**Algorithm Document**

1. **Bullet trajectory under influence of air resistance coefficient**
2. **Team members**

Molotkovski Roman : [megaclaff@gmail.com](mailto:megaclaff@gmail.com) : 0548082996

Mecher Timor : [timam82@gmail.com](mailto:timam82@gmail.com) 0544361502

Melnik Alexander : [pykni5@walla.co.il](mailto:pykni5@walla.co.il) 0508107571

Menachem Gabriel : [s7362454@gmail.com](mailto:s7362454@gmail.com) : 0502255623

1. **Project vision**

Our vision in this Project is to give companies and manufactures the ability to simulate the trajectory of different bullet types under various circumstances in order to get an accurate approximation of the bullet’s trajectory in real life.

1. **Project Mission**

This project aims to enable users to predict a bullet’s trajectory, given the required parameters like bullet’s mass, bullet’s resistance coefficient, bullet’s muzzle velocity, shooting direction, wind direction and wind speed, target distance and target radius. This program may have many uses both in the military and in the private sector.

1. **Stakeholders**

**Weapon manufacturers** – this tool may be useful for developing and testing new weaponry.

**Gun enthusiasts** may find this tool useful for testing their guns in different conditions without having to waste ammunition.

**Military** – the military may find this tool useful both for testing and for military operations.

**Pacifists** and other groups who are against the development of Israeli weaponry may condemn projects like this.

**Sport field** can use this program to train different kind of athletes of shooting area.

1. **Requirements**

**Functional requirements:**

* The user is required to enter all parameters needed:
  + The user enters the bullet’s air resistance coefficient.
  + The user inputs bullet’s muzzle velocity\*.
  + The user inputs shooting direction:
    - The user inputs horizontal degree\*\*.
    - The user inputs vertical degree\*\*.
* Wind direction and speed are chosen randomly from a given list\*\*\*.
* The program shows a spreadsheet that consists of time, bullet’s location, bullet’s velocity.
* The program shows a picture of the target and the bullet’s hit mark on the target.

\* The user must enter a positive value, if an invalid value is entered the user will have to enter a new one.

\*\* The user can deviate up to 10 degrees from target, if an invalid value is entered the user will have to enter a new one.

\*\*\* The wind direction is on the X axis (left or right direction).

**Non - Functional requirements:**

**PR - Performance Requirements.**

Capacity is: 540kb

Reaction time: ~0.03sec

**QA -** **Quality Attributes**

Reliability - action smoothly over time

Availability – service will provide when information are given.

Safety - keeping life and health, physical and mental operators and users.

Security - Protecting the system components and the information she deals. (Encrypt files).

Maintenance - the ability to easily edit product changes and repairs

Useful - user product contribution and achievement in fulfilling his

Goals.

**HC - Hardware Constraint**

Supported operating systems

* Windows 7 SP1 (x86 and x64)
* Windows 8 (x86 and x64)
* Windows Server 2008 R2 SP1 (x64)
* Windows Server 2012 (x64)

Supported architectures

* 32-bit (x86)
* 64-bit (x64)

Hardware requirements

* 1.6 GHz or faster processor
* 1 GB of RAM (1.5 GB if running on a virtual machine)
* 10 GB of available hard disk space
* 600 MB of available hard disk space (language pack)
* 5400 RPM hard drive
* DirectX 9-capable video card running at 1024 x 768 or higher display resolution

1. **Project software Design**



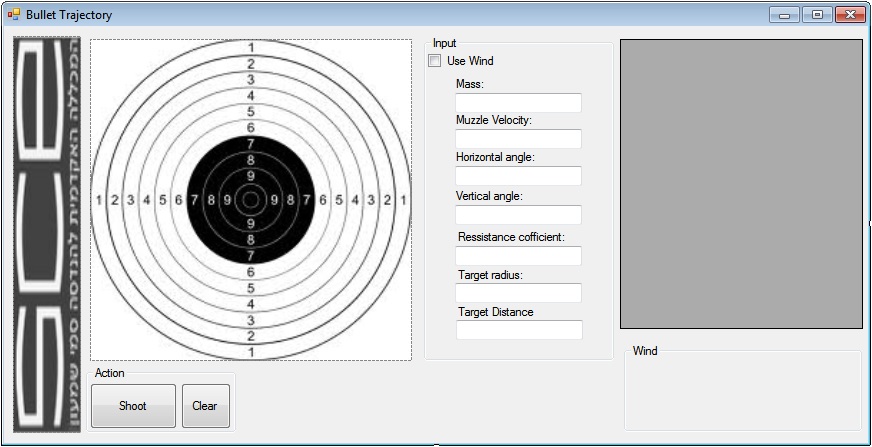
Class diagram:



For a description of each class its methods please look in the comments of the program’s code in section 12.

1. **Project structure including interfaces**

GUI:



The main form consists of input fields which include text boxes and a check box. One the user presses “Shoot” a DataGridView table is shown that describes the bullet’s location and velocity from to hit time and a crosshair appears on the PictureBox that shows where the bullet hit the target. The user is also informed about wind’s speed and direction and also about the distance between the bullet’s hit location and target’s center. The button “Clear” clears all the input fields, the DataGridView table, the crosshair, and the labels.

Note: Positive value in Horizontal angle means angle in right direction while a negative value means left. Positive value in Vertical angle means angle in up direction while a negative value means down.

1. **Software Language**

The is written C# using visual studio 2010.

UML’s provided.

The decision to develop the application in C# was made because it is a popular language among program developers which will make it easy for other developers to modify the application. It is also an easy platform to develop GUI for.

**MC - Management Constraint**

Time table – do to 15.10.2013

1. **Test Plan**

* Input invalid parameters
  + All values must be positive beside the angels.
  + Angel peeks are between +10 to -10.
* Testing the application on different operating systems.
  + The Program has been tested in windows7 and windows8 it wasn’t been tested in other operation systems.
* Black and white box testing.

1. **Maintenance**

The software will be maintained and updated regularly up to a year after its release.

Updates may include bug fixes or new features requested by clients.

1. **Algorithms appearing in the software + discussion**

Progrem code:

Cordinate.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// Cordinates - represents object's location in a 3D space.

/// </summary>

class Cordinates

{

private double x;

private double y;

private double z;

/// <summary>

/// Cordinates constractor

/// </summary>

/// <param name="X">get x cordinates</param>

/// <param name="Y">get y cordinates</param>

/// <param name="Z">get z cordinates</param>

public Cordinates(double X = 0, double Y = 0, double Z = 0)

{

this.x = X;

this.y = Y;

this.z = Z;

}

/// <summary>

/// Geter and seters for x

/// </summary>

public double X

{

get { return this.x; }

set { this.x = value; }

}

/// <summary>

/// Geter and seters for y

/// </summary>

public double Y

{

get { return this.y; }

set { this.y = value; }

}

/// <summary>

/// Geter and seters for z

/// </summary>

public double Z

{

get { return this.z; }

set { this.z = value; }

}

}

}

VelocityVector.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// Velocity vector - 3D vector which consists of velocites on the X, Y and Z axis.

/// </summary>

class VelocityVector

{

private double xVelocity;

private double yVelocity;

private double zVelocity;

/// <summary>

/// Velocity vector constractor

/// </summary>

/// <param name="xVelocity">x velocity</param>

/// <param name="yVelocity">y velocity</param>

/// <param name="zVelocity">z velocity</param>

public VelocityVector(double xVelocity = 0, double yVelocity = 0, double zVelocity = 0)

{

this.xVelocity = xVelocity;

this.yVelocity = yVelocity;

this.zVelocity = zVelocity;

}

/// <summary>

/// Defines the vector by xAngle, yAngle, and vector magnitude.

/// </summary>

/// <param name="xAngle">Vector's angle on the X axis.</param>

/// <param name="yAngle">Vector's angle on the Y axis.</param>

/// <param name="initVelocity">Vector's magnitude.</param>

public void defineByAngles(double xAngle, double yAngle , double initVelocity )

{

xAngle = toRadians(xAngle);

yAngle = toRadians(yAngle);

this.xVelocity = ((initVelocity \* Math.Tan(xAngle)) / Math.Sqrt(1 + Math.Pow(Math.Tan(xAngle), 2) + Math.Pow( Math.Tan(yAngle), 2)));

this.yVelocity = ((initVelocity \* Math.Tan(yAngle)) / Math.Sqrt(1 + Math.Pow(Math.Tan(xAngle), 2) + Math.Pow(Math.Tan(yAngle), 2)));

this.zVelocity = (initVelocity / Math.Sqrt(1 + Math.Pow(Math.Tan(xAngle), 2) + Math.Pow(Math.Tan(yAngle), 2)));

}

/// <summary>

/// Get angle in degrees and returns the radians.

/// </summary>

/// <param name="angle">the angle in degrees</param>

/// <returns>angle in radians</returns>

private double toRadians(double angle)

{

return (Math.PI/180)\* angle;

}

/// <summary>

/// Geter and seter for x velocity

/// </summary>

public double XVelocity

{

get { return this.xVelocity; }

set { this.xVelocity = value;}

}

/// <summary>

/// Geter and seter for y velocity

/// </summary>

public double YVelocity

{

get { return this.yVelocity; }

set { this.yVelocity = value; }

}

/// <summary>

/// Geter and seter for z velocity

/// </summary>

public double ZVelocity

{

get { return this.zVelocity; }

set { this.zVelocity = value; }

}

/// <summary>

/// Returns vector's magnitude.

/// </summary>

/// <returns></returns>

public double getVelocity()

{

return Math.Sqrt(Math.Pow(xVelocity, 2) + Math.Pow(yVelocity, 2) + Math.Pow(zVelocity, 2));

}

}

}

Calculate.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// Calculate is used in Bullet.CalcByDirection method as a parameter.

/// Gives the option to choose whether we want to calculate bullet's location or velocity in

/// a given direction and time.

/// </summary>

enum Calculate { LOCATION, VELOCITY }

}

Direction.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// Direction is used in Bullet.CalcByDirection method as a parameter.

/// Gives the option to choose the direction in which we want to calculate bullet's location / velocity

/// in a given time.

/// </summary>

enum Direction { X, Y, Z }

}

Wind.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// A wind can influence the bullet's trajectory. The wind can be either in left or right direction.

/// A positive value means right and and a negative one means left. Wind's velocity and direction

/// is generated randomly.

/// </summary>

class Wind

{

private double velocity;//Measured in meters per second.

/// <summary>

/// Constructor

/// A random velocity is genetated between -10 to 10 m/s

/// </summary>

public Wind()

{

Random rnd = new Random();

this.velocity = rnd.Next(-10, 10);

}

/// <summary>

/// geter and seter for velocity

/// </summary>

public double Velocity

{

get { return this.velocity; }

set { this.velocity = value; }

}

/// <summary>

/// Returns a string that represents the wind's direction.

/// </summary>

/// <returns>wind is Right or Left or no wind</returns>

public string getDirection()

{

if (this.velocity > 0)

return "Right";

else if (this.velocity < 0)

return "Left";

else

return "None";

}

}

}

Target.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// object Target - represents the target that the bullet is supposed to hit.

/// Since the target is circular it has a radius measured in meters. Target's distance is also

/// measured in meters.

/// </summary>

class Target

{

private double distance;

private double radius;

/// <summary>

/// Target constractor

/// </summary>

/// <param name="distance">get distance value</param>

/// <param name="radius">get radius value</param>

public Target(double distance = 0, double radius = 0)

{

this.distance = distance;

this.radius = radius;

}

/// <summary>

/// geter and seter for Distance

/// </summary>

public double Distance

{

get { return this.distance; }

set { this.distance = value; }

}

/// <summary>

/// geter and seter for Radius

/// </summary>

public double Radius

{

get { return this.radius; }

set { this.radius = value; }

}

}

}

Bullet.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// object Bullet - cossists of bullet's mass in kg, bullet's ressistance coefficient and bullet's velocity

/// vector.

/// </summary>

class Bullet

{

private double mass;

private double ressistance;

private VelocityVector velocityVector;

/// <summary>

/// Bullet constractor

/// </summary>

/// <param name="mass">get the values for mass</param>

/// <param name="ressistance">get the values for ressistamce</param>

/// <param name="velocityVector">get the values for velocity vector</param>

public Bullet(double mass, double ressistance, VelocityVector velocityVector)

{

this.mass = mass;

this.ressistance = ressistance;

this.velocityVector = velocityVector;

}

/// <summary>

/// geter and seters for mass.

/// </summary>

public double Mass

{

get { return this.mass; }

set { this.mass = value; }

}

/// <summary>

/// geter and seters for Ressistance.

/// </summary>

public double Ressistance

{

get { return this.ressistance; }

set { this.ressistance = value; }

}

/// <summary>

/// geter and seters for VelocityVector.

/// </summary>

public VelocityVector VelocityVector

{

get { return this.velocityVector; }

set { this.velocityVector = value; }

}

/// <summary>

/// Calculates bullet's velocity or location in a given direction and time.

/// </summary>

/// <param name="calc">Defines whether velocity or location needs to be calculated.</param>

/// <param name="direction">the direction</param>

/// <param name="wind">Wind can influence the bullet on the X axis.</param>

/// <param name="t">Time t - measured in seconds.</param>

/// <returns>bullet's velocity or location in a given direction and time</returns>

public double CalcByDirection(Calculate calc, Direction direction, Wind wind,

double t)

{

double k = this.ressistance;//ressistance

double m = this.mass;//mass

double b = k / m;//ressistance over mass

double g = 0;//Gravity

double v = 0;//Initial speed

//Define initial speed abd gravity based on the given direction.

switch (direction)

{

case Direction.X:

v = this.velocityVector.XVelocity + wind.Velocity;

break;

case Direction.Y:

v = this.velocityVector.YVelocity \* -1;

g = 9.8;

break;

case Direction.Z:

v = this.velocityVector.ZVelocity;

break;

}

double output = 0;

//Chooses the needed formula according to the given parameters.

switch (calc)

{

case Calculate.LOCATION:

if (k != 0) //if the ressistance not zero

output = (-g + b \* g \* t + b \* v + (g - b \* v) \* Math.Pow(Math.E, -b \* t)) / Math.Pow(b, 2);

else // the ressistance is zero

output = (g \* Math.Pow(t, 2) + 2 \* t \* v) / 2;

break;

case Calculate.VELOCITY:

if (k != 0) //if the ressistance not zero

output = ((g / b) + (Math.Pow(Math.E, -b \* t) \* (v - (g / b))));

else // the ressistance is zero

output = g \* t + v;

break;

}

if (direction == Direction.Y)

output \*= -1;

return output;

}

}

}

Simulation.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Bullet\_Trajectory

{

/// <summary>

/// Simulation - consists of a Bullts, Wind and a Target. Simulation generates all the data needed

/// in order to simulate Bullet's trajectory.

/// </summary>

class Simulation

{

private Bullet bullet;

private Wind wind;

private Target target;

/// <summary>

/// Simulation constractor

/// </summary>

/// <param name="bullet">get bullet values</param>

/// <param name="wind">get wind values</param>

/// <param name="target">get target values</param>

public Simulation(Bullet bullet, Wind wind, Target target)

{

this.bullet = bullet;

this.wind = wind;

this.target = target;

}

/// <summary>

/// Geter and seters for bullet

/// </summary>

public Bullet \_bullet

{

get { return this.bullet; }

set { this.bullet = value; }

}

/// <summary>

/// Geter and seters for wind

/// </summary>

public Wind \_wind

{

get { return this.wind; }

set { this.wind = value; }

}

/// <summary>

/// Geter and seters for target

/// </summary>

public Target \_target

{

get { return this.target; }

set { this.target = value; }

}

/// <summary>

/// Calculat x, y, z direction by sending the velocity starting direction the wind and the time

/// </summary>

/// <param name="t">time</param>

/// <returns>new bellet's velocity vector at any time given.</returns>

public double getBulletVelocity(double t)

{

double x = this.bullet.CalcByDirection(Calculate.VELOCITY, Direction.X, this.wind, t);

double y = this.bullet.CalcByDirection(Calculate.VELOCITY, Direction.Y, this.wind, t);

double z = this.bullet.CalcByDirection(Calculate.VELOCITY, Direction.Z, this.wind, t);

return new VelocityVector(x, y, z).getVelocity();

}

/// <summary>

/// Get the time and use the bullet location direction and the wind

/// </summary>

/// <param name="t">time</param>

/// <returns>new cordinates of x, y, z</returns>

public Cordinates getBulletCordinates(double t)

{

double x = this.bullet.CalcByDirection(Calculate.LOCATION, Direction.X, this.wind, t);

double y = this.bullet.CalcByDirection(Calculate.LOCATION, Direction.Y, this.wind, t);

double z = this.bullet.CalcByDirection(Calculate.LOCATION, Direction.Z, this.wind, t);

return new Cordinates(x, y, z);

}

/// <summary>

/// Function will calculat time to bullet to hit the target

/// </summary>

/// <returns>time that takes to bullet to hit target.</returns>

public double getHitTime()

{

double v = this.\_bullet.VelocityVector.ZVelocity;

double m = this.\_bullet.Mass;

double x = this.\_target.Distance;

double k = this.\_bullet.Ressistance;

if (k != 0)

{

//If k \* x >= m \* v then bullet can not reach target. In that case return -1.

if (k \* x >= m \* v)

return -1;

return (m \* Math.Log((m \* v) / (m \* v - k \* x))) / k;

}

else

return x / v;

}

}

}

Form1.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace Bullet\_Trajectory

{

public partial class MainForm : Form

{

private Simulation simulation;

public MainForm()

{

InitializeComponent();

}

/// <summary>

/// Runs when the user presses the "Shoot" button.

/// Check for invalid inputs, calculates bullet's trajectory and shows the results.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void shoot\_Click\_1(object sender, EventArgs e)

{

//Checks for invalid inputs and informs the user which values need to be changed (if there are any).

string message = inputIsValid();

if (message.Length > 0)

{

MessageBox.Show(message, "Ivalid input");

return;

}

//Builds simulation object.

buildSimulation();

//Checks if the bullet reached to target distance.

if (!reachedTarget())

return;

//Fill DataGrid and paint the crosshair on the target.

fillDataGrid();

paintCrossHairAndShowResults();

}

/// <summary>

/// Checks if the bullet reached to target distance.

/// If the bullet did not reach target distance clear this.dataGridView and inform the user.

/// </summary>

/// <returns>Boolean value</returns>

private bool reachedTarget()

{

if (this.simulation.getHitTime() == -1)

{

this.dataGridView.DataSource = null;

this.targetPicture.Refresh();

showLabelResults(0, false);

return false;

}

else

return true;

}

/// <summary>

/// Builds simulation object from all the user's inputs

/// </summary>

private void buildSimulation()

{

//Bullet:

double mass = Convert.ToDouble(this.m.Text);

double velocity = Convert.ToDouble(this.Vo.Text);

double xAngle = Convert.ToDouble(this.theta1.Text);

double yAngle = Convert.ToDouble(this.theta2.Text);

VelocityVector velocityVector = new VelocityVector();

velocityVector.defineByAngles(xAngle, yAngle, velocity);

double ressistance = Convert.ToDouble(this.k.Text);

//Wind:

Wind wind = new Wind();

if (this.useWind.CheckState == CheckState.Unchecked)

wind.Velocity = 0;

//Target:

double radius = Convert.ToDouble(this.radius.Text);

double distance = Convert.ToDouble(this.distance.Text);

//Build Simulation object:

this.simulation = new Simulation(new Bullet(mass, ressistance, velocityVector),

wind, new Target(distance, radius));

}

/// <summary>

/// Creates a table that shows bullet's location and velocity as a function of time from

/// t = 0 bullt's hit time.

/// </summary>

private void fillDataGrid()

{

double hitTime = this.simulation.getHitTime();

DataTable table = new DataTable();

//Add table colums.

table.Columns.Add("t", typeof(double));

table.Columns.Add("X", typeof(double));

table.Columns.Add("Y", typeof(double));

table.Columns.Add("Z", typeof(double));

table.Columns.Add("Vt", typeof(double));

//Fill table rows with the requiered data from t = 0 till bullt's hit time

//and round all the numbers.

for (double i = 0; i <= 1; i += 0.1)

{

double t = hitTime \* i;

Cordinates cord = this.simulation.getBulletCordinates(t);

table.Rows.Add(Math.Round(t, 3), Math.Round(cord.X, 3), Math.Round(cord.Y, 3),

Math.Round(cord.Z, 3), Math.Round(this.simulation.getBulletVelocity(t), 3));

}

this.dataGridView.DataSource = table;

this.dataGridView.AutoResizeColumns();

}

/// <summary>

/// Check if a given string is a number.

/// </summary>

/// <param name="val"></param>

/// <returns></returns>

private bool isNumeric(string val)

{

Double result;

return Double.TryParse(val, System.Globalization.NumberStyles.Number,

System.Globalization.CultureInfo.CurrentCulture, out result);

}

/// <summary>

/// Checks that all the input is valid.

/// </summary>

/// <returns>string that describes all the invalid inputs.</returns>

private string inputIsValid()

{

string message = "";

//If text boxes are empty write 0.

if (this.theta1.Text.Length == 0)

this.theta1.Text = "0";

if (this.theta2.Text.Length == 0)

this.theta2.Text = "0";

if (this.k.Text.Length == 0)

this.k.Text = "0";

/\*Test all inputs and Values\*/

//Check mass is valid:

if (!(isNumeric(this.m.Text) && Convert.ToDouble(this.m.Text) > 0))

message += "Mass value is invalid - enter a positive value.\n";

//Check muzzle velocity is valid:

if (!(isNumeric(this.Vo.Text) && Convert.ToDouble(this.Vo.Text) > 0))

message += "Muzzle velocity value is invalid - enter a positive value.\n";

//Check horizontal angle is valid:

if (!(isNumeric(this.theta1.Text) && Math.Abs(Convert.ToDouble(this.theta1.Text)) <= 10))

message += "Horizontal angle value is invalid - enter a value between -10 to 10.\n";

//Check vertical angle is valid:

if (!(isNumeric(this.theta2.Text) && Math.Abs(Convert.ToDouble(this.theta2.Text)) <= 10))

message += "Vertical angle value is invalid - enter a value between -10 to 10.\n";

//Check Ressistance cofficient value is valid:

if (!(isNumeric(this.k.Text) && Convert.ToDouble(this.k.Text) >= 0))

message += "Ressistance cofficient value is invalid - enter zero or a positive value.\n";

//Check target radius is valid:

if (!(isNumeric(this.radius.Text) && Convert.ToDouble(this.radius.Text) >= 0))

message += "Target radius value is invalid - enter a positive value.\n";

//Check target distance is valid:

if (!(isNumeric(this.distance.Text) && Convert.ToDouble(this.distance.Text) >= 0))

message += "Target distance value is invalid - enter a positive value.\n";

return message;

}

/// <summary>

/// Paint a crosshair on targetPicture that shows the bullet's hit location relatively to

/// the targe't center.

/// Shows also information about distance from bullet's hit location to target's center,

/// wind speed and direction.

/// </summary>

private void paintCrossHairAndShowResults()

{

//Get bullet hit cordinates.

Cordinates hitCord = this.simulation.getBulletCordinates(this.simulation.getHitTime());

double distanceFromCenter = Math.Sqrt(Math.Pow(hitCord.X, 2) + Math.Pow(hitCord.Y, 2));

//Return if bullet missed target.

if (distanceFromCenter > this.simulation.\_target.Radius)

{

showLabelResults(distanceFromCenter, true);

//Redraw image.

this.targetPicture.Refresh();

return;

}

//Get ratio between picture size and target's radius.

double xRatio = this.targetPicture.Width / (this.simulation.\_target.Radius \* 2);

double yRatio = this.targetPicture.Height / (this.simulation.\_target.Radius \* 2);

//Get hit location on the picture.

Point hitPoint = new Point();

hitPoint.X = (int)((this.targetPicture.Width / 2) + (hitCord.X \* xRatio));

hitPoint.Y = (int)((this.targetPicture.Height / 2) - (hitCord.Y \* yRatio));

using (Graphics crossHair = this.targetPicture.CreateGraphics())

{

//Redraw image.

this.targetPicture.Refresh();

//Paint crosshair:

Pen p = new Pen(System.Drawing.Color.Red, 2);

crossHair.DrawLine(p, hitPoint.X - 15, hitPoint.Y, hitPoint.X + 15, hitPoint.Y);

crossHair.DrawLine(p, hitPoint.X, hitPoint.Y - 15, hitPoint.X, hitPoint.Y + 15);

}

//Show results.

showLabelResults(distanceFromCenter, true);

}

/// <summary>

/// Provides the user with the following information:

/// - Bullet's distance from target's center (assuming the bullet hits the target)

/// - Wind's direction

/// - Wind's speed

/// </summary>

/// <param name="distanceFromCenter"></param>

/// <param name="reachTarget">Describes whether the bullet managed to reach to target distance.</param>

private void showLabelResults(double distanceFromCenter, bool reachTarget)

{

if (reachTarget == false)

this.resultLabel.Text = "Bullet did not reach target.Try to increase muzzle velocity / mass\n" +

"or reduce target distance / ressistance cofficient.";

else

{

if (distanceFromCenter > this.simulation.\_target.Radius)

this.resultLabel.Text = "Bullet missed target";

else

//Print distance from target.

this.resultLabel.Text = "Distance from target center: " + Math.Round(distanceFromCenter, 3)

+ " meters";

}

//Print wind direction and speed.

this.windDirection.Text = "Wind direction: " + this.simulation.\_wind.getDirection();

this.windSpeed.Text = "Wind speed: " + Convert.ToString(Math.Abs(this.simulation.\_wind.Velocity)) +

" m/s";

}

/// <summary>

/// Runs when the user presses the "Clear" button.

/// Clears all the input fields and all the shown results.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void clear\_Click\_1(object sender, EventArgs e)

{

this.targetPicture.Refresh();

this.useWind.CheckState = CheckState.Unchecked;

this.m.Text = "";

this.Vo.Text = "";

this.theta1.Text = "";

this.theta2.Text = "";

this.k.Text = "";

this.radius.Text = "";

this.distance.Text = "";

this.resultLabel.Text = "";

this.windDirection.Text = "";

this.windSpeed.Text = "";

this.dataGridView.DataSource = null;

}

}

}

Velocity Vector :

This is a mathematical explanation for the formulas found in VelocityVector.defineByAngles:

Consider the following system of axes:

X axis is horizontal (points right), Y axis is vertical (points up) and Z axis points forward.

– Horizontal angle, represents the vector’s deviation on the X axis.

– Vertical angle, represents the vector’s deviation on the Y axis.

can be described as a sum of 3 vectors:

Finding vector’s components x, y, z using vector’s magnitude, and angles and :

Finding bullet’s velocity and location as a function of time:

This is a mathematical explanation for the formulas found in Bullet.CalcByDirection:

Finding v as a function of t if

Finding x as a function of t if :

Finding v as a function of t if

Finding x as a function of t if :

Finding bullet’s hit time:

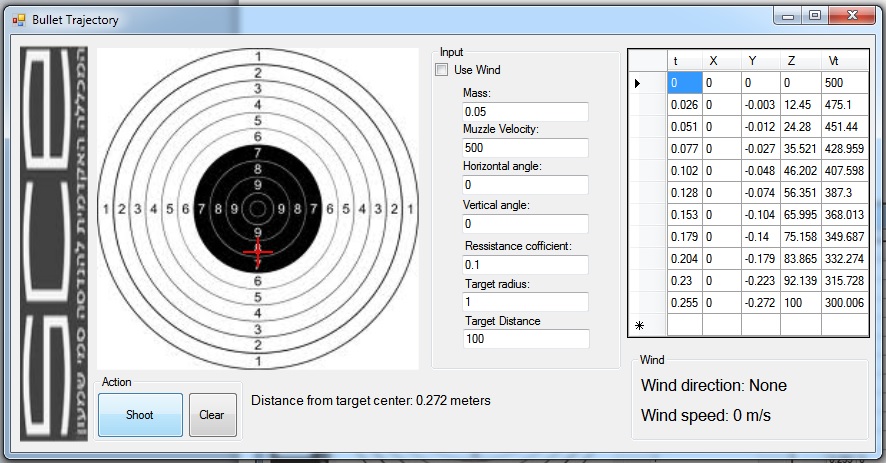
This is a mathematical explanation for the formulas found in Simulation.findHitTime

Note: Since bullet’s hit time is calculated on the Z axis gravity is irrelevant.

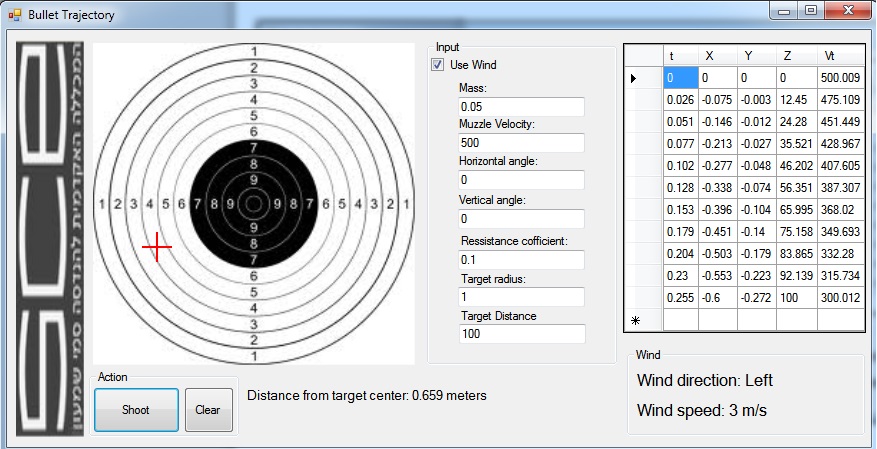
Finding hit time if :

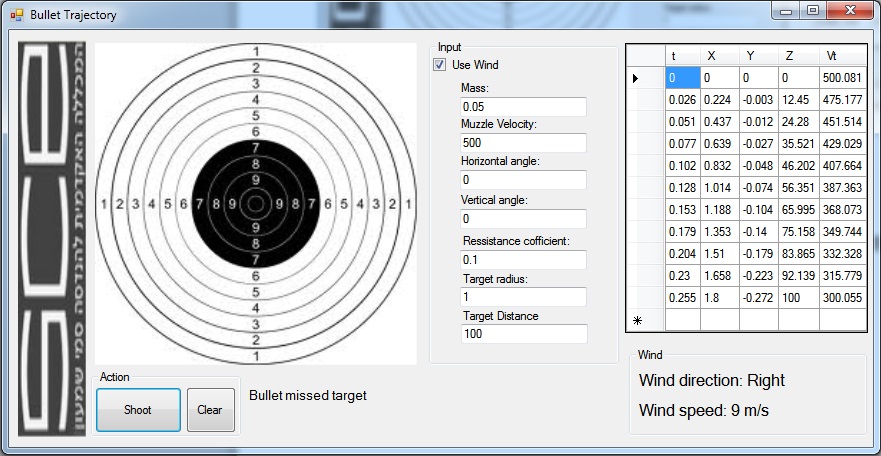
Finding hit time if :

1. **Examples of software execution**

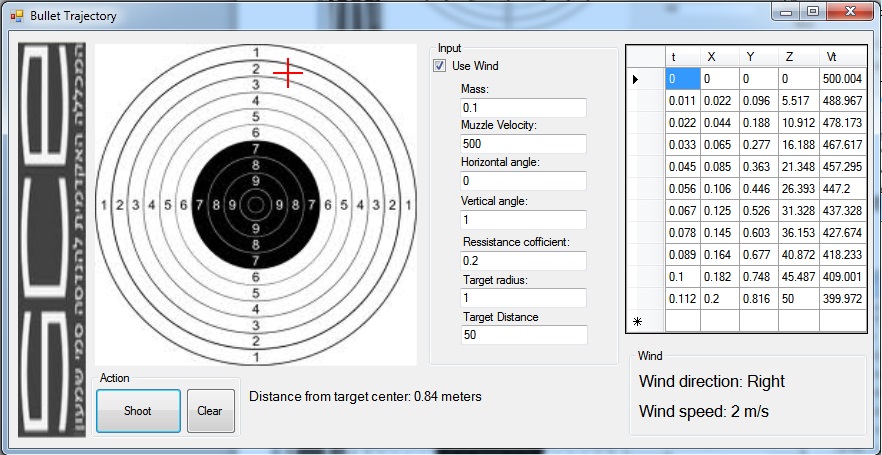
****

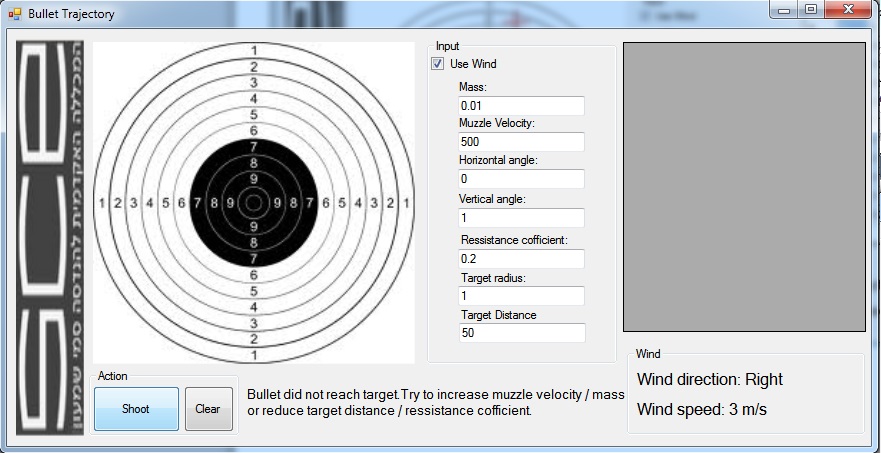
Same inputs with wind:

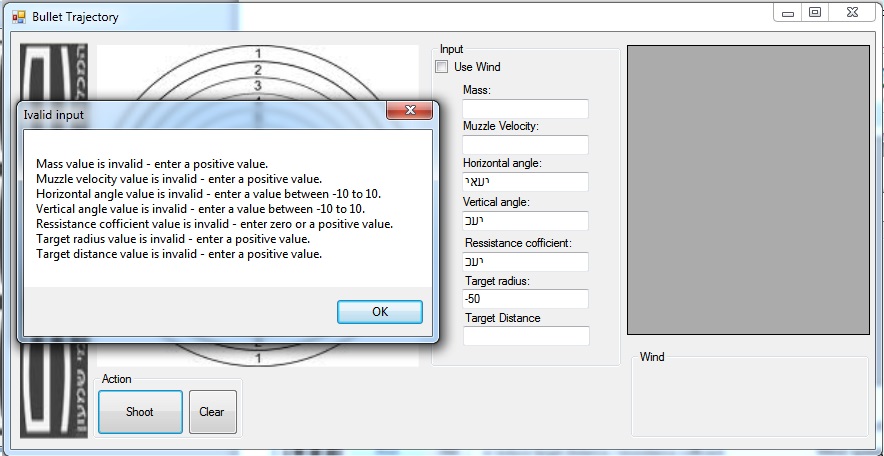


****

It is possible to shoot the bullet in an angle:



It’s also possible for the bullet not to reach target distance:

Example of invalid input:

1. **Additional remarks**

The team work was remarkable we achieve all the targets goals.

Our Team leader Roman’s algorithm cod successfully compiled after a few tries.

1. **References**

PMBOK (Project management book of knowledge)

Waterfall model - <http://www.buzzle.com/articles/waterfall-model-diagram.html>

Lecture 7 – Project 3: Mathematical model for a falling body:



Applications of First Order Differential Equations - Falling Object:

<https://www.youtube.com/watch?v=TMQdGqQuY5w>

1. **Additional remarks including development history**

The main difficulty in the project development was developing the formulas in order to calculate bullet’s hit time and bullet’s velocity and location in a given time. We went through several iterations before we managed to develop the formulas properly. Most of the bugs that happened ware a result of wrong formulas being utilized. Aside from that the rest of the development process went pretty smoothly.