AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA ELEKTRONIKI

PRACA DYPLOMOWA INŻYNIERSKA

Implementacja dwuwymiarowej gry komputerowej z algorytmami sztucznej inteligencji

The implementation of a two dimensional computer game with artificial intelligence algorithms

Autor: *Dawid Waksmundzki*
Kierunek studiów: *Elektronika i Telekomunikacja*
Opiekun pracy: *dr inż. Bartosz Ziółko*

Kraków, 2016r.

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim”, oświadczam, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

Niniejszą pracę dedykuje mojej rodzinie, dziewczynie oraz niedawno zmarłej babci.

Spis treści

Wstęp

Pomysł wykonania gry zrodził się na początku studiów. Chciałem stworzyć własną grę 2D oraz rozwinąć swoje umiejętności programowania. Napisana przeze mnie gra jest typu zręcznościowego, polegająca na sterowaniu przez gracza czołgiem i walce z innymi czołgami oraz na obronie swojej bazy, aby nie dopuścić do zniszczenia jej przez wroga. Wszystkie komunikaty gry są zrealizowane w języku angielskim. Do jej stworzenia wykorzystałem język C++ oraz biblioteki Allegro 5, z którymi miałem już wcześniej do czynienia. Niniejsze biblioteki wspierają systemy: Windows, Linux, Mac OSX, iOS oraz Androida. Zapewniają również obsługę plików dźwiękowych, dodatkowych czcionek, plików INI, obsługę myszki oraz klawiatury, wyświetlanie bitmap. Biblioteki Allegro 5 są bardzo intuicyjne oraz przyjazne dla programisty. Najsłynniejszą grą stworzoną z ich wykorzystaniem jest Icy Tower.

Rozdział pierwszy zawiera część opisową pracy, a więc opis jej celów oraz ich praktyczną realizację w programie. Są to m.in.:

a) Sztuczna inteligencja (algorytm A* służący do znajdowania optymalnej ścieżki do określonego położenia, algorytmy wykorzystywane do poszukiwania gracza oraz algorytmy obsługujące zachowanie przeciwników).

b) Obsługa rozgrywki (założenia techniczne, zasady rozgrywki, umiejętności gracza oraz przeciwników, pauzowanie rozgrywki, bonusy, losowe generowanie map oraz losowe odradzanie przeciwników i przeszkód).

c) Wykrywanie kolizji (opisanie wykrywania kolizji pomiędzy graczem, przeciwnikami, przeszkodami oraz rakietami).

d) Udźwiękowienie gry.

e) Animacje eksplozji (z wykorzystaniem bitmap).

f) Tryby gry (do wyboru przetrwanie, obrona bazy oraz kampania składająca się z sześciu misji).

g) Dodatki do gry (ranking najlepszych wyników dla dziesięciu graczy zawierający pseudonim gracza oraz jego wynik punktowy, główne menu gry obsługiwane za

pomocą myszki z wykorzystaniem bitmap przycisków, różne typy przeciwników, panel boczny z informacjami dla gracza).

Rozdział drugi zawiera dokumentację techniczną programu z opisanym działaniem oraz zastosowaniem najważniejszych funkcji. Pozwala ona szybko zapoznać się z budową aplikacji, klasami oraz funkcjami w niej występującymi. Może również być wykorzystana do ewentualnej rozbudowy programu.

Cel pracy

Celem mojej pracy inżynierskiej było stworzenie dwuwymiarowej gry z inteligentnymi przeciwnikami, którzy potrafiliby stanowić wyzwanie dla gracza. Dodatkowo chciałem, aby zawierała elementy obecne w większości gier. Mam tutaj na myśli:

- dodatkowe umiejętności dla gracza i przeciwników oraz bonusy dla gracza,
- różne rodzaje przeciwników,
- różne tryby gry,
- ranking najlepszych wyników,
- menu obsługiwane za pomocą myszki,
- animacje eksplozji.

Aby osiągnąć obrane cele musiałem na początek wybrać język programowania oraz biblioteki, które pomogą mi wyświetlać bitmapy na ekranie, odtwarzać dźwięk, obsługiwać myszkę oraz obsługiwać różne rodzaje czcionek oraz pliki INI. Do tego celu wybrałem wieloplatformowe biblioteki Allegro 5 oraz język C++. Głównym kryterium wyboru bibliotek oraz języka była ich wcześniejsza znajomość. Dodatkowo biblioteki Allegro 5 są kompatybilne z różnymi systemami operacyjnymi, co pozwala małym nakładem pracy przeportować grę na inne systemy niż Windows.

Kolejnym wyborem był algorytm służący do poruszania się przeciwników i znajdowania trasy do poszukiwanego miejsca na mapie. Wybór padł na algorytm A*. Zapewnia on znalezienie najkrótszej możliwej trasy oraz jest często używany w grach komputerowych.

Następnie musiałem zaplanować jakie zachowania przeciwników zapewniłyby im odpowiednią inteligencję. W tym celu zaimplementowałem własne funkcje odpowiedzialne za zachowanie przeciwników w różnych scenariuszach, które mogą ich spotkać podczas rozgrywki. Dodatkowo musiałem zostawić pewien margines, żeby przeciwnikom też zdarzało się popełnić błąd.

Rozdział 1

Część opisowa

W tym rozdziale opiszę sposób realizacji programu z punktu widzenia teoretycznego.

1.1 Sztuczna inteligencja

Najważniejszym celem pracy było stworzenie inteligentnych przeciwników, którzy stanowiliby wyzwanie dla gracza. Aby tego dokonać, musiałem skorzystać z algorytmu służącego do wyszukiwania optymalnej ścieżki oraz zaimplementować kilka własnych funkcji, które pomogłyby przeciwnikom przeszukiwać teren rozgrywki, poruszać się, walczyć, dzielić zadaniami oraz strzelać do celów.

1.1.1 Algorytm A*

Bardzo ważną kwestią w każdej grze jest problem poruszania się przeciwników i odnajdowania drogi do obranego celu. Powstało wiele algorytmów pozwalających to osiągnąć. W swoim programie skorzystałem z algorytmu A*. Został on opracowany w 1968 roku, lecz pomimo swojego wieku, nadal jest podstawową metodą używaną w grach komputerowych do znalezienia optymalnej ścieżki do obranego celu. Algorytm ten oblicza najbardziej efektywną oraz najszybszą trasę pomiędzy dwoma punktami (węzłami grafu), jednocześnie usuwając te węzły, które nie są dostępne (np. są przeszkodami bądź są już zajęte przez innego przeciwnika). Technika ta oblicza koszt dotarcia do obranego punktu i dodaje do niego przewidywany koszt osiągnięcia celu. Najczęściej koszt to po prostu odległość do celu mierzona od obecnego punktu przeszukiwań. Algorytm w każdym ruchu sprawdza możliwe kierunki dalszej trasy i ponownie wybiera najlepszy kierunek ruchu. Gdy zbadane położenie okazuje się poszukiwanym punktem, algorytm kończy swoje działanie. W innym przypadku zapamiętuje przyległe położenia, by w przyszłości móc je sprawdzić jeszcze raz[1].

Aby algorytm mógł działać w czasie rzeczywistym nie powodując spadku klatek gry, byłem zmuszony podzielić mapę na bloki po 50 pikseli (tyle wynosi rozmiar bitmap graczy, przeszkód oraz przeciwników). Dzięki temu otrzymałem do obliczania optymalnej ścieżki prostokąt o wymiarach 19 na 14 jednostek. Przed implementacją algorytmu w grze sprawdziłem ile czasu zajmie wykonanie pojedynczego obliczenia na

prostokącie o takich wymiarach. Wypełniając mapę losowymi danymi oraz ustawiając określony punkt startu oraz końca, zmierzyłem kilkakrotnie czas potrzebny na obliczenia. Wyniósł on średnio około jednej milisekundy, tak więc mając do obliczenia trasę dla trzech, a nawet czterech przeciwników, czas ten nie jest żadnym ograniczeniem dla aplikacji. Tym bardziej, że algorytm nie jest wykonywany w każdej klatce gry, a tylko wtedy gdy:

a) Zmieni się status poszukiwania gracza lub bazy.

b) Przeciwnik dotrze do określonego położenia w celu ponownego obliczenia trasy, gdyby ta uległa zmianie. Przykładowo, gdy inny przeciwnik lub też cel (gracz) zmieni swoje położenie i trzeba będzie kierować się na inne współrzędne. Również podczas przeszukiwania mapy, przeciwnik będzie potrzebował kolejnych współrzędnych do sprawdzenia obecności gracza.

c) Podczas bliskiej walki z graczem, aby przeciwnik mógł próbować zaatakować gracza z różnych stron.

Dzięki temu algorytmowi przeciwnicy potrafią atakować gracza z różnych stron, gdyż każdy przeciwnik jest polem niedostępnym na mapie algorytmu innego towarzysza.

1.1.2 Poszukiwanie gracza oraz bazy

Przeciwnicy nie znają położenia gracza oraz bazy dopóki nie znajdą się oni w polu zasięgu wzroku jednego z przeciwników. Po udanym wykryciu gracza lub bazy, zostaje zmieniona odpowiednia zmienna, informując pozostałych przeciwników o wykryciu. Następnie są oni informowani o położeniu celu. Zasięg widoku przeciwników jest przedstawiony na Rys. 1. Przerywanymi czarnymi liniami zaznaczyłem ogólnie rozpatrywany obszar. Natomiast przerywanymi liniami koloru niebieskiego zaznaczyłem obszar, który widzi przeciwnik w obecnej sytuacji z przeszkodą. Stała zasięgu wzroku jest konfigurowalna w odpowiednim pliku. Dodatkowo jeśli przeciwnik znajduję się w odległości poniżej 100 pikseli od przeszkody, która całym swoim obszarem znajduje się w jego polu widzenia, to przeciwnik nie wie co znajduje się na całym obszarze jego widoku za tą przeszkodą. Ma to na celu symulację zbliżania się do przeszkody. W prawdziwym świecie im bliżej jesteśmy jakiejś przeszkody tym więcej nam ona zasłania.

Rys. 1 Schemat zasięgu wzroku przeciwnika

Podczas poszukiwania każdy przeciwnik otrzymuje różne koordynaty, do których musi się udać w celu sprawdzenia ich pod kątem obecności gracza i bazy. Sprawdzanie to jest wykonywane w każdej iteracji pętli głównej rozgrywki. Do weryfikacji sprawdzania mapy wykorzystałem tablicę dwuwymiarową o wymiarach pola walki (950 na 700 pikseli). Jej rozmiar jest duży, ale operacje na niej nie powodują znaczącego spadku wydajności. Każdy element tablicy odpowiada pikselowi na mapie o współrzędnych równych położeniu tego elementu w tablicy. Tablica przyjmuje określone wartości symbolizujące odpowiedni stan przeszukania danego piksela. Dokładniej jest to opisane w drugim rozdziale. Sprawdzenie czy gracz lub baza znajdują się w obecnym polu widzenia jest wykonywane w oparciu o to, jaki stan ma wyżej wymieniona tablica na krawędziach współrzędnych poszukiwanego obiektu. Wystarczy sprawdzić kontury, gdyż jeśli któryś ich piksel będzie widoczny, oznacza to, że obiekt jest widoczny dla przeciwników i w praktyce piksele obiektu w pionie lub poziomie (w zależności od pozycji) do tego wykrytego, również są widoczne.

1.1.3 Zachowanie przeciwników

Żeby przeciwnicy stanowili wyzwanie dla gracza, otrzymali oni również szereg zachowań, w zależności od występującego scenariusza. Pierwszym takim zachowaniem jest strzelanie do przeszkód zniszczalnych jeśli znajdują się one na drodze wymaganej do osiągnięcia celu. Program sprawdza czy rakieta wystrzelona z czołgu trafi przeszkodę oraz czy na linii strzału nie ma swojego sojusznika. Droga rakiety do celu zajmuje pewien czas, więc może się jednak zdarzyć, że poruszający się przeciwnik zostanie trafiony pociskiem swojego sojusznika. Można byłoby zastosować algorytm, który by temu zapobiegał, lecz bratobójczy ogień niestety zdarza się również w prawdziwym życiu. Moim celem nie było stworzenie przeciwników idealnych, którzy nie popełnialiby błędów.

Kolejnym pomniejszym algorytmem wspomagającym zachowanie przeciwników jest dzielenie się zadaniami. Przykładowo: jeśli istnieje trzech przeciwników oraz na mapie znajduje się gracz i baza, to podczas wykrycia gracza, a w przypadku nie znalezienia bazy, dwóch przeciwników stara się walczyć z graczem, a trzeci stara się poszukiwać bazy. Natomiast podczas wykrycia bazy, jeden przeciwnik ją atakuje, a pozostali walczą z graczem. Dzięki temu wyzwanie ochrony bazy jest trudniejsze dla gracza. Dodatkowo gracz musi liczyć się z tym, że nie tylko musi walczyć z atakującymi go przeciwnikami, ale również starać się zlikwidować przeciwnika który poszukuje bazy.

Jeśli przeciwnik zostanie sam, a kolejni jego towarzysze mają zostać odrodzeni w ciągu paru sekund, oraz w danym momencie nie prowadzi walki z graczem, zatrzymuje się. Po zatrzymaniu oczekuje on na odrodzenie swoich sojuszników, przy okazji patrolując wszystkie kierunki wokół siebie, i w przypadku wykrycia przed sobą gracza, strzela w jego stronę rakietami. Po odrodzeniu choćby jednego kompana, przeciwnik zaczyna się poruszać. Dodatkowo, jeśli jest on osamotniony, a graczowi pozostało jedno życie, podejmuje on ryzyko i stara się poszukiwać gracza, lub jeśli już go wykrył, stara się go pokonać. Dzięki temu przeciwnicy starają się wykorzystać przewagę liczebną nad graczem i nie angażować się w konflikty jeden na jeden.

Następną cechą pozwalającą zwiększyć możliwości przeciwników jest to, że podczas walki z graczem, zapamiętują ostatni kierunek, w którym się poruszał. W konsekwencji, jeśli na moment stracą gracza z widoku, to pamiętając w jakim kierunku zmierzał, kierują się dokładnie w tę samą stronę.

Kolejną cechą jest reagowanie na ostrzał. Jeśli przeciwnik zostanie trafiony rakietą z kierunku, w którym akurat obecnie nie podąża, zmienia kierunek na ten z którego przyleciała rakieta oraz natychmiastowo, jeśli może, oddaje strzał w tamtym kierunku.

Ostatnim ważniejszym algorytmem jest korekcja ruchów. Podczas poruszania się przeciwnik korzysta z algorytmu A*. Niestety może wystąpić scenariusz, w którym gracz blokuje przeciwnika tak, że ten nie miałby ruchu oraz możliwości trafienia czołgu gracza. W tym wypadku przeciwnik sprawdza, czy można jakoś opuścić tę blokadę i zaatakować gracza z innego kierunku. Jeśli nie ma innych blokad, takich jak sojusznicze czołgi bądź niezniszczalne przeszkody, to manewr ten jest realizowany. Jednak może się zdarzyć tak, że czołg przeciwnika nie będzie mógł trafić w punkt docelowy wyznaczony przez algorytm A*, który dla zapewnienia odpowiedniej

wydajności musiał podzielić mapę na bloki 50 pikselowe. Tutaj z pomocą przychodzi korekcja ruchów, która sprawdza w jakim kierunku obecnie porusza się przeciwnik i stara się korygować o jeden piksel na iterację pętli rozgrywki, położenie przeciwnika. Korekcja jest wykonana tak, aby przeciwnik nie zyskiwał dodatkowych pikseli w kierunku w którym się obecnie porusza. Dzięki temu nie odnosi z tego powodu dodatkowej przewagi nad graczem.

1.2 Obsługa rozgrywki

W tym podrozdziale opiszę założenia techniczne oraz zasady rozgrywki.

1.2.1 Założenia techniczne

Przed rozpoczęciem tworzenia gry musiałem przyjąć jakieś założenia techniczne odnośnie rozdzielczości gry, ilości klatek na sekundę, rodzaju przeszkód oraz bibliotek, które pomogą mi w programowaniu.

Jeśli chodzi o biblioteki to wybór padł na Allegro 5, o czym pisałem już wcześniej.

Rozdzielczość gry zależała od tego jak dużą mapę chcę stworzyć. Finalnym wyborem została rozdzielczość HD tzn. 1280 na 720 pikseli, gdzie pole bitwy zajmuje 950 na 700 pikseli. Dalsza część szerokości ekranu została wykorzystana do stworzenia panelu bocznego z informacjami dla gracza.

Ilość klatek na sekundę, które chciałem osiągnąć i osiągnąłem bezproblemowo wynosi 60. Pisząc klatka mam tutaj na myśli ilość wyrenderowanych scen na sekundę, a więc ilość iteracji pętli rozgrywki. Gra nie jest wymagająca dla kart graficznych, ze względu na prostotę grafiki. Natomiast stawia większe wyzwanie procesorowi, który musi obliczać trasę dla przeciwników oraz przetwarzać algorytm związany z przeszukiwaniem obszaru, który jest wywoływany w każdej pętli rozgrywki. Jednak nie przeszkadza to w żadnej mierze obecnym procesorom, dla których nie jest to dużym wyzwaniem. Grę testowałem na kilku laptopach z procesorami Intel Core i3 oraz Intel Core i5, gra działała płynnie bez żadnych problemów.

Kolejnym punktem, który musiałem przemyśleć przed przystąpieniem do prac, był rodzaj przeszkód jakie chcę umieścić w grze. Zdecydowałem umieścić przeszkody zarówno zniszczalne jak i niezniszczalne. Te pierwsze można zniszczyć po trafieniu w nie określoną ilość razy. Rodzaj przeszkód miał pomniejsze znaczenie, gdyż z uwzględnieniem algorytmu A*, mogę ustalać czy chcę wyszukać trasę bez niszczenia

przeszkód (jeśli jest to możliwe) czy też uwzględnić to, że mogę parę przeszkód na trasie do celu zniszczyć. Domyślnie algorytm wyszukiwania trasy traktuje przeszkody zniszczalne jako niezniszczalne, gdyż na destrukcję każdej przeszkody przeciwnik traci czas. Natomiast gdy takie wyszukiwanie nie znajdzie trasy, to wtedy trasa jest obliczana z uwzględnieniem przeszkód zniszczalnych.

1.2.2 Zasady rozgrywki

Rozgrywka polega na wyeliminowaniu jak największej ilości przeciwników, którzy odradzają się co cztery sekundy. Dodatkowo, jeśli na mapie pojawi się baza, należy jej bronić przed atakiem wrogów. Po zniszczeniu bazy bądź skończeniu się żyć gracza, gra jest zakończona.

Zasady samej rozgrywki są bardzo proste. Natomiast istnieją jeszcze inne zasady, nazwę je techniczne, gdyż odpowiadają za działanie przeciwników oraz środowisko gry.

Po wykryciu gracza uruchamiany zostaje licznik, który zostaje inkrementowany w każdej iteracji pętli rozgrywki. Jeśli w kolejnej iteracji uda się ponownie wykryć gracza, licznik jest zerowany. Natomiast gdy przeciwnicy zgubią gracza, licznik jest inkrementowany do wartości 200. Osiągając tę wartość licznik zostaje dezaktywowany oraz wykrycie gracza zostaje zanegowane, a przeciwnicy rozpoczynają przeszukiwanie terenu od początku. Dzięki temu, przez dodatkowy czas przeciwnicy mają czas odnaleźć gracza, wspomagając się zapamiętanym kierunkiem jego ruchu.

Aby zapewnić graczowi dodatkowe możliwości ukrycia, to w przypadku gdy liczba przeszkód spadnie poniżej określonej wartości (część przeszkód zostanie zniszczona), zostaną one zastąpione nowymi. Pojawienie się nowych przeszkód jest bezkolizyjne, tzn. że nie pojawią się one na graczu, innej przeszkodzie czy przeciwniku. Podobnie ma się sprawa z odrodzeniem przeciwników. Są oni odradzani bezkolizyjnie oraz z pewnym marginesem odległości od kompanów, gracza oraz bazy, jeśli ta istnieje.

Mapy w trybach przetrwania oraz obrony bazy są losowo generowane. Elementy mapy są generowane bezkolizyjnie.

Dodatkowo, jeśli graczowi uda się pokonać wszystkich obecnie żywych przeciwników przed odrodzeniem przynajmniej jednego, przeciwnicy gubią trop bazy oraz gracza. Oczywiście jeśli wcześniej znali ich położenie.

1.2.3 Umiejętności gracza i przeciwników oraz bonusy

Żeby urozmaić rozgrywkę w wielu grach implementowane są pewnego rodzaju umiejętności, które gracz w trakcie postępów rozgrywki lub osiągnięciu określonych celów, może rozwinąć. W swojej grze również zastosowałem takie rozwiązanie. Po zniszczeniu przeciwnika istnieje 15% szans, że pozostawi on po sobie bonus. W celu aktywacji gracz musi swoim czołgiem wjechać na bitmapę bonusu. Jeśli inny przeciwnik go uprzedzi, bonus przepada.

Po zdobyciu bonusu, następuje losowanie jego rodzaju. Gracz może otrzymać:

- zwiększenie liczby dostępnych rakiet,
- dodatkowe życie,
- zwiększenie prędkości rakiet,
- zwiększenie szybkości przeładowania rakiet,
- nalot powietrzny (niszczy wszystkich aktywnych przeciwników),
- dodatkowe życie dla bazy, jeśli ta istnieje.

Umiejętności związane z rakietami mają swój limit, czyli nie mogą być ulepszane w nieskończoność. Dzięki takim atrybutom mogłem stworzyć różne rodzaje przeciwników, różniących się umiejętnościami, ilością żyć oraz bitmapą dla wizualnego rozróżnienia.

Dodatkowo zaimplementowałem symulację uszkodzeń po trafieniu rakietą. Oprócz zmniejszenia pozostałych żyć, zostaje zmniejszona wybrana zdobyta umiejętności przez gracza. Zmniejszanie umiejętności ma oczywiście swój limit i nie działa w nieskończoność.

1.2.4 Pauzowanie rozgrywki

W głównej pętli programu korzystam z licznika, aby uzyskać stałą liczbę klatek gry. Jest on tak ustawiony, aby na sekundę wyświetlanych było 60 klatek. Licznik jest obiektem bibliotek Allegro 5 i ma za zadanie regulować częstotliwość wykonywania funkcji.

Pauzowanie rozgrywki po wciśnięciu klawisza P, polega na tym, że licznik jest zatrzymywany, a więc rozgrywka zostaje wstrzymana. Po kolejnym wciśnięciu klawisza P, gra jest wznawiana, a więc licznik jest ponownie uruchamiany.

Dodatkowo po przegranej rozgrywce lub ukończeniu poziomu w trybie kampanii, gra jest kontynuowana przez dodatkowe dwie sekundy. Dopiero po tym czasie gra się kończy lub też w przypadku kampanii, zostaje załadowana kolejna misja.

1.3 Wykrywanie kolizji

Wykrywanie kolizji jest bardzo istotnym elementem każdej gry komputerowej. W mojej aplikacji algorytm wykrywania kolizji oparłem na porównywaniu współrzędnych, na których znajdują się obiekty. Jest on wykonywany podczas każdego ruchu obiektu zdolnego do poruszania. Algorytm obsługuje gracza, przeciwników, przeszkody, rakiety oraz bonusy. Podczas wykrycia kolizji u gracza lub przeciwnika z przeszkodą, innymi czołgami bądź też z granicą pola walki, jest on cofany na poprzednią pozycję, która jest zapisywana przed każdym ruchem. Celem tego zapisu jest właśnie możliwość cofnięcia ruchu podczas kolizji. Z perspektywy gracza wygląda to tak jakby obiekt stał w miejscu, gdyż nie może się poruszyć w danym kierunku.

Aby ułatwić poruszanie się pomiędzy przeszkodami, byłem zmuszony dodać pewien margines (pięć pikseli, do skonfigurowania w pliku) błędu wykrycia pomiędzy przeszkodami, a graczem lub przeciwnikami. W przeciwnym razie czołgi musiałyby idealnie dopasować się pomiędzy przeszkody. Nie było to dobre rozwiązanie z punktu widzenia gracza. Dla graczy komputerowych sterowanych algorytmami nie robiło to żadnych problemów.

1.4 Udźwiękowienie gry

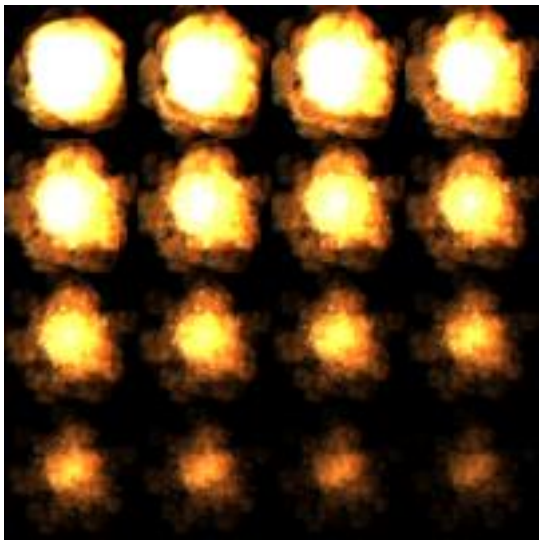
Biblioteki Allegro 5 zawierają moduł odpowiedzialny za obsługę dźwięku oraz plików dźwiękowych. W swojej aplikacji wykorzystuję pięć plików dźwiękowych. Są one odtwarzane w przypadku:

- uderzenia rakiety w czołg,
- uderzenia rakiety w przeszkodę,
- eksplozji czołgu,
- eksplozji przeszkody,

- wystrzału rakiety.

1.5 Animacja eksplozji

Do stworzenia animacji eksplozji wykorzystałem grafikę bitmapową. W pliku graficznym zapisane jest 16 stanów eksplozji. Animacja polega na wyświetlaniu określonego stanu przez określoną liczbę klatek, a następnie wyświetlenie kolejnego.



Rys. 2 Bitmapa eksplozji służąca do stworzenia animacji

1.6 Tryby gry

W swojej grze zaimplementowałem trzy tryby gry.

Pierwszym z nich jest przetrwanie. Gracz ma za zadanie przetrwać, niszcząc jak najwięcej przeciwników. Unicestwieni przeciwnicy są odradzani co cztery sekundy.

Drugim trybem jest obrona bazy. Gracz ma zadania takie same jak w poprzednim trybie z tym, że musi dodatkowo zadbać o to, aby jego baza nie została zniszczona. W przeciwnym wypadku przegra.

Ostatnim trybem jest kampania. Składa się ona z sześciu poziomów, które są wczytywane z plików. Zawierają one deklarację ilości poziomów, położenie gracza, przeciwników oraz przeszkód wraz z ich atrybutami. Dzięki temu można łatwo bez ingerencji w kod aplikacji dodawać nowe poziomy oraz edytować istniejące.

1.7 Dodatki

Aby sama aplikacja była bardziej przyjazna dla użytkownika dodałem do niej parę funkcjonalności.

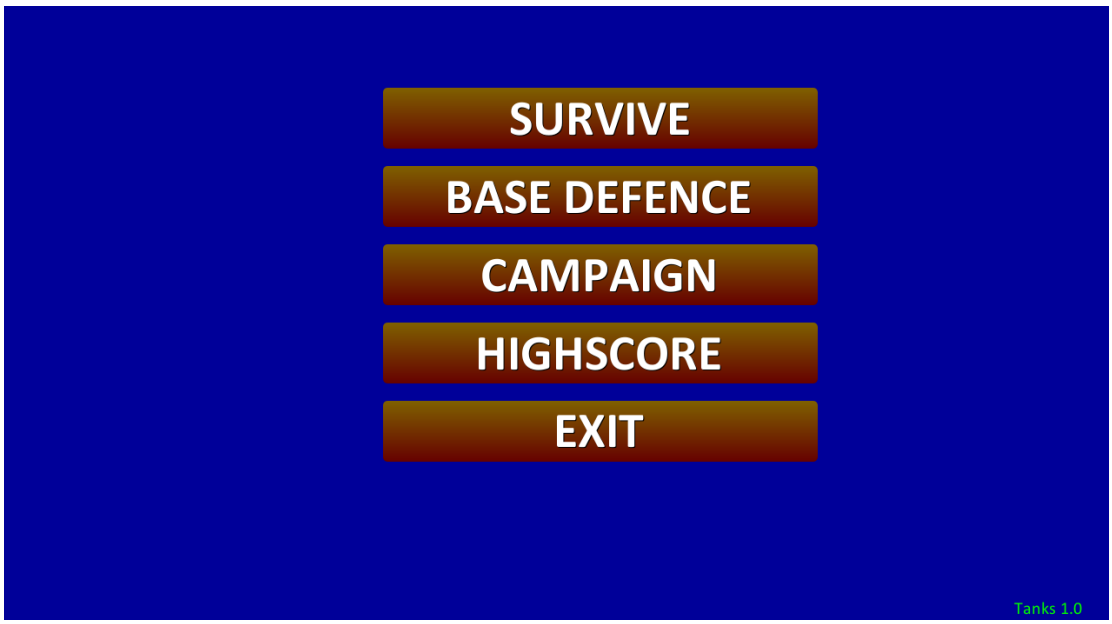
Pierwszą z nich jest ranking dziesięciu najlepszych wyników graczy, osobny dla każdego trybu. Za zniszczenie przeciwników, gracz otrzymuje punkty, które są liczone w zależności od trudności przeciwnika. Jeśli po zakończeniu rozgrywki gracz uzbiera ilość punktów wystarczającą do zapisania go w rankingu najlepszych wyników, pojawia się monit o podanie pięcioliterowej nazwy gracza.

Kolejną dodatkową funkcjonalnością jest główne menu gry obsługiwane za pomocą myszy z wykorzystaniem bitmap przycisków. Działanie polega na sprawdzeniu współrzędnych kliknięcia myszy. Jeśli są one zawarte we współrzędnych przycisku, to zostaje wykonana jego przypisana aktywność.

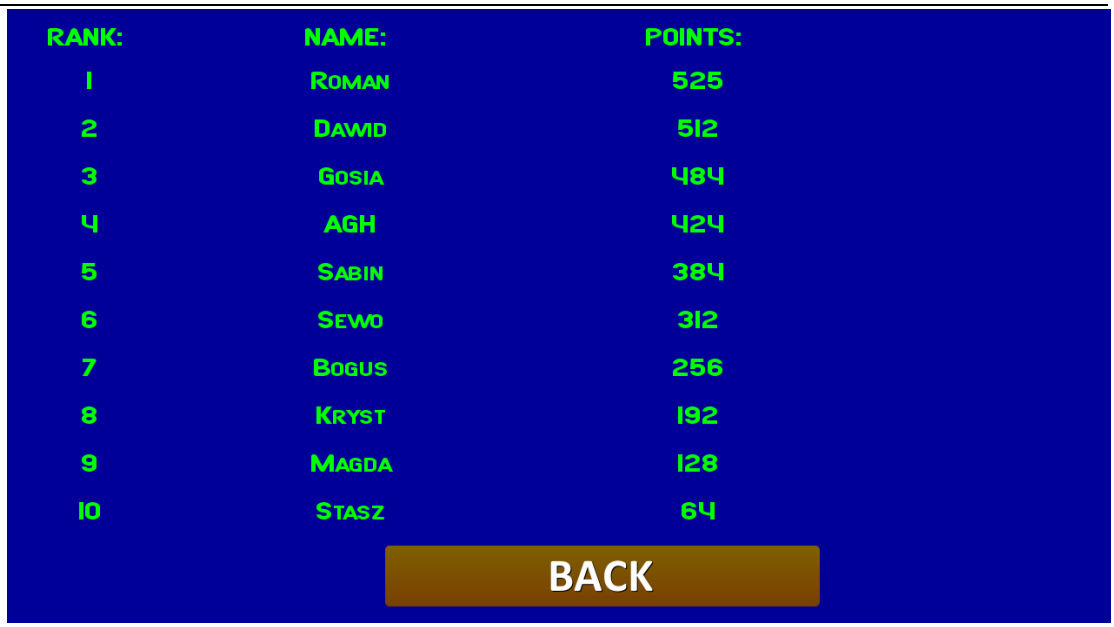
Ostatnim dodatkiem jest panel boczny zawierający informację dla gracza. Prezentuje on:

- ilość punktów, zabójstw oraz pozostałych żyć gracza i bazy (jeśli ta istnieje),
- stan umiejętności,
- obecny poziom oraz progres poziomu (tylko w trybie kampanii).

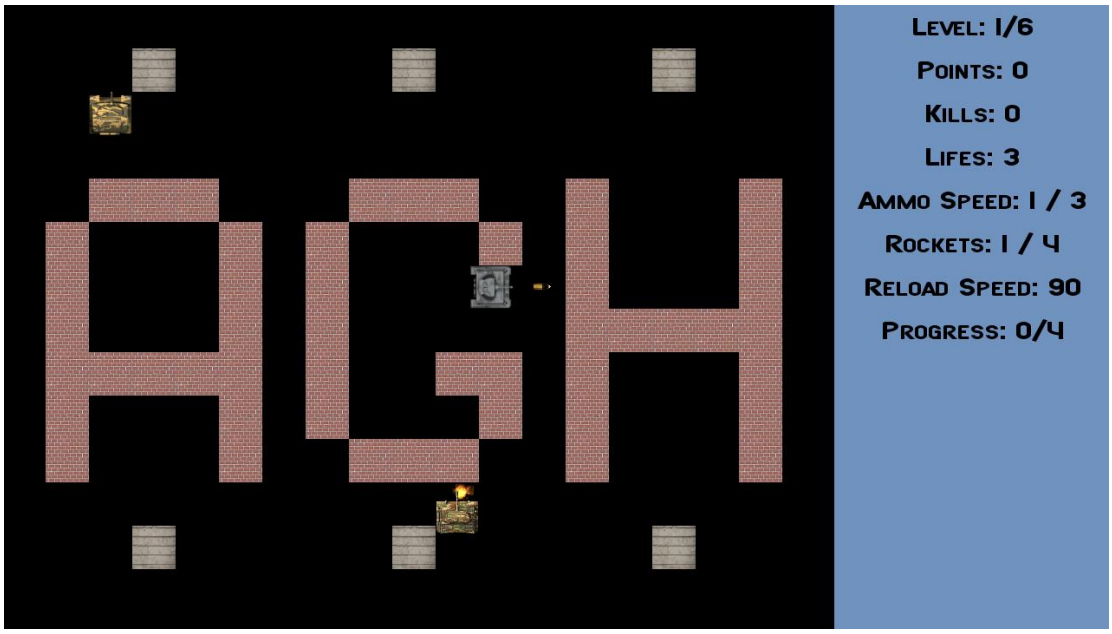
1.8 Zrzuty ekranu



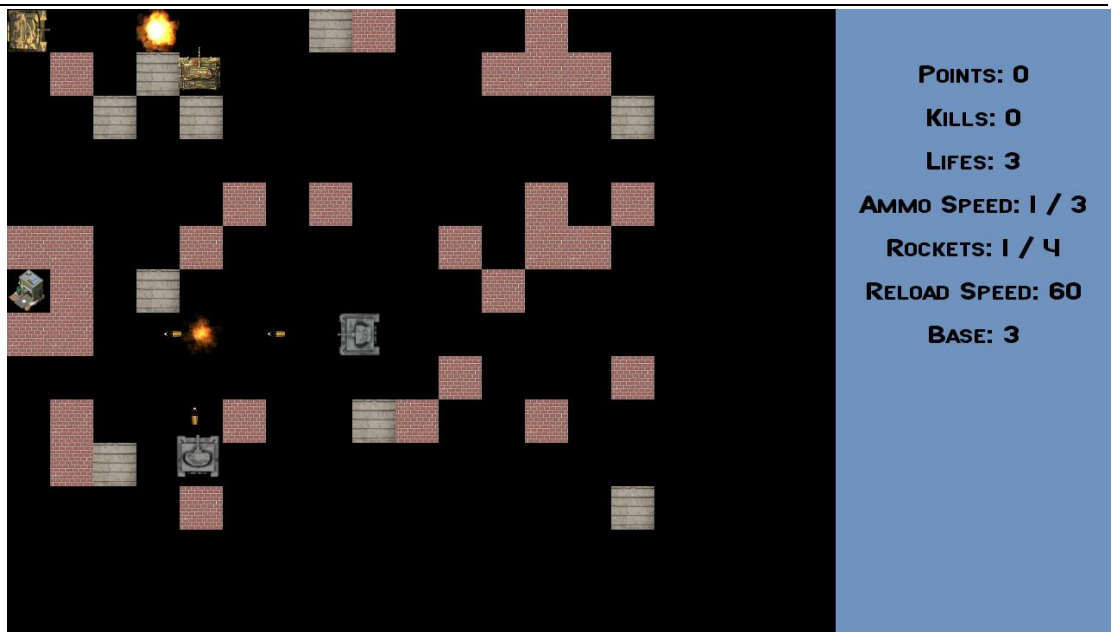
Rys. 3 Menu główne



Rys. 4 Ranking najlepszych wyników



Rys. 5 Zrzut ekranu z rozgrywki w trybie kampanii



Rys. 6 Zrzut ekranu z rozgrywki w trybie obrony bazy

Rozdział 2

Dokumentacja techniczna

W tym rozdziale przedstawię dokumentację techniczną programu. Pomoże ona w zapoznaniu się z kodem aplikacji podczas ewentualnego dalszego jej rozwoju. W każdym podrozdziale omówię co zawiera się w kolejnych plikach. Na koniec opiszę sposób dodawania oraz edytowania poziomów kampanii.

2.1 Plik konfiguracyjny

W pliku config.hpp zostały zdefiniowane liczne stałe umieszczone w wielu miejscach napisanego kodu, lub takie, których zmiana nie powinna wymagać dokładnego przeszukiwania kodu. Ma to na celu większą kontrolę i szybszą edycję parametrów rozgrywki. Przykładami są:

- wymiary mapy,
- częstotliwość wyświetlanych klatek (stała używana do konfiguracji licznika sterującego pętlą rozgrywki),
- rozmiar mapy dla algorytmu A*,
- zdefiniowanie nazw plików dla czcionek, bitmap oraz dźwięków,
- zdefiniowanie stałych używanych wielokrotnie w kodzie, dzięki czemu szybko można zmienić ich wartość.

2.2 Konfiguracja bibliotek Allegro 5

W pliku allegroconfig.cpp znajdują się funkcje związane z:

- inicjalizacją bibliotek Allegro,
- ładowaniem danych używanych przez aplikację,
- zwalnianiem pamięci programu przed wyłączeniem aplikacji,
- wyświetlaniem ostrzeżenia o błędzie.

W tym pliku następuje inicjalizacja głównej biblioteki Allegro, wyświetlacza, obsługi myszki, klawiatury oraz czcionek. Wyświetlacz jest również konfigurowany,

aby okno gry miało odpowiednią rozdzielczość i właściwości. Następuje również ładowanie wszystkich bitmap, czcionek, dźwięków do pamięci. Jeśli któraś nie zostanie załadowana, wyświetlony zostanie błąd, a aplikacja zostanie zamknięta (uprzednio zwalniając zajęta pamięć).

*2.3 A**

W pliku aStar.cpp znajduje się implementacja algorytmu A*. Funkcja obliczająca trasę z punktu startowego do docelowego zwraca wartość typu tekstowego (string). Jej wartościami są cyfry określające kierunek, w którym musi się udać przeciwnik.

Tabela 1 Możliwe zwracane wartości

Wartość zwracana	Kierunek
0	Wschód
1	Południe
2	Zachód
3	Północ
5	Brak trasy

Podczas inicjalizacji algorytmu A*, następuje ustawienie mapy algorytmu. Zaznaczane są miejsca zajęte przez przeszkody, gracza oraz przeciwników.

2.4 Menu oraz ranking najlepszych wyników

W pliku menu.cpp zawarta jest obsługa menu głównego, wpisywania przez gracza nazwy podczas zdobycia dobrego wyniku i zapisywanie oraz ładowanie najlepszych wyników.

Ranking najlepszych wyników jest przechowywany w pliku highscore.cfg. Jest to plik formatu INI. Zapisane są w nim nazwy graczy oraz ich wyniki, osobno dla każdego trybu gry. Do obsługi tego typu plików wykorzystywane są funkcje z bibliotek Allegro. Po zakończeniu rozgrywki, wynik jest porównywany z najlepszymi wynikami z pliku. Jeśli jest lepszy niż którykolwiek z obecnych, to gracz zostaje poproszony o podanie swojego pseudonimu. Maksymalnie może on zawierać pięć znaków. Po akceptacji pseudonimu, wynik zostaje zapisany do pliku.

Menu jest obsługiwane za pomocą myszy. Zawiera bitmapy przycisków, których funkcjonalność jest aktywowana po naciśnięciu lewego klawisza myszy, gdy kursor znajduje się na bitmapie. Po uruchomieniu rozgrywki kursor myszy jest ukrywany, natomiast po powrocie do menu, zostaje ponownie wyświetlony.

2.5 Gracz

Pliki player.cpp oraz player.hpp zawierają klasę gracza oraz funkcje związane z jego obsługą. Klasa posiada atrybuty odpowiadające za:

- zliczanie zabójstw oraz punktów,
- umiejętności (szybkość, ilość oraz czas przeładowania rakiet),
- ilość pozostałych żyć oraz aktywność,
- położenie (współrzędne położenia oraz współrzędne tymczasowe wykorzystywane w przypadku kolizji, żeby cofnąć gracza na poprzednią pozycję, kierunek poruszania),
- zliczanie czasu przeładowania rakiet.

Część atrybutów klasy jest typu protected. Ma to na celu wykorzystanie ich przy dziedziczeniu po klasie gracza. Dodatkowo wykorzystuje przeciążenie operatora przypisania w celu zapisywania stanu gracza w kampanii po zmianie poziomu, aby gracz nie tracił swoich umiejętności.

W klasie występują metody, które są odpowiedzialne za:

- inicjalizacje gracza podczas zwykłej gry oraz kampanii,
- poruszanie się,
- wykrywanie kolizji,
- zerowanie atrybutów gracza,
- odczytywanie i edytowanie atrybutów.

Kierunki poruszania się gracza, rakiet oraz przeciwników określają liczby. Legenda znajduje się w tabeli poniżej.

Tabela 2 Powiązanie wartości zmiennej kierunku z faktycznym kierunkiem

Wartość zmiennej	Kierunek
0	Wschód
1	Zachód
2	Północ
3	Południe

2.6 Przeciwnicy

Pliki enemy.cpp oraz enemy.hpp zawierają klasę przeciwnika oraz funkcje związane z jego obsługą. Posiada atrybuty odpowiadające za:

- położenie (współrzędne położenia oraz współrzędne tymczasowe wykorzystywane w przypadku kolizji, żeby cofnąć gracza na poprzednią pozycję, kierunek poruszania),
- umiejętności (szybkość, ilość oraz czas przeładowania rakiet),
- ilość pozostałych żyć oraz aktywność,
- zliczanie czasu przeładowywania rakiet i ponownego odrodzenia,
- obsługę wykrycia gracza oraz bazy (flagi oraz liczniki wykrycia, mapa przeszukanego terenu),
- ustawienie bitmapy.

Klasa dziedziczy z klasy gracza. Występują w niej metody odpowiedzialne za:

- odczytywanie i edytowanie atrybutów,
- sprawdzanie kolizji podczas rozgrywki oraz przed odrodzeniem,
- inicjalizacja podczas zwykłej rozgrywki oraz w trybie kampanii,
- sprawdzenie czy przeciwnik jest nakierowany na cel,
- poruszanie się, korekcję ruchu oraz strzelanie rakietami,
- przeszukiwanie mapy oraz nadawanie tymczasowych miejsc przeznaczenia,
- zerowanie atrybutów.

Funkcja sprawdzająca kolizję przeciwników z innymi obiektami zwraca wartość liczbową w zależności od obiektu, z którym wystąpiła kolizja. Ma to na celu podjęcie odpowiednich zadań w zależności od rodzaju kolizji.

Tabela 3 Znaczenie wartości zwracanych przez funkcję wykrywającą kolizję

Wartość zmiennej	Znaczenie
0	Brak kolizji
1	Kolizja z niezniszczalną ścianą
2	Kolizja z innym przeciwnikiem
3	Kolizja ze zniszczalną ścianą
4	Kolizja z graczem
5	Kolizja z bazą gracza

Tablica odpowiedzialna za stan przeszukania mapy jest uaktualniana w każdej klatce podczas wykonywania funkcji sprawdzającej obecny zasięg wzroku przeciwnika. Podczas wykonywania funkcji, tablica przyjmuje kilka wartości, których znaczenie jest wyjaśnione w tabeli poniżej.

Tabela 4 Znaczenie wartości tablicy odpowiadającej za stan przeszukania mapy

Wartość zmiennej	Znaczenie
0	Współrzędne nieprzeszukane
1	Współrzędne niewidoczne
2	Współrzędne sprawdzone ale niewidoczne obecnie (są zasłonięte)
3	Współrzędne widoczne obecnie
4	Współrzędne sprawdzone

Funkcje sprawdzające czy przeciwnik jest ustawiony na określony cel (przeszkodę, bazę lub gracza) sprawdzają, czy jeśli z obecnej pozycji wystrzeli on rakietę, to czy trafi ona w cel. Przed strzałem jest sprawdzane, czy w linii strzału nie ma przeszkód niezniszczalnych lub swoich kompanów. Dodatkowo jeśli wybrany cel do ostrzału znajduje się w odległości mniejszej niż 76 pikseli, to przeciwnik zatrzymuję się dopóki cel pozostaje w jego zasięgu.

Przeszukiwanie mapy polega na przydzieleniu każdemu przeciwnikowi określonych współrzędnych, które nie zostały jeszcze sprawdzone. Po ich sprawdzeniu przydzielane są kolejne. Każdemu przeciwnikowi przydzielane są inne współrzędne w różnych częściach mapy, aby nie poruszali się oni zbyt blisko siebie.

W trybie kampanii ilość przeciwników, którzy mogą pojawić się jednocześnie, jest określana podczas każdego poziomu. W pozostałych trybach domyślną wartością jest trzy. Można ją szybko zmienić w pliku konfiguracyjnym.

2.7 Rakiety

Pliki rocket.cpp oraz rocket.hpp zawierają klasę oraz funkcje odpowiedzialne za obsługę rakiet. Klasa rakiet posiada atrybuty odpowiadające za:

- współrzędne położenia,
- kierunek,
- szybkość,
- widoczność,
- przynależność,
- zliczanie wystrzelonych rakiet.

Klasa nie jest dziedziczona przez żadną inną klasę, więc wszystkie jej atrybuty są typu private.

W klasie występują metody odpowiedzialne za:

- odczytywanie i edytowanie atrybutów,
- poruszanie oraz sprawdzanie kolizji,
- obsługę zderzenia z obiektem oraz utworzenie eksplozji.

Rakieta po wykryciu kolizji wywołuje funkcję odpowiedzialną za obsługę zderzenia z danym obiektem. Jeśli obiekt jest możliwy do uszkodzenia, to traci on jedno życie. Dodatkowo wyświetlana jest animacja wybuchu oraz odtwarzany jest dźwięk trafienia lub też zniszczenia, w przypadku unicestwienia obiektu.

2.8 Przeszkody

Pliki wall.cpp oraz wall.hpp zawierają klasę odpowiedzialną za przeszkody oraz funkcje związane z ich obsługą. Część atrybutów klasy ma typ private, a część protected, gdyż jest to klasa z której dziedziczy parę innych klas. Atrybuty odpowiadają za:

- współrzędne położenia,
- pozostałe życia oraz widoczność,
- typ przeszkody (baza, przeszkoda zniszczalna lub niezniszczalna),
- zliczanie czasu do odrodzenia przeszkód, jeśli byłoby ich za mało.

W plikach zawarte są funkcje odpowiedzialne za:

- utworzenie przeszkody z wybranymi parametrami,
- sprawdzanie czy przeszkoda nie będzie kolidowała z innym obiektem,
- obsługę przeszkód w trybie kampanii (inicjalizacja mapy oraz odradzanie).

Podczas tworzenia przeszkody jest sprawdzane, czy nie będzie ona kolidowała z innymi obiektami. Jeśli będzie, to nowe współrzędne są generowane losowo i ponownie jest sprawdzana możliwość wystąpienia kolizji. Dzieję się tak dopóki nie znajdzie miejsca, na którym będzie można ją umieścić.

2.9 Bonusy

Pliki bonus.cpp oraz bonus.hpp zawierają klasę odpowiedzialną za bonusy oraz funkcję związaną z obsługą nalotu powietrznego, który niszczy wszystkich aktywnych przeciwników. Klasa ta dziedziczy z klasy przeszkód. Posiada atrybuty odpowiadające za współrzędne położenia oraz widoczność.

2.10 Eksplozje

Pliki explosion.cpp oraz explosion.hpp zawierają klasę odpowiedzialną za eksplozje oraz funkcje związane z ich obsługą. Klasa ta dziedziczy z klasy przeszkód. Posiada atrybuty odpowiadające za:

- współrzędne położenia,
- czas wyświetlania oraz widoczność eksplozji,

- rozmiary animacji,
- wielkość eksplozji.

Jeśli rakieta uderzy w cel, ale nie niszczy go, następuje mała eksplozja. W przypadku zniszczenia obiektu, wyświetlana jest duża eksplozja. Czas wyświetlania animacji jest ustalany w pliku config.hpp.

Klasa, oprócz odczytywania i edytowania atrybutów, posiada funkcję odpowiedzialną za aktywowanie wyświetlania eksplozji.

2.11 Główny plik

W pliku main.cpp zawarte są funkcje odpowiedzialne za losowanie bonusu, obsługę ładowania poziomów kampanii, pętle główną rozgrywki, czyszczenie atrybutów używanych modeli oraz zmiennych, inicjalizacje gry oraz zmiennych, sterowanie rozgrywką i zliczanie wyświetlonych klatek.

W przypadku zdobycia przez gracza bonusu, wykorzystywana jest funkcja rand służąca do wylosowania liczby. Następnie wylosowana liczba służy do ustalenia bonusu, który otrzyma gracz. Każda opcja ma określony przedział liczbowy, który ją aktywuje.

Licznik klatek na sekundę jest domyślnie wyłączony. Aby go uruchomić należy w config.hpp zdefiniować parametr FPS_COUNTER na 1. Do pomiaru klatek skorzystałem z wątku. W każdej pętli rozgrywki zwiększona zostaje zmienna liczącą ilość klatek. Natomiast w wątku, co sekundę przepisywana jest wartość tej zmiennej do innej, która zostaje następnie wyświetlana na ekranie. Zmienna zliczająca jest wtedy zerowana i zliczanie zaczyna się od początku. Funkcja ta służyła głównie do sprawdzenia, czy program wyświetla pożądaną liczbę klatek na sekundę.

Funkcja inicjalizująca zmienne ma za zadanie zarezerwować pamięć na obiekty klas. Zostaje wykonana na początku uruchomienia programu. Natomiast inicjalizacja gry ma za zadanie załadować wszystkie wymagane moduły biblioteki Allegro.

Sterowanie rozgrywką jest obsługiwane w funkcji ModeGuardian, która jest wywoływana podczas destrukcji przeszkody bądź zniszczenia przeciwnika. Ma za zadanie sprawdzić, czy wymagane jest odrodzenie przeciwnika lub przeszkód. Jeśli jest wymagane, to uruchamia licznik, który po czterech sekundach gry wykonuje odrodzenie. W trybie kampanii funkcja zapewnia ustalenie, czy poziom lub kampania

zostały ukończone. Dodatkowo jeśli na mapie nie istnieją aktywni przeciwnicy, to ustawiamy zmienną odpowiedzialną za wykrycie bazy oraz gracza na wartość fałszywą.

W głównej funkcji rozgrywki następuje wstępna konfiguracja poziomu, czyli ustawienie gracza, przeszkód, przeciwników oraz ich rozmieszczenie. W przypadku kampanii następuje inicjalizacja funkcji ładującej pierwszy poziom kampanii. Po wczytaniu ustawienia poziomu, występuje pętla główna rozgrywki. Na jej początku zaimplementowany jest mechanizm pauzowania rozgrywki oraz start licznika służącego do ustalenia stałej liczby klatek na sekundę. Po wystąpieniu wydarzenia licznika, rozpoczyna się generowanie nowej klatki gry. Rozpoczyna się ona od wyczyszczenia ekranu. Potem wyświetlany jest panel informacyjny dla gracza. Następnie wykonywana jest obsługa wyświetlania bitmap na ekranie oraz obsługa zachowania modeli. Kolejnym krokiem jest wyświetlanie komunikatów o przegranej bądź zwycięstwie, oraz obsługa liczników. Główna pętla rozgrywki jest przerywana po porażce lub ukończeniu kampanii przez gracza oraz po wciśnięciu klawisza escape.

2.12 Tworzenie oraz edytowanie poziomów kampanii

W aplikacji zaimplementowałem możliwość wczytywania poziomów z plików INI. Wykorzystałem to do stworzenia kampanii dla gracza. Stworzonych zostało sześć poziomów. Każdy ma swój własny oddzielny plik. Schemat nazewnictwa plików z misjami wygląda następująco: levelX.tank, gdzie X oznacza numer misji. Ilość poziomów kampanii jest określana w pliku pierwszej misji, a więc level1.tank. Format INI pozwala na tworzenie sekcji i parametrów. Sekcję są wykorzystywane do deklaracji opcji, gracza, przeciwników oraz przeszkód. Każda posiada odpowiednie parametry pozwalające zdefiniować atrybuty obiektów. Schemat nazewnictwa sekcji dla przeszkód oraz przeciwników jest następujący: wallNRX oraz enemyNRX, gdzie X oznacza numer przeszkody oraz przeciwnika (oznaczenie rozpoczyna się od zera).

Podsumowanie i wnioski

Sztuczna inteligencja jest bardzo ciekawym tematem, który cały czas prężnie się rozwija. Podczas tworzenia aplikacji zrozumiałem jakie jest jej znaczenie w każdej grze komputerowej i ile trzeba poświęcić czasu, aby starać się ją jak najbardziej dopracować.

Stworzenie aplikacji pozwoliło mi zwiększyć swoje umiejętności programistyczne oraz zapoznać się z problemem sztucznej inteligencji w grach komputerowych. Wyzwaniem dla mnie było stworzenie działającego algorytmu poruszania się przeciwnika oraz implementacja algorytmu A*. Dodatkowo zapoznałem się również z mechanizmami pisania gier komputerowych. Przekonałem się, co jest wymagane oraz z jakimi problemami trzeba się zmagać, aby stworzyć działającą aplikację.

Najważniejszym dla mnie celem było stworzenie gry, w której przeciwnicy potrafiliby stawiać wyzwanie graczowi. Uważam, że ten cel udało się mi osiągnąć. Pomocne w tym celu były wszelakie algorytmy sterujące przeciwnikami, które udało się mi zaimplementować.

Większość obranych przez siebie celów udało się mi osiągnąć. Chciałem jeszcze wprowadzić obsługę drugiego gracza, aby można było grać w dwie osoby przy jednej klawiaturze lub też przez Internet. Niestety zabrakło mi czasu na dodanie tej funkcjonalności. Pozwala to na dalszy rozwój aplikacji w przyszłości w ramach poszerzania swojej wiedzy z tego tematu, gdyż nie stworzyłem wcześniej żadnej aplikacji która komunikowałaby się z inną za pomocą Internetu. Taki cel jest bardzo ciekawy, gdyż dodatkowym problemem przy takiej rozgrywce jest opóźnienie występujące podczas przesyłania danych przez sieć.

Streszczenie

Głównym celem pracy było stworzenie dwuwymiarowej gry komputerowej, w której przeciwnicy potrafiliby stanowić wyzwanie dla gracza. Do stworzenia gry wykorzystałem język C++ oraz biblioteki Allegro 5, które są kompatybilne z systemami operacyjnymi: Windows, Linux, Mac OSX, iOS oraz Android. Dzięki czemu przeportowanie aplikacji na inne platformy nie wymaga dużego nakładu pracy. Powstała gra jest typu zręcznościowego, polegająca na sterowaniu przez gracza czołgiem i walce z innymi czołgami oraz na obronie swojej bazy, aby nie dopuścić do zniszczenia jej przez wroga. Biblioteki zapewniają obsługę dźwięku, grafiki, myszy, klawiatury oraz czcionek. Do stworzenia inteligentnych przeciwników skorzystałem z algorytmu A*, który pomaga odnaleźć najszybszą trasę do punktu docelowego. Dodatkowo stworzyłem algorytm służący do kontroli zachowania przeciwników, uskutecznienia możliwości przeszukiwania mapy, walki z graczem oraz rozdzielenia celów między siebie. Gra posiada trzy tryby gry: przetrwanie, obrona bazy oraz kampanie, w której gracz ma do ukończenia sześć poziomów. W celu urozmaicenia gry gracz posiada umiejętności, które może zwiększać dzięki bonusom. W grze występuje kilka rodzajów przeciwników różniących się od siebie umiejętnościami oraz bitmapami. Aby uatrakcyjnić poziomy, aplikacja zawiera przeszkody zarówno zniszczalne jak i niezniszczalne. Występują w niej również animacje eksplozji, menu główne obsługiwane za pomocą myszy oraz ranking dziesięciu najlepszych wyników graczy.

Summary

The main aim was to create a two-dimensional computer game in which opponents are able to present a challenge for the player. To create the game I used C++ language and Allegro 5 libraries, which are compatible with the following operating systems: Windows, Linux, Mac OSX, iOS and Android. Thanks to that, porting does not require a lot of work. Created application is arcade game in which player control tank and fight with others tanks and try to prevent the base destruction. Libraries provide support for sound, graphics, mouse, keyboard and fonts. I used A* algorithm to create intelligent opponents. This algorithm helps opponents find the fastest route to the destination. Additionally, I created an algorithm to control behaviour of opponents, effective ability to search map, fight with the player and separation purposes between each other. The game has three game modes: survival, base defence and campaign, in which the player has to complete six levels. In order to diversify the game, the player has the skills, which may increase after earning bonuses. In the game there are a few types of enemies with different skills and bitmaps. To create more attractive levels, the game includes obstacles both destructible and indestructible. There are also animated explosions, the main menu that support mouse and the ranking of ten best results for the players.

Słowa kluczowe

- Allegro 5,
- A*,
- gra komputerowa,
- sztuczna inteligencja,
- czołgi.

Keywords

- Allegro 5,
- A*,
- computer game,
- artificial intelligence,
- tanks.

Bibliografia

[1] Deloura M., Perełki programowania gier – Vademecum profesjonalisty, t.1 i 2, tłum. R. Jońca, Helion, Gliwice 2002.

[2] Politechnika Warszawska, *Algorytm A**, http://iair.mchtr.pw.edu.pl/~bputz/aisd_cpp/lekcja7/segment4/main.htm, stan na dzień: 07.12.2016

[3] Patrick Lester, *Odnajdywanie ścieżki typu A* dla początkujących*, http://www.policyalmanac.org/games/aStarTutorial_pol.htm, *stan na dzień: 07.12.2016*

[4] *Dokumentacja bibliotek Allegro 5.2*, http://liballeg.org/a5docs/5.2.0/refman.pdf, *stan na dzień: 12.12.2016*

[5] Krzysztof Wardziński, *Przegląd algorytmów sztucznej inteligencji stosowanych w grach komputerowych*, http://www.hc.amu.edu.pl/numery/5/wardzinski.pdf, *stan na dzień: 12.12.2016*

[6] Beata Kuźmińska-Sołśnia oraz Tomasz Siwiec, *Zastosowanie algorytmów sztucznej inteligencji w grach komputerowych*, http://www.bks.pr.radom.pl/publikacje/SI%20w%20grach.pdf, *stan na dzień: 12.12.2016*

[7] *Projektowanie gier*, http://home.agh.edu.pl/~woznia/projektowanie%20gier.pdf, *stan na dzień: 12.12.2016*

[8] Politechnika Gdańska, *Proces tworzenia oprogramowania*, http://www.mif.pg.gda.pl/homepages/sylas/students/sem_podypl/io-w04.pdf, *stan na dzień: 13.12.2016*

[9] Tiziana Catarci, *Proces tworzenia gier komputerowych*, http://www.dis.uniroma1.it/~catarci/ps_PDF_files/Games1.pdf, *stan na dzień: 13.12.2016*

Dodatek A.

Spis zawartości dołączonej płyty CD

Praca

dawid_waksmundzki_258950.doc – Tekst pracy zapisany w formacie MS Word,

dawid_waksmundzki_258950.rtf – Tekst pracy zapisany w formacie rozszerzonym,

dawid_waksmundzki_258950.pdf – Tekst pracy zapisany w formacie PDF,

www – Folder zawierający zapisane kompletne strony sieci WEB oraz pliki pdf użyte w bibliografii,

tanks – Folder zawierający aplikację wraz z kodem źródłowym.

Spis ilustracji

Spis tabel