**Caritas Institute of Higher Education**

**2D Computer Game using Microsoft Visual Studio**

Nemuel I. Serneo s2070398

Individual Written Report

Dr. Horence Chan

# Table of contents

1. Abstract
2. Introduction
3. Project Description
4. Algorithm Design
5. User Manual

# Abstract

This report aims to go in depth on the project descriptions, the design of the algorithm and the user manual.

# Introduction

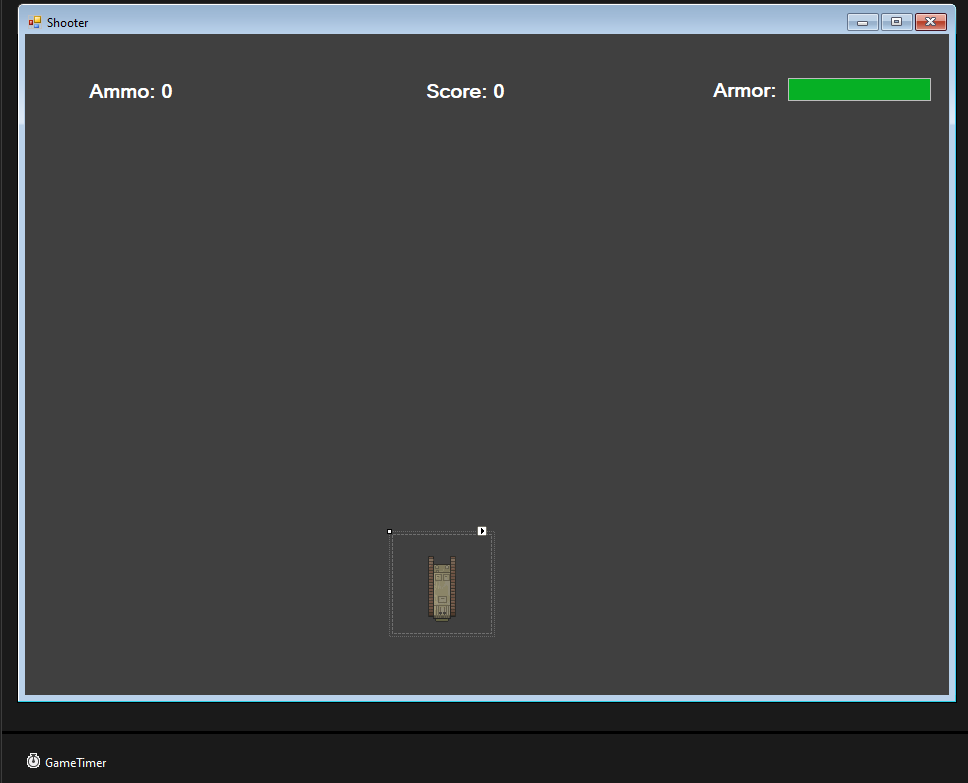
For our project requirement, it was required to design and build a digital entertainment system using C#. As a result, it is decided that Computer Games will be the main focus of our project to showcase the knowledge and design ability learnt throughout the course.

**Project Description**

The theme of my digital entertainment system is a 2D top down shooter game. The player is person inside a “tank” and the enemies are “enemy tanks”. The mission of the player is to destroy the enemy tanks that spawn randomly from anywhere and target you. The player is initially set with 10 ammo bullets. When the player runs out of bullets, ammo crates spawn randomly and the player can obtain them. When the player is touched by the enem tank, their armor is reduced.  
  
In the top panel of the windows form, it consists of a system that counts Ammo, Score of shot down enemies, and Armor durability.

**Algorithm/Code Design**

**Form1.cs[Design]**



**Bulley.cs**

**Bullet Class**

namespace DEProgramming\_Final\_Project // Defines a namespace for the project.

{

internal class Bullet // Defines an internal class named Bullet.

{

public string direction; // Public field that stores the direction of the bullet.

public int bulletLeft; // Public field that stores the left position of the bullet.

public int bulletTop; // Public field that stores the top position of the bullet.

private int speed = 20; // Private field that stores the speed of the bullet.

private PictureBox bullet = new PictureBox(); // Private field that stores the PictureBox object representing the bullet.

private Timer bulletTimer = new Timer(); // Private field that stores the Timer object for controlling the bullet's movement.

public void MakeBullet(Form form) // Public method that creates the bullet on the given form.

{

// The following lines set the properties of the bullet PictureBox and add it to the form's controls.

bullet.BackColor = Color.White;

bullet.Size = new Size(5,5);

bullet.Tag = "bullet";

bullet.Left = bulletLeft;

bullet.Top = bulletTop;

bullet.BringToFront();

form.Controls.Add(bullet);

// The following lines set the properties of the bulletTimer and start it.

bulletTimer.Interval = speed;

bulletTimer.Tick += new EventHandler(BulletTimerEvent);

bulletTimer.Start();

}

private void BulletTimerEvent(object sender, EventArgs e) // Private method that is called whenever the bulletTimer ticks.

{

// The following lines move the bullet in the direction specified by the direction field.

if (direction == "left") { bullet.Left -= speed; }

if (direction == "right") { bullet.Left += speed; }

if (direction == "up") { bullet.Top -= speed; }

if (direction == "down") { bullet.Top += speed; }

// The following lines stop and dispose of the bullet and bulletTimer if the bullet goes out of bounds.

if (bullet.Left < 10 || bullet.Left > 860 || bullet.Top < 10 || bullet.Top > 600)

{

bulletTimer.Stop();

bulletTimer.Dispose();

bullet.Dispose();

bulletTimer = null;

bullet = null;

}

}

}

}

# Form1.cs

**Initiation of values**

public partial class Form1 : Form

{

bool goLeft, goRight, goUp, goDown, gameOver; // Boolean variables for controlling character movement and game status.

string facing = "up"; // String variable to keep track of the direction the player is facing.

int playerHealth = 100; // Integer variable to keep track of the player's health.

int speed = 10; // Integer variable for the speed of something (like a player or enemy).

int ammo = 10; // Integer variable for the amount of ammunition a player has

int enemySpeed = 3; // Integer variable for the speed of the enemy

int score; // Integer variable for the player's score.

Random randNum = new Random(); // Creates a new instance of the Random class for generating random numbers.

List<PictureBox> enemyList = new List<PictureBox>(); // A list to hold PictureBox objects, presumably for enemies.

}

**Timer**

private void GameTimer\_Tick(object sender, EventArgs e)

{

if(playerHealth > 1) // If the player's health is greater than 1...

{

HealthBar.Value = playerHealth; // ...update the health bar to reflect the player's current health.

}

else // If the player's health is not greater than 1...

{

gameOver = true; // ...set the game over flag to true...

player.Image = Properties.Resources.dead; // ...change the player's image to a "dead" image...

GameTimer.Stop(); // ...and stop the game timer.

}

txtAmmo.Text = "Ammo: " + ammo; // Update the ammo text to reflect the player's current ammo count.

txtScore.Text = "Kills: " + score; // Update the score text to reflect the player's current score.

// The following four if statements control the player's movement based on the direction flags and the player's current position.

if (goLeft == true && player.Left > 0)

{

player.Left -= speed;

}

if (goRight == true && player.Left + player.Width < this.ClientSize.Width)

{

player.Left += speed;

}

if (goUp == true && player.Top > 80)

{

player.Top -= speed;

}

if (goDown == true && player.Top + player.Height < this.ClientSize.Height)

{

player.Top += speed;

}

// The following foreach loop checks every control in the form's Controls collection.

foreach(Control x in this.Controls)

{

// If the control is a PictureBox tagged as "ammo" and the player intersects with it...

if (x is PictureBox && (string)x.Tag == "ammo")

{

if (player.Bounds.IntersectsWith(x.Bounds))

{

this.Controls.Remove(x); // ...remove the ammo PictureBox from the form's Controls collection...

((PictureBox)x).Dispose(); // ...dispose of the ammo PictureBox...

ammo += 5; // ...and increase the player's ammo count by 5.

}

}

// If the control is a PictureBox tagged as "enemy"...

if (x is PictureBox && (string)x.Tag == "enemy")

{

// If the player intersects with the enemy...

if (player.Bounds.IntersectsWith(x.Bounds))

{

playerHealth -= 1; // ...decrease the player's health by 1.

}

// The following four if statements move the enemy towards the player and change the enemy's image based on its direction.

if (x.Left > player.Left)

{

x.Left -= enemySpeed;

((PictureBox)x).Image = Properties.Resources.zleft;

}

if (x.Left < player.Left)

{

x.Left += enemySpeed;

((PictureBox)x).Image = Properties.Resources.zright;

}

if (x.Top > player.Top)

{

x.Top -= enemySpeed;

((PictureBox)x).Image = Properties.Resources.ztop;

}

if (x.Top < player.Top)

{

x.Top += enemySpeed;

((PictureBox)x).Image = Properties.Resources.zdown;

}

}

// The following foreach loop checks every control in the form's Controls collection.

foreach (Control j in this.Controls)

{

// If the control is a PictureBox tagged as "bullet" and it intersects with an enemy...

if (j is PictureBox && (string)j.Tag == "bullet" && x is PictureBox && (string)x.Tag == "enemy")

{

if (x.Bounds.IntersectsWith(j.Bounds))

{

score++; // ...increase the player's score by 1...

this.Controls.Remove(j); // ...remove the bullet PictureBox from the form's Controls collection...

((PictureBox)j).Dispose(); // ...dispose of the bullet PictureBox...

this.Controls.Remove(x); // ...remove the enemy PictureBox from the form's Controls collection...

((PictureBox)x).Dispose(); // ...dispose of the enemy PictureBox...

enemyList.Remove(((PictureBox)x)); // ...remove the enemy PictureBox from the enemy list...

MakeEnemies(); // ...and call the MakeEnemies method to make a new enemy.

}

}

}

}

}

# Controls for when key is pressed

private void KeyIsDown(object sender, KeyEventArgs e) // This method is called when a key is pressed.

{

if (gameOver == true) // If the game is over...

{

return; // ...exit the method early.

}

if (e.KeyCode == Keys.Left) // If the left arrow key is pressed...

{

goLeft = true; // ...set the goLeft flag to true...

facing = "left"; // ...set the facing direction to "left"...

player.Image = Properties.Resources.left; // ...and change the player's image to a "left" image.

}

if (e.KeyCode == Keys.Right) // If the right arrow key is pressed...

{

goRight = true; // ...set the goRight flag to true...

facing = "right"; // ...set the facing direction to "right"...

player.Image = Properties.Resources.right; // ...and change the player's image to a "right" image.

}

if (e.KeyCode == Keys.Up) // If the up arrow key is pressed...

{

goUp = true; // ...set the goUp flag to true...

facing = "up"; // ...set the facing direction to "up"...

player.Image = Properties.Resources.up; // ...and change the player's image to an "up" image.

}

if (e.KeyCode == Keys.Down) // If the down arrow key is pressed...

{

goDown = true; // ...set the goDown flag to true...

facing = "down"; // ...set the facing direction to "down"...

player.Image = Properties.Resources.down; // ...and change the player's image to a "down" image.

}

}

**Controls for when key is released**

private void KeyIsUp(object sender, KeyEventArgs e) // This method is called when a key is released.

{

if (e.KeyCode == Keys.Left) // If the left arrow key is released...

{

goLeft = false; // ...set the goLeft flag to false.

}

if (e.KeyCode == Keys.Right) // If the right arrow key is released...

{

goRight = false; // ...set the goRight flag to false.

}

if (e.KeyCode == Keys.Up) // If the up arrow key is released...

{

goUp = false; // ...set the goUp flag to false.

}

if (e.KeyCode == Keys.Down) // If the down arrow key is released...

{

goDown = false; // ...set the goDown flag to false.

}

if(e.KeyCode == Keys.Space && ammo > 0 && gameOver == false) // If the space bar is released, there's ammo left, and the game isn't over...

{

ammo--; // ...decrement the ammo count by 1...

ShootBullet(facing); // ...call the ShootBullet method with the current facing direction...

if (ammo < 1) // If there's no ammo left...

{

DropAmmo(); // ...call the DropAmmo method.

}

}

if(e.KeyCode == Keys.Enter && gameOver == true) // If the enter key is released and the game is over...

{

RestartGame(); // ...call the RestartGame method.

}

}

**Bullet Design**

private void ShootBullet(string direction) // This method is called when a bullet is to be shot. The direction of the bullet is passed as an argument.

{

Bullet shootBullet = new Bullet(); // Create a new instance of the Bullet class.

shootBullet.direction = direction; // Set the bullet's direction.

// Set the starting position of the bullet. It's set to the middle of the player character.

shootBullet.bulletLeft = player.Left + (player.Width / 2);

shootBullet.bulletTop = player.Top + (player.Height / 2);

shootBullet.MakeBullet(this); // Call the MakeBullet method of the Bullet class, passing the current form as an argument.

}

**Creation of Enemies**

private void MakeEnemies() // This method is called to create a new enemy.

{

PictureBox enemy = new PictureBox(); // Create a new PictureBox instance for the enemy.

enemy.Tag = "enemy"; // Set the Tag property of the enemy PictureBox to "enemy".

enemy.Image = Properties.Resources.zdown; // Set the Image property of the enemy PictureBox to an image resource named "zdown".

// Set the Left and Top properties of the enemy PictureBox to random values within specified ranges.

enemy.Left = randNum.Next(0, 900);

enemy.Top = randNum.Next(0, 800);

enemy.SizeMode = PictureBoxSizeMode.AutoSize; // Set the SizeMode property of the enemy PictureBox to AutoSize.

enemyList.Add(enemy); // Add the enemy PictureBox to the enemyList.

this.Controls.Add(enemy); // Add the enemy PictureBox to the form's Controls collection.

player.BringToFront(); // Bring the player control to the front of the z-order.

}

**Ammo Drop**

private void DropAmmo() // This method is called to create a new ammo pickup.

{

PictureBox ammo = new PictureBox(); // Create a new PictureBox instance for the ammo.

ammo.Image = Properties.Resources.ammo\_Image; // Set the Image property of the ammo PictureBox to an image resource named "ammo\_Image".

ammo.SizeMode = PictureBoxSizeMode.AutoSize; // Set the SizeMode property of the ammo PictureBox to AutoSize.

// Set the Left and Top properties of the ammo PictureBox to random values within specified ranges.

ammo.Left = randNum.Next(10, this.ClientSize.Width - ammo.Width);

ammo.Top = randNum.Next(60, this.ClientSize.Height - ammo.Height);

ammo.Tag = "ammo"; // Set the Tag property of the ammo PictureBox to "ammo".

this.Controls.Add(ammo); // Add the ammo PictureBox to the form's Controls collection.

ammo.BringToFront(); // Bring the ammo control to the front of the z-order.

player.BringToFront(); // Bring the player control to the front of the z-order.

}

**Restart Functionality**

private void RestartGame() // This method is called to restart the game.

{

player.Image = Properties.Resources.up; // Set the player's image to an "up" image.

// The following foreach loop removes all the enemies from the form's Controls collection.

foreach (PictureBox i in enemyList)

{

this.Controls.Remove(i);

}

enemyList.Clear(); // Clear the enemy list.

// The following for loop creates three new enemies.

for (int i = 0; i < 3; i++)

{

MakeEnemies();

}

// Reset the movement flags and the game over flag.

goUp = false;

goDown = false;

goLeft = false;

goRight = false;

gameOver = false;

// Reset the player's health, the score, and the ammo count.

playerHealth = 100;

score = 0;

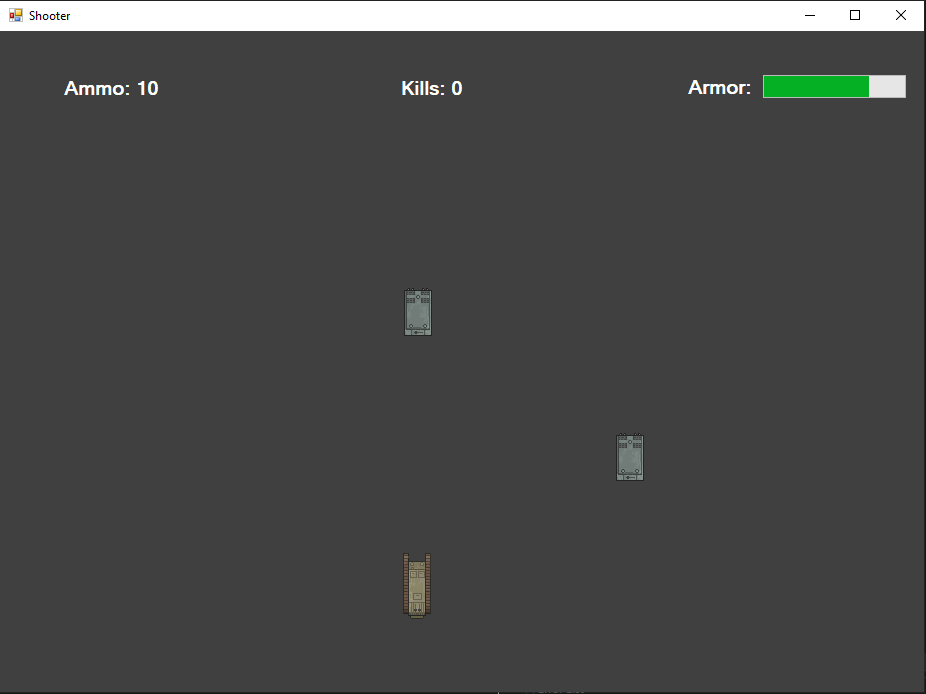
ammo = 10;

GameTimer.Start(); // Start the game timer.

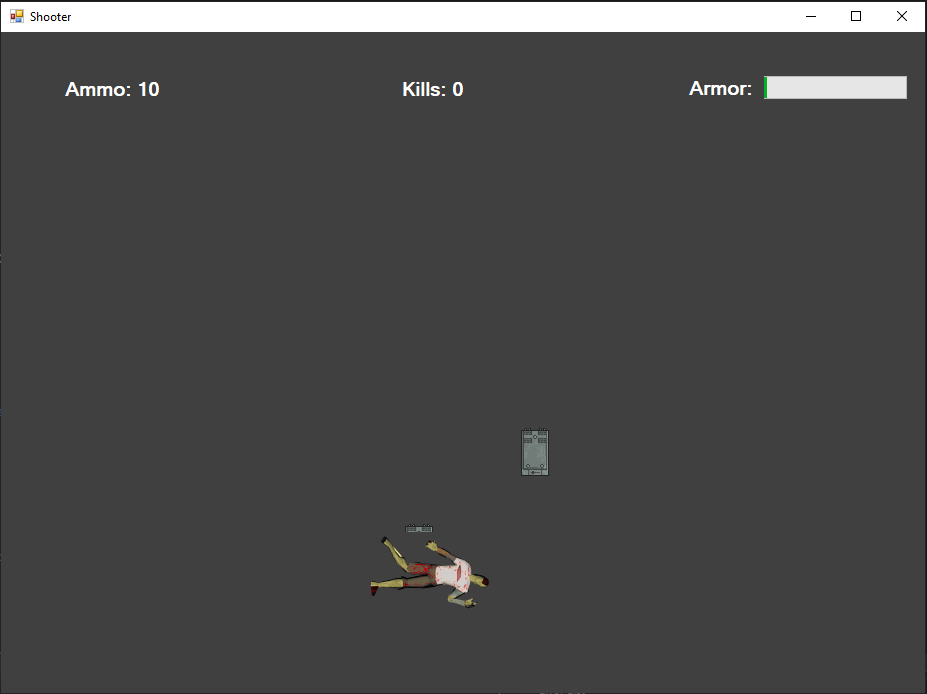
}

**User Manual**

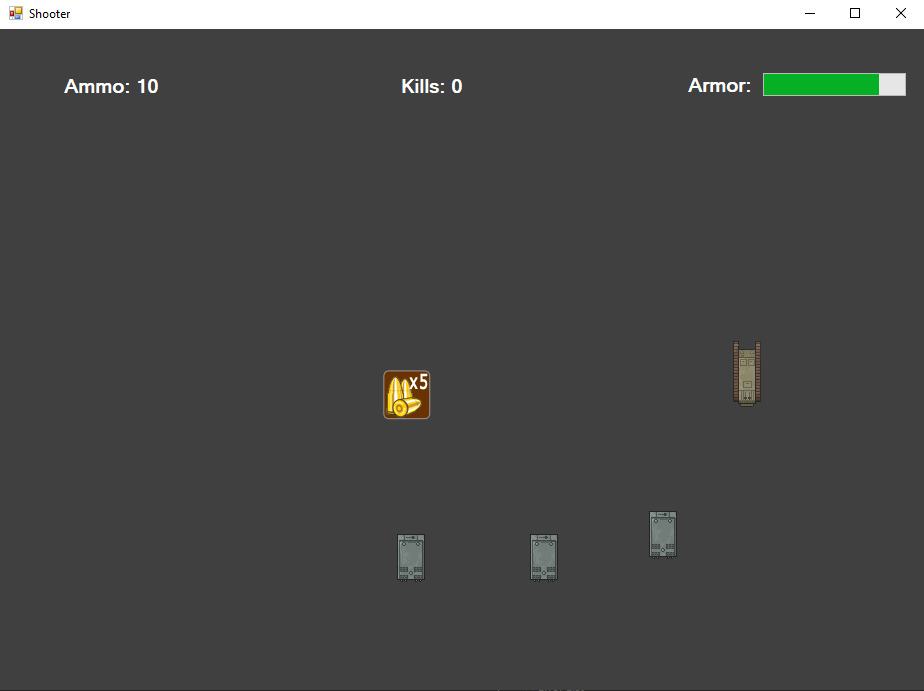
Playable State(Use WASD to move, use SPACE to shoot, enemies will spawn everywhere)



Game Over State(Click Enter to Restart Game)



Picking up Ammo(These will randomly spawn)



Shooting Enemies(use SPACE to shoot enemies)

