Ballistics Trajectory Calculator 4.0

Developer Documentation

- **Defines**
  - #define **WIDTH** 80
    - This defines the width of the grid that will be used to display the trajectories
  - #define **HEIGHT** 25
    - This defines the height of the grid that will be used to display the trajectories
  - #define _CRT_SECURE_NO_WARNINGS
    - to avoid some Visual Studio warnings
  - #define _USE_MATH_DEFINES
    - to use M_PI
- **Structures**
  - **starting_data**: The struct is useful because it holds the user-entered values together, so that it's easier to pass it on to functions
    - double target: The distance of the target, we will ask the user to enter it
    - double wind: The speed of wind, we will ask the user to enter it
    - double v0: The starting velocity, we will ask the user to enter it
    - double m: The mass of the projectile, we will ask the user to enter it
    - double A: The cross sectional area of the projectile, we will ask the user to enter it
  - **trajectory_point**: Will be used in distance(), when the grid needs to be filled up
    - int x: The x coordinate of the point
    - int y: The y coordinate of the point
- **Functions**
  - **int sure(void):** This function is used to confirm the choice of the user (for instance when he chooses to launch the calculations, or chooses to exit the program)
    - The function asks the user whether he is sure or not, then takes in either 1 (yes) or 2 (no). It uses switch, and in case 1, the function returns 0, in case 2 the function returns 1, in the default case we ask the user to enter 1 or 2.
  - **double read_data(void):** This function reads in a double value from the user and returns that double value.
  - **double is_positive(void):** This function reads in a double value greater than 0, then returns it
    - The function uses the read_data function to read in a double from the user, then in a *while* loop if that double is smaller than or equal to 0 it tells the user to enter a number greater than 0, reading in the value again, until it reads a correct value, breaking the while loop, returning the double.
  - **char **grid_create(void):** This function creates and returns the grid that will be used for displaying
    - The function uses malloc() to create a dynamic 2D array
      - It creates a char** array with the size of HEIGHT times char*, to store the rows of the grid
      - Then it creates the rows, WIDTH long also using malloc(), to store characters

        - It checks for insufficient memory, returning an error message and NULL pointer if unable to create the array
        - Otherwise it fills up the whole array with spaces, then returns it
- **void display_trajectory(char **grid):** This function receives the grid as a 2D char array, then prints out the trajectory to the screen
    - The function uses a simple nested *for* loop to print each row of the grid to the screen (first *for* runs HEIGHT times, and in each instance a *for* runs WIDTH times to print each element)
- **void file_write(char **grid, double angle1, double angle2, starting_data start_data, double time):** This function receives a 2D array grid containing the trajectories, the two angles that are the results of the calculations, the starting data as a starting_data struct, and the time of flight. The function writes all this out into a txt file.
    - The function checks whether the creation of the file Ballistics.txt (or opening of existing file) was successful
    - Then the function prints the angle (or angles, if there are two), and the starting data, then displays the graph (sadly the display_trajectory() can't be used, as that prints to the console, but we can do it the same way just printing into a file)
    - At the end the function closes the file, displaying an error message if it was unsuccessful
- **void distance(double i, starting_data start_data, double **angle, double *min, int *found, double *prev_diff, double *prev_dist, double *h_max, double *dist_max, int print, char **grid, double *time):** This function calculates the distance with which we can shoot, finds the angles, and also deals with filling the grid with the actual trajectories
    - The input parameters: **i** – the angle in the calculations, **start_data**: the starting set of data, **angle** – the real value of the angle that will be updated, **min**- the minimum difference from target that will be updated and checked between function calls, **found** – our way of signaling the result, **prev_diff** – the difference in the previous function call, **prev_dist** – the distance reached in the previous function call, **h_max**- max height reached, needed to scale the trajectory for the grid, **dist_max** – maximum distance reached, needed for the grid and to tell whether we can reach the target, **print** – to signal in which function call we need to fill the grid, **grid** – the grid for the trajectory, **time** – the time of landing (time of flight)
    - In the first part, the function makes the calculations for the i angle, with air resistance, using the Euler-method to numerically calculate the trajectory
    - In the second part, if print==1, we fill up our grid, normalizing the x and y distances to fit into the grid, with the maximum height being h_max, and the maximum distance along the x axis being dist_max
    - In the third part, we check whether this is the angle while rising upwards (first angle), the angle while falling downwards (second angle), the only angle, or an angle sufficiently close to the target (this last one is only used when we want to calculate the precise angle)

- **void calculations(starting_data start_data, double* angle1, double* angle2):** This is the function that calls most of the other functions as the calculations begin. (It is better in a large function than it would be in main() for example).
    - start_data: the starting data, angle1 – the first angle, angle2 – the second angle.
    - The function first creates the grid by calling the grid_create() function, then it initializes a lot of auxiliary variables
    - The function then runs distance(), checking for angles from 0 until one angle is found, then it makes more precise calculations, then identifies the type of angle (first angle, or the only one), and if it's the only angle it checks whether the target is reachable or not. If not, the function informs the user, and returns to main.
    - After this the function calculates the second angle with distance() (if there is a second angle), calculates it to be more precise, then fills up the grid, displays it with display_trajectory(), then it writes the data into the file with file_write()
    - In the end, the function frees the 2D array
- **int main(void):** Here in main() we have the menu with switch, where the cases cover reading in the target, wind, mass, area and velocity with read_data or is_positive
    - In main the variables starting_data start_data and the two angles are declared
    - Also, main will call the calculations() when the user chooses to launch it
    - The menu will keep running until the user exits