MIS

Team 3
Shuying Chen, 400080161
Ziyang Huang, 400051063
Guiye Wu, 400089784

Nov 8 2018

Contents

List of Tables

List of Figures

1 Major Revision History

Table 1: **Revision History**

Date	**Version**	**Notes**
08/11/2018	0.3	Revision 0 complete
08/11/2018	0.2	Refined section
07/11/2018	0.1	Draft of Sections

2 Module Hierrchy

Level 1	Level 2
Hardware Hiding Module	
Behaviour Hiding Module	Minesweeper Module Board Module Animation Module Timer Module
Software Decision Module	Cell Module

Table 2: Module Hierarchy

3 Cell ADT Module

3.1 Module

Cell

3.2 Uses

Pygame

3.3 Syntax

3.3.1 Exported Types

Cell = ?

3.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
Cell	$\mathbb{B}$, $\mathbb{Z}$, $\mathbb{Z}$	PointT	invalid_argument
unlock	$\mathbb{T}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$		invalid_argument
flag	$\mathbb{T}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$		invalid_argument
unflag	$\mathbb{T}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$		invalid_argument
clickExplode	$\mathbb{T}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$,$\mathbb{Z}$		invalid_argument
isBomb		$\mathbb{B}$	
isUnlocked		$\mathbb{B}$	
isFlagged		$\mathbb{B}$	
getRow		$\mathbb{Z}$	
getCol		$\mathbb{Z}$	
setNumBombNeighbors	$\mathbb{Z}$		invalid_argument
setIsUnlocked	$\mathbb{Z}$		invalid_argument
getNumBombNeighbors		$\mathbb{Z}$	

3.4 Semantics

3.4.1 State Variables

unlocked: $\mathbb{B}$ (A boolean variable which checks if a cell is locked or unlocked)
$flagged$: $\mathbb{B}$ (A boolean variable which checks if a is set flag or not set flag)
isB: $\mathbb{B}$ (A boolean variable which checks if a cell has bomb or not)
r: $\mathbb{Z}$ (cell position in row)
c: $\mathbb{Z}$ (cell position in column)
nBN: $\mathbb{Z}$ (A value represents the number of bombs around the cell)

3.4.2 Environment Variables

screen: An output frame from Pygame

3.4.3 State Invariant

$0 \leq nBN \leq 8$

3.4.4 Assumptions

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

3.4.5 Access Routine Semantics

Cell($isBomb, row, col$):

- transition: $unlocked, flagged, isB, r, c, nBN := false, false, isBomb, row, col, 0$
- output: $out := self$
- exception: $(r < 0 \lor c < 0) \Rightarrow$ invalid_argument

unlock(screen,r,c,w,h):

- transition: $unlocked := true$
  $(\forall num \in 0..8 \bullet (nBN = num) \Rightarrow Pygame.screen.blit(scale(num, (w, h)), (c * w, r * h))$
- output: None
- exception:$(r < 0 \lor c < 0 \lor w < 0 \lor h < 0) \Rightarrow$ invalid_argument

flag(screen,r,c,w,h):

- transition: $flagged := true$
  $flg := image.load("flag")$
  $Pygame.screen.blit(scale(flg, (w, h)), (c * w, r * h))$
- output: None
- exception: $(r < 0 \lor c < 0 \lor w < 0 \lor h < 0) \Rightarrow$ invalid_argument

unflag(screen,r,c,w,h):

- transition: $flagged := false$
  $blanckcell := image.load("blanckcell")$
  $Pygame.screen.blit(scale(blanckcell, (w, h)), (c * w, r * h))$
- output: None

- exception: $(r < 0 \vee c < 0 \vee w < 0 \vee h < 0) \Rightarrow$ invalid_argument

clickExplode(screen,r,c,w,h):

- transition: $hitmine := image.load(\text{"}hitmine\text{"})$
  $Pygame.screen.blit(scale(hitmine, (w, h)), (c * w, r * h))$
- output: None
- exception: $(r < 0 \vee c < 0 \vee w < 0 \vee h < 0) \Rightarrow$ invalid_argument

isBomb():

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

isUnlocked():

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

isFlagged():

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

getRow():

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

getCol():

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

setNumBombNeighbors(numBombNeighbors):

- transition: $nBN := numBombNeighbors$
- output: None
- exception: $(numBombNeighbors > 8 \vee numBombNeighbors < 0) \Rightarrow$ invalid_argument

setIsUnlocked(isUnlocked):

- transition: $unlocked := isUnlocked$

- output: None
- exception: None

getNumBombNeighbors():

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

4 Board ADT Module

4.1 Template Module

Board

4.2 Uses

Cell,Pygame, timer, Animation

4.3 Syntax

4.3.1 Exported Types

Board = ?

4.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
Board	$\mathbb{Z}$, $\mathbb{Z}$, $\mathbb{Z}$, $\mathbb{Z}$, $\mathbb{Z}$	Board	invalid_argument
refreshCells			
getNumBombNeighbors	$\mathbb{Z}$, $\mathbb{Z}$	$\mathbb{Z}$	invalid_argument
rightClick	$\mathbb{Z}$, $\mathbb{Z}$	screen	invalid_argument
leftClick	$\mathbb{Z}$, $\mathbb{Z}$	screen	invalid_argument
unlockNeighbors	Cell		
isCell	$\mathbb{Z}$, $\mathbb{Z}$	$\mathbb{B}$	
rightAndLeftClick	$\mathbb{Z}$, $\mathbb{Z}$	screen	invalid_argument
getNumFlagNeighbors	$\mathbb{Z}$, $\mathbb{Z}$	$\mathbb{Z}$	invalid_argument
depressNeighbors	$\mathbb{Z}$, $\mathbb{Z}$		invalid_argument
winGame		screen	
looseGame		screen	

4.4 Semantics

4.4.1 State Variables

rs: $\mathbb{Z}$ (number of rows)
cs: $\mathbb{Z}$ (number of column)
w: $\mathbb{Z}$ (width of a cell)
h: $\mathbb{Z}$ (height of a cell)
nBS: $\mathbb{Z}$ (number of bombs)
$numCells$: $\mathbb{Z}$ (number of cells in total)
$numCellsToUncover$: $\mathbb{Z}$ (number of cells that is not covered)

cells: a list of $Cell$(a list of cells represent a board)
T: *timer* (timer function)

4.4.2 Environment Variables

screen: An output frame from Pygame
blanckcell: A variable which stores a blanckcell image

4.4.3 State Invariant

None

4.4.4 Assumptions

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

4.4.5 Access Routine Semantics

Board(x, y):

- transition: $rs, cs, w, h, nBs, numCells, numCellsToUncover, cells, T := rows, cols, width, height, numBombs, rows * cols, 0, [\ ], timer()$
- output: $out := self$
- exception: $(rows < 0 \lor cols < 0 \lor width < 0 \lor height < 0) \Rightarrow$ invalid_argument

refreshCells():

- transition: $numCellsToUncover := numCells - nBs$
  $List := []$
  $\forall i \in 0..numCells \bullet List \mid\mid < i >$
  $shuffle(List)$
  $\forall i \in 0..rs \bullet \forall j \in 0..cs \bullet cells \mid\mid < Cell(List[i + j] < nBs, i, j)) >$
  $\forall x \in 0..rs \bullet \forall y \in 0..cs \bullet \neg cells[x][y].isBomb \Rightarrow$
  $cells[x][y].setNumBombNeighbors(getNumBombNeighbors(x, y))$
- output: None
- exception: None

getNumBombNeighbors(i,j):

- transition: $counter := 0$
  $\forall k \in -1..1 \bullet \forall m \in -1..1 \bullet (k \neq 0 \land m \neq 0 \land cells[i + k][j + m].isBomb) \Rightarrow counter := counter + 1$

- output: $out := counter$
- exception: $(\neg(0 \leq i \leq rs) \vee \neg(0 \leq j \leq cs)) \Rightarrow$ invalid_argument

rightClick(r, c):

- transition: $(\neg cells[r][c].isUnlocked) \Rightarrow (cells[r][c].isFlagged \Rightarrow cells[r][s].unflag(screen, cells[r][c].getRow, cells[r][c].getCol, w, h))$
  $(\neg cells[r][c].isUnlocked) \Rightarrow (\neg cells[r][c].isFlagged \Rightarrow cells[r][s].flag(screen, cells[r][c].getRow, cells[r][c].getCol, w, h))$
  (right click on a cell to set a flag or unset a flag)
- output: $screen$
- exception: $(\neg(0 \leq r \leq rs) \vee \neg(0 \leq c \leq cs)) \Rightarrow$ invalid_argument

leftClick(r, c):

- transition: $((\neg cells[r][c].isFlagged \wedge \neg cells[r][c].isUnLocked) \Rightarrow cells[r][c].isBomb) \Rightarrow cells[r][c].clickExplode(screen, cells[r][c].getRow, cells[r][c].getCol, w, h) \wedge looseGame())$
  $((\neg cells[r][c].isFlagged \wedge \neg cells[r][c].isUnLocked) \Rightarrow \neg cells[r][c].isBomb) \Rightarrow cells[r][c].getNumBombNeighbors = 0 \Rightarrow unlockNeighbors \wedge numCellsToUncover := numCellsToUncover - 1)$
  $((\neg cells[r][c].isFlagged \wedge \neg cells[r][c].isUnLocked) \Rightarrow \neg cells[r][c].isBomb) \Rightarrow numCellsToUncover = 0 \Rightarrow winGame)$
  (left click on a cell to unlock the cell if it is a bomb, loose game, if it is a value, visualize the value, if it is a blanck cell, unlock neighbor cells)
- output: $screen$
- exception: $(\neg(0 \leq r \leq rs) \vee \neg(0 \leq c \leq cs)) \Rightarrow$ invalid_argument

unlockNeighbors(currentCell):

- transition: unlock the neighbors cells until al l the blanck cells are block by the value cells, using the BFS method.
- output: None
- exception: None

isCell(col,row):

- transition: None
- output: $out := 0 \leq row < rs \wedge 0 \leq col < cs$
- exception: None

rightAndLeftClick(r,c):

- transition: Left and right click on a value cell to unlock the neighbor cells
- output: *screen*
- exception: $out := 0 \leq r < rs \wedge 0 \leq c < cs$

getNumFlagNeighbors(i,j):

- transition: Calculate number of neighbor flags from a cell with position (i,j)
- output: None
- exception: $out := 0 \leq row < i \wedge 0 \leq j < cs$

depressNeighbors(i,j):

- transition: loose game if right and left click on a value cell (i,j) that unlock a bomb cell
- output: None
- exception: $out := 0 \leq row < i \wedge 0 \leq j < cs$

winGame():

- transition: wining animation
- output: *screen*
- exception: None

looseGame():

- transition: loosing animation
- output: *screen*
- exception: None

5 timer Module

5.1 Module

timer

5.2 Uses

Pygame

5.3 Syntax

5.3.1 Exported Types

timer = ?

5.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
timer	screen	screen	

5.4 Semantics

5.4.1 Environment Variables

screen: An output frame from Pygame

5.4.2 State Invariant

None

5.4.3 Assumptions

None

5.4.4 Access Routine Semantics

timer(screen):

- transition: initialize a clock counting the time on the screen
- output: $out := screen$
- exception: None

6 Animation Module

6.1 Template Module

Animation

6.2 Uses

Pygame

6.3 Syntax

6.3.1 Exported Types

Animation = ?

6.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
Animation	scrn	Animation	
win		screen	
loose		screen	

6.4 Semantics

6.4.1 Environment Variables

screen : An output frame from Pygame

6.4.2 State Invariant

None

6.4.3 Assumptions

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

6.4.4 Access Routine Semantics

Animation(scrn):

- transition: $scrn = screen$
- output: $out := self$
- exception: None

win():

- transition: Animation of banana rain on the screen
- output: $out := screen$
- exception: None

loose():

- transition: Animation of funny poopoo rain on the screen
- output: None
- exception: None

7 Minesweeper Module

7.1 Module

Minesweeper

7.2 Uses

Board

7.3 Syntax

7.3.1 Exported Types

Minesweeper = ?

7.3.2 Exported Constants

None

7.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
Minesweeper		Minesweeper	
on_init		screen	
on_event	event		
on_cleanup			
on_execute		screen	

7.4 Semantics

7.4.1 State Variables

$widthx$: $\mathbb{Z}$ (the width of the screen)
$heightx$: $\mathbb{Z}$ (the height of the screen)
$runningBackground$: $\mathbb{B}$ (check if background interface is running)
$runningBoard$: $\mathbb{B}$ (check if the board is running)
$ROWS$: $\mathbb{Z}$ (number of row cells)
$COLS$: $\mathbb{Z}$ (number of column cells)
$WIDTH$: $\mathbb{Z}$ (width of a cell)
$HIGHT$: $\mathbb{Z}$ (height of a cell)
$NUMBOMBS$: $\mathbb{Z}$ (number of bombs in the board)

7.4.2 Environment Variables

screen: An output frame from Pygame
setlevel: A variable store the levels image
background: A variable store the background image

7.4.3 State Invariant

None

7.4.4 Assumptions

The Minesweeper constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.

7.4.5 Access Routine Semantics

Minesweeper():

- transition: $widthx, heightx, runningBackground, runningBoard, ROWS, COLS, NUMBOMBS, WIDTH, HIGHT := 860, 640, true, true, 16, 30, 0, 20, 20$
- output: $out := self$
- exception: None

on_init():

- transition: initialize the levels chosen background, if any level region is clicked, runningBackground assign to false, then initialize the game with ROWS * COLS cells and NUMBOMBS bombs depend on the chosen level. The cell size is WIDTH * HIGHT
- output: $out := screen$
- exception: None

on_event(event):

- transition: check the event if it is left click right click or left and right click on the cell to make the corresponding event on the screen
- output: None
- exception: None

on_cleanup():

- transition: clean up screen and exit the game

- output: None
- exception: None

on_execute():

- transition: execute on_event all the time until esc button is pressed or the exit icon is clicked on the screen
- output: $out := screen$
- exception: None