Metody programowania .NET

Sprawozdanie z projektu

Temat: Gra Panzer General

Prowadzący laboratorium: mgr inż. Kamil Małysz

Grupa: WCY18IJ6S1

Wykonawcy: Filipek Daniel
Sokół Igor

Spis treści

1. Ogólny opis projektu

1.1. Opis początkowy

Tematem naszego projektu była implementacja gry komputerowej o nazwie **Panzer General**. Jest to strategiczna gra turowa o tematyce działań wojennych, w którą gra się w dwie osoby. Wzorem gry jest produkcja o tym samym tytule, która ukazała się w 1994 roku.

1.2. Rozgrywka

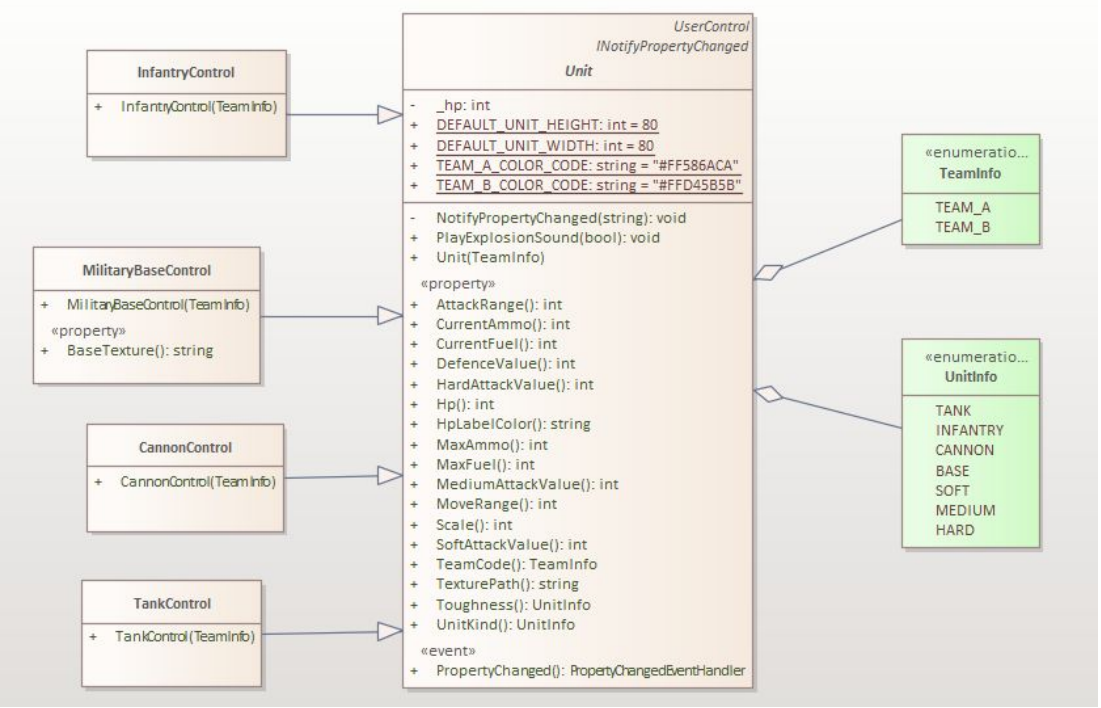
Zadaniem gracza jest zniszczenie głównej bazy przeciwnika zanim on zniszczy jego bazę. Gracze do dyspozycji mają 3 typy jednostek:

- piechotę,
- artylerię,
- czołgi.

Podczas tury gracz może wykonać jednorazowe przesunięcie oraz oddanie strzału z każdej posiadanej jednostki. Po wykonaniu operacji rozpoczyna się tura przeciwnika, która wygląda analogicznie do tury gracza. Gra kończy się wraz z zniszczeniem pierwszej bazy.

2. Diagramy klas

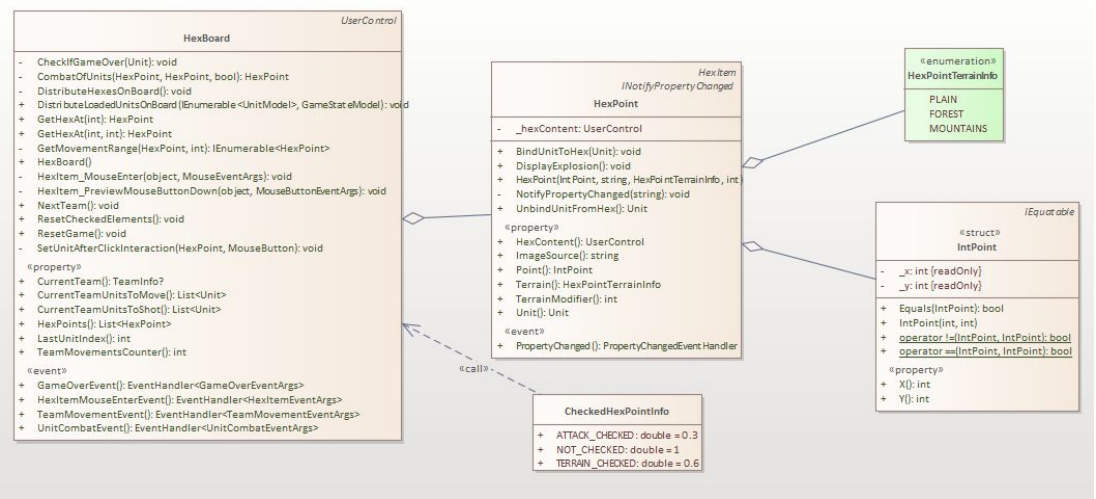
2.1. Klasa “Unit”



Rys.1 - Diagram klas związanych z klasą “Unit”

Klasa ta dziedziczy z klasy “UserControl”, jest więc kontrolką użytkownika. Jest bazą dla tworzenia jednostek różnych kategorii. Zawiera w sobie kluczowe pola, takie jak: zasięg rażenia, ilość życia, przynależność do drużyny, kategoria jednostki itd. Za pomocą agregacji częściowej w klasie “Unit” znajdują się również enumeratory “TeamInfo” oraz “UnitInfo”, które odpowiadają odpowiednio za przydzielenie do drużyny i przydzielenie odpowiedniej kategorii jednostki. Z klasy “Unit” dziedziczą wszystkie dostępne jednostki, czyli “Infantry” (piechota), “MilitaryBase” (baza), “Cannon” (artyleria), “Tank” (czołgi).

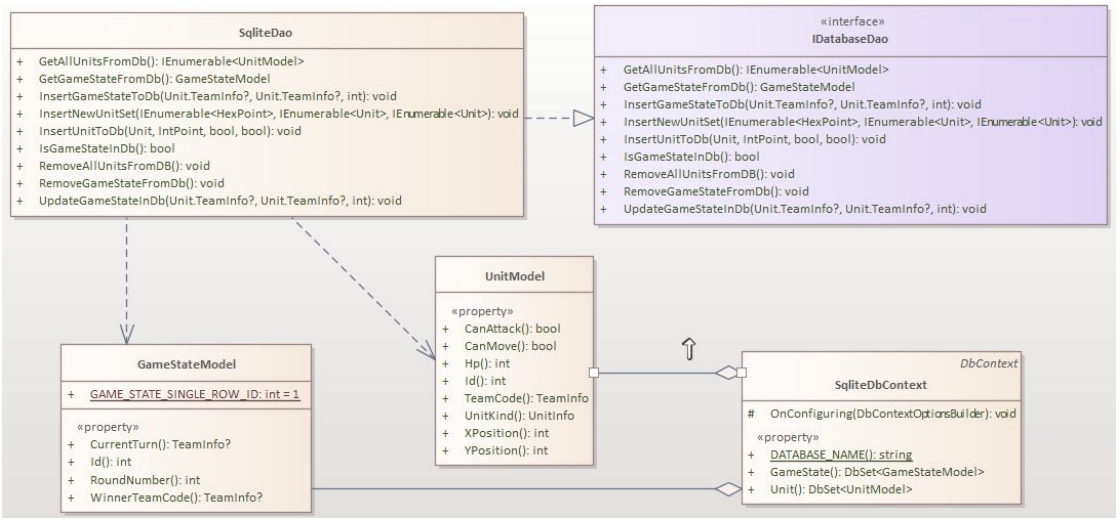
2.2. Klasa “HexBoard”



Rys. 2 - Diagram klas związanych z klasą “HexBoard”

Klasa “HexBoard” jest kluczowa jeśli chodzi o działanie gry. Odpowiada ona za rozmieszczenie elementów na planszy, która składa się z siatki sześciokątów (opis działania siatki heksagonalnej znajduje się w dalszej części sprawozdania). “HexBoard” implementuje listę obiektów klasy “HexPoint”, która dziedziczy z klasy “HexItem” odpowiadającej za sześciokątny wygląd elementu siatki. “HexPoint” za pomocą agregacji częściowej powiązany jest z enumeratorem HexPointTerrainInfo, który informuje o tym jaki typ terenu istnieje na danym elemencie siatki (równina, las lub góry). Atrybutem tej klasy jest również przedstawiciel klasy “IntPoint”, która odpowiada za przechowywanie współrzędnych danego elementu.

2.3. Klasa “SqliteDao”



Rys. 3 - Diagram klas związanych z klasą “SqliteDao”

Klasa “SqliteDao” i wszystkie inne klasy z nią związane odpowiadają za obsługę bazy danych, do której zapisywane są stany gry. “SqliteDao” implementuje interfejs “IDatabaseDao”, który dostarcza swego rodzaju API umożliwiającego dostęp do bazy danych. Klasa “UnitModel” przechowuje najważniejsze informacje dotyczące zapisywanej jednostki, a klasa “GameStateModel” przechowuje informacje dotyczące stanu rozgrywki.

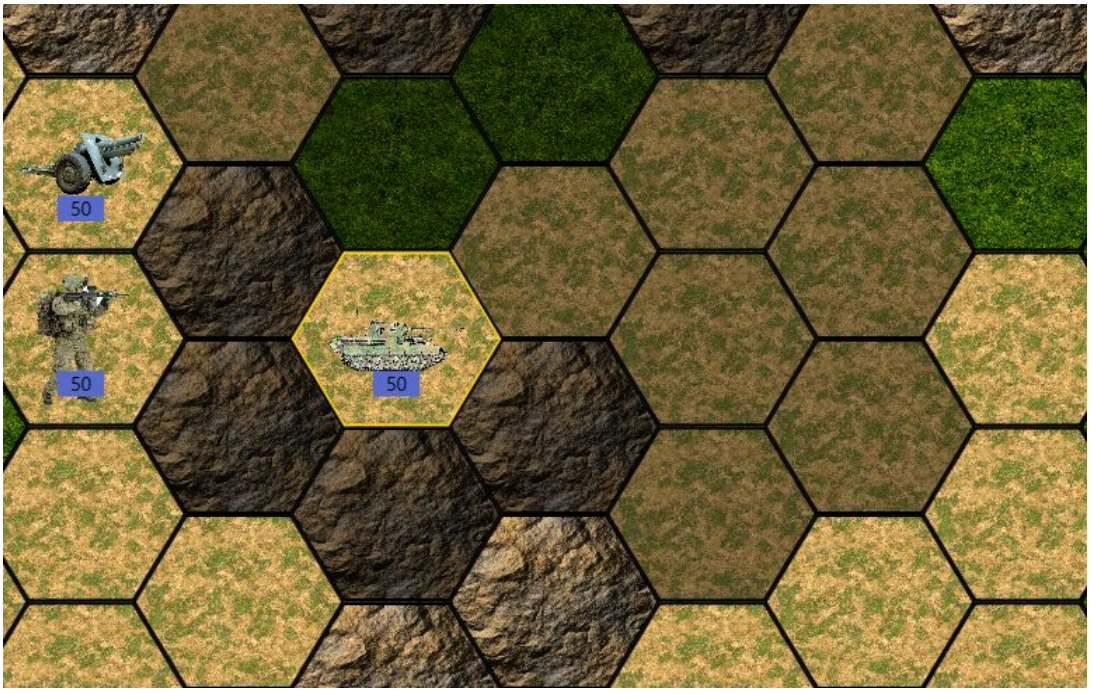
3. Spis dodatkowych funkcjonalności

Gra **Panzer General** zawiera następujące funkcjonalności :

1. Możliwość resetowania rozgrywki,
2. Możliwość zapisywania rozgrywki,
3. Możliwość wczytywania rozgrywki,
4. Statystyki zaznaczonej jednostki możemy podejrzeć klikając odpowiedni przycisk,
5. Gracz może pominąć turę, jeżeli nie widzi sensu prowadzenia dalszych działań na planszy klikając odpowiedni przycisk.

4. Mechanika gry

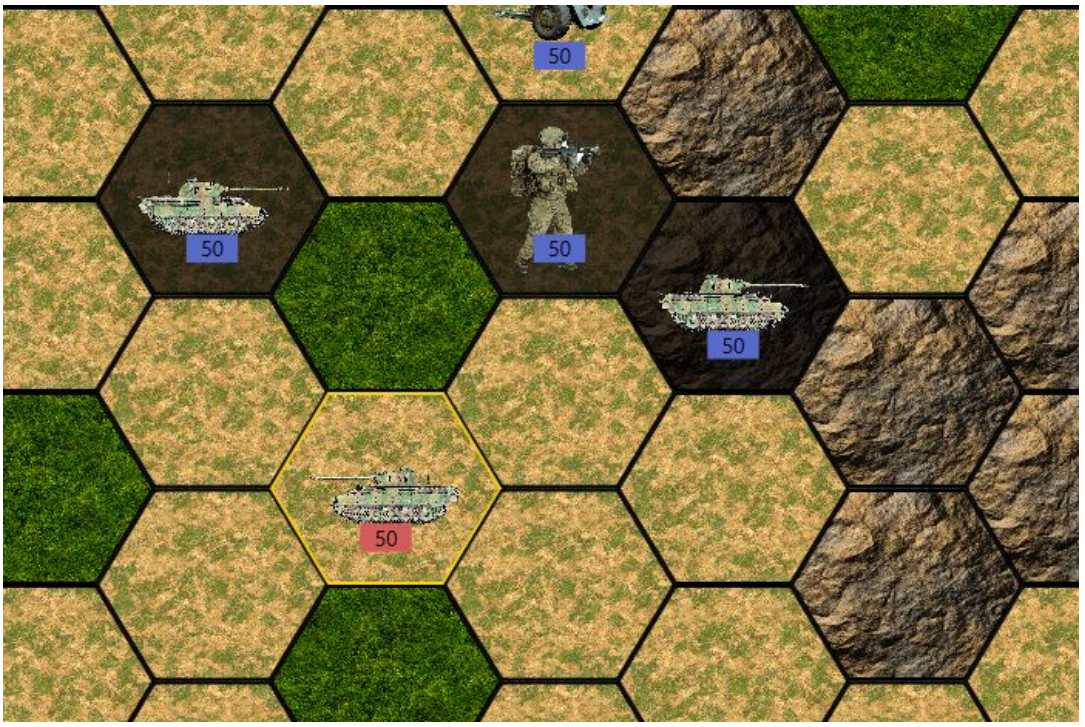
4.1. Mechanika ruchu jednostek



Rys. 4 - Zrzut ekranu z gry przedstawiający zakres ruchu wybranej jednostki

Gracz może przesuwać swoje jednostki tylko podczas swojej tury. Po kliknięciu na wybraną jednostkę, zaciemniają się pola na które można przesunąć jednostkę. Każda jednostka posiada swój zasięg ruchu. Zakres ruchu generowany jest za pomocą metody rekurencyjnej, która co wykonanie sprawdza sąsiadujące pola czy znajdują się w zasięgu ruchu jednostki. Na ruch jednostki wpływa typ terenu danego “HexPoint’a”. Jednostka nie może zająć miejsca innej jednostki.

4.2. Mechanika walki jednostek



Rys. 5 - Zrzut ekranu z gry przedstawiający możliwości strzału wybranej jednostki

Gracz podczas swojej tury może oddać jeden strzał z każdej jednostki (oprócz bazy), pod warunkiem, że w zasięgu strzelającej jednostki znajduje się jednostka przeciwnika. Każda typ jednostki ma określony zasięg rażenia, na podstawie którego program sprawdza czy w zakresie o jego wartości znajduje się jednostka przeciwnika, której pole przyciemnia się. Aby wykonać atak na jednostkę przeciwnika gracz musi wskazać jednostkę prawym przyciskiem myszy. Następuje atak jednostki atakującej a następnie kontratak jednostki atakowanej (o ile nie została zniszczona podczas ataku). Algorytm obliczenia wartości ataku wygląda następująco:

```
foreach (int i in Enumerable.Range(0, attacker.Unit.Hp)
{
        if (R + aI - dI + cI > 15)
        {
                dP++
        }
}
```

gdzie:

- R - losowa liczba całkowita z przedziału <1;21),
- aI - współczynnik ataku jednostki atakującej (z klasy Unit),
- dI - współczynnik obrony jednostki atakowanej (z klasy Unit),
- cI - współczynnik kontrataku - 1 jeśli atak, -1 jeśli kontratak,
- dP - punkty ataku - wartość, którą trzeba odjąć od życia obrońcy.

Pętla wykonuje się tyle razy ile punktów życia ma napastnik.

5. Wykorzystane biblioteki zewnętrzne

5.1. HexGridControl

Biblioteka zapewnia siatkę sześciokątów, która zbudowana została na bazie “ListBoxa”. Klasa “ListBoxItem”, z której obiektów składa się wspomniana przed chwilą klasa, jest dziedziczona przez klasę “HexItem”, która zmienia jej wygląd na sześciokątny element siatki. Analogicznie klasa “HexList” dziedziczy z klasy “ListBox” z tą różnicą, że jej kolekcja zawiera obiekty klasy “HexItem”. Biblioteka zapewnia podstawowe metody pozwalające na zaznaczanie poszczególnych elementów siatki i na podstawowe operacje na nich.

5.2. WpfAnimatedGif

Biblioteka została użyta do zaimplementowania animacji wybuchu, który wyświetla się po zniszczeniu jednostki na jej miejscu. Biblioteka pozwala wprawić w ruch wybrany obraz z rozszerzeniem .gif na kilka sposobów.

5.3. EntityFramework

Biblioteka za pomocą której zaimplementowane zostało połączenie z bazą danych Sqlite.

7. Wykorzystana baza danych

W celu implementacji funkcjonalności zapisywania i wczytywania stanu rozgrywki wykorzystano lokalną bazę danych SQLite. Baza ta została wybrana ze względu na szybkość, prostotę w użyciu oraz brak konieczności łączenia się z bazą w celu wymiany informacji między wieloma instancjami aplikacji (w tym przypadku baza może mieć postać pliku należącego do zasobów projektu).

W celu stworzenia bazy danych należało stworzyć klasę **SqliteDbContext** rozszerzającą klasę **DbContext** (Rys. 6). Klasa ta składa się z obiektów typu **DbSet** reprezentujących tabele w bazie danych oraz metody *OnConfiguring()*, której wywołanie przez EntityFramework powoduje automatyczną konfigurację *connection-string'a*. Po zdefiniowaniu klasy za pomocą komend wprowadzanych w konsoli managera pakietów(*Enable-Migrations, Add-Migrations)* zostały stworzone migracje (Rys. 7) będące szablonem do utworzenia lub modyfikacji bazy danych (podejście *Code First).* Po utworzeniu pierwszej migracji został utworzony plik baz danych, który należało dodać do plików projektowych. Po wykonaniu tych kroków można było przejść do definiowania i korzystania z metod związanych z ***CRUD*** (*create*, *read*, *update*, *delete)* umożliwiając przy tym implementacje zapisu i wczytywania stanu gry.

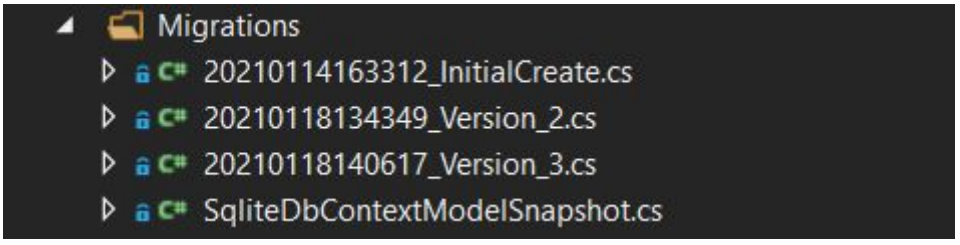
```
class SqliteDbContext : DbContext
{

    1 reference
    public static string DATABASE_NAME { get; } = "PanzerDB.sqlite";


    3 references
    public DbSet<UnitModel> Unit { get; set; }
    4 references
    public DbSet<GameStateModel> GameState { get; set; }

    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlite($"Data source={DATABASE_NAME}");
    }
}
```

Rys. 6 - Klasa używana przez EntityFramework w celu stworzenia bazy danych.

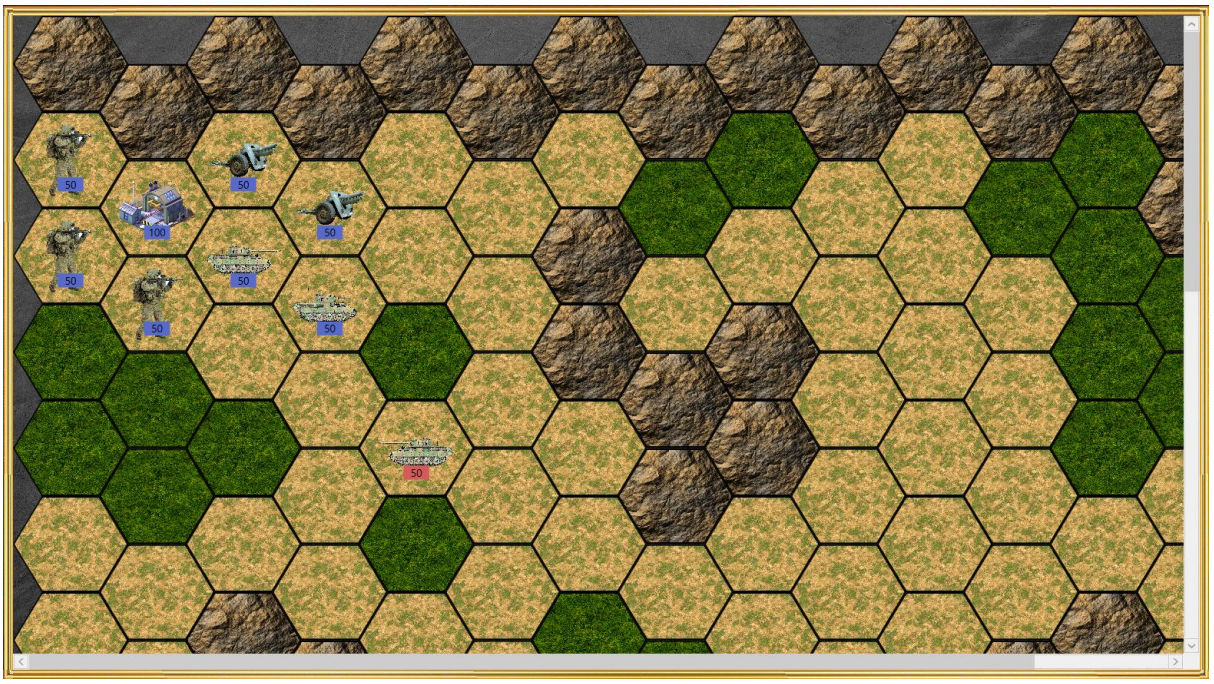


Rys. 7 - Historia migracji w tworzonym projekcie - reprezentacja klasowa

8. Opis GUI

8.1 Okno rozgrywki

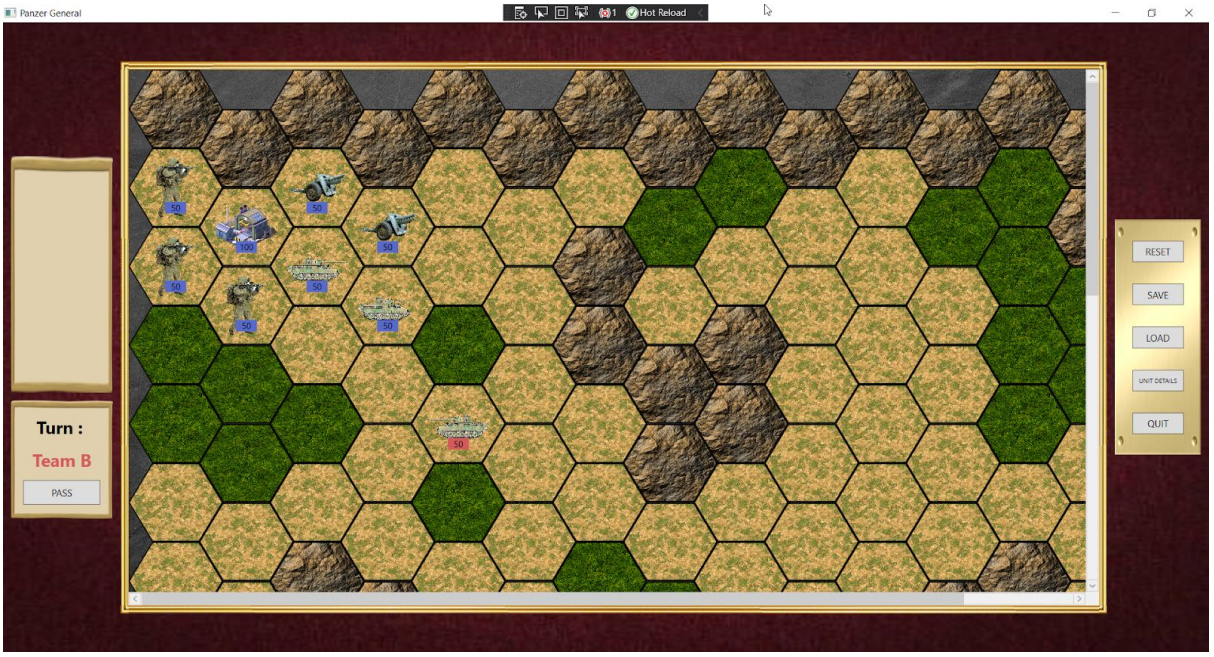
Okno rozgrywki jest obiektem typu **Grid** o 3 kolumnach i 3 wierszach. Powstała macierz złożona z 9 komórek pozwala na wstawienie w skrajne komórki tekstur składających się na ramkę. W centralnej komórce wstawiona jest plansza gry, która dodatkowo zagnieżdżona jest w obiekcie **ScrollViewer**. Umożliwia to przewijanie planszy wertykalnie i horyzontalnie oraz zapobiega rozszerzaniu/scalaniu się heksagonów gdy manipulujemy wielkością okna aplikacji. Powstały efekt widoczny jest na rysunku nr 8. Sama plansza jest obiektem klasy rozszerzającej klasę **ListBox**, gdzie obiektami typu **ListBoxItem** są heksagony. Opisywane rozszerzenie klasy **ListBox** zostało pobrane z biblioteki **HexGridControl**.



Rys. 8 - Okno gry

8.2 Okno aplikacji

Okno aplikacji również jest obiektem typu **Grid** z 9-cioma komórkami. Tym razem w komórce centralnej jest umieszczony cały obiekt o którym była mowa w punkcie **8.0**. W lewej komórce umieszczony jest panel historii ataków, a w prawej komórce - panel przycisków. W panelu historii ataków logowane są zdarzenia potyczki jednostek przedstawiające informacje ile obrażeń zadała dana jednostka w danej potyczce. W panelu przycisków umieszczone są przyciski obsługujące funkcjonalności opisane w punkcie **3**. Okno aplikacji jest skalowalne. Może je rozszerzać i scalać jak tylko nam się podoba, lecz należy pamiętać, że minimalne wymiary okna to **800x600** pikseli. Wygląd okna gry przedstawiono na rysunku nr 9.



Rys. 9 - Okno aplikacji

8.3 Dedykowane okno dialogowe

Aby informować gracza o różnych zdarzeniach i jednocześnie utrzymać stylistykę gry stworzono klasę **PanzerAlertDialog** (Rys.10). Obiekt ten w odróżnieniu od klasycznego **AlertDiaog**'u nie jest tworzony w nowym oknie i może być "podczepiony" pod dowolny obiekt rozszerzający klasę **Panel.** "Podczepienie" objawia się przez ukazanie okienka dialogu na środku panelu oraz zaciemnienie i zablokowanie interakcji z tym panelem (możemy wrócić do aplikacji tylko wtedy gdy obsłużymy dialog). Przykładowy efekt ukazano na rysunku nr 11. Dzięki zastosowaniu wzorca budowniczego i obserwatora dialog możemy dowolnie modyfikować za pomocą metod z klasy **PanzerAlertDialog.Builder** oraz możemy bezpośrednio przekazać co ma się dziać po kliknięciu na dany przycisk dialogu dzięki przekazaniu odpowiednich delegatów do metod *setOnPositiveClickButtonListener()* oraz *setOnNegativeClickButtonListener()*.

Rys. 10 - Dedykowane okno dialogowe

Rys. 11 - Dedykowane okno dialogowe - efekt blokady panelu

9. Wykorzystane wzorce projektowe

9.1. Fabryka

Wzorzec projektowy “Fabryka” wykorzystany został w celu usprawnienia powoływania obiektów konkretnych jednostek (“UnitFactory”) oraz heksagonalnych pól planszy (“HexPointFactory”).

9.2. Obserwator

Wzorzec projektowy “Obserwator” wykorzystany został do zaimplementowania wiązania danych (“data binding’u”) oraz powiadamiania odpowiednich obiektów o zmianie innych obiektów. Przykładem jest powiadamianie UI o zmianie punktów życia wyświetlanej na planszy jednostki po przeprowadzonej walce lub nawet samo powiadomienie o usunięciu danej jednostki z konkretnego pola na planszy.

9.3. Budowniczy

Wzorzec projektowy “Budowniczy” wykorzystany został m.in. do wygodniejszego inicjowania okna dialogowego, które w zależności od potrzeby może być oknem ładowania lub wczytywania.