

Metodologia programowania

Gra „Tanks 2017” – dokumentacja projektu

Projekt wykonany przez:

- Mateusz Wedeł
 - Łukasz Rydziński
 - Paweł Minda
- 165NCI

Spis treści

<u>Gra „Tanks 2017” – dokumentacja projektu</u>	1
<u>1. Wymagania dotyczące gry oraz opis ich realizacji w naszym projekcie</u>	3
a) <u>Podstawowe wymagania:</u>	3
b) <u>Sugerowana funkcjonalność podstawowa (ocena 3):</u>	3
c) <u>Sugerowane elementy funkcjonalności lekko rozszerzonej (ocena 4):</u>	3
d) <u>Sugerowane elementy funkcjonalności rozszerzonej (ocena 5):</u>	4
<u>2. Opis projektu</u>	5
a) <u>Cele i funkcje oprogramowania:</u>	5
b) <u>Wstępny opis projektu:</u>	6
c) <u>Projekt całości systemu:</u>	8
<u>Schemat działania gry</u>	8
<u>Metoda EndGameCheck();</u>	11
d) <u>Implementacja:</u>	12
<u>3. Testy</u>	13
a) <u>Testy poprawności działania Menu</u>	13
b) <u>Testy poprawności działania Gry</u>	13
c) <u>Testy poprawności działania Plików .txt</u>	14

1. Wymagania dotyczące gry oraz opis ich realizacji w naszym projekcie

a) Podstawowe wymagania:

Użytkownik ma możliwość:

- **Zagrać w grę** - gra jest przeznaczona dla dwóch osób
- **Otrzymać informację o wyniku gry** - pojawia się stosowna informacja o tym, który gracz zwyciężył
- **Zapisać, odczytać listę graczy oraz ich wyniki (z możliwością usunięcia graczy)** - możliwość ta została zaprogramowana w formie rankingu graczy. Nick zostaje podany przez gracza przed rozpoczęciem rozgrywki. Szerzej o rankingu w dalszej części dokumentacji

b) Sugerowana funkcjonalność podstawowa (ocena 3):

- **Działająca gra** - gra prawidłowo się uruchamia
- **Informacja o wyniku gry** - w momencie, gdy jeden gracz zostanie pokonany przez drugiego, wyświetlony zostaje stosowny komunikat o zwycięzcy

c) Sugerowane elementy funkcjonalności lekko rozszerzonej (ocena 4):

- **Możliwość wpisywania gracza(y)** - każdy z graczy ma możliwość podania swojego nicku przed rozpoczęciem gry, choć nie jest to wymagane
- **Zapisywanie wyniku w pliku zewnętrznym (w wybranym formacie)** - w pliku zewnętrznym zapisywanych jest 10 najlepszych graczy, którzy wygrali największą ilość pojedynków
- **Odczyt wyników (ranking graczy), możliwość zapisywania lepszego wyniku gracza** - w menu głównym utworzyliśmy opcję pozwalającą obejrzeć ranking najlepszych graczy. Ranking ten wczytywany jest z pliku zewnętrznego
- **Możliwość usunięcia gracza z listy** - gracze zapisywani są na liście z rankingiem, o ile pozwala im na to ich wynik końcowy. Usunięcie graczy z rankingu możliwe jest poprzez wybór odpowiedniego przycisku. Usunąć pojedynczego gracza można bezpośrednio poprzez edycję pliku zewnętrznego.

d) Sugerowane elementy funkcjonalności rozszerzonej (ocena 5):

- **Możliwość zmiany ustawień początkowych (np. czas wyświetlania, wielkość planszy, punktacja, itp.)** – gracz ma możliwość wyboru utworów, które będą odtwarzane w grze oraz w menu głównym. Oprócz tego ma możliwość ustawić poziom głośności z jaką utwory będą odtwarzane. Ustawienia można zapisać do zewnętrznego pliku konfiguracyjnego, z którego opcje są pobierane przy każdym włączeniu programu
- **Możliwość zmiany ustawień wizualnych (np. kolor tła, kolor planszy, rodzaj elementów w grze)** – każdy z graczy ma możliwość wyboru koloru czołgu, którym będzie się poruszać. Dodatkowo istnieje możliwość wyboru mapy po której poruszać się będą mapy. Mapy podobnie jak ranking generowane są z plików zewnętrznych
- **Efekty wizualne, dźwiękowe** – gra została wzbogacona o starannie wykonaną szatę graficzną. Do tego dochodzi szereg efektów dźwiękowych (np. podczas wystrzeliwania pocisku), a wszystko to w parze z klimatycznymi utworami odtwarzanymi w tle

2.Opis projektu

a) Cele i funkcje oprogramowania:

- podać temat projektu,
- sformułować założenia do zadania (opracowanie specyfikacji).

Tematem projektu jest gra „**Tanks 2017**” przeznaczona dla dwóch graczy.

Wzorowaliśmy się na znanych i popularnych w swoich czasach grach „Tank 1991” oraz „Bomberman”.

Gracze "siadają" za sterami swojego czołgu, a ich głównym celem jest zniszczenie swojego przeciwnika.

Przewagę w walce zapewniają różnego rodzaju bonusy, które mogą pojawiać się po niszczeniu blozków oraz gęste zarośla mogące służyć nam za kryjówkę.




Do wyboru mamy kilka różnorodnych map po których możemy siać zniszczenie i desktrukcję.

A wszystko to opracowane jest w piękną grafikę i klimatyczną muzykę.

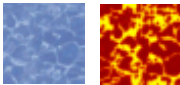

Sterowanie zostało tak zaprojektowane, by gracze bez problemu mogli obsłużyć czołg bez wzajemnego przeszkadzania sobie.

	Gracz 1	Gracz 2
Ruch:	<i>Strzałki</i>	<i>WSAD</i>
Strzelanie:	<i>? / Num0</i>	<i>G / CapsLock</i>




W grze występują różnego rodzaju **bločki**, które różnią się między sobą teksturą oraz właściwościami:

- **Cegła / Magma** – jest to rodzaj bločku, który możemy zniszczyć. To z niego wypadają będą losowo wybrane bonusy (o bonusach niżej).  
- **Krzew** – bloček, którego głównym celem jest zasłonięcie gracza przed przeciwnikiem. Bločka nie można zniszczyć. 
- **Skała** – bloček, który jest niezniszczalny oraz nie można na niego wjechać. Służy jako przeszkoda i motywuje do poszukiwania innej drogi do przeciwnika.



- **Woda / Lawa** – bloczek o tyle ciekawy, że mimo iż nie można na niego wjechać to można strzelać do celu znajdującego się po przeciwnej stronie. Idealny do prowadzenia walki na odległość. 
- **Trawa / Piasek / Magma** – nie tyle bloczek, co grafika która wyznacza nam teren po którym możemy się poruszać. 

Oprócz bloczków w grze występują wspomniane już **bonusy** dzięki którym możemy ulepszyć właściwości naszego czołgu.

- **Życie** – bonus, który regeneruje odniesione w walce obrażenia. Maksymalnie możemy mieć cztery serduszka. 
- **Prędkość** – bonus, dzięki któremu możemy zwiększyć prędkość naszego czołgu. Bonus działa tylko do momentu aż uzyskamy maksymalny poziom prędkości czołgu. 
- **Czas** – bonus, dzięki któremu zwiększamy szybkość ładowania kolejnych pocisków do lufy czołgu. Podobnie jak wyżej, nie możemy przekroczyć progu ustawionego dla najszybszego czasu przeładowania. 

b) Wstępny opis projektu:

- Podać nazwę programu i zrobić podział zadania na komponenty (podać ich nazwy i opisz ich funkcje).

Widok:

- ❖ **StartForm** - klasa ta zawiera metody odpowiedzialne za aktualizację widoku i przekazywanie danych z widoku do modeli. Jest to forma na której znajduje się główne menu gry oraz wszystkie panele z ustawieniami.
- ❖ **Form1** – klasa ta ma podobną zasadę działania co StartForm, jednak wykorzystywana jest dopiero w momencie samej gry.

Klasy:

- ❖ **Block** – model bloczków, które są wykorzystywane na mapie. Zawiera parametry definiujące jego właściwości oraz metody pozwalające na różnego rodzaju działania z nim związane. Np. Zanikanie bloczka po uderzeniu pocisku.
- ❖ **Bonus** – model bonusu, który możemy zebrać po zniszczeniu bloczka. Definiuje jakim rodzajem jest bonus oraz jakie atrybuty dostaniemy po jego zebraniu.
- ❖ **Character** – klasa abstrakcyjna, odpowiada za wspólne parametry czołgów zarówno graczy jak i przeciwników takie jak prędkość, życie czy czas przeładowania. Odpowiada również za otrzymywanie obrażeń od pocisku.
- ❖ **Enemy** – klasa dziedzicząca właściwości po klasie Character. Odpowiada za wszystkie parametry czołgów sterowanych przez AI. Takie jak sterowanie czy w przyszłości atak na gracza.
- ❖ **Record** – klasa, której wykorzystanie jest ściśle związane z rankingiem graczy. Przechowuje ona nick zwycięzcy oraz czas w którym wygrał grę.
- ❖ **Tank** – kolejna klasa dziedzicząca parametry po klasie Charakter. Odpowiada w szczególności za czołgi graczy. Oprócz standardowych właściwości takich jak życie czy prędkość poruszania się – odpowiada również za kolor czołgów czy dźwięki towarzyszące wystrzeleniu lub uderzeniu pocisku.

- Wybrać i wylistować ważniejsze typy i struktury danych (podaj nazwy), nazwy plików używanych w programie itp.

Listy:

- ❖ **Blocks** – lista zawierająca wszystkie bloczki potrzebne na daną mapę.
- ❖ **Bonuses** – lista zawierająca konkretną liczbę bonusów, które mogą znajdować się jednocześnie na planszy.
- ❖ **FileRec** – lista zawierająca rekordy rankingu.

Enumy (typy wyliczeniowe):

- ❖ **DirectionsOfMoving** – enum zawierający wszystkie kierunki ruchu w jakim może poruszać się czołg.
- ❖ **DirectionsOfBulletMoving** – podobnie jak wyżej, z tym że ten typ dotyczy wystrzeliwanych pocisków.
- ❖ **BonusType** – enum, który zawiera w sobie wszystkie możliwe typy bonusów jakie możemy spotkać w grze.

Obiekty (typy referencyjne):

- ❖ **Player1, Player2** – obiekty klasy Tank, które odpowiadają za czołgi graczy.
- ❖ **StartEnemy1-6** – obiekty klasy Enemy, które odpowiadają za czołgi sterowane przez AI. Widoczne są na StartFormie przy uruchomieniu programu.

Linq – do posortowania rekordów w tablicy użyliśmy poleceń z biblioteki Linq.

c) Projekt całości systemu:

- Przygotować opis całego programu w postaci pseudokodu lub schematu blokowego.

Nasz program jest bardzo złożony, wiele klas i metod jest ze sobą powiązanych, stąd też w poniższych pseudokodach przedstawiony zostanie w sposób ogólny działanie programu (po szczegółową implementację zapraszamy do sprawdzenia plików źródłowych).

Schemat działania gry

Legenda:

- ❖ **Initialize_Players();** - funkcja tworzy obiekty o nazwie Player1 i Player2 oraz nadaje im pierwotne parametry.
- ❖ **Initialize_Bonuses();** - funkcja tworzy listę z bonusami, dodaje do niej konkretną liczbę poszczególnych bonusów (różną dla każdej mapy), wczytuje ich tekstury i nadaje typ.
- ❖ **Initialize_Blocks();** - funkcja tworzy listę z blockami oraz dodaje do niej konkretną liczbę poszczególnych bloków (różną dla każdej mapy), wczytuje ich tekstury oraz nadaje właściwości.
- ❖ **Check_Music();** - funkcja pobiera z ustawień aktualnie wybraną muzykę i ją odtwarza.
- ❖ **Check_BackgroundMap();** - funkcja wczytuje konkretną teksturę backgroundu dla konkretnej mapy.
- ❖ **Check_PlayersLocalization();** - funkcja wczytuje konkretną lokalizację graczy dla konkretnej mapy.
- ❖ **HeroMoving();** - funkcja sprawdza czy gracz mogą się poruszać (czy nie ma kolizji) i jeśli tak to wprawia czołgi w ruch.

- ❖ **CheckHeroCollideWithBlok();** - funkcja sprawdza czy gracz nie mają kolizji z jakimś blokiem. Jeśli tak to czołg jest cofany do tyłu.
- ❖ **CheckHeroCollideWithHero2();** - funkcja sprawdza czy gracz nie mają kolizji z sobą nawzajem. Jeśli tak to odejmuje im życie i ustawia na pole startowe.
- ❖ **SpawnMovingBullet();** - funkcja sprawdza czy gracz mogą wystrzelić pocisk i jeśli tak to tworzą pocisk i wprawiają go w ruch.
- ❖ **BulletCollideWithBlok();** - funkcja sprawdza czy pocisk koliduje z blokiem. Jeśli tak to w zależności od typu blocka niszczy go lub nie robi z nim nic i usuwa pocisk z planszy. Oprócz tego ustala czy w miejsce zniszczonego blocka ma pojawić się bonus.
- ❖ **BulletCollideWithMainPanel();** - funkcja sprawdza czy pocisk koliduje z krawędziami pola gry. Jeśli tak to pocisk zostaje usunięty.
- ❖ **BulletCollideWithTank();** - funkcja sprawdza czy pocisk koliduje którymś z czołgów graczy. Jeśli tak to usuwa pocisk z planszy, a czołgowi z którym nastąpiła kolizja odejmowane jest życie.
- ❖ **BonusCollideWithTanks();** - funkcja sprawdza czy bonus koliduje którymś z czołgów graczy. Jeśli tak to usuwa bonus z planszy, a czołgowi z którym nastąpiła kolizja dodawany jest parametr, który niesie za sobą bonus. Pod warunkiem, że czołg gracza nie przekroczył dopuszczalnej wartości dla konkretnego parametru.
- ❖ **UpdatePictures();** - funkcja sprawdza poziom życia graczy oraz ich fazę w jakiej znajduje się ładowanie pocisku. W zależności od tych parametrów aktualizuje obrazki w panelu pod planszą gry.
- ❖ **EndGameCheck();** - funkcja sprawdza czy poziom życia któregoś z graczy nie jest mniejszy lub równy 0. Jeśli tak to gra zostaje przzerwana, wyświetlone zostaje powiadomienie o zwycięzcy, a jego nazwa wraz z wynikiem wpisywana jest do pliku z rankingiem. W przypadku, gdy obaj gracze stracili ostatnie życie w tym samym momencie – wyświetlana jest informacja o remisie.

1. Initialize_Players();
2. Initialize_Bonuses();
3. Initialize_Blocks();
4. Check_Music();
5. Check_BackgroundMap();
6. Check_PlayersLocalization();
7. while (true)
8. HeroMoving();
9. CheckHeroCollideWithBlok();
10. CheckHeroCollideWithHero2();
11. SpawnMovingBullet();
12. BulletCollideWithBlok();
13. BulletCollideWithMainPanel();
14. BulletCollideWithTank();
15. BonusCollideWithTanks();
16. UpdatePictures();
17. EndGameCheck();

*Schemat działania gry
„Tanks 2017” w
postaci pseudokodu*

- Podać bardziej szczegółowy opis ważniejszych algorytmów (w postaci pseudokodu lub schematów blokowych).

Metoda EndGameCheck();

Legenda:

- ❖ **Player1/2.HealthPoints** – zmienna określająca ilość punktów każdego z graczy.
- ❖ **MainTime.Enabled** – zmienna boolowska określająca czy główny timer na planszy jest włączony czy też nie.
- ❖ **TimeReload.Enabled** – podobnie jak wyżej, z tym że ten timer zajmuje się czasem oczekiwania jaki musi upłynąć między kolejnymi strzałami.
- ❖ **MessageBox.Show();** – MessageBox służy do wyświetlania komunikatów. W tym przypadku poinformuje, który z graczy wygrał grę.
- ❖ **FileUpdate();** – Funkcja dodająca zwycięzcę oraz jego wynik do pliku z rankingiem.
- ❖ **BackgroundMusic.controls.stop();** – Funkcja, która spowoduje zatrzymanie muzyki.
- ❖ **StartForm startform = new StartForm();** – Deklaracja nowego obiektu klasy StartForm o nazwie startform.
- ❖ **startform.Show();** – Funkcja powodująca wyświetlenie startforma.
- this.Hide();** – Funkcja powodująca ukrycie aktywnego forma.

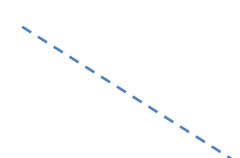
```

if (Player1.HealthPoints <= 0 && Player2.HealthPoints <= 0)
{
    MainTime.Enabled = false;
    TimeReload.Enabled = false;
    MessageBox.Show($"Remis, brak zwycięzcy. Czas to {TimeOfGame}\n Koniec gry!");
    BackgroundMusic.controls.stop();
}
else
{
    if (Player1.HealthPoints <= 0)
    {
        MainTime.Enabled = false;
        TimeReload.Enabled = false;
        MessageBox.Show($"Wygrał gracz {Player2.Nickname}: Czas to {TimeOfGame}\n Koniec gry!");
        FileUpdate(Player2.Nickname);
        BackgroundMusic.controls.stop();
    }

    if (Player2.HealthPoints <= 0)
    {
        MainTime.Enabled = false;
        TimeReload.Enabled = false;
        MessageBox.Show($"Wygrał gracz {Player1.Nickname}: Czas to {TimeOfGame}\n Koniec gry!");
        FileUpdate(Player1.Nickname);
        BackgroundMusic.controls.stop();
    }
}

if (!MainTime.Enabled && !TimeReload.Enabled)
{
    StartForm startform = new StartForm();
    startform.Show();
    this.Hide();
}

```



*Schemat działania funkcji
EndGameCheck() w
postaci kodu*

Metoda **EndGameCheck()** podobnie jak wiele innych jest wywoływana bez przerwy, ze względu na potrzebę ciągłego sprawdzania czy któryś z graczy nie wygrał już gry.

d) Implementacja:

- Kodowanie wykonać z zastosowaniem zasad programowania obiektowego.

Nasz projekt został wykonany z zachowaniem podstawowych zasad programowania obiektowego, występują tutaj podstawowe zależności jak dziedziczenie, polimorfizm, hermetyzacja itp.

3. Testy

a) Testy poprawności działania Menu

Zadania (w Menu):

- Próba wpisania bardzo długiego nicku.
- Próba wyczyszczenia pustej listy z rankingiem.
- Próba zapisu pustych opcji muzyki tła lub/i muzyki w grze.

Wyniki (w Menu):

- Ograniczenie sprawia, że nie można podać nicków dłuższych niż 14/15 znaków.
- Nie ma problemu z czyszczeniem pustej listy.
- Pojawia się komunikat o próbie zapisu pustej opcji. Zapis się nie wykonuje. W dalszym ciągu można opcje zmienić na poprawne i zapisać.

b) Testy poprawności działania Gry

Zadania (w Grze):

- Próba wyjechania czołgiem poza mapę.
- Próba wjechania czołgiem w bloczek.
- Próba wjechania czołgiem w czołg przeciwnika.

Wyniki (w Grze):

- Warunki sprawdzające kolizje z krawędziami planszy sprawiają, że czołg nie może wyjechać poza mapę.
- Warunki sprawdzające kolizje z bloczkami sprawiają, że czołg zostaje odsunięty na bardzo mały dystans od bloczka.
- Przy próbie zderzenia czołgów, oby dwa czołgi tracą jedno serduszko życia i wracają na pozycje startowe.

c) Testy poprawności działania Plików .txt

Zadania (Pliki .txt):

- Usunięcie pliku konfiguracyjnego.
- Usunięcie pliku z rekordami.

Wyniki (Pliki .txt):

- Przy włączeniu gry ustawienia przyjmują wartości domyślne, natomiast plik konfiguracyjny zostaje utworzony po kolejnym zapisaniu ustawień gracza.
- Pojawia się komunikat o braku pliku z rekordami. Gracz proszony jest o ponowne uruchomienie gry. Podczas ponownego uruchomienia plik z rekordami jest już utworzony na nowo.

Gra tworzona była w taki sposób by ilość niespodziewanych sytuacji była jak najmniejsza, przez co staraliśmy się wyeliminować nie porządane wyniki podczas rozgrywki.