CIS 15 Spring 2009, Assignment II

Instructions

- This is the assignment for Unit II.
- It is worth 10 points.
- **It is due on Sunday February 22nd** and must be submitted by email (as below).
- **Follow these emailing instructions:**
  1. Create a mail message addressed to **parsons@sci.brooklyn.cuny.edu** with the subject line **CIS15 HW2**.
  2. Write your name, that is the name under which you registered for the course, in the email (when I get an email from `deathmetal@aol.com` or `pinkprincess@yahoo.com`, I can usually guess whose program it is, but that is not as good as *knowing* whose program it is).
  3. Attach ONLY the **.cpp** source code file created below.
  4. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions . . . (which can make it a lot harder for me to grade your work)

Program description

For this assignment, you will develop a program that simulates a robot playing soccer. The robot moves around a 2-dimensional field and its current location is represented by an (x, y) point in space. Since we use integer values for x and y, we can think of the field as being a grid.

Now, the brain of the robot is very simple. It can only hold one "instruction" at a time. Each instruction will either tell it to move forward or turn 90 degrees, or try to kick the ball. It can turn only clockwise. The program that implements this will have a broad *control loop*, and for each iteration of the loop, the robot must decide what to do (i.e., which command to execute). The aim is for the robot to locate the ball and kick it into a goal. The field is 12×8 units in size, and the x and y coordinates are in the range $[0, 11]$ (i.e., from 0 to 11, inclusive) and $[0, 7]$ respectively.

To model the robot, you will create four classes, each with multiple data and function members, and a `main()`. Each class is described in detail below, with step-by-step instructions for developing the various components of each class and testing the components individually. In the end, you'll write the `main()` and put all the pieces together—and this final product is what you'll submit.

The intermediary test programs are for your benefit as a developer and should not be submitted. These are called *unit tests* and are created to let you test and debug the individual units of a complex program. By the end of the semester, you should have gained the skills and experience to devise and construct your own unit tests, without me giving you step-by-step instructions.

A. Create and test a `point` class

1. Create a `point` class, as we did during the class on February 11th.

   The class should have two `int` data members: x and y. The class should have four function members:

   - `void print() const;`
     This function prints out the values of the data members x and y.
   - `void set(int u, int v);`
     This function sets the values of the data members x and y.

- `int getX()`
  This function returns the value of the x data member.
- `int getY()`
  This function returns the value of the y data member.

2. Create a `main()` to test this class. This should read in two values from the user, store them in an instance of the `point` class, and then print them out.

B. Create a `ball` class

1. Create a class called `ball` that has one data member and two function members:
   - the data member is a `point` object, called `location` which holds the location of the ball.
   - The first function member is `void print() const;`
     This function prints out the location of the ball.
   - The second function member is `void set(int u, int v);`
     This function sets the location of the ball to (u, v). You can do this using the `set` function of the data member `location`.
2. Modify the `main()` that you created in part A to instantiate a `ball` object and then to take the values from the user and set the location of the ball to this position. Again, this `main()` is for unit testing.

C. Build the soccer field

1. Create a class called `field` that has one data member and six function members:
   - The data member is a `ball` object.
   - The first function member is a constructor function that sets the location of the ball member to a random location. This location should be on the field, so the x value should be between 0 and 11, and the y value should be between 0 and 7.
   - The second function member is `void printBall() const;`
     This function prints out the location of the ball
   - The third function member is `void setBall(int u, int v);`
     This function sets the location of the ball to (u, v). You can do this using the `set` function of the `ball` object.
   - The fourth function member is `int getBallX()`
     This function returns the x value of the `ball` member.
   - The fifth function member is `int getBallY()`
     This function returns the y value of the `ball` member.
   - The sixth function member is `bool isGoal()`
     This function returns true if the ball enters one of the goals.
     The goals are areas on the pitch. One goal is is the pair of grid squares $(0, 3)$ and $(0, 4)$ and other goal is the pair of squares $(11, 3)$ and $(11, 4)$.
     In other words, `isGoal` returns true if the ball has a location of $(0, 3)$, $(0, 4)$, $(11, 3)$ or $(11, 4)$
2. Modify the `main()` that you created in part B to instantiate a `field` object and allow the user to repeatedly enter positions for the ball and test whether the ball is in the goal. Have your program print out `GOOOOOOOOOOOOOOOOAL!` if the ball is in the goal. Again, this `main()` is for unit testing and is not to be handed in.

D. Create the robot

1. Create a `robot` class that has the following data and function members:

   - A `point` object to store the robot's current location in the world
   - An enumerated data type, called `orientation_type`, that defines the four directions that the robot could be facing (north, south, east or west), i.e., its "orientation"
   - A variable to store the robot's current orientation
   - A constructor that initialises the robot to a random location and orientation (remember that the enumerated data values are just integers).
   - `void print() const;`
     This function prints the robot's current location and orientation in a pretty format, such as:

     ```
     I am at (4,4) and I am facing east.
     ```

   - `bool forward();`
     This function simulates the robot moving forward one step in the direction that it is facing. It checks to make sure that the robot is not trying to move outside the field. It returns `true` if the robot moves forward successfully and `false` if the robot is at the edge of the field and cannot move forward.
     This function calls `print` to report the location of the robot after it has tried to move.
   - `void turn();`
     This function changes the robot's orientation, simulating a turn in the clockwise direction.
     This function calls `print` to report the location of the robot after it has turned.
   - `bool kick();`
     This function is an attempt to kick the ball. If the robot is at the same location as the ball, then `kick` moves the ball one step in the direction that the robot is facing and the function returns `true`.
     If the ball leaves the field as a result of the kick, it is replaced in a random location on the field.
     If the robot `kicks` when the robot is not at the same location as the ball, then `kick` returns `false` and the ball does not move.

2. Now modify the `main()` from the previous step to instantiate a `field` object and a `robot` object. Define and perform some unit tests on each of the function members in the `robot` class. For example, call `print()` to make sure the robot's position and orientation are correctly initialized. Or, call `forward()` to verify that `forward()` works as you expect. Do the same with `turn()`, then try `kick()` with the robot in the same location as the ball and in a different location from the ball. Again, this `main()` is for your testing, not for submission.

D. Try to score a goal

Finally, you need to program the robot to try to score a goal. You make the design decision about how to do that, but minimally have the robot head for the ball and kick it. (Hint: The robot can get the location of the ball from the `field` object and reduce the difference between its x and y and the ball's x and y.)

Every time the robot kicks the ball, it should print out:

```
Yeah, I kicked the ball!
```

and when it scores a goal, it should print out:

```
GOOOOOOOOOOOOOOOOAL!
```

Keep a count of how many moves the robot makes, and when the program exits, print a message saying how many moves the robot made before scoring the goal.

E. For additional credit

You will get additional credit for any of these that you complete:

- Make sure the robot doesn't kick the ball off the field.
- Give the robot a notion of which "team" it is on, and make sure it only kicks the ball towards the right goal for its team.
- After each goal, restart the game with the robot at a random location in its half of the pitch (its team determines which is "its half"), and the ball at $(5, 4)$.
- After 200 steps, it is half time, and the robot changes team. After 400 steps it is the end of the game. Display the score (how many goals were scored by each team).

G. Marking rubric

This assignment is worth 10 points. The breakdown is as follows:

- `point` class with two data members (x and y) and four function members (print(), set(), getX() and getY())
  (1 point)
- `field` class with one data member (ball) and six function members (the constructor, printBall(), setBall(), getBallX(), getBallY(), and isGoal())
  (2 points)
- `robot` class with two data members, enumerated type definition, and 5 function members (the constructor, print(), forward(), turn(), and kick())
  (2 points)
- decision making for the robot that takes it towards the ball and kicks it.
  (2 points)
- determining if the ball is kicked, reporting when it has been kicked, determining if a goal has been scored.
  (2 points)
- counting and reporting the number of moves made by the robot
  (1 point)