Střední průmyslová škola Ostrov,

příspěvková organizace

ROČNÍKOVÁ PRÁCE

RPG

(Random Power Gates)

(desktopová aplikace)

Studijní obor	**Informační technologie**	
Třída	**I3**	
Školní rok	**2015/2016**	**Jan Novák**

Prohlášení autora

„Prohlašuji, že jsem tuto práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a zdrojů informací."

V Ostrově dne …… …….

Podpis autora

(Zde bude vložené zadání)

Anotace

Obsahem dokumentace je popis vývoje a designu kódu aplikace. Dále jsou zde rozebrány jednotlivé problémy, které nastaly během vývoje jako například kolize s objekty při nestandardním pohybu, načítání mapy ze „seedu“ a další problémy spojené s Monogame Frameworkem

Anotation

Documentation is describing workflow of game development. Next, there are problems described which occur during a development like collisions, loading map from „seed“ and more problems whit Monogame framework itself.

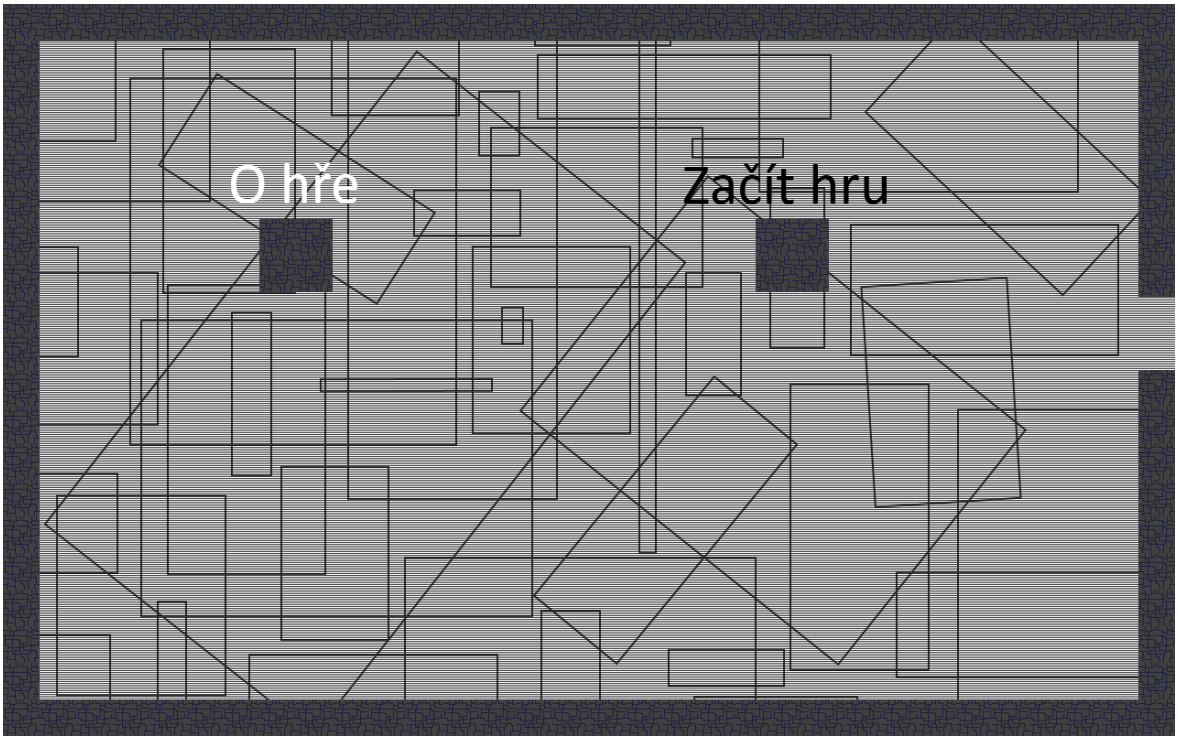
Obsah

Úvod

Cíl této práce byl vytvořit hru s názvem RPG(random power gates), v niž se hráč ocitne v jedné z mnoha arén kde musí porazit své protivníky poté se může odebrat branou do další arény. Aby nakonec skončil v poslední aréně která je zároveň cestou ven. Požitý byl jazyk C# ve verzi 6 společně se Monogame Frameworkem ve verzi 3.5. jako IDE jsem zvolil Visual studio 2015 Comunity. Dokumentace obsahuje popis vývoje tj. jednotlivé ukázky kódů a k nim přidružené pohledy (pokud se jedná o grafiku), popis specifických problému a jejich řešení (pokud je k dispozici).

Rozbor aplikace z uživatelského pohledu (návrh)

Po spuštění hry se ihned ocitnete v „Menu“ aréně kde nejsou žádní nepřátelé jen dva sloupy s nadpisem O hře a Začít hru, kdy po kliknutí na O hře se otevře okno s informacemi a Začít hru vás přenese do první bojové arény. Každá aréna je obehnána zdí se jedním vchodem a jedním východem, který se otevře po vyčištění zóny, kde se právě nacházíte, výjimkou je jen menu



Obrázek 1 – Menu hry

Analytická dokumentace

Zdrojový kod gry je členěn do mnoha tříd což zajišťuje přehlednost a možnost rychle najít chyby. Všechny textury jsou umístěny ve složce Content, což je výchozí složka pro tento účel v prostředí Monogame

Následuje seznam tříd a jejich stručný popis:

AiManager – Stará se o „inteligenci“ a pohyb ne hráčských charakterů (NPC).

Npc – definice autonomního počítačem řízeného animovaného objektu.

AnimatedSprite – Základní třída pro všechny animované objekty.

AtackManager – Stará se o hráčovy útoky tak o utočení nepřátel.

Crosshair – Definice zaměřovače, který hráč používá pro přesnou střelbu.

Game1 – před generovaná třída obsahující vše potřebné od inicializace přes logiku hry až po vykreslení textur.

Global – „Singleton“ třída uchovávající všechny potřebné globální proměné.

MapManager – Stará se o vytvoření mapy ze zdroje (seedu) a popřípadě překreslování mapy při přecházení mezi zónami.

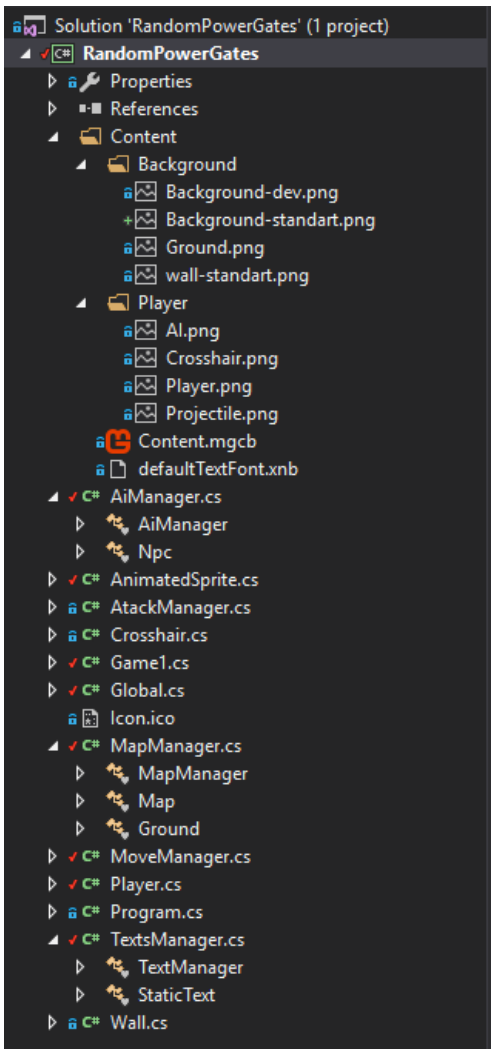
Map – Definice mapy tj. pole jednotlivých polí.

Ground – Definice podlahy nebo země. Určená jako podlážka místa pro průchod do další mapy.

MoveManager – Stará se o pohyb hráče a kolize s herními objekty.

Player – Třída obsahující definici hráče.

Program – Automaticky vygenerovaná třída volající třídu Game1.



Obrázek 2 – seznam tříd

TextsManager – Stará se o vytváření a vykreslování textů na obrazovku.

StaticText – Definice neměnného textu.

Wall – Třída obsahující definici zdi.

Analýza jednotlivých tříd:

AnimatedSprite

Tato třída obsahuje definici a logiku pro animovaný objekt. Jedná se o abstraktní třídu, tudíž není možné při vytvořit instanci objektu. Nicméně „nutí“ derivovanou třídu implementovat všechny nezbytné metody a proměnné pro správné fungování.

Ukázka třídy :

proměnné

```
public Vector2 position;
protected Texture2D objectTexture;
private Rectangle[] objectRectangles;
public Rectangle objectBounds;
private int frameIndex;
private int frameWidth;
protected int timePerFrame = 500;
private int timeSinceLastFrame = 0;
Vector2 direction;
```

Obrázek 3 – proměnné animovaného objektu

metody

```
//konstruktor objektu
public AnimatedSprite(Vector2 position, int timePerFrame)
{
    this.timePerFrame = timePerFrame;
    this.position = position;
}
//metoda na přidání animace
public void AddAnimation(int frames)
{
    frameWidth = objectTexture.Width / frames;
    objectRectangles = new Rectangle[frames];
    for (int i = 0; i < frames; i++)
    {
        objectRectangles[i] = new Rectangle(i * frameWidth, 0, frameWidth, objectTexture.Height);
    }
}
//metoda, která se volá pravidelně a obsahuje logiku
public void Update(GameTime gameTime)
{

    direction = new Vector2(Global.instance.crosshair.position.X - position.X, Global.instance.crosshair.position.Y - position.Y);
    Global.instance.angle = -(float)Math.Atan2(direction.X, direction.Y);

    timeSinceLastFrame += gameTime.ElapsedGameTime.Milliseconds;
    objectBounds = new Rectangle((int)position.X, (int)position.Y, frameWidth, objectTexture.Height);
    if (timeSinceLastFrame >= timePerFrame)
    {
        if (!(frameIndex < objectRectangles.Length - 1))
        {
            frameIndex = 0;

        }
        else
            frameIndex++;
        timeSinceLastFrame = 0;
    }
}
//metoda, která se volá pravidelně a obsahuje vykreslování objektu
public void Draw(SpriteBatch spriteBatch)
{
        spriteBatch.Draw(objectTexture, position, objectRectangles[frameIndex], Color.White);
}
```

Obrázek 4 – metody animovaného objektu

Player

Třída Player obsahuje definici hráče a jako je rychlost pohybu a metodu pro načtení textury do paměti

```
class Player : AnimatedSprite
{
    //rychlost pohybu hráče
    public float objectSpeed = 3.5f;
    //konstruktor hráče
    public Player(Vector2 position, int timePerFrame) : base(position, timePerFrame)
    {

    }
    //metoda, která se volá při spuštění a načítá texturu hráče
    public void LoadContent(ContentManager contentManager, string texturePath)
    {
        //default = "Player/Player.png"
        objectTexture = contentManager.Load<Texture2D>(texturePath);
        AddAnimation(8);
    }
}
```

Obrázek 5 – tělo třídy Player

Move manager

```
class MoveManager
{
    //dočasné kolizní okraje hráče
    Rectangle tempRectangle;
    //metoda volající se při každém pokusu se pohnout do jakéhokoliv směru
    public void Move(KeyboardState keyboardState)
    {
        tempRectangle = Global.instance.player.objectBounds;
        if (keyboardState.IsKeyDown(Keys.W))
        {
            tempRectangle = Global.instance.player.objectBounds;
            tempRectangle.Y -= (int)Global.instance.player.objectSpeed;
            if (!ColisonCheck(tempRectangle))
                Global.instance.player.position.Y -= Global.instance.player.objectSpeed;
        }
        if (keyboardState.IsKeyDown(Keys.S))
        {
            tempRectangle = Global.instance.player.objectBounds;
            tempRectangle.Y += (int)Global.instance.player.objectSpeed;
            if (!ColisonCheck(tempRectangle))
                Global.instance.player.position.Y += Global.instance.player.objectSpeed;
        }
        if (keyboardState.IsKeyDown(Keys.A))
        {
            tempRectangle = Global.instance.player.objectBounds;
            tempRectangle.X -= (int)Global.instance.player.objectSpeed;
            if (!ColisonCheck(tempRectangle))
                Global.instance.player.position.X -= Global.instance.player.objectSpeed;
        }
        if (keyboardState.IsKeyDown(Keys.D))
        {
           tempRectangle = Global.instance.player.objectBounds;
            tempRectangle.X += (int)Global.instance.player.objectSpeed;
            if (!ColisonCheck(tempRectangle))
                Global.instance.player.position.X += Global.instance.player.objectSpeed;
        }
    }
    //metoda testující jestli došlo ke kolizi
    private bool ColisonCheck(Rectangle rectangle)
    {
#if DEBUG
        if (rectangle.Intersects(Global.instance.wall1.GetWallBounds()) || rectangle.Intersects(Global.instance.wall2.GetWallBounds()) || rectangle.Intersects(Global.instance.wall3.GetWallBounds()))
            return true;
        else
            return false;
#endif
        foreach (Wall w in Global.instance.walls)
        {
            if (rectangle.Intersects(w.GetWallBounds()))
                return true;
            else
                return false;
        }
        return false;
    }
}
```

Obrázek 6 – Tělo třídy MoveManager

Tento Manažer se stará o pohyb hráče a kolize. Kolize se testují pomocí vytvoření dočasné kolizní zóny, která je stejná jako kolizní zóna hráče s tím rozdílem že je posunuta do směru kudy se chce hráč pohnout a pokud je splněna podmínka že tato vytvořená zóna nekoliduje s žádnou zónou zdi, hráč se pohne daným směrem

Wall

Třída definující zeď, její pozici, texturu a kolizní zónu.

```
class Wall
{
    //Pozice zdi
    internal Vector2 position;
    //Textura zdi
    private Texture2D wallTexture;
    //Kolizní zóna
    private Rectangle wallBounds;
    //Konstruktor
    public Wall(Vector2 position)
    {
        this.position = position;
    }
    //metoda načítající texturu zdi do paměti
    public void LoadContent(ContentManager contentManager, string texturePath)
    {
        //"Background/wall-standart.png"
        wallTexture = contentManager.Load<Texture2D>(texturePath);
        wallBounds = new Rectangle((int)position.X, (int)position.Y, wallTexture.Width, wallTexture.Height);

    }
    //Get metoda pro získání textury zdi
    public Texture2D GetWallTexture() { return wallTexture; }
    //Get metoda pro získání kolizní zóny zdi
    public Rectangle GetWallBounds() { return wallBounds; }

    //metoda vykreslování
    public void Draw(SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(wallTexture, position, Color.White);
    }
}
```

Obrázek 7 – Tělo třídy Wall

Map manager

Tento manažer se stará o vykreslení celé mapy tj. zdí, podlážky a pozadí. Při generování zdí a podlážky si generátor převezme seed(zdrojový string) z objektu map naplní pole obsažené v mapě a následně vygeneruje mapu samotnou.

Metody Update a Draw jsou stejné jak pro jiné manažery.

```
        #region Walls
        public void addWall(Vector2 WallPosition)
        {
            Global.instance.walls.Add(new Wall(WallPosition));
        }
        #endregion
        #region Background
        public void addGround(Vector2 position)
        {
            Global.instance.grounds.Add(new Ground(position));
        }

        #endregion

        public void Initialize()
        {
#if DEBUG
            //workin' walls
            Global.instance.wall1 = new Wall(new Vector2(40, 40));
            Global.instance.wall2 = new Wall(new Vector2(80, 40));
            Global.instance.wall3 = new Wall(new Vector2(120, 40));

#else
            //Generating map from seed
            int x = -1, y = 0;
            Global.instance.map = new Map();
            for (int i = 0; i < Global.instance.map.defaultMap.Length; i++)
            {
                x++;
                if (Global.instance.map.defaultMap[i] == '1')   //GROUND
                    Global.instance.map.walls[x, y] = 1;
                if (Global.instance.map.defaultMap[i] == '2')   //WALL
                    Global.instance.map.walls[x, y] = 2;
                if (Global.instance.map.defaultMap[i] == 'n')   //NEW LINE
                {
                    x = -1;
                    y++;
                }
            }
            for (int i = 0; i < Global.instance.map.walls.GetLength(0); i++)
            {
                for (int k = 0; k < Global.instance.map.walls.GetLength(1); k++)
                {
                    if (Global.instance.map.walls[i, k] == 1)
                        addGround(new Vector2(40 * i, 40 * k));
                    if (Global.instance.map.walls[i, k] == 2)
                        addWall(new Vector2(40 * i, 40 * k));
                }
            }
#endif
```

Obrázek 8 – kód generování mapy

Map

Mapa je jednoduchá definice mapy jako datového typu s jedním polem a jedním textovým řetězcem

```
class Map
{
    public string defaultMap = "0n0n0n0n000021212121n0002121212";
    public int[,] walls = new int[32, 20];

    public Map()
    {
    }
}
```

Obrázek 9 – Tělo třídy Map

Ground(země / podlážka)

Ground je jednoduchá definice země. Obsahuje texturu a pozici země.

Metoda LoadContent je zde stejná jako v případě definice zdi(Wall)

```
class Ground
{
    public Texture2D groundTexture;
    public Vector2 groundPosition;

    public Ground(Vector2 groundPosition)
    {
        this.groundPosition = groundPosition;
    }
    public void LoadContent(ContentManager contentManager, string texturePath)
    {
        groundTexture = contentManager.Load<Texture2D>(texturePath);
    }
}
```

Obrázek 10 – Tělo třídy Ground

AI Manažer

AI manažer se stará o řízení a vykreslování ne hráčských charakterů. Základní a zatím jediní dostupní boti pouze následují hráče zmízí pokud se ho dotknou.

```
class AiManager
{
    public AiManager()
    {
        //tři testovací boti, kteří zmizí pokud se jich dotkne hráč
        addAI(new Vector2(1000, 700));
        addAI(new Vector2(500, 700));
        addAI(new Vector2(1000, 50));
    }
    //metoda pro přidání NPC na herní plochu
    public void addAI(Vector2 npcPosition)
    {
        Global.instance.npcs.Add(new Npc(npcPosition, 500, 1));
    }
    //Načítání textury
    public void LoadContent(ContentManager contentManager)
    {
        foreach (Npc n in Global.instance.npcs)
        {
            n.LoadContent(contentManager, "Player/AI.png");
        }
    }
    //metoda vykonávající logiku
    public void Update(GameTime gameTime)
    {
        foreach (Npc n in Global.instance.npcs)
        {
            if (Global.instance.player.position.X < n.position.X)
                n.position = new Vector2(n.position.X - n.speed, n.position.Y);
            if (Global.instance.player.position.X > n.position.X)
                n.position = new Vector2(n.position.X + n.speed, n.position.Y);
            if (Global.instance.player.position.Y < n.position.Y)
                n.position = new Vector2(n.position.X, n.position.Y - n.speed);
            if (Global.instance.player.position.Y > n.position.Y)
                n.position = new Vector2(n.position.X, n.position.Y + n.speed);

            n.Update(gameTime);
        }
        for (int i = 0; i < Global.instance.npcs.Count; i++)
        {
            if (Global.instance.npcs[i].objectBounds.Intersects(Global.instance.player.objectBounds))
            {
                Global.instance.npcs.RemoveAt(i);
                i--;
            }
        }
    }
```

Obrázek 11 – Tělo třídy AiManager

NPC(Non Player Character)

NPC neboli ne hráčský charakter. Je definován stejně jako hráčský objekt.

```
class Npc : AnimatedSprite
{
    //rychlost NPC-čka
    public int speed = 1;
    public Npc(Vector2 position, int timePerFrame, int speed) : base(position, timePerFrame)
    {
        this.speed = speed;
    }
    public void LoadContent(ContentManager contentManager, string texturePath)
    {
        //default = "Player/Player2.png"
        objectTexture = contentManager.Load<Texture2D>(texturePath);
        AddAnimation(4);
    }
}
```

Obrázek 12 – Tělo třídy Npc

Crosshair (zaměřovač) [non release]

Zaměřovač měl původně sloužit k přesnému míření při střílení pod jakýkoliv úhlem, nicméně z důvodu technického omezení a nedostatečné dokumentace, jsem byl nucen rotaci vynechat a tak už zaměřovat zbyl jen jako nástroj pro testování například pro generování nových botů za běhu programu.

```
class Crosshair
{
    //Textura zaměřovače
    private Texture2D crosshairTexture;
    //Pozice zaměřovače
    public Vector2 position;


    public Crosshair(){ }
    //načtení obsahu
    public void LoadContent(ContentManager contentManager, string texturePath)
    {
        crosshairTexture = contentManager.Load<Texture2D>(texturePath);
    }
    //metoda vykonávající logiku
    public void Update(GameTime gameTime, MouseState mouseState)
    {
        if (position.X != mouseState.X || position.Y != mouseState.Y)
        {
            position = new Vector2(mouseState.X, mouseState.Y);
        }

    }
    //vykreslování
    public void Draw(SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(crosshairTexture, position, Color.Wheat);
    }
}
```

Obrázek 13 – Tělo třídy Crosshair

Global

Specificky navržená třída, která je takzvaným jedináčkem (z angl. singleton). To zajišťuje, že je pouze jedna instance objektu a žádná už nelze vytvořit. Toto řešení je výhodné zejména při přístupu k proměnné, kdy statický přístup trvá déle než nestatický. Třída Global uchovává všechny proměnné, instance a kolekce ke kterým je potřeba přistupovat odkudkoliv z kódu.

```
class Global
{
    //instance objektu Global
    public static readonly Global instance = new Global();
    //Mapa
    public MapManager mapManager;
    public int windowWidth, windowHeight;

    public List<Wall> walls = new List<Wall>();
    public Map map;
    public List<Ground> grounds = new List<Ground>();

#if DEBUG
    //workin' walls
    public Wall wall1;
    public Wall wall2;
    public Wall wall3;
#endif
    //Pozadí
    public Ground background;
    //Hráč
    public Player player;
    //Zaměřovač
    public Crosshair crosshair;
    //Textura projektilu
    public Texture2D projectileTexture;
    //Manažer útoku
    public AtackManager atackManager;
    public float angle;
    //Manažer textů
    public TextManager textManager;
    //Manažer pohybu
    public MoveManager moveManager;
    //AI & NPX
    public AiManager aiManager;
    public List<Npc> npcs = new List<Npc>();
    //privátní konstruktor objektu Global znemožňující instanci objektu vytvořit
    private Global() { }

}
```

Obrázek 14 – Tělo třídy Global

Řešení zajímavých problémů

Rotace:

V Monogame je vykreslování řešené metodou Draw z třídy SpriteBatch. Díky tomuto řešení je vykreslování hladší, plynulejší a hlavně výpočetně méně náročné. A pokud je potřeba rotovat texturou je to jen otázka změny parametru rotace a parametru z názvem origin, který dle dokumentace levý horní roh a není specifikováno čeho, takže při nastavení čehokoliv jiného než vektor„0“ se začne textura vykreslovat jinde než je stanovená pozice, toto má přímo vliv na kolizní engine a jeho nefunkčnost.

Cyklus foreach = „just for first“

Při používaní cyklu foreach, který má projet celou kolekci a provést kód pro každý prvek, se kód provádí pouze pro první prvek kolekce. Tento problém opět ovlivňuje kolizní engine. Problém se projevuje při testování , jestli hráč nenarazil do zdi, tím že hráč může volně procházet jakýmkoliv objektem krom první přidané zdi do kolekce. Jako řešení se nabídlo vytvoření instance objekt zdi do samostatné proměnné. Což se ukázalo jako velice neefektivní protože by bylo potřeba vytvořit až stovky samostatných proměných.

Testování aplikace

První sestava:

CPU:	AMD Phenom X4 9950 Black Edition
RAM:	8192 MiB
GPU:	Nvidia GTX 550 Ti
OS:	Windows 10 64-bit
Rozlišení monitoru:	1920 * 1080

Druhá sestava:

CPU:	Intel Core I5-5200U, 2,5Ghz
RAM:	8192 MiB
GPU:	Nvidia GeForce 650M
OS:	Windows 10 64-bit
Rozlišení monitoru:	1320 * 768

Závěr

Cíl této práce byl vytvořit hru s názvem RPG(random power gates), v niž se hráč ocitne v jedné z mnoha arén kde musí porazit své protivníky poté se může odebrat branou do další arény. Aby nakonec skončil v poslední aréně která je zároveň cestou ven. Požitý byl jazyk C# ve verzi 6 společně se Monogame Frameworkem ve verzi 3.5. jako IDE jsem zvolil Visual studio 2015 Comunity. Dokumentace obsahuje popis vývoje tj. jednotlivé ukázky kódů a k nim přidružené pohledy (pokud se jedná o grafiku), popis specifických problému a jejich řešení (pokud je k dispozici).

Uvedete, co se vám povedlo a co nepovedlo realizovat. Nápady na možná vylepšení (do budoucna).

Během psaní této dokumentace se povedlo 50% plánovaných věcí, dalších 20% bude realizováno v dalším brzkém vydání. Následných 15% nemá jisto realizovatelnost, jelikož se odráží od úspěšnosti implementování nadcházejících součástí a finálních 15% jsou jen návrhy, které nejsou s aktuálními možnostmi implementovatelné. Jako například plné otáčení hráče a nepřátel.

Seznam použitých zdrojů

Game Development. Stack Exchange Inc. *StackExchange* [online] [cit. 2016-05-29].

Dostupné z : http://gamedev.stackexchange.com/

Seznam použitého softwaru

Microsoft Visual Studio 2015 Comunity (© 2016 Microsoft Corporation.)

Paint.NET (© 2016 Microsoft Corporation.)

Výstřižky (© 2016 Microsoft Corporation.)

Seznam obrázků

Přílohy