|  |
| --- |
|  |
| Miner |
| Dokumentacja Techniczna |
|  |
| **Paweł Kobojek** |
| **2 kwietnia 2014** |

|  |
| --- |
|  |

Zawartość

[1. Wprowadzenie 5](#_Toc384159060)

[2. Technologia 5](#_Toc384159061)

[3. Diagramy klas UML 5](#_Toc384159062)

[3.1 Klasa MinerGame 6](#_Toc384159063)

[Pola (Właściwości): 7](#_Toc384159064)

[Konstruktory: 7](#_Toc384159065)

[Metody: 7](#_Toc384159066)

[3.2 Klasa GameObject 8](#_Toc384159067)

[Pola: 8](#_Toc384159068)

[Konstruktory: 8](#_Toc384159069)

[Metody: 8](#_Toc384159070)

[3.3 Interfejs IActionUsable 9](#_Toc384159071)

[Metody: 9](#_Toc384159072)

[3.4 Interfejs IDiggable 9](#_Toc384159073)

[Metody: 9](#_Toc384159074)

[3.5 Interfejs IJumpable 10](#_Toc384159075)

[Pola (Właściwości): 10](#_Toc384159076)

[Metody: 10](#_Toc384159077)

[3.6 Interfejs IMeleeAttacker 10](#_Toc384159078)

[Pola (Właściwości): 10](#_Toc384159079)

[Metody: 10](#_Toc384159080)

[3.7 Interfejs IRangeAttacker 11](#_Toc384159081)

[Pola (Właściwości): 11](#_Toc384159082)

[Metody: 11](#_Toc384159083)

[3.8 Interfejs IMoveable 11](#_Toc384159084)

[Metody: 11](#_Toc384159085)

[3.9 Klasa BonusObject 12](#_Toc384159086)

[3.10 Klasa Fuel 12](#_Toc384159087)

[Pola (Właściwości): 12](#_Toc384159088)

[Konstruktory: 12](#_Toc384159089)

[3.11 Klasa Laser 13](#_Toc384159090)

[Pola (Właściwości): 13](#_Toc384159091)

[3.12 Klasa KeyLocalizer 13](#_Toc384159092)

[Metody: 14](#_Toc384159093)

[3.13 Klasa Indestructibility 14](#_Toc384159094)

[Pola (Właściwości): 14](#_Toc384159095)

[3.14 Klasa DoubleJump 14](#_Toc384159096)

[3.15 Klasa MoreEnemies 15](#_Toc384159097)

[3.16 Klasa Key 15](#_Toc384159098)

[3.17 Klasa Nanosuit 15](#_Toc384159099)

[Pola (Właściwości): 15](#_Toc384159100)

[Metody: 16](#_Toc384159101)

[3.18 Klasa Miner 16](#_Toc384159102)

[Pola (Właściwości): 16](#_Toc384159103)

[3.19 Klasa Enemy 17](#_Toc384159104)

[Pola (Właściwości): 17](#_Toc384159105)

[Metody: 17](#_Toc384159106)

[3.20 Klasa Kosmojopek 18](#_Toc384159107)

[3.21 Klasa Ufolowca 18](#_Toc384159108)

[3.22 Klasa WladcaLaserowejDzidzy 19](#_Toc384159109)

[3.23 Klasa PoteznySultan 19](#_Toc384159110)

[3.24 Klasa Field 20](#_Toc384159111)

[3.25 Klasa SolidRock 20](#_Toc384159112)

[3.26 Klasa CosmicMatter 20](#_Toc384159113)

[3.27 Klasa TitanRock 21](#_Toc384159114)

[3.28 Klasa SpaceGate 21](#_Toc384159115)

[3.29 Klasa GameMap 21](#_Toc384159116)

[Pola (Właściwości): 22](#_Toc384159117)

[3.30 Klasa User 22](#_Toc384159118)

[Pola (Właściwości): 22](#_Toc384159119)

[4. Menu 22](#_Toc384159120)

[4.1 Klasa MenuScene 23](#_Toc384159121)

[Pola (Właściwości): 23](#_Toc384159122)

[Metody: 24](#_Toc384159123)

[4.2 Klasa StartMenu 24](#_Toc384159124)

[4.3 Klasa MainMenu 24](#_Toc384159125)

[4.4 Klasa SettingsMenu 25](#_Toc384159126)

[4.5 Klasa HighScoreMenu 25](#_Toc384159127)

[4.6 Klasa DifficultyMenu 25](#_Toc384159128)

[4.7 Klasa PauseMenu 25](#_Toc384159129)

[5. Ustawienia i najlepsze wyniki 26](#_Toc384159130)

[5.1 Klasa Settings 26](#_Toc384159131)

[Pola (Właściwości): 26](#_Toc384159132)

[5.2 Klasa Controls 27](#_Toc384159133)

[5.3 Klasa Sound 27](#_Toc384159134)

[Pola (Właściwości): 27](#_Toc384159135)

[5.4 Klasa HighScores 28](#_Toc384159136)

[Pola (Właściwości): 28](#_Toc384159137)

[Metody: 28](#_Toc384159138)

[5.5 Klasa Result 28](#_Toc384159139)

[Pola (Właściwości): 28](#_Toc384159140)

[6. Algorytmy użyte w grze. 29](#_Toc384159141)

[6.1 Algorytm losowego generowania elementów biernych. 29](#_Toc384159142)

[7. Struktura plików 29](#_Toc384159143)

[7.1 Struktura pliku HighScores.xml (informacja o najlepszych wynikach). 29](#_Toc384159144)

[7.2 Struktura pliku Settings.xml z ustawieniami gry. 30](#_Toc384159145)

# 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.

public Settings **Settings**

Pole przechowujące ustawienia.

### 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)

public float **Width**

Pole mówiące o szerokości obiektu.

public float **Height**

Pole mówiące o wysokości obiektu.

protected MinerGame **game**

Referencja na obiekt gry.

public Vector2 **Position**

Dokładna pozycja obiektu na ekranie.

public Vector2 **Velocity**

Prędkość obiektu.

### Konstruktory:

public **GameObject**()

Konstruktor bezparametrowy.

public **GameObject**(MinerGame game)

Konstruktor ustawiający referencję na obiekt gry.

#### Parametry:

*game* Referencja na obiekt gry.

public **GameObject**(MinerGame game, int posX, int posY)

Konstruktor ustawiający referencję na obiekt gry oraz pozycjonujący obiekt na podanym polu

#### Parametry:

*game* Referencja na obiekt gry.

*posX* Pozycja X na mapie.

*posY* Pozycja Y na mapie.

public **GameObject**(MinerGame game, Vector2 position)

Konstruktor ustawiający referencję na obiekt gry oraz pozycjonujący obiekt w podanym miejscu na ekranie.

#### Parametry:

*game* Referencja na obiekt gry.

*position* Dokładna pozycja na mapie.

### Metody:

public abstract void **Draw**(SpriteBatch spriteBatch)

Metoda odpowiadająca za renderowanie obiektu na scenie.

#### Parametry:

*spriteBatch* SpriteBatch, na którym obiekt będzie rysowany.

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

Metoda odpowiadająca za uaktualnianie stanu związanego z obiektem.

#### Parametry:

*gameTime* Czas jaki upłynął od ostatniego wywołania tej metody.

public bool **CollidesWith**(GameObject obj)

Metoda sprawdzająca, czy obiekt koliduje (w dowolnym kierunku) z podanym.

#### Parametry:

*obj* Obiekt, z którym mają być sprawdzane kolizje.

public bool **CollidesHorizontallyWith**(GameObject obj)

Metoda sprawdzająca, czy obiekt koliduje (w kierunku poziomym) z podanym.

#### Parametry:

*obj* Obiekt, z którym mają być sprawdzane kolizje.

## 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 float **JumpImpulse**

Pole określające wartość pionowo skierowanego impulsu nadanego w momencie skoku.

### 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 BonusItem



Abstrakcyjna klasa bazowa dla wszystkich bonusów występujących w grze.

### Pola:

public static int **spawntime**

Statyczne pole przechowujące czas (w milisekundach), co który pojawia się nowy bonus danego typu (o ile to możliwe).

protected int **LifeTime**

Czas (w milisekundach) występowania bonusu.

public int **RemainingLifeTime**

Pozostały czas na mapie.

protected int **TotalUsageTime**

Całkowity czas działania.

public int **RemainingUsageTime**

Pozostały czas działania w milisekundach.

protected Field **field**

Pole na którym znajduje się bonus.

### Konstruktory:

public **BonusItem**()

Konstruktor bezparametrowy.

public **BonusItem**(MinerGame game, int posX, int posY)

Konstruktor ustawiający referencję na obiekt gry oraz pozycjonujący obiekt na podanym polu

#### Parametry:

*game* Referencja na obiekt gry.

*posX* Pozycja X na mapie.

*posY* Pozycja Y na mapie.

### Metody:

public void **Draw**(SpriteBatch spriteBatch)

Metoda odpowiadająca za renderowanie obiektu na scenie.

#### Parametry:

*spriteBatch* SpriteBatch, na którym obiekt będzie rysowany.

public void **Update**(GameTime gameTime)

Metoda odpowiadająca za uaktualnianie stanu związanego z obiektem.

#### Parametry:

*gameTime* Czas jaki upłynął od ostatniego wywołania tej metody.

public static BonusItem **CreateRandomBonus**(MinerGame game, int x, int y)

Tworzy bonus losowego typu.

#### Parametry:

*game*  Referencja na obiekt gry.

*x*  Pozycja X na mapie, gdzie ma być utworzony bonus.

*y*  Pozycja Y na mapie, gdzie ma być utworzony bonus..

## 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.

public const string **ASSET\_NAME**

Ścieżka do obrazka reprezentującego obiekt na scenie.

### Konstruktory:

public **Fuel**(int amount)

Konstruktor tworzący obiekt wraz z zadaną ilością paliwa.

public **Fuel**(MinerGame game, int amount, int posX, int posY)

Tworzy obiekt i ustawia go na danym miejscu.

#### Parametry:

*game*  Referencja na obiekt gry.

*x*  Pozycja X na mapie, gdzie ma być utworzony bonus.

*y*  Pozycja Y na mapie, gdzie ma być utworzony bonus..

## 3.11 Klasa Laser



Klasa odpowiadająca za bonus Laser.

### Pola (Właściwości):

public const int **LASER\_DAMAGE**

Stała określająca wartość obrażeń zadawanych przez laser.

public const int **ASSET\_NAME**

Ścieżka do obrazka reprezentującego obiekt na scenie.

private const int **COOLDOWN**

Czas w milisekundach, który trzeba odczekać pomiędzy kolejnymi użyciami bonusu.

private int **currentCooldown**

Liczba milisekund, która upłynęła od ostatniego użycia bonusu.

### Metody:

public int **UseAction**()

Implementacja interfejsu IActionUsable. Wywołanie tej metody powoduje wystrzał z lasera.

## 3.12 Klasa KeyLocalizer



Klasa odpowiadająca za bonus Lokalizator Kluczy

### Pola (Właściwości):

public const int **ASSET\_NAME**

Ścieżka do obrazka reprezentującego obiekt na scenie.

private const int **COOLDOWN**

Czas w milisekundach, który trzeba odczekać pomiędzy kolejnymi użyciami bonusu.

private int **currentCooldown**

Liczba milisekund, która upłynęła od ostatniego użycia bonusu.

### 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.

public int **MaxDurability**

Określa maksymalną wytrzymałość skafandra.

public int **MaxFuel**

Maksymalna ilość paliwa.

### 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 **ASSET\_NAME**

Ścieżka do obrazka reprezentującego obiekt na scenie.

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).

public Nanosuit **Suit**

Referencja do obiektu skafandra.

public int **Experience**

Referencja do obiektu skafandra.

public int **ExpToNextLevel**

Referencja do obiektu skafandra.

public Direction **LastMoveDirection**

Kierunek ostatniego ruchu, używany do określenia strzału.

private Vector2 **digStartPosition**

Zmienna przechowująca pozycje postaci w momencie rozpoczęcia kopania.

private readonly Vector2 **gravity**

Zmienna tylko do odczytu stanowiąca wektor przyśpieszenia grawitacji.

private conts int **JUMP\_SPEED**

Stała będąca domyślną wartością impulsu skoku.

private KeyboardState **OldKeyState**

Zmienna przechowująca stan klawiatury po ostatnim wywołaniu metody Update. Używana, żeby zapobiec wielokrotnemu odczytaniu jednego przyciśnięcia.

private float **speed**

Wartość impulsu prędkości dodawanego do składowej X prędkości przy wykonaniu ruchu.

private State **State**

Zmienna określająca stan postaci.

## 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.