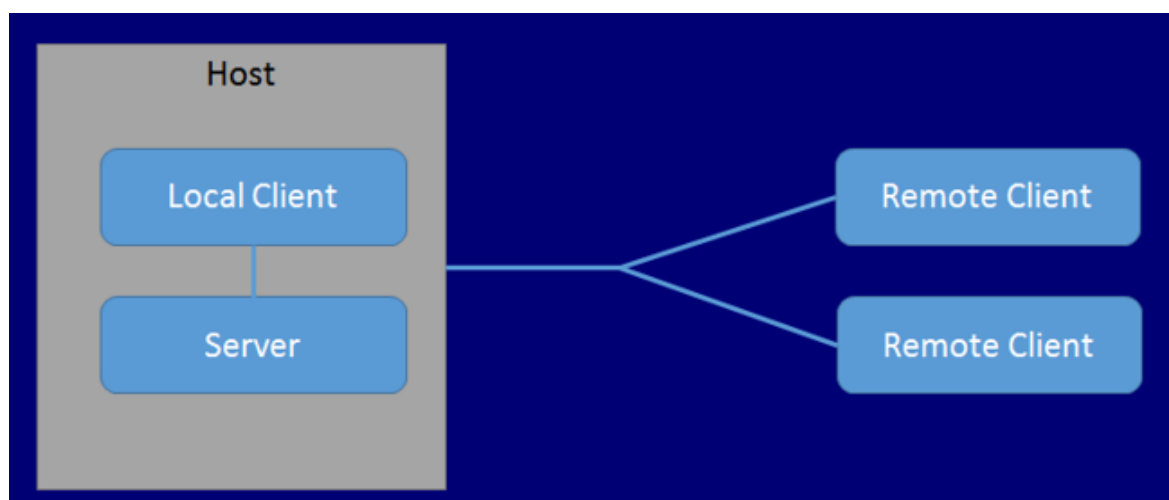
- "NetworkIdentity" komponent odpowiedzialny za identyfikację obiektów w sieci,
- "NetworkBehaviour" podstawowa klasa zachowań sieciowych obiektów; nadpisując ją tworzymy nasze skrypty,
- konfigurowalna, automatyczna synchronizacja transformacji obiektów,
- automatyczna synchronizacja zmiennych skryptowych,
- wsparcie dla umieszczania obiektów sieciowych w scenach Unity.

High Level API (HLAPI) jest systemem zbudowanym na nisko poziomowej warstwie transportowej, komunikującej się w czasie rzeczywistym z innymi warstwami (rys. 3.1), zajmuje się obsługą wielu popularnych zadań wymaganych przez gry sieciowe. Podczas kiedy warstwa transportowa wspiera każdy rodzaj topologii sieciowej, to HLAPI jest serwerem z autorytatywnym systemem; pozwala to na to aby jeden z graczy był zarówno serwerem jak i klientem, więc serwery dedykowane są niepotrzebne w całym procesie. Pracując w tak spójnym systemie, tworzenie gier sieciowych wymaga od deweloperów bardzo mało pracy.



Rys. 3.1. Schemat warstw na których zbudowana jest obsługa sieciowa w Unity.

W sieciowym systemie unity, gry muszą posiadać Serwer oraz wielu Klientów. Gdy nie ma żadnego serwera dedykowanego, jeden z graczy przyjmuje rolę serwera - wtedy nazywamy takiego klienta hostem (rys. 3.2).

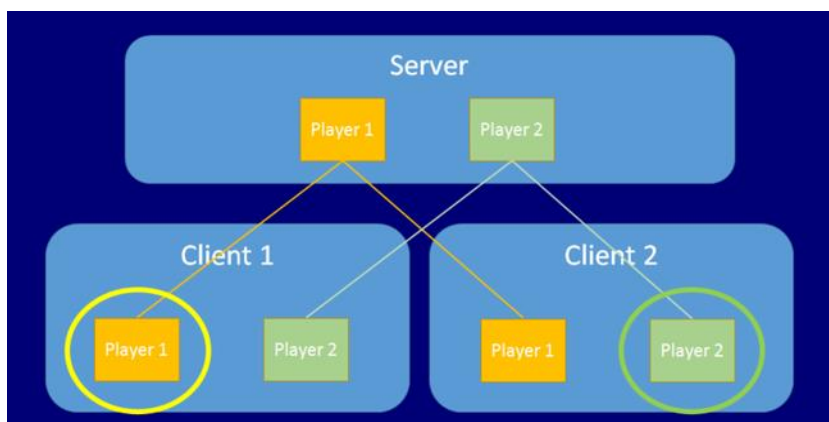


Rys. 3.2. Schemat hosta jako klienta i serwer oraz podłączonych do niego graczy.

Host jest serwerem i klientem w obrębie tego samego procesu. Host używa specjalnego rodzaju klienta zwanego "LocalClient" - klient lokalny podczas gdy pozostali klienci zwani są "RemoteClients". Klient lokalny komunikuje się z serwerem bezpośrednio za pomocą wywołań funkcji czy też kolejki wiadomości, z racji faktu że istnieją w tym samym procesie. Klienci zdalni komunikują się z serwerem za pomocą regularnych połączeń sieciowych. Głównym celem całego systemu jest to aby kod Klienta lokalnego i Klienta zdalnego był taki sam, gdzie deweloper musi myśleć tylko nad jednym rodzajem klienta przez większość czasu.

W systemie, obiekty gracza są specjalne. Każdy obiekt gracza jest połączony z konkretną osobą grającą w grę i do tego obiektu są przekierowywane wszystkie rozkazy przychodzące z serwera. Gracz nie może wywoływać rozkazów na obiektach innych graczy -

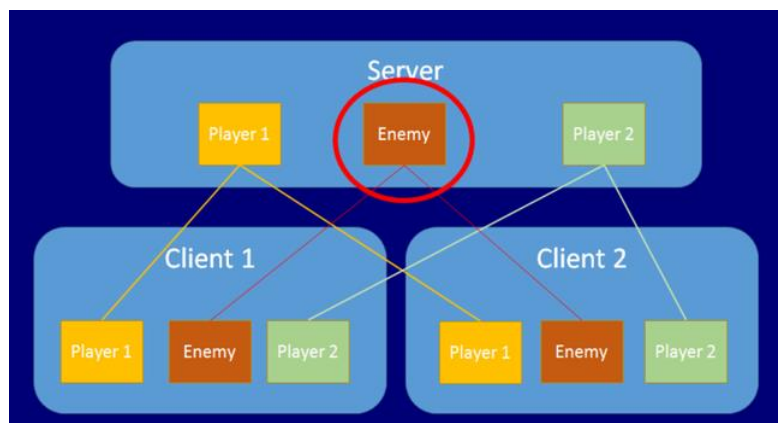
tylko na swoim, więc taki koncept nazywamy obiektem “mojego” gracza. Kiedy nowy gracz jest dodawany i tworzone jest połączenie, dany obiekt gracza zyskuje status “local player” - oznacza to tyle, że dany konkretny obiekt jest reprezentacją aktualnie grającego gracza. Rysunek 3.3 przedstawia dwóch klientów oraz ich “lokalnych graczy”.



Rys. 3.3. Schemat graczy i ich instancji.

Dodatkowo każdy z obiektów danego gracza może posiadać lokalną autorytatywność, oznacza to mniej więcej tyle, że dany klient jest odpowiedzialny za konkretny obiekt. Jest to bardzo często używane podczas kontrolowania ruchu, ale może być też wykorzystywane do innych rzeczy.

Dla obiektów nie-graczy, takich jak przeciwnicy, którzy nie mają bezpośredniego połączenia z konkretnym klientem - ich autorytatywność jest utrzymywana na serwerze (rys. 3.4).

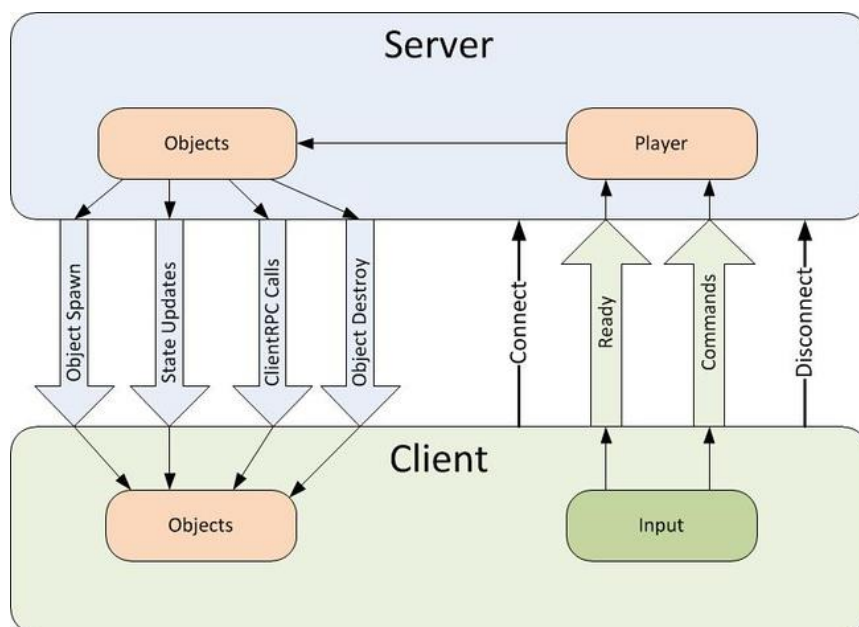


Rys. 3.4.: Autorytatywność zachowana na serwerze.

Wykorzystując właśnie te mechanizmy w Unity, stworzyliśmy naszą grę. Staraliśmy się trzymać standardu i wizji sieciowej dostarczanej wraz z Unity. Wiedząc jak konkretne moduły ze sobą funkcjonują, jak się komunikują i gdzie mają cechy wspólne, wstrzykiwaliśmy nasze zależności obsługowe. Ponieważ chcieliśmy aby gra była pozbawiona jakiegokolwiek zbędnego Menu/GUI musieliśmy podpiąć się pod konkretne zdarzenia obsługowe, tak aby zminimalizować narzut interakcji użytkownika z GUI.

Ponieważ nasza aplikacja miała w założeniu opierać się na pisaniu skryptów przez graczy, a nie wykorzystywaniu zachowań dostarczanych przez deweloperów, musieliśmy zrozumieć jak w Unity wywołać funkcję z poziomu rozgrywki na serwerze. Gdyby gracz używał tylko funkcji napisanych przez nas to samo przekazanie ich na serwer nie byłoby specjalnie trudne, gdyż opierałoby się po prostu na poinformowaniu serwera o tym co teraz chcemy wykonać. W przypadku ręcznie pisanych skryptów, najpierw musieliśmy wywołać kod po stronie lokalnego klienta a potem też na serwerze, tak aby rozgłosił on to na wszystkich pozostałych klientach. I tu właśnie wykorzystaliśmy mechanizm Zdalnego Wywoływania Procedur (RPC).

Jest wiele możliwości przesyłania akcji przez sieć. Jedną z tych dróg jest mechanizm zdalnych wywołań procedur. Istnieją dwa rodzaje wywołań: "Commands" - wysyłane są na serwer przez klientów oraz "ClientRpc" - wysyłane przez serwer do klientów. Przepływ wymiany komunikatów prezentuje rysunek 3.5.



Rys. 3.5. Schemat komunikacji klienta z serwerem i odwrotnie.

W ten sposób, klient reagując na nasz "Input" wysyła informację o tym na serwer, który rozpoznaje nadawcę wiadomości oraz obiekt na którym wykonać rozkaz. Następnie w obrębie swojego autorytatywu wykonuje odpowiednie procedury i rozsyła wynik do wszystkich pozostałych graczy.

Z drugiej strony, są pewne zdarzenia które zachodzą automatycznie na serwerze, bez wpływu żadnego z graczy. Wtedy serwer musi poinformować pozostałych graczy wywołując na ich instancjach odpowiednie funkcje, do tego wykorzystuje się właśnie "ClientRpc". I tak przykładowo, gracz wysyła swoją jednostkę zbierającą do zbierania zasobów ze złoża rudy, swoje rozkazy przesyła na serwer przy pomocy "Commands". Jednakże to serwer zarządza złożem rudy i gdy ono się skończy to musi poinformować resztę graczy, że jest już ono puste, więc wyśle "ClientRpc" do wszystkich, że to konkretne złożo jest puste i nie można go więcej używać.

3.1.7. Połączenie webserwisu i Gry na poziomie protokołu

Webserwis jest wykorzystywany jako miejsce, w którym gracze mogą dobierać sobie przeciwników. W przyszłości może on wyewoluować do miejsca, gdzie gracze będą sobie wzajemnie pomagać z pisaniem coraz bardziej zaawansowanych skryptów oraz udzielać początkującym wskazówek taktycznych, przybliżających ich do zwycięstw. W stanie obecnym webserwis służy tylko do wspomnianego wcześniej doboru przeciwników oraz jako baza dla skryptów przygotowanych wcześniej lub w poprzednich grach przez użytkownika.

Komunikacja gry z webserwisem odbywa się za pomocą protokołu HTTP. Gra wykorzystuje mechanizmy GET i POST do uzyskiwania odpowiednich zasobów. Dane przesyłane metodą POST są enkapsulowane do postaci obiektów w formacie JSON. Użytkownik uruchamia grę z poziomu webserwisu, który następnie przekazuje nazwę użytkownika, token uwierzytelniający, adres IP i port użytkownika serwującego grę. Dane te są przekazywane systemowi użytkownika w postaci URI z identyfikatorem protokołu "sharpwars://". Gra podczas uruchomienia przyjmuje te dane jako argumenty przy uruchamianiu, parsuje je, a następnie automatycznie wypełnia pole adresowe serwera. Nazwa użytkownika i token są wykorzystywane do uwierzytelniania w webserwisie, które jest wymagane do wysyłania wyniku po zakończeniu gry oraz pobierania zdefiniowanych wcześniej przez użytkowników skryptów. Zarówno token jak i nazwa użytkownika są wymagane do edycji i dodawania nowych skryptów przez gracza, które są automatycznie zapisywane na koncie rzeczonoego użytkownika.

Komunikacja gry z webserwisem odbywa się poprzez wbudowaną w silnik Unity klasę WWW.

3.1.8. Przegląd istniejących rozwiązań użytych w projekcie

NodeJS udostępnia menadżera pakietów. Przy jego pomocy mamy możliwość doinstalowywania różnorodnych pakietów dostępnych za darmo w sieci. Wykorzystanie gotowych komponentów znacznie usprawnia pracę, a dodatkowo dzięki otwartości kodu tych komponentów mamy możliwość znalezienia licznych opinii programistów na ich temat. Dzięki temu możemy szybko dowiedzieć się jak rozwiązać związane z nimi typowe problemy oraz poznać ich wady jeszcze zanim zdecydujemy się na ich użycie w naszym projekcie.

Podczas implementacji kodu naszego webserwisu wykorzystaliśmy kilka użytecznych bibliotek znalezionych w sieci na otwartej licencji. Przedstawiliśmy je poniżej w tabeli.

Tabela 3.1: Opis wykorzystanych komponentów NodeJS

Nazwa	Opis
ExpressJS	Jest to framework, który umożliwił nam wygenerowanie najprostszego w swojej postaci serwera NodeJS. Jest to jego minimalna postać, którą mogliśmy rozbudowywać o kolejne moduły. ExpressJS pomógł nam w znacznym stopniu w zorganizowaniu przepływu sterowania w webserwisie.
Mongoskin	Moduł umożliwiający komunikację z bazą danych MongoDB. Jego interfejs składa się z wielu metod obsługujących takie operacje jak wstawianie, odczytywanie, wyszukiwanie czy usuwanie danych z bazy. Dodatkowo każdą metodę można sparametryzować wg własnych

Nazwa	Opis
	potrzeb np. w celu otrzymania szukanego wyniku w postaci listy.
JsonSchema	Każde żądanie trafiające do API webserwisu powinno być sprawdzane pod względem poprawności przesyłanych w nim danych. Sama walidacja po stronie klienta nie jest wystarczająca. Z tego to powodu wykorzystaliśmy komponent, który umożliwia sprawdzanie ciała zapytań na podstawie schematów zapytań, które zawierają takie informacje jak np. liczba kluczy w body, klucze obowiązkowe, maksymalna ilość elementów w tablicy, sprawdzanie poprawności wartości tekstowych przy użyciu wyrażeń regularnych.
Password-hash	Jest to funkcja mieszająca, której użyliśmy do zamiany haseł podawanych przez użytkownika do postaci zaszyfrowanej. Ma to na celu zwiększenie bezpieczeństwa użytkowników zarejestrowanych w systemie.
Nodemailer	Komponent umożliwiający wysyłanie wiadomości email z webserwisu.
Log4JS	Narzędzie służące do logowania komunikatów pochodzących z metod logiki biznesowej webserwisu. Umożliwia również gromadzenie logów w pliku tekstowym.
JsonWebToken	Komponent umożliwiający organizację sesji w przeglądarce. Zasada działania oparta jest o token uwierzytelniający danego użytkownika w systemie, co czyni go rozróżnialnym spośród innych.

3.2. Przygotowanie środowiska deweloperskiego

3.2.1. Przygotowanie środowiska deweloperskiego dla webserwisu

Projekt webserwisu został oparty o otwarte technologie, dlatego też został zaimplementowany i wdrożony na otwartym systemie operacyjnym GNU/Linux. Przykładowa konfiguracja oparta jest o dystrybucję *Ubuntu* - wydanie 14.04 LTS.

1. Do pracy ze źródłami webserwisu potrzebujemy następujących narzędzi: *git*, *gitk*, *git-gui*, *mongodb*, *curl*, *nodejs* i *npm*. Instalujemy je komendą:

```
$ sudo apt-get install curl
$ curl -sL https://deb.nodesource.com/setup_0.12 | sudo bash -
$ sudo apt-get install git gitk git-gui mongodb nodejs npm
```

2. Teraz musimy stworzyć dowiązanie symboliczne (jeśli takowe jeszcze nie istnieje) polecenia *nodejs* jako *node*. Robimy to w następujący sposób:

```
$ sudo ln -s /usr/bin/nodejs /usr/bin/node
```

3. Jesteśmy już gotowi do działania na źródłach. Kod webserwisu przechowywany jest w repozytorium Githuba. Aby go pobrać na dysk wydajemy proste polecenie klonowania repozytorium:

```
$ git clone https://github.com/urbanek32/SharpWars_WebService.git
```


4. Po poprawnym sklonowaniu repozytorium na dysk ściągamy komponenty potrzebne do uruchomienia naszego projektu. Robimy to przechodząc do katalogu ze źródłami webserwisu, a następnie wykonujemy polecenie:

```
$ npm install
```

5. Musimy jeszcze doinstalować programy *grunt* i *bower*. Robimy to następująco:

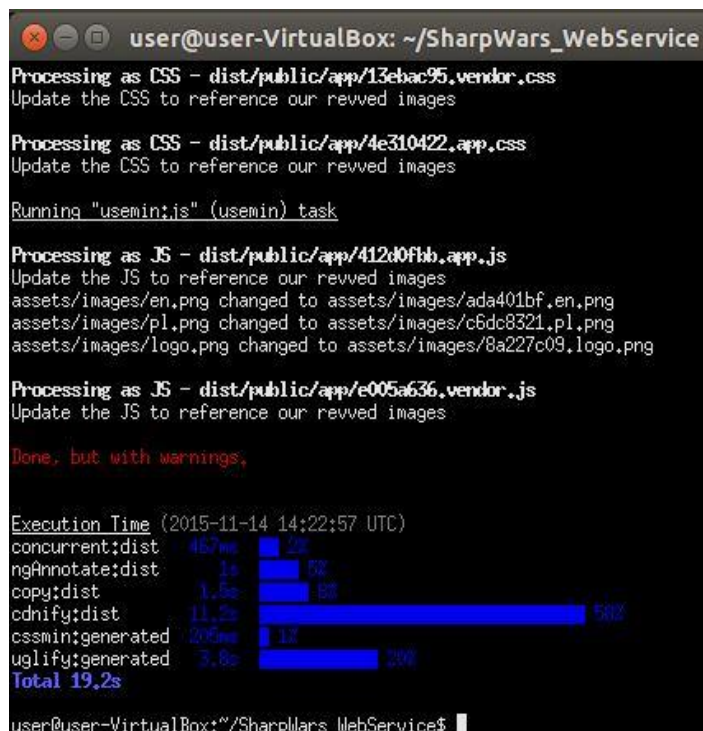
```
$ sudo npm -g install grunt-cli bower  
$ bower install  
$ npm install grunt
```

6. Teraz środowisko pracy jest już gotowe do zbudowania projektu. Robimy to następującym poleceniem:

```
$ grunt
```

7. W przypadku przerwania budowania projektu z powodu wystąpienia ostrzeżeń użyj poniższego polecenia. Powinniśmy uzyskać wynik jak na poniższym rysunku (rys. 3.6).

```
$ grunt --force
```



```
user@user-VirtualBox: ~/SharpWars_WebService  
Processing as CSS - dist/public/app/13ebac95.vendor.css  
Update the CSS to reference our revved images  
  
Processing as CSS - dist/public/app/4e310422.app.css  
Update the CSS to reference our revved images  
  
Running "usemin:js" (usemin) task  
  
Processing as JS - dist/public/app/412d0fbb.app.js  
Update the JS to reference our revved images  
assets/images/en.png changed to assets/images/ada401bf.en.png  
assets/images/pl.png changed to assets/images/c6dc8321.pl.png  
assets/images/logo.png changed to assets/images/8a227c09.logo.png  
  
Processing as JS - dist/public/app/e005a636.vendor.js  
Update the JS to reference our revved images  
  
Done, but with warnings.  
  
Execution Time (2015-11-14 14:22:57 UTC)  
concurrent:dist 465ms 52  
ngAnnotate:dist 1s 52  
copy:dist 1.6s 85  
cdnify:dist 11.2s 562  
cssmin:generated 205ms 12  
uglify:generated 3.8s 206  
Total 19.2s  
user@user-VirtualBox:~/SharpWars_WebService$
```

Rys. 3.6. Wynik użycia polecenia z opcją `--force`

8. Aby obejrzeć wyniki w swojej przeglądarce użyj polecenia:

```
$ grunt serve
```

9. Aby obejrzeć skompilowaną wersję projektu użyj polecenia:

```
$ grunt serve:dist
```

10. Jeśli serwer i klient bazy danych MongoDB nie został zainstalowany do tej pory to wykonaj poniższe polecenie:

```
$ sudo apt-get install mongodb-server mongodb-clients
```

11. Serwer MongoDB uruchamia się poleceniem:

```
$ mongod
```

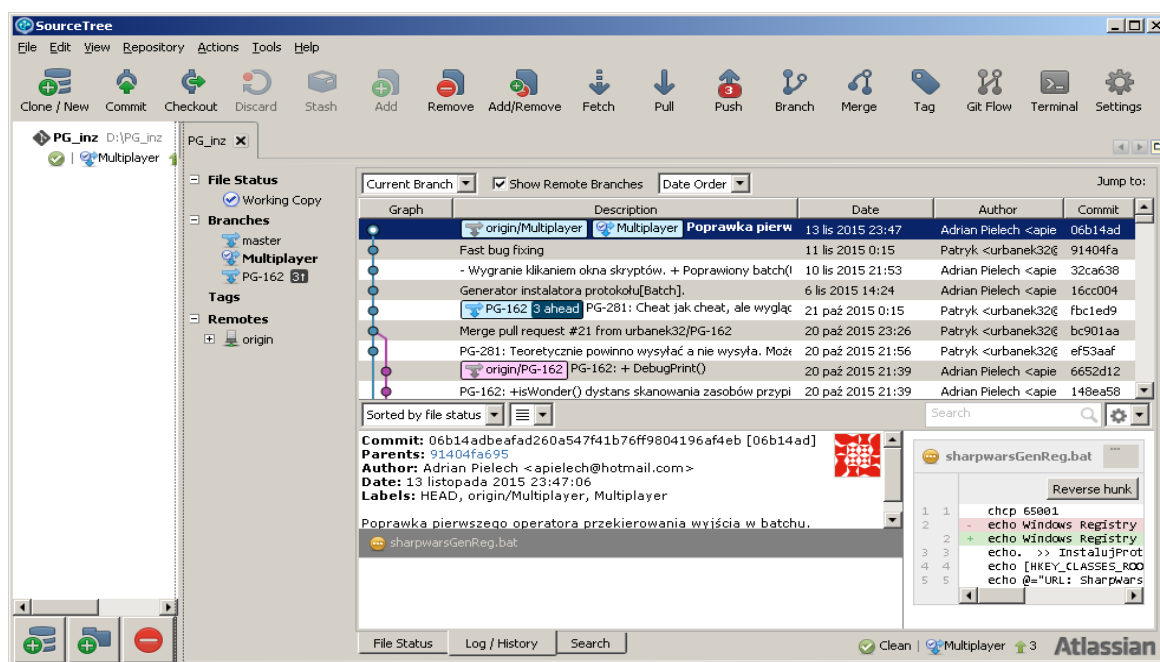
12. Jeśli serwer uruchomił się bez błędów to możesz sprawdzić połączenie z bazą za pomocą klienta. Robi to się poleceniem:

```
$ mongo
```

Jeśli nie wystąpiły żadne błędy to znaczy że wszystko działa i można przystąpić do rozwijania webserwisu. Do pisania kodu i zarządzania projektem używamy środowiska WebStorm wyprodukowanego przez firmę JetBrains.

3.2.2. Przygotowanie środowiska deweloperskiego dla Gry

Do obsługi repozytorium gita używamy aplikacji Atlassian SourceTree (rys. 3.7).



Rys. 3.7. SourceTree - narzędzie do zarządzania repozytorium gita

Gra jest pisana w Unity 5.1.3f1 z użyciem pluginu Unity for Visual Studio 2013. Kod gry edytujemy w środowisku Visual Studio 2013 Community. Do działania języka skryptowego wymagane jest posiadanie Visual C++ Redistributable for Visual Studio 2012.

3.3. Element programowania w grze

SharpWars jest grą, która opiera się na programowaniu jednostek wytworzonych podczas rozgrywki. Dopasowanie stworzonych przez użytkownika algorytmów decyduje w głównej mierze kto zostanie zwycięzcą. Dlatego też powstało w pełni funkcjonalne okienko edycji skryptów jednostki, pozwalające w czasie rzeczywistym edytować skrypty wraz możliwością zapisania ich do spersonalizowanej bazy skryptów w webserwisie.

W prototypie gry został zaimplementowany mechanizm sekwencyjnego przetwarzania instrukcji umożliwiający wykonywanie instrukcja po instrukcji programu napisanego przez użytkownika.

Język programowania dostępny dla graczy został oparty o istniejący język Lua [25]. W grze została wykorzystana jego adaptacja poprzez plugin o nazwie NLua, który umożliwia korzystanie ze środowiska Lua na platformie .NET, w tym również w Unity.

Gra posiada zaimplementowane funkcje wyszukiwania jednostek, zasobów, przemieszczania, określenia typu jednostek, a także zbierania. Użytkownik posiada również możliwość przeprowadzania prostych działań matematycznych takich jak dodawanie, odejmowanie, mnożenie, dzielenie (w tym modulo), potęgowanie i pierwiastkowanie. Język dostępny dla użytkownika oferuje także instrukcje warunkowe *if elseif else* oraz pętle warunkowe *while* do. Wymienione powyżej elementy języka powinny być skromnym, aczkolwiek

wystarczającym zbiorem możliwości do nauki języków programowania dla początkujących użytkowników komputerów chcących zaznajomić się z tematyką programowania.

3.4. Wytworzone artefakty

W trakcie pisania niniejszej pracy inżynierskiej udało nam się wytworzyć działający webserwis oraz grę napisaną w Unity 5. Gra do poprawnego działania wymaga posiadania utworzonego konta w webserwisie.

Webserwis został zaprojektowany z myślą o gromadzeniu wokół siebie społeczności, która będzie mogła rywalizować ze sobą. Zawiera on w sobie samouczek dotyczący podstaw programowania w grze, jak i posiada udokumentowane funkcje do programowania jednostek wraz z wyjaśnieniami użycia. Obecnie najważniejszym elementem webserwisu jest możliwość przygotowania skryptów, które będą użyte w trakcie gry. Ponadto, aby zagrać potrzebujemy założyć pokój gry, do którego dołączą zainteresowani gracze lub dołączyć do istniejącego już pokoju gry.

W prototypie gry grać może ze sobą równolegle do czterech graczy. Logika gry nie ogranicza z góry ich ilości, jednakże na potrzeby prototypu są zdefiniowane tylko cztery punkty startowe dla graczy co dla zachowania komfortu grania pozwala na stworzenie rozgrywki dla maksymalnie czterech osób.

Podczas trwania rozgrywki użytkownik ma możliwość tworzenia budynków, mobilnych jednostek użytkowych oraz jednostek ofensywnych. Zachowanie jednostek użytkowych i ofensywnych może być oprogramowywane przez gracza. Język programowania jednostek został dość dokładnie opisany w samouczku dla graczy. Udostępnione API jednostek pozwala na wyszukiwanie zasobów i ich zbieranie oraz na wyszukiwanie wrogich jednostek i ich atakowanie. Zasięg znajdowania zasobów i wrogów jest różny dla każdego typu jednostek.

Skrypty, które zostaną napisane przez gracza w trakcie rozgrywki mogą być modyfikowane i zapisywane na konto użytkownika w webserwisie. Do tworzenia, edytowania i usuwania powstało okno edycji skryptów jednostki. Po napisaniu skryptu gracz może oprogramować daną jednostkę i uruchomić wykonanie kodu lub przerwać wykonanie poprzedniego, jeśli jeszcze wykonanie kodu nie zakończyło się.

Zwycięstwo gracze mogą osiągnąć poprzez akumulację odpowiedniej ilości zasobów lub wybudowanie specjalnego budynku nazywanego "Cudem".

Po ukończeniu gry wynik rozgrywki jest automatycznie wysyłany do webserwisu, gdzie zostaje zapamiętany w bazie danych. Będąc zalogowanym w systemie, użytkownik ma dostęp do swoich statystyk oraz może porównać się swoimi wynikami z innymi graczami. Porównanie może odbywać się według następujących kryteriów:

- ilość wygranych rozgrywek.
- ilość przegranych rozgrywek.
- ilość uzyskanych punktów w trakcie rozgrywek.
- całkowitej ilości czasu spędzonego w grze.

3.5. Napotkane problemy

Etap implementacji dla programisty zazwyczaj jest weryfikacją jego wcześniejszych założeń. Jeśli założenia od początku są nieprzemyślane, projekt w późniejszej fazie może zakończyć się brakiem sukcesu. Istnieją jednak też mniej destruktywne problemy, które spowodowane są zbyt małym doświadczeniem w używaniu wybranej do realizacji projektu technologii. Nie doprowadzają one do całkowitego fiasko, ale zastosowane rozwiązanie nie jest w pełni idealne. Kilka takich problemów doświadczyliśmy również i my:

- pobieranie adresu IP klienta webserwisu - dopóki nie rozpoczęliśmy implementacji funkcji dołączania gracza do lobby byliśmy przekonani, że adres IP gracza będziemy mogli uzyskać analizując nagłówki żądania http po stronie serwera. Niestety, takiej informacji tam nie znaleźliśmy. Spowodowało to zmianę naszego podejścia. W trakcie implementacji zdecydowaliśmy, że adres ten będziemy uzyskiwali już po stronie klienta, a następnie wyślemy go w żądaniu do serwera. O ile nie ma w tym nic złego, o tyle sam sposób uzyskiwania adresu IP gracza jest nieco niewygodny. W celu jego uzyskania musimy łączyć się do zewnętrznego serwisu (<http://ipinfo.io>), który w odpowiedzi odsyła nam publiczny adres IP klienta. Realizując więc przypadek użycia "Dołączanie do lobby" jesteśmy nieco uzależnieni od dostępności tego serwisu, a jego awaria oznacza dla nas brak możliwości uzyskania adresu IP gracza, a ten w tym przypadku jest dla nas niezbędną informacją. Podobnym problemem jest pobranie lokalnego adresu IP. O ile w przypadku publicznego adresu otrzymujemy jasno jeden adres IP, o tyle w przypadku adresów wewnętrznych problem jest bardziej skomplikowany, ponieważ użytkownik może posiadać kilka kart sieciowych o różnych adresach. Nie możemy określić, który interfejs powinien zostać użyty. Z tego to powodu na chwilę obecną w aplikacji gry bezpośrednio przed rozpoczęciem rozgrywki mamy możliwość edycji adresu IP serwera, do którego się łączymy jako klienci;
- informowanie gracza o zmianie stanu lobby, w którym się znajduje. Był to pierwszy moment, w którym postawiliśmy w wątpliwość korzystanie z systemu REST API. Pojawiła się konieczność zmiany statusu lobby wyświetlanego graczowi w momencie, gdy inny gracz oznaczy się jako "gotowy do gry", bądź moderator pokoju startuje grę. Próby rozwiązania tego problemu rozpoczęliśmy od weryfikacji naszego podejścia do REST API i zastanowienia się czy nasz plan komunikacji z serwerem REST jest dobrze przemyślany. Postanowiliśmy zaimplementować mechanizm odpytywania serwera co kilka sekund w celu weryfikacji czy status pokoju się zmienił. Niestety słabe jeszcze obycie w technologiach, z którymi pracujemy doprowadziło do sytuacji, w której serwer był odpytywany przez całą wizytę użytkownika na stronie, a nie tylko wtedy gdy wyświetlane były informacje o lobby. Problem był dość trudny do wychwycenia gdyż nie powodował spowolnienia pracy webserwisu, a implementowana funkcja działała poprawnie. "Nieskończone odpytywanie" zostało zmergowane do reszty kodu i niezauważone.

Dopiero jakiś czas później, przy pracy nad inną funkcjonalnością i dokładnemu przyjrzeniu się wysyłanym zapytaniom zorientowaliśmy się, że generujemy niepotrzebny narzut na ruch sieciowy między serwerem, a klientem. Dopiero wtedy problem został rozwiązany;

- sekwencyjne przetwarzanie instrukcji w kodzie oprogramowywanych jednostek nie podlegające natychmiastowemu wykonaniu, które w przypadku pętli blokowało cały program. Rozwiązaniem tego problemu okazało się dynamiczne podzielenie kodu na bloki podprogramów, gdzie końcem takiego bloku było wystąpienie nowej instrukcji warunkowej lub jej końca. W przypadku funkcji, która musi zablokować dalsze wykonywanie skryptu, aż do momentu kiedy zostanie wykonana, zostało to rozwiązane poprzez stworzenie 2 asynchronicznych metod powiązanych ze sobą. Pierwszą realizującą dążenie do celu, oraz drugą sprawdzającą czy cel został osiągnięty. Jeśli cel został osiągnięty, oznaczało to opuszczenie instrukcji blokującej dalsze wykonanie;
- dla programisty pracującym w nowoczesnym i szybko rozwijającym się środowisku najważniejsza jest dobrze przygotowana dokumentacja, która pokryje wszystkie najważniejsze zagadnienia potrzebne programiście w jego przyszłej pracy. Tu niestety był wielki problem, gdyż najnowsza wersja, na której rozpoczęliśmy pracę nie posiadała poprawnie udokumentowanej funkcjonalności, a ta jak się potem okazało była bardzo potrzebna. Początek pracy przebiegał dość sprawnie i bez większych problemów, gdyż podstawowa funkcjonalność silnika Unity była opisana na tyle dobrze, że nie było większych problemów z wdrażaniem kolejnych funkcjonalności do aplikacji. Trudności zaczęły się gdy przyszła pora na zaimplementowanie obsługi rozgrywki wieloosobowej. Wersja Unity, której używaliśmy była pierwszą wersją z całkowicie zmienionym systemem sieciowym. Nie istniały żadne fora internetowe, filmiki czy dokumenty umożliwiające wdrożenie się w nowy system. Wymagało to mozolnej pracy i metodologii próba-błąd. Cały proces zajął prawie miesiąc, ale udało się. Zrozumienie nowych mechanizmów i wdrożenie ich w projekt przeszło gładko i po kolejnym miesiącu tworzenia gry uzyskaliśmy prototyp. Teraz gdy jesteśmy już na kroku finalizowania naszej pracy, cała dokumentacja Unity istnieje i jest opisana lepiej niż można by się tego spodziewać.

4. Ewaluacja

4.1. Weryfikacja

Po zakończonych pracach implementacyjnych postawiliśmy przed sobą pytanie: "Czy zrealizowaliśmy postawione sobie cele?". Aby znaleźć na nie odpowiedź postanowiliśmy przyjrzeć się im jeszcze raz i wskazać zaimplementowane przypadki użycia, które realizują dany cel. Tak więc:

1. umożliwienie tworzenia skryptów służących oprogramowywaniu obiektów w rozgrywce - zaimplementowane zostały funkcje zarządzania skryptami, takie jak tworzenie, edycja, usuwanie. Zarządzać nimi można zarówno z poziomu webserwisu jak i również podczas rozgrywki w oknie gry. Mamy możliwość zaprogramowania jednostek występujących w grze. Dzięki napisanym skryptom użytkownik może nakazać obiektom wykonywanie określonych czynności oraz zmianę zachowania w przypadku wystąpienia określonego zdarzenia. Kod napisany przez użytkownika jest interpretowany przez daną jednostkę. Może on być modyfikowany podczas rozgrywki, a uaktualniona wersja zostaje zapisana w bazie jako kolejna rewizja skryptu. Przechowujemy wszystkie rewizje kodu, co w przyszłości może posłużyć temu, aby zaimplementować funkcjonalność umożliwiającą użytkownikowi powracanie do wcześniejszych wersji skryptu;
2. personalizacja rozgrywki sieciowej - w stworzonym przez nas webserwisie mamy możliwość założenia konta, które jednoznacznie identyfikuje danego gracza. Gracze mogą zawierać znajomości poprzez znajomość swoich loginów;
3. statystyki i rankingi - w naszym projekcie wprowadziliśmy mechanizm rozgrywki umożliwiający zdobywanie punktów. Gracze rywalizują ze sobą, a aktualne zestawienia wyników przedstawiane są w tabelach na jednej z podstron webserwisu;
4. organizacja pojedynków - każda rozgrywka oparta jest o lobby, czyli kiluosobowy pokój, do którego dołączają gracze toczący później bój o wygraną. Umożliwiliśmy zakładanie pokoi chronionych hasłem (prywatne lobby), co ma dać użytkownikom opcję prowadzenia rozgrywki w określonym gronie znajomych;

5. czy nasza gra uczy programowania? - w naszym projekcie występuje funkcjonalność polegająca na pisaniu skryptów zachowań jednostek. Zmusza to użytkowników do algorytmicznego myślenia. Na jednej z podstron webserwisu umieściliśmy kilka przydatnych porad oraz w kilkunastu zdaniach przedstawiliśmy główne aspekty programowania. Po przeczytaniu takiego poradnika początkujący programista może poczuć pierwsze smaki tworzenia oprogramowania. Następnie tworząc swój pierwszy skrypt podczas rozgrywki i obserwując jego działanie

w przypadku sukcesu gracz może chcieć zgłębiać tajniki programowania wykraczające poza naszą grę.

4.2. Walidacja

Podczas wytwarzania oprogramowania dla użytkownika ważnym etapem jest sprawdzenie dopasowania produktu do oczekiwań przyszłych klientów. W naszym wypadku postanowiliśmy stworzyć ankietę dotyczącą jakości naszego produktu, którą następnie przedstawiliśmy kilku wybranym użytkownikom, testującym naszą platformę. Poniższe rysunki (rys. 4.1, 4.2, 4.3) przedstawiają pytania jakie zostały im zadane.

Oceń jakość webserwisu

Oceń jakość webserwisu (<http://eti.endrius.tk>).

*Required

Jak oceniasz estetykę webserwisu? *

Oceń jakie są Twoje ogólne wrażenia estetyczne interfejsu webserwisu.

1 2 3 4 5

Fatalnie ☐ ☐ ☐ ☐ ☐ Bardzo dobrze

Jak oceniasz intuicyjność interfejsu webserwisu? *

Jak oceniasz ogólną prostotę przemieszczania się po naszej stronie.

1 2 3 4 5

Trudne w użytkowaniu ☐ ☐ ☐ ☐ ☐ Łatwe i intuicyjne w użytkowaniu

Podziel się z nami opinią o webserwisie. *

Opisz co według Ciebie powinno zostać poprawione oraz jakie błędy w działaniu webserwisu znalazłeś.

Continue »

50% completed

Rys. 4.1.: Pierwsza strona ankiety opiniującej wytworzony produkt. Poświęcona opinii na temat webserwisu.

Ocena jakości gry.

Oceń jakość gry.

Jak oceniasz prostotę instalacji gry? *

1 2 3 4 5

Trudna ☐ ☐ ☐ ☐ ☐ Łatwa

Jak oceniasz prostotę poruszania się w menu gry? *

1 2 3 4 5

Trudna ☐ ☐ ☐ ☐ ☐ Łatwa

Jak oceniasz intuicyjność interfejsu podczas rozgrywki? *

1 2 3 4 5

Trudny w obsłudze ☐ ☐ ☐ ☐ ☐ Łatwy w obsłudze

Oceń prostotę pisania skryptów podczas gry. *

1 2 3 4 5

Z wielkim trudem programuje
jednostki

☐ ☐ ☐ ☐ ☐

Potrafię programować jednostki z zamkniętymi
oczami

Oceń przydatność wprowadzenia do języka programowania w grze. *

1 2 3 4 5

Praktycznie bezużyteczne ☐ ☐ ☐ ☐ ☐ Bardzo pomocne

Oceń jakość opisu funkcji dostępnych w grze. *

1 2 3 4 5

Opisy nie mówią jak użyć funkcji ☐ ☐ ☐ ☐ ☐ Opisy są zrozumiałe i wyczerpujące

Rys. 4.2. Część pierwsza drugiej strony ankiety opiniującej wytworzony produkt. Dotyczy ona opinii o grze, oraz opinii o skryptach.

Jak oceniasz grafikę w grze?

1 2 3 4 5

Brzydka ☐ ☐ ☐ ☐ Ładna

Jakie funkcje chciałbyś dodać do wykorzystania w skryptach? Podaj przykłady, wraz z krótkim opisem. *

Gdybyś mógł, to co poprawiłbyś w grze? *

Menu, grafika, skrypty, proces instalacji, etc..

Czy przed zagranie w naszą grę, programowałeś? *

- ☐ Nie
- ☐ Tak, ale tylko drobne programy i/lub skrypty
- ☐ Tak, brałem udział w większym projekcie(np. uczelnianym, open source, etc...)
- ☐ Tak, jestem zawodowym programistą

Czy granie w naszą grę wpłynęło na twoje umiejętności programistyczne? *

Np. poprawiło zdolność układania algorytmów, zamieniania algorytmów w kod lub inne formy programowania...

- ☐ Nie
- ☐ Tak
- ☐ Nie mam zdania

« Back

Submit

100%: You made it.

Never submit passwords through Google Forms.

Rys. 4.3. Część druga drugiej strony ankiety opiniującej wytworzony produkt. Dotyczy ona opinii o grafice zastosowanej w grze, oraz zawiera pytania o sugestie dotyczące dalszego rozwoju gry i możliwości przy pisaniu skryptów jednostek.

4.3. *Opinie użytkowników*

W badaniu uczestniczyło 15 respondentów. Pierwsza część formularza dotyczyła jakości webserwisu. Uczestnicy mieli ocenić między innymi:

- wrażenia estetyczne interfejsu webserwisu
- trudność poruszania się po stronie
- intuicyjność interfejsu

oraz podzielić się opinią, co według nich powinno zostać poprawione oraz jakie błędy zaobserwowali lub doświadczyli.

Wśród wyników można było zaobserwować zdecydowanie pozytywne opinie. 67% uczestników określiło swoje wrażenia estetyczne na ocenę 5, co dla nas jest motywującym wynikiem, gdyż ta część pracy, do której przyłożyliśmy najmniej pracy zyskała najlepsze oceny.

Trochę gorsze były wyniki dotyczące intuicyjności interfejsu webserwisu. Na 15 badanych, tylko 49% wybrało ocenę 5. Pozostałe 51% uczestników badania określiło się bardziej na poziomie 2-3, zarzucając często, że nie wiedzieli co zrobić i jakie kroki poczynić, aby rozpocząć rozgrywkę.

Z odpowiedzi na pytanie otwarte, czyli o sugestie dotyczące poprawek czy zauważonych błędów, niestety dowiedzieliśmy się najmniej. Zdecydowana większość uczestników odpowiedziała jedynie, że interfejs jest nieintuicyjny, co koniec końców sami zauważyliśmy analizując oceny.

Druga część ankiety dotyczyła już gry właściwej. Zawierała 7 pytań zamkniętych, w których badani mieli ocenić kwestie takie jak:

- trudność instalacji gry,
- intuicyjność poruszania się po menu głównym,
- intuicyjność interfejsu podczas rozgrywki,
- trudność pisania skryptów podczas gry,
- jakość funkcji dostępnych w grze,
- grafikę w grze.

Umieściliśmy również dwa pytania otwarte. Jedno dotyczące funkcji, które użytkownik chciałby dodać do wykorzystania podczas pisania skryptów. Drugie skupiające się na pomysłach i sugestiach, którymi warto się zainteresować.

Analizując oceny użytkowników stwierdzamy, że instalacja naszej gry jest procesem łatwym. Dostarczony przez nas prototypowy instalator pozwala użytkownikom w łatwy i przyjemny sposób przejść przez ten krok. 95% wszystkich ocen to ocena 5.

Zdecydowanie gorzej było w pytaniach odnośnie intuicyjności poruszania się po menu głównym oraz używania interfejsu podczas rozgrywki. Byliśmy świadomi tego, że nie jest on najlepszym ani najładniejszym interfejsem o jakim moglibyśmy marzyć. Wykorzystywał on podstawowe komponenty dostarczane nam przez silnik Unity. Niestety zabrakło nam czasu, aby przetransformować go do postaci przyjaznej użytkownikowi. Nam jako programistom

potrzebny był on tylko do uruchomienia gry, więc nie skupialiśmy się na nim. Potwierdzają to oceny, gdyż 87% badanych oceniło go na ocenę 2.

Analizując opinie dotyczące skryptów oraz języka programowania dostępnego w grze bardzo ciężko jest nam wyciągnąć konkretne wnioski, gdyż oceny są bardzo podzielone. Naszą ankietę przeprowadzaliśmy na osobach, które posiadały doświadczenie w programowaniu oraz wśród tych, które nigdy o programowaniu nawet nie słyszały. Średnia ocen wynosi 3, co nie jest zaskakujące mając na uwadze naszą grupę badanych. Pozytywne oceny wyraziło 57% badanych, zapewne Ci którzy mieli już styczność z programowaniem i potrafili odnaleźć się w środowisku. Pozostałe 43% to oceny z przedziału od 1 do 3. Z dużym prawdopodobieństwem stwierdzamy, że są to opinie osób, dla których programowanie jest tzw. "czarną magią". Pomimo faktu, że dostarczyliśmy poradnik wprowadzający do programowania, to żaden z badanych nie wyniósł z niego dostatecznej wiedzy. Może być to spowodowane zbyt dużym natłokiem informacji w mało przystępnej formie. Zapewne z tego faktu wynikają tak niskie oceny.

Bardzo zaskoczyła nas ocena dotycząca grafiki w grze. Świadomi tego, że nie przedstawiała ona więcej niż kilka chaotycznych kształtów z kolorową teksturą, to została ona dość pozytywnie oceniona. Aż 97% użytkowników oceniło ją na 5. Jest to dla nas wielkim zdziwieniem i trochę trudne do analizy. Z jednej strony mogła im się ona naprawdę spodobać, a z drugiej strony mógł to być sarkazm użyty w celu poinformowania nas, że ta grafika jest gorsza niż za czasów Commodore 64.

Kwestia pytań otwartych nas nie zaskoczyła, wiadomo że dla internautów są to najgorsze pytania. Nikt nie lubi pisać rozbudowanych wypowiedzi i nie spodziewaliśmy się w tym miejscu miarodajnych wyników. Na podstawie tych, które jednak się pojawiły, przeważająca część osób zasugerowała nam przygotowanie planszy instruktażowej, na której gracze prowadzeni "za rękę" mogliby nauczyć się i wdrożyć w proces pisania skryptów. Często pojawiały się też komentarze aby przygotować bardziej intuicyjny interfejs, który byłby przyjaźniejszy dla kolejnych użytkowników.

4.4. Kierunki dalszego rozwoju i plany ulepszeń

Plany najbliższych ulepszeń są bezpośrednio związane z informacjami zwrotnymi otrzymanymi z ankiet. Najsłabsze oceny otrzymaliśmy w tematach wyglądu obu komponentów oraz ich intuicyjności. Części badanych problem sprawiło też programowanie w naszej grze. To właśnie nad tymi elementami należałoby popracować w przyszłości.

Plany ulepszeń webserwisu to w szczególności:

- zwiększenie poziomu intuicyjności lobby poprzez np. komunikaty i historię tego co się działo w pokoju gry,
- uatrakcyjnienie wyglądu strony, poprawa layoutu,
- poszerzenie i przebudowa działu z instrukcjami gry - podział na rozdziały i sekcje, dodanie znacznie większej ilości przykładów.

Powyższe punkty (a szczególnie ostatni) związany jest bezpośrednio z planami ulepszeń gry, ponadto w aplikacji gry:

- uzupełnić sekcję opisującą uruchomienie gry - zaraz po wynikach ankiet, jeszcze w ramach pracy nad produktem dodaliśmy poradnik uruchamiania gry. Należy rozszerzyć go o to co należy zrobić w webserwisie. Można też nagrać krótki film instruktażowy, ukazujący proces uruchamiania;
- do gry należy dodać samouczek - pozwoli on użytkownikowi zrozumieć co dzieje się wokół niego, przeprowadzi przez proces pisania pierwszych skryptów, oraz wyuczy dobrych nawyków. Uważamy, że taki mechanizm będzie przyjaźniejszy niż długi poradnik na stronie.

Projekt wygląda na obiecujący, gdyż merytoryczna część gry została stosunkowo dobrze oceniona. Pozwala to zbierać pomysły na kierunki dalszego rozwoju, takie jak:

- dodanie sztucznej inteligencji do gry - możliwość gry z komputerem. Taki tryb gry powinien umożliwiać ustawienie poziomu trudności,
- możliwość gry z losowymi użytkownikami na wybranym poziomie (wprowadzenie mechanizmu rankingu),
- poszerzenie wachlarza dostępnych elementów w grze - predefiniowanych funkcji, budynków czy pojazdów w grze,
- możliwość tworzenia nowych planszy przez użytkownika.

5. Podsumowanie

W wyniku prowadzonych przez nas prac w ramach projektu dyplomowego inżynierskiego udało nam się zaimplementować dwuczęściowy system umożliwiający prowadzenie rozgrywki sieciowej między graczami. Stworzona przez nas gra powstała z myślą o rozwijaniu umiejętności programowania wśród jej użytkowników, co też udało się zrealizować implementując możliwość pisania własnych skryptów służących sterowaniu jednostkami występującymi w grze. Przed rozpoczęciem realizacji naszego projektu dokonaliśmy przeglądu rynku w celu poszukiwania podobnych rozwiązań. Mając problem ze znalezieniem jakiegoś ciekawszego projektu utwierdziliśmy się w przekonaniu, że realizacja takiego przedsięwzięcia nie będzie czasem zmarnowanym. Programowanie wśród ludzi postrzegane jest jako coś trudnego. To wszystko sprawiło, że chcieliśmy ukazać programowanie trochę w innym świetle dając użytkownikowi możliwość tworzenia prostych skryptów.

Wykorzystane przez nas podejście przy tworzeniu oprogramowania tj. stosowanie się do ogólnie przyjętych konwencji, podział struktury kodu, zachowanie wszelkich standardów, jak i również otwarta licencja, na której oparty jest nasz produkt umożliwia nowym osobom łatwe wdrożenie w projekt, co daje szansę na jego rozwój.

Wszystkie założone przez nas na początku cele zostały zrealizowane, zatem możemy stwierdzić, że projekt zakończył się sukcesem. Spowodowane to było dobrą organizacją pracy w zespole, prawidłowym podziałem obowiązków, koleżeńską pomocą w rozwiązywaniu problemów jak również odpowiednią motywacją promotora. Prawidłowym posunięciem z naszej strony był fakt, iż założyliśmy sobie, że na początku wakacji (tj., lipiec) zaczniemy fazę implementacji projektu. Umożliwiło to przemyślaną realizację zadań i skupienie się na zdefiniowanych celach, a nie na walkę z czasem. Projekt zakończył się sukcesem, ale nie oznacza to, że był on prowadzony wzorcowo. Niektóre funkcjonalności można było zrobić lepiej. Posiadając nasz produkt w obecnej formie chcielibyśmy go dalej rozwijać.

Wykaz literatury

1. M. Cantelon, M. Harter, T.J. Holowaychuk, N. Rajlich: NodeJS w akcji, Helion, Gliwice, 2015.
2. B. Green, S. Seshadri: AngularJS, Helion, Gliwice, 2014.
3. K. Chodorow: MongoDB - The Definitive Guide, Wydanie II, O'Reilly Media, Sebastopol, 2013.
4. L. Richardson, M. Amundsen, S. Ruby: RESTful WebAPIs, O'Reilly Media, 2013.
5. M. Menard: Game Development with Unity, Course Technology, 2012.
6. C. Abt: Serious Games, Upa, 1970.
7. Hevner A., Chatterjee S.: Design Research in Information Systems, Integrated Series in Information Systems, Springer Science Business Media, Heidelberg, 2010
8. March, S. T., Smith, G.: Design and Natural Science Research on Information Technology. Decision Support Systems vol. 15(4), December, pp. 251-266, 1995.
9. M. Kawick: Real-Time Strategy Game Programming Using MS DIRECTX 6.0, Wordware Publishing, USA, 1999.
10. Witold Bolt: Zapachy świeżej kawy: node.js, 2011, Dostępny w internecie: <http://www.trzeciakawa.pl/?p=109>
11. Karolina Marszałek: MongoDB: zarządzanie bazą w chmurze, 2015, Dostępny w internecie: <http://www.crn.pl/artykuly/poradniki-i-case-study/2015/07/mongo-db-zarzadzanie-baza-danych-w-chmurze>
12. I. Bououd, I. Boughzala: A Serious Game Supporting Collaboration Team, Nowy Jork, 2013.
13. Felicia, P. Digital games in schools: A handbook for teachers, http://games.eun.org/upload/GIS_HANDBOOK_EN.PDF
14. Sorensen, B.H. and Meyer, B. Serious games in language learning and teaching- a theoretical perspective, in Proceedings of the 2007 Digital Games research Association Conference (2007)
15. D. Djaouti, J. Alvarez, J.P. Jessel, O. Rampnoux: Origins of Serious Games, 2011.
16. J. Iivari: A paradigmatic analysis of information systems as a design science. In: Scandinavian Journal of Information Systems 19(2), pp. 39-64, 2007.
17. S. Dor: The Heuristic Circle of Real-Time Strategy Process: A StarCraft: Brood War Case Study, in The international journal of computer game research, 2014.
18. S.E. Dinehart: Real-time Strategy Games: History and Evolution, 2008.
19. E. Redmond, J. R. Wilson: Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement, Pragmatic Programmers, 2012.
20. T. Capan: Why The Hell Would I Use Node.js? A Case-by-Case Tutorial, dostępny w internecie: <http://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js>
21. Sue Blackman: Beginning 3D Game Development with Unity: All-in-one, multi-platform game development 1st Edition, 2011
22. Ryan Henson Creighton: Unity 3D Game Development by Example Beginner's Guide Beginner's Guide, 2010
23. Jeff W. Murray: C# Game Programming Cookbook for Unity 3D, 2013
24. Mike Geig: Unity Game Development in 24 Hours, Sams Teach Yourself (Sams Teach Yourself...in 24 Hours), 2013
25. Robert Nystrom: Game Programming Pattern, 2014

Wykaz rysunków

Rys. 1.1. Grafika pochodzi ze strony http://gogolev.net/	10
Rys. 1.2. Grafika pochodzi ze strony http://www.indiegames.com/	11
Rys. 1.3. Grafika pochodzi ze strony http://sourceforge.net/projects/robocode/	12
Rys. 1.4. Grafika pochodzi ze strony http://www.koth.org/pmars/	13
Rys. 1.5. Grafika pochodzi ze strony http://www.infionline.net/~wtnewton/corewar/evol/	13
Rys 2.1. Diagram przypadków użycia dla podsystemu webserwis	17
Rys 2.2. Diagram przypadków użycia dla podsystemu aplikacji gry	18
Rys. 3.1. Schemat warstw na których zbudowana jest obsługa sieciowa w Unity	36
Rys. 3.2. Schemat hosta jako klienta i serwer oraz podłączonych do niego graczy.	36
Rys. 3.3. Schemat graczy i ich instancji.	37
Rys. 3.4.: Autorytatywność zachowana na serwerze.	37
Rys. 3.5. Schemat komunikacji klienta z serwerem i odwrotnie.	38
Rys. 3.6. Wynik użycia polecenia z opcją --force	41
Rys. 3.7. SourceTree - narzędzie do zarządzania repozytorium gita	43
Rys. 4.1.: Pierwsza strona ankiety opiniującej wytworzony produkt. Poświęcona opinii na temat webserwisu.	48
Rys. 4.2. Część pierwsza drugiej strony ankiety opiniującej wytworzony produkt. Dotyczy ona opinii o grze, oraz opinii o skryptach.	49
Rys. 4.3. Część druga drugiej strony ankiety opiniującej wytworzony produkt. Dotyczy ona opinii o grafice zastosowanej w grze, oraz zawiera pytania o sugestie dotyczące dalszego rozwoju gry i możliwości przy pisaniu skryptów jednostek.	50
Rys. 3.1. Backlog w systemie JIRA	61
Rys. 3.2. Lista zadań z ich aktualnym statusem i osobą odpowiedzialną za wykonanie.	62
Rys. 3.3. Szczegółowy opis konkretnego zadania	63
Rys. 3.4. Jedna ze stron dostępnych w naszej wikipedii na serwisie github.com.	63

Wykaz tabel

Tabela 2.1: Przypadek użycia "Pobranie gry"	19
Tabela 2.2: Przypadek użycia "Rejestracja użytkownika"	19
Tabela 2.3: Przypadek użycia "Resetowanie hasła"	19
Tabela 2.4: Przypadek użycia "Zmiana hasła"	20
Tabela 2.5: Przypadek użycia "Aktualizacja danych profilowych"	20
Tabela 2.6: Przypadek użycia "Zarządzanie profilem użytkownika"	20
Tabela 2.7: Przypadek użycia "Tworzenie nowego lobby"	20
Tabela 2.8: Przypadek użycia "Usuwanie istniejącego lobby"	21
Tabela 2.9: Przypadek użycia "Dołączanie do istniejącego lobby"	21
Tabela 2.10: Przypadek użycia "Opuszczanie istniejącego lobby"	21
Tabela 2.11: Przypadek użycia "Przeglądanie dostępnych lobby"	21
Tabela 2.12: Przypadek użycia "Zmiana statusu gotowości gracza"	22
Tabela 2.13: Przypadek użycia "Startowanie lobby"	22
Tabela 2.14: Przypadek użycia "Zarządzanie skryptami użytkownika"	22
Tabela 2.15: Przypadek użycia "Wyświetlanie wyników"	23
Tabela 2.16: Przypadek użycia "Zakończenie lobby"	23
Tabela 2.17: Przypadek użycia "Udostępnianie rozgrywki."	23
Tabela 2.18: Przypadek użycia "Dołączanie do gry"	23
Tabela 2.19: Przypadek użycia "Startowanie gry"	24
Tabela 2.20: Przypadek użycia "Produkowanie czołgu (jednostki)"	24
Tabela 2.21: Przypadek użycia "Sterowanie czołgiem (jednostką)"	24
Tabela 2.22: Przypadek użycia "Atakowanie jednostką wrogiego obiektu"	25
Tabela 2.23: Przypadek użycia "Budowanie nowego budynku"	25
Tabela 2.24: Przypadek użycia "Rozbiórka budynku"	25
Tabela 2.25: Przypadek użycia "Zbieranie zasobów"	25
Tabela 2.26: Przypadek użycia "Programowanie jednostki (wykonywanie skryptu)"	26
Tabela 2.27: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Users”	27
Tabela 2.28: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Tokens”	28

Tabela 2.29: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Scripts”	28
Tabela 2.30: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Lobbies”	29
Tabela 2.31: Tabela przedstawiająca pola wchodzące w skład krotki kolekcji „Scores”	30
Tabela 3.1: Opis wykorzystanych komponentów NodeJS	39

Dodatek A: Raport końcowy

1. *Zespół projektowy*

Opiekun projektu: dr Adam Przybyłek.

Klient projektu: gracze z dowolnego przedziału wiekowego.

Autorzy projektu:

- Joskowski Andrzej (Katedra Algorytmów i Modelowania Systemów)
- Pielech Adrian (student piątego semestru na kierunku Informatyka na wydziale ETI)
- Urban Patryk (Katedra Algorytmów i Modelowania Systemów)
- Żyłowski Michał (Katedra Algorytmów i Modelowania Systemów)

2. *Osiągnięte rezultaty*

Produkty:

Udało nam się wytworzyć dwa komponenty: webserwis (zarządzanie rozgrywką) oraz aplikacja sieciowej gry RTS (część właściwa gry).

Repozytoria z kodem:

- https://github.com/urbanek32/SharpWars_WebService
- https://github.com/urbanek32/SharpWars_GameCore

Tymczasowy serwer www dla webserwisu:

- <http://eti.endrius.tk/>

Instrukcja dla uruchomienia gry:

- http://urbanek32.github.io/SharpWars_GameCore/

3. *Proces realizacji projektu*

a. Organizacja projektu

W bezpośrednim procesie wytwarzanie produktu uczestniczy czterech programistów:

- Joskowski Andrzej - programista wysokopoziomowy z bogatym doświadczeniem w wytwarzaniu aplikacji internetowych, pracującymi z takimi na co dzień.
- Pielech Adrian - programista niskopoziomowy, z dużym doświadczeniem w tematyce assemblera i języków programowania oraz ich interpretacji.

- Urban Patryk - programista mający doświadczenie zarówno w tematyce wysokopoziomowej, niskopoziomowej i komponentach sieciowych.
- Żyłowski Michał - specjalista sieciowy i programista niskopoziomowy.

Projekt podzieliśmy na dwa podprojekty i wybraliśmy dla nich właścicieli. Pierwszym podprojektem stał się webserwis, a jego właścicielem został Andrzej, jako osoba z bogatym doświadczeniem w wytwarzaniu stron internetowych. Michał jako osoba z niewielkim doświadczeniem wysokopoziomowym dołączył do Andrzeja aby w przypadku problemów czerpać z jego wiedzy.

Drugim podprojektem została gra. Jej właścicielem został Patryk, do którego jako wsparcie dołączył Adrian. Adrian miał również za zadanie przygotować obsługę języka programowania dla graczy - kluczowego elementu naszego produktu.

Dopiero pod koniec procesu wytwarzania oba moduły zostały ze sobą połączone.

b. Metodyka wytwarzania oprogramowania

Pracowaliśmy zmodyfikowanym pod nasze potrzeby SCRUMem tj:

- zmienna długość sprintu - ze względu na inne obowiązki zawodowe oraz uczelniane ustalenie sztywnej długości sprintu byłoby kłopotliwe do zastosowania;
- brak codziennych spotkań - ze względu na realizację ogromnej części projektu w wakacje, a co za tym idzie brakiem możliwości spotykania się codziennie na uczelni, postanowiliśmy zrezygnować z codziennych spotkań w zamian za spotkania raz w tygodniu. Były to spotkania typowo techniczne, na których dyskutowaliśmy nad obmyślonymi rozwiązaniami. Codziennie wymienialiśmy się informacjami o postępach używając komunikatora internetowego. Byliśmy stale w kontakcie zdalnym;
- wymagania i backlog były na bieżąco konsultowane z naszym promotorem.

Metodykę SCRUM wybraliśmy ze względu na niewielką liczebność zespołu, zmienność wymagań oraz brak czasu na przygotowanie obszernej dokumentacji. To ostatnie całkowicie i od samego początku przekreślało metodykę RUP.

c. Wsparcie narzędziowe

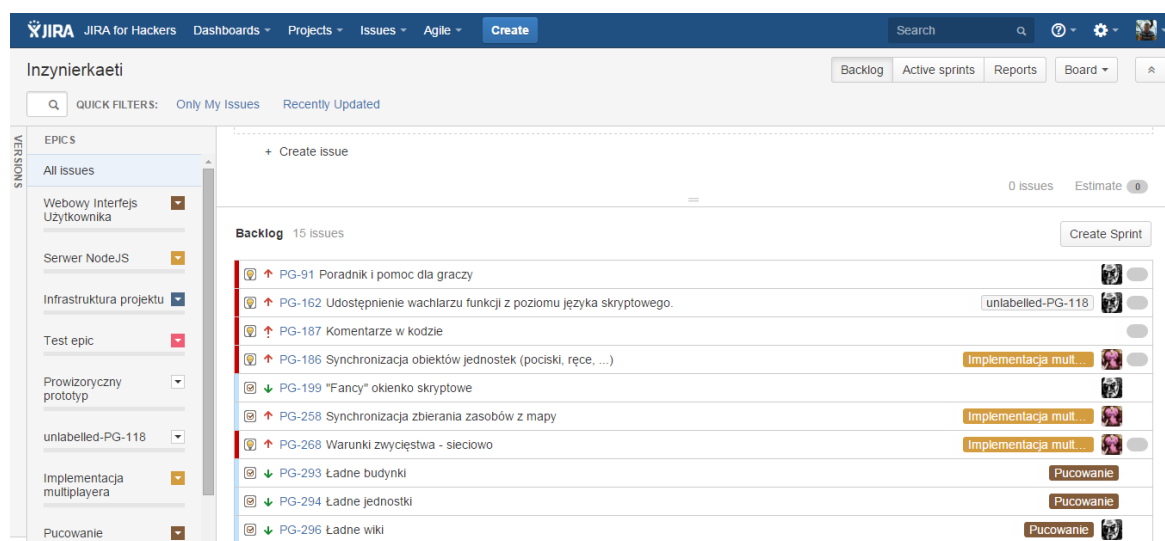
W procesie wytwarzania używaliśmy przede wszystkim dwóch narzędzi pomocniczych. Pierwszym było narzędzie wspierające metodykę SCRUM oraz dające możliwość dzielenia się zadaniami. Wybór padł na narzędzie firmy Atlassian - JIRA. Narzędzie udostępniane jest dla studentów za darmo, po uzyskaniu odpowiedniej licencji. Zdobycie jej wymagało od nas dokonania zamówienia popartego przedstawieniem kilku istotnych informacji dotyczących projektu, w którym oprogramowanie będzie używane oraz jego głównych założeń. Wspiera metodyki Agile, pozwala na rejestrację użytkowników, ustalanie priorytetów dla zadań, dzielenie

projektu na moduły i przypisywanie zadań do poszczególnych modułów. Zadania mają swoje typy - Bug, Improvement, Epic, Story itp. Nasza JIRA została uruchomiona na komputerze w laboratorium seminaryjnym Katedry Inżynierii Oprogramowania.

Drugie narzędzie, z którego korzystaliśmy jest ściśle związane z procesem programowania. Mowa tutaj o repozytorium kodu na githubie oraz samym narzędziu git. Na potrzeby projektu utworzyliśmy dwa repozytoria - jedno dla webserwisu i jedno dla gry. Repozytoria na githubie są dostępne za darmo. My jako studenci mamy możliwość utworzenia określonej liczby prywatnych repozytoriów, co też uczyniliśmy. Każde zadanie (ang. *task*) obecne w JIRZE miało swoje odbicie w postaci gałęzi w repozytorium danego podsystemu. Każde zadanie po wykonaniu było weryfikowane przez inne osoby dzięki możliwości wystawiania tzw. "pull requestów". Po akceptacji "pull requesta" gałąź była scalana z gałęzią „development”, a następnie usuwana. Migawki pochodzące z gałęzi „development” były scalane z naszą główną gałęzią „master” dopiero po osiągnięciu „milestone” w projekcie.

4. Dokumentacja procesowa

Całość dokumentacji procesowej znajduje się w systemie zarządzania projektem JIRA wykorzystywanym przy tworzeniu pracy (rysunki 3.1, 3.2, 3.3). Część dokumentacji znajduje się również w wikipedii udostępnionej w naszym repozytorium kodu na platformie github.com (rys. 3.4). Znajdują się tam głównie rzeczy, które były potrzebne nam jako programistom do szybkiego dostępu i częstej modyfikacji. Znajdowały się tam między innymi informacje o szczegółach naszego API, które było szeroko wykorzystywane przez aplikację gry. Znaleźć tam można również informację na temat konfiguracji środowiska testowego i modułów zależnych.



Rys. 3.1. Backlog w systemie JIRA

JIRA for Hackers

Dashboards
Projects
Issues
Agile
Create

Search

FILTERS

New filter

Find filters

My Open Issues

Reported by Me

Recently Viewed

All Issues

FAVORITE FILTERS

You don't have any favorite filters.

My Open Issues

Edited
Save as

Share
Export
Tools

Project: All
Type: All
Status: All
Assignee: All
Contains text
More
Advanced

Resolution: Unresolved

1-28 of 28

Columns

T	Key	Summary	Assignee	Reporter	P	Status	Resolution	Created	Updated	Due	Components	Si
	PG-91	Poradnik i pomoc dla graczy	Adrian Pielech	Andrzej Joskowski		OPEN	Unresolved	15/Jul/15	26/Oct/15			
	PG-298	Przykładowe skrypty dla graczy	Adrian Pielech	Patryk Urban		OPEN	Unresolved	06/Oct/15	26/Oct/15		GameCore	
	PG-4	Stwórz Backend oparty o NodeJS	Unassigned	Andrzej Joskowski		OPEN	Unresolved	30/Jun/15	26/Oct/15		WebService	
	PG-229	Skrypt do generowania certyfikatów ssl	Andrzej Joskowski	Andrzej Joskowski		OPEN	Unresolved	04/Sep/15	26/Oct/15		WebService	
	PG-171	TEST: testowanie API	Unassigned	Andrzej Joskowski		OPEN	Unresolved	14/Aug/15	26/Oct/15		WebService	
	PG-65	Dodanie mechanizmu CAPTCHA przy logowaniu i rejestracji	Unassigned	Michał Żyłowski		OPEN	Unresolved	12/Jul/15	26/Oct/15		WebService	
	PG-162	Udostępnienie wachlarza funkcji z poziomu...	Adrian Pielech	Adrian Pielech		IN PROGRESS	Unresolved	02/Aug/15	17/Oct/15		GameCore	

Rys. 3.2. Lista zadań z ich aktualnym statusem i osobą odpowiedzialną za wykonanie.

JIRA JIRA for Hackers Dashboards Projects Issues Agile Create

Projekt grupowy - 2016 / PG-94
Lobby

Edit Comment Assign More Close Issue Reopen Issue Admin Export

Details

Type: Story
Priority: Major
Component/s: GameCore, WebService
Labels: None
Sprint: Sprint 2

Status: **RESOLVED** (View Workflow)
Resolution: Fixed

People

Assignee: Unassigned
Reporter: Andrzej Joskowski
Votes: 0
Watchers: Stop watching this issue

Description

Mozliwe flow:

- Gracz loguje się do serwisu.
- Gracz wybiera "Gra".
- Następuje przekierowanie do strony:
 - Gracz tworzy nowe lobby.
 - Gracz dołącza do lobby.
- Jeżeli gracz jest masterem to ma do dyspozycji przycisk "Play", który uaktywnia się, gdy 3 pozostałych graczy zgłosiło aktywność.
- Jeżeli gracz nie jest masterem to po dołączeniu do istniejącego lobby ma do dyspozycji przycisk zgłaszający gotowość.

HipChat discussions

For more HipChat information, you need to link JIRA to HipChat.

Dates

Created: 15/Jul/15 11:01 PM
Updated: 10/Oct/15 10:08 PM
Resolved: 10/Oct/15 10:08 PM

Rys. 3.3. Szczegółowy opis konkretnego zadania

This repository Search Pull requests Issues Gist

urbane32 / SharpWars_WebService PRIVATE

Unwatch 2 Star 0 Fork

Przygotowanie środowiska
endrius1992 edited this page on 9 Jul · 6 revisions

Środowisko

Aby uruchomić projekt nodeJs wykonaj następujące kroki:

- Zainstaluj system linux (kroki podane na przykładzie Ubuntu 14.04 64-bit).
- Zainstaluj git'a: `sudo apt-get install git gitk git-gui`
- Zainstaluj mongoDB: `sudo apt-get install mongodb`
- Zainstaluj nodejs: `sudo apt-get install curl`

```
curl -sL https://deb.nodesource.com/setup_0.12 | sudo bash -
```

```
sudo apt-get install nodejs
```

Pages 8

- Home
- [Front End] Wiadomości: Warning, Info, Success, Error
- Ogólne
- Przygotowanie środowiska
- REST API: Gra
- REST API: Lobby
- REST API: Skrypty
- REST API: Zarządzanie użytkownikami

+ Add a custom sidebar

Rys. 3.4. Jedna ze stron dostępnych w naszej wikipedii na serwisie github.com.

5. Zmiany w trakcie projektu:

a. Organizacja projektu i role członków zespołu

Role w naszym zespole zostały jasno określone już na samym początku współpracy. Zналиśmy swoje mocne i słabe strony. Wspólnie przedyskutowaliśmy to nad czym będziemy pracować i takiego podziału się trzymaliśmy. W trakcie trwania projektu ten podział praktycznie się nie zmieniał. Drobnym wyjątkiem były sytuacje, gdy ktoś miał problemy i nie potrafił sobie

z nimi sam poradzić. Wtedy bardziej utalentowani członkowie zespołu pomagali sobie wzajemnie aby rozwiązać problem.

b. Metodyka i narzędzia

W punkcie 2b i 2c opisaliśmy wg jakiej metodyki i z jakim wsparciem narzędziowym działaliśmy. O ile w przypadku wsparcia narzędziowego byliśmy zdecydowani na korzystanie z oprogramowania JIRA oraz Githuba, o tyle mieliśmy wątpliwości co do samej metodyki. Oczywiście było, że wykorzystanie metodyk sformalizowanych np. RUP nie wchodzi w grę. Bardzo szybko zdecydowaliśmy się na SCRUMa. Początkowo miał być to pełny SCRUM ze wszystkimi jego zaletami i wskazówkami - regularne sprinty, codzienne spotkania w celu omawiania postępów. Niestety te elementy z powodów przytoczonych w punkcie drugim sprawiły konieczność modyfikacji SCRUMa.

Innych zmian w sposobie pracy podczas realizacji projektu nie przeprowadziliśmy.

c. Zakres i harmonogram projektu

Zakres prac, który na początku sobie przyjęliśmy, został zrealizowany w całości. Produktem końcowym miały być dwa komponenty: webserwis oraz aplikacja gry. Oba zostały zaimplementowane wg wcześniejszych założeń.

Harmonogram zrealizowanych przez nas prac został przedstawiony w tabeli poniżej. Jak już wcześniej sygnalizowaliśmy długość naszych iteracji była różna i w głównej mierze zależała od dostępności poszczególnych członków zespołu oraz nakładu prac związanych ze studiami.

Tabela 1: Harmonogram prac realizujących założenia projektu.

Okres	Etap	Wykonane zadania
1.04.2015 - 30.04.2015	Analiza rynku	Etap ten miał na celu dać nam odpowiedź na postawione pytanie "Czy podobne rozwiązanie istnieje na rynku?". Dokonałiśmy rozpoznania, zebraliśmy informacje na temat podobnych projektów.
1.05.2015 - 15.05.2015	Analiza wymagań	Sformułowanie wymagań, określenie celów, określenie architektury, dobór technologii.
16.05.2015 - 30.05.2015	Podział ról	Wyodrębnienie ról w zespole, podział prac na poziomie przypadków użycia.
1.06.2015 - 15.06.2015	Sprint 1	Przygotowanie do pracy: <ul style="list-style-type: none">• uzyskanie licencji studenckich na potrzebne oprogramowanie,• instalacja narzędzia służącego do zarządzania projektem (JIRA) na komputerze znajdującym się w laboratorium katedry,• utworzenie prywatnych repozytoriów na GitHubie,• instalacja i konfiguracja

Okres	Etap	Wykonane zadania
		<p>komunikatorów HipChat i TeamViewer,</p> <ul style="list-style-type: none"> • przygotowanie maszyn deweloperskich.
1.07.2015 - 15.07.2015	Sprint 2	<p>Webserwis:</p> <ul style="list-style-type: none"> • utworzenie szablonu dla projektu webserwisu, • stworzenie wiki dla projektu, • dodanie biblioteki walidacyjnej zapytania REST, • dodanie logiki odpowiedzialnej za zarządzanie użytkownikami po stronie serwera, • przygotowanie wstępnego layoutu strony www, • implementacja mechanizmu uwierzytelniania użytkowników. <p>Gra:</p> <ul style="list-style-type: none"> • utworzenie szablonu dla projektu gry, • utworzenie pierwszych obiektów na mapie gry, • wstępna obsługa myszy na planszy, • sterowanie dla poruszania jednostkami.
16.07.2015 - 31.07.2015	Sprint 3	<p>Webserwis:</p> <ul style="list-style-type: none"> • implementacja mechanizmu sesji użytkownika, • dodanie internacjonalizacji, • aktywacja konta poprzez otrzymanie wiadomości email, • funkcjonalność zmiany/resetowania hasła do konta, • edycja i wyświetlanie informacji o użytkowniku. <p>Gra:</p> <ul style="list-style-type: none"> • zarys użycia języka skryptowego LUA (menadżer języka), • funkcjonalność wyświetlania/dodawania zasobów gracza, • kolejkowanie budowanych jednostek, • możliwość sprzedaży budynku, • implementacja jednostki pobierającej surowce, • obsługa kamery gracza za pomocą strzałek, • animacje (zmiany kolorów podczas budowania budynków), • prototyp strzelania do wroga.
1.08.2015 - 15.08.2015	Sprint 4	<p>Webserwis:</p> <ul style="list-style-type: none"> • tworzenie nowego lobby, • dołączanie do lobby, • startowanie/zatrzymywanie lobby, • zmiana statusu graczy w lobby.

Okres	Etap	Wykonane zadania
		Gra: <ul style="list-style-type: none"> • pierwsze próby nowego systemu nawigowania, • ulepszona inteligencja zbieracza zasobów, • podstawowa synchronizacja obiektów pomiędzy masterem, a klientami.
16.08.2015 - 31.08.2015	Sprint 5	Webserwis: <ul style="list-style-type: none"> • zarządzanie skryptami (dodawanie, usuwanie, edycja itp.), • obsługa lobby po stronie klienta. Gra: <ul style="list-style-type: none"> • synchronizacja postępów budowania jednostek pomiędzy masterem a klientami, • refaktoryzacja kodu.
1.09.2015 - 15.09.2015	Sprint 6	Webserwis: <ul style="list-style-type: none"> • dodanie funkcjonalności usuwania i opuszczania lobby, • przygotowanie serwera do obsługi SSL, • obsługa skryptów po stronie klienckiej, • zmiana logiki resetowania hasła i aktywowania konta na opartą o jednorazowe tokeny, • refaktoryzacja kodu. Gra: <ul style="list-style-type: none"> • implementacja obsługi skryptów w trybie multiplayer, • dodanie animacji latających pocisków, • naprawa niepoprawnego poruszania się jednostki.
16.09.2015 - 30.09.2015	Sprint 7	Webserwis: <ul style="list-style-type: none"> • poprawiony layout strony www, • zmiana elementów API w celach synchronizacji z aplikacją gry, • wystawienie API testowego służącemu testowaniu komunikacji z aplikacją gry. Gra: <ul style="list-style-type: none"> • poprawiona animacja zbieracza zasobów, • implementacja network managera obsługującego API webserwisu, • implementacji warunków zwycięstwa (lokalnego i sieciowego), • udoskonalenie funkcjonalności pisania skryptów dla jednostek, • pierwsze zaprogramowanie jednostki przy użyciu skryptu, • edycja skryptów z poziomu

Okres	Etap	Wykonane zadania
		rozgrywki.
1.10.2015 - 15.10.2015	Sprint 8	<p>Webserwis:</p> <ul style="list-style-type: none"> • refaktoryzacja kodu, poprawiona responsywność strony, • dodawanie adresów IP gracza przy dołączaniu do lobby, • implementacja gromadzenia wyników graczy, prowadzenia statystyk, • przechowywanie rewizji kodów, • dodanie edytora dla tworzenia skryptu. <p>Gra:</p> <ul style="list-style-type: none"> • komunikacja z REST API webserwisu, • nowa mapa gry, • uruchamianie gry z parametrami poprzez webserwis, • dodanie wyświetlania informacji o błędach kompilacji skryptu.
16.10.2015 - 31.10.2015	Sprint 9	<p>Webserwis:</p> <ul style="list-style-type: none"> • dodanie poradnika dla graczy. <p>Gra:</p> <ul style="list-style-type: none"> • dodanie instalatora gry, • dodanie obsługi nowych funkcji skryptowych. <p>Inne:</p> <ul style="list-style-type: none"> • omówienie z promotorem sposobu przeprowadzenia ewaluacji, • prezentacja ukończonego produktu, • uzgodnienie formy (konspektu) pracy inżynierskiej.
1.11.2015 - 30.11.2015	Sprint 10	Napisanie pracy inżynierskiej oraz przeprowadzenie ewaluacji połączonej z analizą opinii klientów i opracowaniem wniosków definiujących ulepszenia projektu. Stworzenie raportu końcowego z przeprowadzonych prac.

d. Rzeczywiste nakłady pracy w stosunku do zakładanych na początku

W przypadku webserwisu udało nam się dość dobrze zaplanować pracę i bez większych trudności zaimplementowaliśmy większość krytycznych elementów do końca września. Wzajemne inspekcje kodu sprawiły, że zminimalizowaliśmy ilość pojawiających się błędów i problemów związanych z webserwisem. Dzięki temu nakład prac tylko minimalnie się powiększył w stosunku do naszego planu.

Inaczej było w przypadku samej aplikacji gry. Pozornie wydawało się, że wszystko idzie w dobrym kierunku. Dla nas jako twórców prace nad grą zakończyły się w październiku. Niestety pojawiło się sporo problemów, które poprawialiśmy niemal do końca listopada. Tutaj rzeczywisty nakład pracy okazał się dużo większy niż zaplanowany.

Nie jesteśmy w stanie przedstawić nakładów pracy w sposób mierzalny, gdyż nigdy nie utworzyliśmy planu mówiącego ile czasu będziemy potrzebowali na implementację poszczególnej funkcjonalności.

6. *Podział wykonanej pracy między członków grupy projektowej*

Poniżej przedstawiony jest wkład poszczególnych członków zespołu w wytworzenie produktu końcowego jakim jest gra SharpWars i jej webserwis, oraz wytworzenie dokumentacji dla wyżej wymienionego produktu.

Andrzej Joskowski:

Dokumentacja:

- “Wstęp” - podrozdziały 1.1, 1.2, 1.3, 1.4, 1.5, 1.8;
- “Analiza i projekt systemu” - podrozdziały 2.3 (część dotycząca webserwisu), 2.5;
- “Implementacja” - podrozdziały 3.1.2, 3.1.4, 3.1.8;
- “Ewaluacja” - podrozdziały 4.1;
- “Podsumowanie” - rozdział 5.

Implementacja:

- nadanie głównej struktury projektu,
- część serwerowa:
 - walidacja zapytań,
 - operacje bazodanowe,
 - mechanizm uwierzytelniania użytkownika,
 - logika zarządzania użytkownikami (rejestracja, logowanie, aktualizacja danych itp.),
 - logika zarządzania lobby (operacje CRUD, dołączanie/opuszczanie lobby przez graczy, rozpoczynanie rozgrywki, zmiana statusu gotowości gracza, kończenie rozgrywki),
 - logika zarządzania skryptami (operacje CRUD),
 - logika zarządzania wynikami (dodawanie, sortowanie, filtrowanie wyników),
 - bezpieczeństwo - oparte o jednorazowe tokeny mechanizmy resetowania hasła oraz aktywacji konta,
 - dodanie internacjonalizacji odpowiedzi serwera.
- część kliencka:
 - formularze dla niektórych funkcjonalności,

- zarządzanie profilem użytkownika,
- edytor tworzenia kodu skryptu,
- zarządzanie “aktywnym” lobby, uruchamianie aplikacji gry poprzez webserwis.

Michał Żyłowski:

Dokumentacja:

- “Analiza i projekt systemu” - podrozdziały 2.1, 2.2, 2.4;
- “Implementacja” - podrozdziały: 3.1.1, 3.1.5;
- “Ewaluacja” - podrozdział 4.4.

Implementacja:

- moduł rejestracji oraz logowania użytkowników w webserwisie (strona kliencka) tj.:
 - utworzenie formularzy na poszczególne elementy (rejestracja, logowanie, zmiana hasła),
 - zmiany hasła użytkownika w webserwisie,
 - resetowania hasła,
 - rejestracji użytkownika,
 - logowanie użytkownika,
 - edycja profilu użytkownika.
- moduł związany z pokojami gry od strony klienckiej tj:
 - utworzenie pokoju dla gracza,
 - zmiana statusu gry,
 - reagowanie na zmiany w pokojach graczy np (zakończenie gry).
- moduł związany ze skryptami graczy po stronie klienckiej webserwisu:
 - dodawania, edycji oraz usuwania predefiniowanych skryptów w webserwisie.
- używając gotowych paczek udostępnionych przez społeczność, z pomocą Andrzeja, poprawnie dodał mechanizm przełączania języka na stronie.
- zadbał o inne kosmetyczne elementy m. in:
 - opcje w menu strony,
 - osobna dyrektywa dla stopki strony,
 - poprawne przetwarzanie przekierowań na stronie,
 - treści semantyczne na podstronach i stronie głównej.
- moduł prezentacji wyników gry.

Patryk Urban:

Dokumentacja:

- “Wstęp” - podrozdziały 1.6, 1.7;

- “Analiza i projekt systemu” - podrozdziały 2.3 (część dotycząca gry);
- “Implementacja” - podrozdziały 3.1.3, 3.1.6;
- “Ewaluacja” - podrozdział 4.3.

Implementacja:

- stworzył projekt gry w Unity.
- zaimplementował w grze:
 - jednostki i budynki,
 - produkcję jednostek w budynkach,
 - zbieranie zasobów, wyszukiwanie ścieżek przemieszczania się, budowanie budowli przez poszczególne jednostki,
 - planszę na której dzieje się rozgrywka oraz planszę startową,
 - kolizje między obiektami,
 - cienie rzucane przez jednostki i budynki,
 - mechanizm generowania efektu cząsteczek pozostawianego za pojazdami ofensywnymi,
 - menu przed rozpoczęciem rozgrywki,
 - całokształt rozgrywki sieciowej w oparciu o komponenty Unity. W skład czego wchodziło (synchronizacja obiektów, autorytatywne przetwarzanie instrukcji, stworzenia serwera i klienta),
 - prototyp okna modalnego dotyczącego obsługi skryptów,
 - prototyp mechanizmu wykonywania skryptów,
 - prototyp klas i systemu dotyczącego wysyłania i parsowania odpowiedzi HTTP,
 - wysyłanie statystyk gry po jej zakończeniu do webserwisu,
 - zaawansowane modele graficzne jednostek do gry,
 - parsowanie parametrów startowych dostarczanych przez webserwis.
- stworzył następujące materiały pomocnicze:
 - napisał szablon pliku tworzącego skojarzenie protokołu *SharpWars://* z grą i nauczył członków zespołu technologii kryjącej się za tą funkcjonalnością,
 - stworzył instrukcję uruchomienia gry.

Adrian Pielech:

Dokumentacja:

- Analiza i projekt systemu - podrozdziały 2.6.7, 2.7;
- Implementacja - podrozdziały 3.2, 3.3;
- Ewaluacja - podrozdział 4.2.

Implementacja:

- dodał wsparcie Visual Studio 2013 Community dla projektu w Unity.

- zaimplementował w grze:
 - menedżer środowiska wykonania skryptów opartych o język Lua,
 - parsowanie skryptów użytkownika do pośredniej postaci bloków kodu,
 - sekwencyjne przetwarzanie bloków kodu,
 - rozwinięcie obsługi modalnego okna edytora skryptów,
 - rozwinięcie obsługi komunikacji sieciowej z webserwisem,
 - możliwość synchronizacji listy skryptów stworzonych w czasie rozgrywki z predefiniowaną listą skryptów użytkownika w webserwisie.
- stworzył następujące materiały pomocnicze:
 - generator odpowiedniego wpisu do rejestru, dodającego obsługę protokołu ShaprWars:// bazując na wcześniejszym prototypie,
 - wprowadzenie do języka programowania jednostek w grze,
 - dokumentację funkcji dostępnych dla użytkownika programującego jednostki.

7. Struktura raportu

Ten punkt zawiera informacje na temat raportu końcowego, uwzględniając odpowiednie umiejscowienie podpunktów względem szablonu. Niektóre punkty szablonu raportu końcowego nie znalazły się w niniejszym załączniku, ponieważ zostały wcześniej jawnie ujęte jako punkty pracy a chcieliśmy uniknąć redundancji.

Tabela 2: Tabela mapowania punktów z szablonu raportu na strukturę naszej pracy

Punkt z szablonu	Punkt w niniejszym załączniku	Praca inżynierska
1. Zespół projektowy	1. Zespół projektowy	-
2. Temat projektu	-	Punkty 1.2 i 1.3.
3. Kontekst projektu	-	Charakterystyka projektu: 1.4. Cele projektu: 1.2. Charakterystyka klienta: 2.1.
4. Osiągnięte rezultaty	2. Osiągnięte rezultaty	-
5. Proces realizacji projektu	3. Proces realizacji projektu	-
6. Dokumentacja a) techniczna b) procesowa	4. Dokumentacja procesowa	Punkty 2.3, 2.5, 3.1.6, 3.1.7.
7. Zmiany w trakcie projektu	5. Zmiany w trakcie projektu	-
8. Podział wykonanej pracy między członków grupy projektowej	6. Podział wykonanej pracy między członków zespołu	-
9. Podsumowanie	-	Punkt 5.
10. Opinia klienta	-	Punkt 4.3.

