AP Computer Science Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Karel Chapter 5 Homework #2 Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Period: \_\_\_\_\_\_\_

1. Write a new predicate leftIsBlocked that determines if there is a wall exactly one-half block away on a robot’s left. Be sure that when it terminates, the robot is on the same corner and facing in the same direction.

**public** **boolean** leftIsBlocked() {

World.*setDelay*(0);

setVisible(**false**);

turnLeft();

**if**(!frontIsClear()) {

turnRight();

setVisible(**true**);

World.*setDelay*(20);

**return** **true**;

} **else** {

turnRight();

setVisible(**true**);

World.*setDelay*(20);

**return** **false**;

}

}

2. Look at the following instruction. Is there a simpler, execution equivalent instruction? If so, write it down; if not, explain why. Hint: A simplifying transformation for the IF may prove useful. Common sense helps too.

if( ! nextToABeeper() )

{

move();

}

else

{

move();

}

move();

3. Assume that a Prospector robot is on a corner with either one or two beepers. Write a new method that commands the robot to face north if it is started on a corner with one beeper and to face south if it is started on a corner with two beepers. Besides facing the robot in the required direction, after it has executed this method there must be no beepers left on the corner. Name this method findNextDirection.

**public** **void** faceNorth() {

**while**(!facingNorth()) {

turnLeft();

}

}

**public** **void** faceSouth() {

**while**(!facingSouth()) {

turnLeft();

}

}

**public** **void** findNextDireciton() {

// Preliminary stuff

World.*setDelay*(0);

// first, check if there is only one beeper

**if**(nextToABeeper()) {

pickBeeper();

**if**(!nextToABeeper()) {

faceNorth();

System.***out***.println("Facing north");

} **else** {

pickBeeper();

**if**(!nextToABeeper()) {

faceSouth();

System.***out***.println("Facing south");

}

}

} **else** {

System.***out***.println("Not next to any beepers");

}

4. Rewrite and check the following new instruction, taking care to interpret all of the robot programming grammar rules correctly. This instruction uses nested IF’s to face a robot toward the east; verify that it is correct by simulation. Hint: When trying to check this instruction, ignore the instruction indentation. One way of doing this is to have someone read you the instruction. While they are reading the instruction, you should keep track of the meaningful components of the instruction. This is exactly what the factory does when it reads instructions to robots at the factory.

public void faceEast()

{

if( ! facingEast() )

{

if(facingWest() ) {

turnLeft();

turnLeft(); }

else

{ if(facingNorth() )

{

turnRight();}

else {

turnLeft();

}

}

}

}

**public** **void** faceEast() {

**while**(!facingEast()) {

turnLeft();

}

}

5. The current version of mysteryInstruction is syntactically correct, but very difficult to read. Simplify it by using the IF transformations.

public void mysteryInstruction ()

{

if(facingWest() )

{

move();

turnRight();

if(facingNorth() )

{

move();

}

turnAround();

}

else

{

move();

turnLeft();

move();

turnAround();

}

}

**public** **void** mysteryInstruction() {

**if**(facingWest()) {

move();

turnRight();

**if**(facingNorth()) {

move();

}

turnAround();

} **else** {

move();

turnLeft();

move();

}

}