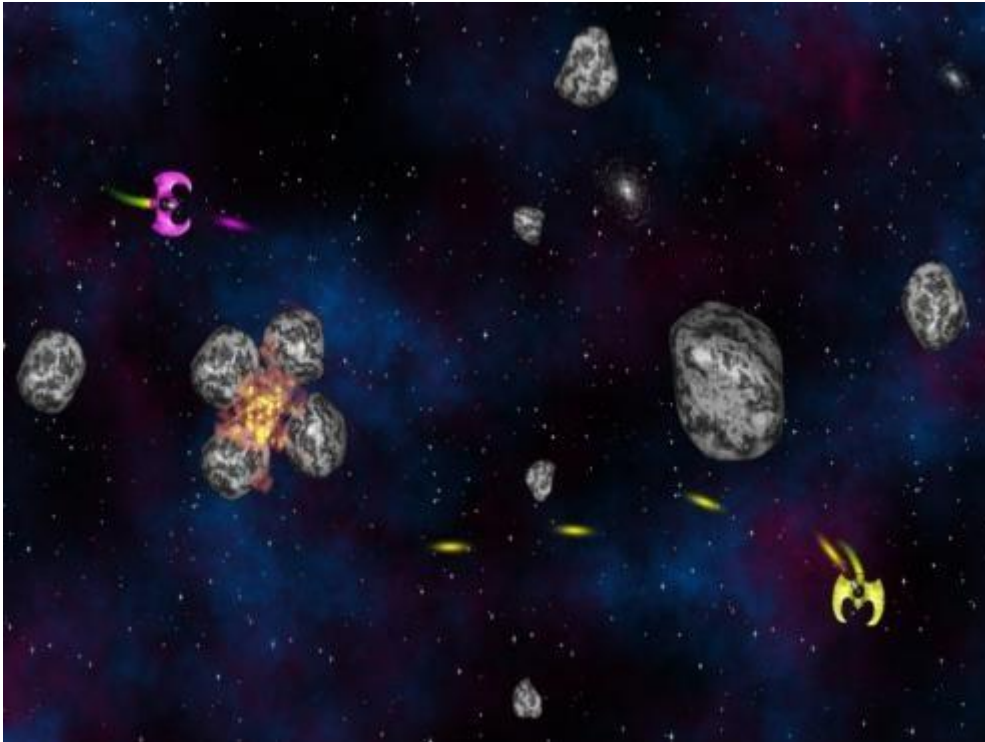


SPACESHIP (up to 100 points based on ranking)

This question is based loosely around the classic arcade game *Asteroids*. The player controls a spaceship which can shoot bullets at rocks. When hit enough times, rocks explode: larger rocks split into smaller rocks; below a certain size, rocks are simply destroyed when they explode. The playing field wraps around, so objects moving off the top reappear on the bottom, and similarly for the left and right edges. Bullets do not wrap around, only ships and rocks.



The player has two controls over their spaceship:

- A movement input, taking the form of a vector with magnitude ≤ 1 (imagine it as the position of a joystick). This vector is scaled and then applied directly to the ship's velocity as acceleration. The ship moves with its velocity, and a drag factor slows the velocity over time in the absence of any movement input.
- A shooting input, also in the form of a vector with magnitude ≤ 1 . The ship's gun turret rotates towards this input direction, and if the gun can fire (there is a limit on the rate of firing), a bullet is fired in the turret's current direction. The turret direction is independent of the ship's velocity and acceleration. If a zero shooting input is provided, the turret does not rotate and no bullet is fired.

One point is scored for each bullet that hits a rock, and a bonus point is scored if the rock explodes. If a spaceship hits a rock, that player is eliminated from the game. If all rocks are destroyed, a new level begins with more rocks (surviving players have their positions reset).

Multiple players take part in the game simultaneously. Their spaceships pass through one another without colliding, and bullets do not affect other spaceships. Players compete simply by scoring more points than one another. Staying alive longer is not taken into account in ranking players (although players that stay alive longer have the opportunity to score more points, of course).

TASK

Your task is to implement an AI routine to control a spaceship in this game. You should write a programme that repeatedly reads the game state for a frame (from standard input), decides what to do, and writes its actions (to standard output):

```
while (standard input stream open)
    Read game state input
    Write AI action as output
```

Your AI will be competing directly against the other entries in the same game.

INPUT

Each game frame of input will consist of the following information. Multiple data items on a line will be separated by a space:

- Line 1 contains useful game data:
 - Width of playing area (integer)
 - Height of playing area (integer)
 - Speed of a bullet (float)
- Line 2 will describe the current state of your ship:
 - Position x coordinate (float)
 - Position y coordinate (float)
 - Velocity x coordinate (float)
 - Velocity y coordinate (float)
 - Radius of ship for collision purposes (float)
 - Whether ship can shoot right now (0 for no, 1 for yes)
- Line 3 will contain a single integer N, the number of rocks
- The following N lines will each contain a single rock description:
 - Position x coordinate (float)
 - Position y coordinate (float)
 - Velocity x coordinate (float)
 - Velocity y coordinate (float)
 - Radius of rock for collision purposes (float)

For collision purposes, all objects are treated as circles. All quantities of floating point type will be presented in the form “-d.dddddE+ddd” or “-d.dddddE-ddd”: in other words,

- Optionally, a leading minus sign (for negative numbers)
- A single digit before the decimal point
- Six digits after the decimal point
- A capital E to separate mantissa from exponent
- A 3-digit exponent with plus or minus sign

Some examples:

-1.611058E-007 0.000000E+000 5.120000E+001

To parse this input format in our contest languages is simple – you don't actually need to worry about the details of the format:

C#

```
// First read data into string
// 'str', then:
float x = float.Parse(str);
```

C++

```
cin >> x;
```

OUTPUT

For a single game frame, your programme should output on a single line, separated by spaces:

- Movement vector x coordinate (float)
- Movement vector y coordinate (float)
- Shooting vector x coordinate (float)
- Shooting vector y coordinate (float)

You may output vectors of any magnitude, although the magnitude will be clamped to 1 by the game code for the purpose of limiting acceleration to a maximum value.

You should output your floating point numbers in the same format used in the input data. You can do this with standard libraries in each of our contest languages as follows (in each case, outputting a float variable called 'x'):

C#

```
Console.WriteLine("{0:E}", x);
```

C++

```
cout.precision(6);
cout << scientific << x << endl;
```

EXAMPLE

This example shows sample input and output for a single game frame:

Sample input
1280 720 2.000000E+001 6.400000E+002 3.600000E+002 0.000000E+000 0.000000E+000 1.920000E+001 1 5 5.954958E+002 5.549856E+002 -4.450419E-001 1.949856E+000 5.120000E+001 5.153020E+002 5.163663E+002 -1.246980E+000 1.563663E+000 5.120000E+001 4.598062E+002 4.467767E+002 -1.801938E+000 8.677673E-001 5.120000E+001 4.400000E+002 3.600000E+002 -2.000000E+000 -1.611058E-007 5.120000E+001 4.598062E+002 2.732232E+002 -1.801938E+000 -8.677676E-001 5.120000E+001
Sample output
8.219957E+002 2.723555E+002 1.801938E+000 -8.677670E-001

SCORING

Entrants will be ranked according to the number of points they score within the game. We will then award scores in the contest as follows:

- 100 points for first place
- 50 points for second place
- 25 points for third place

TESTING YOUR SOLUTION

You will be provided with an interactive application with which to test and visualise your AI.

You will also be provided with a verification programme to run before submitting your programme. Verification involves running your programme on a set of test positions in the game. It will check that your programme produces output of the correct form, and will also make sure that your programme runs at a reasonable speed. In order for the game to run at a reasonable frame rate, we need each AI routine to take no more than 5ms in a typical run. The verification programme will use around 1000 test cases and compute an average speed over these.

Entries which do not pass the verification process will not be entered into the game.

We will provide a separate document detailing how to run the verification and visualisation software.

FURTHER DETAILS

Some other details of interest about the game:

- If your programme terminates for any reason, your ship will explode and be removed from the game. You will keep the number of points you have scored up to that point.
- Rocks do not have any acceleration – their velocity remains constant over their lifetime.
- Getting your ship to move as you want may be easiest to tune simply by trying out the interactive application, but here are some details anyway:
 - The movement input is scaled by 0.75 before being applied directly as acceleration each frame.
 - The player's velocity is multiplied by a drag factor of 0.95 each frame.
 - The player's gun can only shoot once every 5 frames.
 - The player's gun turret is rotated by 0.15 times the difference between its current facing angle and its desired angle (desired angle being the shooting input direction), each frame. If the shooting input is zero, the gun turret does not rotate.
- As far as units go, positions of game objects are given in pixels. Velocities and speeds are given as pixels per game frame (how this relates to time depends on how fast the game runs, but since all game objects behave consistently you shouldn't need to worry about this). Accelerations are just change in velocity per game frame.
- Rocks do not collide with one another.



This work is licenced under a [Creative Commons Licence](https://creativecommons.org/licenses/by-nc/4.0/).