

Spis treści

[1. Wprowadzenie 3](#_Toc484783417)

[a) Cel projektu 3](#_Toc484783418)

[b) Wymagania funkcjonalne 3](#_Toc484783419)

[2. Dokumentacja użytkownika 4](#_Toc484783420)

[a) O programie 4](#_Toc484783421)

[b) Wymagania techniczne 4](#_Toc484783422)

[c) Instalacja 4](#_Toc484783423)

[d) Korzystanie z programu 4](#_Toc484783424)

[3. Dokumentacja techniczna 4](#_Toc484783425)

[a) Podział programu 4](#_Toc484783426)

[b) Działanie programu 5](#_Toc484783427)

[c) Algorytmy i ważniejsze metody 5](#_Toc484783428)

[d) Zastosowane narzędzia 5](#_Toc484783429)

[4. Podsumowanie 5](#_Toc484783430)

[a) Harmonogram oraz podział prac 5](#_Toc484783431)

[b) Wnioski 6](#_Toc484783432)

[c) Spis literatury 6](#_Toc484783433)

# Wprowadzenie

## Cel projektu

Projekt „VoidCraft the game” ma na celu stworzenie gry na urządzenia mobilne działające pod systemem Android. Jednym z głównych celów tego projektu jest stworzenie aplikacji łatwej w rozbudowie tak, aby w przyszłości projekt mógł być nadal rozwijany.

Gracz będzie doświadczał rozgrywki w perspektywie 2D z widokiem z góry na swoją postać. Całość gry przedstawiona została w konwencji survival-rpg, gdzie gracz dowolnie eksplorując udostępniony mu świat, może wchodzić z nim w interakcje w celu stworzenia narzędzi niezbędnych do przeżycia postaci.

Struktura oraz mechanizmy gry zostać miały stworzone przez nas od podstaw, aby poznać lepiej zasady projektowania tego typu aplikacji. Jednymi z najważniejszych celów jaki sobie postawiliśmy, były nauka poprawnego rozplanowywania prac, poznanie nowych narzędzi służących do tworzenia takich aplikacji jak również poprawienie efektywności naszej pracy grupowej.

## Wymagania funkcjonalne

-Możliwość interakcji ze środowiskiem poprzez zbieranie przedmiotów

-Tworzenie przedmiotów przez system craftingu

-Tworzenie budynków

-Otwarty świat

-System dnia i nocy

-Możliwość tworzenia własnych map za pomocą osobnej aplikacji

# Dokumentacja użytkownika

## O programie

VoidCraft the game to gra, która postawi Cię przed wyzwaniem, jakim jest przeżycie na tajemniczej wyspie. Każda noc może okazać się Twoją ostatnią. Czy dasz radę przetrwać?

W Twoje ręce trafiła właśnie gra, która daje Ci możliwość eksploracji otwartego świata oraz poznawanie wciągającej historii bezimiennej kobiety. Specjalnie dla Ciebie opracowaliśmy system carftingu, który pozwoli ci tworzyć narzędzia służące do przetrwania oraz samemu zmieniać swoje otoczenie poprzez stawianie budynków.

Dzięki naszej grze przeżyjesz wspaniałą przygodę oraz poznasz historię, która zmrozi Ci nie raz krew w żyłach. Czy przeżyjesz?

## Wymagania techniczne

* Telefon z systemem Android w wersji 4.4 lub wyższej
* 512 MB pamięci ram lub więcej
* 50 MB wolnego na dysku
* Należy przyznać aplikacji do modyfikacji danych na karcie SD

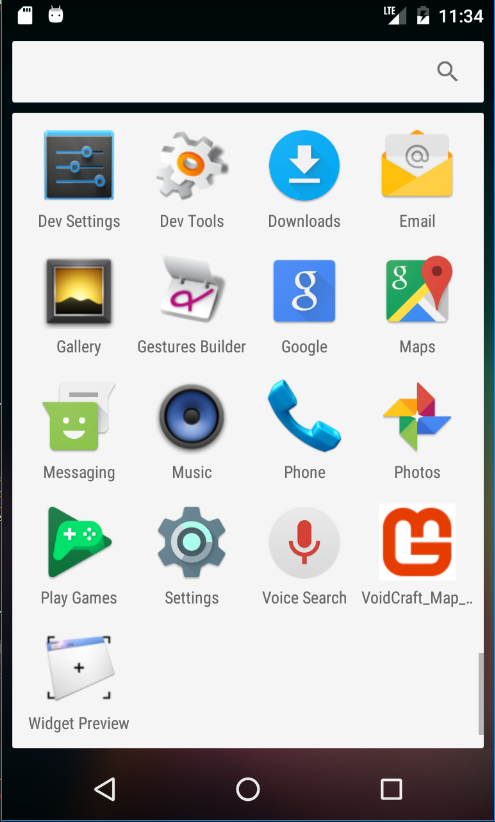
## Instalacja

Należy uruchomić plik z rozszerzeniem .apk oraz postępować według instrukcji. Jeśli aplikacja została pobrana z innego źródła niż SklepPlay to należy wyłączyć w ustawieniach telefonu opcję „Instaluj tylko z zaufanych źródeł”.

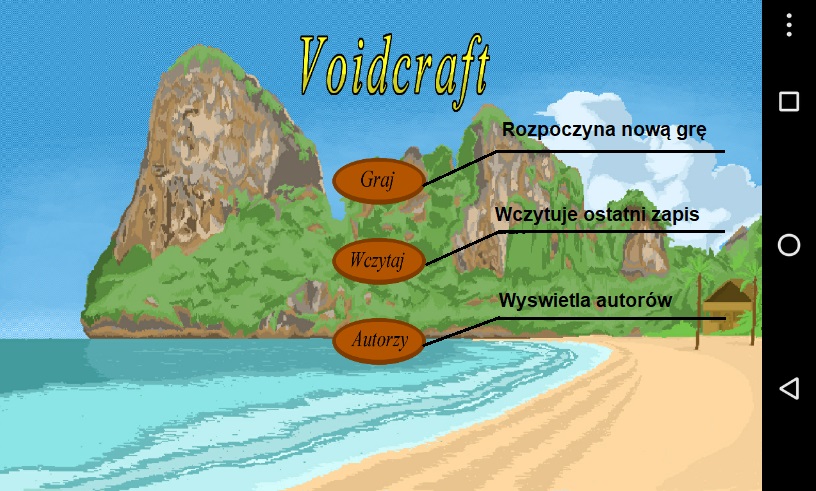
## Korzystanie z programu

TODO: + screeny  
-pierwsze uruchomienie  
-wczytanie oraz zapis gry  
-opcje  
-poruszanie się w grze, crafting i misje  
-autorzy

Aplikację należy uruchamiać z poziomu menu głównego telefonu < poniżej screen tymaczsowy>. Należy ja znaleźć a następnie dotknąć jej ikony.



W wyniku powyższych, naszym oczom ukaże się menu główne gry



# Dokumentacja techniczna

## Podział programu

Na poniższym rysunku jest przedstawiony graf, który odwzorowuje połączenia między głównymi klasami programu.

## 

Poniżej zostały opisane najważniejsze klasy oraz ich najważniejsze z puntu widzenia projektu metody.

ToDo:

-Opis kreatora map

-opis klas oraz diagram

-opis maina, menu w grze

**Klasa Player**

Klasa ta jest odpowiedzialna za przechowywanie wszystkich statystyk gracza takich jak zdrowie, głód, posiadane przedmioty ale także zmienne odpowiadające za poprawną animację postaci np. ilość klatek w szablonie postaci.

Odpowiada ona także za rysowanie animacji w głównej pętli programu, ponadto jest także serializowana do pliku XML, z którego jest odczytywany stan zapisanej gry.

Serializacja tej klasy odbywa się za pomocą metod LoadPlayer oraz SavePlayer.

public static Player LoadPlayer()

{

var store = IsolatedStorageFile.GetUserStoreForApplication();

XmlSerializer xmlFormat = null;

try

{

xmlFormat = new XmlSerializer(typeof(Player));

}

catch (Exception ex)

{

Debug.Write(ex.InnerException.ToString());

}

if (store.FileExists("Gracz.xml"))

{

var fs = store.OpenFile("Gracz.xml", FileMode.Open);

using (StreamReader sw = new StreamReader(fs))

{

return (Player)xmlFormat.Deserialize(sw);

}

}

else throw new Exception("Brak pliku z zapisem gry");

}

public void SavePlayer()

{

var store = IsolatedStorageFile.GetUserStoreForApplication();

XmlSerializer xmlFormat = null;

try

{

xmlFormat = new XmlSerializer(typeof(Player));

}

catch (Exception ex)

{

Debug.Write(ex.InnerException.ToString());

}

if (store.FileExists("Gracz.xml"))

{

store.DeleteFile("Gracz.xml");

}

var fs = store.CreateFile("Gracz.xml");

using (StreamWriter sw = new StreamWriter(fs))

{

xmlFormat.Serialize(sw, this);

}

if (store.FileExists("Gracz.xml"))

{

var fss = store.OpenFile("Gracz.xml", FileMode.Open);

using (StreamReader sr = new StreamReader(fss))

{

string xmls = sr.ReadToEnd();

Debug.Write(xmls);

store.Close();

}

}

else

{

Debug.Write("Plik nie istnieje");

throw new Exception("Brak pliku z zapisem gry");

}

}

}

Rysowanie postaci na mapie odbywa się za pomocą metody Draw, która na podstawie podanej pozycji, ilości klatek, szablonu postaci oraz aktualnej klatki oblicza co ma zostać wyrysowane na mapie.

Sterowanie animacją postaci realizowane jest przy pomocy metody Move, która przyjmuje typ wyliczeniowy „Direction”, zmieniający się odpowiednio podczas naciskania klawiszy sterujących, oraz listę tekstur. Na tej podstawie zmienia aktualnie wyświetlaną teksturę przez metodę Draw.

public void Move(Direction direction, List<Texture2D> tx)

{

switch (direction)

{

case Direction.Idle\_Down:

{ IsMoving = false; Texture = tx [4]; break; }

case Direction.Up:

{ IsMoving = true; Texture = tx [2]; break; }

case Direction.Down:

{ IsMoving = true; Texture = tx [3]; break; }

case Direction.Left:

{ IsMoving = true; Texture = tx [1]; break; }

case Direction.Right:

{ IsMoving = true; Texture = tx [0]; break; }

case Direction.Idle\_Left:

{ IsMoving = false; Texture = tx [5]; break; }

case Direction.Idle\_Right:

{ IsMoving = false; Texture = tx [6]; break; }

case Direction.Idle\_Back:

{ IsMoving = false; Texture = tx [7]; break; }

default:

break;

}

}

Metoda symulująca głód w grze „Spadek\_Glod” przyjmuje jako argument czas gry, dzięki temu możliwe jest obniżanie statystyki głodu z prędkością określoną za pomocą zmiennej Glod\_Predkosc.

public void Spadek\_Glod(GameTime gameTime)

{

Glod\_Czas += gameTime.ElapsedGameTime.Milliseconds\*20;

if (Glod\_Czas > Glod\_Predkosc)

{

Glod\_Czas -= Glod\_Predkosc;

if (GLOD != 0)

{ GLOD -= 1; }

if (GLOD < 20)

{

try

{

materials.Food -= 5;

}

catch (RawMaterials.OutOfFoodException ex)

{

Game1.messages.AddMessage("jesteś głodny”,new Rectangle(10,10,100,100));

HP -= 1;

}

}

Glod\_Czas = 0;

}

}

**Klasa RawMaterials**

Klasa opisuje surowce dostępne w grze. Są to drewno, kamień, liany, metal, woda oraz jedzenie. Posiada ona własne wyjątki opisujące brak danych surowców.

Jedna z właściwości tej klasy:

public int Food

{

get { return \_food; }

set {

if (\_food + value < 0)

{

throw new OutOfFoodException("Zbyt mało jedzenia!");

} else

{

\_food = value;

}

}

}

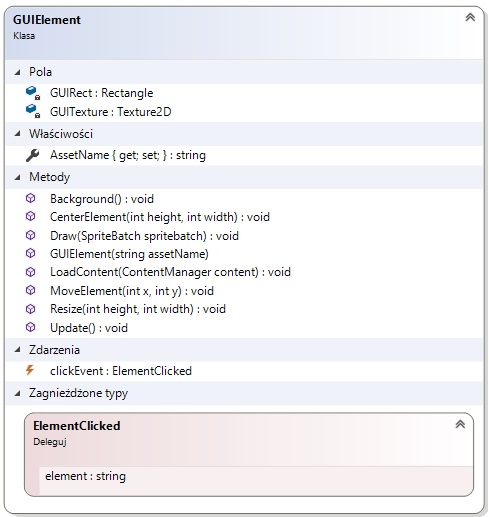
**Klasa Tool**

Klasa opisuje narzędzia, które użytkownik może tworzyć w trakcie rozgrywki. Zawiera informacje o obiekcie **RawMaterials** który opisuje wymagania potrzebne do stworzenia przedmiotu oraz tablicę obiektów **Tool** która stanowi o innych narzędziach potrzebnych do utworzenia danego przedmiotu.

**Menu główne**

Menu główne działa w oparciu o dwie klasy. Pierwsza GUIElement służy do opisu poszczególnych części menu, tj. tło, logo, przyciski. Konstrukcja głównego menu zakłada łatwą rozszerzalność o dodatkowe funkcje.

* **GUIElement**



Klasa ta opiera się na dwóch polach tworzących pojedynczy element ( stąd też nazwa) menu. Pierwsze z nich określa obszar a drugi teksturę. Oprócz pól każdy element ma również przypisane zdarzenie oraz delegat.

Metody Klasy GUIElement:

public void LoadContent(Content Manager content) – metoda ustawiająca domyślne wartości pół elementu, tj. ładuje texturę z pliku oraz przypisuje początkowy obszar ( prostokąt) który będzie zajmował element.

public void Update() – metoda sprawdzająca czy dotknięty został obszar elementu ( prostokąta ) i jeśli tak, wywołuje zdarzenie clickEvent.

public void Draw( SpriteBatch spritebatch) – metoda odpowiedzialna za narysowanie elementu.

public void CenterElement( int height, int width) – metoda służaca do wyświetlania elementów na środku ekranu.

public void Resize ( int height, int width) – metoda pomocnicza pozwalająca manipulację szerokością i wysokością elementu.

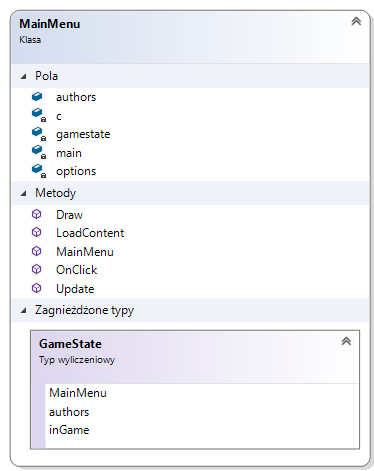
public void Background() – metoda ustawiająca element jako tło, tj. rozciągająca go odpowiednio do wielkości ekranu

public void MoveElement(int x, int y) – przesuwa element bez zmiany jest wymiarów

public event ElementClicked clickEvent

public delegate void ElementClicked(string element) – delegat do późniejszego wykorzystania przy implementacji reakcji na naciśnięcie

* **MainMenu**

****

Klasa zarządzająca kontenerami obiektów typu GUIElement. Poruszanie się po menu głównym polega na zmianie stanu instancji typu wyliczeniowego „GameState” widocznego powyżej. W zależności od stanu wyświetlane są określone obiekty klasy GUIElement.

public void LoadContent(Content Manager content) – metoda wywołująca wczytywanie tekstur dla każdego elementu w poszczególnych kontenerach oraz odpowiednio je rozmieszczająca za pomocą funkcji CenterElement, MoveElement, Background oraz Resize

public void Update() – metoda sprawdzająca w zależności od GameState kliknięcie na przycisk.

public void Draw( SpriteBatch spritebatch) – metoda odpowiedzialna wywoływanie rysowania poszczególnych elementów w zależności od GameState’u

public void OnClick( string element) – metoda dodawana do delegatu określająca zachowanie aplikacji po naciśnięciu na przycisk. Rozróżnianie jest to, czy aktualnie jesteśmy w menu głównym czy w submenu.

## Działanie programu

ToDo: rozgrywka

## Algorytmy i ważniejsze metody

## Zastosowane narzędzia

* + Microsoft Visual Studio Enterprise 2015
  + Biblioteki MonoGame
  + Język C#
  + System kontroli wersji GitHub

## Podsumowanie

## Harmonogram oraz podział prac

Uzupełnijcie sobie sami:

Alan Biegun – twórca kreatora map oraz opracowanie poruszania się po mapie  
Piotr Zuber – planista, opracowanie systemu craftingu  
Konrad Sladkowski – twórca głównego menu  
Jan Snopek – grafik i animator, twórca klasy gracza  
Dawid Ryl - grafik i animator, twórca projektu mapy

## Wnioski

Realizacja tego projektu uświadomiła nam, jak wiele możliwości otwiera podejście obiektowe w programowaniu. Szczególny nacisk położono na staranne rozplanowanie prac oraz komunikacje w zespole, co mimo wielu trudności z czasem zaowocowało.

Projekt przyniósł wraz z wieloma wyzwaniami tyle samo doświadczeń. Co prawda nie tylko tych przyjemnych aczkolwiek na pewno przydatnych w przyszłej pracy zawodowej. Przejrzysty podział obowiązków i ról, ustalanie terminów „deadline” czy wzajemna pomoc nauczyły nas, że komunikacja w zespole jest równie ważna, jak umiejętności programistyczne.

## Spis literatury

* Andrew Troelsen, “Język C# 2010 i platforma .NET 4”
* Serwis internetowy http://www.monogame.net/documentation/?page=main