Minesweeper

CSE 4102 Project Homework 4, Spring 2018

Bryan Arnold

4/15/18

Section: 001

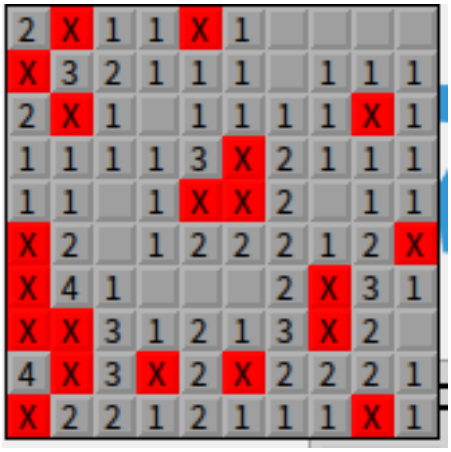
Instructor: Jeffrey A. Meunier

1. Introduction

In this assignment, I will create a version of the popular Minesweeper game. The game was developed in Pharo, which is an implementation of the Smalltalk programming language. This version of the Minesweeper game has all the same logic and rules, but the cells for mines and numbers are all visible. This is just to work out the logic for testing. I wasn't sure whether I was supposed to revert these so that it displayed like traditional Minesweeper, so I just followed the example screenshots in the assignment as a model for my program. I also couldn't get comments to stay for my classes for whatever reason.

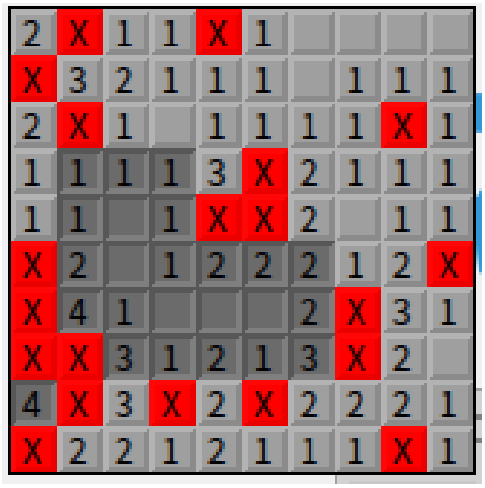
2. Output

The first part of the output will be the very beginning of the game. Each time my program is run, the beginning will be different, as the location of the mines is always random. To start my program, open Playground. Then type 'MineSweeperGame new openInWorld.', then do it. This is what should appear:

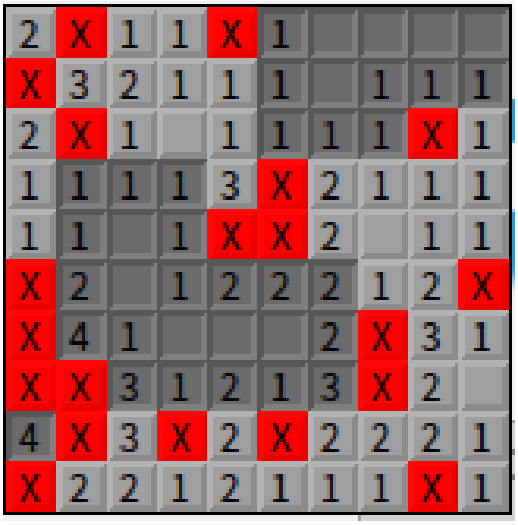


Next, I will press one of the blank cells in the middle of the board to display what happens when one is pressed:

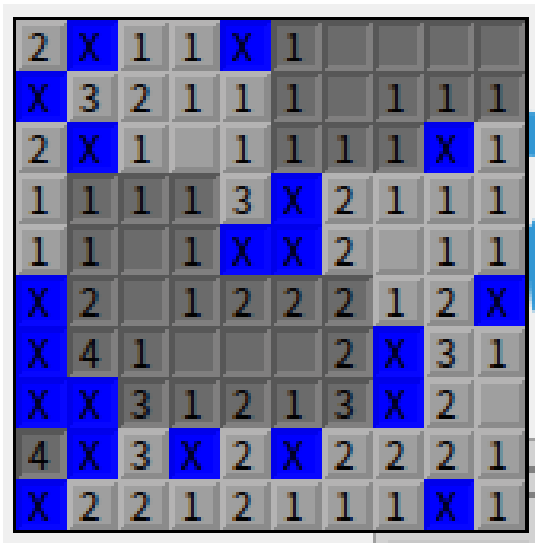
As you can see, any cell that is connected to this cell and outwards until mines are hit will be turned on, or not able to be clicked anymore. Next, I will show what happens when you press a cell with a number on it by clicking the number 4 on the bottom left:



The cell is now turned on, and since there are no blank cells and the cell is surrounded by 4 mines, it will be the only cell to turn on. Next, just to show it again, I will press one of the blank cells on the top right of the board:



This displays the same behavior as the previous time I showed it. Finally, I will show when a mine is clicked. As you can see, just as the screenshots in the assignment, they are indicated as red ‘X’. When one is pressed, all the mines should turn blue:



After clicking one mine, just that occurred. This is when the game would be over. I’m not sure if I was supposed to hide the mines and numbers like traditional minesweeper, so I followed the screenshots from the assignment as my model. Hopefully there won’t be points off for my confusion!

3. Source Code

```
SimpleSwitchMorph subclass: #MineSweeperCell
	instanceVariableNames: 'mouseAction cellAttr cellName cellVal'
	classVariableNames: ''
	poolDictionaries: ''
	category: '0-MineSweeperGame'!
```

```
!MineSweeperCell methodsFor: 'accesing' stamp: 'BryanArnold 4/15/2018 21:20'!
setBlank
	"setBlank Method.
	This method is a sort of constructor for a blank cell in the game.
	It defaults the cell's attribute to 0, which is blank, sets the color to
	white when it is in the off state, and has no name (as it is blank).
	Call cellUpdate afterwards to update this buttons new status as blank."

	cellAttr := 0.
	onColor := Color white.
	cellName := ' '.

	self cellUpdate.! !

!MineSweeperCell methodsFor: 'accesing' stamp: 'BryanArnold 4/15/2018 21:22'!
setNumbered
	"setNumbered Method.
	This method is a sort of constructor for a numbered cell in the game.
	It defaults the cell's attribute to 2, which is a cell with a number in it, sets the color to
	light gray when it is in the off state, and the name is dependent on the increment method.
	When a button is pressed, it will turn dark grey to indicate it has been pushed into the on state.
	Call cellUpdate afterwards to update this button new status."

	offColor := Color lightGray.
	onColor := Color gray darker.
	cellAttr := 2.

	self cellUpdate.! !

!MineSweeperCell methodsFor: 'accesing' stamp: 'BryanArnold 4/15/2018 20:57'!
mouseAction: aBlock
	"mouseAction Method
	Just like in the Light's Out assignment,
	set the mouse action to be the specified block for an action to occur based on what the mouse does."

  ^ mouseAction := aBlock! !

!MineSweeperCell methodsFor: 'accesing' stamp: 'BryanArnold 4/15/2018 21:20'!
increment
	"increment Method.
	This method sets up the cells that will have numbers on them. Each time it is called
	in the future, the name of the cell's number increases, depending on how many mines surround it.
```

```
	Set the name to the string form of a numbered cell, and call update to cellUpdate the
cell's	status."

	self setNumbered.
	cellVal := cellVal + 1.
	cellName := cellVal asString.

	self cellUpdate.! !

!MineSweeperCell methodsFor: 'accesing' stamp: 'BryanArnold 4/15/2018 21:21'!
setMine
	"setMineMethod.
	This method is a sort of constructor for a mine cell in the game.
	It defaults the cell's attribute to 1, which is a mine, sets the color to
	red when it is in the off state (DANGER DANGER), and has a name 'X' (to indicate a
mine).
	When a mine is pressed, the color of it in the on state will be blue with an X as the name
still.
	Call cellUpdateafterwards to update this buttons new status."

	cellAttr := 1.
	offColor := Color red.
	onColor := Color blue.
	cellName := 'X'.

	self cellUpdate.! !


!MineSweeperCell methodsFor: 'testing' stamp: 'BryanArnold 4/15/2018 21:24'!
blankChecker
	"blankChecker Method
	Simple method to check if cell type is blank.
	If the cell is blank, return true,
	else return false."

	^ cellAttr == 0! !

!MineSweeperCell methodsFor: 'testing' stamp: 'BryanArnold 4/15/2018 21:25'!
numberedChecker
	"numberedChecker Method
	Simple method to check if cell type is a numbered cell.
	If the cell is a numbered cell, return true,
	else return false."

	^ cellAttr == 2! !
```

```
!MineSweeperCell methodsFor: 'testing' stamp: 'BryanArnold 4/15/2018 21:25'!
mineChecker
	"mineChecker Method
	Simple method to check if cell type is a mine.
	If the cell is a mine, return true,
	else return false."

	^ cellAttr == 1! !


!MineSweeperCell methodsFor: 'initialization' stamp: 'BryanArnold 4/15/2018 21:24'!
initialize
	"initialize Method
	This method is the default contrustor for a cell in the Minesweeper game.
	It uses the super classes' initialization, sets every name to 0 (no name), and creates the square
	size and default color that the standard minesweeper game uses. All names, attributes, and values
	are defaulted to 0 as well and all cells are in the off state; very similar to Light's Out
	Example."

  super initialize.
  self label: ' '.
  self borderWidth: 2.
  bounds := 0@0 corner: 16@16.
  offColor := Color lightGray.
  onColor := Color gray darker.

	cellAttr := 0.
	cellVal := 0.
	cellName := ' '.

  self useSquareCorners.
  self turnOff.
	self extent: self extent.! !


!MineSweeperCell methodsFor: 'actions' stamp: 'BryanArnold 4/15/2018 21:22'!
cellUpdate
	"cellUpdate Method.
	This method sets the current name of a cell, and resets the state of the
	cell to whatever its state is now (on/off). Like reupdating a button."

	| ext |
	ext := self extent.
	self label: cellName.
```

```
	self extent: ext.

	self isOn
	ifTrue: [ self turnOn ]
	ifFalse: [ self turnOff ].! !

!MineSweeperCell methodsFor: 'actions' stamp: 'BryanArnold 4/15/2018 21:04'!
showCell
	"showCell Method
	This method simply changes the state of a cell that is defaulted in the off state,
	into a cell that is now in the on state."

	self turnOn.! !


!MineSweeperCell methodsFor: 'event handling' stamp: 'BryanArnold 4/15/2018 21:05'!
mouseUp: anEvent
	"mouseUp Method
	This method takes an event that the mouse caused, such as pressing a cell,
	and executes the action that the logic carries out (depending on cell type)."

  mouseAction value! !
```

```
BorderedMorph subclass: #MineSweeperGame
	instanceVariableNames: 'cells mines'
	classVariableNames: ''
	poolDictionaries: ''
	category: '0-MineSweeperGame'!

!MineSweeperGame methodsFor: 'initialization' stamp: 'BryanArnold 4/15/2018 22:41'!
createMine: n
	"createMine: n Method
	This method takes in an input of the dimension of the board (10 by default). Then, it selects a random position in relation to the row and column of the board and sets a mine in place as long as one is not already there. This makes the mine placement truly random each run of the program. Next, the surrounding cells to the mines are checked to give them their numbers according to connected mines and are updated."

	| cell row col |
	row := n atRandom.
	col := n atRandom.
	cell := cells at: row at: col.

	[ cell mineChecker ]
		whileTrue: [row := n atRandom.
			col := n atRandom.
			cell := cells at: row at: col].

	cell setMine.
	mines add: cell.

	self toggleTouchingCells: row at: col with: [ :x :y |
			(cells at: x at: y) mineChecker not ifTrue: [(cells at: x at: y) increment].
		].! !

!MineSweeperGame methodsFor: 'initialization' stamp: 'BryanArnold 4/15/2018 21:54'!
cellsCount
  "cellsCount Method
	This is just the number of cells along each side of the game, so n = 10.
	Thus, n^2 = 100 cells in the game."

  ^ 10! !

!MineSweeperGame methodsFor: 'initialization' stamp: 'BryanArnold 4/15/2018 22:43'!
newCell: i at: j
  "newCell Method
	This method creates a cell for position (i,j). The surroudings of the cell are then checked as to whether or not it is blank or a mine. The cell created is always shown, if it is a mine show
```

```
all of them and change their states, and finally if it is blank call the changeToCell method to
alter the according number of cells to follow the game logic. Heavily based off the Light's Out
method."

  | c origin |
  c := MineSweeperCell new.
  origin := self innerBounds origin.
  self addMorph: c.

  c position: ((i - 1) * c width) @ ((j - 1) * c height) + origin.
  c mouseAction: [c showCell.
                  c mineChecker ifTrue: [ mines collect: [ :x | x showCell ] ].
                  c blankChecker ifTrue: [self toggleTouchingCells: i at: j
                          with: [:x :y | self changeToCell: x at: y]]].

        ^ c! !

!MineSweeperGame methodsFor: 'initialization' stamp: 'BryanArnold 4/15/2018 21:55'!
minesCount
        "minesCount Method
        this is just the number of mines in the game, 20."

        ^ 20! !

!MineSweeperGame methodsFor: 'initialization' stamp: 'BryanArnold 4/15/2018 22:40'!
initialize
        "initialize Method
        This method initializes the game board. It takes the default game board size of 10 by 10,
along with the default number of mines, which is 20. A default cell is created and using a
Matrix to form a board is made with the same cell. Each cell will start out as a default blank
cell but will be changed later. The LinkedList of mines is created so that when one mine is
pressed, all turn on. This makes it simple to create a board as well as track and manipulate its
parts."

  | sampleCell width height n  k |
  super initialize.
  n := self cellsCount.
        k := self minesCount.
  sampleCell := MineSweeperCell new.
  width := sampleCell width.
  height := sampleCell height.

  self bounds: (5@5 extent: ((width*n) @(height*n)) + (2 * self borderWidth)).
  cells := Matrix new: n tabulate: [ :i :j | self newCell: i at: j ].
        mines := LinkedList new.
```

```
	[ k > 0 ]
		whileTrue:
			[ self createMine: n.
			k := k - 1 ]! !


!MineSweeperGame methodsFor: 'game logic' stamp: 'BryanArnold 4/15/2018 22:30'!
toggleDiagonalsOfCell: i at: j with: f
	"toggleDiagonalsOfCell Method
	This method is the same method as for Lights Out with some tweeks. Instead of needing to toggle each of the neighbours of the cell being checked (clicked on), it calls on each cell diagonal of the position to see if they are present and calls a desired function on it."

  ((i > 1) and: (j > 1)) ifTrue: [ f value: i - 1 value: j - 1].
  ((i < self cellsCount) and: (j > 1)) ifTrue: [ f value: i + 1 value: j - 1].
  ((i > 1) and: (j < self cellsCount)) ifTrue: [ f value: i - 1 value: j + 1].
  ((i < self cellsCount) and: (j < self cellsCount)) ifTrue: [ f value: i + 1 value: j + 1].! !

!MineSweeperGame methodsFor: 'game logic' stamp: 'BryanArnold 4/15/2018 22:34'!
toggleTouchingCells: i at: j with: f

	"toggleTouchingCells Method
	This method calls a specified function that will be called on the surrounding 8 cells and will do the according rules of minesweeper on all of them."

  self toggleNeighboursOfCell: i at: j with: f.
	self toggleDiagonalsOfCell: i at: j with: f.! !

!MineSweeperGame methodsFor: 'game logic' stamp: 'BryanArnold 4/15/2018 22:42'!
changeToCell: i at: j
	"changeToCell Method
	This method is determined cell type and follows the main game logic on what to do and how to change cells.
	If the cell is blank, change its state to on and continue changing neighbours.
	If the cell is numbered cell, change its state to on and stop changing neighbours' states.
	If the cell is a mine, reveal all mines and do nothing else."

	| cell |
	cell := cells at: i at: j.
	(cell mineChecker) ifTrue: [ ^ false ].

	(cell isOn not)
		ifTrue: [
			(cell blankChecker) ifTrue: [ cell showCell.
				self toggleTouchingCells: i at: j with: [:x :y |
					self changeToCell: x at: y] ].
```

```
                    (cell numberedChecker) ifTrue: [ cell showCell ].
                ].

        ^ true.! !

!MineSweeperGame methodsFor: 'game logic' stamp: 'BryanArnold 4/15/2018 22:30'!
toggleNeighboursOfCell: i at: j with: f
        "toggleNeighboursOfCell Method
        This method is the same method as for Lights Out with some tweeks. Instead of needing to toggle each  of the neighbours of the cell being checked (clicked on), it calls on each cell above, below, left,       or right to see if they are present and calls a desired function on it."

   (i > 1) ifTrue: [ f value: i - 1 value: j].
   (i < self cellsCount) ifTrue: [ f value: i + 1 value: j].
   (j > 1) ifTrue: [ f value: i value: j - 1].
   (j < self cellsCount) ifTrue: [ f value: i value: j + 1].! !
```