SE 3XA3: Module Interface Specification
Mastermind

Team 204, Trident Inc.
Justin Prez, prezj
Justin Rosner, rosnej1
Harshil Modi, modih1

April 5, 2020

This document shows the complete specification for the modules used for running and playing Mastermind.

Table 1: Revision History

Date	**Developer(s)**	**Change**
March 10, 2020	Justin Prez, Justin Rosner, HarshiL Modi	Initial write-up of MIS document
April 5, 2020	Justin Rosner, Justin Prez, Harshil Modi	Revision 1 of MIS

~~Game~~ Types Module

Module

GameTypes

Uses

None

Syntax

Exported Constants

MAX_ROWS = 10
MAX_COLUMNS = 4

Exported Types

ColourT = {red, blue, yellow, green, white, purple}
ClueT = {correct, semicorrect, incorrect}

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Assumptions

For ClueT, correct indicates a guess of correct position and colour, semicorrect indicates a guess of correct colour but not correct position, and incorrect is a guess that does not match position or colour of the final code.

Button ~~ADT~~ Module

Template Module

ButtonT

Uses

GameTypes

Syntax

Exported Constants

None

Exported Types

ButtonT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
ButtonT	ColourT, $\mathbb{R}$, $\mathbb{R}$	ButtonT	InvalidPointException
get_colour		ColourT	
get_column		$\mathbb{R}$	
get_row		$\mathbb{R}$	
draw_button			

Semantics

State Variables

col: $\mathbb{R}$
row: $\mathbb{R}$
colour: ColourT

Environment Variables

None

State Invariant

$0 \leq \text{col} \leq \text{MAX\_COLUMNS} - 1$
$0 \leq \text{row} \leq \text{MAX\_ROWS} - 1$

Assumptions

The constructor ButtonT is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

ButtonT(x, y, c):

- output: $out := self$
- transition: $col, row, colour := x, y, c$
- exception: $(\neg(0 \leq x \leq \text{MAX\_COLUMNS} - 1) \vee \neg(0 \leq y \leq \text{MAX\_ROWS} - 1)) \Rightarrow$ InvalidPointException

get_colour():

- output: $out := colour$
- exception: None

get_column():

- output: $out := col$
- exception: None

get_row():

- output: $out := row$
- exception: None

draw_button():

- output: None
- transition: Draw the Button to the screen with the colour specified by *colour* at the coordinate (col, row).
- exception: None

Game Board ~~ADT~~ Module

Template Module

BoardT

Uses

ButtonT
GameTypes

Syntax

Exported Constants

None

Exported Types

BoardT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
BoardT		BoardT	
draw_board		BoardT	invalid_argument
add_button	ButtonT		
add_clue	ClueT		
get_buttons		seq of ButtonT	
get_clues		seq of ClueT	

Semantics

State Variables

buttons = seq of ButtonT
clues = seq of ClueT

Environment Variables

~~None~~ Screen = ?

State Invariant

$0 \leq |\text{buttons}| \leq \text{MAX\_ROWS} * \text{MAX\_COLUMNS}$
$0 \leq |\text{clues}| \leq \text{MAX\_ROWS} * \text{MAX\_COLUMNS}$

Assumptions

The constructor BoardT is called for each abstract object before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

BoardT():

- output: *self*
- transition: $buttons, clues := \emptyset, \emptyset$
- exception: None

draw_board():

- output: None
- transition: $\forall\ i \in [0..\text{MAX\_ROW} * \text{MAX\_COLUMNS} - 1] \cdot buttons[i].\text{draw\_button}() \wedge \text{draw\_clue}(clues[i], buttons[i].\text{get\_row}())$
- exception: $(\neg(0 \leq b \leq \text{MAX\_COLUMNS}*\text{MAX\_ROWS}) \vee \neg(0 \leq c \leq \text{MAX\_COLUMNS}*\text{MAX\_ROWS}) \Rightarrow \text{invalid\_argument}$

add_button(b):

- output: None
- transition: $buttons := buttons\ ||\ < b >$
- exception: None

add_clue(c):

- output: None
- transition: $clues := clues\ ||\ < c >$

- exception: None

get_buttons():

- output: $out := buttons$
- exception: None

get_clues():

- output: $out := clues$
- exception: None

Local Functions

draw_clue: ClueT × $\mathbb{N} \rightarrow \#$ *draws clue to screen*
draw_clue$(c, r) \equiv \#$ *draw clue c to the board for row r*

Menu Module

Module

Menu

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
draw_menu			
draw_instructions			

Semantics

State Variables

None

Environment Variables

~~None~~ Screen = ?

State Invariant

None

Assumptions

None

Access Routine Semantics

draw_menu():

- output: None
- transition: Draw the menu to the screen with the proper text formatting and layout.
- exception: None

draw_instructions():

- output: None
- transition: Draw the instructions to the screen with the proper text formatting and layout.
- exception: None

Local Functions

None

Game Board Controller

Module

GameBoardController

Uses

BoardT

Syntax

Exported Constants

None

Exported Types

GameStateT = {win, lose, playing}

Exported Access Programs

Routine name	In	Out	Exceptions
new_game			
next_move	ButtonT	GameStateT	

Semantics

State Variables

win_combo = seq of ButtonT
board = BoardT
num_buttons = $\mathbb{N}$

Environment Variables

None

State Invariant

|winning_combo| = 4

Assumptions

Access Routine Semantics

new_game():

- output: None
- transition: board, win_combo, num_buttons := BoardT(), generate_combo(), 0
- exception: None

next_move(b):

- output: is_end_state()
- transition:
  - num_buttons, board := num_buttons + 1, board.add_button(b)
  - (num_buttons % 4 = 0) ⇒ validate_guess()
  - board.draw_board()
- exception: None

Local Functions

is_end_state() ≡ (∀ i ∈ [|board.getClues()| − 4..board.getClues()| − 1] · (num_buttons%4 = 0) ∧ (board.getClues()[i] = correct) ⇒ winning) ∨ ((num_buttons = MAX_ROWS × MAX_COLUMNS) ∧ (∃ i ∈ [board.getClues()| − 4..board.getClues()| − 1] · board.getClues()[i] ≠ correct) ⇒ losing) ∨ ((num_buttons ≤ MAX_ROWS × MAX_COLUMNS) ∧ (∃ i ∈ [|board.getClues()| − 4..board.getClues()| − 1] · (board.getClues()[i] ≠ correct) ⇒ playing)

validate_guess() ≡ ∀i ∈ [num_buttons − 4..num_buttons − 1] · ((board.get_buttons()[i].get_col = win_combo[i%4].get_col ∧ board.get_buttons()[i].get_colour = win_combo[i%4].get_colour) ⇒ board.add_clue(correct,board.get_buttons()[i].get_row)∨((board.get_buttons()[i].get_col ≠ win_combo[i%4].get_col ∧ board.get_buttons()[i].get_colour = win_combo[i% 4].get_colour) ⇒ board.add_clue(semicorrect,board.get_buttons()[i].get_row)∨ ((board.get_buttons()[i].get_col ≠ win_combo[i% 4].get_col ∧ board.get_buttons()[i].get_colour ≠ win_combo[i% 4].get_colour) ⇒

board.add_clue(incorrect,board.get_buttons()[i].get_row)

generate_combo → seq of ButtonT
generate_combo() # *generate 4 buttons with random colours from ColourT, and then return them in a seq of ButtonT*

Android Emulator ~~Module~~

Module

Main

Uses

GameBoardController

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	In	Out	Exceptions
start_up			
on_screen_press			

Semantics

State Variables

game_state = GameStateT

Environment Variables

button_input: {button_newGame, button_instructions, button_Red, button_Blue, button_Yellow, button_Green, button_White, button_Purple }

State Invariant

None

Assumptions

The assumption for this module is that the start_up is called upon the Android Emulator starting up. Using Flutter libraries, on_screen_press() will be called every time the user clicks a button on the screen.

Access Routine Semantics

start_up():

- output: None
- transition: draw_menu()
- exception: None

on_screen_press($but, game\_state$):

- output: None
- transition:

	$transistion$
$but = button\_newGame$	$new\_game()$
$but = button\_instructions$	$draw\_instructions()$
$but = button\_Red$	$game\_state := next\_move(but)$
$but = button\_Blue$	$game\_state := next\_move(but)$
$but = button\_Yellow$	$game\_state := next\_move(but)$
$but = button\_GREEN$	$game\_state := next\_move(but)$
$but = button\_WHITE$	$game\_state := next\_move(but)$
$but = button\_PURPLE$	$game\_state := next\_move(but)$
$but = invalid\_spot$	$game\_state := game\_state$

- exception: None

Local Functions

None