Fight in Square

Dokumentacja gry

# Informacje wstępne, zespół

Gra została zrealizowana w ramach projektu na zaliczenie przedmiotu "Podstawy Grafiki Komputerowej" w roku akademickim 2015/2016. Członkami zespołu byli:

1. Herman Katarzyna,
2. Drzazga Kamil,
3. Bartocha Łukasz,
4. Bajorek Tomasz.

# Opis projektu, założenia wstępne

Fight in Square to gra strategiczno - zręcznościowa, rozgrywana pomiędzy graczami przez sieć Internet. Można łączyć się z serwerem zarówno w sieci lokalnej, jak też globalnej, o ile komputer serwera posiada publiczny adres IP. Aby można było grać, najpierw należy uruchomić serwer. Zanim jednak się to zrobi, trzeba wybrać mapę, na której będzie się toczyła rozgrywka oraz czas gry.

Po uruchomieniu serwera klienci mogą się do niego połączyć, podając swój nick, adres IP serwera oraz numer portu, na którym on nasłuchuje. Do rozpoczęcia gry jest wymagana obecność na serwerze minimum dwóch graczy. Każdy nowy gracz pojawia się w losowym wolnym miejscu na planszy.

Klient posiada widok planszy z góry. Czołgi są w kolorach grup graczy. Dodatkowo każdy z graczy na swoim kliencie widzi wyróżniony własny czołg. Sterowanie grą odbywa się przy pomocy klawiatury. Ruch następuje po wciśnięciu jednego z klawiszy strzałek, natomiast strzelać można przy pomocy spacji. Można strzelać zarówno do graczy, jak i do przeszkód. Trafiony gracz traci część życia, odpowiednią do mocy trafiającego go pocisku. Jeżeli kula kilkakrotnie trafi w przeszkodę, może ją zniszczyć. Istnieją również przeszkody niezniszczalne.

Gracze podzieleni są na dwie grupy - czerwoną i niebieską. Podczas dołączania kolejnych graczy są oni tak przypisywani do grup, aby przewaga liczebna żadnej z nich nie wynosiła więcej niż jedną osobę. Jeżeli gracz zabije przeciwnika z innej drużyny, dostaje punkty. Wówczas małe punkty otrzymują wszyscy gracze jego koloru. Jeżeli natomiast zabije czołg o takim kolorze, jak jego własny, traci punkty.

Każdy gracz na początku posiada 100 punktów życia. Traci je, jeżeli zostanie trafiony pociskiem. Jeżeli liczba punktów życia spada do zera lub poniżej, gracz umiera i znika z planszy na 10 sekund. Po tym czasie pojawia się w wylosowanym wolnym miejscu na planszy.

Na ekranie oprócz mapy cały czas są wyświetlane lista punktów graczy wraz z zaznaczeniem ich drużyn oraz czas jaki upłynął od początku gry i limit czasu, ustawiony na serwerze. jest też przycisk, umożliwiający rozłączenie się.

Celem gry jest zdobycie przez drużynę jak największej ilości punktów. Gra kończy się po upływie czasu określonego na serwerze. Na ekranie podsumowującym pojawia się informacji o tym, kto wygrał, oraz ranking graczy - każdy z nich jest motywowany do gry nie tylko liczbą punktów drużyny, ale również jego własnych.

Do gry został dołączony edytor map, który ułatwia ich tworzenie. Jest on przygotowany w formie strony HTML, którą można otworzyć lokalnie przy pomocy przeglądarki internetowej. Elementy mapy posiadają swoje id, które należy wpisywać w odpowiednie miejsce. Kolumny planszy są oddzielane spacjami, a wiersze znakami nowej linii. Podczas tworzenia mapy na bieżąco jest generowany jej podgląd. Jeżeli klikając w dany element planszy, można go zmienić. Po zakończeniu projektowania gotową mapę można zapisać do pliku.

## 2.1. Ograniczenia nałożone na parametry gry

* gra rozpoczyna się gdy ilość graczy wyniesie 2; gdy w trakcie gry ta ilość zmaleje, gra ma nadal trwać;
* serwer może obsłużyć maksymalnie 8 graczy równocześnie;
* nazwa gracza musi być unikatowa na serwerze;
* nazwa gracza nie może być pusta;
* nazwa gracza nie może być dłuższa niż 16 znaków;
* nazwa gracza musi zawierać jedynie cyfry, małe i duże litery oraz znaki "\_" i ".";
* minimalny czas gry - 30 sekund;
* maksymalny czas gry - 10 minut;

## 2.2. Punktacja w grze

Zasady punktacji, obowiązującej w grze są następujące:

* każdy gracz rozpoczyna z ilością punktów równą 0;
* liczba punktów musi być z przedziału [0; 255]. Jeśli nastąpi przepełnienie - należy to potraktować jako bug aplikacji;
* zastrzelenie czołgu ze swojej drużyny -5 pkt;
* zestrzelenie czołgu z drużyny przeciwnej +3 pkt;
* ktoś z drużyny zastrzeli wrogi czołg +1 pkt;

# 3. Analiza projektu

## 3.1. Dane wejściowe

Serwer podczas gry rozsyła ramkę z planszą, na której znajdują się gracze i pociski. Schemat wszystkich ramek przesyłanych przez sieć został dołączony jako załącznik do niniejszej dokumentacji.

Rodzaj pól, jakie znajdują się na planszy jest wczytywany z pliku z mapą przez serwer przed rozpoczęciem gry. Struktura tego pliku jest bardzo prosta – kolejne pola w jednym wierszu planszy są oddzielone od siebie spacjami, natomiast wiersze planszy są rozdzielone znakami nowej linii. Każde pole określa identyfikator, tożsamy z nazwą pliku, zawierającego obraz wstawiany w to miejsce na planszy. Warunkiem poprawnego działania programu jest, aby identyfikator był cyfrą, a nie liczbą lub innym słowem.

Danymi wejściowymi dla serwera mogą być również ramki, przychodzące od klienta. Zawierają one ruchy gracza w poprzednim okresie gry, liczonym jako czas między kolejnymi rozesłaniami stanu gry przez serwer. Na ich podstawie może on prowadzić rozgrywkę.

Danymi wejściowymi dla klienta są nick gracza, oraz adres IP i numer portu, na którym nasłuchuje serwer. Te dane gracz podaje przed każdą próbą połączenia się z serwerem. Od strony sieci do klienta przychodzą z kolei ramki, zawierające aktualny stan planszy. Aplikacja kliencka nie rozgrywa gry, lecz tylko rysuje przejścia pomiędzy kolejnymi stanami gry, przysłanymi jej przez serwer. Zawierają one listę elementów planszy, informację o wszystkich graczach, biorących udział w rozgrywce, aktualne pozycje czołgów oraz informacje o pociskach, znajdujących się na planszy.

## 3.2. Dane wyjściowe

Danymi wyjściowymi, wysyłanymi z klienta do serwera i odwrotnie są ramki. Klient przesyła na początku żądanie połączenia się, a potem akcje wykonane przez gracza. Natomiast serwer odsyła odpowiedzi na żądania oraz rozsyła wszystkim podpiętym graczom ramkę stanu gry.

Aktualny stan gry w danym momencie jest wyświetlany na ekranie klienta tak, aby gracz widział, jak przebiega gra.

Zarówno ramki wejściowe jaki i wyjściowe są przesyłane w postaci ciągu znaków typu QByte i sa interpretowane przez odpowiednie dekodery.

## 3.3. Struktury danych

### 3.3.1. Ramki

Serwer i klient komunikują się ze sobą przy pomocy specjalnych, zaprojektowanych przez nas ramek, którymi przesyłają między sobą informacje:

* wysyłane przez klienta:
  + żądanie połączenia z serwerem - wysyłane przy każdej próbie połączenia; zawiera pusty identyfikator który zostanie mu dopiero naddany przez serwer) oraz nazwę gracza;
  + informacja o wykonanej akcji – jest wysyłana zawsze, kiedy gracz przesunie czołg lub wystrzeli w aktualnym kierunku; ramka jest przystosowana, by móc przenosić w sobie obie obydwa komunikaty;
  + żądanie rozłączenia z serwerem - wysyłane, jeśli w danym momencie gracz postanawia zakończyć grę; serwer po jej odebraniu usuwa go z mapy; jeżeli połączenie zostanie przerwane bez wysyłania tej ramki, serwer wykryje ten fakt i również usunie gracza;
* wysyłane przez serwer:
  + akceptacja połączenia - serwer wysyła tą ramkę, jeśli połączenie zostanie nawiązane; przesyła w niej przypisany graczowi identyfikator oraz czas, jaki upłynął już od rozpoczęcia gry;
  + stan gry – ramka, która zawiera aktualny stan gry w momencie jej wysłania; przekazuje ona klientowi wszystkie potrzebne informacje, aby mógł on na bieżąco rysować przebieg gry na ekranie; niesie w sobie także informacje o graczach;
  + odpowiedź na żądanie rozłączenia - odsyłana do klienta, jeśli serwer przyjął żądanie rozłączenia; jest to ostatnia ramka, jaka jest przesyłana między klientem a serwerem przed zamknięciem połączenia;
  + informacja o końcu gry – ramka wysyłana, kiedy upłynie czas gry; przesyła informacje o wszystkich graczach wraz z ich wynikami; po jej rozesłaniu serwer rozłącza klientów;

Uwagi:

1. w ciągu połączenia całego klient-serwer większość ramek jest wysyłana tylko raz, jedynie ramki akcji mogą być wysyłane dużo częściej (maks. 5 razy na sekundę) oraz ramki stanu gry, które są przesyłane przez serwer co 200 ms;
2. szczegółowa struktura ramek została przedstawiona na obrazku, będącym załącznikiem

### 3.3.2. Model danych

Klient oraz serwer korzystają z tego samego modelu danych, pracując na obiektach klas, których kod jest wspólną częścią obydwu programów.

Klasy modelu definiują wszystkie potrzebne cechy gracza, takie jak: identyfikator, nazwa, grupa, punkty, zdrowie, pozycja i kierunek ustawienia. Określają również właściwości pocisków, znajdujących się na planszy: identyfikator, początkowa pozycja, kierunek, identyfikator gracza, który wystrzelił pocisk, jego moc oraz ilość okresów gry, w których znajduje się on na planszy. Klasy modelu definiują również planszę do gry – jako dwuwymiarowy kontener na jej elementy, będący potomkiem klasy *QVector*.

Elementami planszy mogą być przeszkody. Czołg nie może na nie najechać. Część z nich może zostać zniszczona poprzez strzelanie do nich. Dlatego model przeszkody przechowuje informację o tym, czy może zostać zniszczony, a jeśli tak, to zawiera też informację o swoim aktualnym "zdrowiu".

Wszystkie trzy modele: gracza, pocisku oraz planszy - posiadają możliwość stworzenia na podstawie samych siebie ramki lub jej części, która zostanie wysłana do klienta. Z drugiej strony umożliwiają też ustawienie swojego stanu na podstawie otrzymanej ramki tak, aby klient mógł odtworzyć obiekt o takich samych właściwościach, jak jego odpowiednik na serwerze.

### 3.3.3. Sygnały i sloty

Qt posiada mechanizm sygnałów i slotów w celu komunikowania się obiektów między sobą. Sygnał i slot deklarowane są tak samo jak funkcje składowe klasy, przy czym sygnału się nie implementuje. Gdy sygnał jest przypięty po slot to wtedy jeśli wyemitowany zostanie sygnał, to slot go odbierze i się wykona. Ważne jest to, by deklaracje sygnału i slotu były ze sobą zgodne.

W aplikacji znajdują się następujące połączenia:

Sygnały w klasie Game (klient):

* Sygnał pojawienia się błędu i jego obsługa w slocie w MainWindow;
* Sygnał o tym, że model został zaktualizowany i obsługa w MainWindow;
* Sygnał zmiany stanu gry obsługiwany w MainWindow i Canvas; dostępne stany aplikacji: NO\_PLAYING, CONNECTING, SENDING\_HELLO, WAITING\_FOR\_PLAYER, PLAYING, GAME\_OVER, SENDING\_GOODBYE, DISCONNECTING.

Sygnały w klasie Server:

* Sygnał zapisania stanu aplikacji w klasie Server do dziennika (audyt) i obsługa w ServerWindow;
* Sygnał o tym, że nowy klient został dodany i obsługa w ServerWindow;
* Sygnał o tym, że klient został usunięty i obsługa w ServerWindow;
* Sygnał o rozłączeniu i obsługa w ServerWindow.

Dodatkowo sygnały i sloty wspierają obsługę komunikacji klient – serwer.

# 4. Interfejs

Nasz program składa się z dwóch części - klienta i serwera, z których każdy ma swój interfejs.

## 4.1. Klient

Interfejs klienta składa się z trzech podstawowych okien: startu, gry i zakończenia.

### 4.1.1. Okno startu

To pierwsze okno, które pojawia się po uruchomieniu klienta. Gracz może tam wybrać swoją nazwę, a także podać adres IP serwera, z którym chce się połączyć oraz numer portu, na którym nasłuchuje serwer. Po wpisaniu wszystkich danych gracz może podjąć próbę połączenia się z serwerem. Jeśli ona się nie powiedzie, może ją przerwać przez kliknięcie w przycisk. Dodatkowo może otworzyć okno z informacjami pomocy oraz zamknąć klienta.

W dolnej części okna znajduje się pasek z informacjami o statusie klienta. Pojawiają się tam również komunikaty od serwera oraz informacje o błędach takich, jak np. podanie nieprawidłowej nazwy gracza.

### 4.1.2. Okno gry

Pojawia się ono, jeśli graczowi uda się połączyć z serwerem gry. W centralnej części znajduje się widok planszy z góry. Są tam elementy planszy, czołgi w takich kolorach, jak grupy graczy, do których one należą oraz wystrzelone pociski. Czołg gracza jest wyróżniony specjalną obwódką.

W prawej części okna znajduje się menu gry oraz podstawowe informacje o rozgrywce. Poniżej nazwy gry znajduje się nazwa gracza w kolorze grupy, do której on należy. Niżej znajduje się aktualny poziom jego życia oraz moc pocisków, którymi strzela. Pod tym są umieszczone informacje o punktach , które gracz zdobył oraz czasie, jaki upłynął od rozpoczęcia rozgrywki. Jest tam też czas maksymalny gry, który został ustawiony na serwerze.

Poniżej tych informacji znajduje się tabela z punktami wszystkich graczy, uczestniczących w rozgrywce. Każdy z nich jest oznaczony kolorem grupy, do której należy.

W dolnej części tego panelu jest przycisk, pozwalający graczowi rozłączyć się z serwerem. Jeżeli to uczyni, tracona jest informacja o jego punktach. Jeśliby znowu połączył się z serwerem podczas tej samej rozgrywki, jego punkty są liczone od 0.

### 4.1.3. Okno zakończenia gry

Wyświetla się ono po zakończeniu gry. W centralnej części znajduje się informacja, czy drużyna gracza wygrała czy przegrała. Zaraz pod nią są podane liczby punktów obydwu drużyn.

Poniżej znajduje się tabela z wynikami wszystkich graczy, którzy byli połączeni z serwerem w chwili zakończenia gry.

## 4.2. Serwer

Posiada on tylko jedno okno. W górnej części można wybrać mapę, na której będzie się toczyć rozgrywka.

Poniżej znajduje się pole wyboru portu, na którym będzie nasłuchiwał serwer, a także czas trwania rozgrywki oraz przycisk uruchamiający lub rozłączający serwer.

Pod tym jest lista graczy, którzy są połączeni z serwerem, wraz z ich identyfikatorami oraz grupami, do których należą.

W dolnej połowie okna znajduje się zapis logów. Są tam niektóre informacje wyświetlane przez serwer. Można tam obserwować wybrane działania serwera.

Na samym dole znajduje się pasek, w którym pojawiają się komunikaty o błędach takich, jak np. próba uruchomienia serwera bez wybrania jakiejkolwiek mapy.

### 4.3. Edytor map

W centralnej części znajduje się interaktywny podglad mapy.

Poniżej jest menu zapisu do pliku, gdzie można podać nazwę pliku. Jeszcze niżej znajduje się przycisk umożliwiajacy wypełnienie całej mapy danym typem elementu. Ułatwia on pracę z edytorem.

Na samym dole znajduje się przycisk, pokazujący lub ukrywający źródło mapy. Można je w każdej chwili podejrzeć, a jego ewentualna modyfikacja przez użytkownika spowoduje zaktualizowanie się podglądu.

# 4. Podział projektu i wyodrębnienie zadań

W pierwszej fazie projektowania gry podzieliliśmy ją na dwie podstawowe części: klient i serwer. Wyodrębniliśmy również wspólne dla obydwu części struktury danych: graczy, pocisków, planszy.

Dodatkowym – oddzielnym – elementem projektu był edytor map, którego szczegółowy projekt pojawił się po zdefiniowaniu struktury plików z mapami.

## 4.1. Klient

Podstawowymi częściami, na które można go podzielić, są:

* główne okno aplikacji – zawiera całość interfejsu, wyświetlanego graczowi;
* manager połączenia - odpowiada za zarządzanie połączeniem;
* obsługa sterowania grą;
* manager sprite'ów - dostarcza elementy planszy na podstawie ich identyfikatorów;
* klient TCP – odpowiada za nawiązanie i obsługę połączenia na poziomie protokołu TCP z serwerem;
* animator – oblicza kolejne klatki animacji;
* plansza – rysowanie planszy.

## 4.2. Serwer

Serwer można podzielić na następujące podstawowe elementy:

* serwer TCP - obsługuje połączenia z klientami na poziomie protokołu TCP;
* model gry – umożliwia przechowywanie i zarządzanie strukturami danych w grze oraz udostępnia je odpowiednim częściom programu;
* kontener akcji – zbiera wszystkie akcje gry, otrzymane od podpiętych klientów i dostarcza je kontrolerowi;
* kontroler – przeprowadza rozgrywkę na podstawie aktualnego stanu planszy i akcji, które wykonali gracze w danym cyklu gry;
* ładowarka plansz – parsuje plik z planszą i na jego podstawie tworzy strukturę zrozumiałą dla innych części programu;
* manager gry – odbiera informacje od graczy o wykonanych przez nich akcjach i odsyła im plansze z aktualnym stanem gry;

# 5. Wybór narzędzi potrzebnych do realizacji

Gra została napisana w języku C++. Już na samym początku zdecydowaliśmy się na wybór biblioteki Qt w wersji 5.5 jako podstawowego dostarczyciela licznych komponentów naszej gry. Przy jej pomocy został stworzony interfejs zarówno klienta, jak również serwera. Dodatkowo w projekcie używaliśmy innych klas tej biblioteki, takich jak np. QTcpSocket, QByteArray, QTest, QVector, QMap, QPoint i inne. Dodatkową zaletą biblioteki Qt jest jej przyjazna struktura, zaprojektowana w oparciu o nowoczesne wzorce programowania.

Wybór biblioteki Qt zdeterminował wybór środowiska, w którym był tworzony nasz projekt. Z powodu licznych narzędzi, wspierających tworzenie programu w Qt naszą grę tworzyliśmy przy użyciu programu QT Creator. Dzięki niemu otrzymaliśmy możliwość graficznego projektowania okien aplikacji oraz rozbudowany debugger, dedykowany pod bibliotekę Qt.

Również kompilacja odbywała się przy pomocy tego środowiska. Wykorzystuje ono kompilator języka C++ z grupy GCC (GNU Compiler Collection).

W celu wygodnego zarzadzania wersjami projektu i zoptymalizowania współpracy w całym zespole tworzącym projekt, zdecydowaliśmy się skorzystać z systemu kontroli wersji GIT. Aby nie musieć samodzielnie zakładać jego serwera, wykorzystaliśmy portal *github.com*, udostępniający za darmo tą usługę.

Aby zapewnić sprawną komunikację miedzy członkami zespołu, założyliśmy własną grupę na portalu społecznościowym *facebook.com*.

# 6. Podział pracy i analiza czasowa

Podział pracy wynikał bezpośrednio z wyodrębnienia poszczególnych elementów naszej gry. Z powodu zależności między różnymi modułami ich powstawanie zostało rozłożone w czasie tak, aby w założonym na początku okresie projekt mógł zostać zrealizowany. Przy dokonywaniu tego planu uwzględniliśmy ewentualne opóźnienia i zdarzenia losowe, aby w trakcie realizacji projektu mógł on zostać skończony w terminie.

## 6.1. Podział pracy

Każdy z członków zespołu otrzymał zdefiniowane na początku zadania i był odpowiedzialny za określone elementy projektu:

1. Herman Katarzyna - odpowiedzialna za wyświetlanie prezentację gry po stronie klienta, animacje elementów planszy wykonywane na podstawie zmian między kolejnym stanami planszy, przychodzącymi z serwera;
2. Drzazga Kamil - jego zadaniem było połączenie modułów gry w działającą całość, współprojektant szkieletu systemu; odpowiedzialny za dokumentację kodu;
3. Bartocha Łukasz - odpowiedzialny za przeprowadzanie rozgrywki na serwerze, sprawdzanie poprawności ruchów graczy i aktualizowanie planszy gry;
4. Bajorek Tomasz – autor architektury klient - serwer gry, odpowiedzialnej za transmisję danych oraz obsługę komunikacji miedzy serwerem i klientami; współprojektant szkieletu systemu; odpowiedzialny za dokumentację projektu;

Niektóre części gry powstawały przy współpracy kilku osób.

## 6.2. Analiza czasowa

Rozkład czynności w czasie przedstawiał się następująco:

* projekt szkieletu aplikacji - początkowa faza projektu, określająca strukturę całości aplikacji wraz z podziałem na moduły oraz komunikacją między nimi (październik 2015);
* stworzenie architektury klient – serwer, obsługa komunikacji pomiędzy nimi i synchronizacji przesyłanych komunikatów, wykrywanie i obsługa ewentualnych błędów transmisji (październik/listopad 2015);
* projekt interfejsów klienta i serwera (listopad/grudzień 2015)
* zaprojektowanie dokładnej struktury klienta i serwera (grudzień 2015);
* praca nad konkretnymi modułami - równolegle nad klientem i serwerem: (grudzień 2015 / styczeń 2016):
  1. klient:
     + zaprojektowanie i implementacja algorytmów animowania gry;
     + testowanie działania klienta;
  2. serwer:
     + tworzenie stanów gry w kolejnych cyklach rozgrywki;
     + ładowanie map z pliku;
     + testowanie działania serwera;
* integracja modułów (styczeń 2016);
* testowanie aplikacji i poprawa błędów (styczeń 2016);
* zebranie dokumentacji i przygotowanie projektu do oddania (styczeń 2016);

Tworzenie dokumentacji odbywało się przez cały czas trwania projektu:

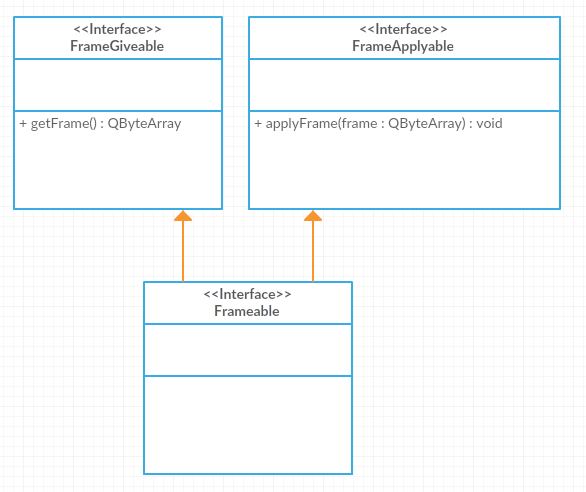
1. zebranie wymagań (październik 2015);
2. projekt systemu (październik 2015);
3. dokumentacja kodu (październik 2015 - styczeń 2016);
4. dokumentacja testów (styczeń 2016);
5. łączenie wszystkich elementów dokumentacji (styczeń 2016);

# 7. Mechanizm ramek w komunikacji klient – serwer

Ramki są to tablice znaków z zakodowaną w pewien sposób informacją. Są one transmitowane przez sieć protokołem TCP. Za obsługę wysyłania bądź odbierania tych ramek odpowiadają klasy TcpClient i TcpServer, które znajdują się odpowiednio na kliencie i na serwerze.

Ramki są reprezentowane jako QByteArray, który opakowuje ciąg znaków i oferuje przydatny interfejs do manipulowania nimi.

W celu usprawnienia tworzenia ramek bądź ich aplikowaniu na obiekty utworzony został pomocniczy interfejs: Frameable (realizowany w C++ jako klasa abstrakcyjna). Dziedziczy on po FrameGiveable i po FrameApplyable. Struktura tych interfejsów jest nastepująca:



Jeżeli wiemy, że na podstawie obiektu danej klasy będzie można utworzyć jakąkolwiek ramkę, to klasa ta dziedziczy po FrameGiveable. Natomiast gdy wiemy, że jakaś ramka może zmieniać wartości obiektów jakiejś klasy, to klasa ta dziedziczy po FrameApplyable.

Poniżej znajduje się lista klas, które dziedziczą po tych interfejsach:

* Player – dziedziczy po Frameable, gdyż musi mieć możliwość utworzenia ramki na podstawie swoich pól (id gracza, jego grupa, ilość punktów, poziom życia, pozycja (x, y), kierunek zwrotu czołgu, imię gracza, czy żyje i jak nie żyje, to przez ile okresów gry). Informacje te są potrzebne do tego, by serwer mógł wysłać ramkę ze stanem planszy co określony interwał czasu (u nas: 200 ms). Klasa ta potrafi przyjmować ramkę – potrzebne to jest dla klienta, który musi mieć informację o każdym z graczy by odpowiednio namalować planszę i to, co na niej się dzieje.
* Shot – dziedziczy po Frameable, bo musi mieć możliwość utworzenia ramki ze swoich pól (czyli o informacjach o pocisku: id pocisku, id przypisanego gracza, pozycja startowa (x, y), kierunek lotu, czas trwania lotu i siła uderzenia). Podobnie jak dla klasy Player, informacje są potrzebne do wysłania przez serwer stanu planszy. Klasa potrafi przyjąć ramkę i zaaplikować ją na wspomniane pola bo klient musi posiadać informacje o wystrzałach by poprawnie je namalować.
* Board – również typu Frameable. Klasa zawiera tablicę 2D (a dokładnie QVector<QVector<BoardElement> >) z polami planszy, które można w dużym uproszczeniu można traktować jak tablicę z polami zawierającymi id elementu planszy. Dziedziczenie po Frameable jest konieczne, bo serwer musi rozsyłać ramki z tymi informacjami co 200ms (getFrame()) i klient musi je odbierać i interpretować (applyFrame()).
* Model – dziedziczy po Frameable. Jest to klasa zbiorcza na planszę, graczy i pociski. Ona jest pośrednikiem do Player, Shot i Board. Przyjmuje ramki i je produkuje poprzez wywołanie applyFrame() i getFrame() na Board, Player i Shot.
* PlayerAction – typu Frameable. Dziedziczenie po tym interfejsie jest konieczne, gdyż informacje o akcjach gracza (kierunek poruszenia się gracza i informacja o tym, czy pocisk został wystrzelony) muszą zostać wysłane z klienta na serwer, czyli klient musi utworzyć ramkę poprzez getFrame() i serwer musi jak zaaplikować poprzez applyFrame().
* Game – typu FrameApplyable ze względu na to, by klient mógł odebrać ramkę i od serwera i ją przetworzyć (m. in poprzez wywołanie applyFrame() na jednym z modelu). Klasa Game zawiera się w kliencie i zbiera wszystkie informacje o stanie gry. Posiada dwa egzemplarze klasy Model: model1 i model2. Jest to konieczne do płynnej animacji – chcemy mieć możliwość płynnie animować obiekty niezależnie od tego, jak często przyjdzie ramka z serwera. Do tego płynnego przejście potrzebne są 2 modele.
* NetworkManager – klasa, która przyjmuje wszystkie ramki od serwera w kliencie. Dlatego też dziedziczy po FrameApplyable i nadpisuje metodę applyFrame().

# 8. Organizacja klas

Diagram z narysowanymi na nim relacjami między poszczególnymi klasami aplikacji został dołączony jako załącznik do dokumentacji.

## 8.1. Klient

Podstawową klasą, która zbiera akcje użytkownika jest MainWindow. Dziedziczy ona po QWidget i jest powiązana z plikiem mainwindow.ui, który zawiera interfejs użytkownika.

Klasa ta trzyma jako swoje pole obiekty typu Canvas, NetworkManager i Game. Canvas to obiekt klasy QWidget, na którym odbywa się malowanie. NetworkManager zarządza ramkami i komunikacją z serwerem. Game jest odpowiedzialny za zbieranie informacji o grze.

Klasa Canvas posiada jako właściwość klasę Drawer, która jest wykorzystywana do malowania planszy (na płótnie Canvas). Klasa Drawer zawiera klasę pomocniczą (helper) Sprites, która dostarcza interfejs do obrazków używanych do namalowania planszy.

NetworkManager posiada klasę TcpClient. Zastosowany został wzorzec projektowy o nazwie Fasada, gdyż ukrywany jest dostęp do TcpClient, który zawiera funkcjonalność nieco niższego poziomu obsługi naszej aplikacji.

Klasa Game posiada wszystkie informacje na temat gry. Zawiera m. in. 2 obiekty typu Model w celu przechowania aktualnego stanu gry i 1 obiekt typu PlayerAction w celu przechowania akcji wykonywanych przez użytkownika.

## 8.2. Serwer

Główną klasą na serwerze jest ServerWindow, która jest typu QWidget. Powiązana jest ona z zaprojektowanym interfejsem gui w programie QtCreator. ServerWindow zawiera w sobie obiekt typu Data w celu składowania danych na serwerze (model + akcje użytkownika) i klasę Server, która jest menagerem sterującym wszystkim co dzieje się w programie.

Klasa Server opakowuje klasę TcpServer służącą do realizacji serwera TCP. Zawiera także klasę służącą do Ładowania plików z mapami – Loader i klasę, która steruje logiką całej gry – Controller.

## 8.3. Klasy wspólne

Klasami wspólnymi dla klienta i serwera jest Model (zawiera informacje o planszy, pociskach i czołgach), PlayerAction (przechowuje ruchy gracza). Model zawiera Board, Shot, Player i BoardElement, które też są współdzielone. Dodatkowo współdzielony jest typ wyliczeniowy DIRECTION, który przechowuje kierunek lub jego brak.

# 9. Algorytmy użyte w programie

Przedstawione tutaj algorytmy w całości zostały opracowane przez nasz zespół.

## 9.1. Animacja ruchów czołgów i pocisków

Animacje (przesuwanie czołgów i pocisków) są rysowane przy użyciu 3 informacji – stanu poprzedniego, stanu następnego oraz współczynnika przejścia z jednego stanu w drugi. W kolejnych przerysowaniach obrazu do położenia ze stanu poprzedniego dodawana jest różnica między oboma stanami przemnożona przez współczynnik. Ponieważ jest on wyliczany dla każdego rysowania, kolejne przerysowania czołgu lub pocisku dają wrażenie płynnego ruchu.

## 9.2. Animacja wybuchów

Animacja wybuchów jest zrealizowana w inny sposób. Dzięki osobnej klasie, w której przechowywany jest stan animacji, są rysowane kolejne sprite'y. Przy każdym narysowaniu klatki wybuchu, w klasie zmieniana jest faza animacji, a gdy animacja się skończy (dojdziemy do pustej klatki) jest ona usuwana. Konieczność narysowania animacji wybuchu wykrywana jest poprzez zauważenie zniknięcia pocisku - porównanie stanu poprzedniego z następnym.

# 10. Dokumentacja kodu

Kod programu został udokumentowany zgodnie ze standardem, stosowanym przez generator dokumentacji *doxygen*. Wygenerowana przez niego dokumentacja kodu jest dołączona do projektu.

# 11. Testy aplikacji

Aplikacja była poddawana testom na każdym etapie jej tworzenia.

## 11.1. Testy jednostkowe

Kody testów jednostkowych znajdują się w katalogu z kodem programu. W testach wykorzystywano klasę *QTest*.

## 11.2. Testy modułów

### 11.1.1 Klient

Przeprowadziliśmy testy klienta gry:

|  |  |  |
| --- | --- | --- |
| Przedmiot testów | Rezultat | Uwagi |
| Malowanie tła | OK |  |
| Malowanie klatek wybuchu | Naprawione | Błąd w obliczaniu numeru klatki |
| Malowanie czołgu aktywnego gracza | OK |  |
| Malowanie lotu pocisku | Naprawione | Błąd w obliczeniu odległości: brak wartości bezwzględnej |
| Obracanie malowanych obiektów | OK |  |
| Wykrywanie akcji użytkownika | OK |  |
| Wysyłanie ramek | Naprawione | Błąd w tworzeniu ramki żądania połączenia |
| Wyświetlanie komunikatów | OK |  |

### 11.1.2. Serwer

Przeprowadziliśmy testy serwera gry:

|  |  |  |
| --- | --- | --- |
| Przedmiot testów | Rezultat | Uwagi |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 11.3. Testy systemowe i funkcjonalne

Aplikacja po wydaniu pierwszych wersji klienta i serwera była przez nas wielokrotnie testowana. Testy te polegały na uruchomieniu obydwu programów w trybie debugowania i przeprowadzaniu rozgrywek. Do projektu zostało dołączone kilka przykładowych map, na których prowadziliśmy testy.

Podczas testów sprawdzaliśmy następujące przypadki testowe:

|  |  |  |
| --- | --- | --- |
| Przypadek testowy | Rezultat | Uwagi |
| Próba połączenia się tylko jednego gracza | OK | Jeden gracz nie mógł rozpocząć gry - oczekiwał na połączenie drugiego; |
| Próba połączenia się dziewiątego gracza | OK | Dziewiąty gracz nie mógł się połączyć, otrzymywał stosowny komunikat |
| Wpisanie zbyt długiej nazwy użytkownika | Naprawione | Nałożono ograniczenie na program, aby zwracał odpowiedni komunikat w przypadku zaistnienia tej sytuacji |
| Użycie w nazwie użytkownika polskich znaków | Naprawione | Program zwraca komunikat o błędnej nazwie użytkownika |
| Strzelanie do gracza | OK | Gracz po trafieniu w niego pocisków traci punkty życia |
| |  |  |  | | --- | --- | --- | | Zabicie gracza i jego odrodzenie na planszy | OK | Gracz po utracie całego życia ginie i za 10 sekund odradza się w losowym miejscu; miejsce to jest puste: nie ma tam ani innego gracza ani przeszkody | | Zderzenie się czołgu z przeszkodą | Naprawione | Poprawa funkcji liczącej odległości między czołgiem a przeszkodą | |  |  |
| Obliczanie czasu trwania gry | Naprawione | Program początkowo liczył czas gry od podłączenia się pierwszego gracza, bez czekania na połączenie się drugiego |
| Naliczanie punktów | OK | Punkty naliczają się zgodnie z przyjętą punktacją |
| Zderzanie się czołgu z czołgiem | OK | Czołgi nie najeżdżają na siebie |
| Wyłączanie klienta bez jego rozłączenia z serwerem | Naprawione | Serwer początkowo źle usuwał gracza, który nie rozłączył się przy pomocy ramki z żądaniem rozłączenia |
|  |  |  |

# 12. Podsumowanie

## 12.1. Wdrożenie

Aplikację udało się poprawnie uruchomić na komputerze z systemem Linux. Przeprowadziliśmy kilka rozgrywek i nie zauważyliśmy błędów w jej funkcjonowaniu.

Do poprawnego działania zarówno klient jak i serwer wymagają, aby komputer, na którym zostały uruchomione, był podłączony do sieci. Jeżeli będzie to Internet, serwer i klienci mogą znajdować się na całym świecie. W tym przypadku należy jednak szczególnie uwzględnić możliwe opóźnienia, które wpłyną na jakość działania gry. Jeżeli serwer zostanie uruchomiony w sieci lokalnej a komputer nie będzie posiadał publicznego adresu IP, będzie można się z nim połączyć tylko z tej sieci.

Możliwe jest uruchomienie na jednym komputerze zarówno klienta, jak i serwera. Wówczas klient ten będzie się łączył z lokalnym adresem IP.

## 12.2. Wnioski

Projekt udało się zrealizować w wyznaczonym terminie. Wszyscy członkowie zespołu wywiązali się z przydzielonych im obowiązków.

Aplikacja spełnia wszystkie podstawowe założenia, ustalone przed rozpoczęciem projektu. Z założeń dodatkowych udało się stworzyć zniszczalne przeszkody na planszy, które czołgi mogą burzyć podczas trwania rozgrywki. Pozostałe funkcjonalności okazały się dla nas zbyt czasochłonne. Projekt ten nie był naszym jedynym zajęciem w tym semestrze i nie udało się nam zrealizować wszystkich wymagań dodatkowych. Naszym głównym celem stało się więc solidne wykonanie wszystkich podstawowych wymagań.

Dodatkowym wymaganiem, nieuwzględnionym na początku projektu, stała się możliwość ładowania map z pliku. Umożliwiło to tworzenie praktycznie dowolnych plansz zbudowanych z dostępnych elementów. Wprowadzenie tej funkcjonalności do projektu poskutkowało napisaniem edytora map, w celu ich łatwiejszego tworzenia.

Projekt jest przygotowany do dalszego rozwoju. W przyszłości można dodać niezrealizowane do tej pory funkcjonalności, takie jak dodatkowe bonusy, pojawiające się losowo na planszy (np. dodatkowe punkty, regeneracja życia, wzrost siły pocisków, chwilowa ochrona przed pociskami), nowe rodzaje broni o różnej sile.

Pomysłem, który rozważaliśmy było też uwzględnienie wszystkich graczy z przeciwnej drużyny, którzy trafili zabitego gracza przy dodawaniu punktów za jego zabicie (asysty). Innym pomysłem byłoby wprowadzenie możliwości dołączenia do gry jako obserwator – bez uczestniczenia w grze.

Z aspektów technicznych można by jeszcze bardziej skompresować ramki – pojedyncze znaki zastąpić bitami. Ponadto można by spróbować umożliwić zmianę rozmiarów okna tak, aby dostosowywało się ono do rozmiarów ekranu użytkownika - ze skalowaniem interfejsu gry.

Kolejnym, rozważanym przez nas pomysłem, byłoby dodanie nowych elementów planszy, np. lasu, w którym gracz mógłby się ukryć za drzewami oraz bagien, spowalniających poruszanie się gracza. Jeszcze jednym elementem gry urozmaicającym rozgrywkę, mogłoby być dodatnie opcji "Zachmurzenie", po której włączeniu (przycisk na serwerze) w losowym czasie i miejscu pojawiałyby się na planszy chmury, które przysłaniałyby jej część wszystkim graczom.

Jesteśmy zadowoleni z wykonanego projektu i jeżeli tylko pozwoli nam na to czas, będziemy go nadal rozwijać.