

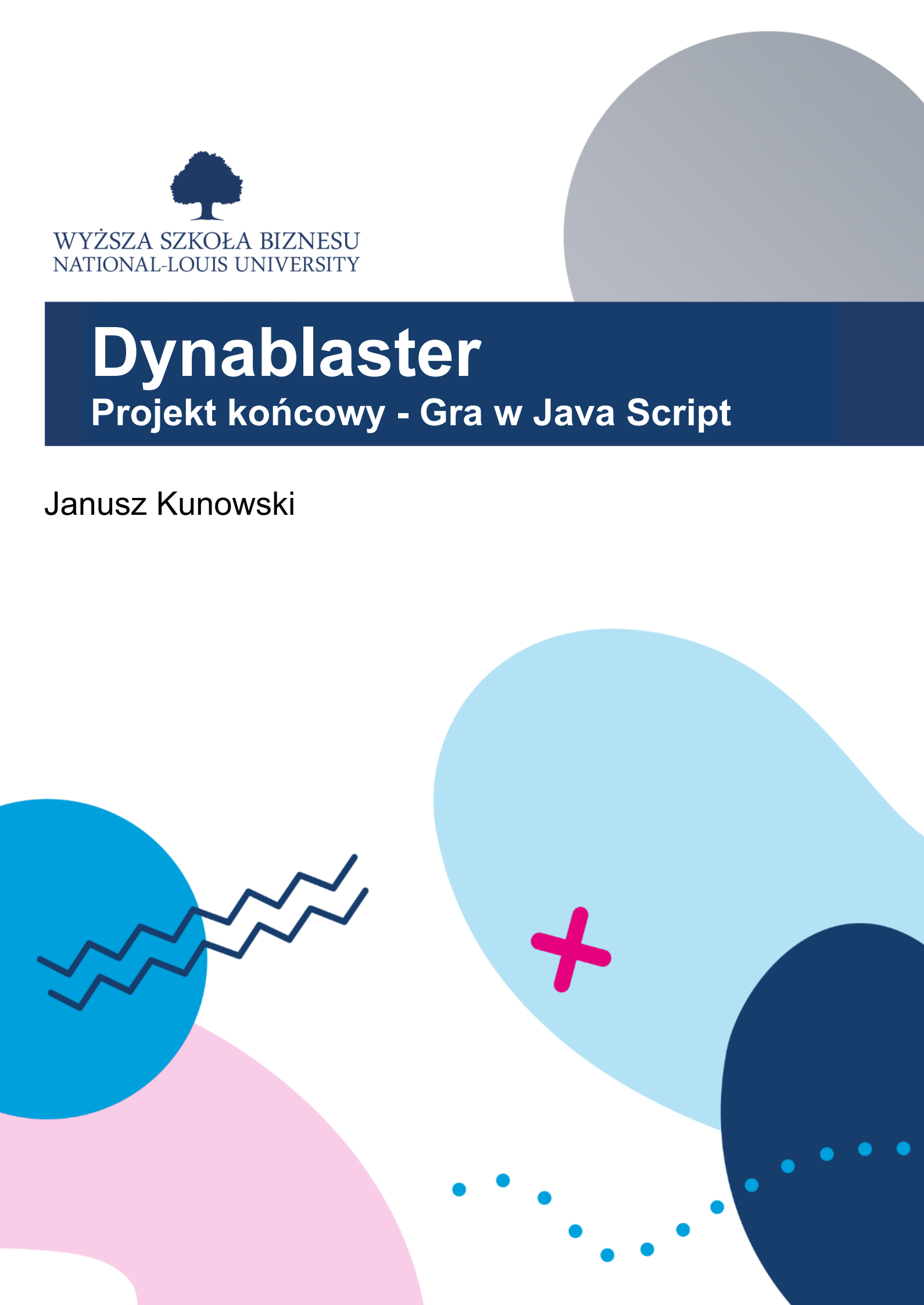


WYŻSZA SZKOŁA BIZNESU
NATIONAL-LOUIS UNIVERSITY

Dynablaster

Projekt końcowy - Gra w Java Script

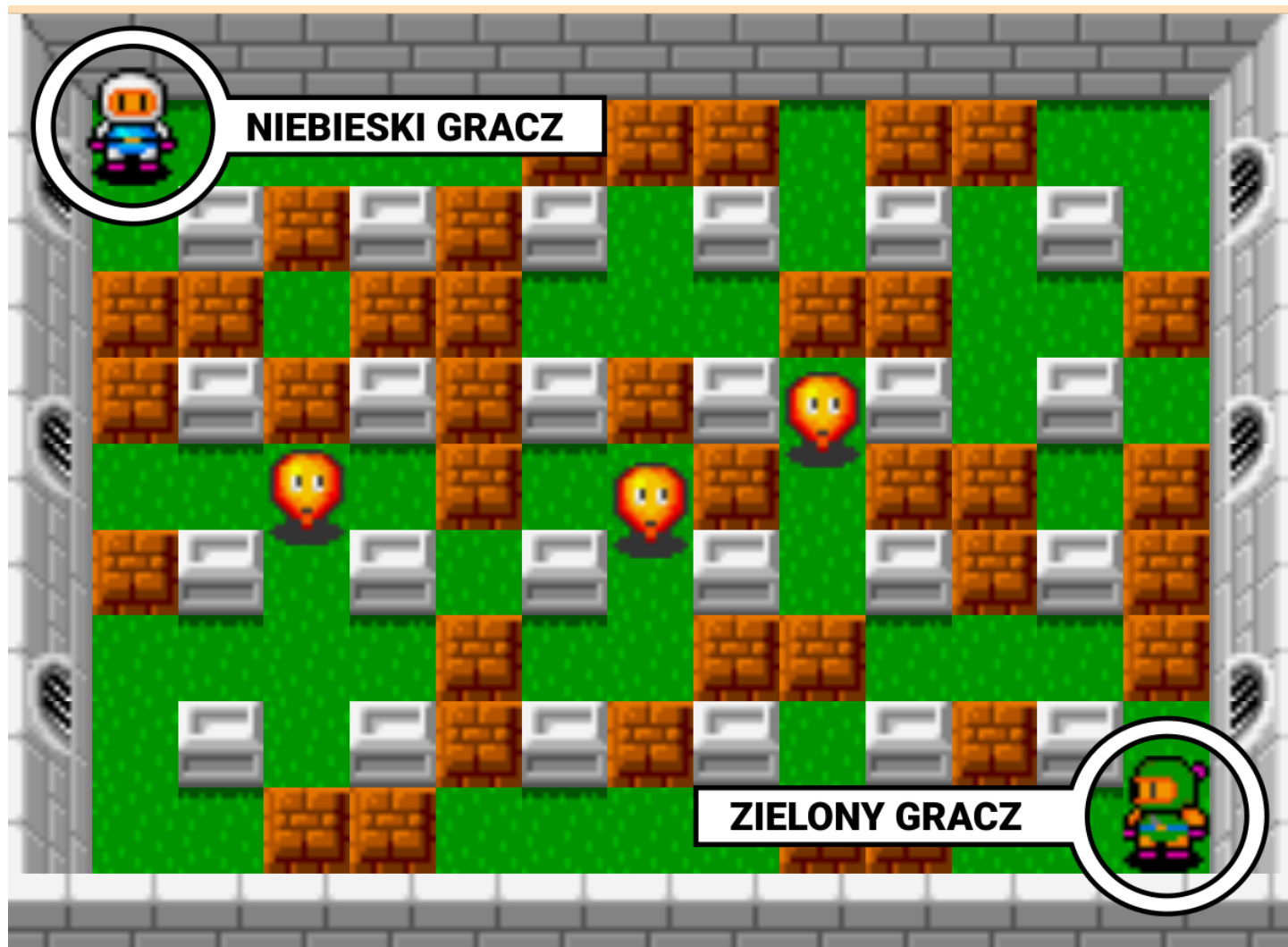
Janusz Kunowski



1. Wstęp

Jako projekt zaliczeniowy postanowiłem odtworzyć grę z mojego dzieciństwa pod tytułem Dynablaster, znana także jako Bomberman.

Postanowiłem się skupić na aspekcie gry wieloosobowej (przy jednym komputerze). Na starcie jest dwóch graczy: Niebieski i Zielony.



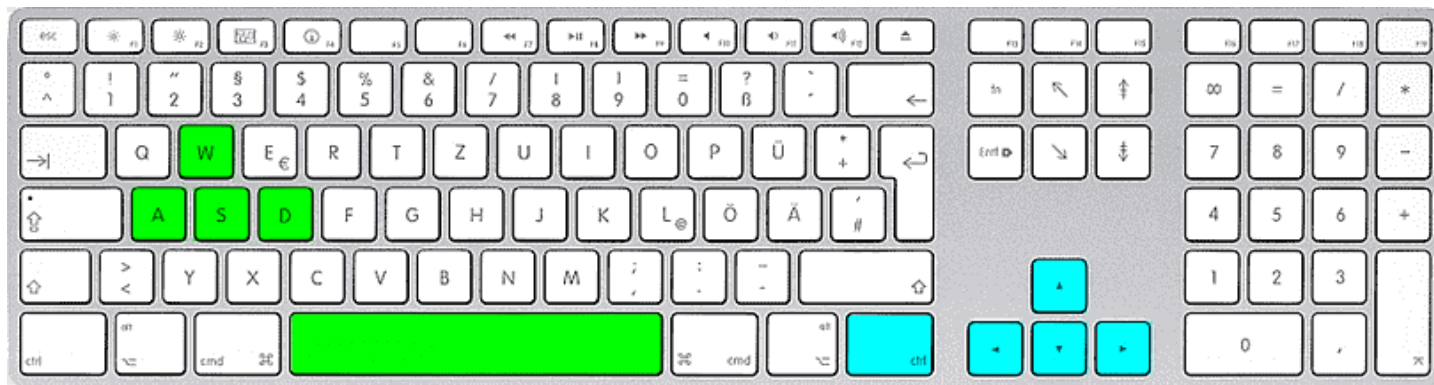
Celem każdego z graczy jest wyeliminowanie drugiego - ten, który zostanie wygrywa.

Każdy z graczy może stawiać bomby, które mogą niszczyć pomarańczowe cegły torując drogę, oraz mogą zniszczyć chodzące po mapie stworki i drugiego gracza.

Po pokonaniu drugiego gracza pojawia się z napis z informacją o tym, który gracz wygrał i gra się kończy.

Niebieski WYGRYWA!

2. Sterowanie






	 GRACZ ZIELONY 	 GRACZ NIEBIESKI 
Ruch do góry		
Ruch do dołu		
Ruch w prawo		
Ruch w lewo		
Postawienie Bomby	Spacja	Prawy ctrl

3. Mechaniki

3.1. Stawianie bomb

Gracz może stawiać bomby, którymi może zniszczyć:

-  pomarańczowe ściany,
-  potwory,
-  drugiego gracza

Na początku gracz może postawić jednocześnie tylko jedną bombę, jednak ilość ta może zostać powiększona w toku rozgrywki.

3.2. Niszczanie pomarańczowych ścian

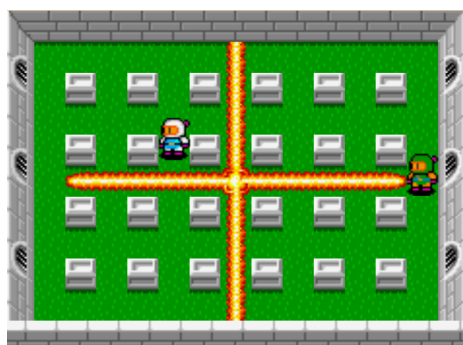
Pomarańczowe ściany blokują przejście. By utworzyć sobie drogę gracz może je zniszczyć przy pomocy bomb.



3.3. Znajdźki



Po zniszczeniu pomarańczowej ściany gracz może odkryć w jej miejscu znajdźkę, która zwiększy jego możliwości.



Siła eksplozji po zebraniu 4 znajdziek.



3.4. Potworki

Dodatkowym utrudnieniem są potwory poruszające się po planszy. Potrafią one zabić poruszającego się gracza. By je zmylić gracz musi stanąć nieruchomo, co z kolei może wykorzystać drugi gracz.

4. Zastosowane rozwiązania

4.1. Plansza do gry

Mapa układana jest z kafelków



Przy układaniu ich są stosowane różne reguły, na przykład kafelek trawy z cieniem stawiany jest w sąsiedztwie ścian, a trawa bez cienia w pozostałych miejscach. Niezniszczalne szare ściany układane są w regularnych odstępach. Zaś pomarańczowe cegły, znajdźki, oraz przeciwnicy rozmieszczani są losowo. Dodatkowo istnieje zasada sprawiająca, że cegły nie są stawiane w okolicy miejsc startowych graczy

Logika planszy do gry znajduje się w pliku **Level.js**

Kafelki mapy przechowywane są w tablicy dwuwymiarowej **Level[x][y]**.

Istnieje także tablica **DynamicObjects** przechowująca:

- 🕒 Znajdźki,
- 💣 Bomby,
- 💣 Eksplozje,
- 👾 Potwory,
- 👤 Graczy,
- 🧱 Pomarańczowe ściany.

```
Draw() : void
{
    for(let y : number = 0; y < this.SizeY; y++)
    {
        for(let x : number = 0; x < this.SizeX; x++)
        {
            this.Level[x][y].Draw();
        }
    }

    for (let i : number = 0; i < this.DynamicObjects.length; i++)
    {
        this.DynamicObjects[i].Draw();
    }
}
```

4.2. [Gfx.js] StaticSprite i AnimatedSprite

Ponieważ wszystkie grafiki pobierane są z jednej tekstury postanowiłem utworzyć pomocnicze klasy **StaticSprite** i **AnimatedSprite** służące do wyświetlania i animowania obrazków. Dzięki temu kod gry nie jest zaciemniony przez dublującą się logikę.

W klasie **AnimatedSprite** możemy dodatkowo wykorzystać delegata **OnAnimationEnded**, który zostanie wywołany po odtworzeniu animacji. Mechanikę tę wykorzystują pomarańczowe cegły (które po odtworzeniu animacji burzenia są usuwane z mapy), oraz eksplozja, która po tym jak się jej animacja odegra także znika z mapy.

```
class Brick
{
  1 usage  kunajk +1 *
  constructor()
  {
    let Sprite : HTMLImageElement = new Image();
    Sprite.src = "DynablasteOnline.png";
    this.BrickGfx = new StaticSprite(Sprite, 38, 131, 16, 16, 1, 0, Scale);
    this.DestroyedGfx = new AnimatedSprite(Sprite, 7, 56, 149, 16, 16, 6, 2, Scale, false);
    this.DestroyedGfx.OnAnimationEnded = {Instance: this, Function: this.OnDestroyed};
  }

  1 usage  new *
  OnDestroyed() :void
  {
    Mapa.RemoveObject(this);
  }
}
```

4.3. [Collision.js] Kolizje

By nie dublować kodu związanego z wykrywaniem kolizji, została utworzona klasa **RectangleCollision**

```
this.Collision = new RectangleCollision(0, 0, 16*Scale, 16*Scale, CollisionFlags.Block);
this.Collision.SetParent(this);
```

Klasa ta umożliwia ustawienia rodzica przy pomocy metody **SetParent**, dzięki czemu nie trzeba się martwić o aktualizowanie jej pozycji.

4.4. Main.js

Projekt wyewoluował z zadania robionego na potrzeby laboratorium 3 "*Sprites - sterowanie postacią*" co widać po pliku Main.js.

Główne zadania tego pliku to:

- Początkowe przygotowanie gry;
- Obsługa klawiszy;
- Zawiera klasy graczy.

4.5. Maszyna stanów

W plikach gry znajdują się także pliki **GameStateMachine.js**, oraz **GameState_Game.js** i **GameState_Menu.js**. Chciałem zastosować maszynę stanów by zrealizować przechodzenie pomiędzy grą a menu głównym, niestety nie starczyło mi czasu by to wdrożyć.

5. Podsumowanie

Pomimo uczucia niedosytu, została utworzona gra w którą mogą z powodzeniem zagrać dwie osoby.

Udało się zaimplementować:

- Sterowanie postaciami dwóch graczy;
- Kolizje;
- Eksplozje o zmiennej wielkości;
- Niszczanie ścian;
- Przeciwników;
- Znajdźki zwiększające możliwości gracza;
- Wykrywanie końca rozgrywki;
- Udźwiękowienie gry.

Dlatego pomimo, że chciałbym zrobić jeszcze więcej myślę, że projekt zakończył się sukcesem.

Gra została stworzona w taki sposób, by w przyszłości dało się ją rozszerzyć do poziomu bardziej złożonej gry, zawierającą np. Menu Główne, oraz kampanie dla jednego gracza.