Following The Green Goblin

Booleans and Their Friends

Tuesday, October 2, 2007

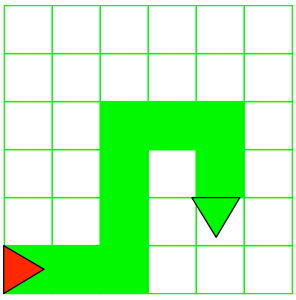
CS111 Computer Programming

Department of Computer Science
Wellesley College

On the Trail of the Green Goblin

Oh, no! The Green Goblin has escaped!

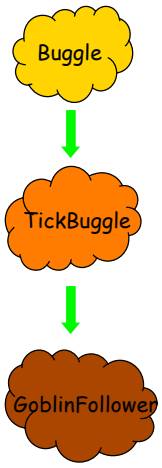
How can Spidey follow his trail?



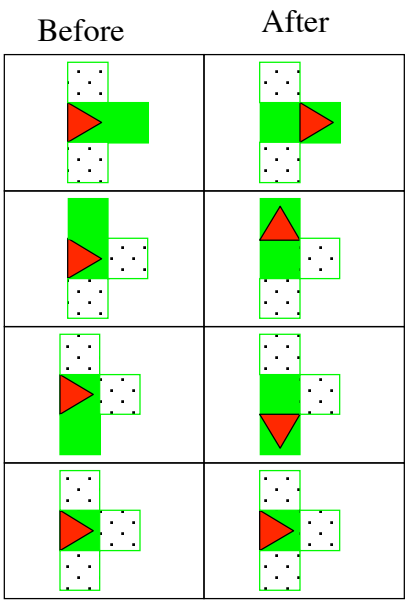
(Assume goblin never doubles back
or crosses/touches his own path.)

GoblinWorld

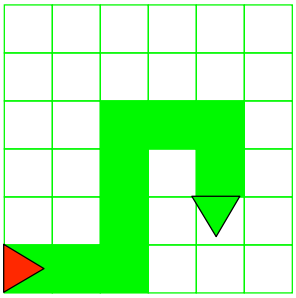
```
public class GoblinWorld extends BuggleWorld {
  public void run () {
    GoblinFollower spidey = new GoblinFollower();
    spidey.tick64();
  }
  …
}

public class GoblinFollower extends TickBuggle {

  public void tick() {
    followGoblin();
  }

  public void followGoblin() {
    // code to move one step along the
    // Green Goblin's trail
  }

}
```



Movement Rules for followGoblin()



(relative to current Buggle orientation)



stands for a blank cell or a wall

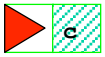
Wishful Thinking

Suppose we have a method with the following specification:

```
public boolean isFacingTrail (Color c);
```

Returns true if this buggle is facing a cell filled with color c; otherwise returns false. Leaves the state of the buggle unchanged.

Pictorial case analysis:



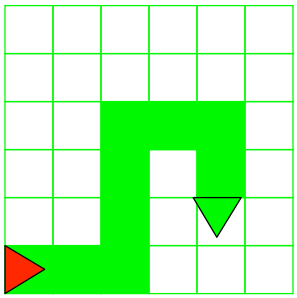
`true`

`false` (c2 different from c)

`false`

How can we use isFacingTrail() to define followGoblin()?

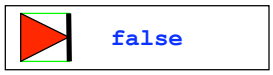
followGoblin()

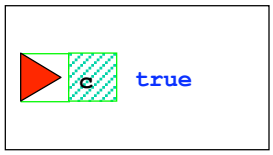


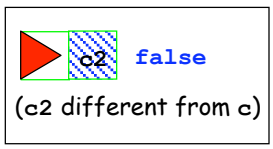
```
public void followGoblin() {
    if (isFacingTrail(Color.green)) {
        forward();
    } else {
        left();
        if (isFacingTrail(Color.green)) {
            forward();
        } else {
            right(); right(); // Turn right of initial heading
            if (isFacingTrail(Color.green)) {
                forward();
            } else { // I must be right behind him!
                left(); // Return to initial heading
            }
        }
    }
}
```

isFacingTrail(Color c)

```
public boolean isFacingTrail(Color c) {
  if (isFacingWall()) { // wall in front
    return false;
   } else { // cell in front of me
     brushUp();
     forward();
     if (getCellColor().equals(c)) {
          // compare colors with .equals()
       backward(); // undo forward();
       brushDown(); // undo brushUp();
       return true;
     } else {
       backward(); // undo forward();
       brushDown();// undo brushUp();
       return false;
     }
   }
}
```







Boolean Expressions & Declarations

- Like int, float, double, and char, boolean is a Java primitive data type.
- A boolean is one of two values: true or false.
- We have already seen many examples of boolean expressions:

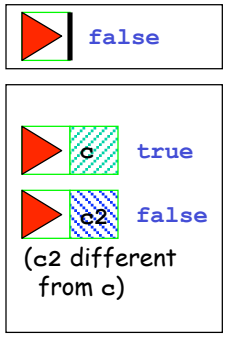
```
true
becky.isOverBagel()
peterParker.isFacingWall()
spidey.isFacingTrial(Color.green)
```

- We can declare boolean variables like any other variables:

```
boolean flag = true;
boolean result = getCellColor().equals(c);
boolean atWall;
atWall = greenGoblin.isFacingWall();
```

A More Concise isFacingTrail(Color c)

```
public boolean isFacingTrail(Color c) {
  if (isFacingWall()) {
    return false;
  } else { // cell in front of me
    brushUp();
    forward();
    boolean result = getCellColor().equals(c);
    backward(); // undo forward();
    brushDown(); // undo backward();
    return result;
    }
}
```



Testing for equality in Java

- Use == for primitive types (int, double, float, char, boolean)
  For example:

1 == 1	true
1 == 2	false
false == false	true
true == false	false

- Object equality can be tested in two ways:

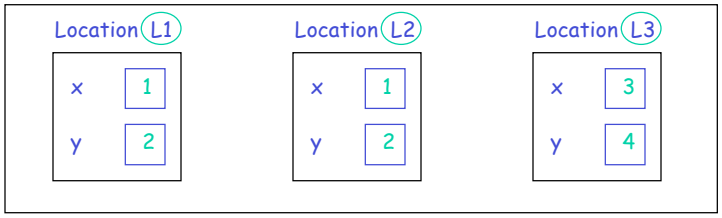
  1) == tests object identity: do two object references refer to the same object in ObjectLand?

  2) .equals() tests structural equality of objects: do two objects have the same state?

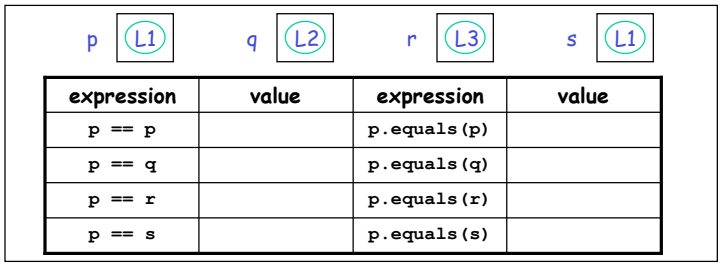
  - two location are .equals() if they have the same x and y coords
  - two colors are .equals() if they have the same red/green/blue components
  - two buggles are .equals() if they have the same state for their positions, headings, colors, and brushDowns --- i.e., if their positions, headings, and colors are .equals() and their brushDowns are ==.

Example: == vs. .equals() for Locations

ObjectLand



A Frame in ExecutionLand



expression	value	expression	value
`p == p`		`p.equals(p)`	
`p == q`		`p.equals(q)`	
`p == r`		`p.equals(r)`	
`p == s`		`p.equals(s)`	

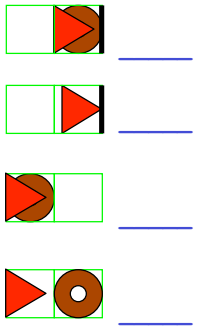
Other Ways to Compare Numbers

`<`	(less than)
`<=`	(less than or equal)
`!=`	(not equal)
`>=`	(greater than or equal)
`>`	(greater than)

Expression	Value
3 < 4	
3 < 3	
3 <= 3	
3 != 3	
3 != 4	
5 >= 5	
5 > 5	
5 > 4	

The Conjunction (AND) Operator: && †

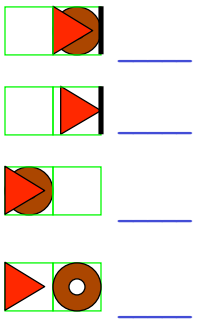
- The expression exp1 && exp2 is true precisely when the expression exp1 is true AND the expression exp2 is true.
- For example, `isFacingWall() && isOverBagel()`



†A single & means something else

The Disjunction (OR) Operator: || †

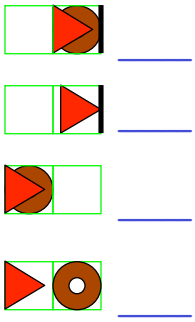
- The expression exp1 || exp2 is true precisely when either expression exp1 is true OR expression exp2 is true OR both.
- For example, `isFacingWall() || isOverBagel()`



†A single | means something else

The Negation (NOT) Operator: !

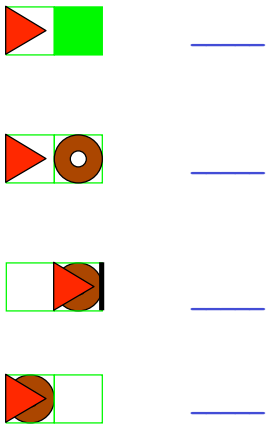
- The expression ! exp is true precisely when the expression exp is false.
- For example, `!isFacingWall()`



These operators can be combined

```
!isFacingTrail(Color.green)
&& isOverBagel()
```

```
!(isFacingTrail(Color.green)
  || isFacingWall())
```



Boolean expressions can be very expressive

```
boolean honor = ((class == 2008) && (gpa >= 3.75))
                || ((class == 2009) && (gpa >= 3.8));


if ((betty.getPosition().x == 1)
    && (betty.getPosition().y == 2)) {
  betty.setColor(Color.green);
} else {
  betty.setColor(Color.blue);
}

return (becky.getColor().equals(Color.red))
       || (becky.getColor().equals(Color.blue))
       || (becky.getColor().equals(Color.yellow));


bob.setBrush((! (bernice.getHeading().equals(Direction.EAST)))
             && ((becky.getHeading().equals(Direction.WEST))
                 || (betty.getHeading().equals(Direction.SOUTH))));
```


Gotchas (common bugs to avoid)

Don't write: `if (xcoord = 1) …`

Do write: `if (xcoord == 1) …`

Don't write: `betty.getLocation() == new Location(1,1)`

Do write: `betty.getLocation().equals(new Location(1,1))`

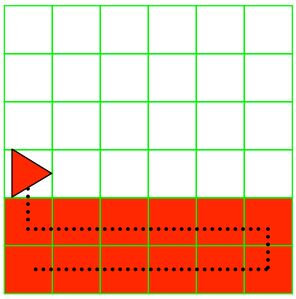
Also, avoid == for equality of colors, directions, strings, …

Don't write: `col.equals(Color.red || Color.blue)`

Do write: `col.equals(Color.red) || col.equals(Color.blue)`

SnakeWorld

Want Sandra buggle to traverse every cell in BuggleWorld in a snake-like fashion*, starting from (1,1) facing EAST.



* There is an English word for this traversal pattern: **boustrophedonic**

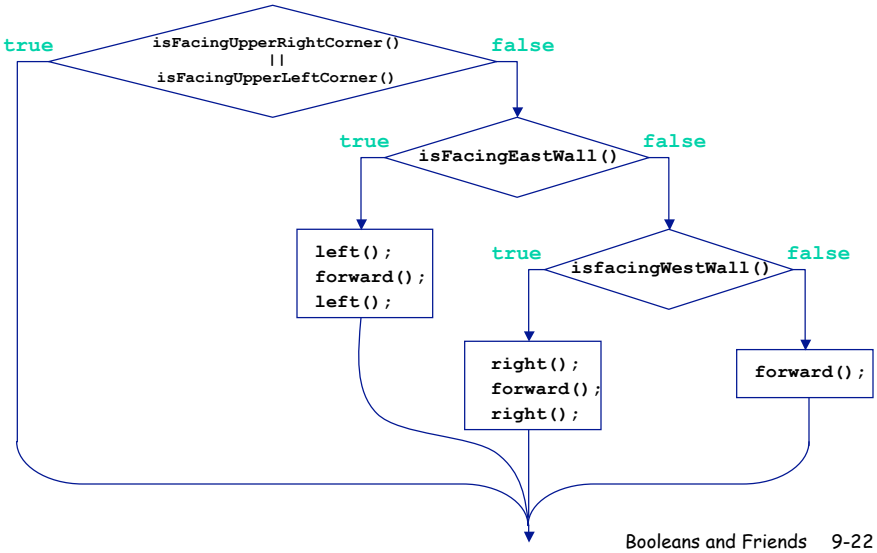
SnakeWorld in Java

```
public class SnakeWorld extends BuggleWorld {
  public void run() {
    SnakeBuggle sandra = new SnakeBuggle();
    sandra.tick128();
  }
}

public class SnakeBuggle extends TickBuggle {

  public void tick() {
      snakeStep();
  }

  public void snakeStep() {
    // code to move one step in snake-like
    // fashion goes here.
  }

}
```

Control Rules for snakeStep()

State	Action
top right corner, facing EAST 	do nothing
top left corner, facing WEST 	do nothing
at wall, facing EAST 	
at wall, facing WEST 	
no wall 	move forward

Control Diagram for snakeStep()



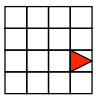
Writing snakeStep() in Java

```
public class SnakeBuggle extends TickBuggle {

  public void snakeStep() {
    if (isFacingUpperRightCorner() || isFacingUpperLeftCorner()) {
      // do nothing
    } else if (isFacingEastWall()) {
      left();
      forward();
      left();
    } else if (isFacingWestWall()) {
      right();
      forward();
      right();
    } else { // not facing any wall
      forward();
    }
  }
  // Need to add auxiliary methods here
  ...
}
```


Auxiliary methods

```
public boolean isFacingEastWall() {
  return isfacingWall()
         && (getHeading().equals(Direction.EAST));
}
// isFacingWestWall() is similar
```

facing east wall



```
public boolean isFacingUpperRightCorner() {
  return isfacingEastWall() && isWallToLeft();
}
// isFacingUpperLeftCorner() is similar
```

facing upper right corner

```
public boolean isWallToLeft() {
  left();
  boolean result = isFacingWall();
  right();
  return result;
}
// isWallToRight() is similar
```

A More Efficient, But Less Clear, Solution

```
public void snakeStep() {
    if (isFacingWall()) {
        if (getHeading().equals(Direction.EAST)) {
            left();
            if (isFacingWall()) { // top right corner
                right();          // reset direction to EAST
            } else {        // facing EAST at wall
                forward();
                left();
            }
        } else { // must be facing WEST
            right();
            if (isFacingWall()) { // top left corner
               left();            // reset direction to WEST
            } else {        // facing WEST at wall
               forward();
               right();
            }
        }
    } else { // no wall
        forward();
    }
}
```


Standard Simplifications

Suppose BE is any boolean expressions and S is any sequence of statements. Strive to make the following simplifications in your code:

Unnecessarily Complex Expression	Simpler Expression
BE == true	BE
BE == false	! BE
if (BE) {return true;} else {return false;}	return BE;
if (BE) {return false;} else {return true;}	return ! BE;
if (BE1) {return BE2;} else {return false;}	return BE1 && BE2;
if (BE1) {return true;} else {return BE2;}	return BE1 || BE2;
boolean result = BE; return result;	return BE
if (BE) {S; return true;} else {S; return false;}	boolean result = BE; S; return result

Example: Simplifying .equals() for Location

```
public boolean equals (Location loc) {
  if (this.x == loc.x) {
    if (this.y == loc.y) {
      return true;
    } else {
      return false;
    }
  } else {
    return false;
  }
}
```

simplify

```
public boolean equals (Location loc) {
  if (this.x == p.x) {
    return (this.y == p.y)
  } else {
     return false;
  }
}
```

simplify

```
public boolean equals (Location loc) {
  return (this.x == p.x) && (this.y == p.y);
}
```