**Intro**

Here we have a simulator, in which a very simple robot lives in a very simple world. By giving Robot a set of commands, you can direct it to perform certain tasks within its world.

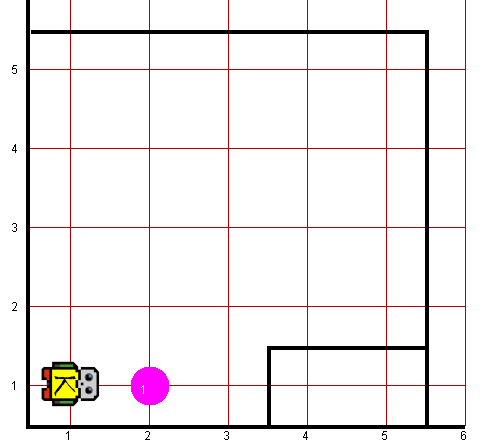
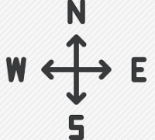
Robot understands only a very small number of predefined commands. The details are easy to master. Even so, you will discover that solving a problem can be extremely challenging.

By starting with Robot, you can concentrate on solving problems from the very beginning. And because Robot encourages imagination and creativity, you can have quite a lot of fun along the way.

**Robot’s world**

Robot’s world is defined by **streets** running horizontally (east-west) and **avenues** running vertically (north-south). The intersection of a street and an avenue is called a **corner**.

Robot can only be positioned on corners and must be facing one of the four standard compass directions (north, south, east, west). A sample Robot world is shown below. Here Robot is located at the corner of 1st Street and 1st Avenue, facing east.



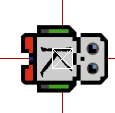
Several other components of Robot’s world can be seen in this example. The object in front of Robot is a **beeper**. Robot can only detect a beeper if it is on the same corner. The solid lines in the diagram are **walls**. Walls serve as barriers within Robot’s world. Robot cannot walk through walls and must instead go around them. Robot’s world is always bounded by walls along the edges, but the world may have different dimensions depending on the specific problem Robot needs to solve.

**What can Robot do?**

Robot responds to a very small set of commands:

|  |  |
| --- | --- |
| **Built-in** **COMMANDS** | **DESCRIPTION** |
| move() | Asks Robot to move forward one block. Robot cannot respond to a move() command if there is a wall blocking its way. |
| pickBeeper() | Asks Robot to pick up one beeper from a corner and stores the beeper in its beeper bag, which can hold an infinite number of beepers. Robot cannot respond to a pickBeeper() command unless there is a beeper in the current corner. |
| putBeeper() | Asks Robot to take a beeper from its beeper bag and put it down on the current corner. Robot cannot respond to a putBeeper() command unless there are beepers in its beeper bag. |
| turnLeft() | Asks Robot to rotate 90 degrees to the left (counterclockwise). |
| turnRight() | Asks Robot to rotate 90 degrees to the right (clockwise). |
| turnAround() | Asks Robot to rotate 180 degrees |

It is also important to recognize that several of these commands place restrictions on Robot’s activities. If Robot tries to do something illegal, such as moving through a wall or picking up a nonexistent beeper, an error condition occurs. At this point, Robot dies and does not execute any remaining commands. Dead Robot is grayed out with a cross(x).



To prevent the robot from dying and ensuring that Robot solves the problem completely, you need to use control statements.

**Control Statements**

Statements that affect the order in which a program executes are called control statements. Control statements generally fall into the following two classes:

1. *Conditional statements*: Conditional statements specify that certain statements in a program should be executed only if a particular condition holds. In Robot, you specify conditional execution using an **if** statement.
2. *Iterative statements*: Iterative statements specify that certain statements in a program should be executed repeatedly, forming what programmers call a loop. Robot supports two different iterative statements: a **for** statement that is useful when you want to repeat a set of commands a predetermined number of times and a **while** statement that is useful when you want to repeat an operation as long as some condition holds.

In case you want to check some condition before acting on a command, you can use if statement:

if (*conditional test*) {

*statements to be executed if the condition is true*

}

In some cases, you need to choose between two alternative courses of action. For these cases, the **if** statement in Robot has an extended form that looks like this:

if (*conditional test*) {

*statements to be executed if the condition is true*

} else {

*statements to be executed if the condition is false*

}

A for statement, which specifies that you want to repeat some operation a predetermined number of times. The structure of the for statement appears complicated primarily because it is actually much more powerful than anything Robot needs. The only version of the for syntax that Robot uses is

for (int i = 0; i < count; i++) {

*statements to be repeated*

}

A while statement has the following general form:

while (conditional test) {

*statements to be repeated*

}

You will have to use the **nested** versions of these control statements as well

Below are the conditions that you can use in control statement constructs.

|  |  |  |
| --- | --- | --- |
| **CONDITIONS** | | **DESCRIPTION** |
| beepersInBag() | noBeepersInBag() | Are there beepers in Robot’s bag? |
| beepersPresent() | noBeepersPresent() | Any beepers on this corner? |
| facingEast() | notFacingEast() | Is Robot facing east? |
| facingNorth() | notFacingNorth() | Is Robot facing north? |
| facingSouth() | notFacingSouth() | Is Robot facing south? |
| facingWest() | notFacingWest() | Is Robot facing west? |
| frontIsClear() | frontIsBlocked() | Is there a wall in front of Robot? |
| leftIsClear() | leftIsBlocked() | Is there a wall to Robot’s left? |
| rightIsClear() | rightIsBlocked() | Is there a wall to Robot’s right? |

**Quick Reference Card**

|  |  |
| --- | --- |
| **Built-in commands:**  move();  pickBeeper();  putBeeper();  turnLeft();  turnRight();  turnAround(); | **Conditional Statements:**  **if (*condition*) {**  ***statements executed if condition is true***  }  if (***condition***) {  ***statements executed if condition is true***  } else {  ***statements executed if condition is false***  } |
| **Robot program structure:**  /\* \* Comments may be included anywhere in  \* the program between a slash-star and  \* the corresponding star-slash characters.  \*/  import robotResources.NGCRobot;  /\* Definition of the new class \*/  public class ***name*** extends NGCRobot {  constructor added by default  public void run() {  ***statements in the body of the method***  }  ***definitions of private methods***  } |
| **Iterative statements:**  for (int i = 0; i < ***count***; i++) {  ***statements to be repeated***  }  while (***condition***) {  ***statements to be repeated***  } |
| **Condition names:**  frontIsClear() frontIsBlocked()  leftIsClear() leftIsBlocked()  rightIsClear() rightIsBlocked()  beepersPresent() noBeepersPresent()  beepersInBag() noBeepersInBag()  facingNorth() notFacingNorth()  facingEast() notFacingEast()  facingSouth() notFacingSouth()  facingWest() notFacingWest() |