Miner

Dokumentacja Techniczna

Paweł Kobojek

2 kwietnia 2014

Zawartość

1. Wprowadzenie

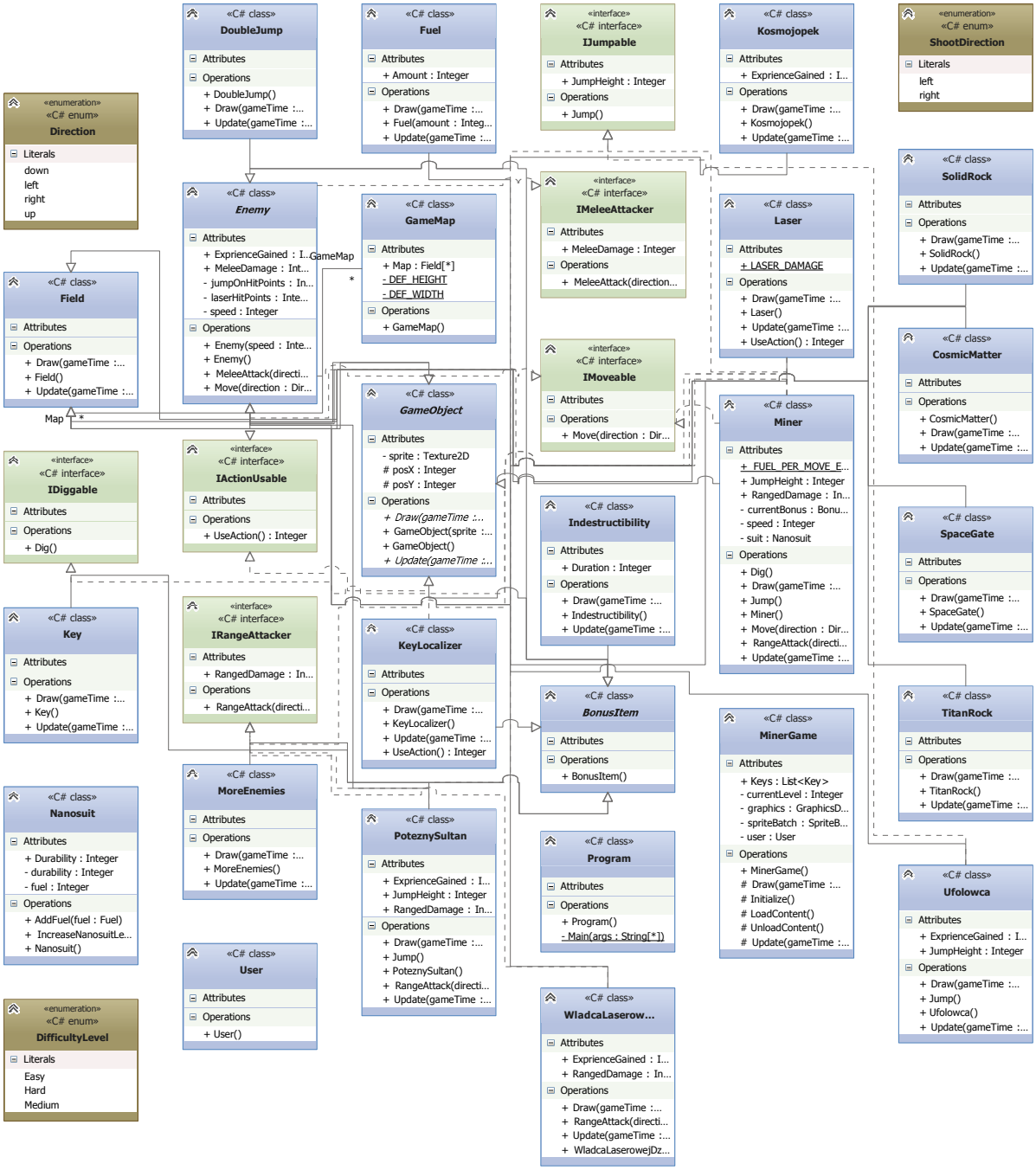
Ten dokument stanowi dokumentację techniczną gry Miner. Autorem specyfikacji biznesowej jest Kamil Żak. Dokument zawiera diagramy klas UML wraz z opisem, a także opis najważniejszych algorytmów i organizacji struktury plików.

2. Technologia

Do zaimplementowania gry zostanie użyta biblioteka Microsoft XNA 4.0. Kod programu będzie napisany w środowisku .NET w języku C#.

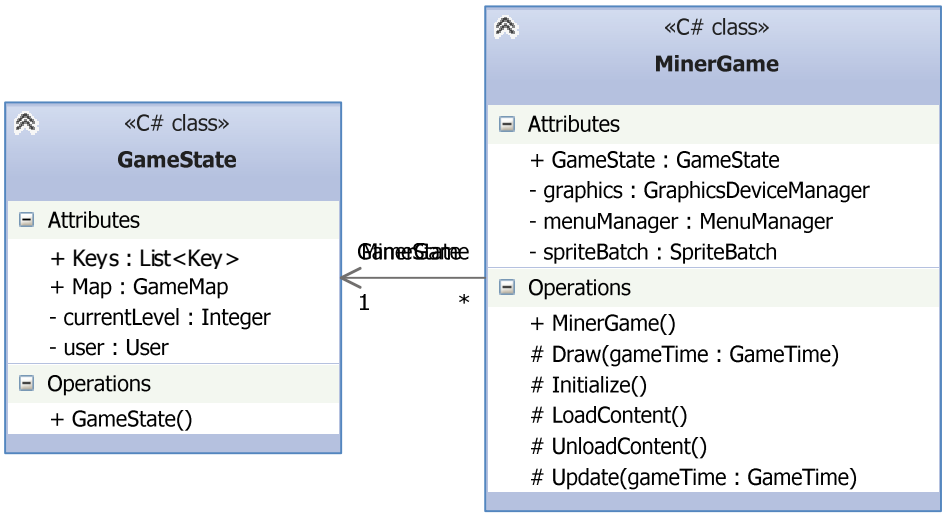
3. Diagramy klas UML

Poniżej przedstawiony jest diagram UML zawierający wszystkie klasy, które zostaną zaimplementowane w grze (za wyjątkiem klas związanych z renderowaniem menu, patrz: rozdział 4 oraz ustawieniami, patrz: rozdział 5). Z uwagi na ich ilość diagram ma charakter jedynie poglądowy i nie jest zbyt czytelny. Szczegółowe opisy wszystkich klas znajdą się w dalszej części tego dokumentu.



3.1 Klasa MinerGame

Główna klasa gry. Dziedziczy z klasy Microsoft.Xna.Framework.Game.



Pola (Właściwości):

```
public GameState GameState
```

Pole przechowujące stan gry. Patrz: 3.31

```
private MenuManager menuManager
```

Pole służące do zmiany aktualnie wyświetlanego menu.

Konstruktory:

```
public MinerGame()
```

Metody:

```
protected override void Initialize()
```

Metoda odpowiadająca za inicjalizację zasobów potrzebnych do uruchomienia gry.

```
protected override void LoadContent()
```

Metoda odpowiadająca za ładowanie zasobów graficznych.

```
protected override void UnloadContent()
```

Metoda odpowiadająca zwolnienie zasobów.

```
protected override void Update(GameTime gameTime)
```

Metoda odpowiadająca za aktualizowanie stanu gry.

Parametry:

GameTime gameTime czas jaki upłynął od ostatniego wywołania tej metody.

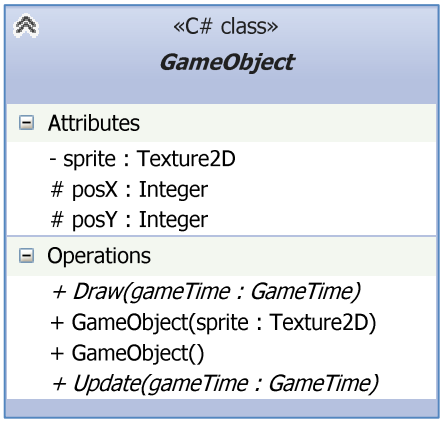
```
protected override void Draw(GameTime gameTime)
```

Metoda odpowiadająca za renderowanie sceny.

Parametry:

gameTime Czas jaki upłynął od ostatniego wywołania tej metody.

3.2 Klasa GameObject



Abstrakcyjna klasa bazowa dla wszystkich obiektów wyświetlanych na scenie.

Pola:

```
private Texture2D sprite
```

Pole przechowujące informacje o obrazku powiązanym z tym obiektem.

```
protected int posX
```

Pole informujące o współrzędnej X położenia obiektu na mapie. Stanowi indeks pola w tablicy reprezentującej mapę (patrz: klasa GameMap, pole map)

```
protected int posY
```

Pole informujące o współrzędnej Y położenia obiektu na mapie. Stanowi indeks pola w tablicy reprezentującej mapę (patrz: klasa GameMap, pole map)

Konstruktory:

```
public GameObject(Texture2D sprite)
```

Konstruktor przyjmujący obrazek.

Metody:

```
public abstract void Draw(GameTime gameTime)
```

Metoda odpowiadająca za renderowanie obiektu na scenie.

Parametry:

gameTime Czas jaki upłynął od ostatniego wywołania tej metody.

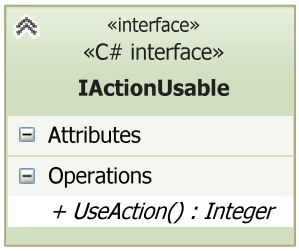
```
public abstract void Update(GameTime gameTime)
```

Metoda odpowiadająca za uaktualnianie stanu związanego z obiektem.

Parametry:

gameTime Czas jaki upłynął od ostatniego wywołania tej metody.

3.3 Interfejs IActionUsable



Interfejs zapewniający możliwość bycia użytym przez gracza za pomocą przycisku akcji.

Metody:

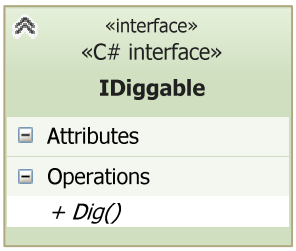
```
public int UseAction()
```

Metoda wywoływana po wciśnięciu przez gracza przycisku akcji.

Wartość zwracana:

int Znaczenie zależne od implementacji.

3.4 Interfejs IDiggable



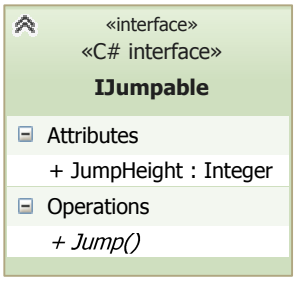
Interfejs gwarantujący możliwość wykonania kopania.

Metody:

```
public void Dig()
```

Metoda odpowiadająca za kopanie.

3.5 Interfejs IJumpable



Interfejs gwarantujący możliwość skakania.

Pola (Właściwości):

```
public int JumpHeight
```

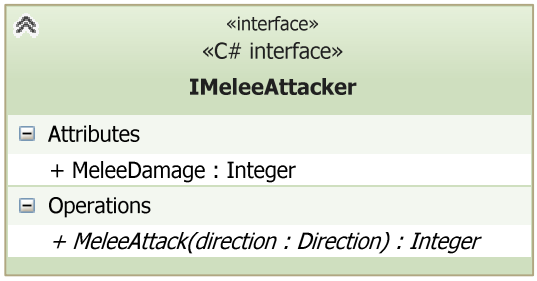
Pole określające wysokość skoku (wyrażoną w polach).

Metody:

```
public void Jump()
```

Metoda odpowiadająca za wykonanie przez postać skoku na wysokość równą JumpHeight.

3.6 Interfejs IMeleeAttacker



Interfejs zapewniający możliwość zadawania obrażeń w zwarciu.

Pola (Właściwości):

```
public int MeleeDamage
```

Pole określające wartość obrażeń zadawanych w zwarciu.

Metody:

```
public int MeleeAttack(Direction direction)
```

Metoda wywoływana przez postać atakującą w zwarciu.

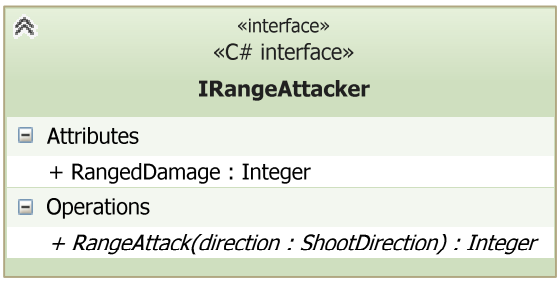
Parametry:

direction	wyliczenie określające kierunek ataku

Wartość zwracana:

int	wartość zadanych obrażeń

3.7 Interfejs IRangeAttacker



Interfejs zapewniający możliwość zadawania obrażeń na odległość.

Pola (Właściwości):

```
public int RangedDamage
```

Pole określające wartość obrażeń zadawanych w zwarciu.

Metody:

```
public int RangeAttack(ShootDirection direction)
```

Metoda wywoływana przez postać atakującą na odległość.

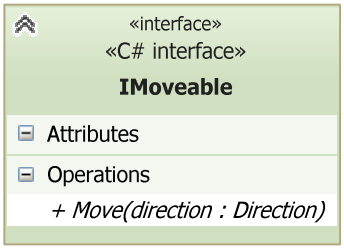
Parametry:

direction	wyliczenie określające kierunek strzału

Wartość zwracana:

int	wartość zadanych obrażeń

3.8 Interfejs IMoveable



Interfejs zapewniający możliwość poruszania się po mapie.

Metody:

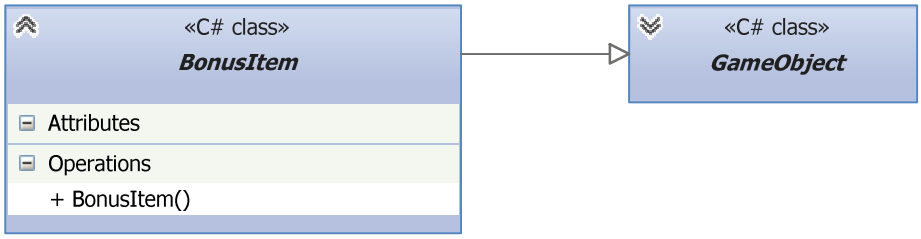
```
public void Move(Direction direction)
```

Ruch obiektu o jedno pole w kierunku Direction.

Parametry:

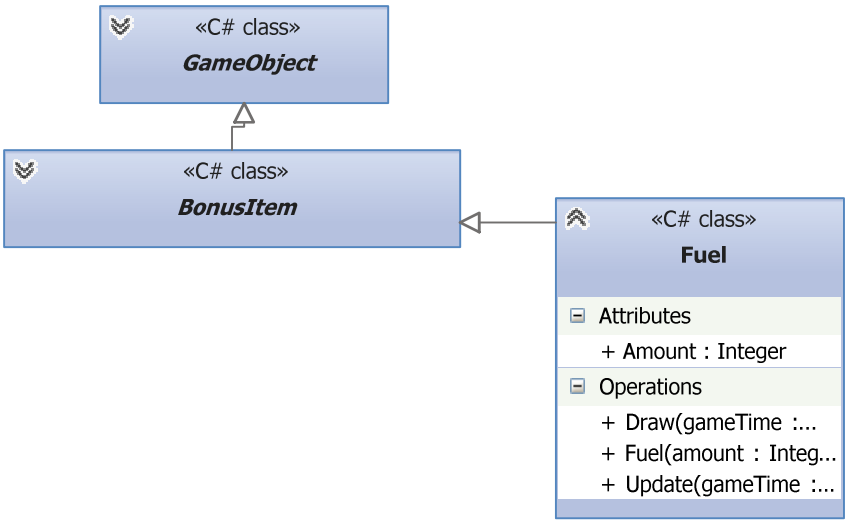
direction	wyliczenie określające kierunek ruchu

3.9 Klasa BonusObject



Abstrakcyjna klasa bazowa dla wszystkich bonusów występujących w grze.

3.10 Klasa Fuel



Klasa określająca Paliwo (bonus).

Pola (Właściwości):

`public int Amount`

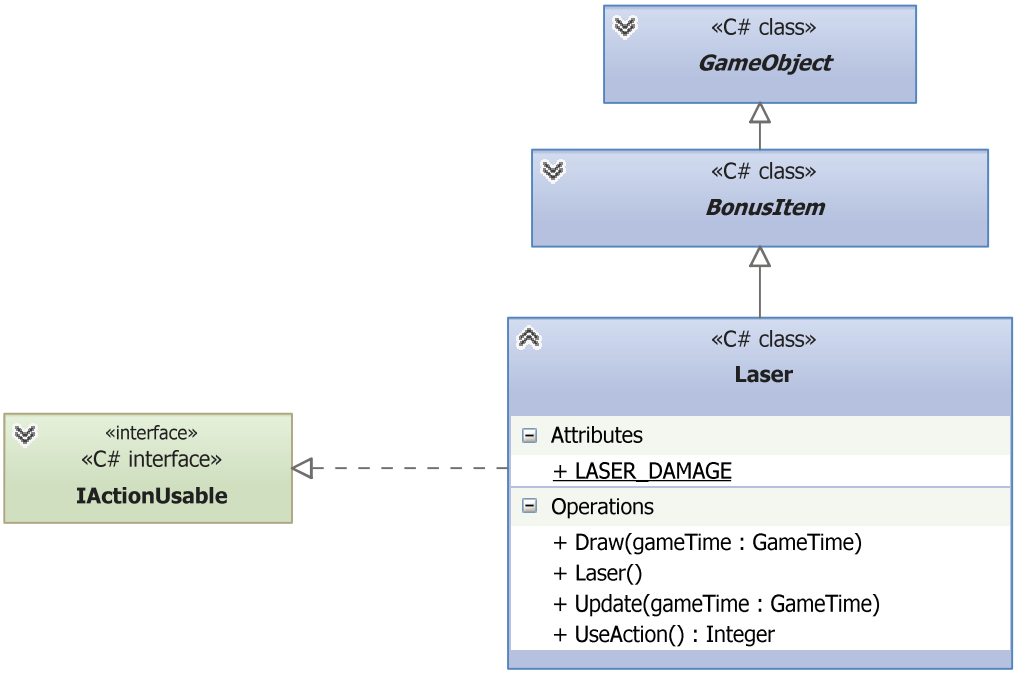
Ilość paliwa w danym bonusie. Gdy gracz zbierze ten bonus to ilość paliwa jego skafandra jest zwiększana o wartość tego pola.

Konstruktory:

`public Fuel(int amount)`

Konstruktor tworzący obiekt wraz z zadaną ilością paliwa.

3.11 Klasa Laser



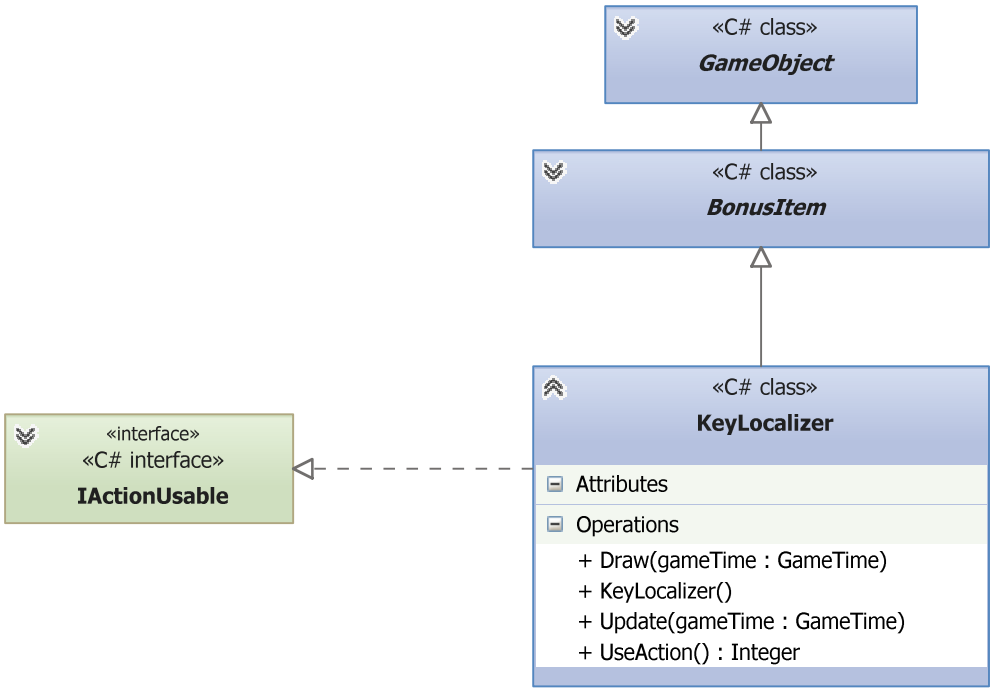
Klasa odpowiadająca za bonus Laser.

Pola (Właściwości):

```
public const int LASER_DAMAGE
```

Stała określająca wartość obrażeń zadawanych przez laser.

3.12 Klasa KeyLocalizer



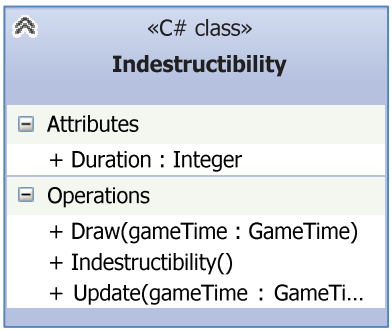
Klasa odpowiadająca za bonus Lokalizator Kluczy

Metody:

```
public int UseAction()
```

Implementacja interfejsu IActionUsable. Wywołanie tej metody powoduje odkrycie trzech pól pod używającym go.

3.13 Klasa Indestructibility



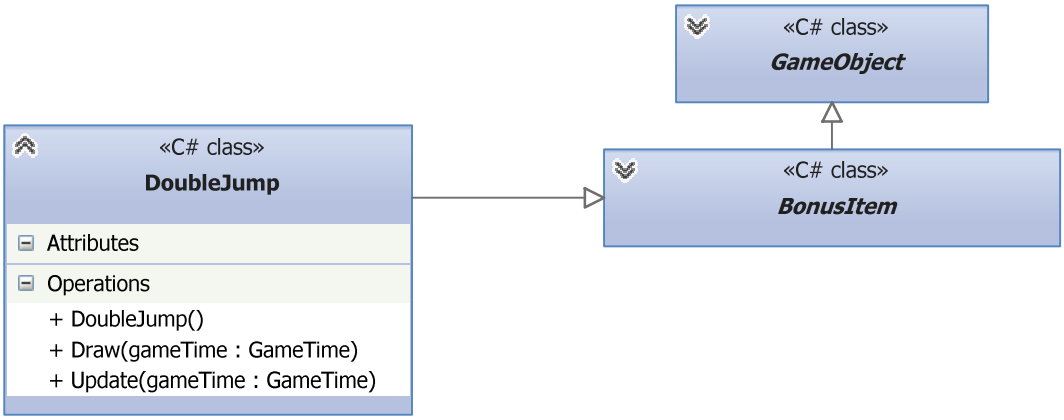
Klasa odpowiadająca za bonus Niezniszczalność.

Pola (Właściwości):

```
public int Duration
```

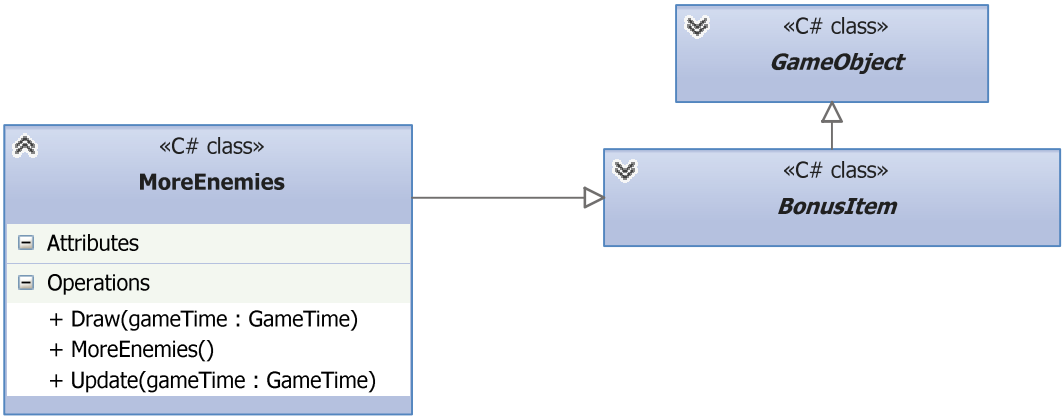
Czas trwania bonusu (w sekundach).

3.14 Klasa DoubleJump



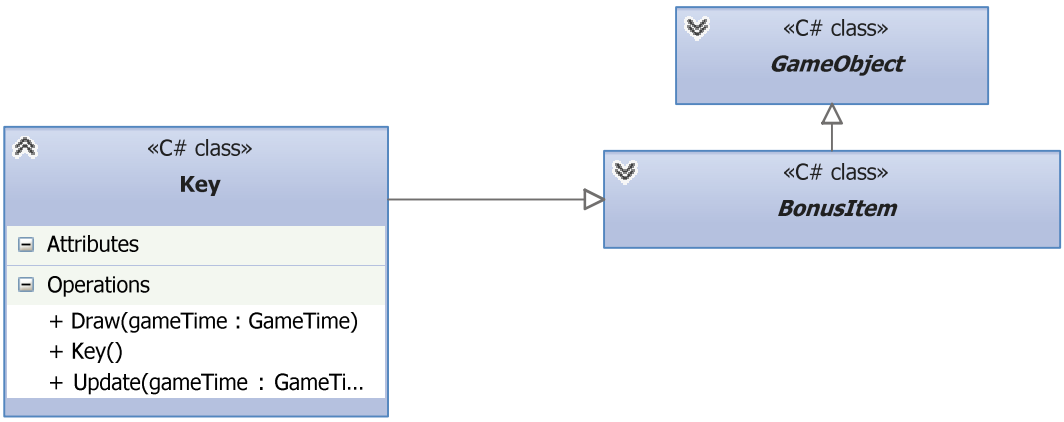
Klasa odpowiadająca za bonus Podwójny Skok.

3.15 Klasa MoreEnemies



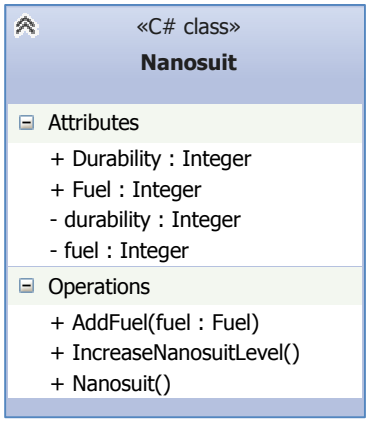
Klasa odpowiadająca za bonus Zwiększona Liczba Wrogów.

3.16 Klasa Key



Klasa odpowiadająca za bonus Klucz.

3.17 Klasa Nanosuit



Klasa reprezentująca skafander Minera.

Pola (Właściwości):

```
public int Durability
```

Określa aktualna wytrzymałość skafandra. Jeśli spadnie do 0 (lub mniej) to gra się kończy.

```
public int Fuel
```

Ilość paliwa. Spada co ruch. Jeśli spadnie do 0 to gra się kończy.

Metody:

```
public void AddFuel(Fuel fuel)
```

Metoda dodająca ilość paliwa w zależności od bonusu który odnaleziono.

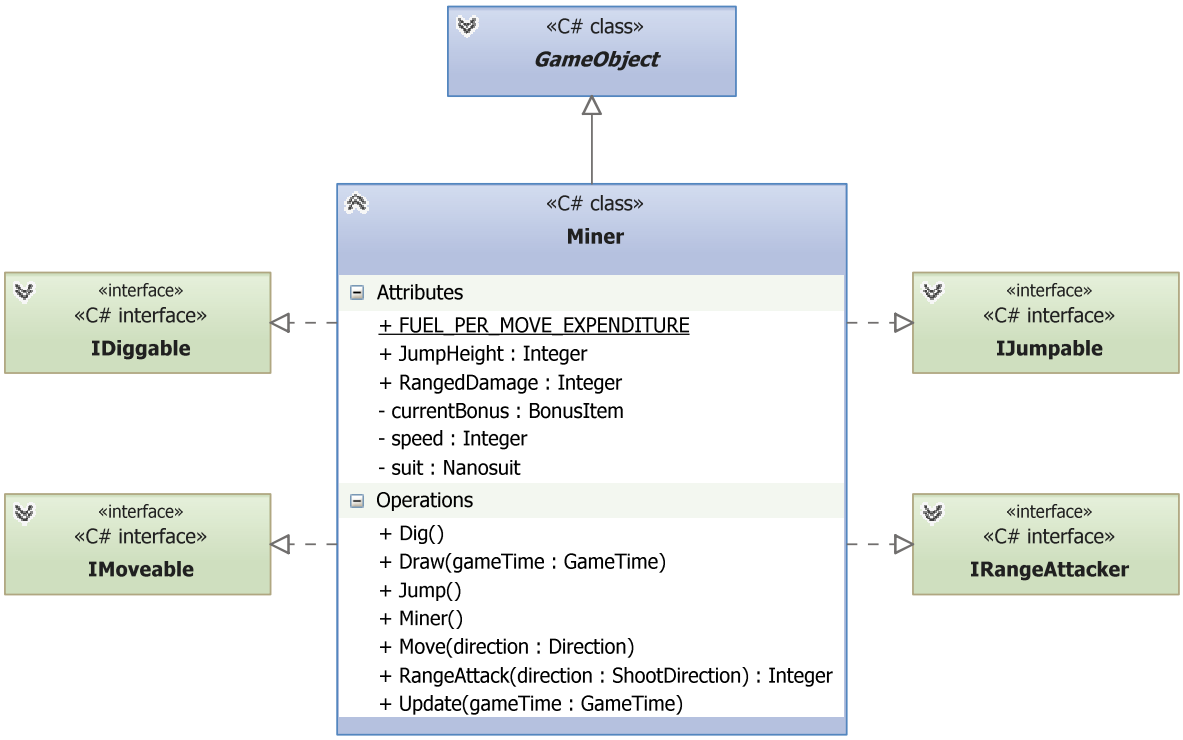
Parametry:

fuel	Bonus który ma być dodany.

```
public void IncreaseNanosuitLevel()
```

Metoda zwiększająca maksymalne ilości wytrzymałości i paliwa (* 1.5 co poziom doświadczenia). Wywoływana wraz ze wzrostem poziomu doświadczenia postaci.

3.18 Klasa Miner



Klasa reprezentująca postać sterowaną przez gracza.

Pola (Właściwości):

`public const int` **`FUEL_PER_MOVE_EXPENDITURE`**

Stała określająca zużycie paliwa na ruch.

`private BonusItem` **`currentBonus`**

Aktualny bonus utrzymywany przez postać (null jeśli nie ma żadnego).

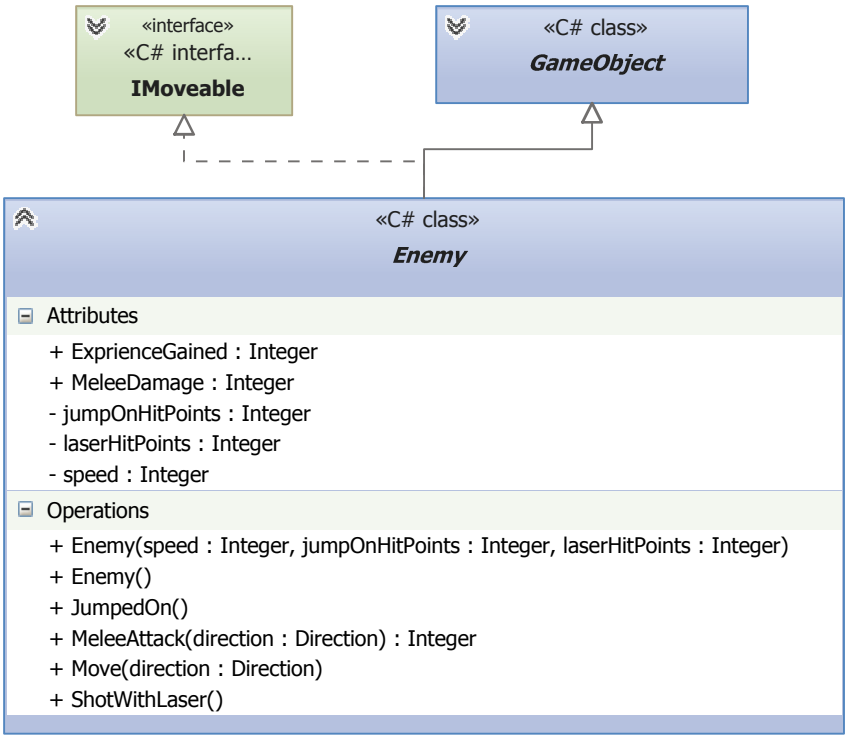
`private int` **`speed`**

Określa szybkość poruszania się postaci (w polach na sekundę).

`private Nanosuit` **`suit`**

Referencja do obiektu skafandra.

3.19 Klasa Enemy



Abstrakcyjna klasa bazowa dla wszystkich przeciwników.

Pola (Właściwości):

`public int ExprienceGained`

Liczba punktów doświadczenia za pokonanie przeciwnika.

`private int jumpOnHits`

Ilość skoków, które trzeba wykonać na przeciwnika by zginął.

`private int laserHitPoints`

Ilość strzałów z laser, które trzeba wykonać by przeciwnik zginął.

Metody:

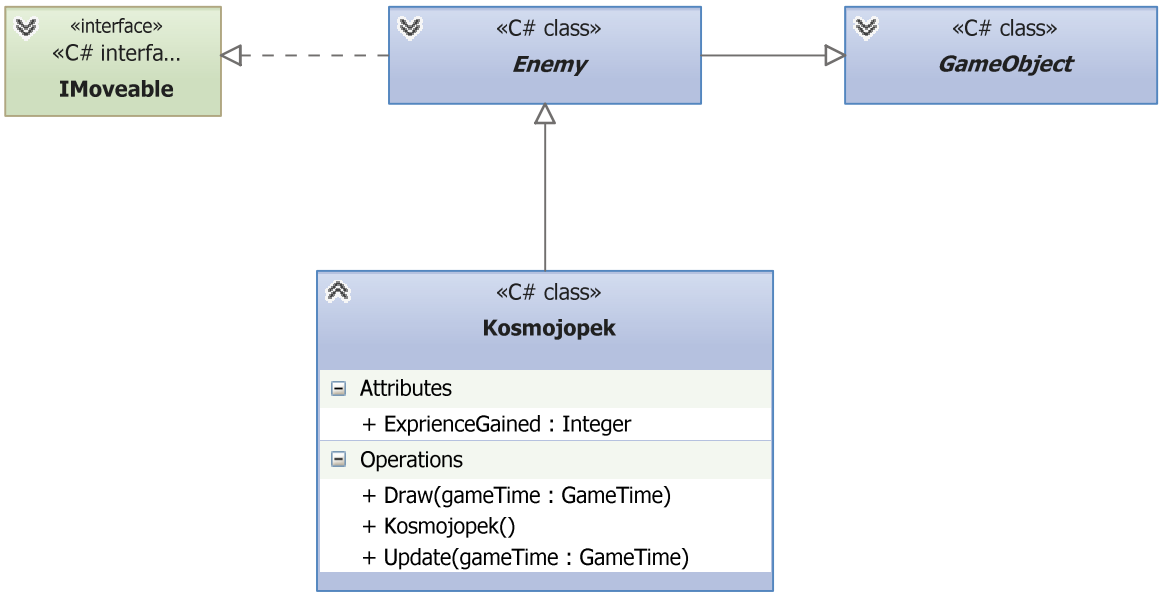
`public void JumpedOn()`

Metoda wywoływana, gdy postać Minera naskoczy na przeciwnika.

`public void ShotWithLaser()`

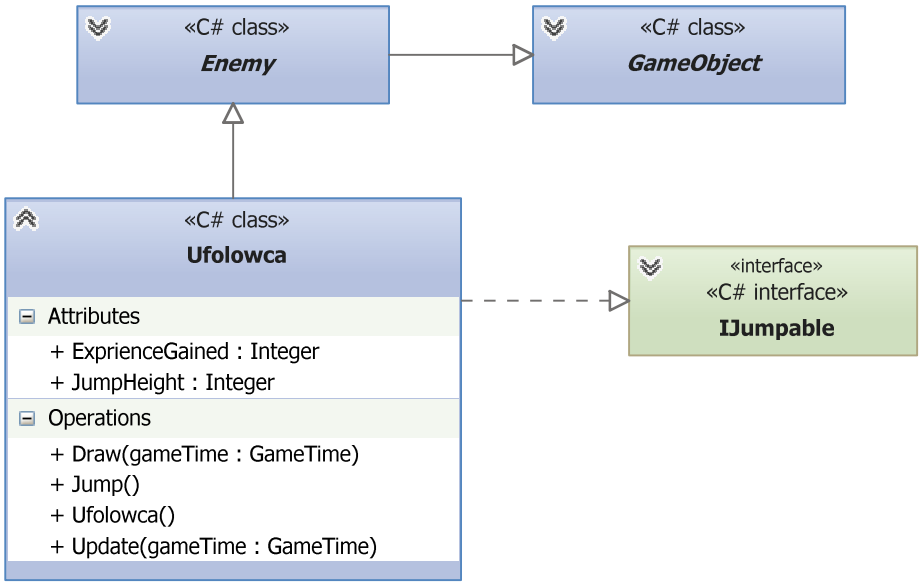
Metoda wywoływana, gdy postać Minera odda strzał z lasera w przeciwnika.

3.20 Klasa Kosmojopek



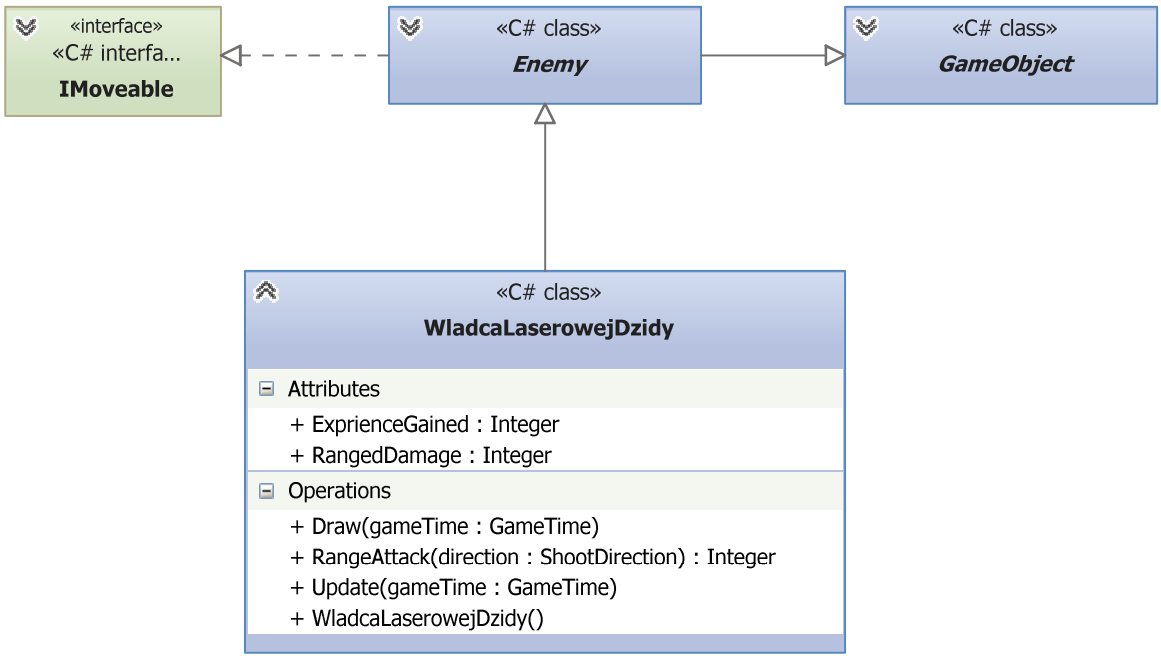
Klasa odpowiadająca za przeciwnika typu Kosmojopek.

3.21 Klasa Ufolowca



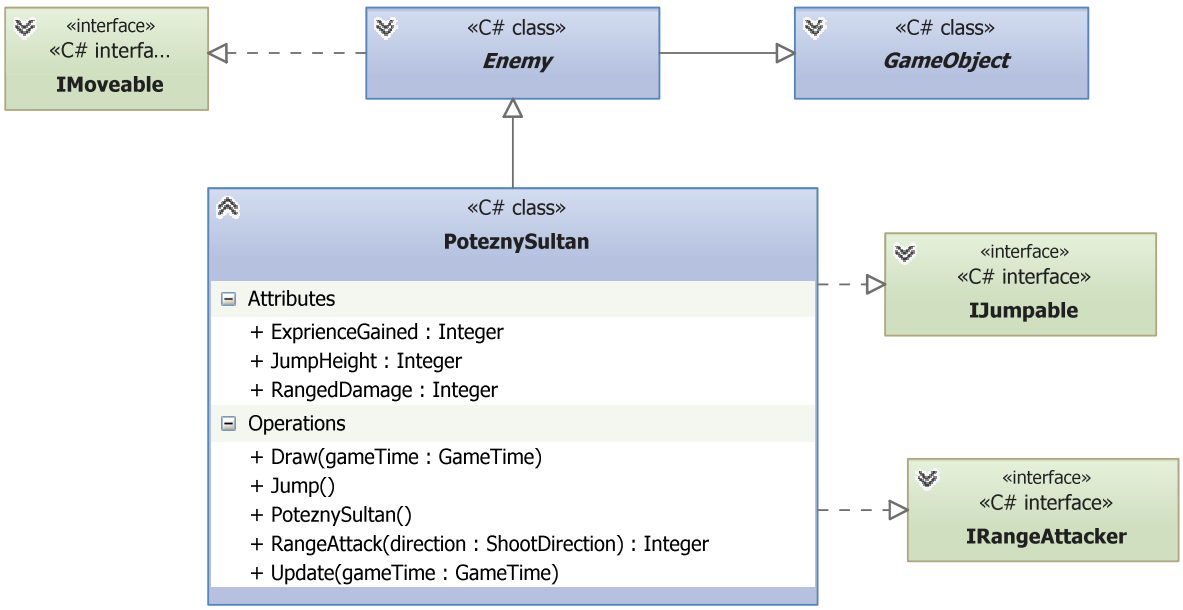
Klasa odpowiadająca za przeciwnika typu Ufołowca.

3.22 Klasa WladcaLaserowejDzidzy



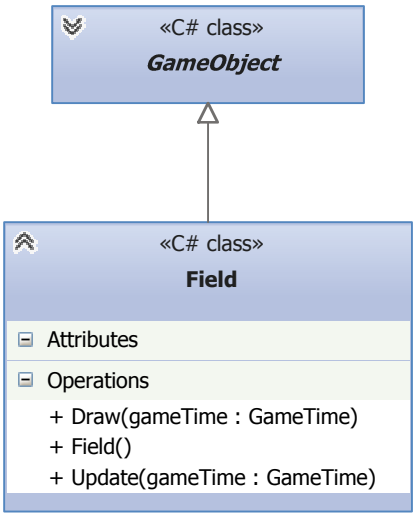
Klasa odpowiadająca za przeciwnika Władca Laserowej Dzidy.

3.23 Klasa PoteznySultan



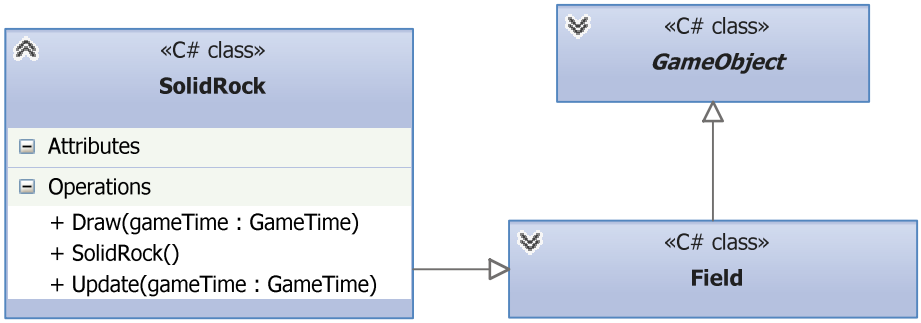
Klasa odpowiadająca za przeciwnika Potężny Sułtan Kosmitów

3.24 Klasa Field



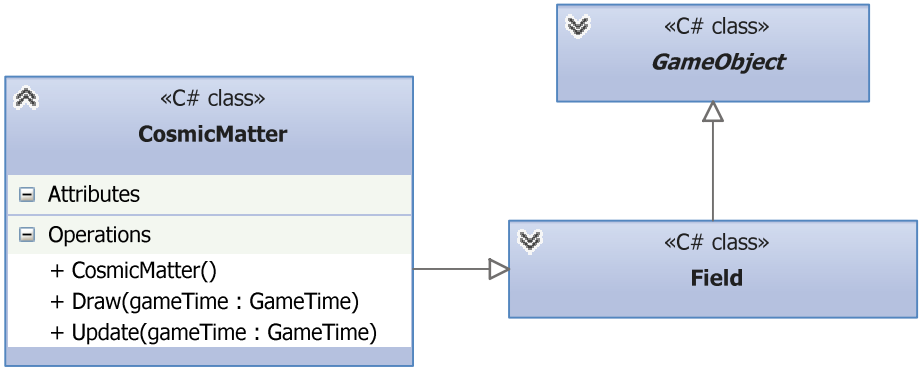
Klasa reprezentująca pole na mapie.

3.25 Klasa SolidRock



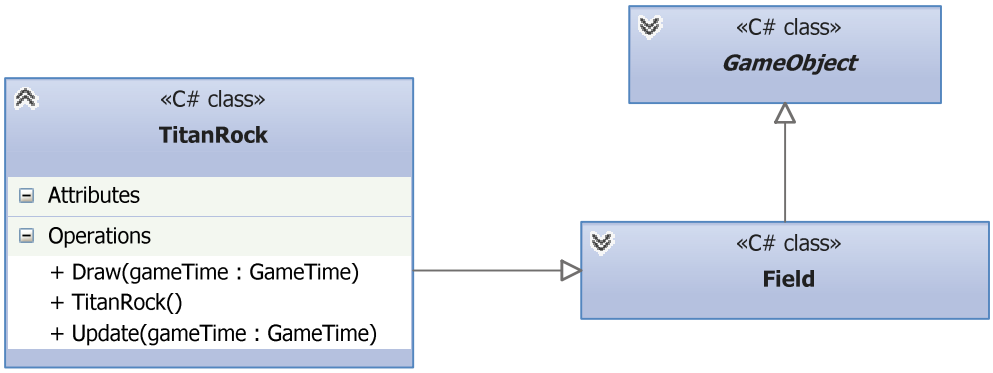
Klasa reprezentująca pole typu Lita skała.

3.26 Klasa CosmicMatter



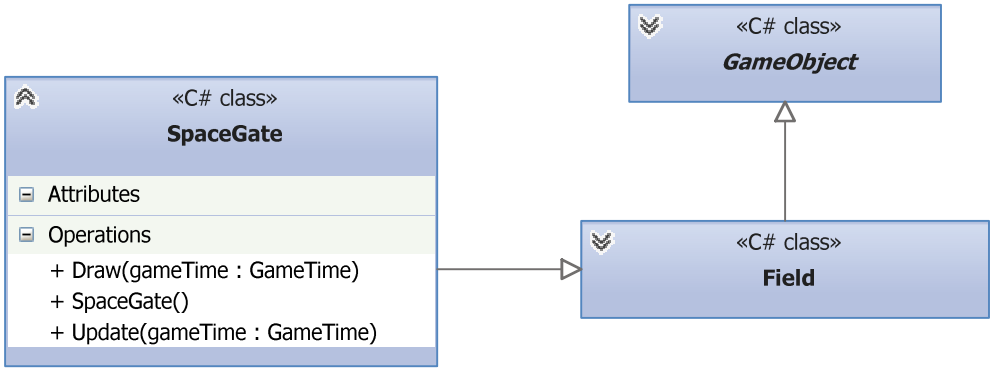
Klasa reprezentująca pole typu Kosmiczny budulec.

3.27 Klasa TitanRock



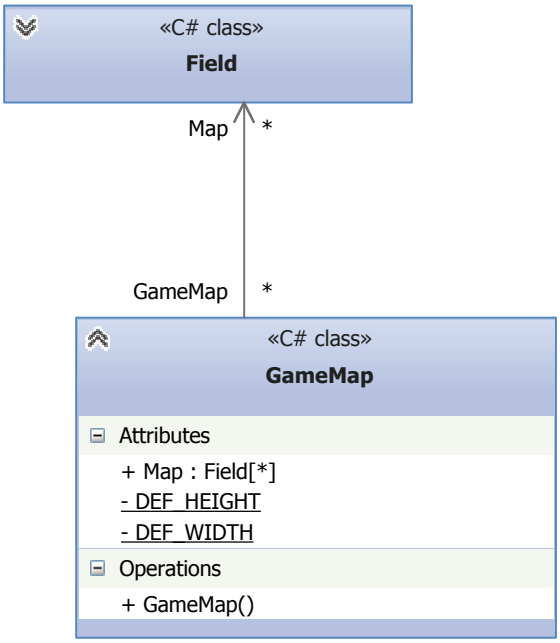
Klasa reprezentująca nieprzekopywalne pole typu Tytanowa skała.

3.28 Klasa SpaceGate



Klasa reprezentująca Kosmiczne Wrota.

3.29 Klasa GameMap



Klasa reprezentująca mapę gry.

Pola (Właściwości):

```
private int DEF_HEIGHT
```

Stała określająca wysokość mapy w polach.

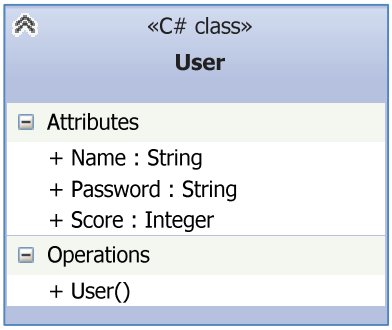
```
private int DEF_WIDTH
```

Stała określająca szerokość mapy w polach.

```
public Field[,] Map
```

Tablica wielkości DEF_WIDTH x DEF_HEIGHT stanowiąca rzeczywistą mapę gry.

3.30 Klasa User



Klasa reprezentująca użytkownika.

Pola (Właściwości):

```
public string Name
```

Imię gracza

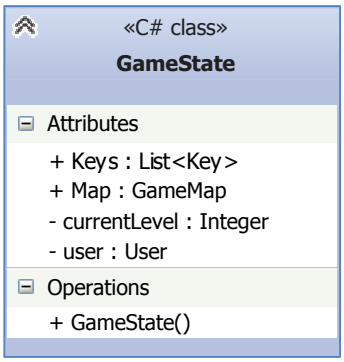
```
public string Password
```

Hasło gracza

```
public int Score
```

Aktualny wynik gracza.

3.31 Klasa GameState



Klasa odpowiedzialna za przetrzymywanie informacji o stanie gry. Plik xml otrzymany z jej serializacji jest jednocześnie stanem gry.

Pola (Właściwości):

`public List<Key> Keys`

Lista zebranych kluczy.

`public GameMap Map`

Mapa gry. Przechowuje również informacje o obiektach znajdujących się na polach.

`private int currentLevel`

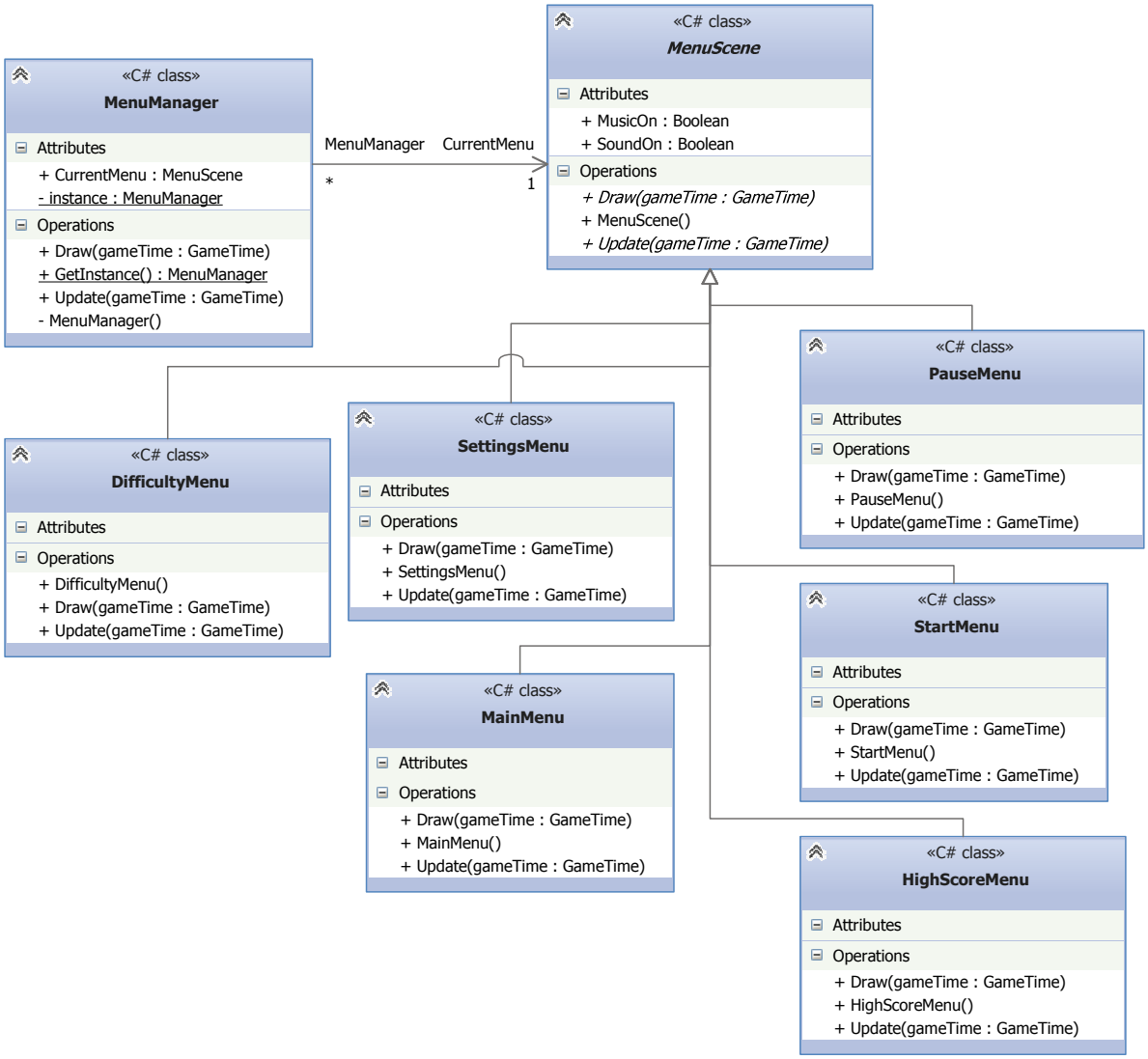
Aktualny poziom.

`private User user`

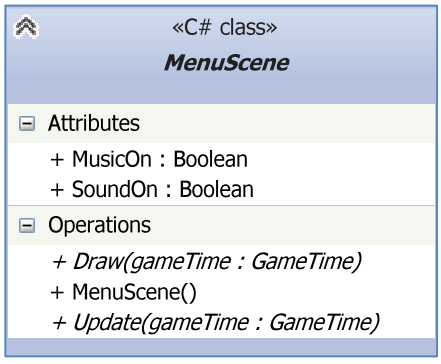
Aktualnie grający użytkownik.

4. Menu

Poniżej przedstawiony jest diagram klas UML dla klas związanych z renderowaniem scen Menu.



4.1 Klasa MenuScene



Klasa bazowa dla wszystkich ekranów (w tym głównego ekranu gry). Zwiera wspólne opcje wszystkich menu (opcje związane z dźwiękiem).

Pola (Właściwości):

```
private bool MusicOn
```

Pole związane z opcją włączania/wyłączania muzyki.

```
private bool SoundOn
```

Pole związane z opcją włączania/wyłączania efektów dźwiękowych.

Metody:

```
public abstract void Draw(GameTime gameTime)
```

Metoda służąca do renderowania danej sceny (danego menu).

Parametry:

GameTime gameTime	czas jaki upłynął od ostatniego wywołania tej metody.

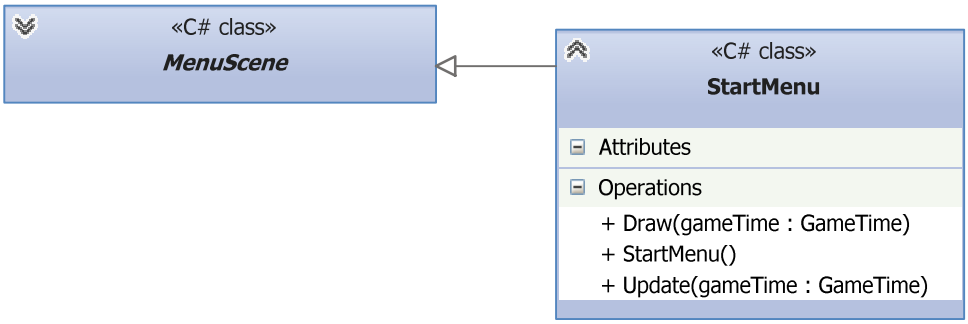
```
public abstract void Update(GameTime gameTime)
```

Metoda odpowiedzialna za uaktualnianie stanu danej sceny (m. in. za obsługę przycisków w menu).

Parametry:

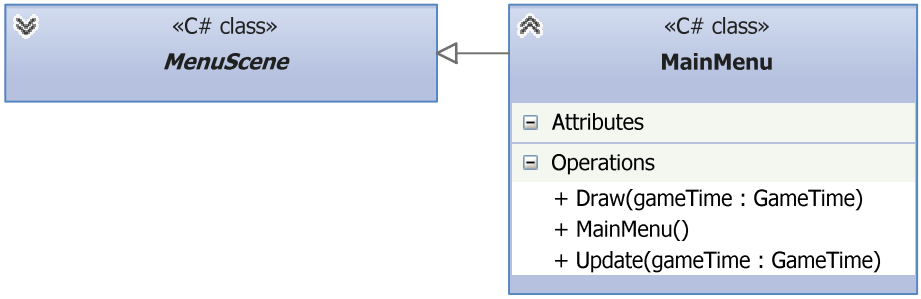
GameTime gameTime	czas jaki upłynął od ostatniego wywołania tej metody.

4.2 Klasa StartMenu



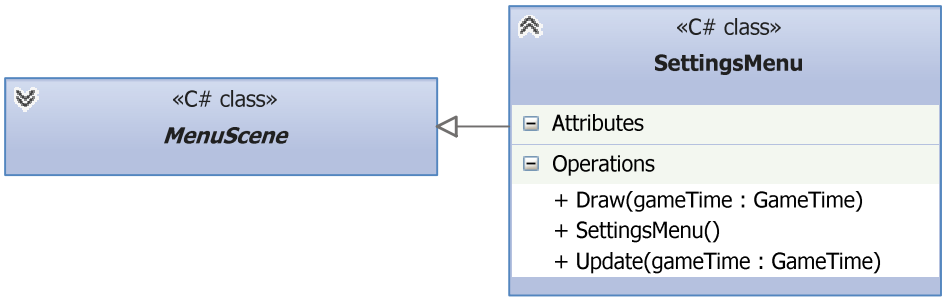
Klasa odpowiedzialna za menu startowe (logowania) (patrz punkt 3.2 dokumentacji wstępnej).

4.3 Klasa MainMenu



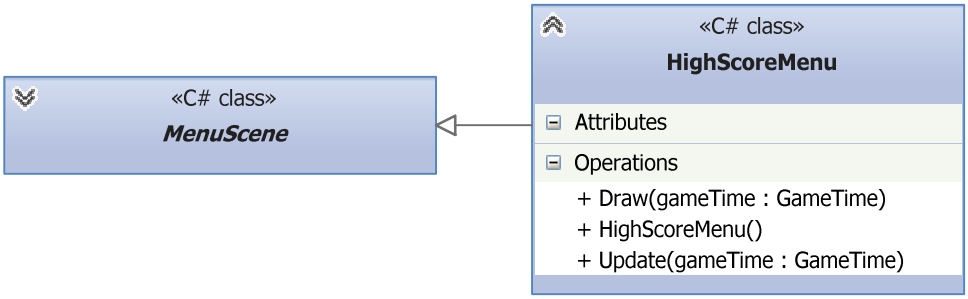
Klasa odpowiedzialna za menu główne (patrz punkt 3.3 dokumentacji wstępnej).

4.4 Klasa SettingsMenu



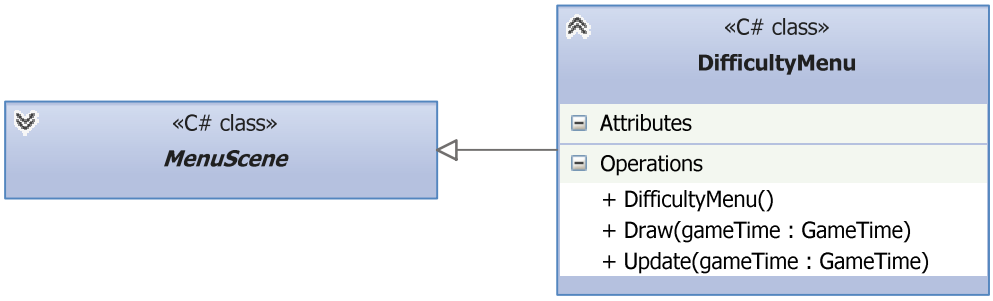
Klasa odpowiedzialna za menu ustawień (patrz punkt 3.4 dokumentacji wstępnej).

4.5 Klasa HighScoreMenu



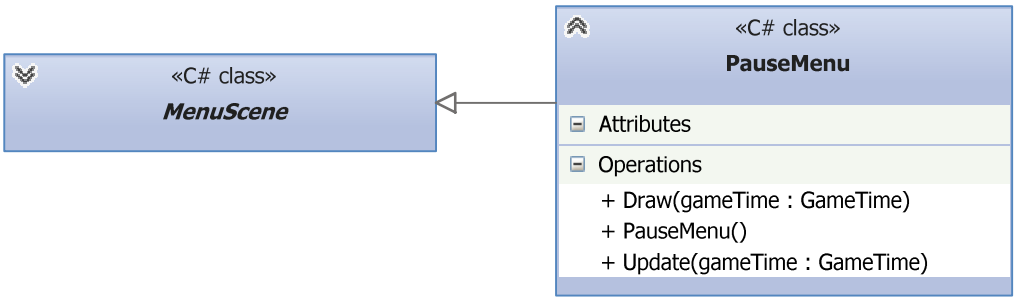
Klasa odpowiedzialna za menu najlepszych wyników (patrz punkt 3.5 dokumentacji wstępnej)

4.6 Klasa DifficultyMenu



Klasa odpowiedzialna za menu wyboru poziomu trudności (patrz punkt 3.6 dokumentacji wstępnej).

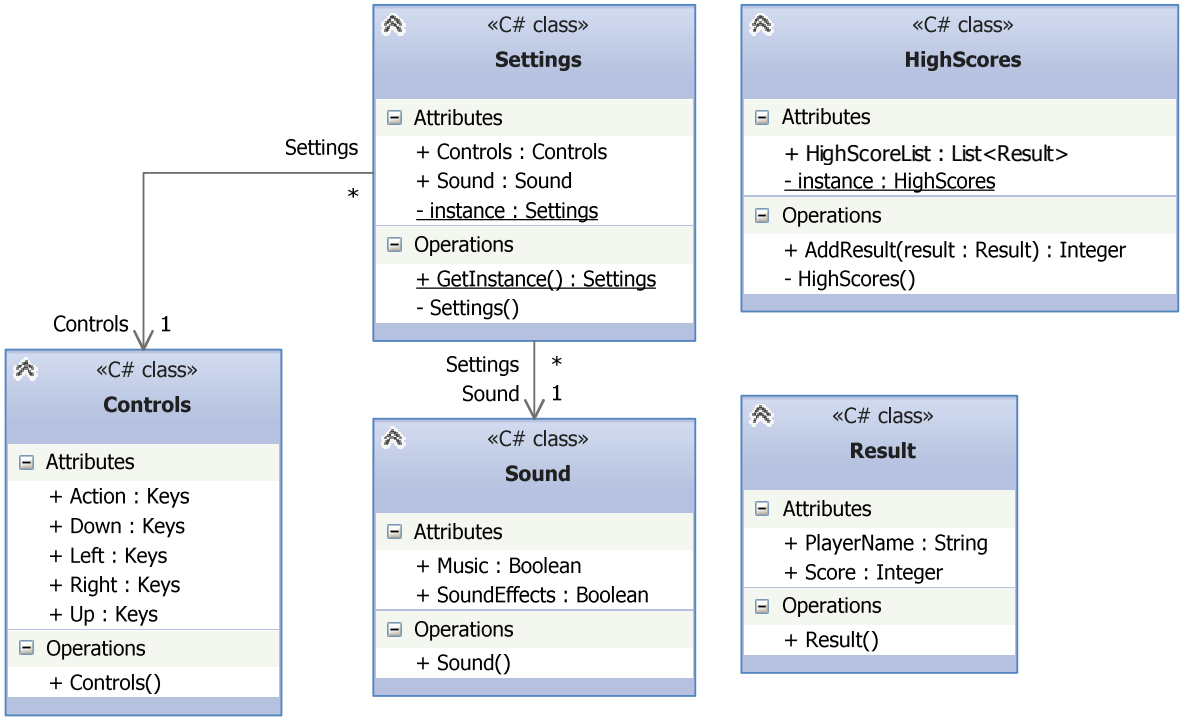
4.7 Klasa PauseMenu



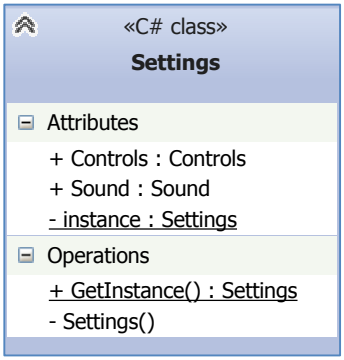
Klasa odpowiedzialna za menu dostępne w czasie pauzy (patrz punkt 3.7 dokumentacji wstępnej).

5. Ustawienia i najlepsze wyniki

Do obsługi ustawień oraz najlepszych wyników służą poniższe klasy.



5.1 Klasa Settings



Klasa przechowująca informacje o ustawieniach gry. Jest to singleton.

Pola (Właściwości):

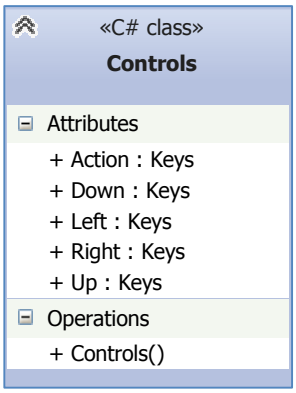
`public Controls` **`Controls`**

Pole przechowujące informacje o ustawieniach związanych ze sterowaniem.

`public Sound` **`Sound`**

Pole przechowujące informacje o ustawieniach związanych z dźwiękiem.

5.2 Klasa Controls



Klasa przechowująca informacje o ustawieniach związanych ze sterowaniem

`public Keys` **`Action`**

Kod przycisku akcji (domyślnie: spacja).

`public Keys` **`Down`**

Kod przycisku odpowiadającego za kopanie (domyślnie: S).

`public Keys` **`Left`**

Kod przycisku odpowiadającego za ruch w lewo (domyślnie: A).

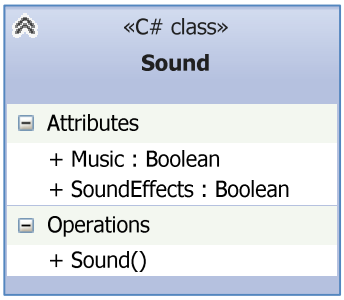
`public Keys` **`Right`**

Kod przycisku odpowiadającego za ruch w prawo (domyślnie: D).

`public Keys` **`Up`**

Kod przycisku odpowiadającego za skok (domyślnie: W).

5.3 Klasa Sound



Klasa przechowująca informacje związane z dźwiękiem.

Pola (Właściwości):

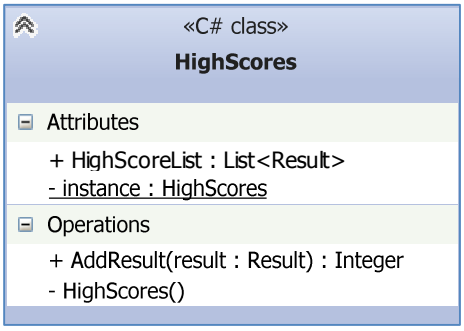
`public bool` **`Music`**

Pole przechowujące informacje o wł./wył. muzyki.

```
public bool SoundEffects
```

Pole przechowujące informacje o wł./wył. efektów dźwiękowych.

5.4 Klasa HighScores



Klasa przechowująca informacje o najlepszych wynikach. Jest to singleton.

Pola (Właściwości):

```
public List<Result> HighScoreList
```

Lista przechowująca najlepsze wyniki. Implementacja klasy zapewnia, że są one posortowane malejąco, oraz jest ich najwyżej 10.

Metody:

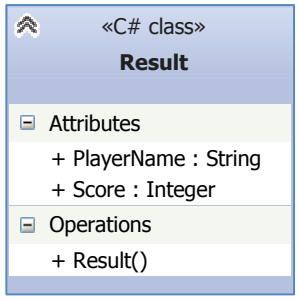
```
public int AddResult(Result result)
```

Metoda dodająca wynik do listy najlepszych wyników. Jeśli podany jej jako argument wynik jest za mały, to ta metoda dba o to, aby nie został dodany. Nowy rekord wstawiany jest zawsze tak, aby wyniki były posortowane malejąco.

Parametry:

result	wstawiany wynik.

5.5 Klasa Result



Klasa przechowująca pojedynczy wpis wyniku. Zawiera informacje o wyniku oraz jego autorze.

Pola (Właściwości):

```
public string PlayerName
```

Imię gracza, który osiągnął zapisany w obiekcie tej klasy wynik.

`public int Score`

Wartość wyniku.

6. Algorytmy użyte w grze.

6.1 Algorytm losowego generowania elementów biernych.

Algorytm ten służy do wylosowania elementów biernych w sposób przedstawiony w dokumentacji wstępnej jako "inteligentny", tj. w taki, że losowane są ich zbiory a nie pojedyncze elementy.

Algorytm zakłada istnienie statycznej kolekcji wzorców, z której losuje się wzorzec i sprawdza możliwość jego dopasowania:

```
miejsce_na_mapie = (0,0)

while(miejsce_na_mapie nie wyszło poza mapę)
        wzorzec = C.getRandom();
        if( wzorzec mieści się w aktualnym miejscu )
                wstaw_wzorzec(miejsce_na_mapie, mapa)
        else
                przytnij_wzorzec(wzorzec, miejsce_na_mapie, mapa) // Przycina wzorzec tak, aby
                                                                  // zmieścił się na mapie
                wstaw_wzorzec(miejsce_na_mapie, mapa)

        miejsce_na_mapie.X += wzorzec.dlugosc * 2;
        miejsce_na_mapie.Y += wzorzec.wysokosc * 2;
}
```

7. Struktura plików

Gra korzysta z plików w formacie XML do przechowywania informacji o ustawieniach gry oraz o wynikach a także o stanie gry.

7.1 Struktura pliku HighScores.xml (informacja o najlepszych wynikach).

Zapisywane jest 10 najlepszych wyników. Struktura pojedynczego wyniku przedstawia się następująco:

```
<Result>
        <PlayerName>
```

```
            Jack
        </PlayerName>
        <Score>
            1000
        </Score>
</Result>
```

Powyższy XML otrzymuje się poprzez serializację obiektu klasy Result do XML.

Plik HighScores.xml zawiera listę maksymalnie 10 takich obiektów, np.:

```
<HighScores>
    <HighScoreList>
        <Result>
            <PlayerName>
                Jack
            </PlayerName>
            <Score>
                1000
            </Score>
        </Result>
        <Result>
            <PlayerName>
                Jackie
            </PlayerName>
            <Score>
                900
            </Score>
        </Result>
        <Result>
            <PlayerName>
                Jacob
            </PlayerName>
            <Score>
                500
            </Score>
        </Result>
    </HighScoreList>
</HighScores>
```

(Powyższy plik przedstawia sytuację, gdy mamy aktualnie 3 najlepsze wyniki).

7.2 Struktura pliku Settings.xml z ustawieniami gry.

Plik Settings.xml przechowuje ustawienia gry (np. sterowanie). Jego struktura jest następująca:

```
<Settings>
    <Controls>
        <Up>
            1
        </Up>
        <Down>
            2
        </Down>
```

```
            <Left>
                3
            </Left>
            <Right>
                4
            </Right>
            <Action>
                5
            </Action>
        </Controls>
        <Sound>
            <SoundEffects>
                true
            </SoundEffects>
            <Music>
                false
            </Music>
        </Sound>
</Settings>
```

W przypadku sterowania przechowywana wartość jest kodem klawisza według XNA.

7.3 Struktura pliku zapisanej gry.

Struktura tego pliku nie jest tu przedstawiana. Wynika ona ze sposobu serializacji obiektu klasy GameState.