9 kwietnia 2014
Kamil Żak, Y2
245564

BOMBBROTHERS

Na podstawie gry
Bomberman

Dokumentacja Techniczna

Spis treści

Spis treści 2
1. Wprowadzenie 3
2. Diagram klas, opis wybranych metod klas 3
2.1. Diagram klas 3
2.2. Przepływ zdarzeń 4
2.3. Elementy pomocnicze 4
2.4. Opis ważniejszych klas 5
2.4.1. GameLoop 5
2.4.2. GObject 5
2.4.3. Player 5
2.4.4. Logic 6
2.4.5. WindowManager 6
2.4.6. Sounds 6
2.4.7. FileManager 7
2.5. Opis wybranych metod 7
2.5.1. Action() - klasa Lighter 7
2.5.2. SaveGame() - klasa FileManager 7
2.5.3. CreateUser() - klasa FileManager 7
2.5.4. OnOffMusic() - klasa Sounds 8
2.5.5. Draw() - klasa GObject 8
2.5.6. Collision(Bonus) - klasa ShyBro 8
3. Algorytmy 8
3.1. Poziom łatwy 8
3.2. Poziom średni 8
3.3. Poziom trudny 9
4. Obsługa plików 10
5. Instrukcja użytkownika 10
5.1. Krótki opis gry 10
5.2. Zasady gry 11
5.3. Poziomy trudności 11
5.4. Sterowanie 12
5.5. Ekran logowania 12
5.6. Menu główne 13
5.7. Ekran rozpoczęcia nowej gry 13

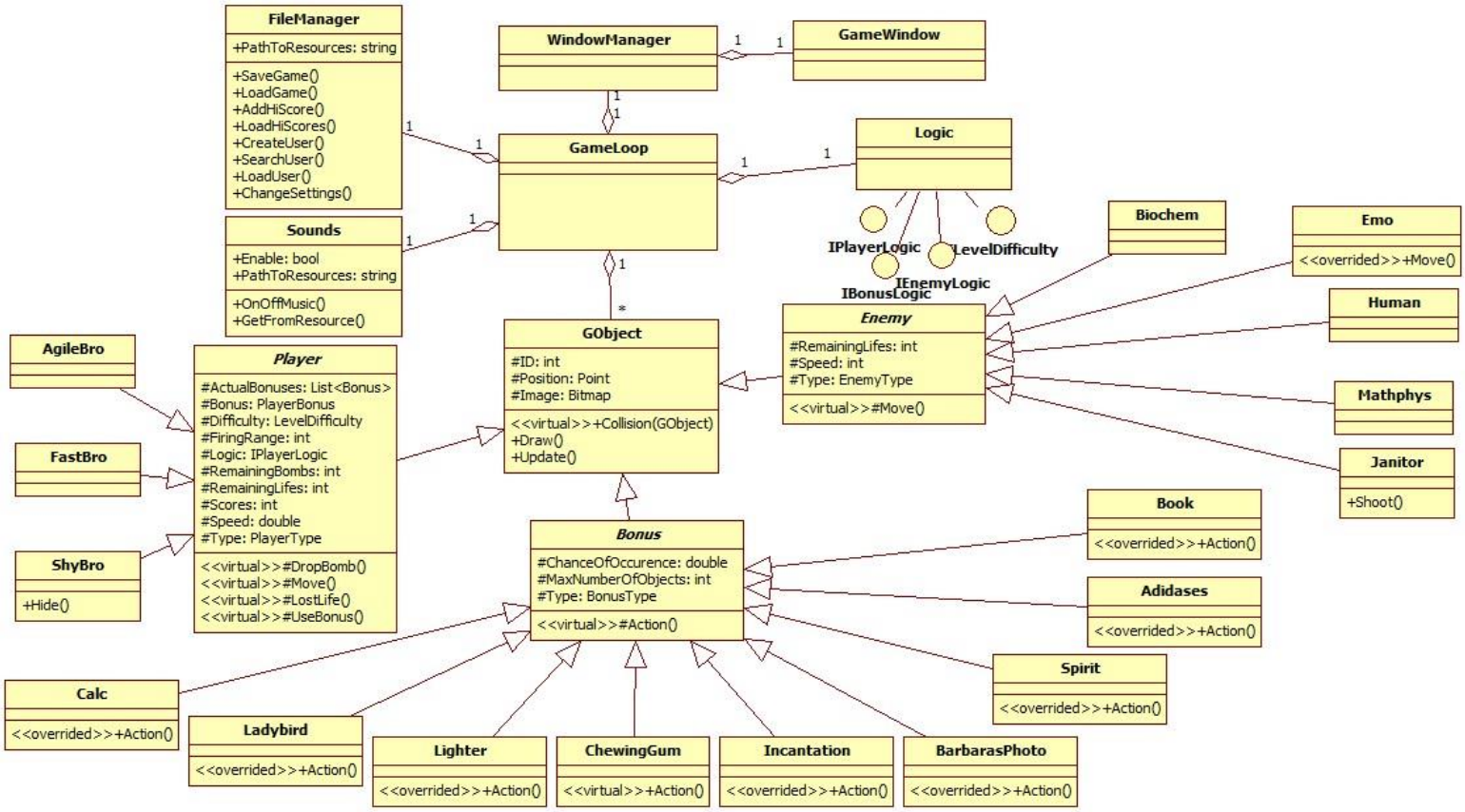
1. Wprowadzenie

Dokumentacja techniczna gry BombBrothers na podstawie dokumentacji wstępnej autorstwa Przemysława Rybaka. Dokument zawiera ogólny diagram klas, elementy pomocnicze (nie ujęte na uprzednio wymienionym diagramie), przepływ zdarzeń działania programu dla konkretnego przypadku, opisy ważniejszych klas oraz opisy ważniejszych metod wraz z pseudokodem danych metod. Zawarte są także opisy kluczowych algorytmów, obsługa plików oraz instrukcja użytkownika.

2. Diagram klas, opis wybranych metod klas

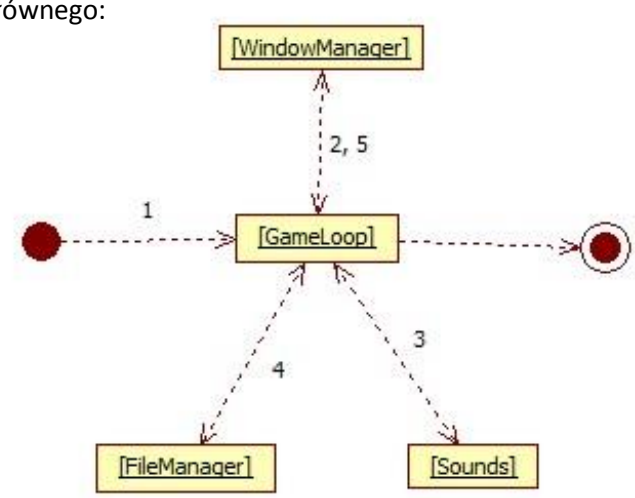
UWAGA ! Na poniższym diagramie nie są ujęte wszystkie klasy, czy struktury. Nie przedstawiona jest m.in. klasa odpowiadająca za opis mapy. Klasa *GameLoop* wykorzystuje obiekty innych klas, sama oprócz bycia pętlą i timer`em nie wykonuje ważniejszych czynności. Metody klasy *Logic* są implementacją interfejsów. Klasa *GameWindow* jest tylko prezentacją jednego z kilku ekranów dostępnych w grze. Nie są także uwzględnione wartości zwracane przez metody, ale w większości przypadków jest to typ *void* lub *bool*.

2.1. Diagram klas



2.2. Przepływ zdarzeń

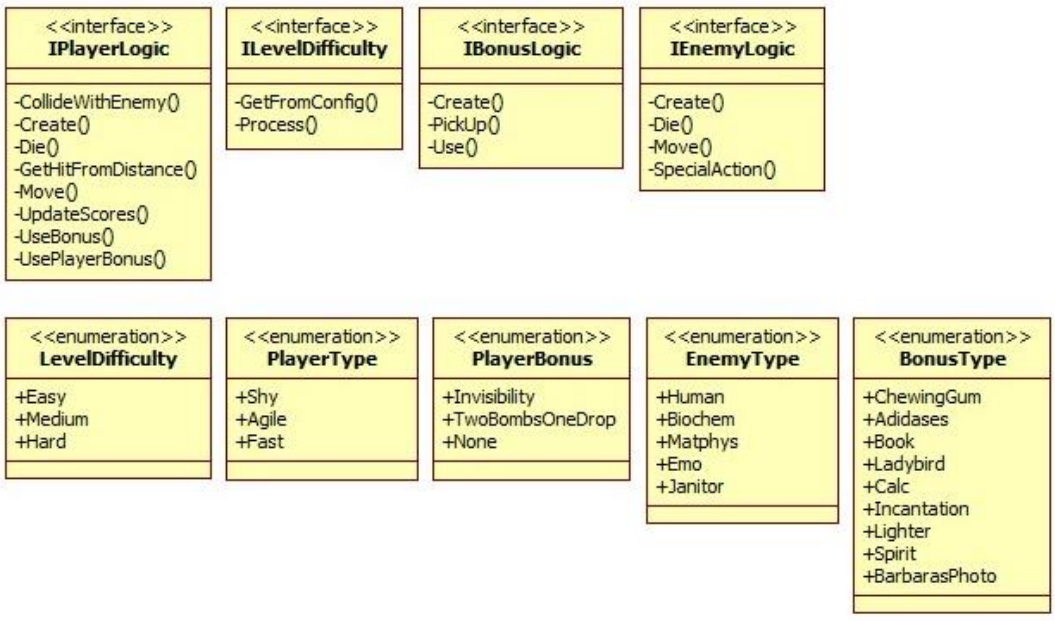
Przykładowy przepływ zdarzeń dla sytuacji od uruchomienia programu, do wczytania menu głównego:



Operacja 1: Uruchomienie programu, tworzony obiekt głównej klasy *GameLoop*
Operacja 2: *GameLoop* tworzy instancje klas *WindowManager*, *FileManager*, *Sounds* (i innych), poprzez *WindowManager* wyświetlany jest ekran logowania
Operacja 3: Razem z wyświetlaniem uruchamiana jest muzyka – tło dla gry (na początku uruchomienia programu zawsze włączona)
Operacja 4: Po wpisaniu przez użytkownika danych do logowania metody z *FileManager* odpowiednio tworzą bądź logują użytkownika
Operacja 5: Poprzez *WindowManager* wyświetlany jest ekran menu głównego dla aktualnie zalogowanego użytkownika.

2.3. Elementy pomocnicze

Elementami pomocniczymi są interfejsy oraz typy wyliczeniowe *enum*:



Ponadto użyte zostaną nie wymienione wcześniej struktury służące do zapisu do plików:

- ***GameState*** – zawiera elementy aktualnie rozgrywanej gry potrzebne do późniejszego jej odtworzenia; takimi elementami są m.in. lista przeciwników wraz z ich umiejscowieniem na mapie, lub liczba punktów aktualnie zdobytych
- ***Score*** – zawiera m.in. punkty zdobyte przez danego użytkownika, nazwę użytkownika i datę dodania wyniku
- ***Inne***, które okażą się pomocne podczas procesu implementacji

2.4. Opis ważniejszych klas

2.4.1. GameLoop

Główna klasa – rdzeń całej gry. Jest pętlą w której każdy obieg to zbiór zdarzeń, które dana klasa obsługuje. Zarządza oknami pojawiającymi się na ekranie, np. po zalogowaniu poprzez *WindowManager* przełącza aktualne okno na okno głównego menu. Zarządza dźwiękiem (pozwala go odtwarzać w tle, bądź nie), tworzy wszystkie obiekty występujące podczas trwania gry (okno właściwe gry), które przechowuje w prywatnej liście obiektów typu *GObject*. Obsługuje wszystkie nadchodzące zdarzenia poprzez wykorzystanie odpowiednich modułów, np. wciśnięcie przycisku po wpisaniu danych do logowania uruchamia zdarzenie logowanie, które jest obsługiwane przez moduł *FileManager*. Obsługuje również zarządzanie obiektami na mapie (poprzez moduł *Logic*): sprawdza czy nastąpiły kolizje, uaktualnia pozycję obiektów na mapie, renderuje je itp.

2.4.2. GObject

Abstrakcyjna klasa bazowa dla wszystkich obiektów fizycznych występujących podczas rozgrywki (gracz, wróg, bonus). Zawiera numer identyfikacyjny danego obiektu, jego pozycję na mapie (parametry x i y kartezjańskiego układu współrzędnych) oraz bitmapę reprezentującą dany obiekt. Metody publiczne *Update*() i *Draw*() uaktualniają status obiektu i rysują go na mapie odpowiednio. Wirtualna metoda publiczna *Collision*() jest nadpisana w odpowiednich miejscach drzewa dziedziczenia, w zależności od rodzaju danej kolizji. Dziedziczą po niej abstrakcyjne klasy: *Player*, *Enemy*, *Bonus*.

2.4.3. Player

Abstrakcyjna klasa dziedzicząca po *GObject*, jest klasą bazową pośrednią (zaraz po *GObject*) dla klas *AgileBro*, *FastBro*, *ShyBro* (reprezentują konkretne typy graczy), które będą tworzone podczas rozgrywki. Wszystkie jej atrybuty oraz metody są chronione (*protected*), ponadto każda metoda jest wirtualna, ponieważ każdy rodzaj bohatera może implementować różne wersje danej metody.

Atrybuty:
(typ nazwa: opis)

- *List<Bonus> ActualBonuses*: przechowuje listę aktualnie zebranych przez gracza bonusów
- *PlayerBonus Bonus*: informacja o specjalnej właściwości, która cechuje konkretną postać; do użycia podczas rozgrywki
- *LevelDifficulty Difficulty*: informacja o poziomie trudności na jakim rozgrywana jest gra, od tego zależy m.in. ilość punktów za odpowiednie zdarzenia, oraz szybkość danej postaci
- *int FiringRange*: wartość (w jednostkach pola) zasięgu rażenia bomby stawianej przez gracza; może być zwiększone przez odpowiedni bonus, lub z każdym kolejnym poziomem w zależności od typu gracza
- *IPlayerLogic Logic*: zbiór metod zawartych w interfejsie *IPlayerLogic*, dotyczących tylko i wyłącznie obiektu *Player*
- *int RemainingsBomb*: ilość bomb pozostałych do użycia przez gracza
- *int RemainingLifes*: ilość żyć pozostałych graczowi w aktualnej rozgrywce; jeśli atrybut osiągnie wartość 0 gra jest kończona ze stanem porażki
- *int Scores*: punkty jakie gracz zebrał w danej rozgrywce
- *double Speed*: prędkość (jednostka/sekunda) z jaką gracz przemieszcza się po planszy
- *PlayerType Type*: rodzaj postaci; od tego atrybutu zależą m.in. specjalne właściwości, czy szybkość gracza

2.4.4. Logic

Główna klasa zajmująca się aspektem logiczno-fizycznym gry. Implementuje interfejsy: *IPlayerLogic*, *IBonusLogic*, *IEnemyLogic*, *ILevelDifficulty*, które zawierają komplet metod do obsługi każdego rodzaju zdarzenia każdego dostępnego obiektu. W zależności od rodzaju obiektu, będzie mu przypisywany obiekt *Logic* typu odpowiedniego interfejsu, co spowoduje, że dla danego obiektu dostępne będą tylko te metody, których faktycznie może używać.

2.4.5. WindowManager

Klasa zarządzająca pojawiającymi się na ekranie oknami. Metody z jej klasy wywoływane z klasy *GameLoop*, w zależności od nadchodzących zdarzeń.

2.4.6. Sounds

Klasa zawiera ścieżkę do pliku dźwiękowego, który ma być tłem aplikacji. Ponadto zawiera atrybut typu *bool*, który jest zmieniany w zależności od tego, czy użytkownik chce słuchać dźwięków, bądź nie. Metody pozwalając na wczytanie muzyki, oraz na odtwarzanie jej, bądź zastopowanie.

2.4.7. FileManager

Klasa zarządzająca obsługą plików wykorzystywanych w grze. Dostępna z poziomy klasy *GameLoop* jako instancja obiektu.

Atrybuty:
(typ nazwa: opis)

- *string PathToResources*: ścieżka do folderu z zasobami plikowymi

Metody:
- *SaveGame*(), *LoadGame*() – zarządzają plikiem zawierającym zapisane stany gry
- *AddHiScore*(), *LoadHiScore*() – zarządzają plikiem zawierającym listę najlepszych wyników
- *CreateUser*(), *SearchUser*(), *LoadUser*() – zarządzają plikiem zawierającym konta (dane) użytkowników
- *ChangeSettings*() – zarządza plikiem ustawień gry (m.in. ustawień klawiszy sterowania)

Więcej szczegółów w sekcji Obsługa plików ►

2.5. Opis wybranych metod

2.5.1. Action() - klasa Lighter

```
public void Action()
{
	Licznik podłożeń bomby z bonusem = 0;
	Dopóki powyższy licznik postawień bomby mniejszy od 3
		Zablokuj stawianie bomby dopiero po podłożeniu drugiej bomby;
}
```

2.5.2. SaveGame() - klasa FileManager

```
public void SaveGame()
{
	Uzupełnij strukturę GameState danymi zebranymi z aktualnie rozgrywanej gry;
	Serializuj strukturę i zapisz do pliku HighScores.xml;
}
```

2.5.3. CreateUser() - klasa FileManager

```
public void CreateUser()
{
	Serializuj i zapisz strukturę zawierającą informacje o użytkowniku do pliku Users.xml;
}
```

2.5.4. OnOffMusic() - klasa Sounds

```
public void OnOffMusic()
{
    Jeśli muzyka jest włączona
        Zmień wartość parametru Enable na false;
    W przeciwnym przypadku
        Zmień wartość parametru Enable na true;
}
```

2.5.5. Draw() - klasa GObject

```
public void Draw()
{
    Przypisz bloczkowi o współrzędnych zawartych w atrybucie Point bitmapę z atrybutu Image;
}
```

2.5.6. Collision(Bonus) - klasa ShyBro

```
public void Collision(Bonus)
{
    Dodaj Bonus do atrybutu (listy bonusów) ActualBonuses aktualnego gracza;
}
```

3. Algorytmy

Wszelkie algorytmy opisane w tym dziale dotyczą ruchu przeciwników. Dzielą się one na zwykłe (statyczne poruszanie), oraz inteligentne (dynamiczne poruszanie). Ich implementacja zależy od poziomu trudności i klasy przeciwnika.

3.1. Poziom łatwy

Na poziomie łatwym tylko *Woźny* używa inteligentnego poruszania się. Przeciwnicy klasy *Human*, *Matfiz* oraz *Biochem* (grupa I) poruszają się w jednym kierunku, następnie po napotkaniu przeszkody (każdej włącznie z ławką) losowo zmieniają kierunek. Przeciwnik klasy *Emo* (grupa II) stoi w miejscu i co pewną stałą czasu obraca się zgodnie z ruchem wskazówek zegara.

3.2. Poziom średni

Również tylko *Woźny* ma inteligentne poruszanie. Przeciwnicy klas grupy I poruszają się albo inteligentnie, albo statycznie ($\left\lceil \frac{1}{3} \right\rceil$ porusza się inteligentnie – wybierane losowo). Dodatkowo przeciwnicy z tej grupy mogą przechodzić przez ławki, jeśli tak będzie prowadziła najkrótsza ścieżka. Przeciwnik z grupy II stoi w miejscu, obraca się co stały czas, ale w losową stronę.

3.3. Poziom trudny

Woźny porusza się inteligentnie, $\left\lceil \frac{2}{3} \right\rceil$ przeciwników z grupy I (wybierane losowo) porusza się inteligentnie (mogą przechodzić przez ławki), a przeciwnik grupy II stoi w miejscu, obraca się co losowy czas, w losową stronę.

Algorytm dla statycznego poruszania się przeciwników **grupy I**:

```
void StaticMoveI()
{
	Wylosuj stronę w którą masz iść (lewo, góra, prawo, dół);
	Dopóki wylosowana strona zawiera następny bloczek z przeszkodą
		Wylosuj nową stronę;
	Idź w podaną stronę, aż do napotkania przeszkody;
}
```

Algorytm poruszania się przeciwnika **grupy II** dla poziomu trudnego:

```
void StaticMoveII()
{
	Dopóki wylosowany kierunek zawiera następny bloczek z przeszkodą
		Wylosuj kierunek w którego stronę nastąpi obrót (lewo, góra, prawo, dół);
	Wylosuj czas za jaki nastąpi zmiana pozycji;
	Odczekaj wylosowany czas, a następnie obróć się w wylosowanym kierunku;
}
```

Algorytm inteligentny będzie opierał się na algorytmie *A** przeszukiwania grafu. Na potrzeby tego algorytmu, mapa jest zamieniana w następujący sposób:

- wierzchołkiem może być każde pole macierzy planszy, jeśli tylko nie jest przeszkodą trwałą
- krawędzie istnieją tylko w kierunkach poziomych i pionowych i tylko wtedy kiedy dane dwa pola sąsiadujące ze sobą nie zawierają przeszkód trwałych
- wierzchołkiem startowym w algorytmie jest pozycja przeciwnika, a końcowym aktualna pozycja gracza

```
void AIMove()
{
	Znajdź najkrótszą ścieżkę do gracza używając algorytmu A* na grafie stworzonym z mapy;
	Idź według wyznaczonej ścieżki;
}
```

Oczywiście pozycja gracza zmienia się w czasie, więc metoda ta musi być wywoływana co ruch przeciwnika o jeden bloczek.

4. Obsługa plików

Całą obsługą plików zajmuje się klasa *FileManager*. Plikami dostępnymi w programie są pliki o rozszerzeniu XML. Odczyt i zapis działają dzięki serializacji i deserializacji plików XML. Dostępne pliki:

- SavedGames.xml
- HighScores.xml
- Settings.xml
- Users.xml

SavedGames.xml zawiera:

- cały obiekt klasy *GameState*
- nazwę użytkownika, dla którego jest zapisana gra
- datę zapisania gry
- poziom trudności, na jakim była rozgrywana gra
- ID zapisu gry (wyliczane kolejnością dodawania do pliku)

HighScores.xml zawiera:

- cały obiekt struktury *Score*
- nazwę użytkownika, który zdobył daną ilość punktów
- datę dodania wpisu z punktami
- poziom trudności, na jakim była rozgrywana gra
- ID wyniku (wyliczane kolejnością dodawania do pliku)

Settings.xml zawiera:

- ścieżkę do pliku dźwiękowego odtwarzanego podczas działania programu
- wartość znakową (*char*) dla 7 rodzajów akcji (idź do góry, do dołu, w lewo, w prawo, podłóż bombę, użyj bonusu postaci, użyj bonusu gry)
- może zawierać ścieżki do bitmap obiektów pojawiających się w grze (w podstawowej wersji, te parametry będą ustawione na stałe – atrybut dla przyszłych rozszerzeń)

Users.xml zawiera:

- nazwę użytkownika
- datę dodania użytkownika
- datę ostatniego logowania
- ID użytkownika (wyliczane kolejnością dodawania do pliku)

Grafika i inne dźwięki mogą być ładowane bezpośrednio z resource'ów.

5. Instrukcja użytkownika

5.1. Krótki opis gry

BombBrothers jest zręcznościową grą 2D o widoku „z lotu ptaka". Gracz wciela się w jedną z trzech postaci-braci: Szybkiego, Nieśmiałego lub Zręcznego, a jego misją jest pokonanie groźnego Woźnego, który przejął władzę w Szkole, oraz uwolnienie pięknej Barbary z rąk złoczyńcy. Poza głównym przeciwnikiem gracz musi pokonać hordy podwładnych Woźnego, które są rozlokowane na jednym z

pięciu poziomów. To na ostatnim po pokonaniu podwładnych pojawia się sam Boss. Naszą bronią są ręcznie zrobione bomby o wysokiej skuteczności, jednak podczas rozgrywki wspomogą nas pojawiające się przeróżne bonusy, które mogą ułatwić nam pokonanie przeciwników.

5.2. Zasady gry

Celem gry jest jak najszybsze przejście wszystkich pięciu poziomów oraz pokonanie Woźnego. Jeśli się to uda – wygrywamy. Porażka może nastąpić w kilku przypadkach:

- zabraknie nam w arsenale bomb a na danym poziomie będą jeszcze nie unicestwieni wrogowie
- po przekroczeniu czasu przewidzianego na dany etap
- w sytuacji gdy stracimy wszystkie życia naszej postaci

Utrata życia następuje w wyniku kolizji z przeciwnikiem atakującym z bliska, lub z przeciwnikiem atakującym na odległość (w tym przypadku mowa o kontakcie wzrokowym z przeciwnikiem klasy Emo w zasięgu kilku pól)

Wszelkie informacje o odległości tyczą się metryki taksówkowej, gdyż mapa jest zbudowana na planie macierzy dwuwymiarowej i dostępne są tylko przejścia poziome i pionowe (nie ma ukośnych).

5.3. Poziomy trudności

W grze dostępne są 3 poziomy trudności gry:

- Łatwy
- Średni
- Trudny

Poziomy trudności różnią się ilością żyć na początku każdego etapu, ilością czasu przeznaczoną na etap, oraz szybkością przeciwników. Ilość żyć i czasu są opisane w Dokumentacji Wstępnej. Ponadto na każdym poziomie inaczej są przyznawane punkty za konkretnego przeciwnika, czy zebranie bonusu.

Punkty za pokonanych przeciwników:

Rodzaj przeciwnika	Poziom trudności	Punkty
Matfiz, Human, Biochem	Łatwy	1 * X
	Średni	2 * X
	Trudny	3 * X
Emo	Łatwy	2 * X
	Średni	4 * X
	Trudny	8 * X
Woźny	Łatwy	15 * X
	Średni	30 * X
	Trudny	45 * X

Punkty za zebrane bonusy:

Rodzaj bonusu*	Poziom trudności	Punkty
Guma do żucia, Książka, Biedronka, Kalkulator, Inkantacja, Barbara	Łatwy	1 * X
	Średni	2 * X
	Trudny	3 * X
Spirytus, Zapalniczka	Łatwy	2 * X
	Średni	4 * X
	Trudny	6 * X
Adidasy	Łatwy	3 * X
	Średni	6 * X
	Trudny	9 * X

* Podział wyznaczany na podstawie procentowej szansy wystąpienia podczas rozgrywki

Gdzie X jest podstawowym współczynnikiem punktowania na poziomie łatwym. Współczynnik ten będzie dobrany podczas implementacji gry, tak aby uatrakcyjnić rozgrywkę.

5.4. Sterowanie

Sterowanie opiera się na 7 klawiszach podstawowych, których wartości można zmienić za pomocą opcji Menu Głównego: Sterowanie oraz na 3 klawiszach, których wartości nie da się zmienić.

Domyślne wartości:

AKCJA	PRZYCISK
Idź do góry	UP
Idź do dołu	DOWN
Idź w lewo	LEFT
Idź w prawo	RIGHT
Podłóż bombę	SPACJA
Użyj bonusu postaci	CTRL
Użyj bonusu zebranego podczas rozgrywki	SPACE
Pauza gry	P
Zapisanie gry	S
Wyjście do menu głównego gry	ESC

Wznowienie gry po przejściu w tryb pauzy „P” następuje za pomocą dowolnego klawisza. Klawisze „P”, „S”, „ESC” są traktowane jako stałe, nie wolno ich zmieniać. Także klawisze zmienne nie mogą mieć wartości przypisanych klawiszom stałym.

5.5. Ekran logowania

Należy podać login (oraz hasło – w wersji rozszerzonej) użytkownika na jaki chcemy się zalogować, lub jaki chcemy stworzyć (jeśli nie ma danego użytkownika w bazie). Jeśli podamy login, który istnieje w bazie, pojawi się notyfikacja o istnieniu danego loginu i opcjach do wyboru:
- Zaloguj na aktualnym koncie
- Stwórz nowe konto

Ma to na celu pozwolenie nowemu użytkownikowi wyboru czy chce grać na czyimś koncie, czy woli stworzyć swoje własne (w przypadku, gdy akurat podany login będzie zajęty).

W wersji z hasłem problem ten się nie pojawia i logowanie nastąpi tylko po podaniu odpowiedniego loginu i hasła. Gdy login się powtórzy a hasło będzie złe – pojawi się informacja o złym haśle. Jeśli login będzie unikalny, zostanie stworzone nowe konto o podanym loginie i haśle.

5.6. Menu główne

- **Kontynuuj** – opcja uruchomienia ostatnio zapisanej gry danego użytkownika (jeśli taka była)
- **Nowa gra** – uruchamia ekran nowej gry, z którego (po wybraniu odpowiednich opcji) można przejść do głównego ekranu gry
- **Najlepsze wyniki** – wyświetlana jest lista z najlepszymi wynikami (10 najlepszych wyników)
- **Sterowanie** – uruchamiany jest ekran zmiany ustawień sterowania gry; po zapisaniu ustawień są wprowadzane odpowiednie zmiany w pliku Settings.xml (w wersji rozszerzonej nazwa pozycji menu, zmieniona jest na „Ustawienia” i pozwala na inne modyfikacje programu np. zmiana ścieżki do pliku dźwiękowego odtwarzanego podczas gry)
- **Wyjście** – kończy pracę programu

5.7. Ekran rozpoczęcia nowej gry

Należy wybrać rodzaj postaci jaką chcemy grać, oraz poziom trudności naszej rozgrywki.

5.8. Lista zmian po zaimplementowaniu gry

- Klasa GameLoop została odrzucona, jako zbędna.

  Powód: pętla gry jest domyślnie stworzona i używana w WPF.

- Zrezygnowano z klasy Logic i interfejsów, które ona implementuje.

  Powód: dane metody wykorzystywały informacje z konkretnych klas, a więc żeby nie zmniejszać jakości kodu konkretne metody zawierające logikę zostały przeniesione do konkretnych klas używających je.

- Zmiana rozszerzenia plików z .xml na .dat.

  Powód: stworzony same scheme i na ich podstawie wygenerowano modele. Pliki .dat jak sugeruje nazwa zawierają dane (w tym przypadku w postaci tekstów xml).

Niektóre z powyższych klas, zależności czy funkcjonalności nie zostały wykorzystane (zaimplementowane) w aktualnej wersji programu. Jednak ich implementacja i użycie są zgodne z niniejszą dokumentacją techniczną.