

Projekt indywidualny - dokumentacja techniczna

Mateusz Bakała

31 maja 2016

1 Cel projektu

Celem projektu jest stworzenie prostej gry Pacman 2000. Niniejszy dokument zawiera pełną dokumentację techniczną do projektu aplikacji, listę użytych technologii, bibliotek oraz wzorców projektowych, które zostaną wykorzystane podczas implementacji. Dodatkowo zostaną wyjaśnione szczegóły dotyczące działania aplikacji, w szczególności opis wykorzystywanych algorytmów oraz diagram stanów i klas dla całej aplikacji. Dokument jest rozszerzeniem dostarczonej specyfikacji "Pacman 2000" i kontekst wszelkich informacji zawartych w tym dokumencie jest podany tamże.

2 Opis aplikacji

Pacman to gra zręcznościowa, w której gracz kieruje kulką - tytułowem pacmanem. Gra toczy się na planszy podobnej do labiryntu z licznymi skrzyżowaniami. Gracz musi kierować Pacmanem i przemieszczać go po planszy w ten sposób, aby zebrać porozrzucane na niej monety, jednocześnie unikając przeciwników. Jeśli przeciwnik dotknie Pacmana, to gracz traci życie i staje się chwilowo nieśmiertelny (o ile jeszcze ma jakieś życia). Gdy gracz zbierze wszystkie monety na planszy, to plansza ponownie zostaje zapełniona monetami, a poziom trudności gry zwiększa się - przeciwnicy stają się szybsi i trudniej przed nimi uciekać. Celem gry jest zdobycie jak największej ilości punktów aż do utraty ostatniego życia.

Aplikacja poza trybem rozgrywki posiada kilka innych trybów. W dowolnym momencie gra może zostać zapauzowana. Możliwy jest zapis i odczyt gry w celu kontynuowania poprzedniej gry. W aplikacji będą pamiętane dodatkowo najlepsze wyniki uzyskane w grze.

3 Opis użytych technologii i bibliotek

Aplikacja zostanie zaimplementowana przy użyciu platformy .NET (wersja 4.6) w języku C# (wersja 6). Główną używaną technologią będzie Windows Presentation Foundation (WPF), zapewniająca bardzo dobre wsparcie w tworzeniu aplikacji desktopowych z zaawansowanym graficznym interfejsem użytkownika.

4 Wymagania sprzętowe i systemowe

Z racji użytych technologii, aplikacja do uruchomienia i poprawnego funkcjonowania wymaga obecności następujących komponentów:

- System Windows 7 lub nowszy
- Zainstalowana platforma .NET w wersji 4.6 (lub wyższej).

Aplikacja powinna uruchamiać się na praktycznie wszystkich komputerach spełniających powyższe wymagania. Do poprawnego i płynnego działania aplikacji zalecane jest dodatkowo:

- Procesor minimum 1GHz
- 800MB pamięci RAM

5 Opis użytych wzorców projektowych

Ze względu na użytą technologię - Windows Presentation Foundation (WPF), która oferuje znakomite wsparcie dla mechanizmu wiązania danych (data binding), naturalnym wydaje się wykorzystanie w aplikacji wzorca architektonicznego Model-View-ViewModel (MVVM). Z uwagi na specyfikę aplikacji - jest to gra, która może wymagać dynamiki, sztywne trzymanie się tego wzorca może być nieco uciążliwe. Dlatego też możliwe będą bardzo niewielkie odstępstwa od niego, które zostaną wymuszone przez implementację logiki gry. Jednakże, podporządkowanie się głównej idei wzorca, czyli odseparowaniu warstwy prezentacji od warstwy logiki będzie ściśle przestrzegane.

Ponadto z powodu użycia właściwości typu DependencyProperty (w kontrolkach dziedziczących z klasy GameElement (patrz sekcja **Diagramy klas**) oraz interfejsu zdarzeń z języka C# zostanie w tych miejscach wykorzystana implementacja wzorca Obserwator (Observer).

Ponadto, do obsługi różnych algorytmów poruszania się wrogów (za pomocą interfejsu IMovementAlgorithm) zostanie zaimplementowany wzorec Strategia.

6 Diagram stanów aplikacji

Tworzona gra jest aplikacją desktopową, zatem będzie ona składała się z co najmniej kilku widoków (ekranów). Ze względu na zastosowanie wzorca MVVM, można wyróżnić naturalny podział aplikacji na następujące stany:

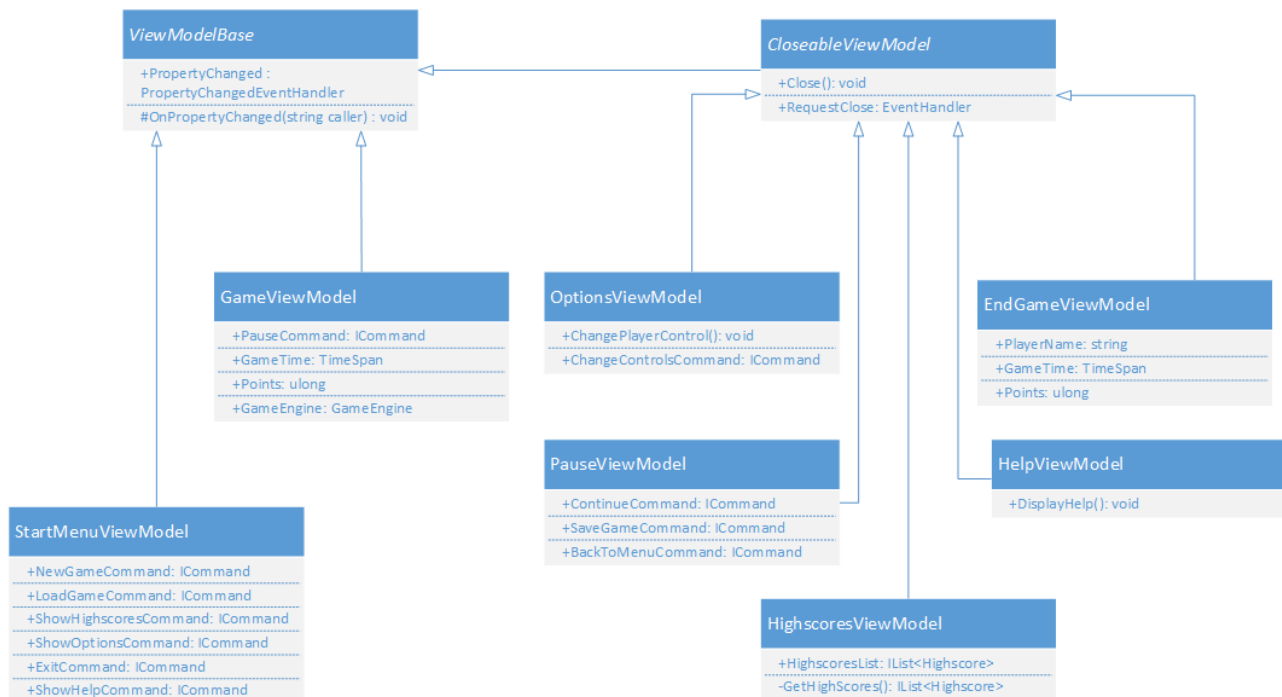
1. Ekran startowy - początkowy ekran, złożony z menu startowego gry. Zawiera ono następujące opcje: Nowa gra, Wczytaj grę, Najlepsze wyniki, Opcje, Pomoc oraz Wyjście.
2. Gra aktywna - widok gry. W tym stanie przebiega rozgrywka.
3. Pauza - tryb pauzy. Rozgrywka zostaje wstrzymana. Gracz może kontynuować grę, zapisać jej stan, lub też powrócić do menu startowego.
4. Widok najlepszych wyników - stan wyświetlający użytkownikowi informację o najlepszych uzyskanych wynikach w grze.

- Szczegółowe zależności i przejścia pomiędzy stanami znajdują się na poniższym diagramie stanów UML:



Poniżej przedstawione zostały uproszczone diagramy klas dla aplikacji.

Poniższy diagram jest diagramem dla klas typu ViewModel. Są to klasy obsługujące użycie wzorca MVVM. Współpracują one z widokami (które będą napisane w XAMLu), udostępniają im dane do wyświetlania oraz zarządzają przełączaniem się widoków.

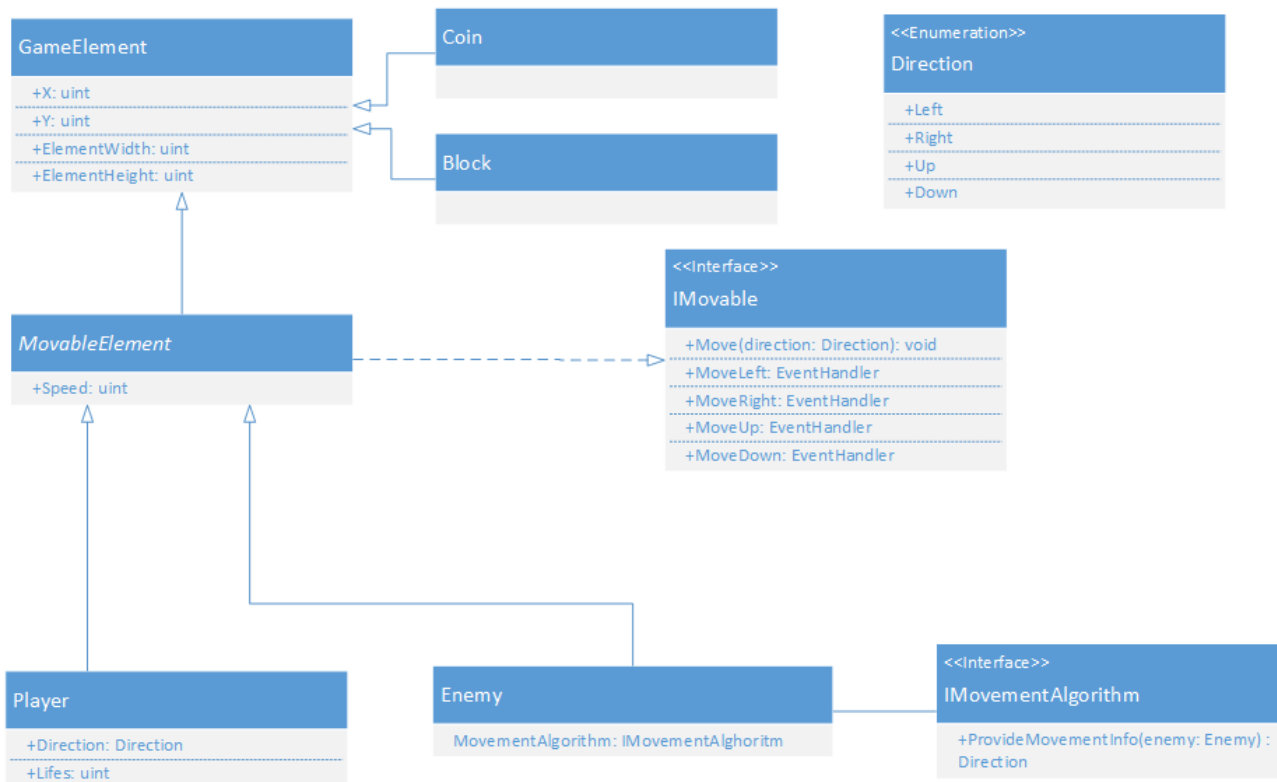


Rysunek 2: Diagram klas dla ViewModeli

Bazowa klasa abstrakcyjna `ViewModelBase` zapewnia podstawową implementację interfejsu `INotifyPropertyChanged`, który pozwala na łatwe wsparcie powiadamiania widoków o zmianie właściwości w klasie (poprzez mechanizm data binding). Dodatkowo klasa `CloseableViewModel` obsługuje widoki, które mogą zostać zamknięte lub przełączone przez użytkownika. Pozostałe z przedstawionych klas, odpowiadają za obsługę konkretnych ekranów (widoków) w grze. Dodatkowo klasa `GameViewModel` zawiera nieopisany na diagramie obiekt typu `GameEngine`. Jest to obiekt odpowiedzialny za logikę gry i zostanie opisany później.

7.2 Diagram klas dla elementów gry

Elementem w grze będziemy nazywali każdy graficzny element, który może zostać umieszczony na planszy do gry. Poniżej przedstawiony jest diagram klas dla elementów:



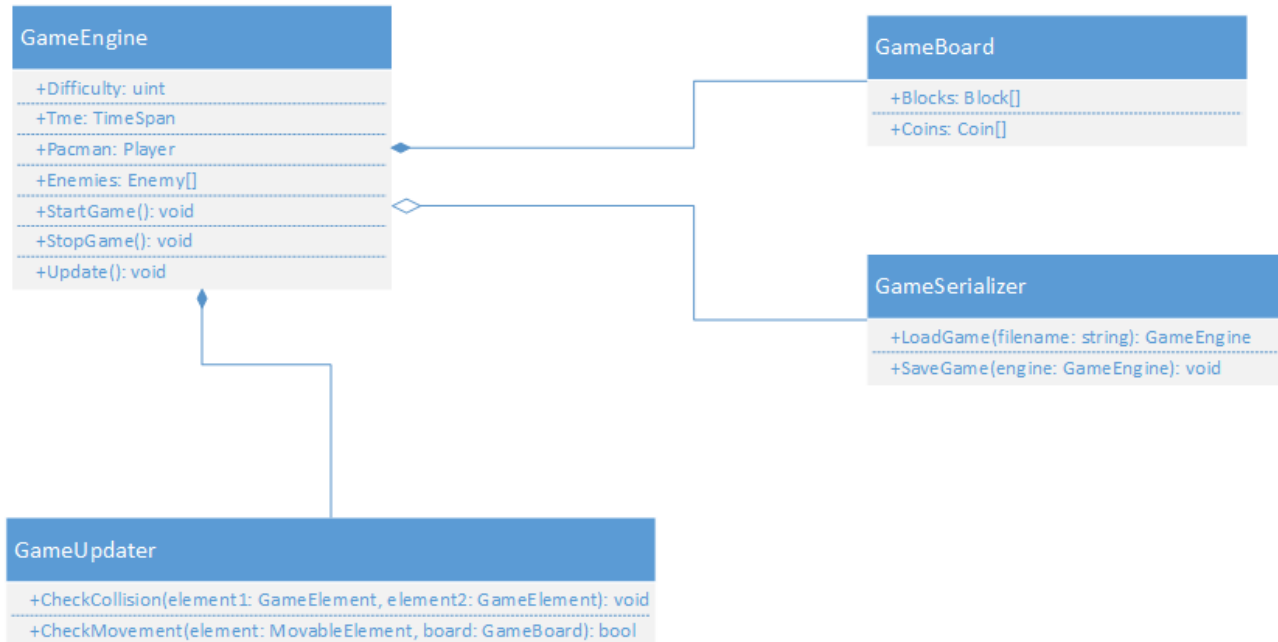
Rysunek 3: Diagram klas dla elementów w grze

Każda klasa, poczynając od `GameElement`, dziedziczy (bezpośrednio lub pośrednio) z klasy `CustomControl`, będąc zatem kontrolką WPF. Zapewnia to tym klasom bycie częścią graficznego interfejsu użytkownika. Zgodnie z filozofią technologii WPF, kontrolki takie mają odseparowany wygląd graficzny od logiki. Zatem wszystkie przedstawione tu klasy są odpowiedzialne za logikę, a nie definiują żadnych cech wizualnych związanych z widocznymi elementami w graficznym interfejsie użytkownika. Takie właściwości zostaną podane za pomocą obiekty `Style` (z technologii WPF) w osobnym pliku.

Jak widać wprowadzony został podział i hierarchia klas, która łatwo pozwala rozszerzać kolejne elementy o dodatkowe funkcjonalności (np. możliwość poruszania się). Klasa `GameElement` zawiera właściwości definiujące pozycję i wielkość danego elementu. Klasy odpowiedzialne za obsługę gracza i przeciwników (odpowiednio `Player` i `Enemy`) dziedziczą z `MovableElement` zdolność poruszania się i bazowy interfejs ruchu. Dodatkowo klasa `Enemy` została wyposażona w obiekt typu interfejsu `IMovementAlgorithm`, który będzie dostarczał informację o kierunku, w którym powinien się poruszyć wróg. Jest to interfejs, ponieważ możliwe jest zaimplementowanie różnych sposobów i algorytmów poruszania się przeciwników w zależności od poziomu trudności (patrz również **Uwaga** w sekcji **Opis algorytmów**).

7.3 Logika gry

Poniżej przedstawiono diagram klas odpowiedzialnych za zarządzanie logiką gry.



Rysunek 4: Diagram klas definiujących logikę gry

Główną klasą realizującą logikę gry jest klasa **GameEngine**. Zawiera ona podstawowe składowe związane z grą: informacje o planszy, graczu i przeciwnikach oraz definiuje podstawowe operacje konieczne do rozgrywki: jej rozpoczęcie i zatrzymanie. Przedstawiono ponadto klasy **GameUpdater** oraz **GameSerializer** odpowiedzialne za odpowiednio aktualizację stanu gry oraz zapis i odczyt gry. Ponadto klasa **GameBoard** jest kontrolką WPF typu `UserControl` i realizuje graficzną stronę obsługi planszy do gry.

8 Opis algorytmów

8.1 Opis algorytmu przemieszczania się wrogów

Algorytm będzie naiwny. Prosty schemat działania: przeciwnik musi się poruszać w stronę Pacmana. Są 4 możliwości ruchu (lewo, prawo, góra, dół). Oczywiście z uwagi na kształt planszy, nie zawsze wszystkie możliwości ruchu są dozwolone. Wróg lokalizuje Pacmana, a następnie stwierdza jak w stosunku do niego jest położony i wybiera spośród dozwolonych kierunków ruchu ten, który przybliży go do gracza. Jeśli ma do wyboru więcej niż jeden taki kierunek (np. Pacman jest położony w górę i w lewo od wroga to wybiera losowy z tych kierunków). W przypadku gdy kierunki, które przybliżą wroga do gracza są zablokowane przez

ścianę planszy, to wróg porusza się w losowym dostępnym kierunku.

Uwaga: Być może oprócz powyższego algorytmu poruszania wrogów, zostanie zaimplementowany również inny (bardziej inteligentny) sposób poruszania się przeciwników w stronę gracza. Przykładem może być wykorzystanie algorytmu A dzięki któremu przeciwnicy będą mogli wykonywać ruchy w kierunku gracza po najkrótszej możliwej ścieżce na planszy (traktowanej jako graf). Autor niniejszego dokumentu zostawia sobie prawo do podjęcia ostatecznej decyzji w tej kwestii w terminie późniejszym. W przypadku implementacji lub jej braku, końcowa dokumentacja techniczna zostanie odpowiednio uzupełniona i zmodyfikowana.