CS1010E Practice Exercise: Predator and Prey (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Predator and Prey

Topic Coverage

- Assignment and expressions
- Nested control statements
- Functions and procedures

Problem Description

We would like to simulate the predator-prey behavior in animals. Each predator or prey moves around in a grid world. The prey wanders around aimlessly, while the predator targets the prey moving ever so closely to it.

The world is a `10 x 10` grid with each position represented as a coordinate `(x, y)`. The integer values of `x` and `y` ranges from `0` to `9`. The figure below depicts the world with the predator (uppercase `P`) and prey (lowercase `p`) are situated somewhere within the world.

```
   0   1   2   3   4   5   6   7   8   9
0  +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
1  +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
2  +---P---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
3  +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
4  +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
5  +---+---+---+---+---p---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
6  +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
7  +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
8  +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |
9  +---+---+---+---+---+---+---+---+---+---+
```

Each, time the prey will only move to one of its eight neighbouring positions. For example, if the prey starts off at `(5, 5)`, then it can move to any of the eight positions denoted by `*`.

```
   4   5   6
4  *---*---*
   |   |   |
5  *---O---*
```

```
   |   |   |
 6 *---*---*
```

The behavior of the predator is slightly more complex. In order to catch the prey, it follows the following strategy.

1. The predator first smells for the prey to determine how far away it is.
2. The predator then moves forward (forward is relative to the direction it is facing), and smells the prey again. If the scent is weaker (i.e. the distance between them is increased), it will make a right turn; otherwise keep the current facing direction.
3. Repeat steps 1 and 2 above until the predator catches the prey (i.e. the distance between them is zero).

The predator will always start off facing north, so if it is situated at `(1, 2)`, a forward move will bring it to `(1, 1)`. We represent the predator by its position `(x, y)` as well as its facing direction `(dx, dy)`. If the predator is north facing, `dx` and `dy` would be `0` and `-1` respectively. This allows the forward movement to be expressed as `(x+dx, y+dy)`.

The distance metric used is the city-block (or Manhattan) distance between the predator and the prey. This is the minimal number of steps needed for the predator to catch hold of the prey following the four major directions. For example, if the predator is located at `(1, 4)` and the prey at `(4, 2)`, then the distance is `5`.

```
   0   1   2   3   4   5
0  +---+---+---+---+---+-
   |   |   |   |   |   |
1  +---+---+---+---+---+-
   |   |   |   |   |   |
2  +---+---+---+---p---+-
   |   |   |   |   |   |
3  +---+---+---+---+---+-
   |   |   |   |   |   |
4  +---P---+---+---+---+-
   |   |   |   |   |   |
5  +---+---+---+---+---+-
   |   |   |   |   |   |
```

Task

Write a program that simulates the predator-prey behavior by first requesting the user for the `(x, y)` positions of both the prey and predator. The program then reads a sequence of ordered pairs `(dx, dy)` with $-1 \le dx \le 1$ and $-1 \le dy \le 1$ and that represents the movements of the prey to the neighboring eight locations given by $(x_{prey}+dx, y_{prey}+dy)$. Every move of the prey is followed by one move of the predator. The program stops when the predator catches the prey, i.e. their locations are the same.

This task is divided into several levels. Read through all the levels (from first to last, then from last to first) to see how the different levels are related. **You may start from any level.**

Level 1

Name your program `gotcha1.c`

Write a program that reads in four integers representing the x_{prey} and y_{prey} locations of the prey and x_{pred} and y_{pred} loc
Output the positions on separate lines.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a r

```
$ ./a.out
4 6
3 7
prey(4,6)
PRED(3,7)
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < gotcha.in | diff - gotcha1.out
```

To proceed to the next level (say level 2), copy your program by typing the Unix command

```
cp gotcha1.c gotcha2.c
```

Level 2

Name your program `gotcha2.c`

Write a program that reads in four integers representing the x_{prey} and y_{prey} locations of the prey and x_{pred} and y_{pred} loc
Output the positions on separate lines.

The program then reads in a series of **ten** pairs of values `dx` and `dy` representing the movements of the prey. The position
after each pair of values read. You may assume that the movement of the prey remains within the confines of the grid wor

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a

```
$ ./a.out
4 6
3 7
prey(4,6)
PRED(3,7)
1 0
prey(5,6)
-1 0
prey(4,6)
0 1
prey(4,7)
-1 -1
prey(3,6)
-1 -1
prey(2,5)
0 1
prey(2,6)
-1 -1
prey(1,5)
1 1
prey(2,6)
-1 -1
prey(1,5)
-1 -1
prey(0,4)
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < gotcha.in | diff - gotcha2.out
```

To proceed to the next level (say level 3), copy your program by typing the Unix command

```
cp gotcha2.c gotcha3.c
```

Level 3

Name your program `gotcha3.c`

Write a program that reads in four integers representing the x_{prey} and y_{prey} locations of the prey and x_{pred} and y_{pred} loc
Output the positions on separate lines.

The program then reads in a series of **ten** pairs of values `dx` and `dy` representing the movements of the prey. The position
after each pair of values read. You may assume that the movement of the prey remains within the confines of the grid wor

After every move of the prey, move the predator with respect to its current direction (facing north). If the move makes the
world, it remains at that position.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a

```
$ ./a.out
4 6
3 7
prey(4,6)
PRED(3,7)
```

```
1 0
prey(5,6)
PRED(3,6)
-1 0
prey(4,6)
PRED(3,5)
0 1
prey(4,7)
PRED(3,4)
-1 -1
prey(3,6)
PRED(3,3)
-1 -1
prey(2,5)
PRED(3,2)
0 1
prey(2,6)
PRED(3,1)
-1 -1
prey(1,5)
PRED(3,0)
1 1
prey(2,6)
PRED(3,0)
-1 -1
prey(1,5)
PRED(3,0)
-1 -1
prey(0,4)
PRED(3,0)
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < gotcha.in | diff - gotcha3.out
```

To proceed to the next level (say level 4), copy your program by typing the Unix command

```
cp gotcha3.c gotcha4.c
```

Level 4

Name your program `gotcha4.c`

Write a program that reads in four integers representing the x_{prey} and y_{prey} locations of the prey and x_{pred} and y_{pred} loc
Output the positions on separate lines.

The program then reads in a series of **ten** pairs of values dx and dy representing the movements of the prey. The positior
after each pair of values read. You may assume that the movement of the prey remains within the confines of the grid wor

After every move of the prey, move the predator with respect to its current direction (initially facing north) and **make it turı**
makes the predator leave the grid world, it remains at that position.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a r

```
$ ./a.out
4 6
3 7
prey(4,6)
PRED(3,7)
1 0
prey(5,6)
PRED(3,6)
-1 0
prey(4,6)
PRED(4,6)
0 1
prey(4,7)
PRED(4,7)
```

```
-1 -1
prey(3,6)
PRED(3,7)
-1 -1
prey(2,5)
PRED(3,6)
0 1
prey(2,6)
PRED(4,6)
-1 -1
prey(1,5)
PRED(4,7)
1 1
prey(2,6)
PRED(3,7)
-1 -1
prey(1,5)
PRED(3,6)
-1 -1
prey(0,4)
PRED(4,6)
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < gotcha.in | diff - gotcha4.out
```

To proceed to the next level (say level 5), copy your program by typing the Unix command

```
cp gotcha4.c gotcha5.c
```

Level 5

Name your program `gotcha5.c`

Write a program that reads in four integers representing the x_{prey} and y_{prey} locations of the prey and x_{pred} and y_{pred} loc
Output the positions on separate lines.

The program then reads in a series of **ten** pairs of values `dx` and `dy` representing the movements of the prey. The position
after each pair of values read. You may assume that the movement of the prey remains within the confines of the grid wor

After every move of the prey, move the predator with respect to its current direction (initially facing north) and make it turn
makes the predator leave the grid world, it remains at that position.

In addition, output the city-block distance between the predator and prey **before and after every move**. You may use the
`stdlib.h`.

`int abs(int x);`
Returns the absolute value of `x`.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a

```
$ ./a.out
4 6
3 7
prey(4,6)
PRED(3,7)
1 0
prey(5,6)
distance before PRED moves 3
PRED(3,6)
distance after PRED moves 2
-1 0
prey(4,6)
distance before PRED moves 1
PRED(4,6)
distance after PRED moves 0
0 1
prey(4,7)
```

```
distance before PRED moves 1
PRED(4,7)
distance after PRED moves 0
-1 -1
prey(3,6)
distance before PRED moves 2
PRED(3,7)
distance after PRED moves 1
-1 -1
prey(2,5)
distance before PRED moves 3
PRED(3,6)
distance after PRED moves 2
0 1
prey(2,6)
distance before PRED moves 1
PRED(4,6)
distance after PRED moves 2
-1 -1
prey(1,5)
distance before PRED moves 4
PRED(4,7)
distance after PRED moves 5
1 1
prey(2,6)
distance before PRED moves 3
PRED(3,7)
distance after PRED moves 2
-1 -1
prey(1,5)
distance before PRED moves 4
PRED(3,6)
distance after PRED moves 3
-1 -1
prey(0,4)
distance before PRED moves 5
PRED(4,6)
distance after PRED moves 6
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < gotcha.in | diff - gotcha5.out
```

To proceed to the next level (say level 6), copy your program by typing the Unix command

```
cp gotcha5.c gotcha6.c
```

Level 6

Name your program `gotcha6.c`

Write a program that reads in four integers representing the x_{prey} and y_{prey} locations of the prey and x_{pred} and y_{pred} loc
Output the positions on separate lines.

The program then reads in a series of pairs of values dx and dy representing the movements of the prey. You may assum
the prey remains within the confines of the grid world.

Compare the city-block distances between the predator and prey **before and after every move of the predator** and mak
distance becomes greater.

Generate an output for the locations after each pair of prey and predator moves.

The program ends after the predator catches the prey.

The following is a sample run of the program. User input is underlined. Ensure that the last line of output is followed by a

```
$ ./a.out
4 6
```

```
3 7
prey(4.6) vs PRED(3,7)
1 0
prey(5,6) vs PRED(3,6)
-1 0
prey(4,6) vs PRED(3,5)
0 1
prey(4,7) vs PRED(4,5)
-1 -1
prey(3,6) vs PRED(5,5)
-1 -1
prey(2,5) vs PRED(5,6)
0 1
prey(2,6) vs PRED(4,6)
-1 -1
prey(1,5) vs PRED(3,6)
1 1
prey(2,6) vs PRED(2,6)
GOTCHA!
```

Click here to submit to CodeCrunch.

Check the correctness of the output by typing the following Unix command

```
./a.out < gotcha.in | diff - gotcha6.out
```

Submission (Course)

Select course: CS1010E (2017/2018 Sem 1) - Programming Methodology ▼

Your Files:

SUBMIT (only .java, .c, .cpp and .h extensions allowed)

To submit multiple files, click on the Browse button, then select one or more files. The selected file(s) will be added to the upload queue. You can repeat this step to add more files. Check that you have all the files needed for your submission. Then click on the Submit button to upload your submission.