SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

19 Maze Tracing Robot Example

Dr. Spencer Smith

Faculty of Engineering, McMaster University

February 15, 2018

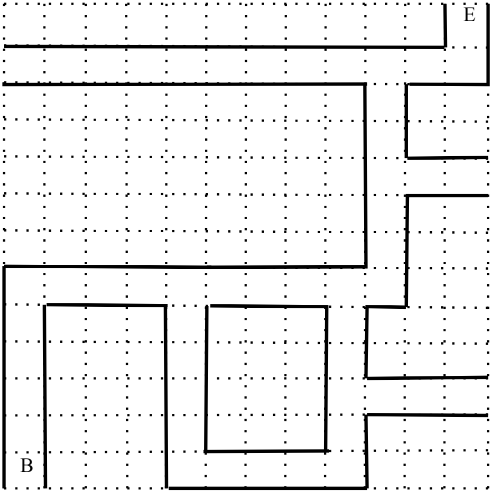
19 Maze Tracing Robot Example

- Administrative details
- Dr. v. Mohrenschildt's maze tracing robot
  - see GitLab
  - Content section on Avenue
- MIS for maze_storage

Administrative Details

- Assignment 2
  - Part 1: February 20, 2018
  - Partner Files: February 26, 2018
  - Part 2: March 2, 2018
- Midterm exam
  - Wednesday, February 28, 7:00 pm
  - 90 minute duration

Dr. v. Mohrenschildt's Maze Tracing Robot Example



Likely Changes?

- What is the first step in the design process?

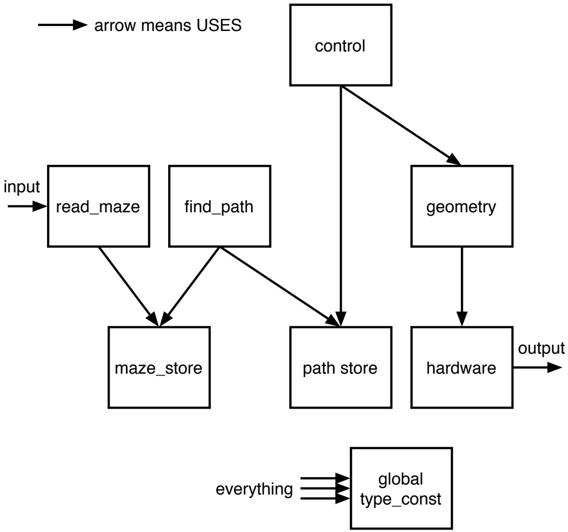
Likely Changes?

- What is the first step in the design process?
- What are some potential likely changes?

Maze Tracing Robot Expected Changes

- Changes to the geometry of the robot such that the mapping from a position to the robot inputs is different
- Changes to the hardware interface to the robot
- Changes to the input format of the maze
- Changes to any constant values
- Changes to the data structure to store the maze
- Changes to the path finding algorithm

Maze Tracing Robot Uses Hierarchy



Maze Tracing Robot MG

- Module name: maze_storage
  - Secret: how the maze is stored
  - Service: stores the maze
  - Module prefix: ms_
- Module name: load_maze
  - Secret: where and how the maze file is read in
  - Service: loads the maze
  - Module prefix: lm_
- Module name: find_path
  - Secret: the algorithm for finding the shortest path
  - Service: finds the shortest path through the maze
  - Module prefix: fp_

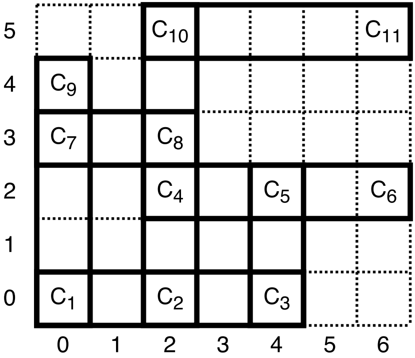
Maze Tracing Robot MG Continued

- Module name: control
  - Secret: how the arm moves from position to position and how the buttons are checked
  - Service: controls the movement of the arm
  - Module prefix: cn_
- Module name: geometry
  - Secret: how the calculations from cell coords to robot coords are performed
  - Service: handles geometric positioning of the arm
  - Module prefix: gm_
- Module name: hardware
  - Secret: how it interfaces with the robot
  - Service: handles hardware aspects of arm (movement and button checking)
  - Module prefix: hw_

Maze Tracing Robot MG Continued

- Module name: types_constants
  - Secret: how the data structures are defined and constants defined
  - Service: provides standard variable types and constants to modules

Understanding Maze Storage



Path length is measured by the number of grid blocks. What length are these paths?

C_1, C_2, C_{10}, C_{11}
C_1, C_2, C_4, C_8, C_{10}, C_{11}

maze_storage MIS

Module

maze_storage

Uses

types_constants #*provides NUM_X_CELLS, NUM_Y_CELLS*

Exported Access Programs

Routine name	In	Out	Exceptions
...	...	...	...

What are some potential access programs?

maze_storage Exported Access Programs

Routine name	In	Out	Exceptions
ms_init			
ms_set_maze_start	cell		ms_not_initialized, ms_cell_out_of_range
ms_set_maze_end	cell		ms_not_initialized, ms_cell_out_of_range
ms_get_maze_start		cell	ms_not_initialized
ms_get_maze_end		cell	ms_not_initialized
ms_set_wall	cell, cell		ms_not_initialized, ms_not_valid_wall, etc.
ms_is_connected	cell, cell	boolean	ms_not_initialized, ms_cell_out_of_range

cell = tuple of (x: integer, y: integer)

maze_storage Semantics

State Variables – Ideas?

State Invariant: none

Assumptions

ms_get_maze_start and ms_get_maze_end are not called until after the corresponding set routines have been called.

maze_storage Semantics

State Variables

maze: set of tuple of (cell, cell)
start : cell
end : cell
is_init : boolean := *false*

State Invariant: none

Assumptions

ms_get_maze_start and ms_get_maze_end are not called until after the corresponding set routines have been called.

Access Routine Semantics

ms_init():

- transition:
- exception:

ms_set_maze_start(c):

- transition:
- exception:

ms_set_maze_end(c):

- transition:
- exception:

Access Routine Semantics

ms_init():

- transition: $maze, is\_init := \{\}, true$
- exception: none

ms_set_maze_start(c):

- transition:
- exception:

ms_set_maze_end(c):

- transition:
- exception:

Access Routine Semantics

ms_init():

- transition: $maze, is\_init := \{\}, true$
- exception: none

ms_set_maze_start(c):

- transition: $start := c$
- exception:

ms_set_maze_end(c):

- transition: $end := c$
- exception:

Access Routine Semantics

ms_init():

- transition: $maze, is\_init := \{\}, true$
- exception: none

ms_set_maze_start(c):

- transition: $start := c$
- exception: $exc := (\neg is\_init \Rightarrow \text{ms\_not\_initialized} \mid \neg\text{cell\_in\_range}(c) \Rightarrow \text{ms\_cell\_out\_of\_range})$

ms_set_maze_end(c):

- transition: $end := c$
- exception: $exc := (\neg is\_init \Rightarrow \text{ms\_not\_initialized} \mid \neg\text{cell\_in\_range}(c) \Rightarrow \text{ms\_cell\_out\_of\_range})$

Access Routine Semantics Continued

ms_get_maze_start():

- output:
- exception:

ms_get_maze_end():

- output:
- exception:

Access Routine Semantics Continued

ms_get_maze_start():

- output: $out := start$
- exception:

ms_get_maze_end():

- output: $out := end$
- exception:

Access Routine Semantics Continued

ms_get_maze_start():

- output: $out := start$
- exception: $exc := (\neg is\_init \Rightarrow \text{ms\_not\_initialized})$

ms_get_maze_end():

- output: $out := end$
- exception: $exc := (\neg is\_init \Rightarrow \text{ms\_not\_initialized})$

Access Routine Semantics Continued

ms_set_wall(c1, c2):

- transition: ?
- exception: ?

Access Routine Semantics Continued

ms_set_wall(c1, c2):

- transition: $maze := maze \cup \{< c1, c2 >\}$
- exception: ?

Access Routine Semantics Continued

ms_set_wall(c1, c2):

- transition: $maze := maze \cup \{< c1, c2 >\}$
- exception: $exc := (\neg is\_init \Rightarrow$ $\text{ms\_not\_initialized} \mid \text{wall\_is\_point}(c1, c2) \vee$ $\text{wall\_is\_diagonal}(c1, c2) \vee \text{wall\_is\_out\_of\_range}(c1, c2) \Rightarrow$ $\text{ms\_not\_valid\_wall})$

Access Routine Semantics Continued

ms_is_connected(c1, c2):

- output:
- exception:

 # Assume that all intermediate segments are in the set of maze walls. Could rephrase to allow to cover a portion of a segment. The more general case is covered on 2017 Midterm

Access Routine Semantics Continued

ms_is_connected(c1, c2):

- output:
  $out := \exists p : \text{sequence of cell} | p[0] = c1 \wedge p[|p| - 1] = c2 \wedge \forall (i : \mathbb{N} | 0 \leq i \leq |p| - 2 : < p[i], p[i + 1] > \in maze)$
- exception:

 # Assume that all intermediate segments are in the set of maze walls. Could rephrase to allow to cover a portion of a segment. The more general case is covered on 2017 Midterm

Access Routine Semantics Continued

ms_is_connected(c1, c2):

- output:
  $out := \exists p : \text{sequence of cell} | p[0] = c1 \wedge p[|p| - 1] = c2 \wedge \forall (i : \mathbb{N} | 0 \leq i \leq |p| - 2 :< p[i], p[i+1] > \in maze)$
- exception: $exc := (\neg is\_init \Rightarrow \text{ms\_not\_initialized} \mid \neg \text{cell\_in\_range}(c1) \Rightarrow \text{ms\_cell\_out\_of\_range} \mid \neg \text{cell\_in\_range}(c2) \Rightarrow \text{ms\_cell\_out\_of\_range})$

 # Assume that all intermediate segments are in the set of maze walls. Could rephrase to allow to cover a portion of a segment. The more general case is covered on 2017 Midterm

Local Functions

cell_in_range : cell $\rightarrow$ boolean

wall_is_point: cell $\times$ cell $\rightarrow$ boolean

wall_is_diagonal: cell $\times$ cell $\rightarrow$ boolean

Local Functions

cell_in_range : cell $\rightarrow$ boolean

- cell_in_range (c) $\equiv (0 \leq c.x < \text{NUM\_X\_CELLS}) \wedge (0 \leq c.y < \text{NUM\_Y\_CELLS})$

wall_is_point: cell $\times$ cell $\rightarrow$ boolean

wall_is_diagonal: cell $\times$ cell $\rightarrow$ boolean

Local Functions

cell_in_range : cell $\rightarrow$ boolean

- cell_in_range (c) $\equiv (0 \leq c.x <$ NUM_X_CELLS$) \wedge (0 \leq c.y <$ NUM_Y_CELLS$)$

wall_is_point: cell $\times$ cell $\rightarrow$ boolean

- wall_is_point (c1, c2) $\equiv c1 = c2$

wall_is_diagonal: cell $\times$ cell $\rightarrow$ boolean

Local Functions

cell_in_range : cell $\rightarrow$ boolean

- cell_in_range (c) $\equiv (0 \leq c.x < \text{NUM\_X\_CELLS}) \wedge (0 \leq c.y < \text{NUM\_Y\_CELLS})$

wall_is_point: cell $\times$ cell $\rightarrow$ boolean

- wall_is_point (c1, c2) $\equiv c1 = c2$

wall_is_diagonal: cell $\times$ cell $\rightarrow$ boolean

- wall_is_diagonal (c1, c2) $\equiv \neg((c1.x = c2.x) \vee (c1.y = c2.y))$

Additional Specifications for Determining the Path

pathT = sequence of cell
validPath : pathT $\rightarrow$ boolean

Additional Specifications for Determining the Path

pathT = sequence of cell

validPath : pathT → boolean

- validPath (p)
  $\equiv (p[0] = \text{ms\_get\_maze\_start}() \wedge p[|p|-1] = \text{ms\_get\_maze\_end}() \wedge \forall (i : \mathbb{N} | 0 \leq i \leq |p|-2 : \text{ms\_is\_connected}(p[i], p[i+1]))$

Additional Specifications for Determining the Path

pathT = sequence of cell
validPath : pathT → boolean

- validPath (p) $\equiv (p[0] = \text{ms\_get\_maze\_start}() \wedge p[|p|-1] = \text{ms\_get\_maze\_end}() \wedge \forall (i : \mathbb{N} | 0 \leq i \leq |p| - 2 : \text{ms\_is\_connected}(p[i], p[i+1]))$

How would you specify the length of a wall?

Additional Specifications for Determining the Path

pathT = sequence of cell
validPath : pathT → boolean

- validPath (p)
  $\equiv (p[0] = \text{ms\_get\_maze\_start}() \wedge p[|p|-1] = \text{ms\_get\_maze\_end}() \wedge \forall (i : \mathbb{N} | 0 \leq i \leq |p|-2 : \text{ms\_is\_connected}(p[i], p[i+1]))$

How would you specify the length of a wall?
lenWall: tuple of cell → integer

$$\text{lenWall}(< c1, c2 >) \equiv (c1.x = c2.x \Rightarrow |c1.y - c2.y|$$

$$|c1.y = c2.y \Rightarrow |c1.x - c2.x|)$$

Shortest Path

How would you specify the length of a path?

How would you specify whether a path is the shortest path?

Shortest Path

How would you specify the length of a path?
lenPath: pathT $\rightarrow$ integer

$$\text{lenPath}(p) \equiv +(i : \mathbb{N} | 0 \leq i < (|p|-1) : \text{lenWall}(< p_i, p_{i+1} >))+1$$

How would you specify whether a path is the shortest path?

Shortest Path

How would you specify the length of a path?
lenPath: pathT $\rightarrow$ integer

$$\text{lenPath}(p) \equiv +(i : \mathbb{N} | 0 \leq i < (|p| - 1) : \text{lenWall}(< p_i, p_{i+1} >)) + 1$$

How would you specify whether a path is the shortest path?
isShortest: pathT $\rightarrow$ boolean

$$\text{isShortest}(p) \equiv \forall (q : \text{pathT}$$

$$| \text{validPath}(q) : \text{validPath}(p) \wedge \text{lenPath}(p) \leq \text{lenPath}(q))$$

Midterm Questions

The Midterm 2017 has several questions related to mazes