POLITECHNIKA WROCŁAWSKA
WYDZIAŁ ELEKTRONIKI

KIERUNEK: INFORMATYKA (INF)
SPECJALNOŚĆ: INŻYNIERIA INTERNETOWA (INT)

PRACA DYPLOMOWA MAGISTERSKA

Analiza wydajności serwera gry sieciowej z wieloma użytkownikami

Analysis of performance of the server of an Internet multiuser game

AUTOR:
inż. Sławomir Dobroć

PROWADZĄCY PRACĘ:
dr hab. inż. Henryk Maciejewski
Katedra Informatyki Technicznej
(W4/K9)

OCENA PRACY:

WROCŁAW 2015

Spis treści:

1. Wprowadzenie

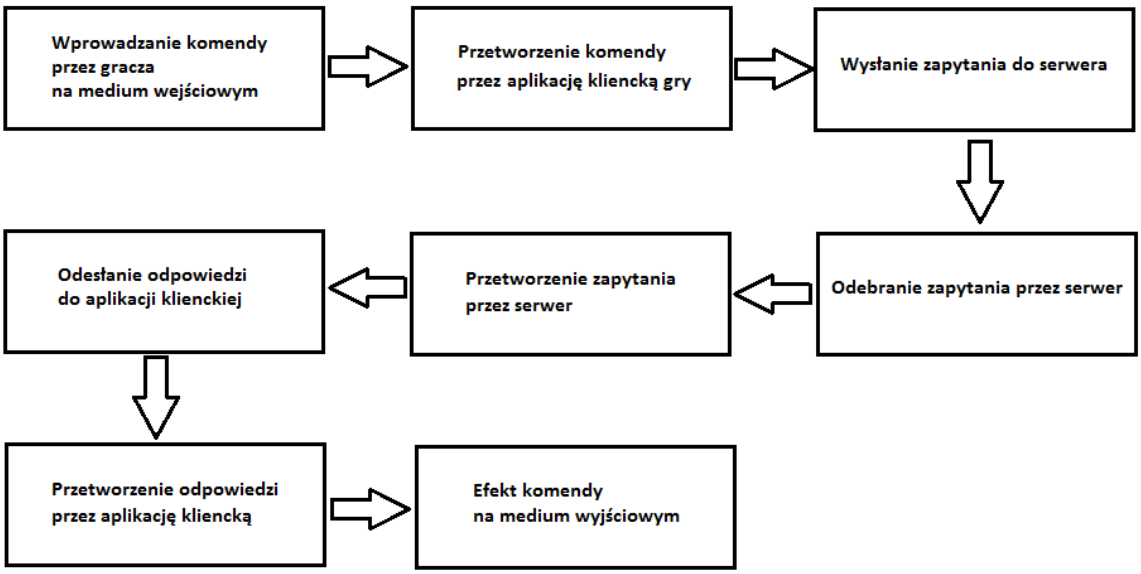
Wieloużytkową grą sieciową (ang. *Massively Multiplayer Online Game – MMOG*) nazywa się grę komputerową w której bierze udział naraz wielu graczy podłączonych do wspólnego serwera za pośrednictwem sieci internetowej (lub w szczególnych przypadkach: sieci lokalnej). Takie gry stają się coraz popularniejsze wśród graczy. Swoją popularność zawdzięczają dodatkowym wrażeniom, których nie są w stanie dostarczyć graczom gry jednoosobowe, na przykład współzawodnictwo z innymi ludzkimi graczami, czy rozgrywka kooperacyjna z przyjaciółmi. Serwery, które muszą obsługiwać rozgrywki sieciowe mają za zadanie utrzymywać rozgrywkę na jak najwyższym poziomie dla każdego gracza z osobna, to znaczy:

- utrzymywać stałe połączenie z klientem
- odpowiadać natychmiast na przychodzące zapytania od klienta
- aktualizować klienta na podstawie akcji wykonywanych przez innych klientów w jego otoczeniu

Ponadto twórcy gier komputerowych stawiają przed serwerami dodatkowe oczekiwania, mające na celu maksymalizować ich zyski z gry, które ograniczają się do stwierdzenia: maszyna pełniąca rolę serwera powinna mieć tylko tyle dostępnych zasobów, ile jest jej niezbędne do obsłużenia żądanej ilości klientów. Zasoby w tym wypadku to zarówno możliwości pojedynczej maszyny będącej serwerem (takie jak moc obliczeniowa, czy pamięć RAM) jak i zasoby sieciowe (prędkość i dostępność łącza). Stosunek ilości graczy do zasobów potrzebnych pojedynczemu serwerowi nazywa się wydajnością serwera - pożądane jest obsługiwanie jak największej ilości klientów przy minimalnych nakładach obliczeniowych serwera.

Niniejsza praca porusza wybrane tematy optymalizacyjne aplikacji sieciowych zwanych grami sieciowymi. Zawarto w niej rozległe wyjaśnienia wszystkich technicznych pojęć związanych z tego typu programami. Opisane zostały badania wybranych fragmentów aplikacji pod kątem wykazania problemów wydajnościowych, a następnie zaproponowane zostały rozwiązania znalezionych problemów. Główne aspekty, które autor porusza związane są z ruchem sieciowym generowanym między klientem a serwerem gry sieciowej, a także strukturą danych którą serwer gry musi dysponować w każdym momencie. Oba te elementy mają bezpośredni wpływ na komfort rozgrywki dla gracza. Największy poziom takiego komfortu można uzyskać, kiedy gracz ma wrażenie gry całkowicie płynnej. Płynność gry

oznacza, że gracz w ogóle nie odczuwa skutków komunikacji swojej aplikacji klienckiej z serwerem gry. Oznacza to, że dla gracza niezauważalny jest czas, jaki jego aplikacja musi czekać od momentu wydania komendy w grze przez medium wejściowe (na przykład: klawiaturę) do momentu zobaczenia efektu tej komendy na medium wyjściowym (na przykład: monitorze).

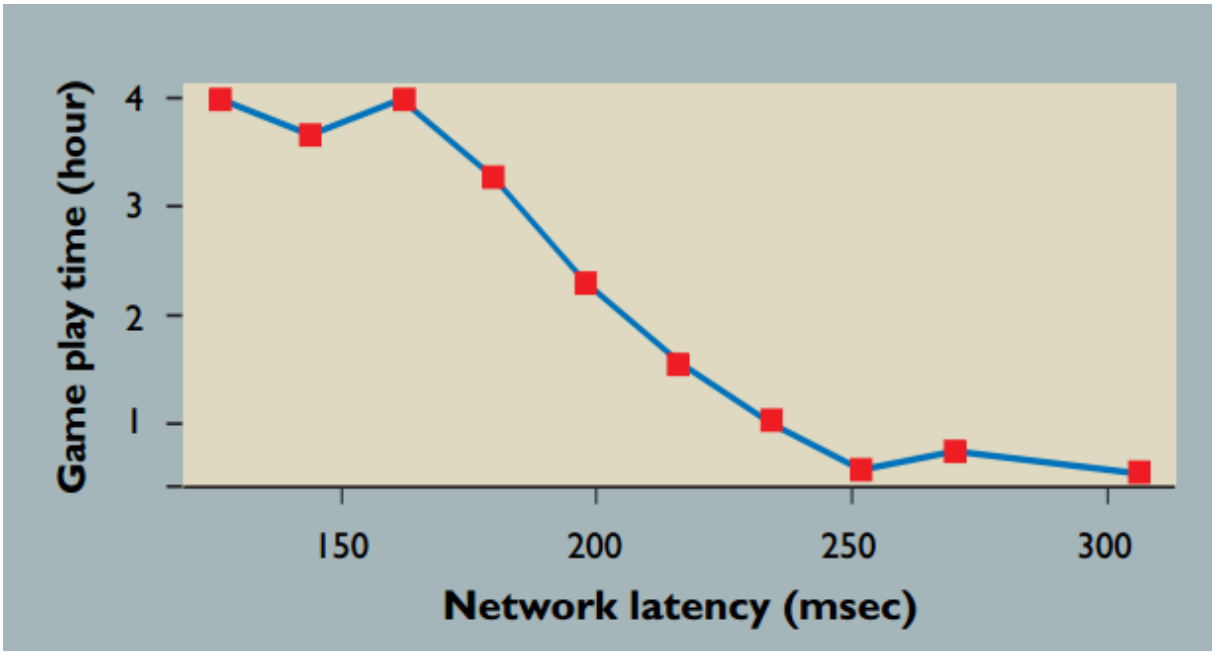


Rysunek 1.1 Kolejne akcje składające się na płynność rozgrywki w grze sieciowej

Aplikacja klienta gry sieciowej służy tylko do reagowania na odpowiedzi serwera i przekazywania zapytań klienta do serwera. Zanim więc komenda wprowadzona przez gracza pojawi się jako efekt na medium wyjściowym, musi być przetworzona przez serwer, który decyduje o tym w jaki sposób na nią odpowiedzieć. To oznacza, że pojedyncze polecenie wprowadzane do gry musi przejść przez łańcuch zdarzeń jak zaprezentowano na rysunku 1.1. Opóźnienia, jakie wywołuje taki łańcuch zdarzeń nie mogą przekroczyć pewnych krańcowych wartości, które przyczyniają się bezpośrednio do rezygnacji graczy z dalszej gry. Czułość graczy na takie opóźnienia jest bardzo różna i nie da się udzielić jednoznacznej odpowiedzi na pytanie jaki poślizg w czasie odpowiedzi jest za duży, gdyż dla każdej osoby jest to inna wartość[3]. W niniejszej pracy, na podstawie wykresu 1.1, ustalono umowne wartości opóźnień do których autor będzie się odnosił w czasie analizy wyników symulacji. Są to:

- Do 150 milisekund - brak tendencji do opuszczania gry
- Pomiędzy 150 a 200 milisekund - lekkie tendencje do opuszczania gry
- Powyżej 200 milisekund - bardzo duże tendencje do opuszczania gry

Oprócz omawianych w tej pracy aspektów ruchu sieciowego i struktury danych przetwarzanych przez serwer, jest wiele innych zagadnień związanych z pracą serwerów gier sieciowych. One również mają wpływ na wydajność serwerów i zostały omówione w rozdziale 8.



Wykres 1.1 Zależność opóźnień w grze od czasu gry w grach sieciowych[3]

Rozdział drugi przybliża czytelnikowi pojęcie gry sieciowej oraz serwera gry sieciowej pod kątem omawianych problemów. Wyjaśnione zostaje w nich dlaczego powstała ta praca oraz w jaki sposób należy spojrzeć na omawiane zagadnienie, aby móc zlokalizować ewentualne problemy optymalizacyjne.

Rozdział trzeci opisuje napisany przez autora pracy program komputerowy, który został użyty do niektórych badań wydajności. Opisane są w nim także kryteria, którymi kierował się autor badając aplikację i szukając problemów optymalizacyjnych.

Rozdziały czwarty i piąty przedstawiają badania przy pomocy powyższej aplikacji i zaproponowane rozwiązania. W każdym z tych rozdziałów został poruszony i kompleksowo omówiony osobny problem wydajnościowy wraz z wykazaniem, że zaproponowane rozwiązanie poprawia jakość omawianego serwera gry sieciowej.

Rozdział szósty skupia się wokół kolejnego problemu, który z opisanych w nim powodów nie mógł zostać zbadany przy pomocy wyżej wspomnianego programu. Problem omawiany jest w naturze teoretycznej, jednak poparty wieloma praktycznymi przykładami.

Ostatnie rozdziały zostały poświęcone na podsumowanie pracy i zaproponowanie dalszego obszaru prac.

2. Struktura gry sieciowej

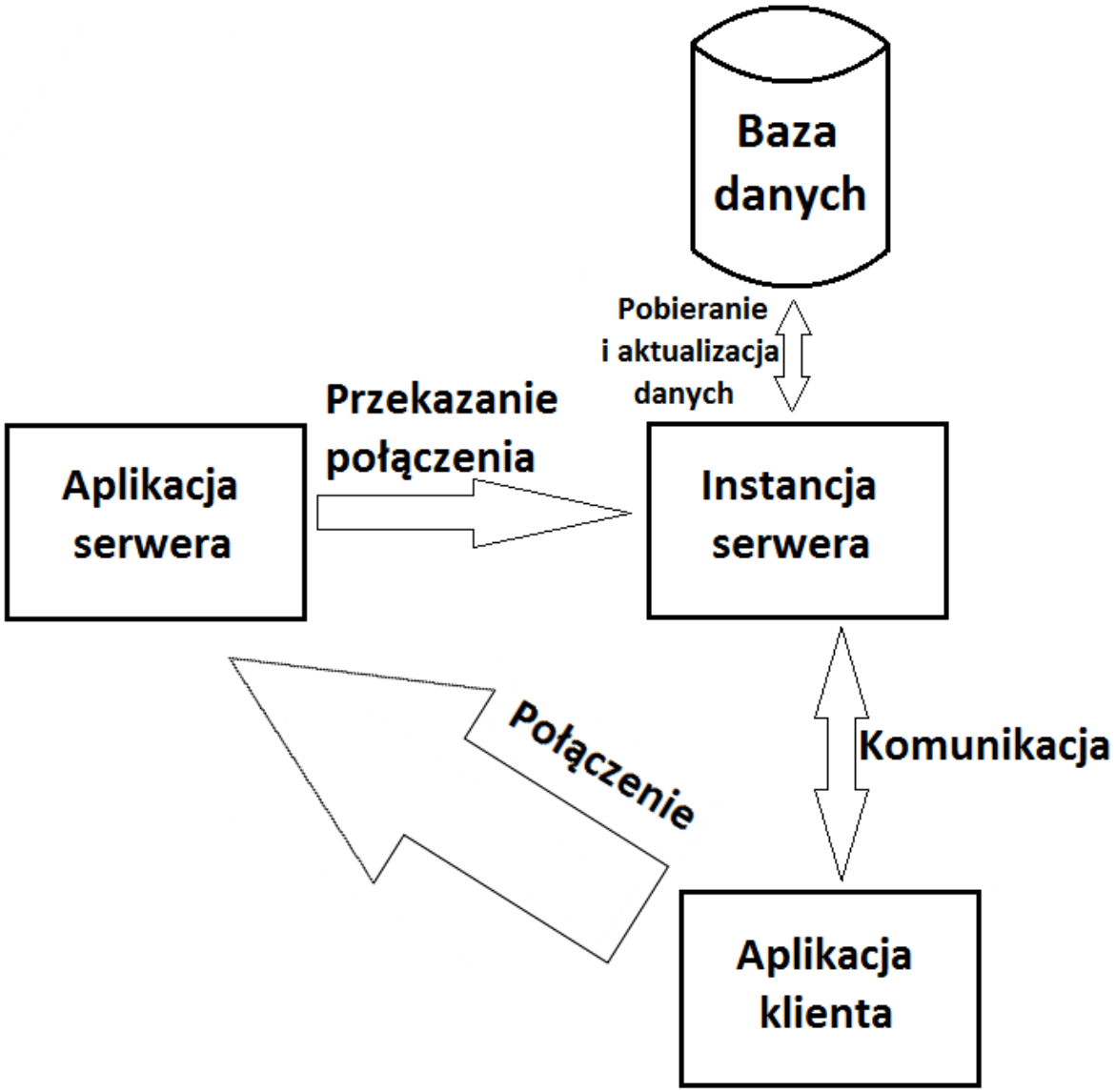
Gra sieciowa, niezależnie od jej konkretnego gatunku, składa się z co najmniej dwóch aplikacji: klienta oraz serwera. Głównym zadaniem serwera jest zbieranie połączeń od klientów i zarządzanie nimi w taki sposób, aby umożliwić ich wspólną interakcję. W tym celu niezbędne jest utrzymywanie stałego połączenia między klientem a serwerem poprzez pewien (wybrany) protokół komunikacyjny. Najczęściej w tym celu wybierane są protokoły TCP lub UDP, jednak spotykane są również ich pośrednie warianty[1]. Jest to bezpośrednio zależne od rodzaju gry sieciowej, który aktualnie się rozważa.

Tabela 2.1 Podstawowe rodzaje gier sieciowych

Rodzaj gry sieciowej	Opis gry sieciowej
Turowa	Tylko jeden gracz naraz wykonuje akcję, reszta go obserwuje
Strategia czasu rzeczywistego	Gracze kontrolują wiele zasobów (zwane jednostkami w grze), które naraz mogą ze sobą wchodzić w interakcję w czasie rzeczywistym
Przygodowa	Gracze kontrolują jedną postać na ustalonej mapie - mogą przy jej pomocy wchodzić w interakcję z elementami gry lub innymi graczami
Akcja	Rozgrywka bardzo dynamiczna w której gracz musi się wykazać refleksem, najczęściej współzawodnicząc z wieloma innymi graczami naraz

W niniejszej pracy omawiany jest trzeci rodzaj gry opisanej w tabeli 2.1 - gra przygodowa. Różne rodzaje gier stawiają nacisk na różne aspekty komunikacji sieciowej. Przykładowo gry akcji wymagają najczęściej protokołu UDP z powodu ilości wiadomości na jednostkę czasu, które mogą zostać wysłane[4]. Opisywany w tej pracy rodzaj gier, przygodowe, charakteryzuje się tym, że każdy gracz jest ustaloną (jedną) postacią w grze w dużym świecie gry z wieloma dodatkowymi elementami, takimi jak przedmioty, postaci niezależne (o podobnej charakterystyce co postać gracza, lecz sterowane przez serwer) lub inni gracze. Ważną cechą rozważanego typu gry jest również fakt, że na ustalonej wirtualnej przestrzeni (mapie gry) naraz może być podłączonych setki lub tysiące graczy. Oznacza to, że pojedynczy serwer musi być zaprojektowany w taki sposób, aby był w stanie obsłużyć takie ogromne ilości połączeń.

Aplikacja serwera, która odpowiada za zarządzanie połączeniami przychodzącymi od nowych klientów, nie służy sama w sobie do obsługiwania rozgrywki klienta. W aplikacji wydzielone są dodatkowe procesy (lub wątki, zależnie od implementacji) do których główna aplikacja przekazuje takie połączenie. Taki dodatkowy proces nazywany jest instancją serwera.



Rysunek 2.1 Struktura gry sieciowej i powiązania między jej elementami

Ponieważ ilość połączeń na pojedynczy serwer może być tak ogromna, ważny jest odpowiedni dobór danych, które instancja serwera będzie wysyłać do klienta. W przeciwnym wypadku połączenie z klientem zostałoby bardzo szybko przeciążone, co skutkowałoby

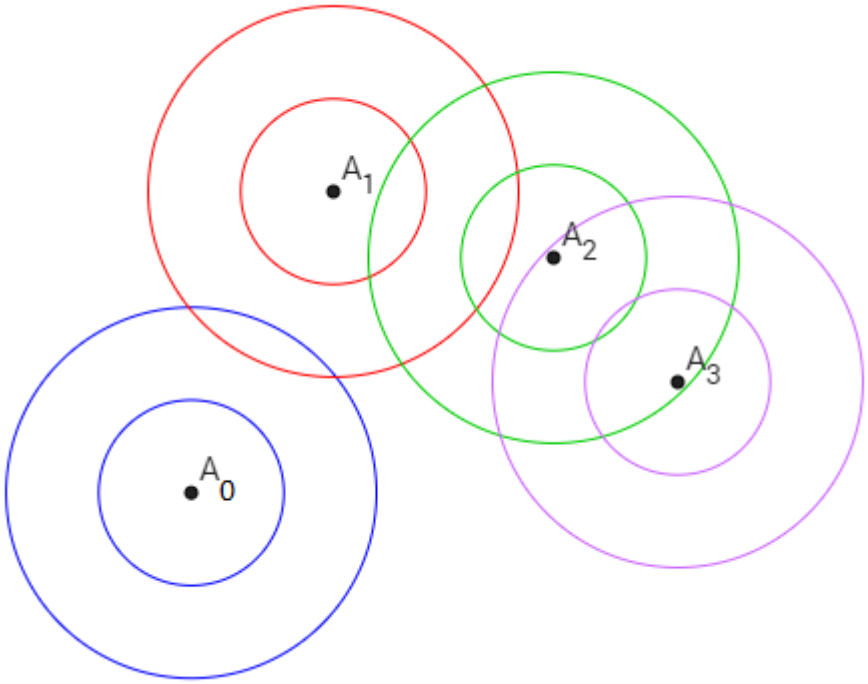
ogromnymi opóźnieniami po stronie klienta lub nawet zerwaniem połączenia. W tym celu w grach sieciowych implementowane są Systemy Zarządzania Zainteresowaniem (SZZ - ang. *Interest Management System*[2]). Po stronie aplikacji serwera zostaje wydzielona część funkcjonalności odpowiedzialna za podział danych, które instancja serwera normalnie wysłałaby do klienta. Odbywa się to w taki sposób, aby ze wszystkich danych wydzielić tylko te, które są w aktualnym momencie istotne dla aplikacji klienta i tylko te dane zostają faktycznie wysłane.

Jedną z metod implementacji SZZ jest Pole Zainteresowania (PZ - ang. *Area of Interest[5]*). Metoda ta wprowadza dwa dodatkowe pojęcia do gry sieciowej, które stają się atrybutami gracza:

- pole widzenia - zasięg dookoła postaci, który jest przez nią widoczny. Wszystko wewnątrz tego pola jest prezentowane ludzkiemu graczowi na monitorze komputera
- pole zainteresowania - rozszerzenie pola widzenia, które ma w sobie dane o miejscu do którego postać gracza może się poruszyć w najbliższej przyszłości. Wszystko wewnątrz tego pola musi być znane aplikacji klienta

Pole zainteresowania jest zawsze większe od pola widzenia i obie te wartości są reprezentowane przez punkt (położenie postaci gracza na mapie) i promień odległości. Celem takiego zabiegu jest ograniczenie ilości danych, które instancja serwera musi dostarczyć aplikacji klienta z założeniem, że nie wiadomo w którą stronę gracz poruszy się w najbliższej przyszłości. Mając takie same informacje o obiektach w określonym promieniu dookoła gracza, serwer nie musi wykonywać dodatkowych obliczeń związanych z przewidywaniem w którą stronę gracz poruszy się w najbliższej przyszłości.

Widoczna na rysunku 2.2 wizualizacja przedstawia pole widzenia i pole zainteresowania każdego gracza A_0 - A_3 jako okręgi o mniejszym i większym promieniu ze środkiem w miejscu, gdzie gracz faktycznie się znajduje. W przedstawionej sytuacji żaden z graczy A_0, A_1, ani A_2 nie mają o sobie nawzajem informacji, gdyż ich wzajemne położenia nie zawierają się w polach zainteresowań pozostałych postaci. Jedynie gracze A_2 oraz A_3 są świadomi siebie nawzajem, a raczej: aplikacja klienta każdego z nich ma informację o tym drugim, jednak sami gracze jeszcze siebie nawzajem nie widzą.



Rysunek 2.2 Wizualizacja pola widzenia i pola zainteresowania dla kilku postaci

Instancje serwerów odpowiadające za tych graczy, którzy mają być świadomi swojej wzajemnej obecności muszą przesyłać na bieżąco informacje o tych graczach do swojej aplikacji klienckiej dopóki taki obiekt (inny gracz) pozostaje w polu zainteresowania aktualnego klienta. To wymaga, aby instancje serwera miały swobodny dostęp do informacji o wszystkich obiektach, jakie znajdują się aktualnie w grze.

3. Środowisko do badania zachowania serwera

Do badania zachowania serwera przygotowano specjalne programy symulujące jego zachowanie w różnych warunkach. Aplikacje te miały za zadanie wykonywać podstawowe funkcje gry sieciowej przy jednoczesnym zbieraniu danych statystycznych zarówno ze strony klienta gry jak i serwera. Serwer został stworzony tak, aby realizował schemat z rysunku 2.1. Aplikacja na ustalonym porcie nasłuchuje na połączenia od klientów i tworzy nowy wątek instancji, który zajmuje się dalszą komunikacją z klientem. Aplikacją klienta w symulacji był prosty program do łączenia się z serwerem i bezustannego wysyłania nowych zapytań imitujących komendy gracza (takie jak poruszanie się po mapie) z jednoczesnym zbieraniem danych do pomiarów. Ponieważ w symulacji aplikacja klienta była zautomatyzowana, nie było żadnych momentów bezczynności programu. Takie momenty w normalnych aplikacjach występują w momentach, kiedy gracz nie wprowadza żadnych komend (na przykład zastanawia się nad kolejnym ruchem lub analizuje wynik ostatniej operacji na ekranie). Na potrzeby badania wydajności serwera, aplikacja która generuje ruch bezustannie znacznie lepiej zobrazuje sytuację skrajną kiedy serwer nie jest już w stanie obsłużyć większej ilości klientów przy wybranych ustawieniach.

W pracy istotny był wybór poszczególnych cech aplikacji, które można ze sobą bezpośrednio porównać. Inne cechy były dobierane tak, aby móc wykazać ich graniczne wartości, czyli takie przy których większa wartość skutkowałaby drastycznym obniżeniem wydajności serwera. Tabela 3.1 opisuje właśnie takie cechy, jednak niektóre z nich zostały jedynie wymienione, gdyż wymagają głębszej analizy. Ilość komunikatów wysłanych przez klienta aplikacji oznacza ile razy podłączony już gracz wykonał akcję, zanim postanowił wyłączyć aplikację. Cecha ta w jednoznaczny sposób definiuje długość przeprowadzania badania. Zautomatyzowany program do uruchamiania klientów gry został zaprogramowany w taki sposób, że po podłączeniu do serwera każdy z nich czeka na pozostałych zanim zaczną zasypywać swoje instancje serwerów zapytaniami. Oznacza to, że każdy klient aplikacji w danym badaniu powinien mieć zbliżone dane na temat czasu oczekiwania na odpowiedź od serwera.

Tabela 3.1 Wybrane cechy aplikacji do badania zachowania serwera

Cecha	Wybrane wartości cechy w aplikacji
Protokół komunikacyjny	TCP
Rodzaj przesyłanych danych	JSON, binarne
Ilość graczy podłączonych do jednego serwera	20, 40, 80, 160
Źródło współdzielenia danych między instancjami serwerów	Baza danych, blok pamięci współdzielonej
Ilość komunikatów wysłanych przez klienta aplikacji	Różne, stałe dla wszystkich podłączonych klientów aplikacji
Wielkość pola zainteresowania gracza	Objaśnione w rozdziale 7.
Wielkość mapy gry	Objaśnione w rozdziale 7.
Podział mapy na Sektory	Objaśnione w rozdziale 7.

Trzy ostatnie wiersze tabeli 3.1 odnoszą się do cech, które są ze sobą ściśle powiązane i wymagają określonych modeli matematycznych aby móc wybrać ich konkretne wartości. Pole zainteresowania gracza zostało omówione szczegółowo w poprzednim rozdziale. Wielkość mapy bezpośrednio wpływa na pole zainteresowania graczy, gdyż na większej mapie gracze rzadziej będą się spotykać. Oznacza to również, że serwer ma więcej statycznych elementów gry, które musi ewentualnie obsłużyć. Pojęcie Sektorów mapy jest związane zarówno z wielkością mapy w grze, jak i z wielkością pola zainteresowania graczy. Opisuje sposób w jaki mapa zostaje podzielona na mniejsze, przylegające do siebie segmenty które są obsługiwane przez osobne serwery, dzięki czemu obciążenie pracy jest rozproszone pomiędzy wiele aplikacji serwerów.

W tabeli 3.1 rodzajem przesyłanych danych określono różne rodzaje formatowania, jakim można poddać dane potrzebne do przesłania. Formatowanie stosuje się, aby przygotować dane do przesłania przez sieć tak, aby po odebraniu ich przez drugą stronę mogły zostać przywrócone do pierwotnej postaci. Zabieg ten jest niezbędny, ponieważ wszystko co jest przesyłane protokołem TCP jest domyślnie interpretowane jako podstawowy łańcuch znaków. Zaproponowany w tym wypadku format danych JSON[6] (ang. *JavaScript Object Notation* - notacja obiektów JavaScript) jest najprostszy w implementacji. Dane nim serializowane zajmują najmniej bajtów w porównaniu do innych dedykowanych notacji, takich jak XML[6] (ang. *Extensible Markup Language* - rozszerzalny język znaczników).

Druga z zaproponowanych opcji, formatowanie binarne, cechuje się skompresowaniem do minimum ilości danych przesyłanych w każdej wiadomości. Rozwiązanie to niesie jednak za sobą konieczność własnego zaimplementowania protokołu serializowania i deserializowania danych.

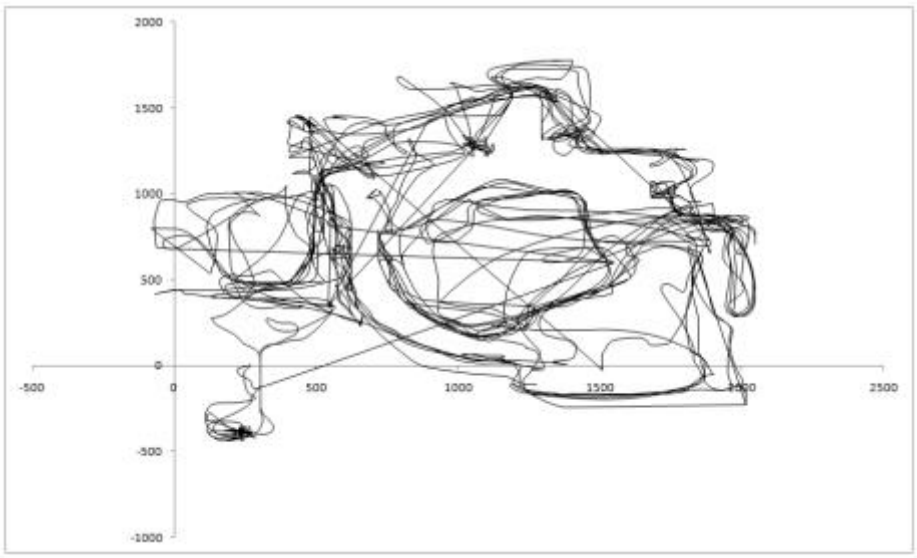
Serwer był testowany w warunkach idealnych - bez żadnych zakłóceń sieciowych i maksymalnej dostępności zasobów lokalnych, co oznacza, że na maszynie na której uruchomiono serwer nie były uruchomione żadne inne aplikacje. Maksymalną dostępność zasobów sieciowych zapewniono poprzez użycie adresu lokalnego (ang. *localhost* - adres lokalny komputera). Mierzonymi wielkościami były:

- średni czas potrzebny serwerowi na przetwarzanie danych od momentu otrzymania zapytania od klienta do wysłania odpowiedzi
- średnia ilość danych w bajtach wysyłana do klienta w każdej odpowiedzi
- średni czas oczekiwania klienta na odpowiedź od serwera od momentu wysłania zapytania do momentu otrzymania odpowiedzi

Taki dobór danych pozwalał na udzielenie ważnych odpowiedzi w każdym badaniu:

- ile czasu serwer potrzebował na przetworzenie danych?
- ile czasu klient oczekiwał na odpowiedź, zanim gracz mógł zobaczyć efekty komendy na ekranie i wykonać kolejny ruch?
- ile czasu zajęła sama komunikacja między aplikacjami?
- jak dużo danych było przesyłanych w każdej wiadomości?

W innych pracach, które zajmują się analizą wydajności serwera[2] możliwe były dokładniejsze analizy ruchu graczy w grze sieciowej. Działo się tak dlatego, że autor rozważał w takiej pracy konkretną grę sieciową oraz konkretną mapę w tej grze. Efektem takiego uściślenia zakresu prac były pomiary do analizy jak na rysunku 3.1.



Rysunek 3.1 Ruch gracza na pewnej określonej mapie[2]

W niniejszej pracy taka dokładność nie jest możliwa, gdyż autor nie rozważa konkretnej gry, a jedynie klasę gier sieciowych. Wymagało to pewnych uogólnień w badaniach i przyjmowania założeń, które w szczególnych przypadkach byłyby znane jako dane (na przykład: znajomość punktu początkowego gracza na mapie).

W celu poprawnej interpretacji badań potrzebne było ustalenie pewnego punktu odniesienia - idealnych wyników, jakie jest w stanie zaoferować serwer dla podstawowych ustawień. Na podstawie tabeli 3.1 wyznaczono takie podstawowe ustawienia serwera jako:

- dane przesyłane formatem JSON
- jeden podłączony gracz
- dane współdzielone przez bazę danych
- gracz wykonał dwieście akcji w czasie sesji

Jak napisano wcześniej, formatowanie metodą JSON wymaga odpowiedniego serializowania danych. Notacja ta wykorzystuje odpowiednie znaki, aby zamodelować relację „zmienna-wartość”, a także grupować dane w listy. W ten sposób przedstawić można dowolny rodzaj zmiennych niezależnie od wykorzystywanego języka programowania, ponieważ dane sformatowane w JSON są zawsze łańcuchem znaków. W niniejszej pracy potrzebne było zdefiniowanie podstawowych typów informacji, które mogą być przekazywane między klientem a serwerem gry sieciowej. Listing 3.1 przedstawia trzy rodzaje danych niezbędne do przeprowadzenia symulacji: Odpowiedź, Aktualizację Gracza i Aktualizację Środowiska.

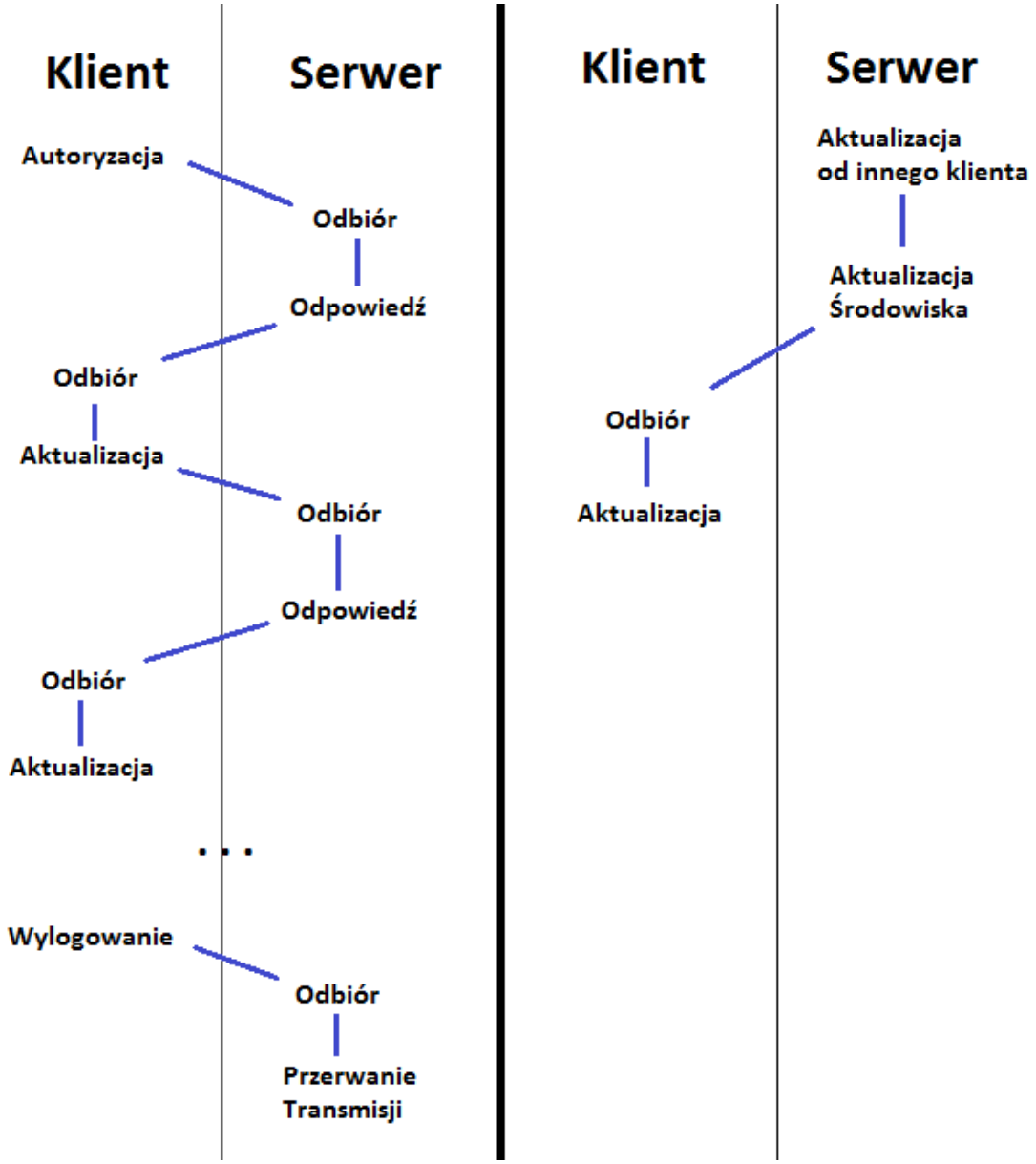
```
{
    "value0": {
        "type()": "OkResponse",
        "serverAllows": true
    }
}
{
    "value0": {
        "type()": "UpdatePlayer",
        "delta": {
            "first": 5,
            "second": 0
        },
        "actionId": 0
    }
}
{
    "value0": {
        "type()": "UpdateEnvironment",
        "changesVector": [
            {
                "state": 0,
                "position": {
                    "first": 50,
                    "second": 50
                },
                "id": 4
            }
        ]
    }
}
```

Listing 3.1 Przykłady trzech typów informacji, jakie serwer i klient mogą między sobą przekazywać, zaprezentowane w formacie JSON

Pierwsza z wiadomości (Odpowiedź) udziela prostej informacji, czy serwer zaakceptował ostatnią wysłaną do niego komendę od klienta aplikacji. Tylko serwer wysyła tę wiadomość. Drugi rodzaj wiadomości (Aktualizacja Gracza) jest wysyłany przez klienta aplikacji i zawiera w sobie informację jaką zmianę swojego stanu klient próbuje wykonać i przy pomocy jakiej akcji (na przykład ruch lub skok). Serwer musi tę wiadomość odpowiednio

przetworzyć, głównie pod kątem sprawdzenia czy postać gracza może taką akcję wykonać. W razie braku przeciwwskazań zostaje wysłana pozytywna Odpowiedź. Jeśli natomiast serwer z jakiegoś powodu nie może pozwolić na taką akcję, wysłana Odpowiedź jest negatywna.

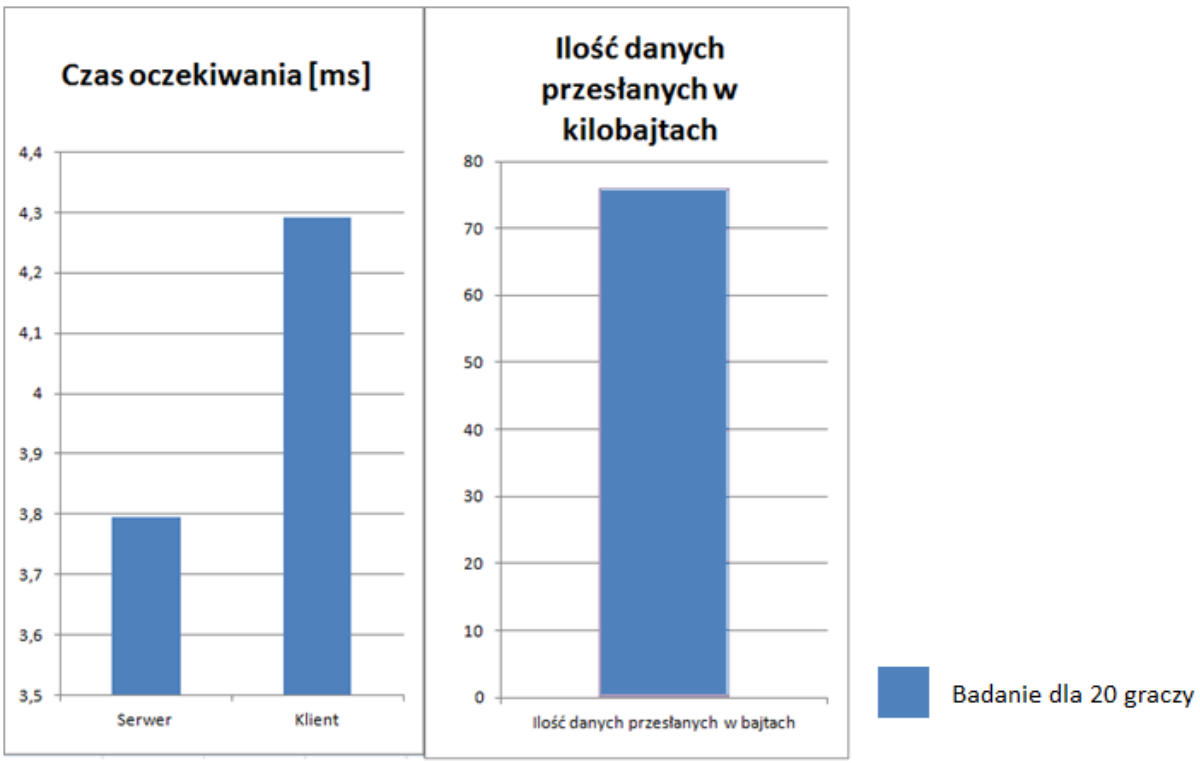
Ostatni rodzaj wiadomości (Aktualizacja Środowiska) jest znacząco inny od pozostałych, ponieważ zawiera w sobie listę danych, *changesVector*. Jest to informacja wysyłana przez serwer do klienta aplikacji, która ma go poinformować o zmianach jakie zaszły w stanach i położeniach innych graczy w pobliżu.



Rysunek 3.2 Przebieg komunikacji Klient - Server. Po lewej stronie: przebieg komunikacji od zalogowania się do wylogowania przez klienta. Po prawej stronie: aktualizacja klienta o dane innych klientów

Wiadomości te zostały przedstawione na rysunku 3.2, który prezentuje zaimplementowaną w symulatorze komunikację między klientem a serwerem gry. Przedstawiona po lewej stronie rysunku "Autoryzacja" i "Wylogowanie" były obsługiwane przez wiadomość Aktualizacja Gracza z odpowiednimi identyfikatorami akcji, które miały poinformować serwer o rodzaju aktualizacji. Kiedy następuje "Wylogowanie", transmisja zostaje po prostu przerwana - klient informuje w ten sposób o zerwaniu połączenia, a serwer dostosowuje się do tej akcji. Po lewej stronie rysunku widoczna jest jedyna możliwa sytuacja, kiedy serwer wysyła do klienta wiadomość bez jego wyraźnego zapytania. Klient nie potwierdza serwerowi w specjalny sposób faktu, że wiadomość dostał (inny niż standardowe potwierdzenie otrzymania pakietów TCP).

Dla wcześniej ustalonych podstawowych ustawień serwera gry przeprowadzono badanie z zebraniem danych pomiarowych. Wyniki zebrane na wykresie 3.1 zostały określone mianem wzorcowych i będą używane do porównywania z nimi wyników dalszych badań. Dalsze wykresy będą miały tę samą strukturę trzech pod-wykresów.



Wykres 3.1 Badanie wzorcowe dla podstawowych ustawień aplikacji przy formatowaniu JSON i współdzieleniu bazą danych

Wykres 3.1 składa się z trzech pod-wykresów: lewego, środkowego i prawego:

- czas oczekiwania (wykres lewy) prezentuje czas oczekiwania, w milisekundach, jaki aplikacja klienta spędziła na czekaniu na odpowiedź od serwera oraz czas jaki aplikacja serwera spędziła na przetwarzaniu danych zanim wysłała odpowiedź do klienta. Z różnicy tych wartości można policzyć także czas spędzony wyłącznie na ruchu sieciowym, kiedy aplikacja klienta już zaczęła czekać, ale aplikacja serwera jeszcze nie zaczęła swojej pracy
- ilość danych przesłanych w kilobajtach (wykres prawy) oznacza całkowitą ilość przesłanych informacji we wszystkich wiadomościach. Nie uwzględnia się tutaj rozmiaru nagłówków protokołu komunikacyjnego, jednak dzięki wiedzy o ilości akcji wykonanych przez klienta (a przez to: ilości odpowiedzi serwera) można to łatwo policzyć. Tak samo na podstawie ilości odpowiedzi można policzyć ile bajtów przypadało średnio na każdą wiadomość

4. Komunikacja między klientem a serwerem

Odpowiedni dobór protokołu komunikacyjnego jest jedną z najważniejszych decyzji, jaką musi podjąć programista przy projektowaniu gry sieciowej. Badania[1, 2] pokazują, że w zależności od konkretnej gry wybierane są najczęściej protokoły TCP, UDP lub ich mieszanki. Mieszanką tych protokołów nazywa się sytuację, kiedy jedna aplikacja ma aktywne połączenie obu protokołów, jednak zależnie od aktualnie wykonywanej akcji używany jest jeden lub drugi.

Tabela 4.1 Podstawowe protokoły komunikacyjne w grach sieciowych

Protokół	Kiedy używany[1]
TCP	Rozgrywka logiczna (gry turowe, strategie czasu rzeczywistego). Dostarczenie pakietu jest priorytetem.
UDP	Rozgrywka dynamiczna (gry akcji, zręcznościowe). Kolejność wysłania i odebrania pakietów jest priorytetem.
Mieszany	Rozgrywka nie ma jednoznacznie zdefiniowanego charakteru. Niektóre elementy mają charakter logiczny, a inne dynamiczny.

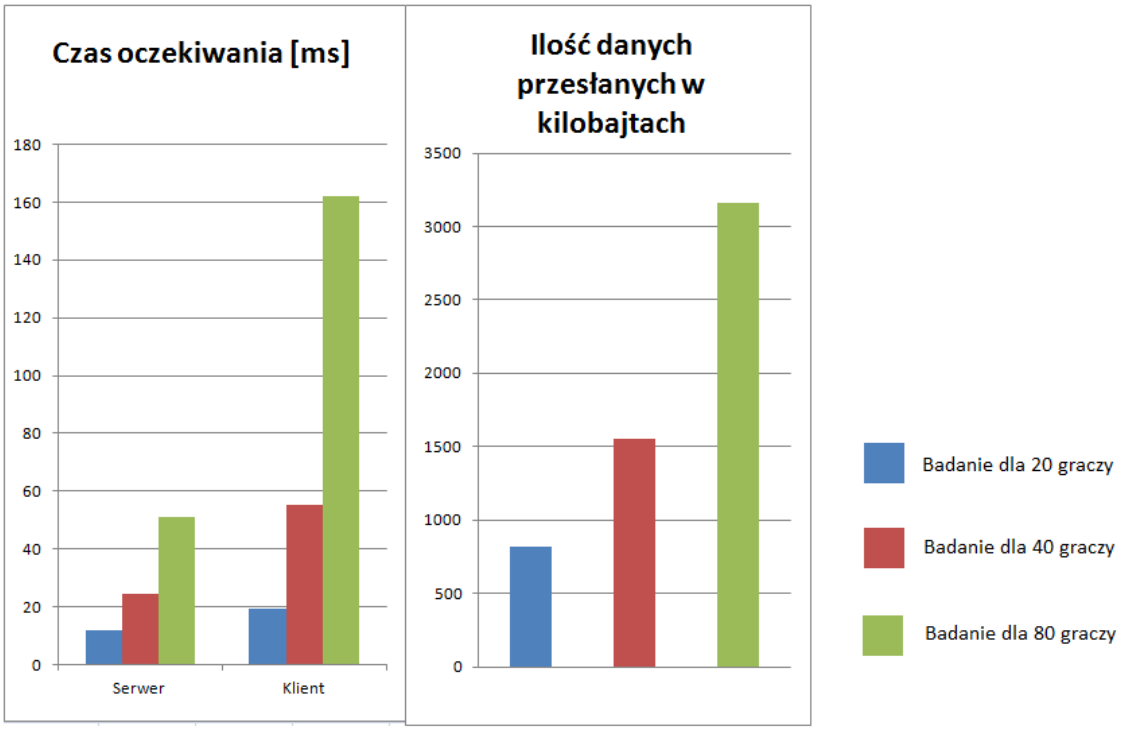
W rozważanym typie gier przygodowych nie jest oczywiste którego protokołu komunikacyjnego używać. Gry takie nie mają jednoznacznie zdefiniowanej dynamiki rozgrywki - mogą być równie dynamiczne co gry akcji lub nastawione na strategię jak gry turowe. W tej pracy zdecydowano się na użycie protokołu TCP, który zdaniem autora ułatwia mierzenie danych. Połączeniowa natura protokołu TCP wymusza na aplikacjach blokowanie na operacjach wysłania wiadomości dopóki druga strona ich nie odbierze. Wspomagało to mierzenie czasu komunikacji między aplikacjami, co było pożądaną cechą w tej pracy. Inną zaletą protokołu TCP jest jego niezawodność względem UDP. Znacznie mniej pakietów danych zostaje zgubionych w ruchu sieciowym, co w ściśle określonych środowiskach komunikacyjnych (jak przykład na rysunku 4.1) jest nieocenioną zaletą. Tracone pakiety są w przypadku tej pracy bardzo niepożądanym zjawiskiem, który bardzo zakłócałby wyniki

badań, ponieważ symulacje miały odbywać się w warunkach idealnych i jedyną mierzoną wartością były opóźnienia spowodowane obciążeniem sieci.

Rysunek 4.1 Komunikacja między klientem a serwerem w dwóch wariantach: poprawna (po lewej) i z utraconym pakietem aktualizacji (po prawej)

Zgodnie z wykresem 1.1 opóźnienia w komunikacji powyżej 200 ms nie są już akceptowalne przez graczy. Sytuacja pokazana na rysunku 4.1 z wariantem utraconego pakietu aktualizacji kosztowałaby aplikację jednorazowo bardzo duże opóźnienie, ponieważ klient gry aktywnie czeka na odpowiedź od serwera. Istnieją metody po stronie aplikacji klienta, które pozwalają radzić sobie z chwilowymi przestojami[2], jednak nie są one tematem tej pracy i najczęściej naprawiają jedynie symptomy problemu, a nie jego źródło.

Podstawowym problemem komunikacji sieciowej są opóźnienia wynikające ze zbyt dużego ruchu w sieci: zbyt dużo podmiotów wysyłających dane, zbyt dużo danych wysyłanych naraz przez jeden podmiot, lub oba te problemy naraz. Wykonane zostały badania mające na celu sprawdzić zarówno występowanie, jak i rozległość tego zjawiska w rozpatrywanej grze sieciowej. Przedstawiony na listingu 3.1 sposób serializowania danych formatem JSON został wykorzystany do przesyłania danych w opisany wcześniej sposób na rysunku 3.2.



Wykres 4.1 Badanie wpływu ilości graczy na wydajność serwera przy formatowaniu JSON i współdzieleniu bazą danych

Badania zostały przeprowadzone dla scenariuszy z różną ilością klientów: 20, 40 oraz 80. Wszyscy gracze podłączeni byli do wspólnego serwera. Badania przeprowadzono w warunkach domowych (na komputerze osobistym) i nie udało się ich przeprowadzić dla wariantu z setką graczy, ponieważ zanotowany został drastyczny spadek wydajności: zasoby sieciowe się zaczęły wyczerpywać i część klientów dawała przez to niemiarodajne wyniki. Z tego też powodu ilość osiemdziesięciu klientów podłączonych do serwera została uznana za wartość graniczną.

Wykres 4.1 prezentuje wyniki badań. Widoczne są znaczące zmiany w zachowaniu serwera w porównaniu do wykresu 3.1. Wykres lewy pokazuje wielomianowy wzrost czasu oczekiwania w stosunku do ilości podłączonych graczy. Zgodnie z wykresem 1.1 przekroczony został pierwszy graniczny czas oczekiwania w którym gracze mogą stracić zainteresowanie grą. W trzecim wariancie badania aplikacja klienta czeka na odpowiedź ponad 160 milisekund, co jest wartością wymagającą poprawy. Z wykresu można wyczytać, że większość tego czasu, bo aż 110 milisekund, zostało wykorzystane na sam ruch sieciowy. Wykres prawy pokazuje jak ogromne ilości danych były przesyłane w trakcie sesji. Wiedząc, że w trakcie sesji każdy klient wykonał 200 akcji, łatwo można policzyć, że każda akcja w wariancie z osiemdziesięcioma klientami niosła za sobą ponad 15 kilobajtów danych przesłanych protokołem TCP. Szybka analiza informacji przekazywanych między klientem a

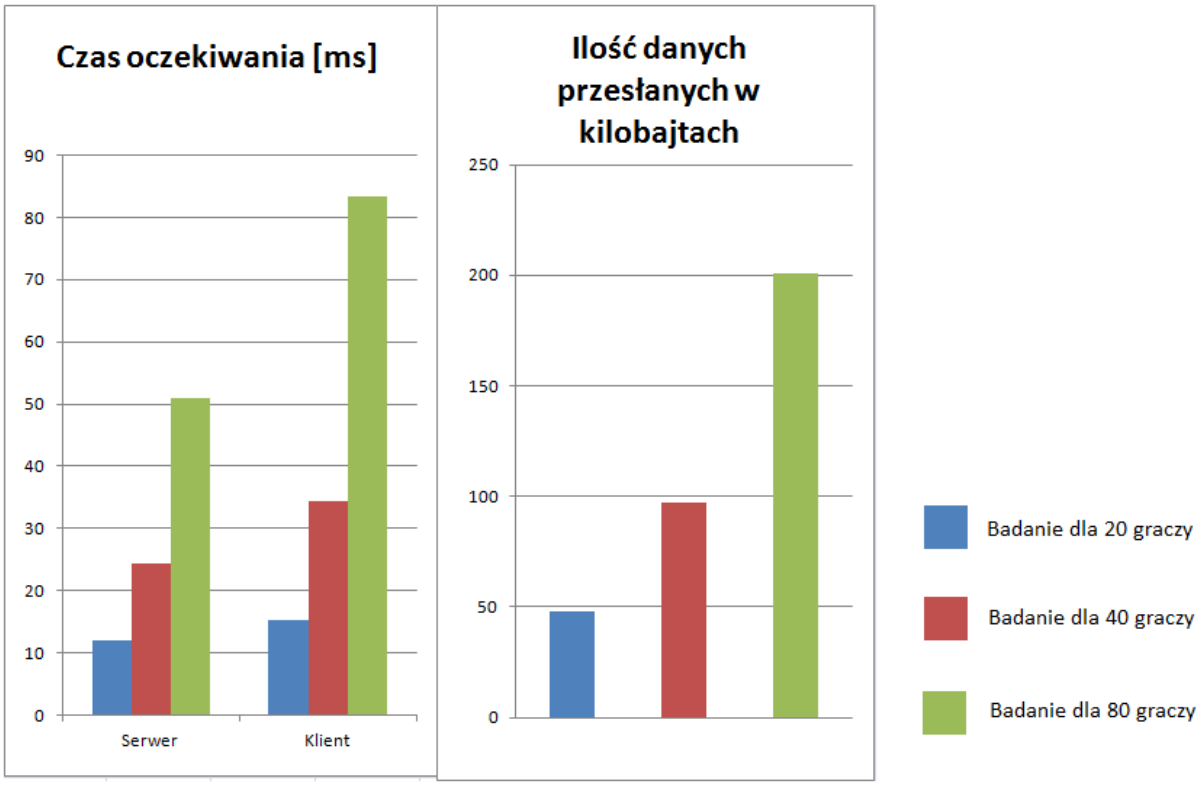
serwerem pokazała, że znaczną część z tego stanowiły wiadomości Aktualizacji Środowiska.

Aby poprawić wyniki badania, zdecydowano się zmienić serializowanie danych formatem JSON na formatowanie binarne. Zabieg ten miał na celu zarówno zmniejszenie informacji przekazywanych, jak i ograniczenie czasu spędzonego przez klienta aplikacji na czekaniu na odpowiedź serwera. Konieczne było zdefiniowanie własnego protokołu serializującego dane, który minimalizowałby ich ilość, jednocześnie przesyłając tę samą informację, co format JSON.

```
0'1;
1'5'5'0;
2'"0'50'50'4;
```

Listing 4.1 Przykłady trzech typów informacji o treści identycznej jak w listingu 3.1, lecz formatowane binarnie

W aplikacji potrzebne były tylko specyficzne rodzaje informacji do przekazywania na linii klient - serwer. Dzięki temu powstał protokół serializowania binarnego zaprezentowany na przykładzie w listingu 4.1. Przenosi on dokładnie te same informacje, co listing 3.1, jednak w znacznie bardziej skompresowanej formie. Każda wiadomość na początku ma jeden znak mający zidentyfikować rodzaj wiadomości. Następnie następuje apostrof ', który oznacza opisywanie wartości kolejnej zmiennej. Znak średnika **;** oznacza koniec aktualnej wiadomości, zaś znak cudzysłowu " oznacza dla wiadomości Aktualizacja Środowiska początek kolejnej pozycji w wektorze.

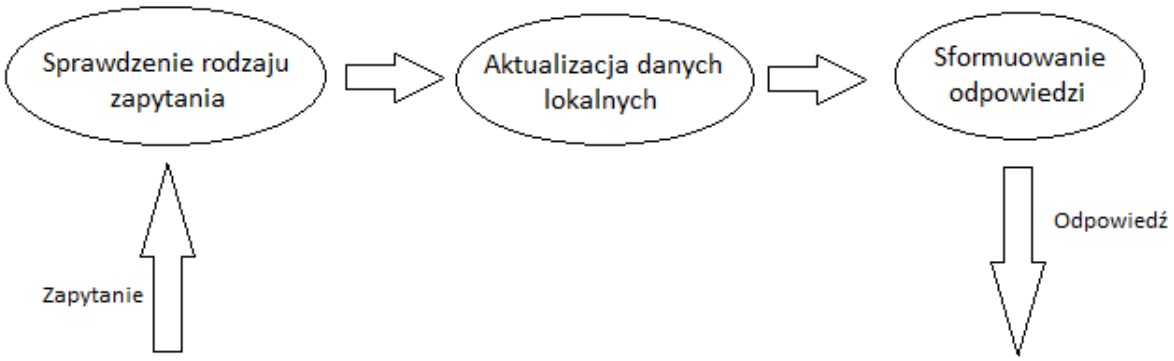


Wykres 4.2 Badanie wpływu ilości graczy na wydajność serwera przy formatowaniu binarnym i współdzieleniu bazą danych

Badania, których wyniki zebrano na wykresie 4.2, przeprowadzono dla takich samych trzech scenariuszy, co badania z wykresu 4.1. Z eksperymentów, w których użyto formatowania binarnego jasno wynika, że zmniejszenie ilości danych przesyłanych za każdą akcją klienta bezpośrednio wpływa na czas przesyłania danych w sieci. Ilość bajtów na każdą akcję spadła do około jednego kilobajta, zaś czas spędzony na ruchu sieciowym zmalał do około 30 milisekund (dla wariantu z 80 graczami). Istotną obserwacją jest również fakt, że czas przetwarzania danych przez serwer nie uległ prawie żadnej zmianie. Wynika to z faktu, że pomiędzy tymi badaniami nie były przeprowadzane żadne zmiany mające odciążyć serwer - zmienił się jedynie format wiadomości przesyłanych protokołem TCP.

5. Przetwarzanie danych przez serwer

W grach sieciowych serwer jest punktem centralnym, przez który przechodzą wszystkie dane. Musi on odbierać zapytania od klientów, przetwarzać je zgodnie z zaprogramowaną logiką i wysłać odpowiedź do tego samego źródła. Musi także posiadać bazę danych z którą na bieżąco się synchronizuje, aby zawsze mieć aktualne dane na temat wszystkich obiektów w grze[7].



Rysunek 5.1 Pojedyncza iteracja instancji serwera od otrzymania zapytania do wysłania odpowiedzi

Pojedyncza instancja serwera zawsze jest odpowiedzialna tylko za jednego gracza i jego dane w bazie danych. Rysunek 5.1 przedstawia podstawowe akcje, które taki serwer podejmuje, kiedy otrzymuje zapytanie od klienta. Najważniejszym elementem tego schematu jest "Aktualizacja danych lokalnych", ponieważ w ten sposób instancja serwera komunikuje się z pozostałymi serwerami. Dane, które muszą tutaj zostać aktualizowane to:

- informacje o jego własnym kliencie - w tym wypadku to instancja serwera wysyła dane do bazy danych, informując ją o zmianach u klienta gry
- informacje o innych obiektach - instancja serwera pobiera takie dane i porównuje je z ostatnimi własnymi informacjami o nich, tworząc w ten sposób różnicę w informacjach, które następnie musi wysłać do klienta przy najbliższej okazji

Z punktu widzenia wielożytkowości gry istotnym zagadnieniem jest jaki wpływ na wydajność ma ilość podłączonych klientów. Połączenie z bazą danych musi być samo w sobie odpowiednio synchronizowane, w przeciwnym wypadku powstaną nieprzewidywalne efekty, takie jak wyścigi o dane. Synchronizacja w aplikacji serwera została zapewniona przez

muteks, który jest współdzielony przez wszystkie instancje serwera w ramach tej samej aplikacji. Za sekcję krytyczną zostało uznane każdorazowe odnoszenie się do bazy danych w celu aktualizowania danych lub pobrania nowego ich segmentu.

W podstawowych konfiguracjach, baza danych była wykorzystywana w sposób ciągły. Każda instancja serwera pobierała z niej dane za każdym razem, kiedy musiała sprawdzić, czy klient tego serwera nie ma przedawnionych informacji o innych obiektach w grze. Rozwiązanie takie zostało zastosowane ze względu na swoją niezawodność. Każda instancja serwera musiała synchronizować w ten sposób dane o swoich klientach, a dzięki wykorzystaniu muteksu było zapewnione, że dostęp do sekcji krytycznej zaoferuje najnowsze dane z bazy danych. Wydajność serwera przy takim wykorzystaniu bazy danych można zaobserwować na wykresie 4.2. Czas przetwarzania danych przez serwer, jaki został tam zmierzony bezpośrednio opisuje zmieniające się zachowanie aplikacji w odnoszeniu się do sekcji krytycznej. Zaobserwowane zmiany mają charakter wielomianowy, co oznacza, że dla scenariusza badań z 160 graczami, sam czas mierzenia po stronie serwera wynosiłby około 110 - 120 milisekund. W takim wypadku czas oczekiwania na odpowiedź po stronie klienta przekroczyłaby wartość progową w taki sam sposób, jak zostało to opisane przy wynikach badań wykresu 4.1.

Zaproponowanym rozwiązaniem w tej kwestii było częściowe zastąpienie bazy danych inną metodą współdzielenia danych. W tym celu wykorzystano fakt, że wszystkie instancje serwera wywodzą się od wspólnego serwera (rysunek 2.1) i istnieją jako wątki wywodzące się od niego. Dzięki temu możliwe było stworzenie swoistego bloku pamięci dzielonej pomiędzy tymi wątkami, który zawierałby wszystkie niezbędne dane, które dotąd musiały być dostarczane przez bazę danych, czyli informacje o podłączonych graczach. Nowa metoda wciąż zakłada wykorzystywanie bazy danych, ale w znacznie mniejszym stopniu. Aplikacja serwera przy uruchomieniu tworzy współdzielony blok i przekazuje go jako wskaźnik do każdej instancji serwera, która zostanie stworzona. Sam segment pamięci został zaimplementowany jako następujący typ zmiennej w języku C++[9]:

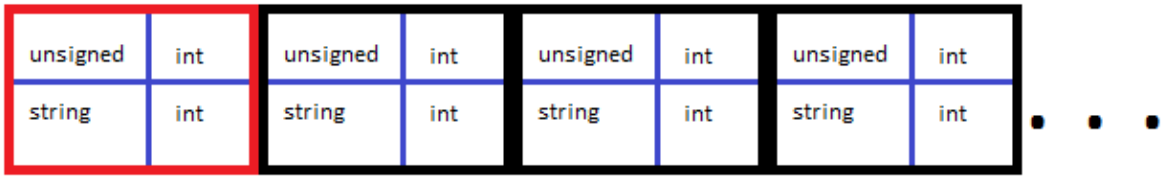
```
std::shared_ptr<std::vector<std::tuple<unsigned, std::string, int, int > > >
```

Zaprezentowany typ składa się z wielokrotnie zagnieżdżonego szablonowania. Jego celem jest przechowywanie danych w taki sposób, aby jednocześnie było wygodne przekazywanie go w programie głównym do wątków podrzędnych, a także żeby przechowywał w sobie wszystkie niezbędne dane.

Zdefiniowany typ zmiennej mocno opiera się o standardową przestrzeń nazw, która w języku C++ przechowuje bardzo dużo przydatnych rodzajów kontenerów i algorytmów[9]. Poszczególnymi składowymi tego typu zmiennej są:

- inteligentny wskaźnik (ang. *smart pointer*) nazwany tutaj *shared_ptr*. Jego główne zadanie jest takie samo jak standardowego wskaźnika w języku C++, czyli przechowywanie adresu na zmienną. Inteligentny wskaźnik ma ponadto mechanizm automatycznego zwalniania swojej pamięci w momencie kiedy ostatni odnośnik do jego obiektu zostanie zdjęty. Oznacza to, że w przypadku dzierżenia wielu wskaźników na jeden obiekt, ostatni wskaźnik który zostanie usunięty (lub wskazany na inny obiekt) jednocześnie usuwa wskazywany obiekt. Zyskuje się dzięki temu pewność, że pamięć zajmowana przez obiekt zostanie zwolniona kiedy już nie będzie potrzebna (ale nie wcześniej)[9].
- wektor zmiennych (ang. *vector*) jest sekwencyjnym kontenerem na zmienne określonego rodzaju. Wektor jest rozszerzeniem dynamicznie alokowanej tablicy, co oznacza że elementy w nim mogą zostać wywołane za pomocą odpowiedniego indeksu. Operacje na ostatnim elemencie wektora (lub wpisanie nowego elementu na jego koniec) jest bardzo szybką operacją[9]. Jedyne niebezpieczeństwo powolnej operacji pojawia się kiedy wpisanie nowego elementu wymagałoby realokacji całego wektora z powodu zbyt małej ilości pamięci w obecnym wektorze. Problem ten jednak można rozwiązać rezerwując zawczasu odpowiednią ilość pamięci na rzecz wektora.
- krotka (ang. *tuple*) jest zbiorem niejednorodnych zmiennych o stałej wielkości. Jest rozszerzeniem standardowego typu pary obiektów. Pozwala przechowywać dowolną ilość zmiennych o różnych rodzajach. Swoją popularność zawdzięcza wprowadzeniu szablonów o dowolnej liczbie parametrów (ang. *variadic template*), które pojawiły się w C++11. Tak jak w przypadku wektorów, do elementów krotki można odwoływać się poprzez indeks elementu.

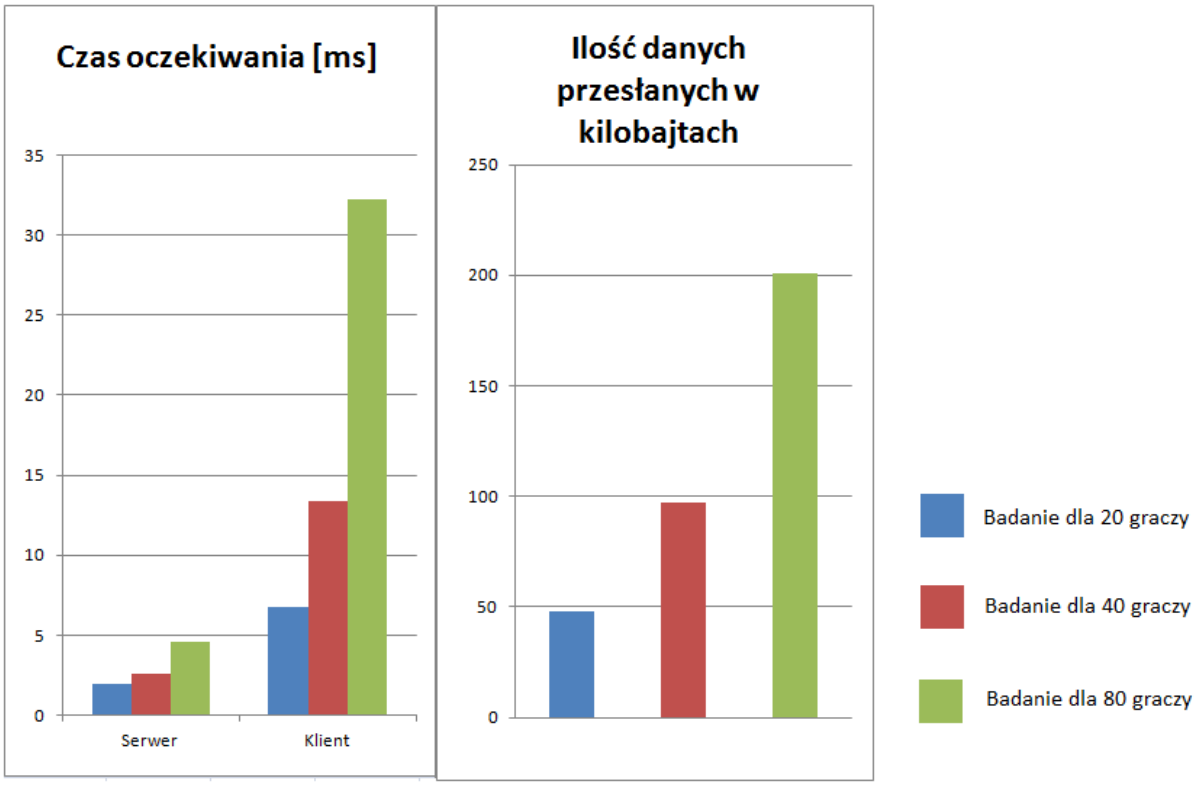
Aby lepiej zrozumieć omawiany typ zmiennej, przedstawiono go na rysunku 5.2. Wskaźnikiem w tym opisie jest pierwszy element wektora, wyróżniony czerwonym kolorem. Wszystkie pogrubione ramki są krotkami, czyli elementami wektora. Ze schematu jasno wynika, że zmienną jest wskaźnik na wektor krotek z których każda przechowuje typy podstawowe: liczbę całkowitą bez znaku, łańcuch znaków oraz dwie liczby całkowite ze znakiem. Te cztery ostatnie zmienne (podstawowe typy danych) miały reprezentować podstawowe informacje o postaci gracza tak, jak robiła to baza danych: identyfikator postaci, jej imię, oraz dwie składowe współrzędnych dwuwymiarowych: *x* i *y*.



Rysunek 5.2 Wizualizacja zaproponowanego typu danych.

Taki rodzaj zmiennej jest tworzony w programie głównym serwera jako pusty wektor o odpowiednio dużej zarezerwowanej pamięci na taką ilość krotek, ilu klientów przewiduje scenariusz. Następnie wskaźnik na ten obiekt jest przekazywany do każdej instancji serwera, która nawiązuje połączenie z nowym klientem. Instancja wtedy pobiera jednorazowo z bazy danych informacje o użytkowniku, którego aktualnie opisuje i wpisuje jego dane do wektora, zaś przy zrywaniu połączenia - usuwa ten wpis. W ten sposób na przestrzeni badania wektor jest wypełniony informacjami tylko o tych użytkownikach, którzy są aktualnie podłączeni. Przy każdym ruchu wykonywanym przez gracza, instancja serwera na bieżąco aktualizuje jego wpis w wektorze. Przy Aktualizacji Środowiska, pobierane są dane o wszystkich innych graczach z wektora i porównywane są z lokalną kopią wektora. W opisanych wcześniej badaniach wszystkie te dane są pobierane z bazy danych.

Przeprowadzone zostało badanie serwera z nowymi ustawieniami współdzielenia danych. Wyniki badania są widoczne na wykresie 5.1. Tak jak w poprzednich badaniach, symulacje przeprowadzono w scenariuszach z 20, 40 oraz 80 graczami. Jak wynika z wykresu, czas potrzebny na operacje na serwerze zmniejszył się drastycznie, tak samo jak czas oczekiwania na odpowiedź po stronie klienta. Mimo tak znacznej poprawy należy pamiętać, że największym scenariuszem badań, jakie są prowadzone na symulacjach to 80 graczy naraz. Z punktu widzenia dużych sieciowych gier przygodowych nie jest to duża liczba, gdyż takie gry w ramach jednej sesji są w stanie gromadzić nawet tysiące graczy. Trendy, które można wyczytać na wykresach z wynikami badań, wskazują, że dla takich scenariuszy wyniki będą katastrofalne niezależnie od przyjętych strategii optymalizacyjnych dla pojedynczego serwera.

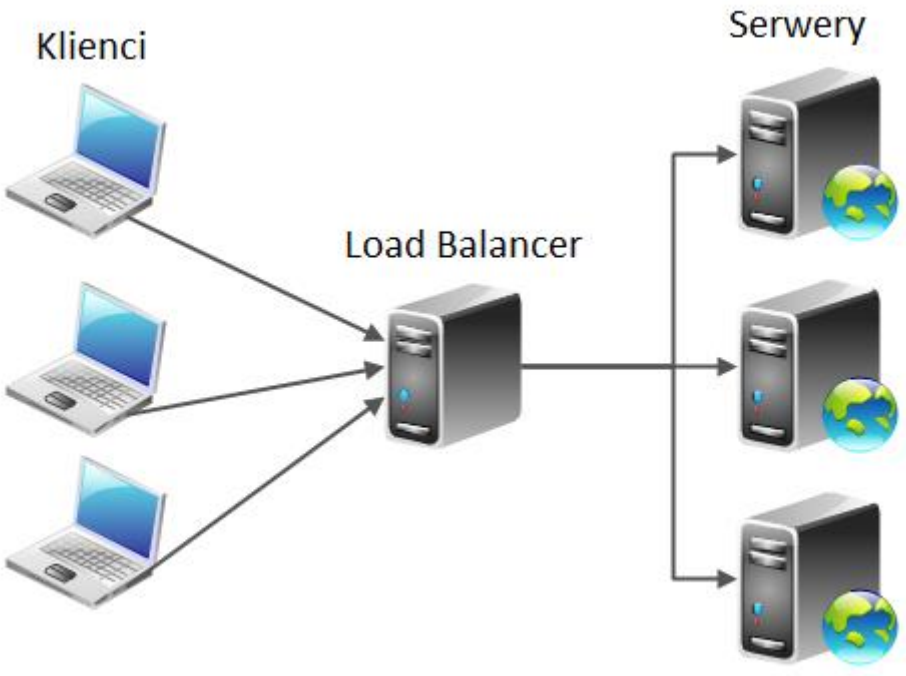


Wykres 5.1 Badanie wpływu ilości graczy na wydajność serwera przy formatowaniu binarnym i współdzieleniu segmentem pamięci

6. Podział serwera na mniejsze jednostki

W dotychczasowych badaniach rozważany był wariant jednego centralnego serwera, który przyjmował kolejne połączenia od klientów i komunikował się z nimi w zaprogramowany sposób. Jak zostało wykazane w poprzednim rozdziale, takie podejście nie jest efektywne w rozważanej pracy. Aby efektywnie zarządzać znaczną ilością klientów, serwer gry sieciowej musi być w jakiś sposób rozproszony. Oznacza to, że musi istnieć pewna logika dystrybucji pracy pomiędzy wiele fizycznych serwerów tak, aby zapewnić że każdy z nich ma równe obłożenie ilości połączeń do klientów.

Techniki takie nazywane są równoważeniem obciążenia (ang. *load balancing*) i są popularnym rozwiązaniem przy wszelkiego rodzaju serwerach sieciowych[11]. Szczególną popularnością cieszą się rozwiązania dla bezstanowych serwerów HTTP, które obsługują strony internetowe, gdyż algorytmy równoważenia obciążenia dla takich serwerów są stosunkowo proste. Serwer, który realizuje zadanie równoważenia obciążenia nazywa się Load Balancerem.



Rysunek 6.1 Wizualizacja działania równoważenia obciążenia dla serwerów sieciowych

W praktyce Load Balancer jest dodatkowym serwerem dla usługi sieciowej, którego głównym zadaniem jest odbieranie połączeń od klientów i przekierowywanie ich do faktycznego serwera, który obsłuży połączenie. Logika ta została zaprezentowana na rysunku 6.1.

Najpopularniejszymi algorytmami do równoważenia obciążenia są[11]:

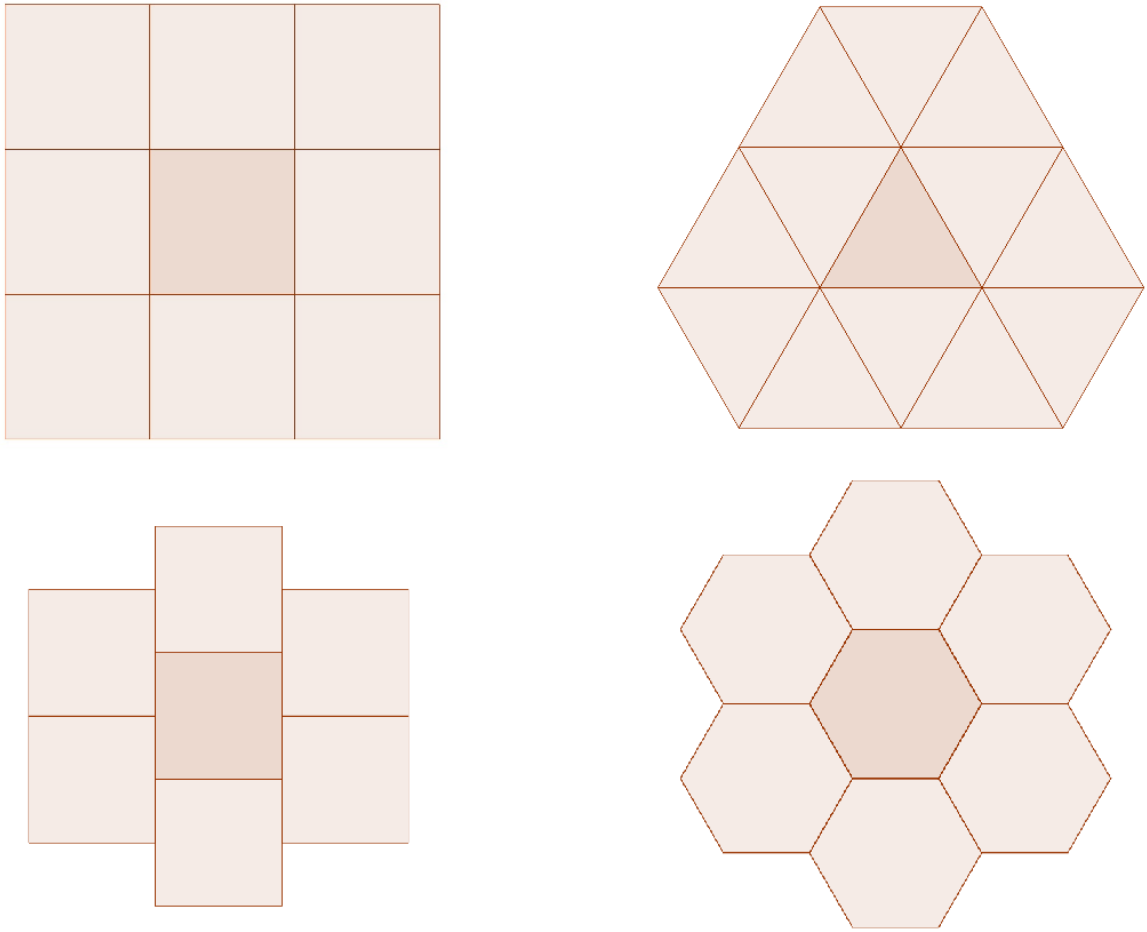
- algorytm karuzelowy (ang. *round-robin*) - każdy serwer po kolei otrzymuje nowe połączenia, tak aby każdy z nich miał ich średnio tyle samo. Algorytm ten nie uwzględnia priorytetów połączeń, ani faktu, że niektóre połączenia mogą trwać krócej od innych, wobec czego po pewnym czasie obciążenie może być bardzo nierównomiernie rozdysponowane.
- najmniejsze obłożenie - Load Balancer przechowuje w sobie informację o połączeniach do każdego serwera i priorytetach każdego z nich. Na tej podstawie jest w stanie obliczyć, który serwer ma aktualnie najmniejsze obłożenie i do niego przekazywane jest nowe połączenie.

Jak zostało wcześniej napisane, powyższe rozwiązania są popularne dla bezstanowych serwerów HTTP, jednak w rozważanej pracy opisywane są serwery gier sieciowych, których głównym celem jest przechowywanie danych o aktualnym kliencie i przetwarzanie zapytań w oparciu o te dane.

Gry sieciowe oparte o pewną wirtualną przestrzeń, na której poruszają się gracze, muszą udostępniać swoim klientom informacje o zmieniających się obiektach w tej przestrzeni. To, czym są takie obiekty, zależy od konkretnej implementacji gry sieciowej. W jednej z najpopularniejszych gier sieciowych tego samego rodzaju co omawiana w tej pracy, *World of Warcraft*, lista rodzajów obiektów w grze, z którymi można wejść w interakcję (a zatem są o zmiennym stanie) sięga już blisko połowy miliona pozycji[12]. W badaniach optymalizacyjnych, które zostały przeprowadzone w poprzednich rozdziałach tej pracy, zostało wykazane, że udostępnianie klientowi wszystkich informacji o wszystkich innych obiektach (w tym wypadku: innych graczach) nie jest efektywne. Na wykresach z wynikami badań (na przykład wykresie 5.1) można wyczytać, że ilość danych przesyłanych w zależności od ilości innych obiektów zwiększa się liniowo. Oznacza to, że równoważenie obciążenia w przypadku wielożytkowej gry sieciowej musi przede wszystkim polegać na ograniczeniu ilości danych, które są przesyłane do klienta.

W rozdziale 2. niniejszej pracy został opisany pokrótce system, który realizuje takie ograniczenie. System Zarządzania Zainteresowaniem, który został tam przedstawiony, zakłada podział danych od klientów na obszary w zależności od ich położenia oraz pola zainteresowania. Jednakże w takiej postaci SZZ nie definiuje w jaki sposób rozdzielić klientów pomiędzy rozproszone serwery przy pomocy Load Balancera. W tym celu w grach sieciowych wprowadza się segmentację wirtualnego świata gry na mniejsze fragmenty.

Ideą takiego rozwiązania jest ograniczenie kompetencji każdego serwera do konkretnej części świata gry tak, aby miał w sobie dane tylko o graczach i obiektach, które również znajdują się w tym ograniczonym obszarze gry. Mapę można podzielić na dowolne figury geometryczne, jednak najczęściej stosuje się w tym celu proste wielokąty foremne[2], zaprezentowane na rysunku 6.2



Rysunek 6.2 Różne możliwości segmentacji mapy gry sieciowej[2]

Powodem dla którego preferowane są tego typu proste obiekty jest fakt, iż w tym wariancie wszystkie obiekty mają takie same wymiary. Można zatem obliczyć zakres segmentu mapy dowolnego serwera na podstawie minimalnych danych o jego położeniu (na przykład: środek jego segmentu mapy) i ogólnie dostępnych informacjach o aktualnie używanej metodzie segmentacji.

Przedstawione na rysunku 6.2 figury, które zostały wybrane jako możliwe do segmentacji mapy to:

- kwadrat o ośmiu sąsiadach
- trójkąt równoboczny
- kwadrat o sześciu sąsiadach
- sześciokąt foremny

Figury te zostały wybrane ze względu na dwie charakterystyczne cechy: są wielokątami foremnymi i można całkowicie wypełnić przestrzeń dwuwymiarową używając tylko tego typu obiektów. Na rysunku wyraźnie zaznaczono środkowy segment mapy, wraz z jego najbliższymi sąsiadami. Ilość sąsiadów danego serwera jest istotna dla omawianego rozwiązania. Wcześniej opisany został SZZ, który opiera się o traktowanie gracza na mapie jako okręgu o promieniu równemu jego polu zainteresowania. Oznacza to, że serwer musi być w stanie dostarczyć klientowi wszystkie informacje, które znajdują się wewnątrz tego pola zainteresowania, nawet jeśli nie należą one do segmentu mapy aktualnego serwera. Z tego powodu serwer musi posiadać informacje nie tylko o swoim segmencie mapy, ale także o wszystkich swoich bezpośrednich sąsiadach.

Porównywanie między sobą poszczególnych metod segmentacji mapy polega na sprawdzeniu jak często gracz poruszający się po takiej mapie przekroczy swoim polem zainteresowania granicę mapy. Akcja taka jest sygnałem dla serwera, że sam gracz może w niedalekiej przyszłości przekroczyć tę granicę i wejść na teren należący do innego serwera. Akcja taka nazywana jest migracją gracza. Serwer musi wtedy przekazać swoje połączenie z graczem sąsiedniemu serwerowi na terenie którego znajduje się teraz postać gracza.

Istnieją pewne stałe zależności o których trzeba pamiętać przy porównywaniu segmentów:

- rozmiary wielokątów muszą być tak dobrane, aby wszystkie wielokąty miały to samo pole powierzchni. Jest to warunek konieczny do zapewnienia, że przy różnych rodzajach segmentów serwer będzie miał wciąż taki sam fragment mapy pod swoją odpowiedzialnością.
- pole zainteresowania gracza powinno być znacznie mniejsze od długości krawędzi wielokąta. Przy pewnych granicznych wartościach (na przykład: promień pola równy połowie długości krawędzi kwadratu) rozważania nad ilością sąsiadów przestają mieć sens, ponieważ gracz zawsze miałby widocznego sąsiada, nawet gdyby stał w środku segmentu mapy.

W poprzednich badaniach mierzony był jedynie czas odpowiedzi serwera dla klienta gry sieciowej oraz ilość przesyłanych danych w bajtach. Konkretne ruchy graczy nie były istotne w tamtych badaniach, zaś zdefiniowane w tej pracy środowisko zostało tak opracowane, że gracze poruszają się idealnie losowo. Zaaplikowanie takiego sposobu badania do aktualnie poruszanego problemu zaowocowałoby niemiarodajnymi wynikami, ponieważ losowa natura poruszania się graczy nie jest w stanie odwzorować spodziewanych zachowań. Wykonywanie badań na symulacjach w tym wypadku ma sens tylko dla konkretnych danych poruszania się faktycznych graczy w grze sieciowej, jak zostało to opisane przy rysunku 3.1.

Z powodu braku takich danych, postanowiono przedstawić idealny rozkład graczy znajdujących się na segmencie mapy. Teoretyczna sytuacja, jaka jest tutaj omawiana, oznacza że gracz ma takie samo prawdopodobieństwo znajdowania się w każdym miejscu na mapie. Wiadomo również, że różne pola na segmencie mapy dają graczowi informacje o różnych sąsiadach serwera. Wobec powyższego można stworzyć model geometryczny, który opisuje ilu sąsiadów miałby gracz znajdujący się w konkretnym fragmencie mapy. W ten sposób zostaną opisane wszystkie cztery typy segmentów mapy przedstawione na rysunku 6.2.

Pierwszym rodzajem segmentu jest kwadrat z ośmioma sąsiadami. Rysunek 6.3 przedstawia tę figurę wraz z informacjami w których fragmentach obiektu ilu sąsiadów miałby gracz. Liczbę takich aktywnych sąsiadów zaznaczono na schemacie czerwonymi cyframi. Przykładowo kwadrat wewnętrzny ograniczony wierzchołkami *MNOP* oznacza pole, gdzie gracz nie ma żadnych aktywnych sąsiadów. Jak zostało wcześniej napisane, ilość aktywnych sąsiadów zależy nie tylko od wymiarów samego segmentu mapy, ale także od pola zainteresowania gracza. Na rysunku 6.3 widać to pole jako dowolny odcinek ograniczający promień ćwiartki koła w rogu kwadratu, na przykład odcinek *AE*.

Na podstawie danych z rysunku 6.3 można obliczyć ilu średnio aktywnych sąsiadów ma gracz, który znajdowałby się na takiej mapie. W tym celu należy wyliczyć pola wszystkich obiektów, które składają się na kwadrat *ABCD* i przemnożyć każdy z nich przez ilość aktywnych sąsiadów, które reprezentuje ten segment. Równanie 6.1 przedstawia ogólny wzór na obliczenie średniej ilości sąsiadów. Należy zwrócić uwagę na fakt, że w obliczaniu pól powierzchni poszczególnych segmentów jest pomijany segment bez żadnych aktywnych sąsiadów, gdyż nie dodaje on żadnej wartości do liczonej średniej.

$S - średnia\ ilość\ aktywnych\ sąsiadów$

$r - promień\ zainteresowania$

$a - krawędź\ wielokąta$

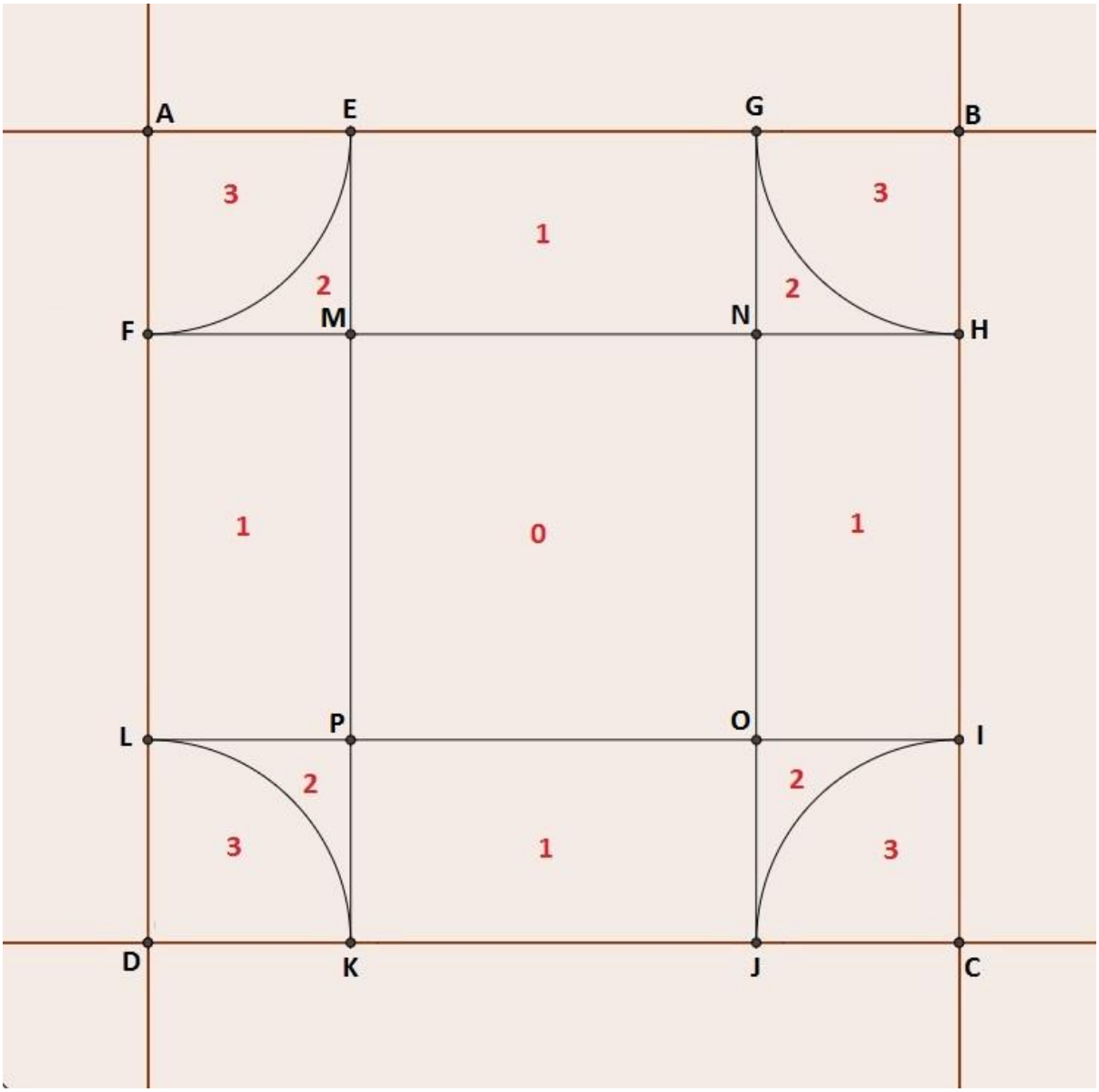
$n - maksymalna\ ilość\ sąsiadów\ w\ danym\ segmencie$

$x_i - ilość\ segmentów\ o\ i - aktywnych\ sąsiadów$

$P_i - pole\ pojedynczego\ segmentu\ i$

$P_s - pole\ całego\ rozważanego\ segmentu$

$$S = \frac{\sum_{i=1}^{n} P_i \cdot i \cdot x_i}{P_s} \qquad (6.1)$$



Rysunek 6.3 Kwadrat z ośmioma sąsiadami podzielony na segmenty ilości aktywnych sąsiadów

Wykorzystując równanie 6.1 i schemat z rysunku 6.3 można policzyć średnią ilość aktywnych sąsiadów dla kwadratu z ośmioma sąsiadami w zależności od promienia zainteresowania r oraz krawędzi kwadratu a. Poszczególne odcinki ze schematu rysunku 6.3, które są potrzebne do obliczenia pól powierzchni dane są następująco:

$$|AB| = |BC| = |CD| = |AD| = a$$
$$|AE| = |AF| = |BG| = |BH| = |CI| = |CJ| = |DK| = |DL| = r$$
$$|EG| = |MN| = |IH| = |NO| = |JK| = |OP| = |FL| = |MP| = a - 2r$$

Poniższa tabela 6.1 opisuje poszczególne pola powierzchni, które są potrzebne do obliczenia.

Tabela 6.1 Opis fragmentów pól powierzchni dla kwadratu o ośmiu sąsiadach

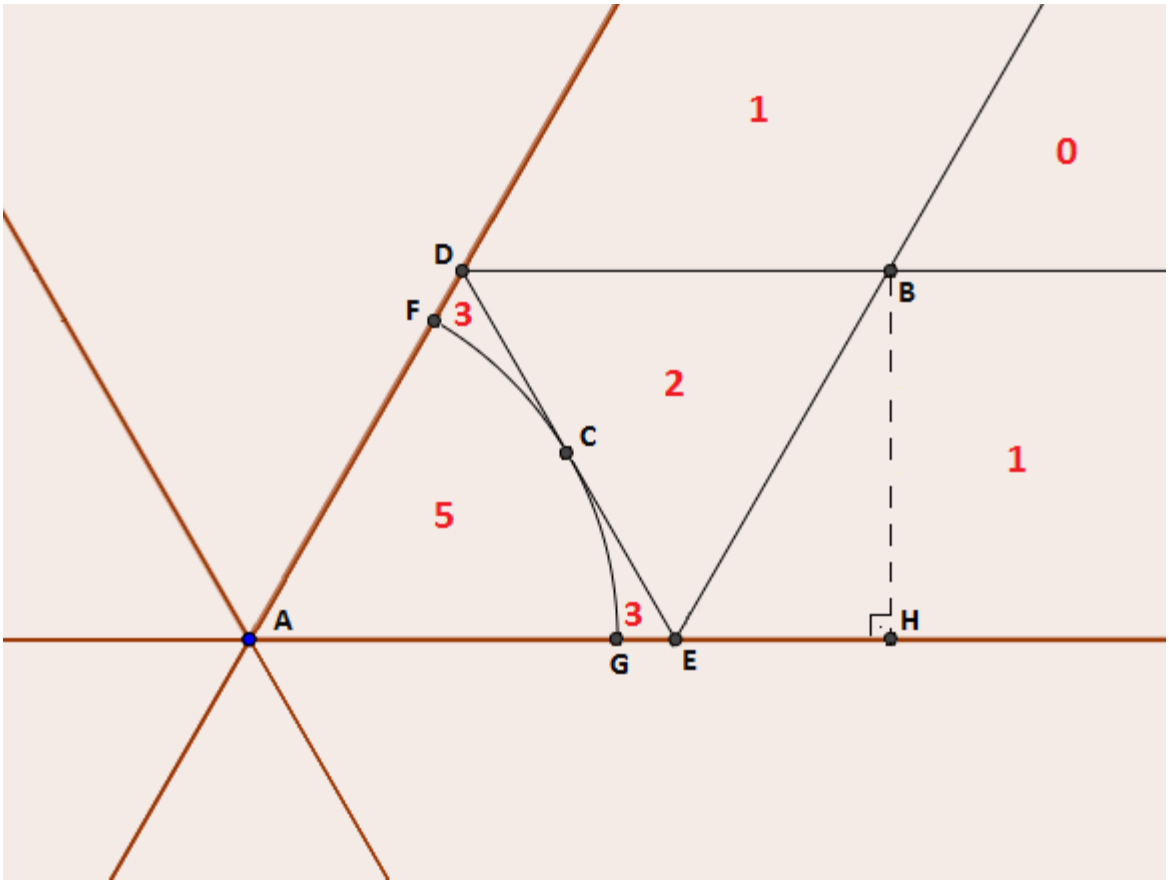
Oznaczenie	Opis figury	Przykład występowania na rysunku 7.3 [figura ograniczona punktami]
P_1	Prostokąt o bokach równych *r* oraz *a - 2r*	EGMN
P_2	Różnica między polem kwadratu o boku *r* i czwartą częścią koła o promieniu *r*	EFM
P_3	Czwarta część koła o promieniu *r*	AEF
P_s	Kwadrat o boku *a*	Cała figura

$$n = 3$$
$$x = \{4, 4, 4\}$$
$$P_s = a^2$$
$$P_1 = (\mathrm{a} - 2\mathrm{r})\mathrm{r} = \mathrm{ar} - 2\mathrm{r}^2$$
$$P_3 = \frac{1}{4}\pi r^2$$
$$P_2 = r^2 - P_3 = \left(1 - \frac{1}{4}\pi\right) r^2$$
$$S_1 = \frac{\sum_{i=1}^{n} P_i \cdot i \cdot x_i}{P_s}$$
$$S_1 = \frac{(P_1 \cdot 1 \cdot 4 + P_2 \cdot 2 \cdot 4 + P_3 \cdot 3 \cdot 4)}{a^2}$$
$$S_1 = \frac{4(ar - 2r^2) + 8\left(1 - \frac{1}{4}\pi\right)\mathrm{r}^2 + 12 \cdot \frac{1}{4}\pi\mathrm{r}^2}{a^2}$$

$$S_1 = \frac{4r}{a} + \frac{\pi r^2}{a^2} \qquad (6.2)$$

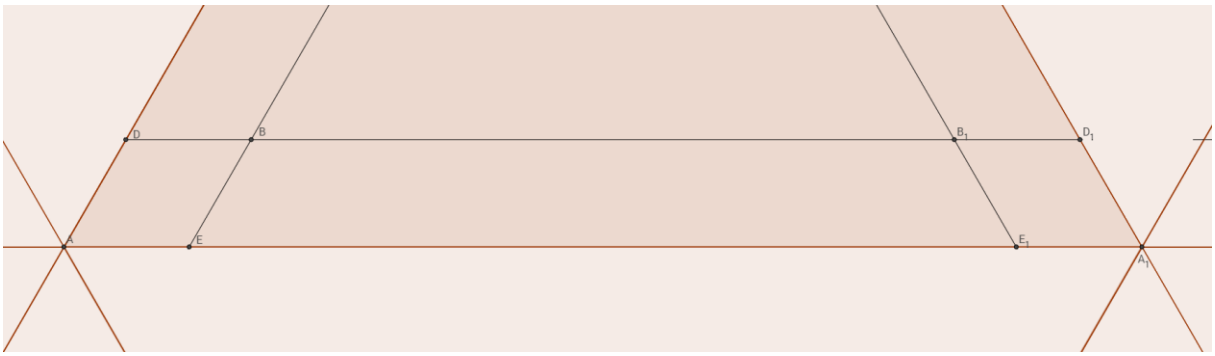
Każde z pól P_i opisuje pole pojedynczego segmentu z *i* aktywnymi sąsiadami. Wszystkie wystąpienia pól P_i na rysunku 6.3 mają takie same wymiary, dzięki czemu wystarczające jest obliczenie pola jednego z nich i przemnożenie przez ich ilość oraz przez ilość aktywnych sąsiadów na którą wskazują.

Drugim rozważanym rodzajem segmentu jest trójkąt równoboczny, którego każdy wierzchołek ma pięciu sąsiadów. Rysunek 6.4 przedstawia występowanie aktywnych sąsiadów w analogiczny sposób jak rysunek 6.3, jednak w przypadku trójkąta skupiono się tylko na jednym z jego wierzchołków. Pozostałe dwa wierzchołki mają taki sam rozkład aktywnych sąsiadów, a dzięki skupieniu się tylko na jednym z nich, widać wyraźniej zależności pól, które należy policzyć od zmiennych *r* i *a*. W tym wypadku również korzysta się ze wzoru 6.1 z uwzględnieniem, że trójkąt równoboczny nie ma żadnego miejsca z czterema sąsiadami.



Rysunek 6.4 Trójkąt równoboczny podzielony na segmenty ilości aktywnych sąsiadów (widok wierzchołka)

W poprzednim przypadku z kwadratem o ośmiu sąsiadach określenie wymiarów w poszczególnych segmentach było stosunkowo proste. W przypadku trójkąta dodatkowych wyjaśnień wymaga segment P_1.



Rysunek 6.5 Trójkąt równoboczny z wyszczególnieniem trapezu P_1

Rysunek 6.5 przedstawia dokładnie, że pole P_1 jest trapezem równoramiennym ograniczonym wierzchołkami BEB_1E_1. Numeracja została celowo dobrana jako te same litery z dolnymi indeksami, aby podkreślić fakt, iż drugi wierzchołek trójkąta ma taki sam rozkład segmentów, jak pierwszy wierzchołek, widoczny na rysunku 6.4. Na podstawie tych dwóch rysunków można określić miary kolejnych odcinków potrzebnych do obliczenia pól segmentów:

$|AA_1| = a$
$|AG| = |AF| = |BH| = r$
$|AE| = |AD| = |DE| = |DB| = |BE| = |AD| = |A_1E_1| = |A_1D_1| = |B_1D_1| = |B_1E_1| = z$
$|EH| = y$
$|EE_1| = c$
$|BB_1| = d$

Zmienne *a* i *r* są znane jako kolejno bok rozważanego trójkąta oraz promień zainteresowania gracza, jednak zmienne *z, y, c* oraz *d* nie są znane. Przy pomocy pozostałych miar można je obliczyć następująco:

$$z = |BE| = \frac{|BH|}{\sin|\sphericalangle BEH|} = \frac{r}{\sin 60°} = \frac{2}{\sqrt{3}}r = \frac{2}{3}\sqrt{3}r$$

$$y = |EH| = \frac{|BH|}{tan|\sphericalangle BEH|} = \frac{r}{tan\,60°} = \frac{r}{\sqrt{3}} = \frac{\sqrt{3}}{3}\mathrm{r}$$

$$c = |EE_1| = |AA_1| - |AE| - |A_1E_1| = a - 2z = a - \frac{4}{3}\sqrt{3}\mathrm{r}$$

$$d = |BB_1| = |EE_1| - 2|EH|$$

$$d = c - 2y = c - \frac{2}{3}\sqrt{3}r = a - 2\sqrt{3}r$$

Poniższa tabela 6.2 opisuje poszczególne pola powierzchni, które są potrzebne do obliczenia.

Tabela 6.2 Opis fragmentów pól powierzchni dla trójkąta równobocznego

Oznaczenie	Opis figury	Przykład występowania na rysunku 7.4 lub 7.5 [figura ograniczona punktami]
P_1	Trapez równoramienny o wysokości *r*, dolnej podstawie *c* i górnej podstawie *d*	BEB_1E_1
P_2	Trójkąt równoboczny o boku *z*	BDE
P_3	Połowa różnicy między polem trójkąta równobocznego o boku *z* i szóstą częścią koła o promieniu *r*	CDF
P_5	Szósta część koła o promieniu *r*	AGF
P_s	Trójkąt równoboczny o boku *a*	Cała figura

Dzięki tym danym można policzyć średnią ilość aktywnych sąsiadów *S* dla trójkąta równobocznego, posiłkując się wzorem 6.1. Na początek potrzebne jest ustalenie znanej z wzoru 6.1 zmiennej *n* oraz zbioru *x*:

$$n = 5$$

$$x = \{3, 3, 6, 0, 3\}$$

$$P_s = \frac{a^2\sqrt{3}}{4}$$

$$P_1 = \frac{\mathrm{c} + \mathrm{d}}{2}\mathrm{r} = \frac{2a - \frac{10}{3}\sqrt{3}r}{2}\mathrm{r} = \left(a - \frac{5}{3}\sqrt{3}r\right)r = ar - \frac{5}{3}\sqrt{3}r^2$$

$$P_2 = \frac{\sqrt{3}}{4}\mathrm{z}^2 = \frac{\sqrt{3}}{3}\mathrm{r}^2$$

$$P_5 = \frac{1}{6}\pi r^2$$

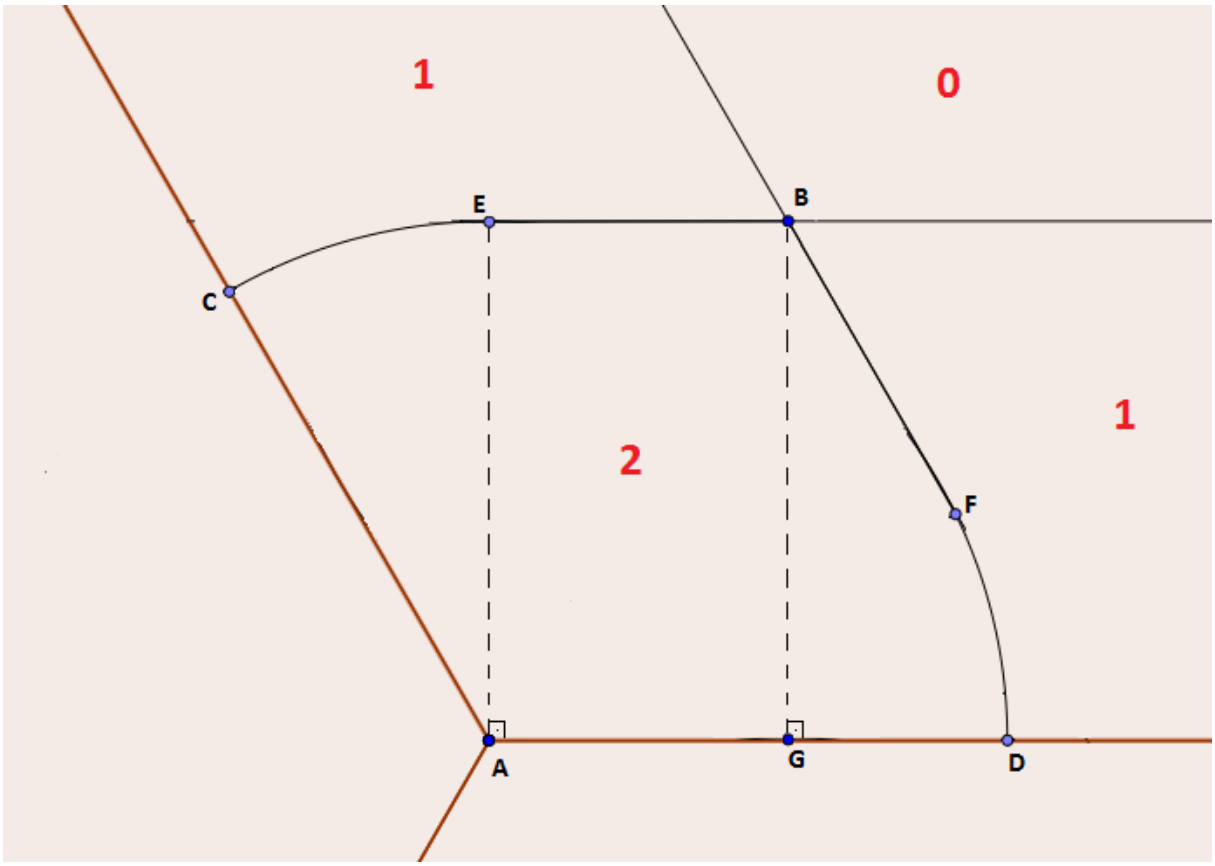
$$P_3 = \frac{1}{2}(P_2 - P_5) = \frac{1}{2}\left(\frac{\sqrt{3}}{3}\mathrm{r}^2 - \frac{1}{6}\pi r^2\right) = \frac{1}{2}r^2\left(\frac{2\sqrt{3}-\pi}{6}\right) = \frac{2\sqrt{3}-\pi}{12}r^2$$

$$S_2 = \frac{\sum_{i=1}^{n} P_i \cdot i \cdot x_i}{P_s}$$

$$S_2 = \frac{3\left(ar - \frac{5}{3}\sqrt{3}r^2\right) + 6\left(\frac{\sqrt{3}}{3}\mathrm{r}^2\right) + 18\left(\frac{2\sqrt{3}-\pi}{12}r^2\right) + 15\left(\frac{1}{6}\pi r^2\right)}{\frac{a^2\sqrt{3}}{4}}$$

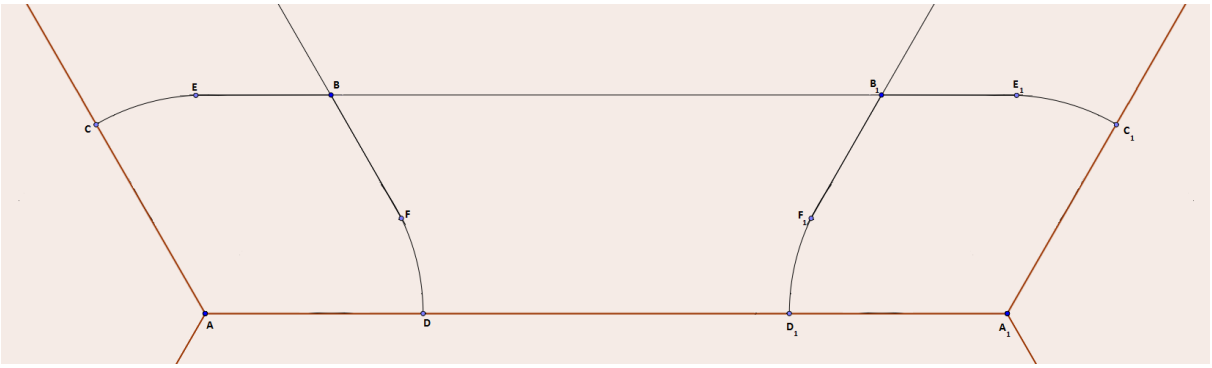
$$S_2 = \frac{12r}{a\sqrt{3}} + \frac{4\pi r^2}{a^2\sqrt{3}} \qquad (6.3)$$

Kolejnym rozważanym typem segmentów jest sześciokąt foremny. W tym przypadku, podobnie jak w trójkącie równobocznym, autor skupił się na pojedynczym wierzchołku sześciokąta.



Rysunek 6.6 Sześciokąt foremny podzielony na segmenty ilości aktywnych sąsiadów (widok wierzchołka)

Na rysunku 6.6 widać fragmentację sześciokąta w zależności od ilości aktywnych sąsiadów. W tym wypadku może być maksymalnie tylko dwóch aktywnych sąsiadów. Podobnie jak w przypadku trójkąta równobocznego, pole P_1 nie jest w pełni widoczne na rysunku 6.6, wobec czego załączony został drugi schemat, rysunek 6.7, który prezentuje to pole wyraźniej. Tak jak w poprzednim przypadku, numeracja punktów przy drugim wierzchołku celowo jest zbliżona do numeracji przy pierwszym wierzchołku.



Rysunek 6.7 Sześciokąt foremny z wyszczególnieniem segmentu P_1

Z powyższego rysunku wynika, że obliczenie pola P_1, które jest ograniczone wierzchołkami $BFDB_1F_1D_1$ nie jest możliwe do obliczenia wprost. Zostało zauważone natomiast, że pole to jest częścią trapezu ABA_1B_1, co zostało wykorzystane do obliczenia szukanego segmentu. Obliczenia zostały rozpoczęte od zdefiniowania wszystkich miar i zmiennych potrzebnych do obliczenia P_1 i P_2.

$$|AA_1| = |EE_1| = a$$
$$|AC| = |AE| = |AF| = |AD| = |BG| = |A_1C_1| = |A_1E_1| = |A_1F_1| = |A_1D_1| = |B_1G_1| = r$$
$$|BE| = |BF| = |B_1E_1| = |B_1F_1| = b$$
$$|BB_1| = c$$
$$|\sphericalangle EAB| = |\sphericalangle CAE| = |\sphericalangle FAD| = |\sphericalangle E_1A_1B_1| = |\sphericalangle C_1A_1E_1| = |\sphericalangle F_1A_1D_1| = 30°$$

Oprócz wypisanych długości odcinków, ważne są w tym przypadku miary niektórych kątów. Kąty takie jak $|\sphericalangle CAE|$ bezpośrednio określają jaka część koła należy do segmentu P_2. Tak jak w poprzednim przypadku obliczeń na rzecz trójkąta równobocznego, zmienne a i r są znane, jednak zmienne b oraz c muszą zostać obliczone. Wykorzystuje się do tego przede wszystkim fakt, iż odcinek |AE| tworzy kąt prosty z dolnym bokiem sześciokąta $|AA_1|$. Zdefiniowane zostały również zakresy pól P_1 oraz P_2, które nie są prostymi figurami i obliczenie ich jest

możliwe tylko przy podziale ich na mniejsze figury lub rozszerzenie ich pola do większego kształtu od którego można odjąć pole powierzchni mniejszej figury.

$$b = |BE| = |AE| \cdot \tan|\sphericalangle EAB| = r \cdot \tan 30° = \frac{\sqrt{3}}{3} r$$

$$c = |BB_1| = |EE_1| - |BE| - |B_1E_1| = a - 2b = a - \frac{2\sqrt{3}}{3} r$$

$$P_1 = P|BDFB_1D_1F_1| = P|ABA_1B_1| - (P|ABF| + P|ADF| + P|A_1B_1F_1| + P|A_1D_1F_1|)$$

$$P_2 = P|ADFBEC| = P|ACE| + P|ABE| + P|ABF| + P|ADF|$$

Powyższe obliczenia i spostrzeżenia zostały zgrupowane w tabeli 6.3.

Tabela 6.3 Opis fragmentów pól powierzchni dla sześciokąta foremnego

Oznaczenie	Opis figury	Przykład występowania na rysunku 7.6 lub 7.7 [figura ograniczona punktami]
P_1	Różnica między trapezem równoramiennym o podstawach *a* i *c* oraz wysokości *r*, a sumą dwóch trójkątów prostokątnych o przyprostokątnych *b* i *r* oraz szóstej części koła o promieniu *r*	$BDFB_1D_1F_1$
P_2	Suma dwóch trójkątów prostokątnych o przyprostokątnych *b* i *r* oraz szóstej części koła o promieniu *r*	ADFBEC
P_s	Sześciokąt foremny o boku *a*	Cała figura

$$P_1 = \frac{a+b}{2} r - 2 \cdot P|ABFD|$$

$$P_1 = \frac{a + a - \frac{4\sqrt{3}}{3} r}{2} r - \frac{\pi + 2\sqrt{3}}{6} r^2$$

$$P_1 = ar - (\sqrt{3} + \frac{\pi}{6}) r^2$$

Po obliczeniu wszystkich pól cząstkowych, można obliczyć średnią ilość aktywnych sąsiadów *S* dla sześciokąta foremnego, posiłkując się wzorem 6.1.

$$n = 2$$

$$x = \{6, 6\}$$

$$P_s = \frac{3 \cdot a^2\sqrt{3}}{2}$$

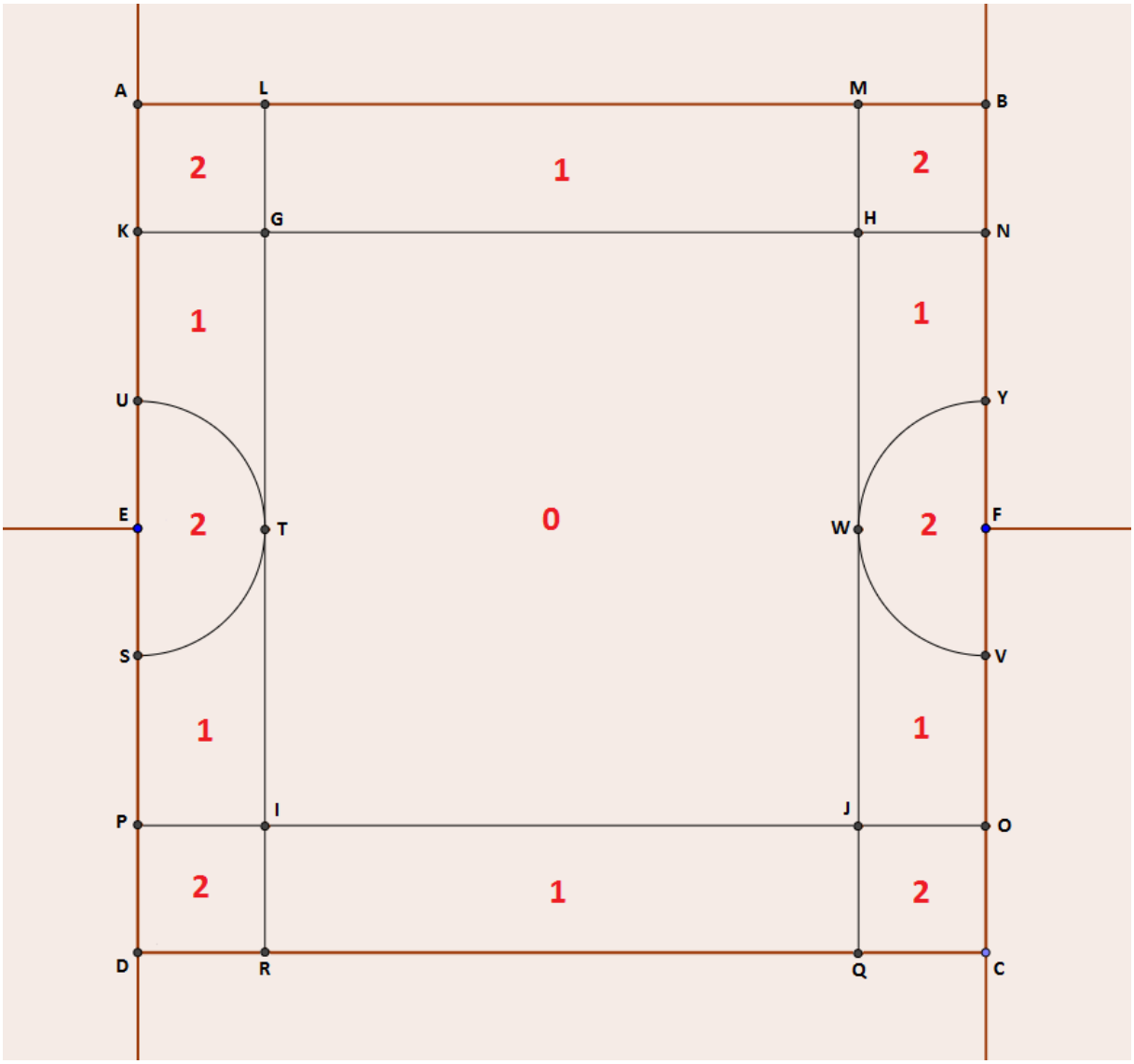
$$S_3 = \frac{\sum_{i=1}^{n} P_i \cdot i \cdot x_i}{P_s}$$

$$S_3 = \frac{P_1 \cdot 1 \cdot 6 + P_2 \cdot 2 \cdot 6}{P_s}$$

$$S_3 = \frac{6\left(ar - \left(\sqrt{3} + \frac{\pi}{6}\right)r^2\right) + 12\left(\frac{\pi + 2\sqrt{3}}{6}r^2\right)}{\frac{3 \cdot a^2\sqrt{3}}{2}}$$

$$S_3 = \frac{4r}{a\sqrt{3}} + \frac{2(\pi+2\sqrt{3})r^2}{3 \cdot a^2\sqrt{3}} \qquad (6.4)$$

Ostatnim z rozważanych typów jest kwadrat z sześcioma sąsiadami. Jest to modyfikacja pierwszego typu kwadratu w której co druga kolumna segmentów jest przesunięta o połowę długości krawędzi. Zgodnie ze źródłami[2] ma to zapewnić połączenie zalet wariantów kwadratu o ośmiu sąsiadach i sześciokąta foremnego. Kwadrat miał aż ośmiu sąsiadów i wiele miejsc w których mógł mieć trzech z nich aktywnych. Dzięki przesunięciu co drugiej kolumny, całkowita liczba sąsiadów została zmniejszona do sześciu, zaś maksymalna ilość aktywnych sąsiadów do dwóch (patrz rysunek 6.8). Otoczenie kwadratu w ten sposób bardzo zbliżyło go do schematu sześciokąta, jednak rozwiązanie to ma jeszcze jedną zaletę. W kartezjańskim układzie współrzędnych obliczenie punktów należących do wielokąta jest o wiele łatwiejsze dla figur składających się wyłącznie z linii pionowych i poziomych (kwadrat) niż dla figur, które mają linie ukośne.



Rysunek 6.8 Kwadrat o sześciu sąsiadach podzielony na segmenty ilości aktywnych sąsiadów

Zgodnie z rysunkiem 6.8, w schemacie kwadratu z sześcioma sąsiadami są różne kształty segmentów w których są takie same ilości aktywnych sąsiadów. Przykładem są prostokąt |LMGH| i figura |KGTU|. Odróżnia to aktualny przypadek od wcześniejszych segmentów, które były obliczane, ponieważ wcześniej wszystkie segmenty o tej samej liczbie aktywnych sąsiadów miały ten sam kształt i pole powierzchni.

Tak jak poprzednio, obliczenia zostały rozpoczęte od zdefiniowania zmiennych w schemacie.

$$|AB| = |BC| = |CD| = |AD| = a$$

$$|EU| = |ET| = |ES| = |FY| = |FW| = |FV| = |AL| = |LG| = |GK| = |AK| = |BM|$$
$$= |BN| = |HM| = |HN| = |CO| = |CQ| = |JQ| = |JO| = |DR| = |DP|$$
$$= |IP| = |IR| = r$$

$$|LM| = |GH| = |IJ| = |RQ| = |KP| = |GI| = |NO| = |HJ| = a - 2r = b$$

$$|EK| = |GT| = |EP| = |IT| = |FN| = |FO| = |JW| = |HW| = \frac{a}{2} - r = c$$

Podobnie jak w poprzednich obliczeniach, nie wszystkie zmienne są wyłącznie znanymi odcinkami *a* i *r*, jednak w tym przypadku zmienne *b* i *c* daje się bardzo łatwo zapisać za pomocą tych odcinków.

Pola powierzchni segmentów w zależności od ilości aktywnych sąsiadów zostały opisane w tabeli 6.4.

Tabela 6.4 Opis fragmentów pól powierzchni dla kwadratu o sześciu sąsiadach

Oznaczenie	Opis figury	Przykład występowania na rysunku 7.8 [figura ograniczona punktami]
P_1	Prostokąt o bokach *r* i *b*	LMHG
P_1'	Różnica między prostokątem o bokach *r* i *c* oraz czwartą częścią koła o promieniu *r*	KGTU
P_2	Kwadrat o boku *r*	ALKG
P_2'	Połowa koła o promieniu *r*	UTSE
P_s	Kwadrat o boku *a*	Cała figura

Z powodu różnorodności kształtu segmentów o tej samej ilości aktywnych sąsiadów, wzór 6.1 nie może być tutaj wykorzystany w swojej ogólnej formie. Obliczenia średniej ilości aktywnych sąsiadów *S* należy więc zacząć od postaci szczególnej. Każda liczba przy kolejnym polu P_i wyraża iloczyn ilości wystąpień P_i na rysunku 6.6 oraz ilości *i* aktywnych sąsiadów, którą reprezentuje to pole.

$$S_4 = \frac{2 \cdot P_1 + 4 \cdot P_1' + 8 \cdot P_2 + 4 \cdot P_2'}{P_s}$$

$$P_s = a^2$$

$$\mathrm{P_1} = \mathrm{rb} = \mathrm{r(a - 2r)} = \mathrm{ar} - 2\mathrm{r}^2$$

$$\mathrm{P_1'} = \mathrm{rc} - \frac{\pi}{4}\mathrm{r}^2 = \mathrm{r}\left(\frac{a}{2} - \mathrm{r}\right) - \frac{\pi}{4}\mathrm{r}^2 = \frac{\mathrm{ar}}{2} - (1 + \frac{\pi}{4})\mathrm{r}^2$$

$$\mathrm{P_2} = \mathrm{r}^2$$

$$P_2' = \frac{\pi}{2}r^2$$

$$S_4 = \frac{2(ar - 2r^2) + 4\left(\frac{ar}{2} - (1 + \frac{\pi}{4})r^2\right) + 8r^2 + 4 \cdot \frac{\pi}{2}r^2}{a^2}$$

$$S_4 = \frac{4r}{a} + \frac{3\pi r^2}{a^2} \qquad (6.5)$$

Po wykonaniu wszystkich obliczeń (wzory 6.2 do 6.5) można przejść do sprawdzenia która średnia S_i daje najlepszą wartość w zależności od wielkości przyjętego segmentu. Aby to zrobić, należy przekształcić zmienne *a* (reprezentujące bok każdego wielokąta) do wspólnej postaci. Zostało wcześniej napisane, że porównywanie różnych rodzajów segmentów jest możliwe tylko przy takim doborze ich wymiarów, aby wszystkie miały takie samo pole powierzchni. Oznacza to, że zmienne *a* z kolejnych wzorów S_1 do S_4 trzeba rozdzielić na zmienne a_1 do a_4 i, na podstawie kształtu wielokąta, które przedstawia konkretne a_i, obliczyć ich wartość w postaci, którą można bezpośrednio porównać. Zmienne a_1 i a_4 oznaczają boki kwadratu, zatem oznaczenie a_4 nie jest potrzebne.

Tabela 6.5 Opis kolejnych rozważanych wielokątów foremnych

Wielokąt	Pole powierzchni	Miara boku a_i *przyrównana do miar boków kolejnych figur*:		
		Kwadratu	Trójkąta	Sześciokąta
Kwadrat	$a_1{}^2$	N/D	$a_1 = \frac{a_2\sqrt[4]{3}}{2}$	$a_1 = \frac{3^{\frac{3}{4}} \cdot a_3\sqrt{2}}{2}$
Trójkąt	$\frac{a_2{}^2\sqrt{3}}{4}$	$a_2 = \frac{2a_1\sqrt[4]{3}}{3}$	N/D	$a_2 = a_3\sqrt{6}$
Sześciokąt	$\frac{3 \cdot a_3{}^2\sqrt{3}}{2}$	$a_3 = 3^{\frac{4}{3}} \cdot a_1\sqrt{2}$	$a_3 = \frac{a_2\sqrt{6}}{6}$	N/D

Przy przyjętym założeniu, że niezależnie od wybranego kształtu segmentu mapy, jego pole powierzchni będzie stałe, należy obliczyć zależności między zmiennymi a_1, a_2 oraz a_3. Oblicza się to przyrównując do siebie kolejne pola powierzchni, na przykład kwadratu i trójkąta, aby obliczyć zależność między a_1 i a_2. Wyniki takich obliczeń zostały zebrane na w tabeli 6.5.

Przekształcenie wzorów 6.2 do 6.5 może zostać wykonane przy pomocy dowolnej zmiennej a_1, a_2 lub a_3 w oparciu o tabelę 6.5. W tym przypadku autor zdecydował się zrobić to przy pomocy a_2.

$$a_1 = \frac{a_2 \sqrt[4]{3}}{2}$$

$$a_3 = \frac{a_2 \sqrt{6}}{6}$$

$$S_1 = \frac{4r}{a_1} + \frac{\pi r^2}{{a_1}^2} = \frac{8r}{a_2 \sqrt[4]{3}} + \frac{4\pi r^2}{{a_2}^2 \sqrt{3}}$$

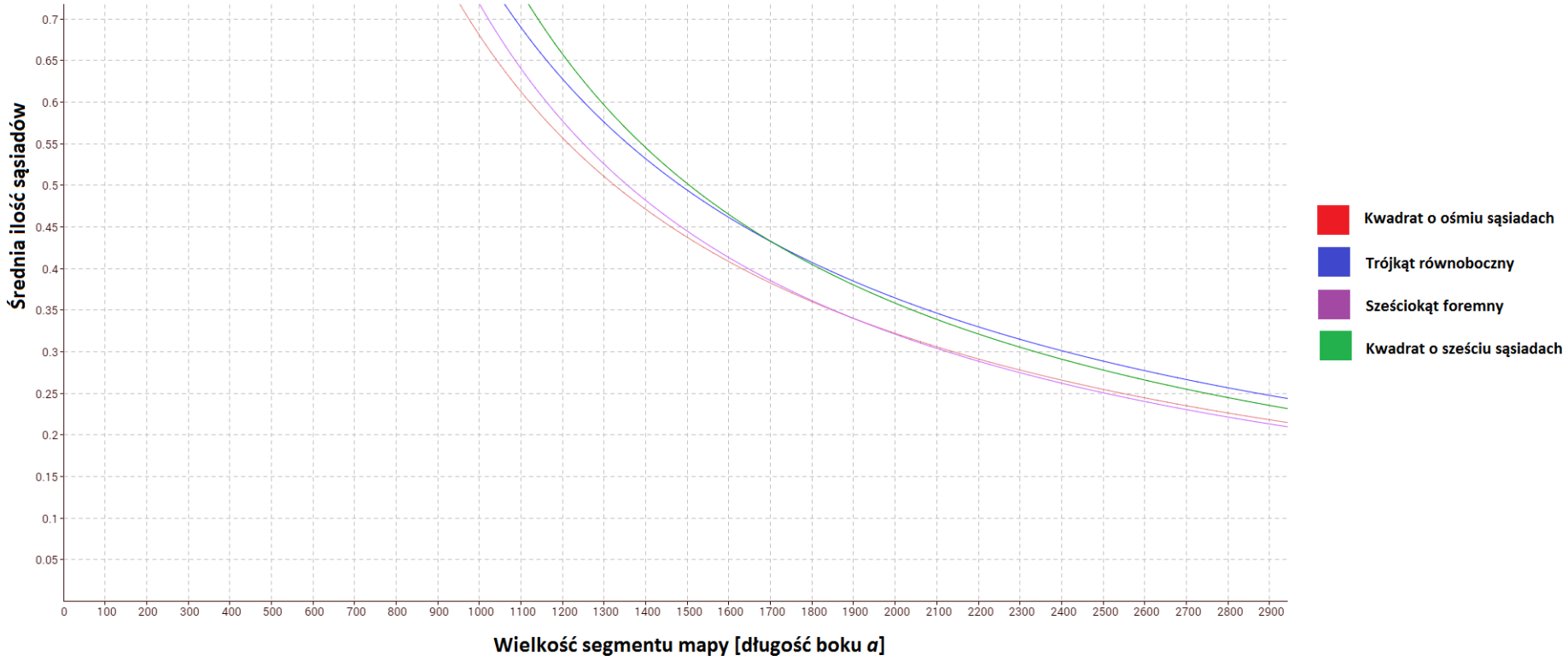
$$S_2 = \frac{12r}{a_2 \sqrt{3}} + \frac{4\pi r^2}{{a_2}^2 \sqrt{3}}$$

$$S_3 = \frac{4r}{a_3 \sqrt{3}} + \frac{2(\pi + 2\sqrt{3}) r^2}{3 \cdot {a_3}^2 \sqrt{3}} = \frac{24r}{a_2 \sqrt{18}} + \frac{72(\pi + 2\sqrt{3}) r^2}{18 \cdot {a_2}^2 \sqrt{3}} = \frac{8r}{a_2 \sqrt{2}} + \frac{4(\pi + 2\sqrt{3}) r^2}{{a_2}^2 \sqrt{3}}$$

$$S_4 = \frac{4r}{a_1} + \frac{3\pi r^2}{{a_1}^2} = \frac{8r}{a_2 \sqrt[4]{3}} + \frac{12\pi r^2}{{a_2}^2 \sqrt{3}}$$

Oznaczenie r w powyższych wzorach określa promień zainteresowania graczy w grze sieciowej i jest stałe (takie samo) dla wszystkich rodzajów segmentów. Oznacza to, że za r można podstawić konkretne (wybrane) wartości liczbowe, aby móc ostatecznie porównać średnie S_i w zależności od a_2.

Zaobserwowano, że dla różnych wartości r trendy zależności między funkcjami S_i są takie same, wobec czego załączono tylko jeden wykres, który je przedstawia. Na wykresie 6.1 zebrano wyniki badania dla wybranego $r=100$. Interpretując wyniki wykresu należy pamiętać o założeniu, że promień zainteresowania r powinien być znacznie mniejszy od boku wielokąta a. Jak można wyczytać z wykresu, zależności między kolejnymi rodzajami segmentów zaczynają się dopiero kiedy r staje się około szesnastokrotnie mniejsze od a. Wówczas wykresy poszczególnych S_i krzyżują się ze sobą i zaburzana jest dotychczasowa hierarchia segmentów. W pierwszej części wykresu najlepiej prezentuje się segment kwadratu z ośmioma sąsiadami, który ma najmniej aktywnych sąsiadów. Kiedy jednak zmienna a_2 przekroczy dwudziestokroć wartości r, trend ten się zmienia i lepszym rozwiązaniem staje się sześciokąt foremny. Podobne zachowanie można zaobserwować pomiędzy trójkątem równobocznym, a kwadratem z sześcioma sąsiadami.



Wykres 6.1 Porównanie ilości średnich sąsiadów dla różnych segmentów w zależności od wielkości segmentu dla przyjętego r=100

Analiza wyników z powyższego wykresu nie jest prosta i w dużej mierze pokazuje, że różne rozwiązania oferują mają swoje pozytywne, jak i negatywne strony. Przystępując do tego zadania, należy wziąć pod uwagę nie tylko sam wykres, ale także cechy wynikające z budowy omawianych segmentów.

Wykres wykazał, że w pierwszej części dominującym segmentem jest kwadrat o ośmiu sąsiadach. Rozwiązanie to jest najprostsze w implementacji programowej, gdyż kwadrat jest figurą, której każdy kąt wewnętrzny ma 90 stopni. Jest to bardzo ważna cecha, która znacząco ułatwia opisywanie i przetwarzanie takiej figury w kartezjańskim układzie współrzędnych. Segment ten ma jednak tę wadę, że serwer, który go obsługuje musi zawsze mieć informacje o wszystkich swoich sąsiadach, a w tym wypadku oznacza to osiem dodatkowych segmentów, zaś w przypadku granicznym serwer może zostać zmuszony wysyłać do klienta informacje o trzech z nich.

W drugiej części wykresu (kiedy a_2 przekroczy dwudziestokroć wartości r) lepszym rozwiązaniem staje się sześciokąt foremny. Figura ta ma również tę zaletę w stosunku do wcześniej wybranego kwadratu, że ma jedynie sześciu sąsiadów, a sytuacja graniczna zmusi serwer do wysłania informacji na temat jedynie dwójki sąsiadów. Wadą tego rozwiązania jest konieczność implementacji samego sześciokąta w programowej interpretacji kartezjańskiego układu współrzędnych.

Drugi rodzaj kwadratu, który wcześniej został opisany jako będący kompromisem pomiędzy pierwszym rodzajem kwadratu a sześciokątem, daje znacznie gorsze rezultaty na wykresie niż jego pierwowzory. Mimo to może zostać uznany jako dobre rozwiązanie z punktu widzenia tamtych cech - sześciu sąsiadów, maksymalnie dwóch aktywnych sąsiadów naraz i łatwość implementacji w układzie kartezjańskim.

Trójkąt równoboczny jako jedyny nie wykazał żadnych cech pozytywnych. Jego ogólna ilość sąsiadów i możliwa ilość aktywnych sąsiadów jest rekordowo duża na tle pozostałych rozwiązań, zaś wykres 6.1 wykazał, że jego średnia również jest za wysoka.

Ostatecznie, zdaniem autora pracy, najlepszym rozwiązaniem jest sześciokąt foremny. Figura ta daje najlepsze faktyczne rezultaty, zaś jej wada (trudność implementacji programowej) może zostać łatwo pokonana, jeśli tylko przyjąć założenie że dwuwymiarową mapę przedstawia się za pomocą trzech wymiarów[13]. W dwuwymiarowej przestrzeni dostępne są cztery ruchy do wykonania (dodanie lub odjęcie jednego z wymiarów), co jest dobrze dostosowane do badania granic figur takich jak kwadrat lub prostokąt. W trójwymiarowej przestrzeni dostępnych jest już sześć różnych ruchów, co pozwala na badanie w taki sam sposób granic sześciokątów.

7. Dalszy rozwój prac

W niniejszej pracy zostały poruszone tylko niektóre z wielu problemów optymalizacyjnych, jakie dotykają gry sieciowe:

- w jaki sposób najskuteczniej przesyłać dane
- jak instancja serwera powinna współdzielić dane z innymi instancjami
- w jaki sposób ograniczyć rosnące zależności przesyłanych i przechowywanych przez serwer danych w zależności od ilości podłączonych graczy

W tej pracy przyjęto wiele założeń, które w innych warunkach mogą okazać się cennym źródłem informacji na temat wydajności serwerów gier sieciowych. Przykładami takich przyjętych założeń są:

- sieciowa gra przygodowa - opisane zostały pokrótce inne rodzaje gier sieciowych, które cechują się zupełnie innymi problemami niż ta, która została omówiona w tej pracy
- idealne warunki sieciowe - w jaki sposób serwer powinien sobie radzić z zakłóceniami połączenia?
- wykorzystanie protokołu TCP - w pracy wspomniano, że protokół UDP nie jest dobrym rozwiązaniem dla omawianych gier przygodowych, jednak istnieją również inne rozwiązania omówione między innymi w [1].
- badanie wyłącznie czasu opóźnienia i ilości przesyłanych danych - innymi czynnikami, które powinny być zbadane są ilość utraconych pakietów lub tendencje do zrywania połączenia w określonych warunkach
- brak danych do badania segmentacji mapy - zdobycie danych poruszania się graczy w konkretnej grze pozwoliłoby ukierunkować badania na tę właśnie grę. Pozwoliłoby to także zbadać przedstawione w tej pracy teoretyczne rozważania na temat segmentacji mapy w realnych warunkach

Autor zachęca również do zbadania wpływu środowiska programowego na wydajność gier sieciowych. W tym wypadku można porównać różne technologie tworzenia aplikacji, na przykład języki programowania lub bazy danych.

8. Wnioski końcowe

Temat pracy okazał się bardzo rozległy. Poruszone zostały jedynie niektóre tematy optymalizacyjne gier sieciowych, które skupiały się wokół komunikacji serwera z klientem aplikacji i szybkości przetwarzania danych. Nie ulega wątpliwości, że głównym zadaniem serwera gry sieciowej jest komunikacja z klientem w taki sposób, aby udostępniać mu dane o innych użytkownikach aplikacji. W opinii autora, większość problemów optymalizacyjnych w zakresie omawianej pracy sprowadza się do ustalenia jak najskuteczniej i najszybciej przeprowadzać tę komunikację. W trakcie prac znalezionych zostało wiele czynników, które utrudniały pracę serwera właśnie w zakresie opóźnionej lub pogorszonej komunikacji.

Z badań na temat ilości przesyłanych danych w sieci (rozdział czwarty) wynika, że jest to istotny czynnik, który wpływa znacząco na wydajność serwera. Dzieje się tak dlatego, że formatowanie JSON, które zostało użyte pierwotnie, jest dobre tylko w przypadku połączeń w których zapytania są wykonywane rzadko. Przykładem jest przeglądanie stron internetowych. Klient (przeglądarka) odpytuje bezstanowy serwer HTTP tylko w momencie wchodzenia na nową stronę internetową, wobec czego koszty ponoszone przez dużą ilość przesyłanych informacji są niewielkie. Ponadto przeglądarki internetowe są dobrze dostosowane do odbierania standardowych formatów danych, takich jak JSON. W przypadku rozważanego serwera gry sieciowej rozwiązanie to było złe, ponieważ dane były przesyłane na bieżąco, aby utrzymywać klienta w stałej synchronizacji z otoczeniem. Rozwiązanie z formatowaniem binarnym było w tym wypadku znacznie lepsze z powodu mniejszego nakładu ruchu sieciowego, a także niskimi kosztami własnej implementacji takiego protokołu komunikacyjnego (w przypadku przeglądarki internetowej byłoby to niemożliwe). W tym zakresie autor zachęca przede wszystkim do zbadania innych protokołów komunikacyjnych (w tej pracy został użyty TCP). Zbadanie możliwości wykorzystywania kilku połączeń różnymi protokołami naraz (na przykład: TCP i UDP) jest również obiecującą gałęzią do dalszych prac, gdyż daje możliwość rozdzielenia omawianej w pracy instancji serwera na co najmniej dwie: taką, która musi wysyłać dane często (na przykład: aktualizacje środowiska gracza - UDP) oraz taką, która musi wysyłać dane precyzyjnie (na przykład: reakcja na zapytanie gracza - TCP).

Z badań na temat przetwarzania danych przez serwer wynika, że pobieranie informacji z bazy danych jest bardzo kosztowne czasowo. Inteligentna struktura danych wewnątrz programu nadaje się o wiele lepiej do ciągłego przetwarzania informacji z jedynie rzadką

ingerencją ze strony bazy danych (aby synchronizować dane globalne). Dalsze prace w tym zakresie mogłyby polegać na badaniu samych baz danych - które technologie bazodanowe się lepiej nadają do ciągłego odpytywania przez wiele połączeń naraz (w tej pracy została użyta baza danych MySQL).

Najwięcej uwagi zostało poświęcone segmentacji mapy serwera. Zdaniem autora analiza, wnioski oraz znalezione źródła naukowe mogą stanowić dobrą podstawę do przeprowadzenia praktycznych eksperymentów w tym zakresie. Równoważenie pracy jest w przypadku serwera gry sieciowej najważniejszym zagadnieniem optymalizacyjnym, ponieważ pozwala zrównoleglić pracę pomiędzy wieloma maszynami, co bezpośrednio przyczynia się do większej wydajności aplikacji sieciowej jako całości. Ustalana jest wtedy hierarchia zależności, która dąży do rozdzielenia jedynie niezbędnych informacji i wykonywanych prac pomiędzy kolejne elementy serwera. Przykładowo instancja serwera nie wie nic na temat ilości wszystkich połączeń z graczami, o czym wie dopiero serwer nasłuchujący na połączenia do klientów. Stworzenie takiej inteligentnej sieci zależności w której aplikacja jako całość wykonuje swoją pracę wydajnie, jednak jest podzielona na mniejsze elementy, które pozwalają na maksymalne zrównolegnienie pracy, jest również możliwością do dalszego rozwoju tej pracy.

W takim wypadku serwer gry sieciowej musi zostać rozbity na drzewo niezależnych aplikacji w których każda ma ściśle określone zadanie, które polega na przetworzeniu pewnych danych i odesłaniu odpowiedzi lub przekazaniu zapytania dalej. Zalążek takiego drzewa został przedstawiony już w tej pracy, gdzie serwer główny nasłuchuje na nowe połączenia i przekazuje je do instancji serwera, która zajmuje się dalszą komunikacją z klientem.

9. Spis tabel, wykresów i rysunków

Spis tabel

Spis wykresów

Spis rysunków

Spis listingów

10. Literatura

1. Chen-Chi Wu, Kuan-Ta Chen, Chih-Ming Chen, Polly Huang, and Chin-Laung Lei, "On the Challenge and Design of Transport Protocols for MMORPGs," Multimedia Tools and Applications (special issue on Massively Multiuser Online Gaming Systems and Applications), Vol. 45, No. 1, Oct, 2009

2. Kusno Prasetya, "Efficient Methods for Improving Scalability and Playability of Massively Multiplayer Online Game (MMOG)", Dissertation submitted in fulfilment of the requirements of the degree of Doctor of Philosophy for the School of Information Technology, Bond University, Feb 2010

3. Kuan-Ta Chen, Polly Huang, Chin-Laung Lei, "How Sensitive Are Network Quality?", Communications of the ACM, Nov 2006

4. "FAQ - Multiplayer and Network Programming", GameDev.net, dostęp online 16.05.2015: http://www.gamedev.net/

5. A. El Rhalibi, M. Merabti, and Y. Shen. "Aoim in peer-to-peer multiplayer online games." In ACE ’06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology, strona 71, New York, NY, USA, 2006. ACM.

6. "JSON Tutorial", w3schools.com, dostęp online 16.05.2015: http://www.w3schools.com/json/

7. "Building a simple yet powerful MMO game architecture", ibm.com, dostęp online 16.05.2015: http://www.ibm.com/developerworks/library/ar-powerup1/

8. "How does a mutex work? What does it cost?", mortoray.com, dostęp online 16.05.2015: http://mortoray.com/2011/12/16/how-does-a-mutex-work-what-does-it-cost/

9. Nicolai M. Josuttis, "C++ Biblioteka Standardowa", Wydanie 2, Wydawnictwo Helion 2014, ISBN 978-83-246-5576-2

10. Fengyun Lu, Simon Parkin, Graham Morgan, "Load Balancing for Massively Multiplayer Online Games", Proceedings of 5th ACM SIGCOMM NetGames, 2006

11. "What is Load Balancing?", nginx.com, dostęp online 16.05.2015: http://nginx.com/resources/glossary/load-balancing/

12. "Wowhead, the World of Warcraft database", wowhead.com, dostęp online 16.05.2015: http://www.wowhead.com/

13. "Hexagonal Grids", redblobgames.com, dostęp online 16.05.2015: http://www.redblobgames.com/grids/hexagons/