Kostecki Kacper
300236

Kulpa Katarzyna
300248

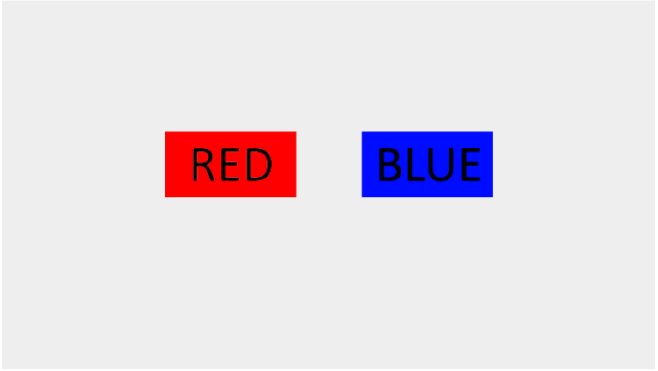
Projekt PROZ sprawozdanie

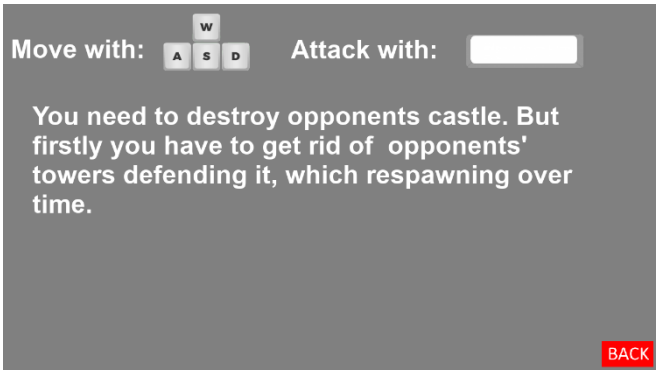
Tower defense multiplayer – gra dla dwóch lub więcej graczy.

Ostateczna wersja projektu różni się lekko od planowanej wersji. Zasady ostatecznej wersji:

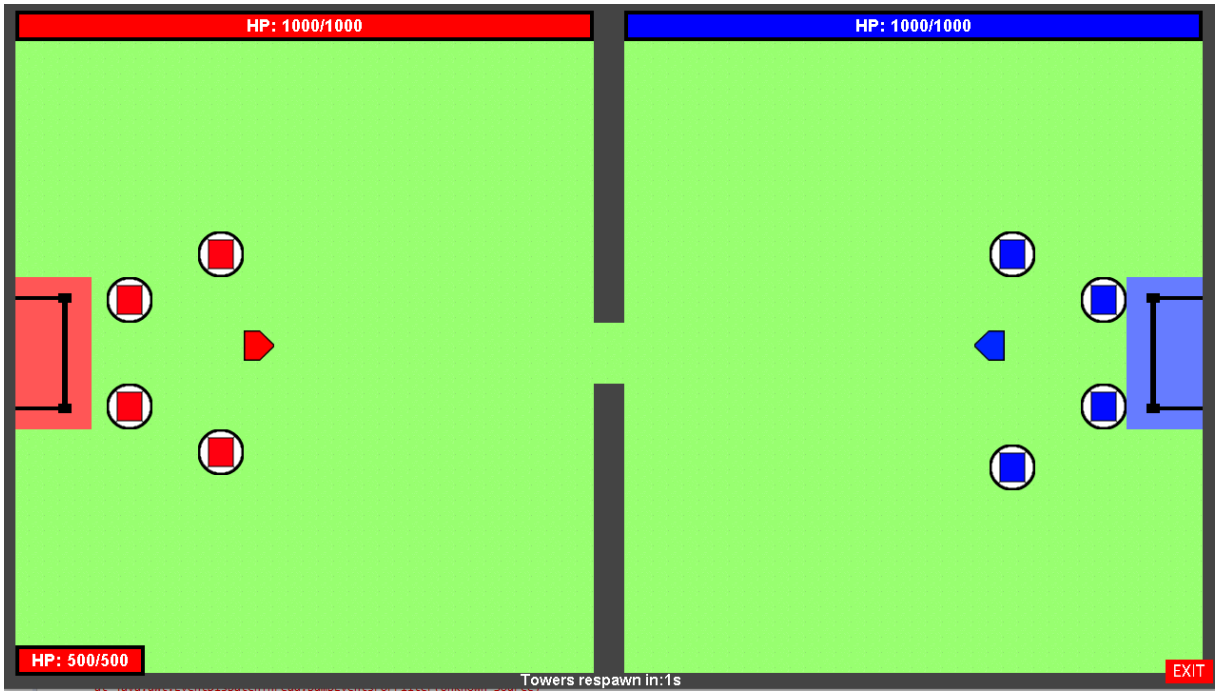
Zaczynając grę wyświetlone zostaje GUI.

Aby zobaczyć zasady gry należy nacisnąć HOW TO PLAY, aby zacząć grę START, a by wyjść EXIT. Kiedy wybierzemy opcję START musimy dokonać wyboru w której drużynie chcemy grać (RED lub BLUE). Następnie wybieramy czy chcemy utworzyć nowy serwer lub czy chcemy dołączyć do już istniejącego serwera. W następnym kroku jeżeli zadecydowaliśmy o dołączeniu do serwera musimy podać jego ip. Następnie podajemy naszą nazwę użytkownika. Po jej podaniu przechodzimy do właściwej rozgrywki. Jeżeli jesteśmy hostem i żaden z przeciwników jeszcze nie dołączył do rozgrywki, wyświetla się napis, iż musimy poczekać na przeciwnika. Dopóki nikt nie dołączy do drużyny przeciwnej gracze nie mogą się ruszać, atakować itd.

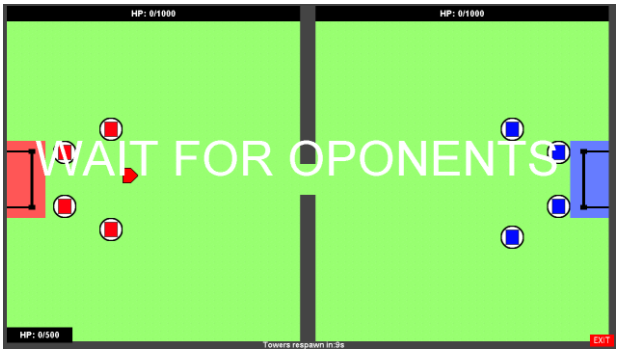


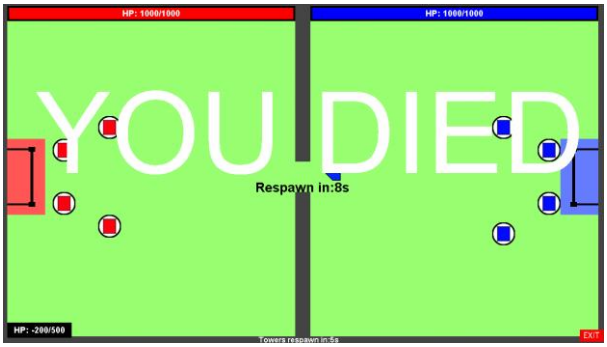


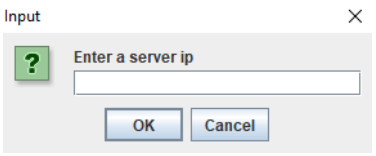
Akcja rozgrywa się na planszy, na której każda drużyna ma swoją bazę oraz 4 wieże stojące przy niej.

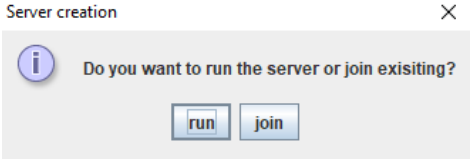


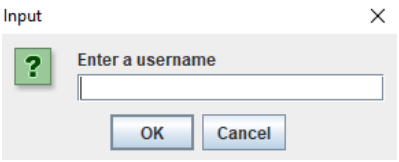
Gracze poruszają się przyciskami WSAD. Ponad to mogą strzelać do siebie poprzez kliknięcie spacji, każdy trafiony w przeciwnika pocisk odbiera mu 100 jednostek życia. Użytkownik ma łącznie 500 jednostek życia (aktualny stan swojego zdrowia może zobaczyć w rogu), kiedy je straci pojawia mu się komunikat, że umarł i musi poczekać 10 sekund, aby móc grać dalej. Użytkownicy mogą się leczyć poprzez powrót do bazy – pola odpowiedniego koloru drużyny leczą każdego gracza stojącego na nich 10 punktów życia na sekundę.











Gra kończy się w momencie, kiedy któraś z drużyn zniszczy bazę przeciwnika. Można ją zacząć niszczyć tylko wtedy, kiedy wszystkie wieże są zniszczone. Każdy trafiony pocisk zabiera 100 jednostek życia bazy z 1000 możliwych. Utrudnieniem jest to, że wszystkie wieże odbudowują się po upływie konkretnego czasu, który wyświetla się na dole planszy. Kiedy któraś z drużyn zniszczy bazę wyskakuje komunikat o wygranej, natomiast drużynie przegranej wyskakuje komunikat o przegranej. Paski zdrowia obydwu baz są pokazane na górze planszy.Kiedy gracz chce opuścić grę może nacisnąć przycisk EXIT lub po prostu zamknąć okno gry.

Przydział obowiązków:

- Kacper Kostecki:
  - Podstawy mechaniki gry: system kolizji, generowanie mapy, obsługa jednostek (entity),
  - Stworzenie gui oraz wszelkich aspektów wizualnych gry,
  - Stworzenie silnika gry – obsługa „renderowania" oraz „ticków",
  - Testy jednostkowe.
- Katarzyna Kulpa:
  - Obsługa sieciowa gry: stworzenie modułu klienta oraz serwera,
  - Obsługa przesyłania pakietów między serwerem oraz klientami,
  - Implementacja mechaniki gry na platformę sieciową – odpowiednie dopasowanie mechaniki gry, aby było możliwe granie sieciowe w czasie rzeczywistym,
  - Usuwanie wszelkich błędów, które wystąpiły w trakcie implementacji nowych mechanik.

Opis wykorzystanych rozwiązań w kodzie źródłowym:

Najważniejszym elementem kodu źródłowego gry jest klasa Game. To w niej inicjalizowany jest główny wątek odpowiedzialny za działanie gry oraz tzw. „game loop", dzięki któremu nie wykonujemy określonych działań „renderowania" oraz „tickowania" cały czas, a jedynie przez pewną określoną liczbę razy na sekundę, co znacząco poprawia wydajność. Wszelkie ważne zmienne lub metody określonych klas, do których chcielibyśmy się dostać z dowolnego miejsca kodu są przechowywane w klasie Handler.

Plansza gry jest tworzona na bazie „tile", czyli pojedynczych, powtarzalnych kwadratów, które razem tworzą całą mapę. Dzięki takiemu rozwiązaniu możemy z łatwością modyfikować planszę dodając kolejne przeszkody lub wieżyczki.

Wszelkie jednostki są tworzone według tej samej koncepcji, dlatego została stworzona abstrakcyjna klasa Entity. W przypadku przemieszczających się jednostek została również stworzona klasa Creature, która dziedziczy po klasie Entity oraz obsługuje poruszanie się, czego główna klasa jednostek nie obsługuje, ponieważ nie każda jednostka musi się przemieszczać po planszy.

Wszystkie jednostki są przechowywane w menedżerze, który jest przypisywany do mapy. Każdy gracz posiada swój własny menedżer. Wszelkie wejścia od gracza (klawiatura, myszka, okno) również są obsługiwane w odpowiednich menedżerach.

Obsługa sieciowa gry:

- Wszelkie sieciowe klasy są przechowywane w paczce net,
- Klasa GameServer obsługuje serwer stworzony przez gracza – zostaje stworzony oddzielny wątek, który przetwarza wszelkie aspekty dotyczące serwera (np. odbieranie i wysyłanie pakietów). Jest ona wywoływany tylko i wyłącznie przez gracza, który chce uruchomić serwer, czyli zostać hostem gry,
- Każdy gracz, który chcą dołączyć do rozgrywki, wywołuje nowy wątek w klasie GameClient, w której obsługiwane są wszystkie aspekty rozgrywki po stronie klienta np. tworzenie pocisków wystrzelonych przez przeciwników lub dodawanie samych przeciwników do gry.
- Klient użytkownika pobiera wszystkie przetworzone rzeczy od głównego wątku gry wywoływanego w głównej klasie Game.

GameClient oraz GameServer komunikują się między sobą za pomocą protokołu UDP wysyłając informacje w pakietach. Każdy pakiet to string zawierając na samym początku swoje ID oraz wszelkie potrzebne informacje, które muszą dotrzeć od jednego gracza do wszystkich pozostałych.

Jeżeli użytkownik dołącza do gry to jest wysyłany pakiet login, w którym zawarta jest m.in. nazwa użytkownika. Natomiast w przypadku zerwania połączenia wysyłany jest pakiet disconnect, w celu poprawnego usunięcia postaci użytkownika z serwera.

Aspekty, które mogłyby zostać zaimplementować w lepszy sposób:

- Przemieszczenia jednostek

Również poprzez pakiety obsługiwane jest poruszanie się przeciwnika oraz atakowanie. Niestety w tym przypadku popełniliśmy błąd implementując poruszanie się jednostek bazujące na klatkach gry. W wyniku tego, kiedy graczowi spada liczba klatek na sekundę (fps) porusza się on wolniej. Tak samo w przypadku spadku jakości połączenia między klientem oraz serwerem – mniej pakietów dociera do gracza, a co za tym idzie mniej razy jednostka się przemieszcza, ponieważ jest to zależne od liczby pakietów. Lepszą praktyką byłoby zastosowanie poruszania się bazującego na czasie: jednostka zamiast przemieszczać się np. 3 piksele na klatkę, powinna przemieszczać się 90 pikseli na sekundę, co przy 30 fpsach dałoby taki samo rezultat, ale w przypadku spadku wydajności komputera użytkownika nie miałoby w pływu na pozycję gracza. Tak samo w przypadku spadku jakości połączenia – mniejsza ilość dostarczonych pakietów nie oznaczałaby wolniejszego przemieszczania się użytkowników, a jedynie skoki w przemieszczaniu, co jest naturalnym zjawiskiem w grach komputerowych, w przypadku słabego połączenia. Z powodu początkowej złej implementacji, nie była możliwa zmiana tej mechaniki bez gruntowego zmieniania całej mechaniki poruszania się w grze, w której skład wchodzą wszelkie kolizje.

- Obsługa konkurencji

Brak specjalnej ochrony przeciwko jednoczesnemu modyfikowaniu tej samej jednostki, co w skrajnych przypadkach może doprowadzić do fatalnych błędów i wyłączania się serwera. Błąd ten występuje rzadko, jedynie w wyjątkowych sytuacjach, dlatego z początku nie zwróciliśmy na niego uwagi. Niestety na późniejszym etapie tworzenia gry zaimplementowanie dostępu do menedżera jednostek w taki sposób, aby zniwelować problem konkurencji stało się bardzo trudne. Z uwagi na to, że błąd ten występuje rzadko oraz tego, iż robiliśmy projekt we dwie osoby, dlatego nie mieliśmy, aż tyle czasu by poświęcić mu wymaganą uwagę, postanowiliśmy pominąć ten błąd, aby skupić się na naprawianie bardziej dokuczliwych błędów.

- Lepsza synchronizacja sieciowa rozgrywki

Z powodu formy w jakiej wysyłane są pakiety między użytkownikami, może dochodzić do desynchronizacji rozgrywki między nimi. Prowadzi to do dużych błędów takich jak: inna pozycja gracza u każdego użytkownika przez co nie można go trafić. Pociski docierają do celów w innym czasie przez co w przypadku bliskiego czasu odnowienia się wieżyczek może doprowadzić do sytuacji, w której u jednego użytkownika gra się skończy, ponieważ baza zostanie zniszczona, a u innego rozgrywka będzie nadal trwała. Rozwiązać ten problem można by, poprzez zmodyfikowanie obsługi jednostek, tak aby przesyłać konkretniejsze informacje o pozycji danej jednostki, a ostateczne dotarcie do celu oraz zniszczenie go byłoby przetwarzane przez serwer i wysyłane do klientów. W wyniku takiej zmiany jeżeli doszłoby do opóźnień w połączeniu, a co za tym idzie do desynchronizacji rozgrywki, dane jednostki nie zostałyby zniszczone u nikogo lub zniszczone u wszystkich – zależnie od tego jakie informacje dotarły do serwera. W połączeniu z usprawnionym przemieszczaniem się jednostek powinno to spowodować ograniczenie desynchronizacji do skrajnych przypadków.

- Lepsza obsługa logowania i wyrzucania z serwera użytkownika

Mieliśmy w zamiarze dodanie do rozgrywki ograniczenia ilości użytkowników, lecz było to bardzo trudne do implementacji w obecnej formie obsługi sieciowej. Bardzo przydatne byłaby lepsza funkcjonalność wysyłania pakietów z serwera do pojedynczych klientów. Osiągnąć to można by poprzez np. nadawanie unikatowych identyfikatorów użytkownikom. Niestety pomyśleliśmy o możliwych niedogodnościach wywołanych przez słabe identyfikowanie graczy za późno by móc sensownie to usprawnić bez poważnych zmian w obsłudze sieciowej gry.

Testy jednostkowe głównie skupiają się na kolizji jednostek. W wyniku błędów w ładowaniu tekstur, na potrzeby testów stworzyliśmy w nich testowe jednostki specjalnie przeznaczone do tych celów. Sprawdzaliśmy w nich czy pozycja jednostki jest poprawnie obliczana w metodach dotyczących kolizji jednostek. Niestety testy ograniczyliśmy jedynie do testów jednostek, ponieważ implementując inne aspekty gry nie zwracaliśmy dużej uwagi na możliwość ich testowania jednostkowego, co uniemożliwiło na późniejszym etapie stworzenie sensownych testów, które nie wymuszałyby modyfikacji kodu gry.

Dokładny opis kodu źródłowego to komentarze w nim zawarte:

Kod źródłowy: https://github.com/kkosteck/PROZ

Możliwości dalszego rozwoju gry:

Staraliśmy się stworzyć kod źródłowy w taki sposób, aby wszelkie dodatkowe mechaniki były łatwe do zaimplementowania. Mogłyby to być między innymi:

- Różne rodzaje wieżyczek dla których została stworzona abstrakcyjna klasa i co zaimplementowaliśmy pośrednio tworząc bazę przeciwnika, która technicznie jest również wieżyczką,
- Ulepszenie ataków gracza poprzez zaimplementowanie innych rodzajów pocisków lub nawet innych rodzajów ataków np. atak samonaprowadzający poprzez implementacje algorytmów poszukiwania ścieżki,
- Różne możliwości zwycięstwa – oprócz zniszczenia bazy przeciwnika rozgrywka mogłaby odbywać się w ograniczeniu czasowym po którego upływie wygrywałby gracz na podstawie specjalnie określonej zasady np. liczby zniszczonych wieżyczek,
- Różne rodzaje map: mimo tego, iż gra używa jedynie jednej mapy to kod źródłowy został stworzony tak, aby nie było problemu z dodaniem innych rodzajów plansz, które różniłyby się np. wielkością lub rozłożeniem wieżyczek
- Licznik „pingu” graczy, dzięki któremu można by podejrzeć w jakim stanie jest nasze połączenie z serwerem i czy nie ma z nim żadnych problemów.