פרוייקט בהנדסת תוכנה

שם התלמיד: דמיטרי קליניצ'נקו

בית ספר: מקיף ג'

עיר: אשדוד

כיתה: יב' 8

שנת הגשה: 2011 תשע"א

שם הפרוייקט: ZombieMadness

שפת כתיבה: C#

מורה: פיימר אנטולי

תוכן עניינים

מבוא

בחרתי את המשחק מפני שרציתי לעשות משחק יריות כלשהוא, בהתחלה המשחק היה אמור להיות משהו אחר אך בזמן שעבדתי על המשחק חשבתי על רעיונות נוספים שלקחו את המשחק לכיוון אחר, לא בחרתי את המשחק מסיבה מסויימת אלא הכל נוצר כתוצאה מהחלטות שונות במהלך הפיתוח

תיאור המשחק

המשחק הנו משחק דו מימדי מזווית מבט מלמעלה, משחקים עם המקלדת והעכבר והמטרה היא להרוג כמה שיותר זומבים לפני שהזומבים יהרגו אותך, בחלק העליון של המסך יש בצד שמאל מד חיים אשר יורד כל פעם כאשר נוגעים בזומבי, ומד טעינת נשק בצד ימין אשר עולה לפי כמות הזמן שלוקח לנשק להיטען. בחלק התחתון של המסך יש מד אשר סופר את כמות הזומבים שנהרגו.

מטרות הפרוייקט

המטרות שלי בתחילת השנה היה להכין משחק יריות מהנה, הרעיון של המשחק בשלבי הפיתוח היה שונה קצת וככל שהתקדמתי יותר הרעיון התפתח לכיוונים אחרים, לבסוף הגעתי למה שהכנתי עכשיו. אני מרוצה מאוד מהתוצר הסופי ולא הפריע לי שהקונספט של המשחק השתנה במקצת במהלך פיתוח המשחק. בנוסף ישנה אפשרות לשחקן יחיד ו2 שחקנים עם תקשורת, אך טרם הספקתי לסיים את התקשורת כתוצאה

מחוסר זמן, אך התקשורת עובדת ויש אפשרות ש2 הדמויות יזוזו, יש עוד כמה דברים שצריך להוסיף על מנת שהתקשורת תעבוד במלואה.

דרישות מערכת

-מערכת הפעלה מסוג Windows Xp/Vista/Win7
-50 מגהבייט פנויים בכונן להתקנת המשחק
-512 זכרון RAM
-כרטיס מסך התומך בDirectX 9.0c ו1.1 Pixel Shader
-Net Framework 3.0.
-Net Xna Framework 3.1.

כלים וסביבת עבודה

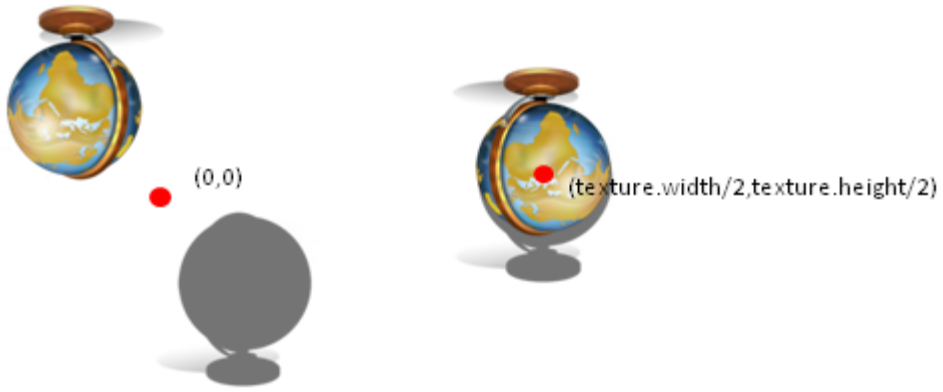
המשחק נכתב בשפת #C
בעזרת טכנולוגיית XNA
Visual Studio 2008-
פיתוח המשחק
GIMP-
עריכת תמונות

בעיות

במהלך פיתוח המשחק נתקלתי בכמה בעיות, בעיה אחת שזכורה לי היא בדיקת התנגשות בין גופים, תחילה השתמשתי בהתנגשות מלבנים פשוטה, אך זה לא היה מספיק מפני שהבדיקה לא מדוייקת, לכן ניסיתי להשתמש בבדיקה של Per

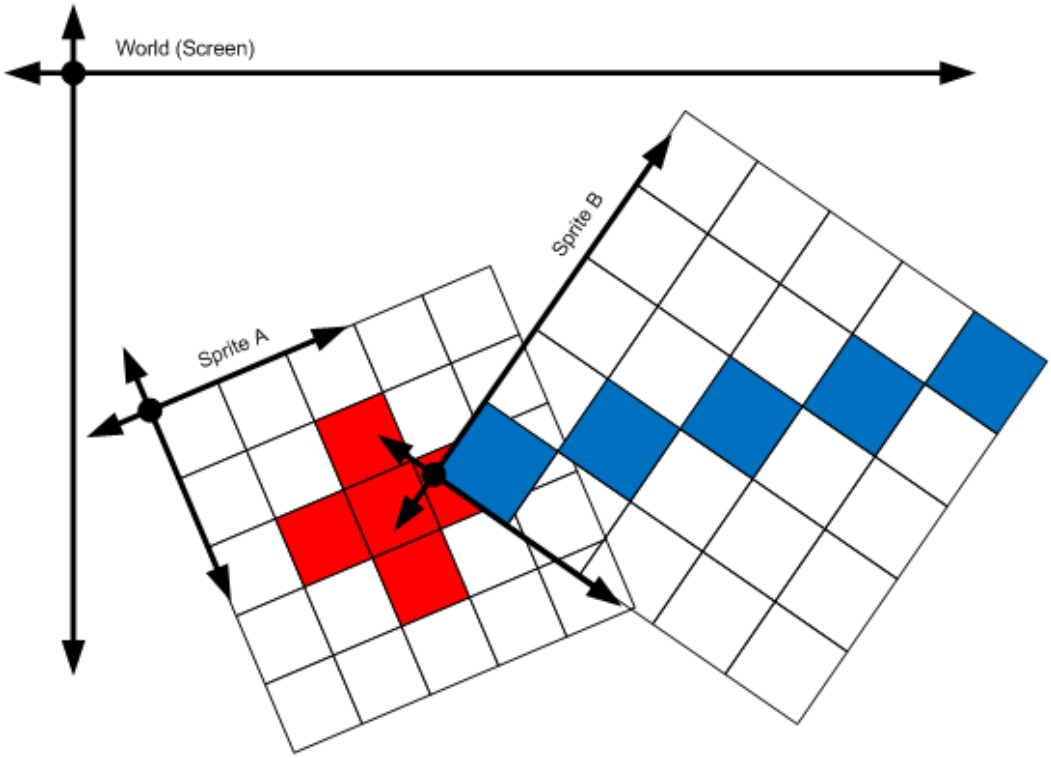
Pixel שמשווה בין 2 התמונות ובודקת אם הפיקסלים נפגשים, הבדיקה מסננת את החלקים השקופים של התמונה ובודקת רק את החלקים הצבעוניים של התמונה, הבעיה הגיעה כאשר היה צורך לבדוק את האובייקטים כאשר הם מסובבים, לבסוף אחרי חיפושים הצלחתי למצוא את הדרך לבצע את הבדיקה על האובייקטים כאשר הם מסובבים.

השיטה עובדת בצורה הבאה:
כאשר מסובבים תמונה בXNA הסיבוב מתבצע על ציר אשר נקבע על פני התמונה, כלומר יש 2 משתנים, נקודת הציר וזווית הסיבוב
להלן תמונה הממחישה סיבוב של 90 מעלות ב2 נקודות ציר שונות:



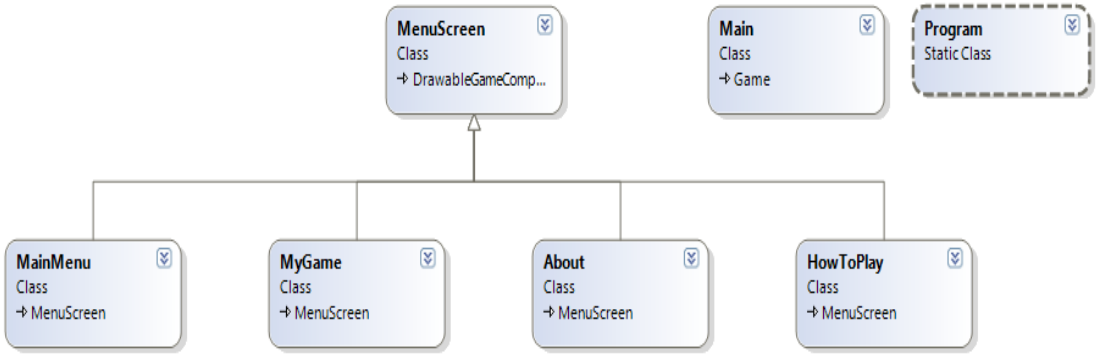
על מנת לא להעמיס על המערכת קודם בודקים התנגשות של מסגרות האובייקטים ורק אז לעבור לבדיקה של התנגשות של Per Pixel אז יש צורך בסיבוב המסגרות של כל אובייקט, אך לא ניתן לסובב מסגרת מלבנית, על מנת לפתור את זה יש צורך בלחלק את המלבן ל2 משולשים ואותם לסובב ביחד בתור מסגרת מלבנית. כל מה שנותר הוא לקחת את הפיקסלים מתוך המסגרות ולבצע את ההשוואה, אך הבעיה היא שהפיקסלים שנלקחים הם לא מהתמונה בלבד, לכן על מנת לפתור את הבעיה הזאת יש צורך לצייר את התמונה(בצורה בלתי נראית)

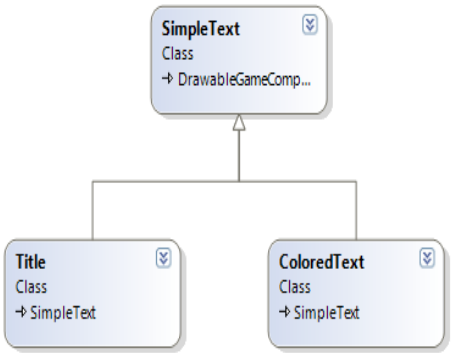
לקחת תצלום מהמסך של רק התמונות כאשר הם מתנגשות,
ללא הרקע ולבצע את ההשוואה.



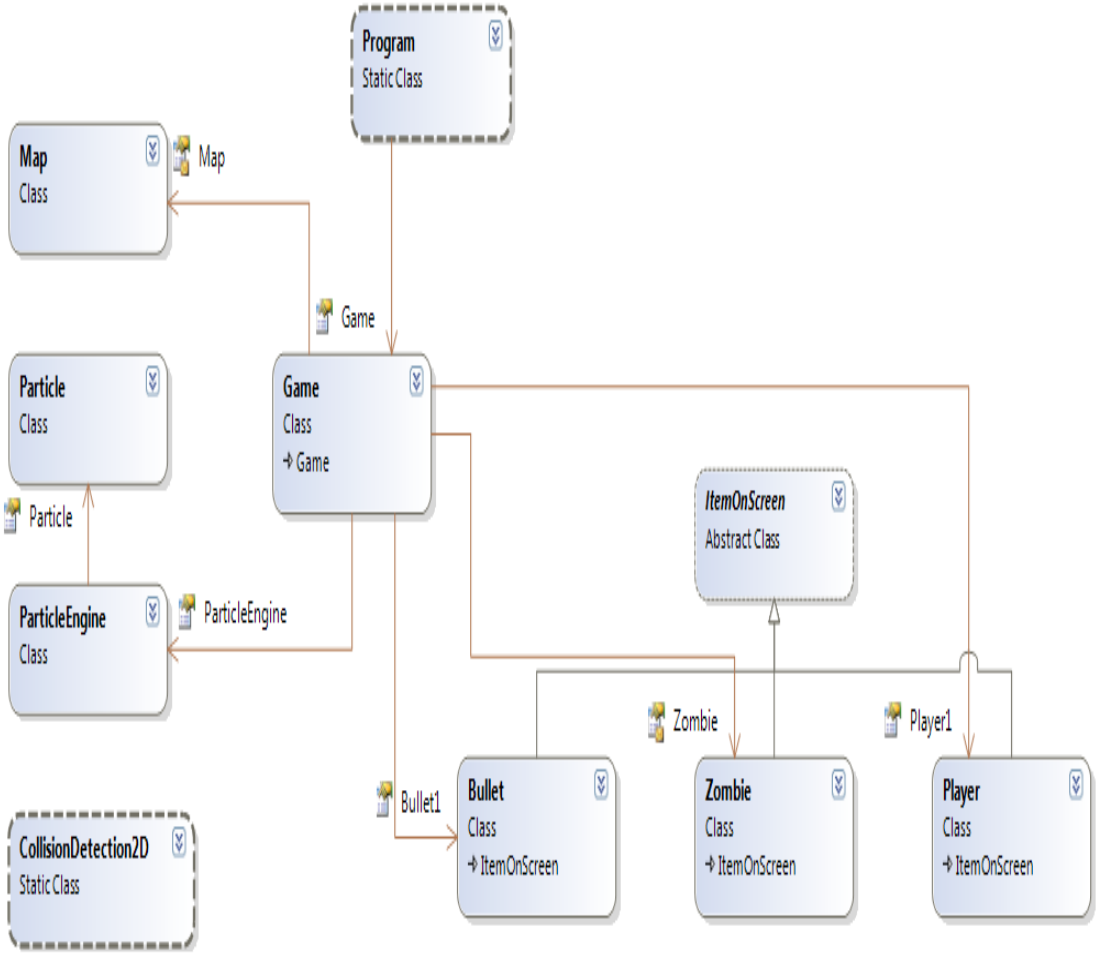
תיאור המחלקות

מחלקות התפריט:





מחלקות המשחק:


Program
Static Class
Map
Class
Map
Game
Particle
Class
Game
Class
Game
Particle
ItemOnScreen
Abstract Class
ParticleEngine
Class
ParticleEngine
Zombie
Player1
Bullet1
Bullet
Class
ItemOnScreen
Zombie
Class
ItemOnScreen
Player
Class
ItemOnScreen
CollisionDetection2D
Static Class

תיאור המחלקות והפעולות:

MenuScreen:

```
/// <summary>
/// This is the father class for the screens which derive from it
/// </summary>
public abstract class MenuScreen : DrawableGameComponent
```

Fields:

```
/// <summary>
/// List of standing messages
/// </summary>
protected List<StandingText> listOfMessages;
/// <summary>
/// Distance between each message
/// </summary>
protected const float DISTANCE = 80;
/// <summary>
/// Current Keyboard state
/// </summary>
protected KeyboardState state;
/// <summary>
/// Last Keyboard state - needed to define one click
/// </summary>
protected KeyboardState lastKeyboardState;
/// <summary>
/// The title of the screen
/// </summary>
protected Title title;
/// <summary>
/// X offset from the position
/// </summary>
protected const int BEGINNING_X_OFFSET = 100;
/// <summary>
/// Offset for the title
/// </summary>
protected Vector2 TITLE_OFFSET = new Vector2(-300, -50);
/// <summary>
/// Offset for each menu item
/// </summary>
protected Vector2 MENU_ITEM_OFFSET = new Vector2(-400, -200);
```

Functions:

Protected MenuScreen() – `Constructor`
Public override void Initialize() - `Initializing the scene`
Public override void Update() - `Updating the scene`
Public void LeaveScreen() - `Leaving the screen and removing the component`
Protected void NextOneWaving() - `Moving the waving forward`
Protected void PreviousOneWaving() - `Moving the waving back`

SimpleText:

```
    /// <summary>
    /// Class needed to draw a menu item with Simple text
    /// </summary>
    public class StandingText : DrawableGameComponent
```

Fields:

```
        /// <summary>
        /// The message
        /// </summary>
        protected string message;
        /// <summary>
        /// The font
        /// </summary>
        protected SpriteFont font;
        /// <summary>
        /// The sprite batch
        /// </summary>
        protected SpriteBatch spriteBatch;
        /// <summary>
        /// The position - center of the screen
        /// </summary>
        protected Vector2 position;
        /// <summary>
        /// true if the menu item is leaving the screen
        /// </summary>
        protected bool isLeaving;
        /// <summary>
        /// true if the menu item is entering the screen
        /// </summary>
        private bool isEntering;
        /// <summary>
        /// Step for the alpha to decrease or increase
        /// </summary>
        public const int ALPHA_OFF_STEP = 5;
        /// <summary>
        /// Step for the position when leaving or entering the screen
        /// </summary>
        public const int POSITION_STEP = 10; // 5 for school 10 for
home
        /// <summary>
        /// The color - the RGB is const but the Alpha changes
        /// </summary>
        protected Color ourColor;
        /// <summary>
        /// Position where the text stops moving
        /// </summary>
        protected float FADE_IN_X_POS;
        /// <summary>
        /// Vector for the menu item offset
        /// </summary>
        protected Vector2 MENU_ITEM_OFFSET = new Vector2(-400, -200);
```

Functions:

Public StandingText() - `Constructor`
Protected override void LoadContent – `Loading the content`
Protected override void Update() – `Updating the scene`
Protected override void Draw() – `Drawing the scene`
Public void LeaveScreen() – `Called when menu item needs to leave the screen`
Public void EnterScreen() – `Called when menu item needs to enter the screen`

Title:

```
/// <summary>
/// Class needed to draw the title
/// </summary>
public class Title : DrawableGameComponent
```

Fields:

```
/// <summary>
/// Position for the tile
/// </summary>
protected Vector2 position;
/// <summary>
/// Text of the title
/// </summary>
protected string text;
/// <summary>
/// Color of the title
/// </summary>
protected Color color;
/// <summary>
/// Sprite batch to draw the title
/// </summary>
protected SpriteBatch spriteBatch;
/// <summary>
/// Font for writing the title
/// </summary>
protected SpriteFont font;
/// <summary>
/// True if the title is leaving the screen
/// </summary>
protected bool isLeaving;
/// <summary>
/// True if the title is entering the screen
/// </summary>
protected bool isEntering;
/// <summary>
/// Step for leaving and entering for the alpha
/// </summary>
protected const int ALPHA_OFF_STEP = 5;
/// <summary>
/// Step for leaving and entering for the position
/// </summary>
```

```
protected const int POSITION_STEP = 5;
/// <summary>
/// The position after which we stop entering
/// </summary>
private readonly float ULTIMATE_POSITION;
```

Functions:

Public Title() – `Constructor`
Public override void LoadContent() – `Loading the content`
Public override void Update() – `Updating the scene`
Public override void Draw() – `Drawing the scene`
Public void EnterScreen() – `Called when the title is entering the screen`
Public void LeaveScreen() – `Called when the title is leaving the screen`

Colored Text:

```
/// <summary>
/// Class needed to draw a colored text
/// derives from our standing text class
/// </summary>
class ColoredText : StandingText
```

Fields:

```
/// <summary>
/// Scale parameter for the text
/// </summary>
protected float scale;
/// <summary>
/// Maximum scale
/// </summary>
protected const float MAX_SCALE = 1.2f;
/// <summary>
/// Minimum scale
/// </summary>
protected const float MIN_SCALE = 1.0f;
/// <summary>
/// True if the text is increasing, false if decreasing
/// </summary>
protected bool isScalingUp = true;
```

Functions:

Public ColoredText() – `Constructor`
Public override Update() – `Updating the coloring`
Public override Draw() – `Drawing the scene`

ZombieMadness Game:

Particle Engine Classes:

Particle:

```
/// <summary>
/// Class that creates a singe particle object
/// </summary>


public class Particle
```

Fields:

```
/// <summary>
/// The texture that will be drawn to represent the particle
/// </summary>
public Texture2D Texture { get; set; }
/// <summary>
/// The current position of the particle
/// </summary>
public Vector2 Position { get; set; }
/// <summary>
/// The speed of the particle at the current instance
/// </summary>
public Vector2 Velocity { get; set; }
/// <summary>
/// The current angle of rotation of the particle
/// </summary>
public float Angle { get; set; }
/// <summary>
/// The speed that the angle is changing
/// </summary>
public float AngularVelocity { get; set; }
/// <summary>
/// The color of the particle
/// </summary>
public Color Color { get; set; }
/// <summary>
/// The size of the particle
/// </summary>
public float Size { get; set; }
/// <summary>
/// The 'time to live' of the particle
/// </summary>
public int TTL { get; set; }
```

Functions:

Public Particle() – `Constructor`
Public Void Update() – `Updated particles time to live and movement`
Public Void Draw() – `Draws the particle`

ParticleEngine:

```
/// <summary>
/// This class manages the particles in the game
/// </summary>
```

```
public class ParticleEngine
```

Fields:

```
/// <summary>
/// Random number generator
/// </summary>
private Random random;
/// <summary>
/// Starting position of the particle emitter
/// </summary>
public Vector2 EmitterLocation { get; set; }
/// <summary>
/// list of particles
/// </summary>
private List<Particle> particles;
/// <summary>
/// Particle texture
/// </summary>
private Texture2D textures;
/// <summary>
/// Bullet object for the emitter location position
/// </summary>
private Bullet bullet;
/// <summary>
/// Game screen size
/// </summary>
private Vector2 screenSize;
/// <summary>
/// Amount of particles
/// </summary>
public int TotalParticles { get; set; }
```

Functions:

Pubic ParticeEngin()e – `Constructor`
Private Particle GenerateNewParticle() – `Generates new particle`
Public void UpdateBullet() – `Updates the bullet object`
Public void UpdateParticlesMovement() – `Moves the particles`
Public int GetParticlesAmount() – `Returns the amount of particles`
Public void Draw() – `Draws the particles`

Bullet

```
/// <summary>
/// This class creates a single bullet object
/// that used by the player
/// this class inherits from the ItemOnScreen base class
/// </summary>
public class Bullet : ItemOnScreen:
```

Fields:

```
/// <summary>
/// Bullet movement speed
/// </summary>
private const int speed = 30;
/// <summary>
/// Random angle for the recoil effect
/// </summary>
private double randAngle;
```

Functions:

Pubic Bullet() – Constructs a new bullet with a random angle to create a recoil effect

CollisionDetection2D:

```
/// <summary>
/// This class manages the collision detection engine
/// </summary>
public static class CollisionDetection2D
```

Functions:

Public static RenderTarget2D() AdditionalRenderTargetForCollision - getting a render target to check collision on from the game.

Public static bool BoundingRectangle() – Receives 2 Lists of 4 Vector2 points and check if the rectangles are bounding by creating the rectangles with 2 triangles and rotating them.

public static bool BoundingRectangleNR() – Simple bounding rectangles check without rotation

public static bool BoundingTriangles() – Receives 2 Lists of 3 Vector2 Points and checks if the triangles are bounding

The per pixel collision function

```
/// <summary>
/// Per pixel collision detection that supports rotation
/// </summary>
/// <param name="Texture1">The first sprite's texture</param>
/// <param name="Texture2">The second sprite's texture</param>
/// <param name="Pos1">The first sprite's position</param>
/// <param name="Pos2">The second sprite's position</param>
/// <param name="Orig1">The first sprite's origin of rotation
(texture reference point)</param>
/// <param name="Orig2">The second sprite's origin of rotation
(texture reference point)</param>
```

```
/// <param name="Rect1">The first sprite's bounding rectangle</param>
/// <param name="Rect2">The second sprite's bounding rectangle</
param>
/// <param name="Theta1">The first sprite's rotation</param>
/// <param name="Theta2">The second sprite's rotation</param>
/// <param name="spriteBatch">The current Spriteatch</param>
/// <returns>True if they overlap/False otherwise</returns>

public static bool PerPixelWR(Texture2D Texture1, Texture2D Texture2,
Vector2 Pos1, Vector2 Pos2, Vector2 Orig1, Vector2 Orig2,
List<Vector2> Rect1, List<Vector2> Rect2,
float Theta1, float Theta2,SpriteBatch spriteBatch)
```

Map:

```
    /// <summary>
    /// This class manages and draws the map
    /// </summary>
```

Fields:

```
        /// <summary>
        /// Map X*Y
        /// </summary>
        private const int mapSize = 50;
        /// <summary>
        /// Map tile X*Y
        /// </summary>
        private const int tileSize = 50;
        /// <summary>
        /// Map 2d array tiles
        /// </summary>
        private int[,] map;
        /// <summary>
        /// Tile textures list
        /// </summary>
        private List<Texture2D> tiles = new List<Texture2D>();
```

Functions:

Pubic Map() – `Constructor`
Private void ReadMapFile() – `Read the map from file`
Public void Draw() – `Draws the ma`

ItemOnScreen

```
    /// <summary>
    /// This is a base class for an active item
    /// that appear on screen
    /// </summary>
    public abstract class ItemOnScreen:
```

Fields:

```
/// <summary>
/// Item's X
/// </summary>
protected float x;
/// <summary>
/// Item's Y
/// </summary>
protected float y;
/// <summary>
/// Item's angle
/// </summary>
protected double angle;
/// <summary>
/// Item's width
/// </summary>
protected int width;
/// <summary>
/// Item's height
/// </summary>
protected int height;
/// <summary>
/// Item's image
/// </summary>
protected Texture2D image;
/// <summary>
/// Item's rotation origin point
/// </summary>
protected Vector2 origin;
```

Functions:

Public List<Vector2> RectanglePoints() – `Returns a list of vector2 points for the collision detection`

Public static Vector2 RotatePoint() – `Used by the previous function in order to rotate the rectangle`

Public void Draw() – `Draws the item`

Player:

```
/// <summary>
/// This is the main player class
/// It inherits the ItemOnScreen Base Class
/// </summary>
public class Player : ItemOnScreen
```

Fields:

```
/// <summary>
/// Player's health
```

```
/// </summary>
private int health;
/// <summary>
/// Player's current ammo amount
/// </summary>
private int ammo;
/// <summary>
/// Bullet counter
/// </summary>
private int currentBullet = 0;
/// <summary>
/// Player's walking speed
/// </summary>
private int speed = 5;
/// <summary>
/// Is player alive
/// </summary>
private bool alive;
/// <summary>
/// Bullet's diggresion angle for the recoil
/// </summary>
private double digressionAngle = 10;
/// <summary>
/// Bullets array
/// </summary>
private Bullet[] Magazine = new Bullet[30];
```

Functions:

Public Player() – `Constructor`
Public void Shoot() – `Shoots a bullet`
Public void DecreaseAmmo() – `Decreases total ammo amount when reloading`
public void DecreaseHealth() – `Decreases total health when player receives damage`

Zombie:

```
/// <summary>
/// This is the zombie class
/// it inherits from the ItemOnScreen base class
/// </summary>
```

```
class Zombie:ItemOnScreen
```

Fields:

```
/// <summary>
/// The zombie's total health
/// </summary>
private int health;
/// <summary>
/// The zombie's walking speed
/// </summary>
private int speed;
/// <summary>
/// boolean to check if the zombie is alive
/// </summary>
private bool alive;
/// <summary>
/// Player to attack
/// </summary>
private Player player;
```

Functions:

Public Zombie() – `Constructor`
Public void WalkToPlayer() – `Walking towards the player`
Public void DecreaseHealth() – `Decreases the total health amount of the zombie when a bullet hits it`
public void StartPoint() – `Sets the starting point where the zombie will start walking towards the player from.`

Game:

```
/// <summary>
/// This is the main game class
/// </summary>
```

```
public class Game : Microsoft.Xna.Framework.Game
```

Fields:

```
        private Int64 ShootDelay = 0, ReloadDelay = 0, dAngleDelay =
0, SpawnDelay = 0, DamageDelay = 0, PauseDelay = 0;
        private Texture2D Blank, Cursor, ReloadBar, Bullet,particle;
        private Map map;
        private SpriteFont font;
        private Vector2 ScreenSize;
        private ParticleEngine particleEngine;
        private Player Player;
        private bool shoot = false, Reloading = false;
        private bool paused = false;
        public static ContentManager content;

        private SoundEffect GunShot;
        private SoundEffect GunReload;
        private SoundEffect ZombieDeath;

        private List<Zombie> Zombies = new List<Zombie>();
        private List<Zombie> ZombieBodies = new List<Zombie>();

        int SpawnCount = 0, BodiesCount = 0;
```

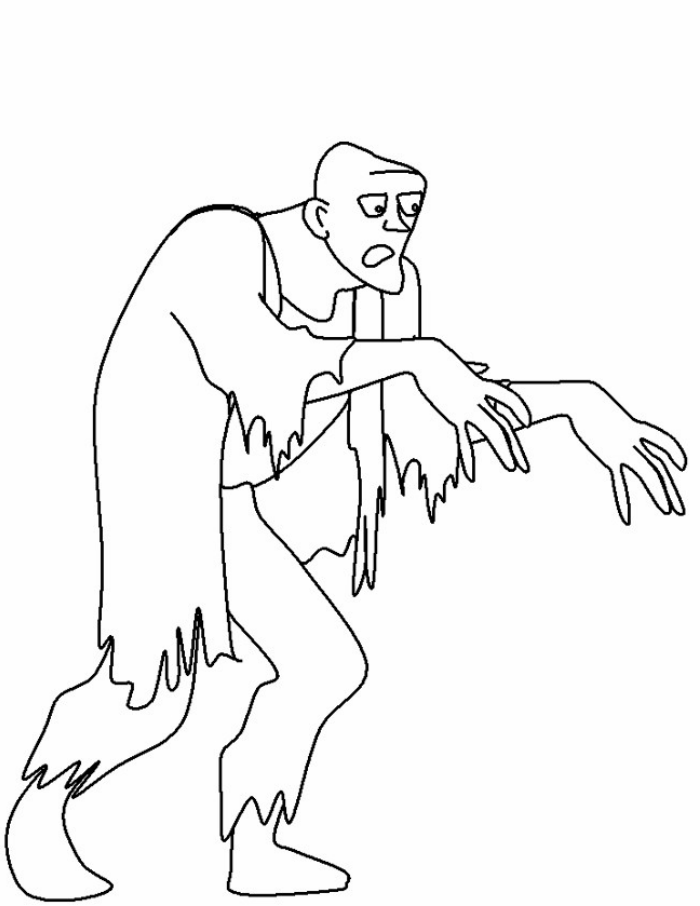
Functions:

public Game() – Constructor
private void ProcessKeyboard() – Processes the key that were pressed
public void SpawnZombie() – Spawns the zombies
private void DrawScenery() – Draws the surroundings
private void DrawPlayer() – Draws the player
private void DrawZombies() – Draws the zombies
private void DrawZombieBodies() – Draws the zombie bodies
private void DrawBullets() – Draws the bullets
private void DrawPlayerInfo() – Draws the player's info i.e. health
private void DrawPauseMenu() – Draws the pause menu options
private void Recoil() – Activates the gun recoil when shooting
private void Reload() – Reloads the weapon
public void CheckBulletCollision() – Checks the bullet collision with the zombies
public void CheckZombieCollision() – Checks the zombie collision with the player
public void SetUpConnectForm() – Sets up the connection form
public void HostClicked() – Called if the player chose to be the host
public void ClientClicked() – Called if the player chose to be the client
public void UpdateNetwork() – Updates the network status and sending messages to the peer
public void HandleNetworkingMessage() – Receives the peer's message and handles it

הצעות לשיפור

-להוסיף שלבים
-להוסיף אפשרות למשחק לכמה שחקנים

-להוסיף מצבי משחק שונים
-להוסיף דמויות שונות
-להוסיף אנימציות



<u>מדריך למשתמש</u>

תפריט המשחק

מסך המשחק, יש לתקוף את הזומבים אשר באים

מצב השהיית המשחק

בתוך המשחק,השחק פוגע בזומבים

השחקן נפסל, יש לבחור אופציה להמשך


15
The zombies got you!, you have managed to kill 13 Zombies!
Press Q to quit to menu
Press R to play again
Kills:13

ביבליוגרפיה

Riemers.net-אתר המלמד XNA וDirectX למתחילים

Google.com-כל בעיה אשר נתקלתי בה נעזרתי בגוגל למציאת הפתרון