CmpE 150.03 Introduction to Computing, Fall 2020

Homework 4 – Due: 12/12/2020, 23.59

You will write a Python script to implement an ASCII Atari-like game where a spaceship navigates and shoots asteroids using three parameters: **x** as the length of the asteroid cluster, **y** as the width of the asteroid cluster, and **g** as the current distance to the asteroid cluster. You will take these parameters from the user at the beginning of the game. The game is played with three actions: **left** to go one space left, **right** to go one space right, and **fire** to shoot a laser to destroy an asteroid in front of the spaceship. There is also an **exit** action to exit the game. You will take these actions from the user at each step of the game. Full example games are provided as .txt files, both input and output. Please read the assignment description **and** check out the examples before beginning to write any code or sending emails.

Please make sure you follow these rules in your implementation:

1. Assume that **x** >= 0, **y** > 0, **g** >= 0 always. We won't be testing for very large numbers, so do not worry about efficiency or timeouts.

2. Assume that inputs for **x**, **y**, and **g** will always be integers.

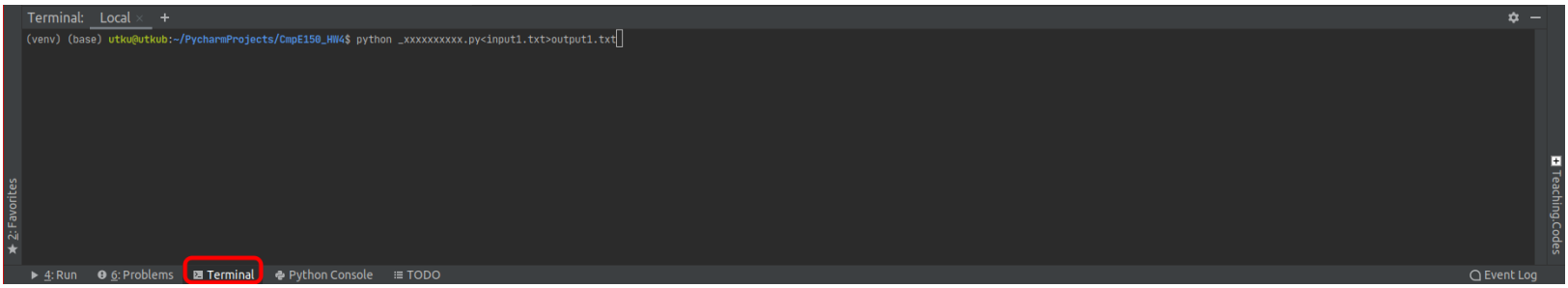
3. Assume that the actions **fire**, **left**, **right**, and **exit** are not case sensitive, they can be given like: Fire, EXIT, RiGhT etc.

4. You are not allowed to use statements that have not been covered in class as of 24/11/20 (Week 5 of the course on Moodle). For example, you cannot use functions (def).

5. Do not use any imports!

6. You can test your code by playing the game yourself and using PyCharm's terminal window while in the project.

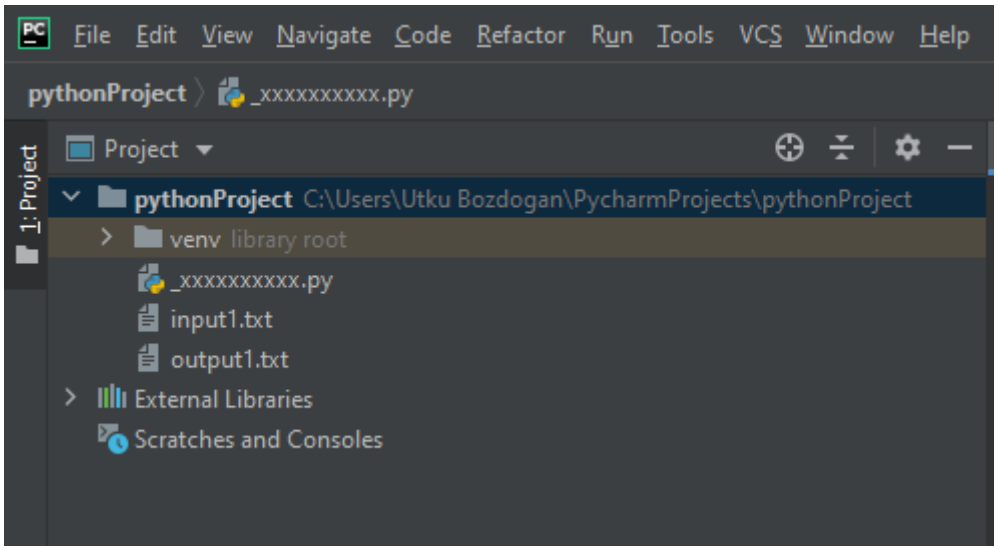
   Follow the format exactly as in the figure below.



Change the .py filename to your own, change the input and output filenames to the one you would like to try. < and > are very important, there should be no empty spaces between them and the file names. There should be a single empty space between python and your filename. This will run your code using the input from the input file and output it to the output file. You can then check the output file and see if it matches.

Note that when doing it this way unlike playing the game yourself, the input file contains the user input in order, and output file does not contain that input, so do not be surprised.

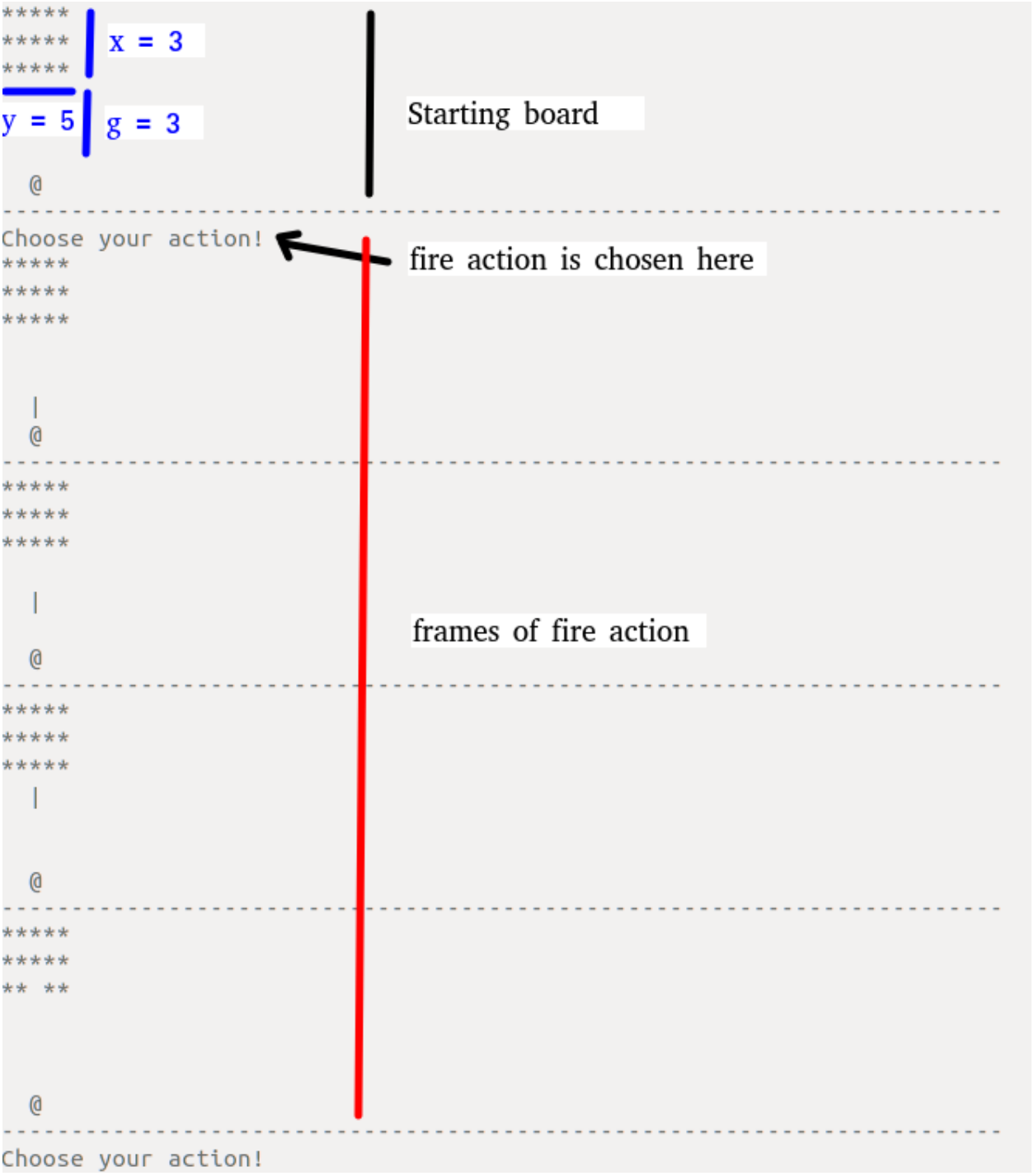
The files should be located under the project exactly as in the figure below for this to work.

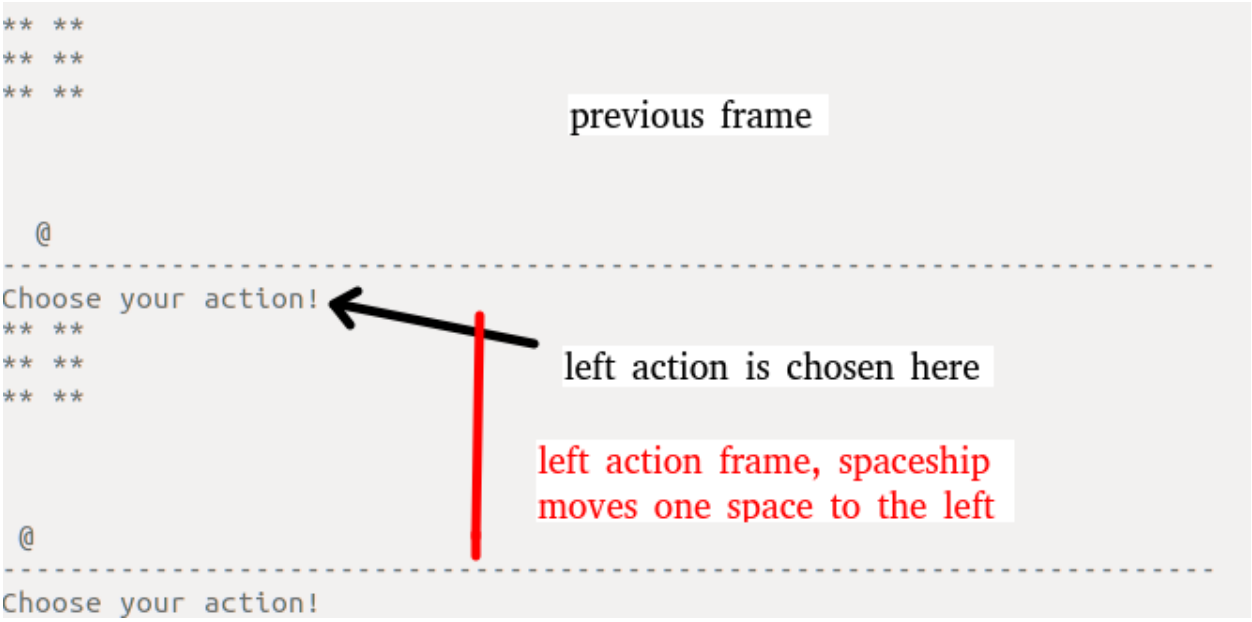


Implementation Details:

- You are expected to print everything to the terminal, **do not write to a file.**
- You are given some code at the beginning of the file, to take the inputs **x**, **y**, and **g**. Do not change those lines and use those as your variables.
- The asteroid cluster is **x** lines long and **y** characters wide, and it is **g** lines away from the spaceship. The spaceship is on another line as well.
- The spaceship starting position is at the middle of the line if **y** is odd, and on one space left of the middle of the line if **y** is even.
- There are 3 valid play actions: **fire**, **left**, **right** and 1 **exit** action. For any other action given, assume that there is a discussion in the ship and no action can be executed that **time**, although **time** passes, so print the last board configuration and continue with the next step.
- When the game starts, the first thing should be to print the board, showing the initial situation to the user. Assume that this is at **time** 0 and then ask the user for an action with the prompt: **Choose your action!** Then, execute this action and print the relevant frames, and advance the **time** by 1.

- When **time** is a multiple of 5, that means enough time has passed and asteroids get closer to the spaceship by 1 line.
- The game ends when the user exits with the **exit** action, when all the asteroids are destroyed, or when the spaceship is **hit by the asteroids** when the cluster gets too close. In all these cases, print the **score** and the relevant messages.
- **Hit by asteroids**: If there is at least 1 asteroid in any position on the closest line before the ship, and the **time** then becomes a multiple of 5 when any of the asteroids are still there, this will cause the game to be over.
- If the game is over due to **hit by asteroids**, you need to print **GAME OVER**, followed by the last board configuration, and then print the score.
- If the game is over because all the asteroids are destroyed, then the user has won. Then print the current board configuration, followed by **YOU WON!**
- If the game is over due to the user choosing the **exit** action, then print the last board configuration, followed by the score.
- The **score** is the number of asteroids destroyed at the end of the game. It should be printed at the end of each game, however the game ends as **YOUR SCORE:** followed by the value of **score** after printing the board.
- Try to consider all possible scenarios. For example, the user starts the game, and the first command is **exit**. In this case you should print the initial board configuration, then print the last board configuration, then print the score, which is 0.
- There are 72 separator hyphens (-) at each instance of their use.
- You can think of the game in terms of actions and frames. See the figures below.


*****
*****
*****
x = 3
y = 5
g = 3
Starting board
@
Choose your action!
fire action is chosen here
frames of fire action
Choose your action!



- Each action corresponds to one or more frames. **left** action moves the spaceship to the left then prints the updated frame, **right** action moves the spaceship to the right then prints the updated frame, **fire** action shoots a laser which travels until either it impacts an asteroid **or** until it reaches the other end of the frame. If an asteroid is hit by a laser, it should be destroyed.
- If the ship is already at the leftmost position of the board, **left** action does nothing but time passes. If the ship is already at the rightmost position of the board, **right** action does nothing, but time passes.

Submission: You will submit a single python file over Moodle. Your .py file should be named with the underscore character (_) followed by your student number (e.g. _2019700030.py). Do not use any Turkish characters or python keywords in your variable names.

Partial Submission: If you cannot generate exactly the correct output, you should still submit your code. Try to do as much as you can. Partial credit will be given if your submission is able to produce the correct outputs for **x** = 5, **y** = 10, **g** = 3 case. If you can produce the correct output for multiple configurations but not all, you will still get partial credit, and if you only have minor errors such as blank space count not matching or empty line count is not exactly as specified, you will still get partial credit provided the rest of the output is accurate.

Late Submission: Allowed with penalty: -% 10*(number_of_days_late)^2. Example case: You are 2 days late, and you got 90 from evaluation. You will get 90 * (1 - 0.4) = 54 as your final grade. You will get 0 if you are more than 3 days late.

Evaluation Procedure: We will evaluate your code based on the program output for many test cases. We will be looking for exact matches for full credit. You can check whether your output matches exactly the correct output by copying the output from the provided example output files, and copying your output to an online text diff checker, like https://www.diffchecker.com for example.

Good luck.