Project 02: BlackBox

Release date: 2/16
Due date: 3/2 by 11:59pm

Goals

- Practice using two-dimensional arrays
- Practice using methods to modularize code
- Practice using objects to represent abstract concepts (e.g. games)

Introduction

Your second project will be an implementation of the BlackBox game. For those unfamiliar with the game, you can read more about its rules on Wikipedia or try playing the game here. Our implementation will be a little different to make it a little easier and to adapt to the terminal interface.

Introduction to BlackBox

The objective of BlackBox is to deduce the location of multiple objects (referred to henceforth as "balls"). The player projects a line from a valid location on the edge of the grid, referred to as an entry port. This line interacts with the balls on the game board based on a set of rules (defined below). Using their knowledge of these interactions, the player's goal is to accurately guess the location of these balls using as few projections as possible.

The BlackBox Game Board

BlackBox is played with a square, n-by-n matrix. The number of rows and columns is set according to a difficulty level (case insensitive) chosen by the player before beginning play. The difficulty levels are shown here on the right.

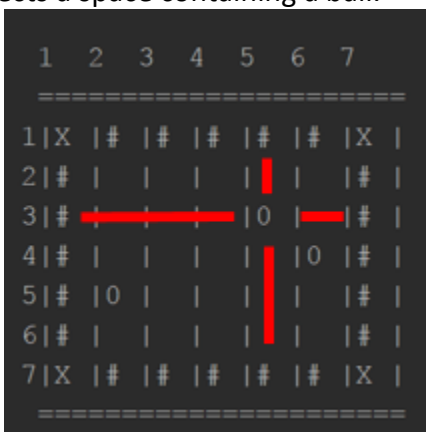
Difficulty	Dimensions
Easy	5 rows, 5 columns
Medium	7 rows, 7 columns
Hard	8 rows, 8 columns

The game board will have three hidden balls placed in any location. A sample of what your initial board should look like for an "easy" game is shown below.

Note there are 5 rows and 5 columns of unknown space where the balls may be hiding. Each "| |" potentially contains one of the balls.

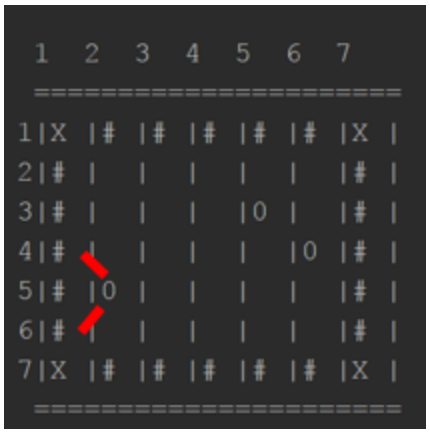
In this game board, the '#' characters are the entry ports the user will select to fire from. The line will follow a trajectory beginning at that '#' character toward the '#' character on the other side, subject to the rules of interaction defined for balls and lines. Upon reaching any side of the game board, the firing entry port and the receiving entry port can take on any one of the following states:

- **Hit (H)**: A hit is defined as a beam that does not leave the game board, which can only happen if a beam intersects a space containing a ball.

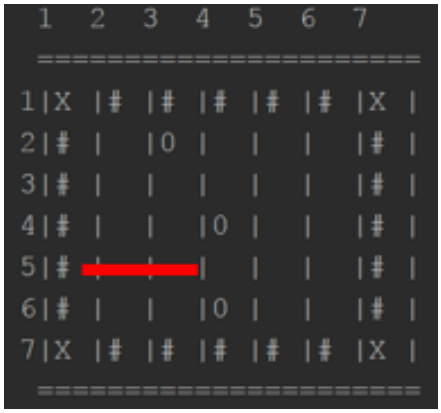


  Note that the balls are visible only for the purpose of this demonstration. In this diagram, firing from any of the (1,5), (3,1),(3,7) or (7,5) ports will result in a *hit*. Upon reaching the ball, the firing entry port will be replaced with an 'H' character.

- **Reflection (R)**: A reflection is defined as a beam that exits the game board through the same entry port that fired it. There are two scenarios in which a reflection may occur.

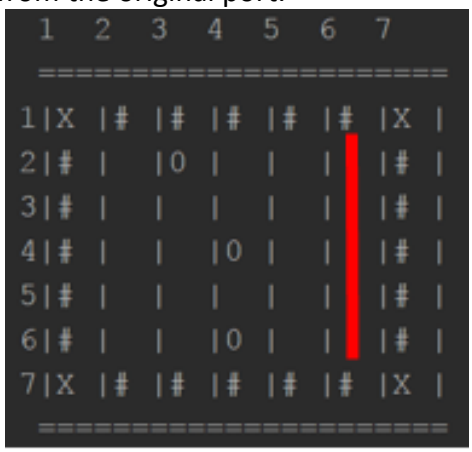


If a ball is located next to the axis of the firing entry port (no space between the axis and the ball) and is located in a space 45 degrees from the entry point. In the image above, entry ports (4,1) and (6,1) would satisfy this condition and be replaced by an ‘R’ character if fired from.



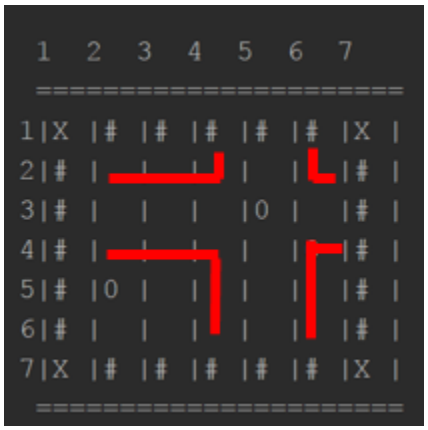
If two balls are one space away from each other and are either directly horizontal or vertical with respect to the other, any beam that enters the space immediately before the gap will be returned to its entry port. In the image above, entry port (5,1) would satisfy this condition and be replaced by an ‘R’ character if fired from.

- **Straight ray (integer)**: A straight ray is defined as a beam that exits the game board after following a straight line from the original port.



In the image above, either beam fired from (7,6) or (1,6) would be considered straight and both the firing and receiving ports would be replaced by an integer showing that this was the nth straight ray or deflection. The first straight ray or deflection would replace the corresponding ports with a '1', the next with a '2', and so on.

- **Deflection (integer)**: A deflection is defined as a beam that exits the game board in any manner other than those mentioned above.



In the image above, any beam that enter the space directly diagonal to a ball will change direction and move toward the entry port that is neither in the direction of the ball nor the original entry port. Any beam fired from (2,1), (1,4), (1,6), (2,7), (4,7), (7,4), (7,6), or (4,1) will result in a deflection and the firing and receiving ports will be replaced by an integer following the same rules for the straight line. Note that deflections and straight rays are not counted separately.

It is important to note that the beam will not always follow a direct trajectory: a beam may deflect then reflect, ultimately resulting in a reflection. You should consider which entry port is firing the beam and which is receiving the beam when classifying the port.

Beam Rules of Interaction

The rules of interaction between beams and balls can be determined from the above images, but to help you determine the interactions these rules are defined below. These rules are displayed in order of decreasing precedence.

1. When there is a ball on the beam's path, the beam never exits through a port.
2. When there is a ball at the diagonal edge of the grid it prevents a ray from entering the grid and will be reflected back to the port from which it entered.
3. When the beam reaches two balls adjacent to the ray shown (see **Reflection**) in the beam will return to the entry port from which it was fired.
4. When from the left or from the right side in front of the beam stands a ball, the beam will be deflected by 90 degrees away from the ball.

Guessing

The player must correctly guess the location of the three balls on the board in order to receive a score. Up to three locations may be selected for guesses at any given time in play. A player guesses by inputting the coordinates of the guess. For example, a player guessing that a ball exists at (2,3) is shown in the demonstration at the end of this document. An '*' character will be placed in the corresponding space.

Guesses are not finalized until the user enters "submit". Once the user enters "submit", the program must check if three guesses have been made. If true, the location of the balls the user failed to find will be marked with a '0' (zero) character. Otherwise, the game will not allow the player to submit.

Scoring

The user's score will be equal to the number of moves made by the user, including guesses and repeated entries (entering submit will not add to their score). After submitting with three guesses made, the game will determine whether the player receives a score based on the accuracy of their guessing. Players that did not guess all the balls correctly will not receive a score. Players that do guess all the balls correctly will receive a score.

The *high score* will be displayed after the completion of each game. If the user failed to correctly identify the location of each ball in each of the previous games, the high score will be "none". Otherwise, the high score will be the lowest score achieved (note the mismatch in naming schemes, just following arcade conventions).

Input

You will not be required to perform input validation: you may assume the user will enter a comma separated integer pair with no spaces followed by pressing the enter key.

The user must be able to enter "quit" to end the program and "submit" to submit their guesses (case insensitive).

Formatting

Your game board should be formatted such that a player can reasonably interpret the game. There will be no test cases examining formatting: we will test your program manually for evaluating how the game plays, but there will be some automatic testing of methods that can be easily evaluated without printing.

Ball Placement

Balls should be placed randomly for each playthrough. You do not have to match a specific procedure for placing these, as the tests will not specify layouts.

Skeleton Outline and Methods to Be Written

You will be responsible for the implementation and usage of the following methods in your program. Note that unless otherwise specified, **all methods should be non-static and public**.

Constructors to be written

Method Name	Description
BlackBox()	The default constructor to set all the integer variables to 0, initialize box[][] to a char array of size 0 and to set the Boolean end to false.
BlackBox(int size, int numball,int numlink,boolean end,int score)	The parameterized constructor to set the value of the variables to variables passed as parameters and to initialize the box[][] with size rows and columns. We will use this method to test your solution on Vocareum.

Getters and setters to be written

Method Name	Description
char[][] getbox()	Returns box, an n-by-n array containing the '#' and 'X' characters, the location of the balls, and the location of the player guesses. Note that for "easy", box should be 7-by-7 to accommodate the '#' and 'X' characters.
int getscore()	Returns the current score, the number of moves the player has made.
int getNumball()	Returns numball, the number of balls remaining for the player to guess (initially 3).
int getNumlink()	Returns numlink, the counter for the number of straight rays and deflections (initially 0)
boolean getend()	Returns end, a variable signaling whether the current game has ended (user has submitted with three guesses)
int getSize()	Returns the size of the game board, which is the length of either side.
void setbox(char box[][])	Sets the box to the value given in the parameter.
void setSize(int size)	Sets the size to the value given in the parameter.
void setNumball(int Numball)	Sets the numball to the value given in the parameter.
void setNumlink(int Numlink)	Sets the numlink to the value given in the parameter.
void setEnd(boolean end)	Sets the end to the value given in the parameter.

void setScore(int score)	Sets the score to the value given in the parameter.

Functions to be written

Method Name	Description
public static void main()	The main function takes input for the difficulty level, creates an instance of BlackBox, and calls the methods initialize() and playgame()
void initialize()	The initialize prepares the box[][] for the game. It Places 3 random balls.
void playgame()	The playgame method is the main controller and input place for the game. It calls check() and printbox() when needed.
void printbox()	The printbox method prints out the matrix in a particular format as given in the example.
void check(int i,int j)	The check method takes in the row and column in the matrix. If the row and column corresponds to an entry port, it checks for a hit, reflection, deflection or straight ray (calling the corresponding methods). Otherwise, it places a ball.
boolean placeball(int i,int j)	The placeball method checks if a guess ball can be placed at a given row and column. Return false if a ball is already there, else it returns true.
boolean hitcheck(int i,int j)	The hitcheck method checks if a beam projected from from row i, column j results in a hit. Return true if the beam ends in a hit. Otherwise, return false. If it is a hit occurs, replace the projecting entry port with an 'H' in place of '#'.
boolean reflectionCheck(int i,int j)	The reflectioncheck method checks if a beam projected from from row i, column j results in a reflection. Return true if the beam ends in a reflection. Otherwise, return false. If it is a reflection occurs, replace the projecting entry port with an 'R' in place of '#'.
boolean deflectionCheck(int i,int j)	The deflectioncheck method checks if a beam projected from from row i, column j results in a deflection. Return true if the beam results in a deflection. Otherwise, return false. If it is a deflection occurs, replace the projecting

	and exiting entry ports with the current value of numlink in place of '#'.
boolean straightRay(int i,int j)	The straightRay method checks if a beam projected from from row i, column j results in a straightRay. Return true if the beam results in a straightRay. Otherwise, return false. If it is a straightRay occurs, replace the projecting and exiting entry ports with the current value of numlink in place of '#'.

Submission

Items required for submission via Blackboard:

- BlackBox.java

Rubric

There will be an automatic grading and a manual grading component for this assignment. If manual grading for automatically graded portions will be allowed after the deadline.

Automatically Graded (50 points)

- (5 points) Game board is properly initialized for each difficulty level (testing the initialize method)
- (5 points) Game allows the placement of only three balls (testing the initialize and check methods)
- (10 points) Game properly checks entry ports for hits (testing the check method)
- (10 points) Game properly checks entry ports for reflections (testing the check method)
- (10 points) Game properly tracks the number of straight rays and deflections (testing the check method)
- (10 points) Game properly calculates player score (testing the check method)

Manually Graded (50 points)

- (50 points) Gameplay follows guidelines specified here in the handout.