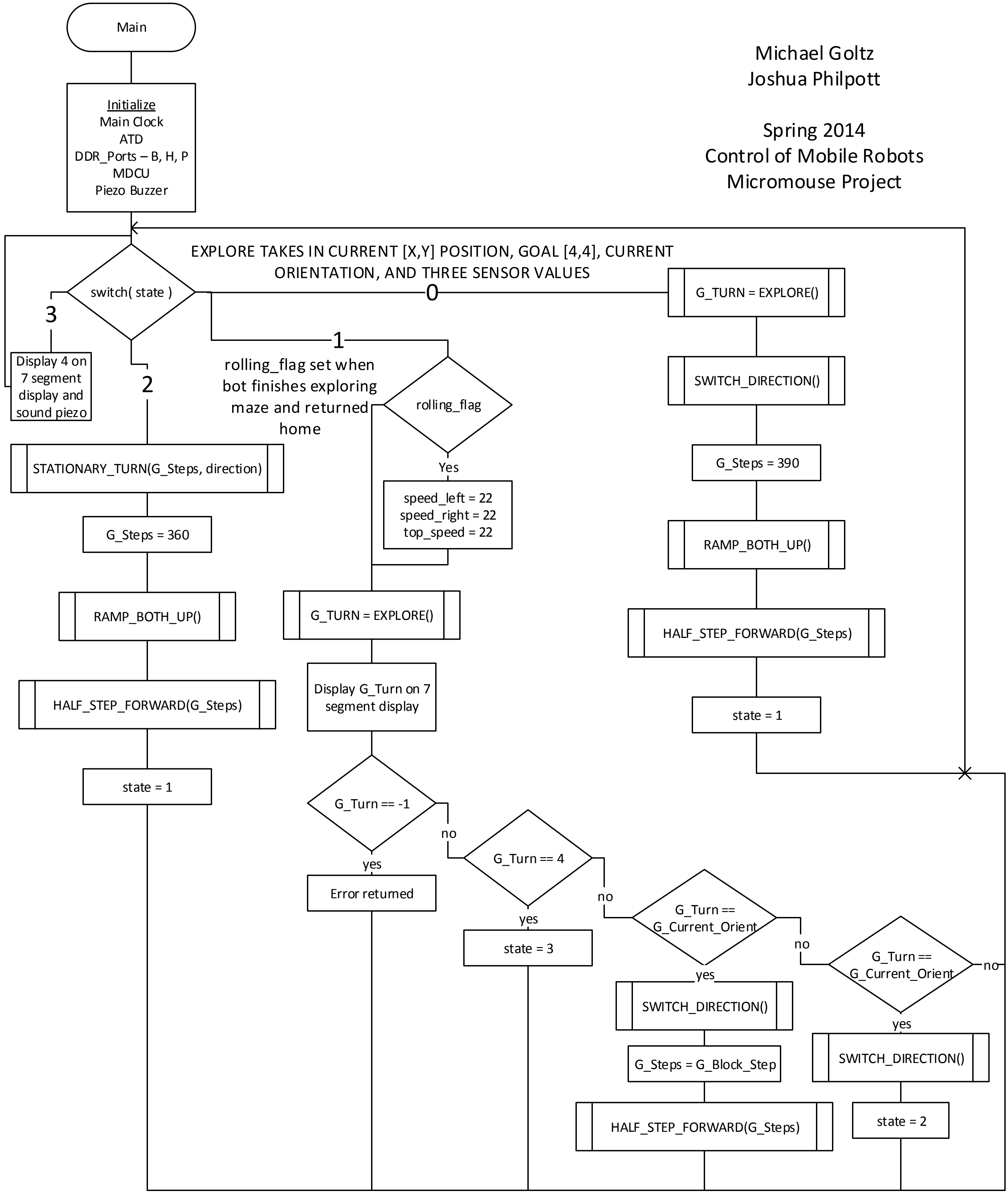
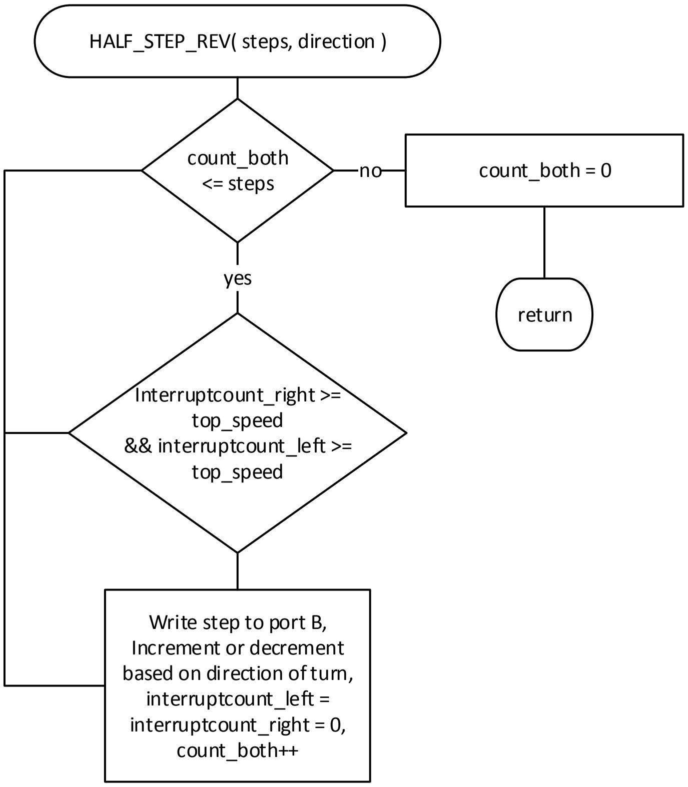
Michael Goltz
Joshua Philpott

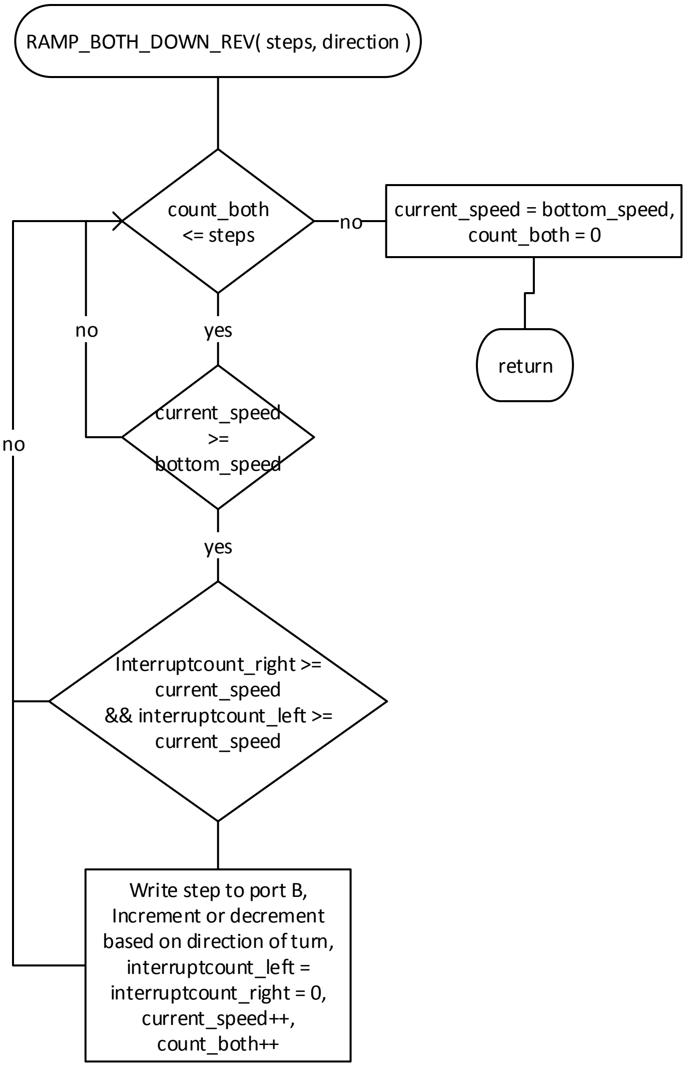
Spring 2014
Control of Mobile Robots
Micromouse Project

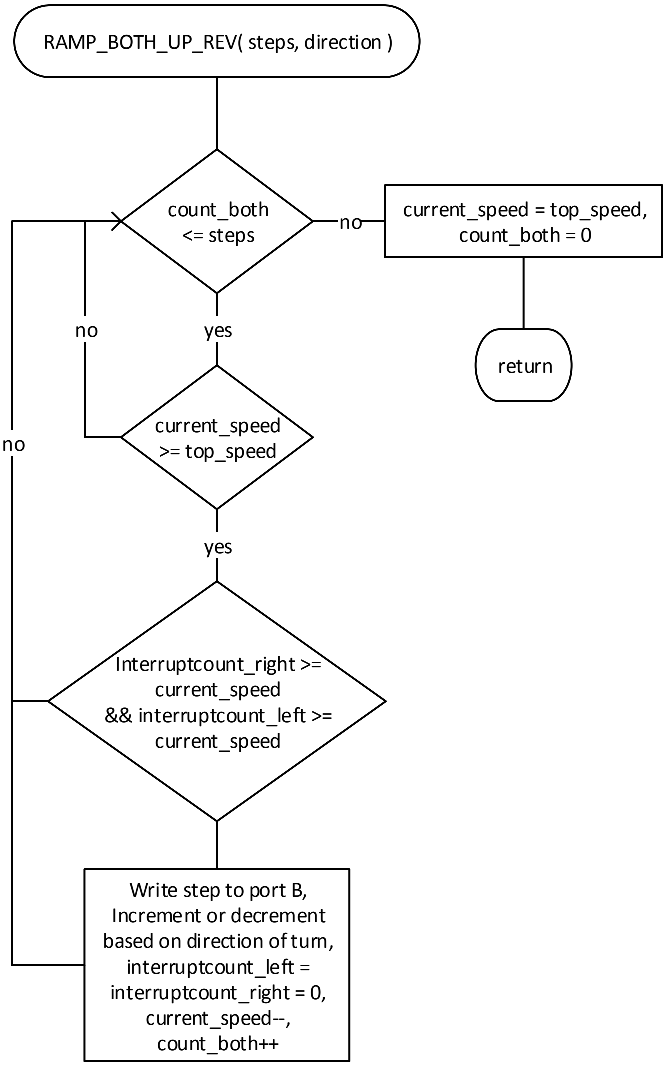

Main
Initialize
Main Clock
ATD
DDR_Ports – B, H, P
MDCU
Piezo Buzzer
switch(state)
EXPLORE TAKES IN CURRENT [X,Y] POSITION, GOAL [4,4], CURRENT ORIENTATION, AND THREE SENSOR VALUES
0
G_TURN = EXPLORE()
SWITCH_DIRECTION()
G_Steps = 390
RAMP_BOTH_UP()
HALF_STEP_FORWARD(G_Steps)
state = 1
1
rolling_flag set when bot finishes exploring maze and returned home
rolling_flag
Yes
speed_left = 22
speed_right = 22
top_speed = 22
G_TURN = EXPLORE()
Display G_Turn on 7 segment display
G_Turn == -1
yes
Error returned
no
G_Turn == 4
yes
state = 3
no
G_Turn == G_Current_Orient
yes
SWITCH_DIRECTION()
G_Steps = G_Block_Step
HALF_STEP_FORWARD(G_Steps)
no
G_Turn == G_Current_Orient
yes
SWITCH_DIRECTION()
state = 2
no
2
STATIONARY_TURN(G_Steps, direction)
G_Steps = 360
RAMP_BOTH_UP()
HALF_STEP_FORWARD(G_Steps)
state = 1
3
Display 4 on 7 segment display and sound piezo

Michael Goltz
Joshua Philpott

Spring 2014
Control of Mobile Robots
Micromouse Project

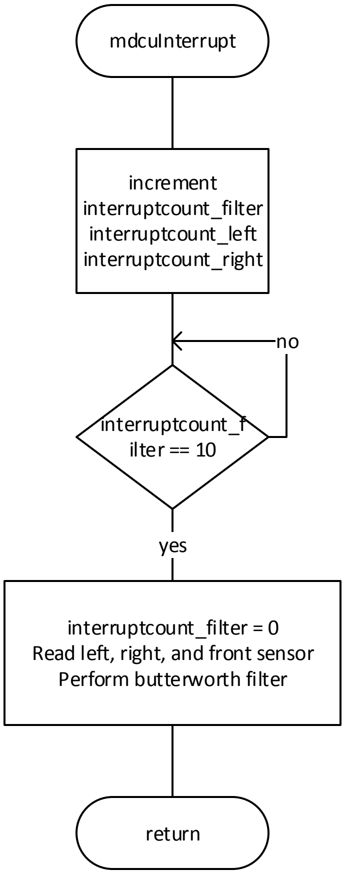

HALF_STEP_REV(steps, direction)
count_both <= steps
no
count_both = 0
return
yes
Interruptcount_right >= top_speed && interruptcount_left >= top_speed
Write step to port B, Increment or decrement based on direction of turn, interruptcount_left = interruptcount_right = 0, count_both++

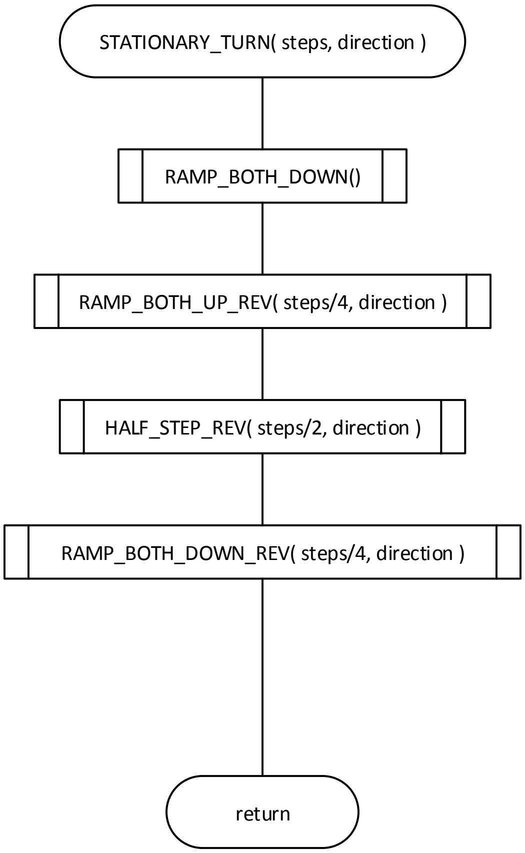

RAMP_BOTH_DOWN_REV(steps, direction)
count_both <= steps
no
current_speed = bottom_speed, count_both = 0
return
yes
no
current_speed >= bottom_speed
no
yes
Interruptcount_right >= current_speed && interruptcount_left >= current_speed
Write step to port B, Increment or decrement based on direction of turn, interruptcount_left = interruptcount_right = 0, current_speed++, count_both++

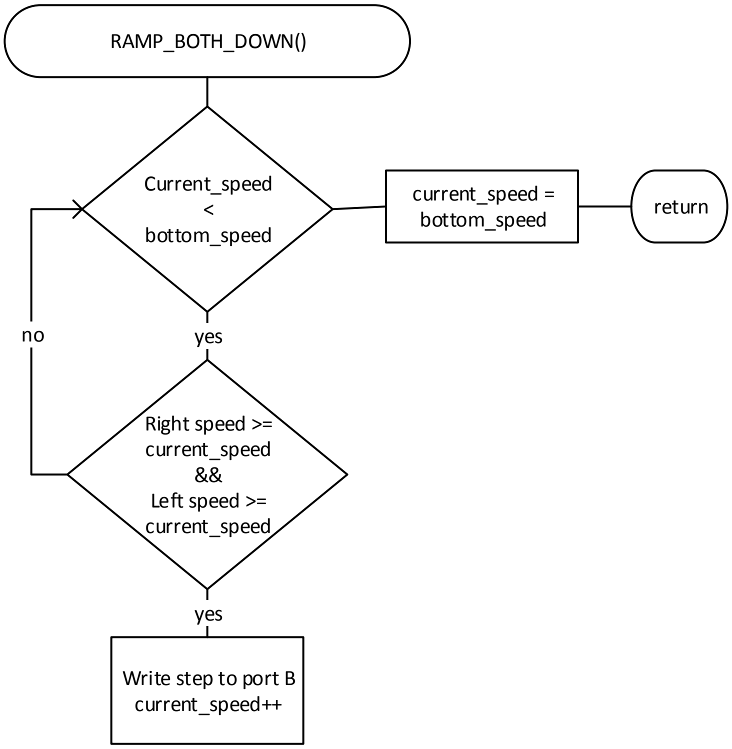

RAMP_BOTH_UP_REV(steps, direction)
count_both <= steps
no
current_speed = top_speed, count_both = 0
return
yes
no
current_speed >= top_speed
no
yes
Interruptcount_right >= current_speed && interruptcount_left >= current_speed
Write step to port B, Increment or decrement based on direction of turn, interruptcount_left = interruptcount_right = 0, current_speed--, count_both++

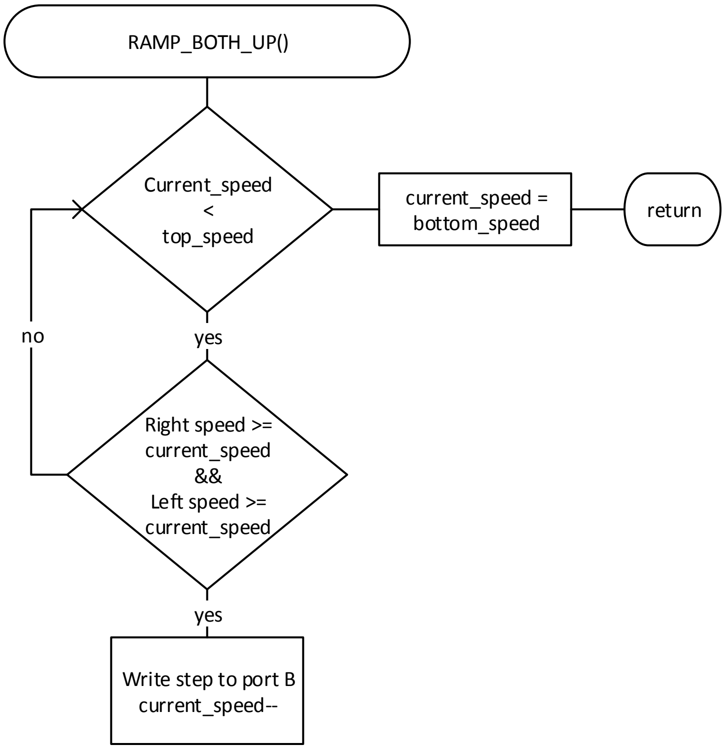
Michael Goltz
Joshua Philpott

Spring 2014
Control of Mobile Robots
Micromouse Project


mdcuInterrupt
increment
interruptcount_filter
interruptcount_left
interruptcount_right
no
interruptcount_filter == 10
yes
interruptcount_filter = 0
Read left, right, and front sensor
Perform butterworth filter
return


STATIONARY_TURN(steps, direction)
RAMP_BOTH_DOWN()
RAMP_BOTH_UP_REV(steps/4, direction)
HALF_STEP_REV(steps/2, direction)
RAMP_BOTH_DOWN_REV(steps/4, direction)
return

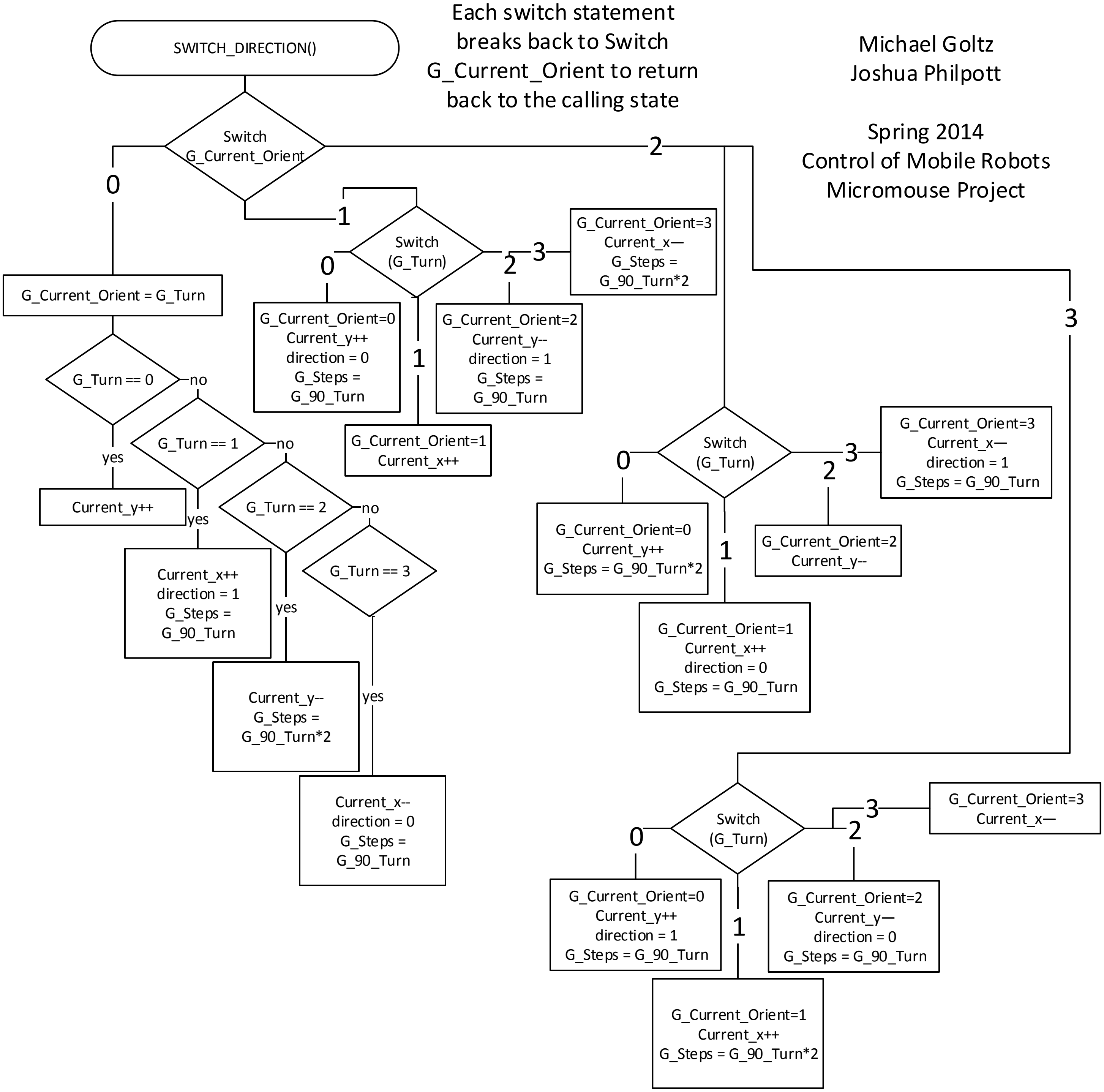




Each switch statement breaks back to Switch G_Current_Orient to return back to the calling state

Michael Goltz
Joshua Philpott

Spring 2014
Control of Mobile Robots
Micromouse Project


SWITCH_DIRECTION()
Switch G_Current_Orient
0
G_Current_Orient = G_Turn
G_Turn == 0
yes
Current_y++
no
G_Turn == 1
yes
Current_x++ direction = 1 G_Steps = G_90_Turn
no
G_Turn == 2
yes
Current_y-- G_Steps = G_90_Turn*2
no
G_Turn == 3
yes
Current_x-- direction = 0 G_Steps = G_90_Turn
1
Switch (G_Turn)
0
G_Current_Orient=0 Current_y++ direction = 0 G_Steps = G_90_Turn
1
G_Current_Orient=1 Current_x++
2
G_Current_Orient=2 Current_y-- direction = 1 G_Steps = G_90_Turn
3
G_Current_Orient=3 Current_x— G_Steps = G_90_Turn*2
2
Switch (G_Turn)
0
G_Current_Orient=0 Current_y++ G_Steps = G_90_Turn*2
1
G_Current_Orient=1 Current_x++ direction = 0 G_Steps = G_90_Turn
2
G_Current_Orient=2 Current_y--
3
G_Current_Orient=3 Current_x— direction = 1 G_Steps = G_90_Turn
3
Switch (G_Turn)
0
G_Current_Orient=0 Current_y++ direction = 1 G_Steps = G_90_Turn
1
G_Current_Orient=1 Current_x++ G_Steps = G_90_Turn*2
2
G_Current_Orient=2 Current_y— direction = 0 G_Steps = G_90_Turn
3
G_Current_Orient=3 Current_x—

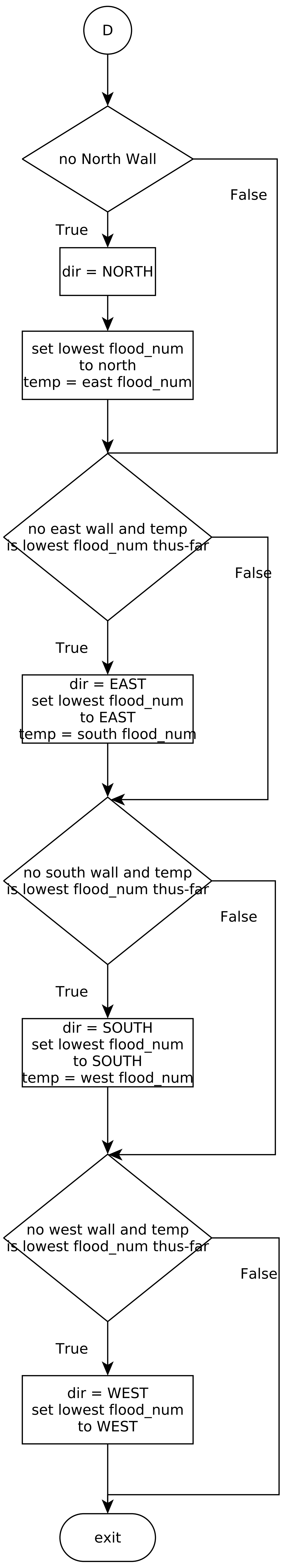
explore:

The explore algorithm started as simply the explore logic. Through programming the assignment, it eventually became the function which determines the next move for the entirety of the maze.

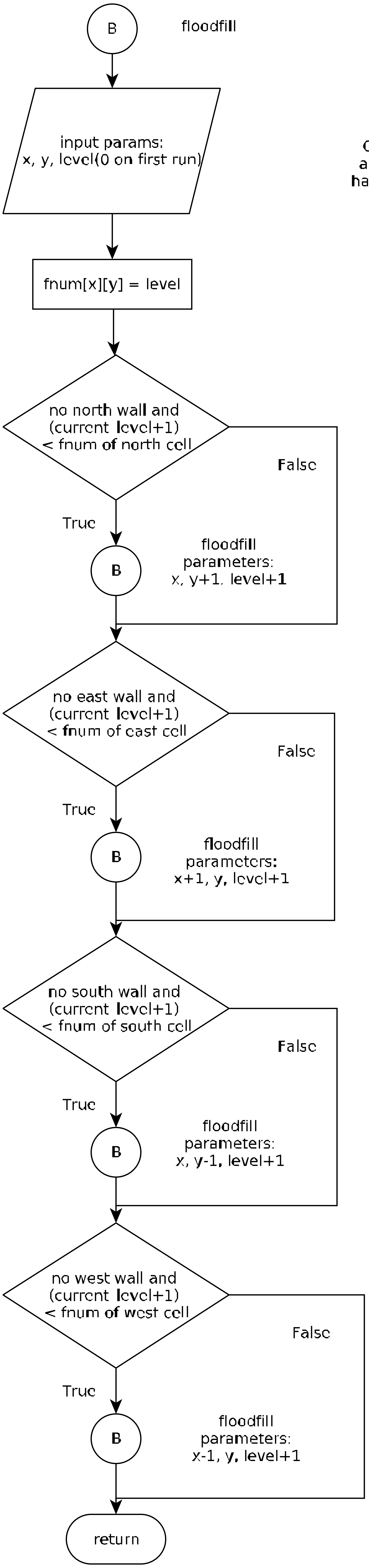
Based off some flags (which hopefully are fairly self descriptive), explore implements different methods to provide the bot with the next move at each cell (including drive back to home and race).

Basic algorithm:
If the maze is not fully explored, run a goal seeking floodfill algo to provide the next direction
If the maze has been explored and it's not at (0,0), drive to (0,0)
If the bot is back at (0,0), follow the path created by floodfill (shortest path in our case)
If the bot is at (4,4), return 4.

explore
Init
int dir = -1;
static int found_goal=0;
static int maze_explore_complete=0;
static int maze_return_start=0;
maze has not been fully explored
False
True
E
map the cell walls based off sensor readings
visited[y][x] = 1
x and y are at the goal?
False
True
found_goal = 1
goal has been found
False
True
A
increment_v subroutine finds next unvisted location closest to robots curent position
no unvisited locations left to visit
False
True
maze_explore_complete = 1
goal_x = next_visit_x
goal_y = next_visit_y
maze_explore_complete
at (0,0) ?
False
True
maze_return_start = 1
goal_x = 0
goal_y = 0
maze_return_start == 1
False
True
at (4,4)?
False
True
return COMPLETE
set floodfill numbers to 0
B
floodfill
D
look in all available directions and move towards the lowest floodfill value (the floodfill value shows the fastest path to 4,4 with the currently known walls)
dir >= 0
False
True
return DIRECTION
return ERROR



D
no North Wall
False
True
dir = NORTH
set lowest flood_num to north
temp = east flood_num
no east wall and temp is lowest flood_num thus-far
False
True
dir = EAST
set lowest flood_num to EAST
temp = south flood_num
no south wall and temp is lowest flood_num thus-far
False
True
dir = SOUTH
set lowest flood_num to SOUTH
temp = west flood_num
no west wall and temp is lowest flood_num thus-far
False
True
dir = WEST
set lowest flood_num to WEST
exit

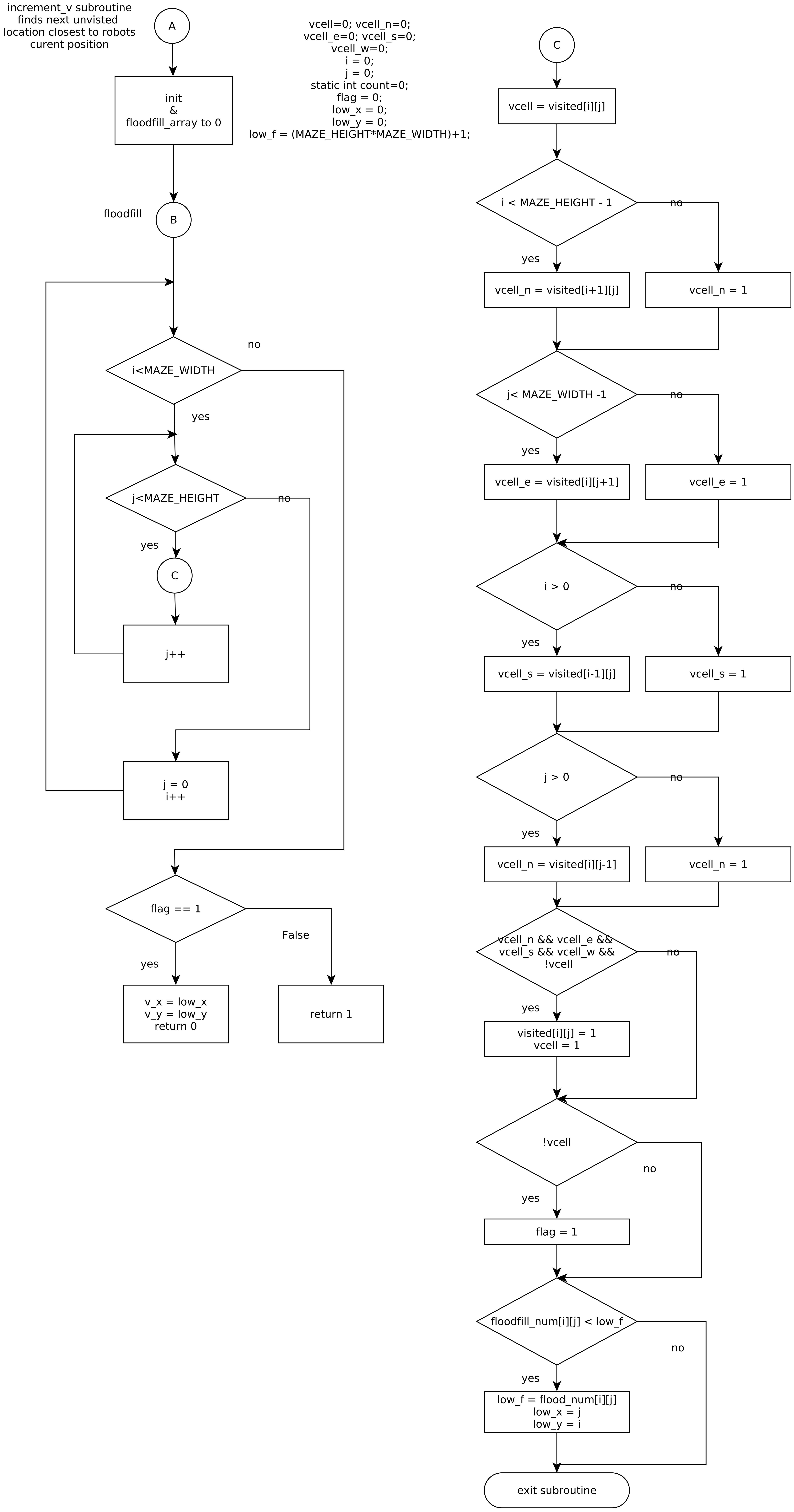


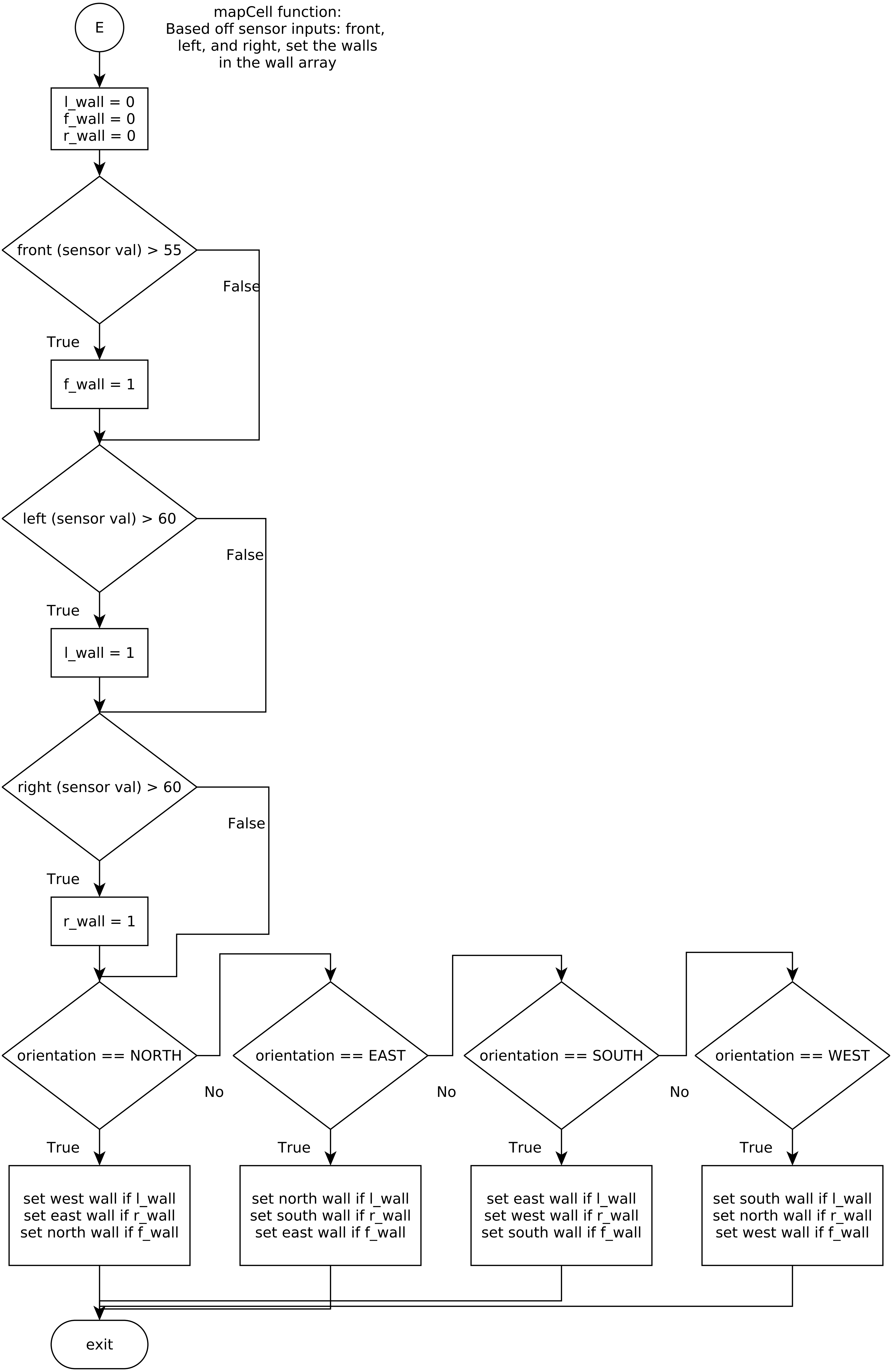
floodfill is a recursive algorithm

Takes parameters x,y, and level

Sets fnum array to level (indicates distance away from start)

Checks north, east, south, and west cells and runs floodfill (with level + 1) if the cell has a floodfill value > (level+1) and no walls between it


increment_v subroutine
finds next unvisted
location closest to robots
curent position
A
init
&
floodfill_array to 0
floodfill
B
i<MAZE_WIDTH
no
yes
j<MAZE_HEIGHT
no
yes
C
j++
j = 0
i++
flag == 1
yes
False
v_x = low_x
v_y = low_y
return 0
return 1
vcell=0; vcell_n=0;
vcell_e=0; vcell_s=0;
vcell_w=0;
i = 0;
j = 0;
static int count=0;
flag = 0;
low_x = 0;
low_y = 0;
low_f = (MAZE_HEIGHT*MAZE_WIDTH)+1;
C
vcell = visited[i][j]
i < MAZE_HEIGHT - 1
no
yes
vcell_n = visited[i+1][j]
vcell_n = 1
j< MAZE_WIDTH -1
no
yes
vcell_e = visited[i][j+1]
vcell_e = 1
i > 0
no
yes
vcell_s = visited[i-1][j]
vcell_s = 1
j > 0
no
yes
vcell_n = visited[i][j-1]
vcell_n = 1
vcell_n && vcell_e &&
vcell_s && vcell_w &&
!vcell
no
yes
visited[i][j] = 1
vcell = 1
!vcell
no
yes
flag = 1
floodfill_num[i][j] < low_f
no
yes
low_f = flood_num[i][j]
low_x = j
low_y = i
exit subroutine



mapCell function:
Based off sensor inputs: front,
left, and right, set the walls
in the wall array