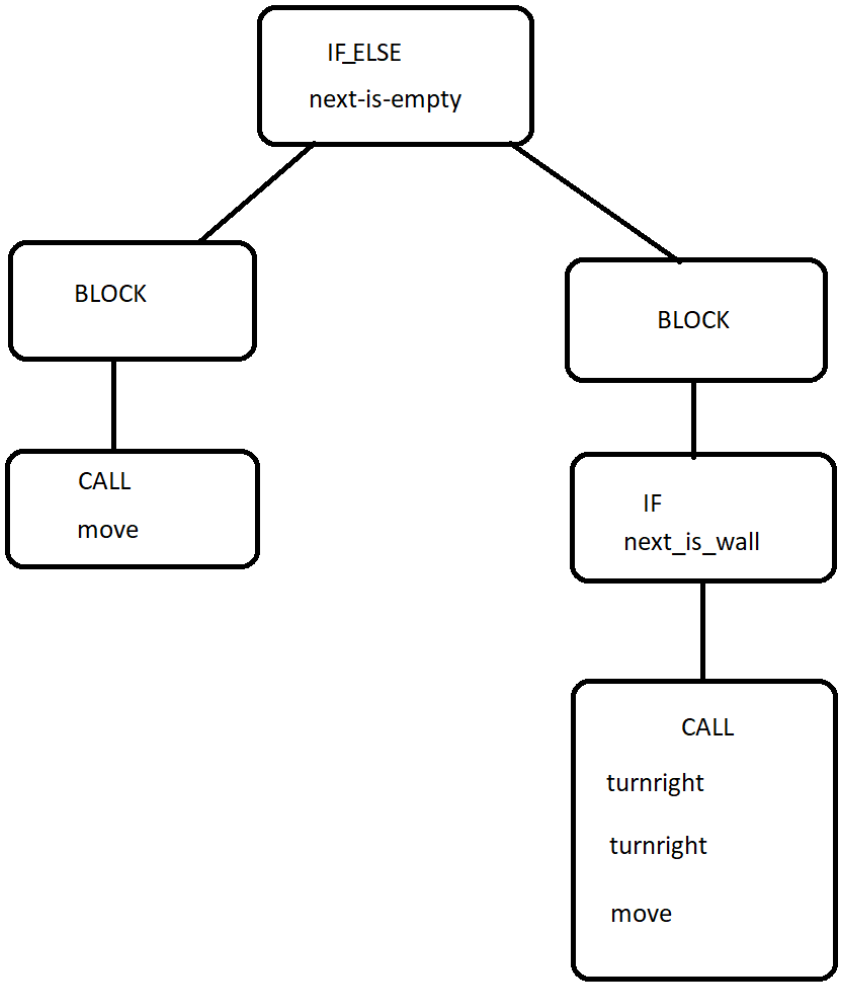
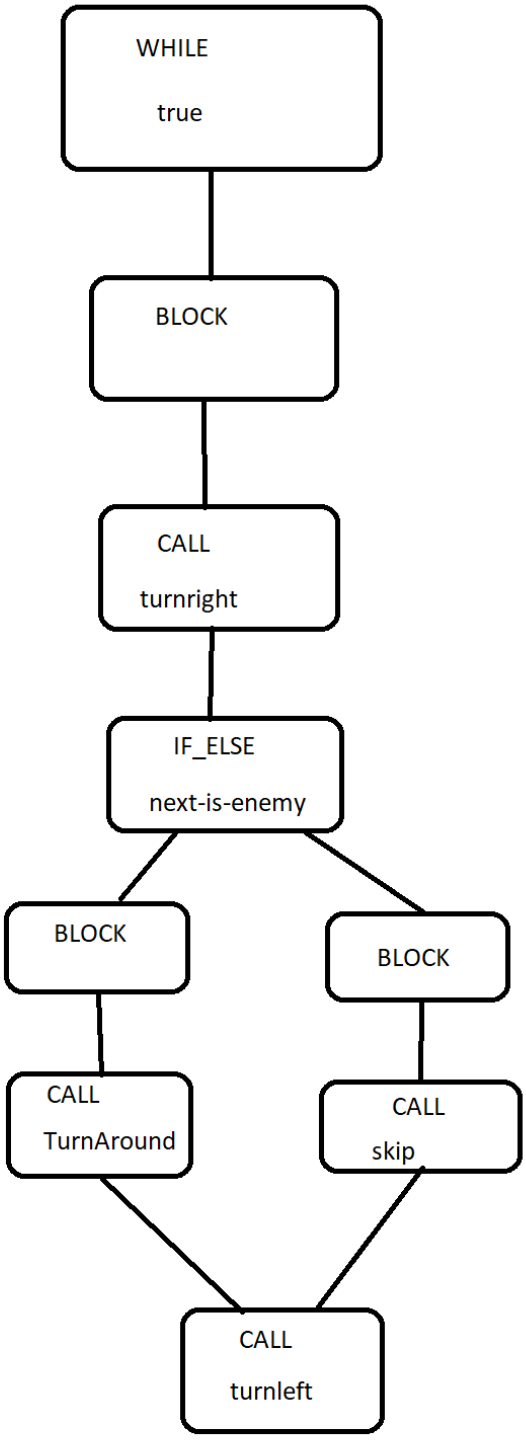
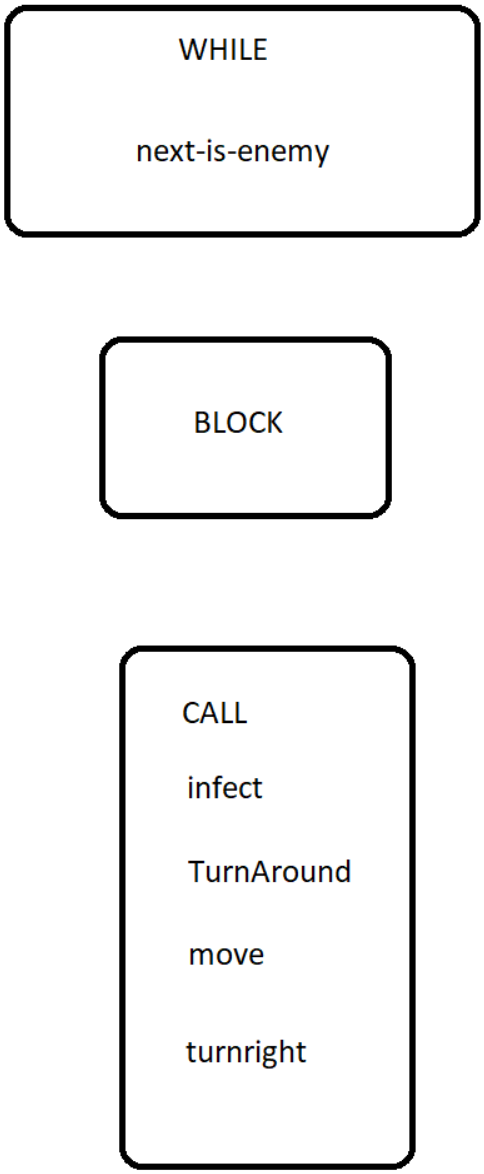
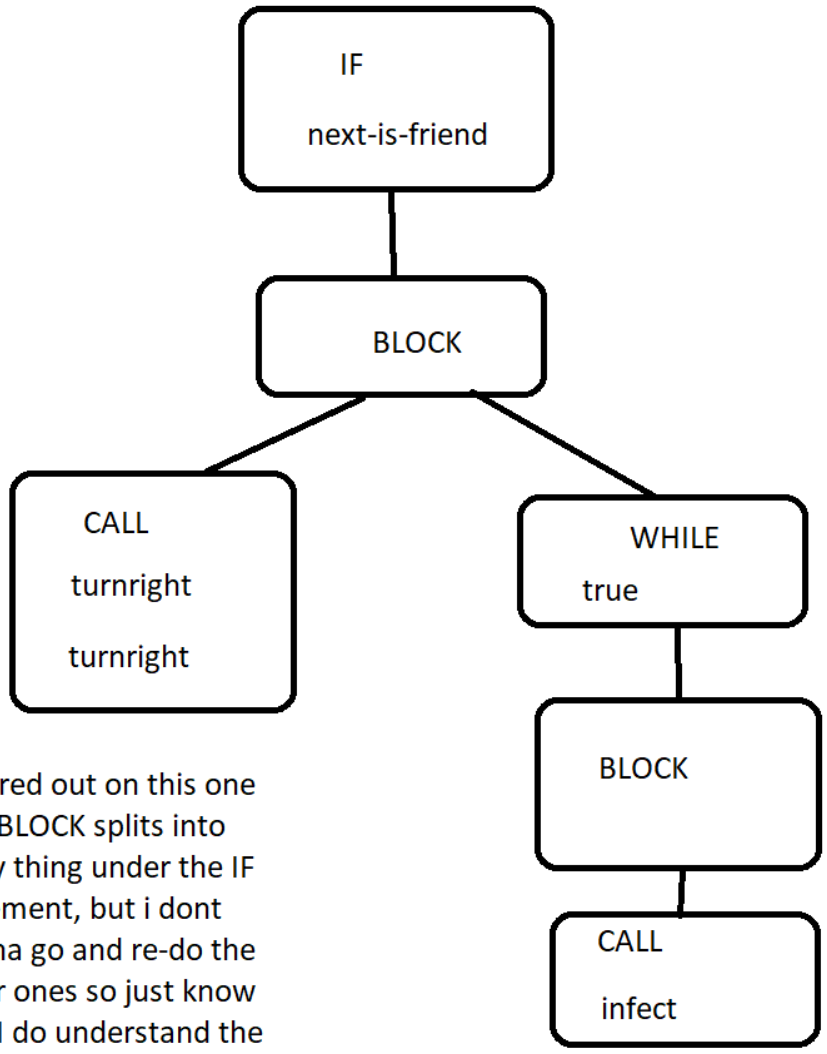
Homework 22

1.

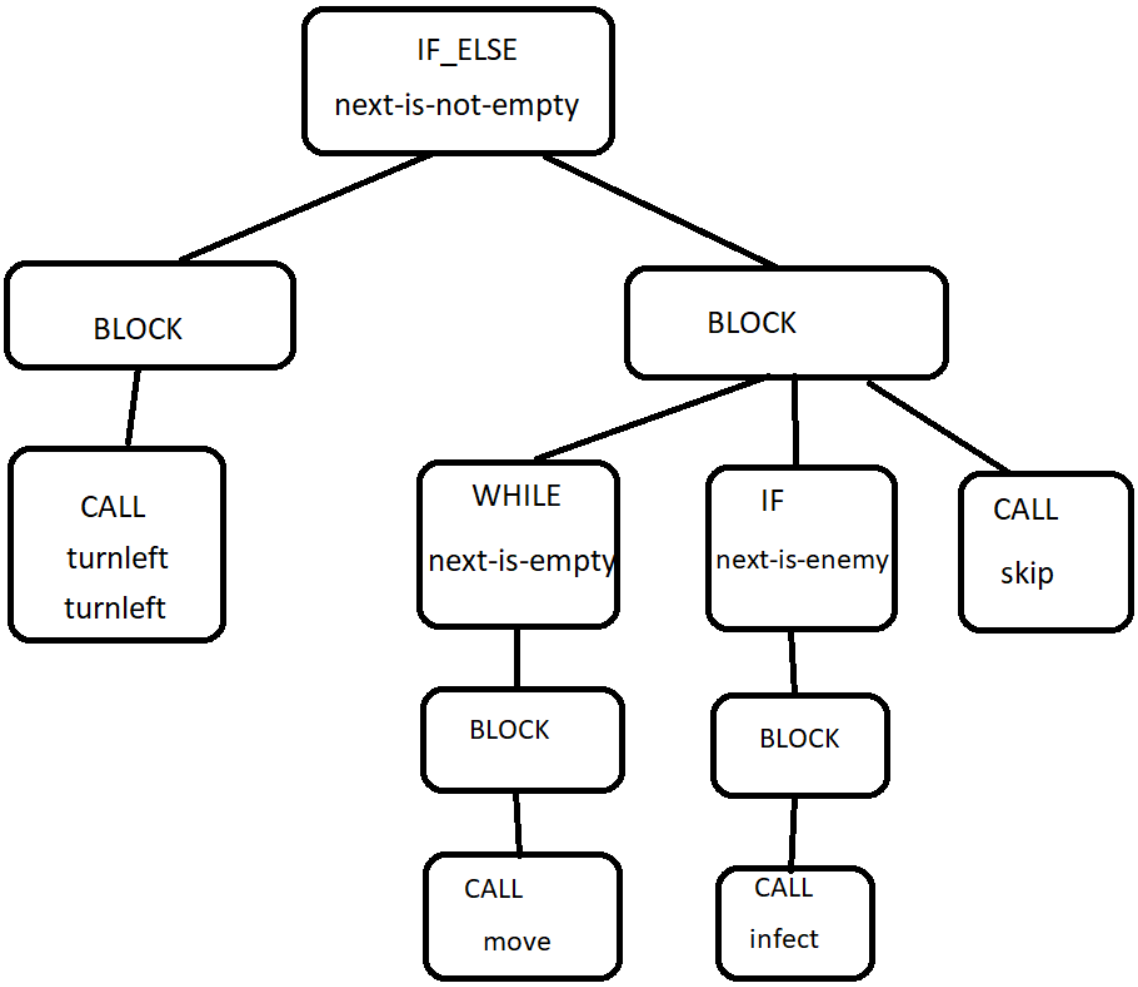

IF_ELSE
next-is-empty
BLOCK
BLOCK
CALL
move
IF
next_is_wall
CALL
turnright
turnright
move


WHILE
true
BLOCK
CALL
turnright
IF_ELSE
next-is-enemy
BLOCK
BLOCK
CALL
TurnAround
CALL
skip
CALL
turnleft


WHILE
next-is-enemy
BLOCK
CALL
infect
TurnAround
move
turnright



i figured out on this one that BLOCK splits into every thing under the IF statement, but i dont wanna go and re-do the other ones so just know that I do understand the concept.


IF_ELSE
next-is-not-empty
BLOCK
BLOCK
CALL
turnleft
turnleft
WHILE
next-is-empty
IF
next-is-enemy
CALL
skip
BLOCK
BLOCK
CALL
move
CALL
infect

2.

```
/**
* Reports the number of calls to primitive instructions (move, turnleft,
* turnright, infect, skip) in a given {@code Statement}.
*
* @param s
* the {@code Statement}
* @return the number of calls to primitive instructions in {@code s}
* @ensures <pre>
* countOfPrimitiveCalls =
* [number of calls to primitive instructions in s]
* </pre>
*/
public static int countOfPrimitiveCalls(Statement s) {
int count = 0;
switch (s.kind()) {
case BLOCK: {
/*
* Add up the number of calls to primitive instructions
* in each nested statement in the BLOCK.
*/
for (int i=0; i < s.lengthOfBlock(); i++) {
count += countOfPrimitiveCalls(s.removeFromBlock(i));
}
break;
```

```
}
case IF: {
/*
* Find the number of calls to primitive instructions in
* the body of the IF.
*/
for (int i=0; i < s.lengthOfBlock(); i++) {
count += countOfPrimitiveCalls(s.removeFromBlock(i));
}
break;
}
case IF_ELSE: {
/*
* Add up the number of calls to primitive instructions in
* the "then" and "else" bodies of the IF_ELSE.
*/
for (int i=0; i < s.lengthOfBlock(); i++) {
count += countOfPrimitiveCalls(s.removeFromBlock(i));
}
break;
}
case WHILE: {
/*
* Find the number of calls to primitive instructions in
* the body of the WHILE.
```

```
*/
for (int i=0; i < s.lengthOfBlock(); i++) {
count += countOfPrimitiveCalls(s.removeFromBlock(i));
}
break;
}
case CALL: {
/*
* This is a leaf: the count can only be 1 or 0. Determine
* whether this is a call to a primitive instruction or not.
*/
if (s.kind().equals(IDENTIFIER)) {
count++;
}
break;
}
default: {
// this will never happen...can you explain why?
// because all possible results are already addressed
break;
}
}
return count;
}
```