Due Date:

Friday April 18, at 11:59 p.m.

Important:
- All projects are to be completed individually; duplicate programs will not receive credit.
- Your code must compile and run on the systems in Dragas.
- Review the grading algorithm before you submit your project!

Background:
You have been tasked with completing the projectile motion game. This is the final project of the semester. This project requires an understanding of the following topics:

- Variables and scope.
- Functions—i.e., prototypes, definitions, and invocations.
- Arrays of structs.
- Structs passed by reference.
- Structs passed by constant reference.
- Nested loops—both while and for.

Description:
This project—Project 5—is a complete—albeit rough—game. The basic logic is defined as follows:

1. Parse the input file.
2. Setup the Game environment.
3. Setup the Players.
4. While at least one player remains and all players wish to continue:
   a. Prompt each player—with health remaining—to select an action:
      i. Move the tank—if fuel remains.
      ii. Launch a projectile.
         1. Prompt for initial speed.
         2. Prompt for elevation.
      iii. Skip the current turn—i.e., forfeit this turn.
   b. If the player wishes to move his/her tank:
      i. Prompt for a value based on remaining fuel.
      ii. Update the player position.
c. If the player wishes to launch a projectile:
   i. Calculate the impact location.
   ii. Compute the difference from each target.
      1. If difference <= EFFECTIVE_RADIUS.
         a. Compute damage and update the player’s round_score.
         b. Add 1 fuel unit to the player’s fuel
      iii. Compute the difference from each player—including the current player.
         1. Repeat the same logic as described for targets.
         2. If difference <= EFFECTIVE_RADIUS.
            a. Compute damage and update the current player’s round_score.
            b. Add 10 fuel unit to the current player’s fuel.
            c. Update all hit players’ health.

   d. If the player wishes to skip the round:
      i. Print a message.

   e. All players with 0 health remaining are automatically skipped.

Instructions:

- Preliminary Instructions
- Read this prompt completely.
- Review P4-solution-1.cpp
- You may declare and define additional functions.
- You must properly indent and document your code.
- Read the provided template—including all comments.
  - Familiarize yourself with the updated Player struct.
  - Familiarize yourself with the Game struct.
  - Review arrays of structs—see Lab Assignment 8.
- Familiarize yourself with the main function.
- Remember to make a plan before starting the project.
- Make no changes other than those explicitly listed.

- Coding Style
  - Remember to initialize all variables.
  - Remember to comment your code.

- Input File Format
  - The input file is formatted as follows:
    - Line 1: lower and upper boundaries.
    - Line 2: Number of players.
    - Line 3: Number of targets.
    - Lines 4+: Target Position.
• Algorithm
  • Do NOT modify the **Main** function.
  • Complete the **validateNumericInput** function.
    • Implement the following logic:
      • While to_validate is not in the range \([\text{bound}_\text{lower}, \text{bound}_\text{upper}]\)
        a. i.e., \(\text{to}_\text{validate} < \text{bound}_\text{lower} \text{ or } \text{to}_\text{validate} > \text{bound}_\text{upper}\)
        b. Re-prompt the user with an error message.
          i. See the sample output.
          ii. Your error message must match.
    • Return the updated **to_validate**.
  • Complete the **increaseFuel** function.
    • Add to the player’s current fuel.
    • **Hint:** \(\text{player}.\text{fuel} += \text{to}_\text{add}\)
  • Complete the **applyDamage** function.
    • Subtract damage from the player’s health.
    • **Hint:** \(\text{player}.\text{health} -= \text{to}_\text{subtract}\).
    • If the resulting health is less than 0:
      • Set the player’s health to zero.
  • Complete the **updateScore** function.
    • Add to the player’s score, the value **to_add**.
    • **Hint:** see the previous hints.
  • Complete the **move** function.
    • Move the player by the value specified in **to_move**.
    • Deduct the distance moved from the player’s fuel.
      • E.g., -6.28 and 6.28 should both **reduce** remaining fuel.
      • **Hint:** \(\text{abs}(\text{to}_\text{move})\).
  • Complete the **computeFinalScore** function.
    • Apply a multiplier—i.e. scaling factor—to the player’s score.
    • Store the adjusted player score.
      • **Hint:** update player.score.
    • Use the following equation to compute the multiplier:
      • \(\left(\frac{\text{remaining health}}{\text{STARTING HEALTH}}\right) \times \left(\frac{\text{remaining fuel}}{\text{STARTING_FUEL}}\right)\)
  • Complete the **computeDamage** function.
    • Compute the damage value based on difference.
    • You are **limited** to one **return** statement
    • If difference is less than 1:
      • The damage is 999.
    • Otherwise the damage is:
      • \(\text{EFFECTIVE_RADIUS} / \text{difference}\)
    • **Hint:** With ternary operator, this is one line.
• Complete the `printRoundSummary` function.
  • Print the following with appropriate labels and formatting:
    • Player position.
    • Player health.
    • Player fuel.
  • One blank line.
  • Round score.
  • Player score.
  • See the sample output.

• Complete the `printMovementSummary` function.
  • Print the following with appropriate labels and formatting:
    • Player position.
    • Player fuel.
  • See the sample output.

• Complete the `printFinalSummary` function.
  • Print the following **General** output:
    • Field:
      a. lower bound
      b. upper bound
    • # of targets
    • # of rounds
    • # of players.
  • One blank line
  • Print the following for all players:
    • **Hint:** for loop.
    • Player score.
    • Player position.
    • Player fuel.
    • Player score.
    • One blank line
  • Player health.
  • See the sample output.

5. **Output Formatting**

• All output must match the sample output.
• Make use of `setw`, `left`, `right`, and `right--` where appropriate.
Enter the input filename: P5-input-2.txt

Round 1: Player 1

Current Position : 0.00
Current Health : 46.00
Current Fuel : 18.00

Available Actions

M. Move (Max 10.00 meters)
F. Fire Projectile
S. Ship Turn

Make a selection : F

Fire Projectile

Enter an Initial Speed : 73.5
Enter an Elevation : 45

Projectile Summary

Initial Speed : 73.50
Elevation : 45.00
Impact Pos. : 550.00

Player : 0
Position : 550.00 (meters)
Error : 0.69 (meters)
Points : 997.00

Summary: Player 1

Position : 0.00
Health : 46.00
Fuel : 18.00
Round Score : 999.00
Total Score : 999.00

Round 1: Player 2

Current Position : 550.00
Current Health : 0.00
Current Fuel : 18.00

Skipping Player 2.

Round 1: Player 3

Current Position : 1100.00
Current Health : 48.00
Current Fuel : 18.00

Available Actions

M. Move (Max 10.00 meters)
F. Fire Projectile
S. Ship Turn

Make a selection : m

Move Cannon

Enter a distance : -20
Value must be in [-10.00, 10.00]: 20
Value must be in [-10.00, 10.00]: -8

Summary: Player 3

Position : 1092.00
Fuel : 2.00

Continue the game? <Y/N>: y
Round 2: Player 1

Current Position : 8.00
Current Health : 46.00
Current Fuel : 20.00

Available Actions

M. Move (Max 20.00 meters)
F. Fire Projectile
S. Skip Turn

Make a selection : f

Fire Projectile

Enter an Initial Speed : 34
Enter an Elevation : 45

Projectile Summary

Initial Speed : 34.00
Elevation : 45.00
Impact Pos. : 117.04

Summary: Player 1

Position : 8.00
Health : 46.00
Fuel : 20.00

Round Score : 0.00
Total Score : 999.00

Round 2: Player 2

Current Position : 556.00
Current Health : 0.00
Current Fuel : 18.00

Skipping Player 2.

Round 2: Player 3

Current Position : 1092.00
Current Health : 48.00
Current Fuel : 2.00

Available Actions

M. Move (Max 2.00 meters)
F. Fire Projectile
S. Skip Turn

Make a selection : s

Player 3 has forfeited his/her turn.

Continue the game? (Y/N): y
Round 3: Player 1

Current Position = 9.00
Current Health = 45.00
Current Fuel = 28.00

Available Actions
M. Move (Max 20.00 meters)
F. Fire Projectile
S. Skip Turn

Make a selection = f

Fire Projectile

Enter an Initial Speed = 32.5
Enter an Elevation = 45

Projectile Summary

Initial Speed = 32.50
Elevation = 45.00
Impact Pos. = 107.67
Target At = 100.00 (meters)
Error = 7.57 (meters)
Points = 1.30
Target At = 101.00 (meters)
Error = 6.67 (meters)
Points = 1.50
Target At = 102.00 (meters)
Error = 5.67 (meters)
Points = 1.76
Target At = 103.00 (meters)
Error = 4.67 (meters)
Points = 2.14

Summary: Player 1

Position = 9.00
Health = 45.00
Fuel = 28.00
Round Score = 6.71
Total Score = 106.71

Round 3: Player 2

Current Position = 550.00
Current Health = 0.00
Current Fuel = 10.00

Skipping Player 2.

Round 3: Player 3

Current Position = 1092.00
Current Health = 48.00
Current Fuel = 2.00

Available Actions
M. Move (Max 2.00 meters)
F. Fire Projectile
S. Skip Turn

Make a selection = n

Move Cannon

Enter a distance = 1

Summary: Player 3

Position = 1093.00
Fuel = 1.00

Continue the game? (Y/N): n
Sample Input File:

Submission Details:

- Your TA will review the process of submitting your assignments in class.
- You will submit an electronic copy of your .cpp file using Blackboard.
- Name your source code file using your CS login.
  - The format is `csloginP#.cpp`.
  - For example, John Smith would save his source code in a file named `j smithP#.cpp`.
  - # refers to the project number—e.g. Project 3 would be `jsmithP3.cpp`.
- You will be allowed to make more than one submission—**only your latest submission will be graded.**
- Remember to check that you uploaded the correct file onto blackboard! If you are unsure—check with your lab TA. **Before the deadline!**
- Save a copy of the file you hand in. Leave the file unchanged after your submission until you receive a grade on the project assignment.
- **Projects are to be completed individually;** duplicate programs will not receive credit, and will be reported as a violation of the ODU Honor Code.
Tips:

- You should test your solution with different data sets.
  - Examine the input files provided.
  - Create your own input files for your own testing purposes.
- Do not echo print the data values after reading them from the file.
- **Start early. In fact, start right now.**
- **Properly indent your code.**
- Remember that you can work in the Problem Solving Lab in Dragas 103 or the Lab located in room 3104 in the Engineering & Computational Science building. Hours for both are posted on the CS home page.
- Remember that you can use the Remote Desktop Connection.