Projekt indywidualny - dokumentacja techniczna

Mateusz Bąkała

1 czerwca 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 scecyfikacji "Pacman 2000" i kontekst wszelkich informacje 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 nięsmiertelny (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 klika 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. Poza tym zostały użyte następujące biblioteki (doinstalowane do projektu jako pakiety nuget):

1. Prism (wersja 6.1.0) - bibloteka wspierająca użycie wzorca architektonicznego MVVM, typowego dla aplikacji desktopowych tworzonych w technologii WPF.
2. Fody (wersja 1.29.2) - biblioteka oferująca automatyczne wstrzykiwanie implementacji interfejsu INotifyPropertyChanged oraz komend.
3. Autofac (wersja 3.5.2) - biblioteka wspierająca implementację z zastosowaniem wzorca Dependency Injection.
4. MahApps.Metro (wersja 1.2.4) - biblioteka ze stylami do interfejsu graficznego.
5. xunit (wersja 2.1.0) - biblioteka do tworzenia testów jednostkowych.
6. Moq (wersja 4.5.8) - biblioteka służąca do mockowania używanego w testach jednostkowych.

4 Wymagania sprzętowe i systemowe

Z racji użytych technologii, aplikacja do uruchomienia i poprawnego funkcjonaowania 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 powyżesze wymagania. Do poprawnego i płynnego działania aplikacji zalecane jest dodatkowo:

- Procesor minimum 1GHz
- 800MB pamięci RAM

5 Opis użytych wzorców projektowych i architektonicznych

1. 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.
2. 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).

3. Do obsługi różnych algorytmów poruszania się wrogów (za pomocą interfejsu IMovementAlgorithm) w implementacji zostanie wykorzystany wzorzec Strategia.

4. W implementacji procesu tworzenia planszy do gry zostanie użyty wzorzec budowniczego (Builder) (poprzez implementację interfejsu IGameBuilder (patrz sekcja **Diagramy klas**)

Ponadto w implementacji będą przestrzegane reguły dobrego stylu programowania obiektowego poprzez przestrzeganie zasad SOLID (w szczególności Single Responsibility Principle oraz Dependency Injection Principle).

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 oraz Wyjście.

2. Gra aktywna - widok gry. W tym stanie przebiega rozgrywka.

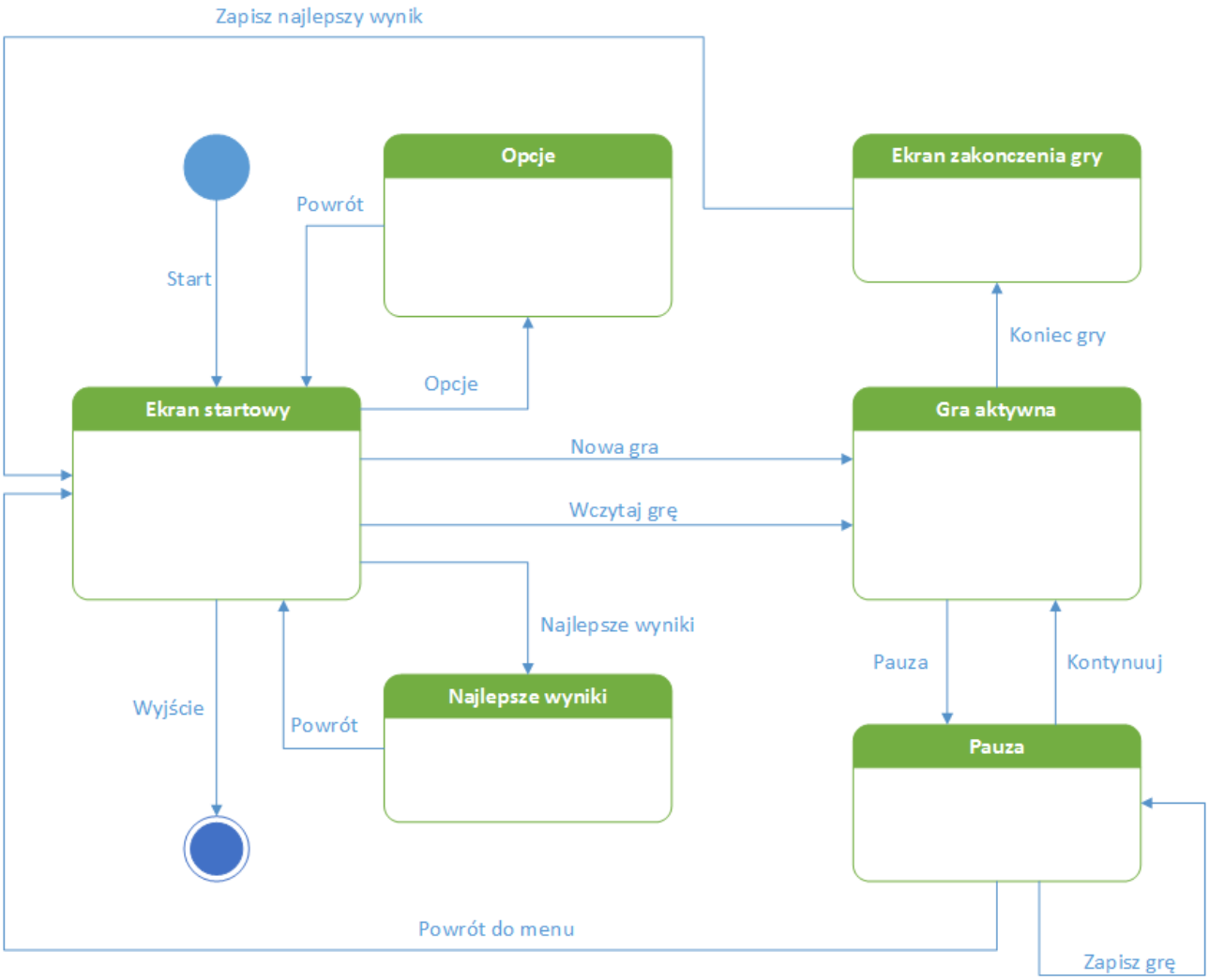
3. Pauza - tryb pauzy. Rozgrywka zostaje wstrzymana. Gracz może kontynuować grę, zapisać jej stan, lub też powrócić do menu startowego.

4. Widok najlepszych wyników - stan wyświetlający użytkownikowi informację o najlepszych uzyskanych wynikach w grze.

5. Widok opcji - pozwala użytkownikowi zmienić klawisze sterowania.

6. Ekran końca gry - pozwala na zapis najlepszego wyniku.

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



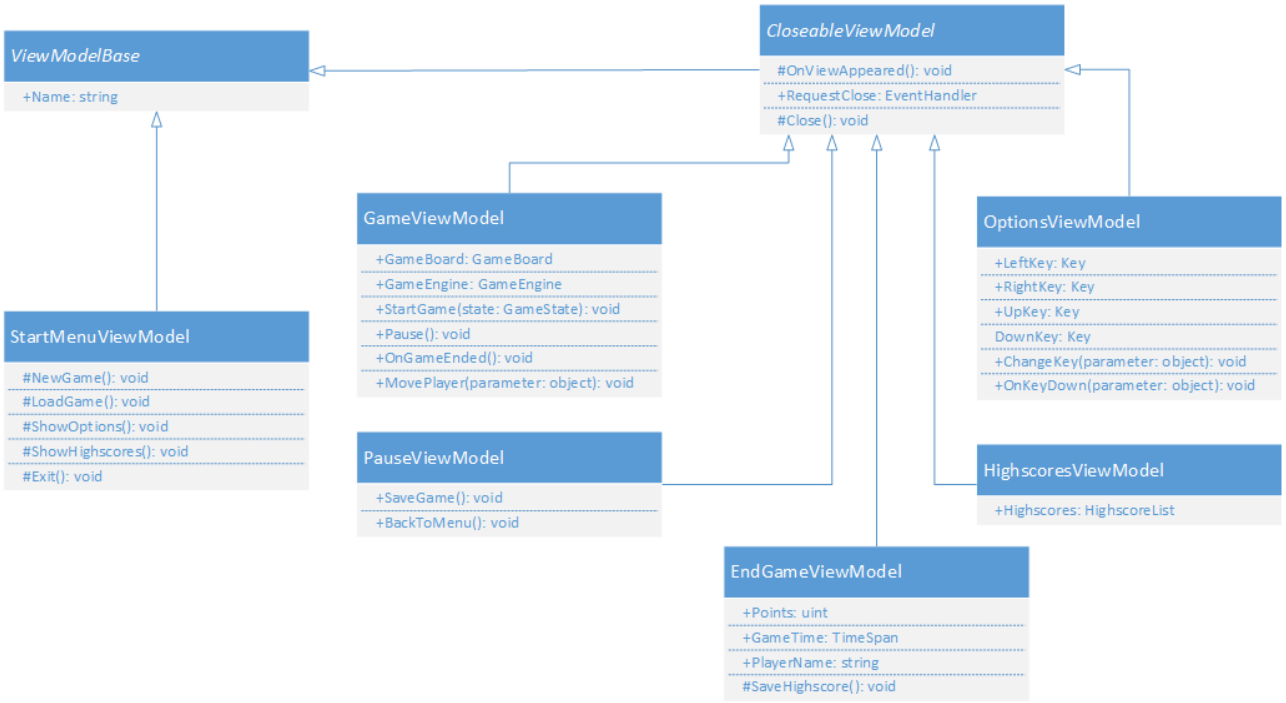
Rysunek 1: Diagram stanów dla całej aplikacji

7 Diagramy klas

Poniżej przedstawione zostały uproszczone diagramy klas dla aplikacji.

7.1 Klasy ViewModel

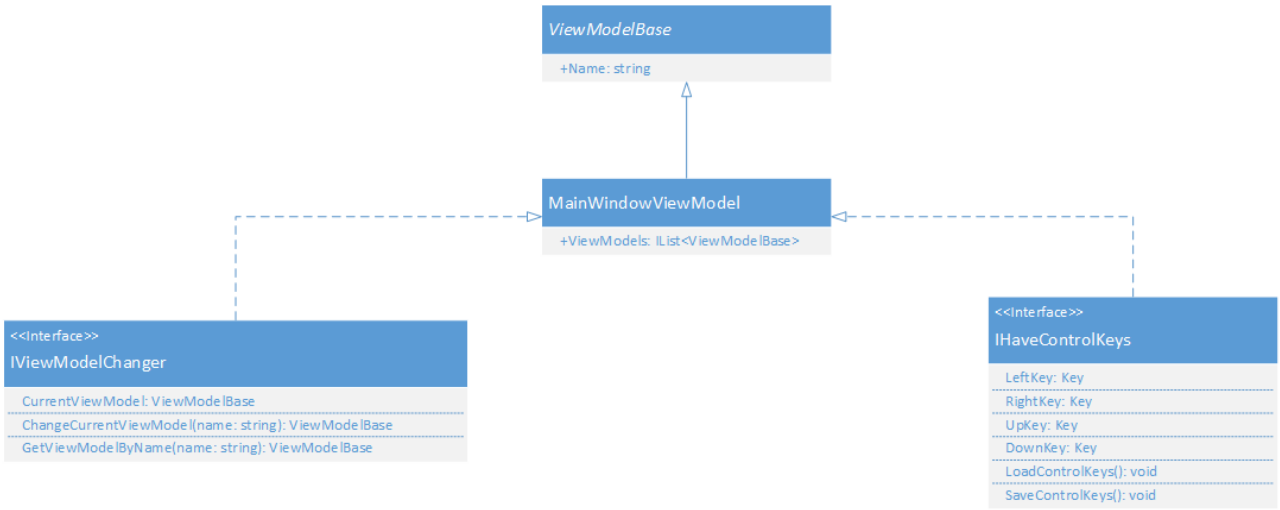
Poniższy diagram jest diagramem dla klas typu ViewModel. Są to klasy obsługujące użycie wzorca MVVM. Współpracują one z widokami (które będą napisane w XAMLu) oraz udostępniają im dane do wyświetlania.



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.

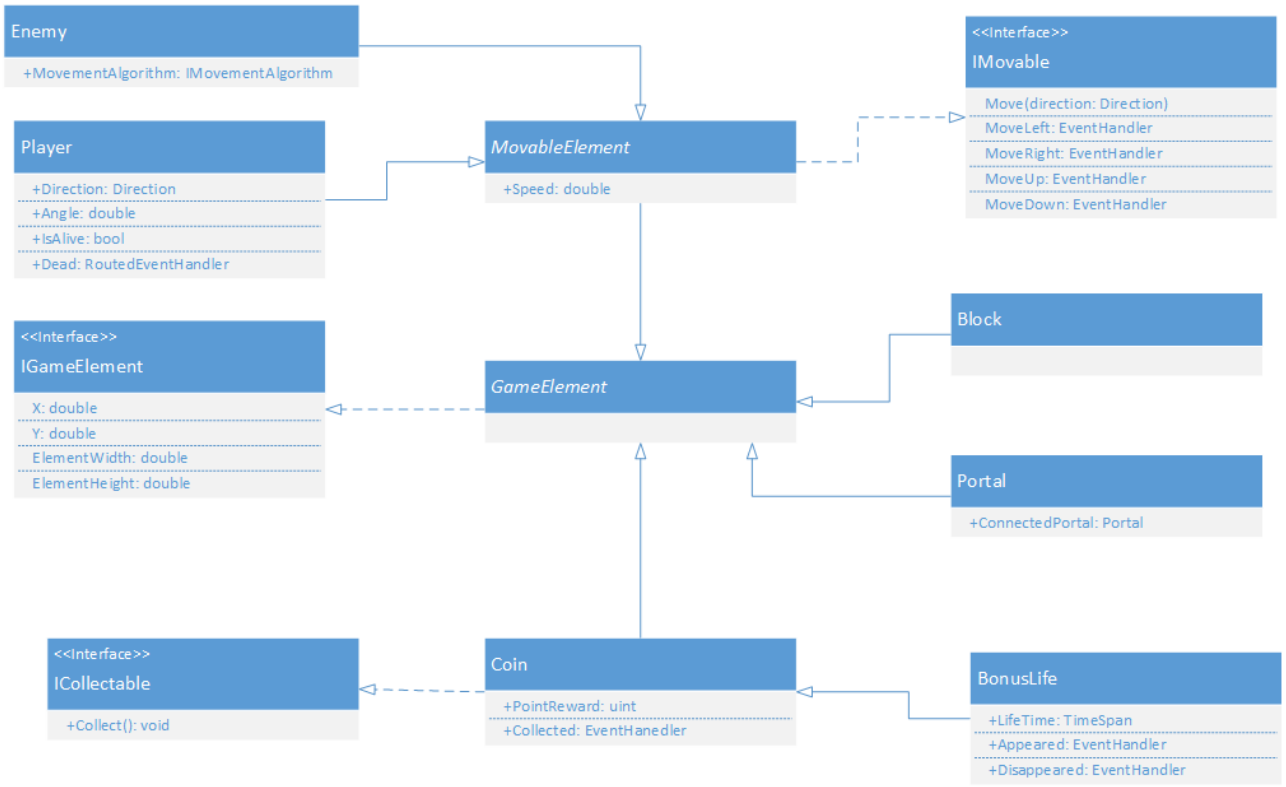
Istnieje jeszcze jedna bardzo ważna klasa dziedzicząca z klasy ViewModelBase o nazwie MainWindowViewModel. Zapewnia ona interakcję z głónym oknem aplikacji. Odpowiada w szczególności za obsługę przełączania widoków (implementując interfejs IViewModelChanger). Ponadto udostępnia ona dla widoku głównego okna aplikacji właściwości z przyciskami, których wciśnięcie powoduje ruch gracza (właściwości te udostępnia interfejs IHaveControlKeys).



Rysunek 3: Diagram klas dla klasy MainWindowViewModel

7.2 Diagram klas dla elementów gry

Elementem w grze będziemy nazywali każdy graficzny element, który może zostać umieszczony na planszy do gry.Każda klasa, poczynając od GameElement, dzidziczy (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 obiektów Style (z technologii WPF) w osobnym pliku.
Poniżej przedstawiony jest diagram klas dla elementów:

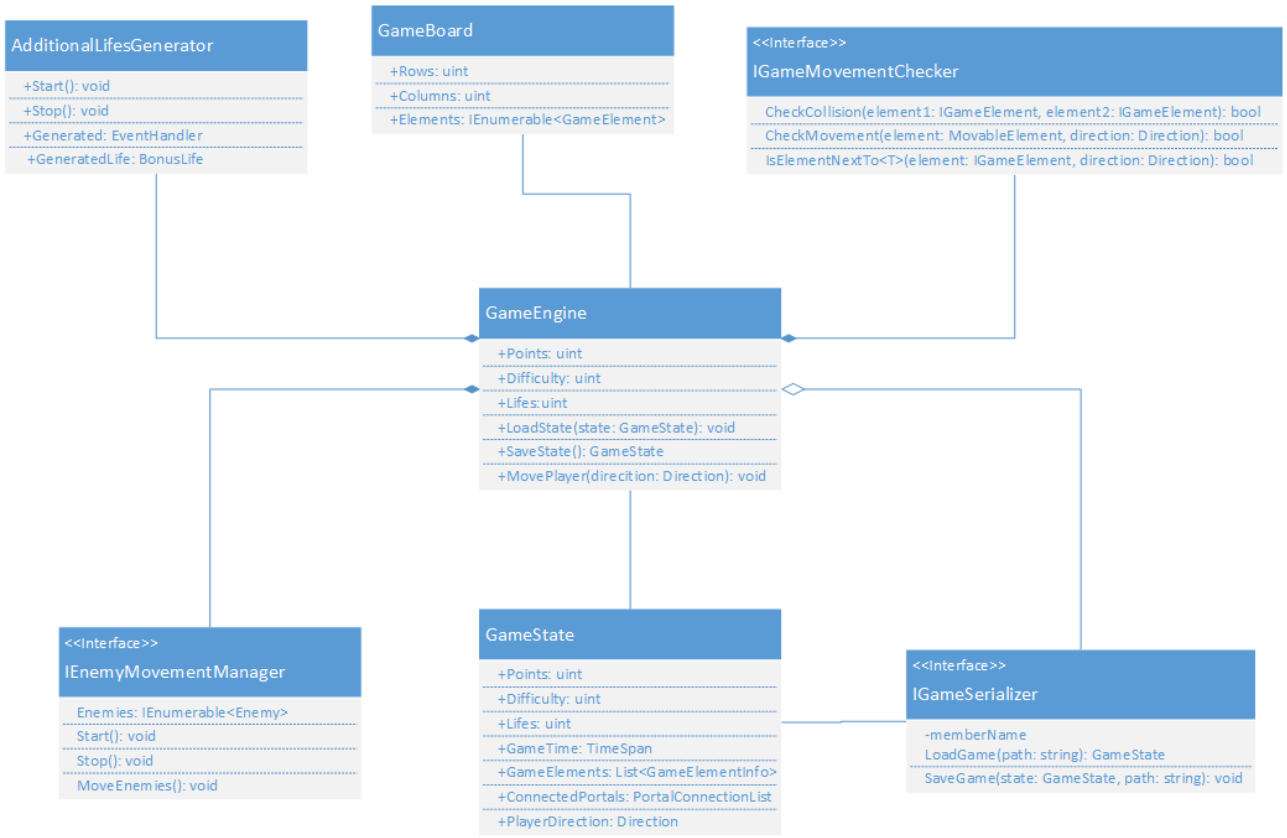


Rysunek 4: Diagram klas dla elementów w grze

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 definujące 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 IMovementAlghorithm, 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**).
Odrębną grupę klas stanowią klasy Coin oraz BonusLife, które implementuja interfejs ICollectable. Są to elementy, których zebranie powoduje zdobycie punktów przez gracza.
Ponadto z GameElement dziedziczy również klasa Portal, która reprezentuje obiekt na planszy umożliwiający teleportację do innego połączonego z nim portalu na planszy.

7.3 Logika gry

Poniżej przedstawiono diagram klas odpowiedzialnych za zarządzanie logiką gry.



Rysunek 5: Diagram klas definiujących logikę gry

Główną klasą realizującą logikę gry jest klasa GameEngine. Zawiera ona podstwowe 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

Zostanie zaimplementowany inny niż zaproponowany w dokumentacji wstępnej (bardziej inteligentny i co za tym idzie również nieco bardziej skomplikowany)

sposób poruszania się przeciwników w stronę gracza. Wykorzystany zostanie grafowy algorytm A, dzięki któremu przeciwnicy będą mogli wykonywać ruchy w kierunku gracza po najkrótszej mozliwej ścieżce na planszy (traktowanej jako graf). Dzięki temu zostanie osiągnięty znacząco trudniejszy poziom rozgrywki, niż w przypadku algorytmu nawinego proponowanego w dokumentacji wstępnej.