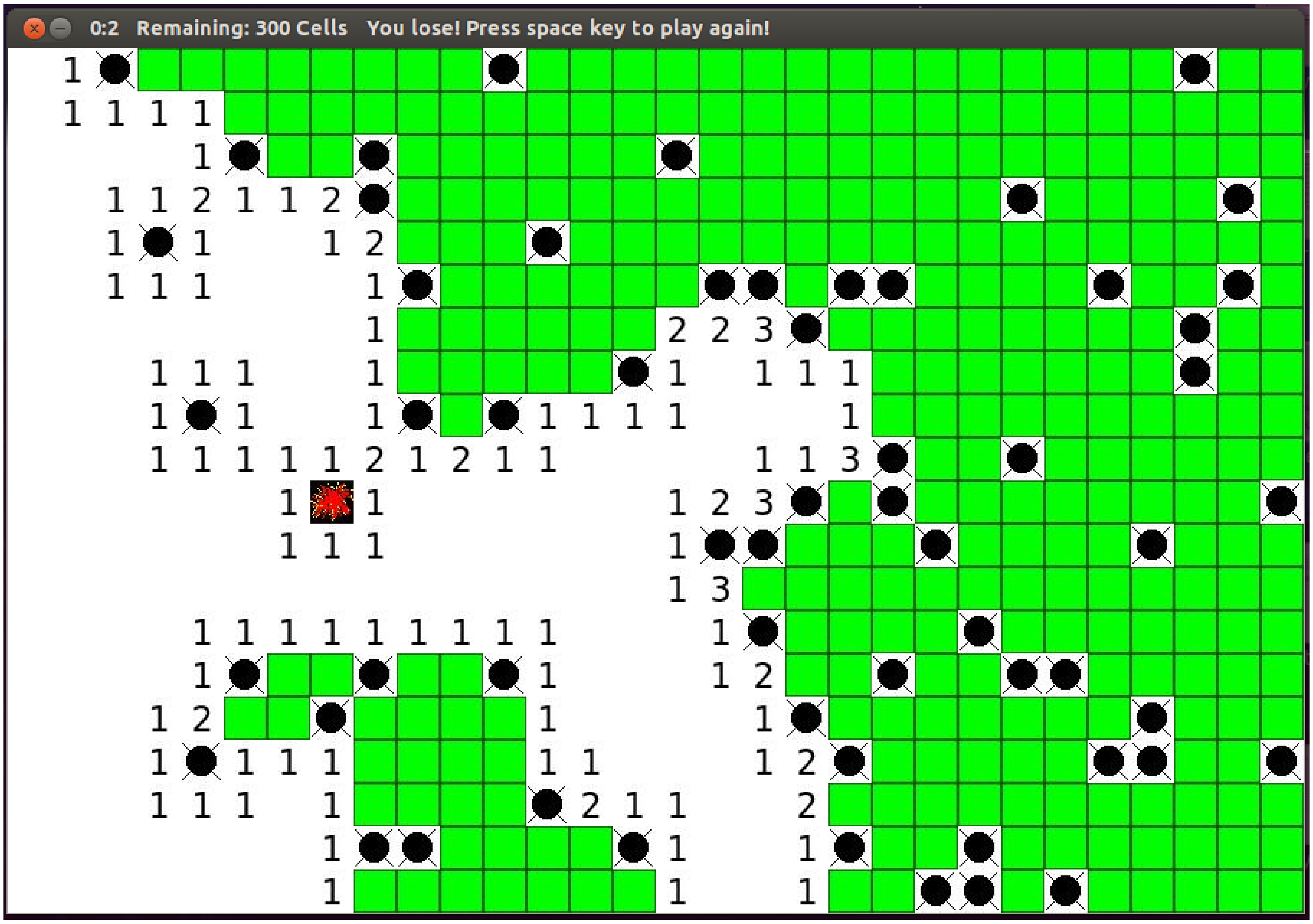
“Introduction to Computational Science”
Final Project
Mine Sweeper Game



0:2 Remaining: 300 Cells You lose! Press space key to play again!

Farhad Ramezanghorbani

Fall 2013

Logic of the Game:

The game is played by revealing squares of the grid by clicking or otherwise indicating each square. If a square containing a mine is revealed, the player loses the game. If no mine is revealed, a digit is instead displayed in the square, indicating how many adjacent squares contain mines; if no mines are adjacent, the square becomes blank. The player uses this information to deduce the contents of other squares, and may either safely reveal each square or mark the square with flag as containing a mine. At the end if all cells which are not included mine revealed, player wins.

This program's code contains:

1. Pygame library which is contains sets of modules and classes for writing games using graphical user interface.
2. A class called Cell, which inherits base class called Rect, which is contains attributes for storing each cell position and length and width (rectangular coordinates), each element in matrix of game is a rect in pygame module. Attributes of cell class are:

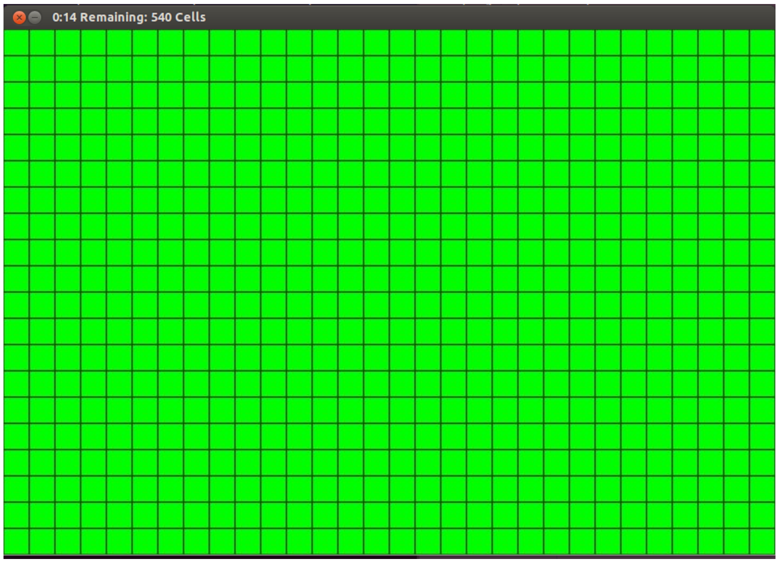
1.	value:	which is number of mines in contact with the element	[0,8]
2.	flag:	which is used for marking possible mine positions	True or False
3.	disp:	situation of a cell – already clicked or not	True or False
4.	bomb	if element contains mine or not	True or False

   Inside of Cell class I have written two function for searching and uncovering each cell after clicking on them.

   Steps (pseudo like-code)

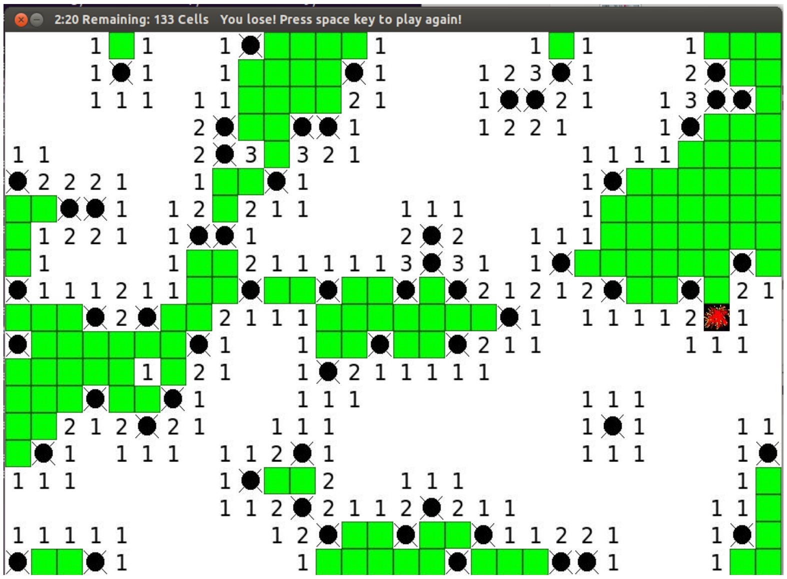
```
– Initializing display
– Defining total number of cells and number of mines
– Importing Images
– Infinite loop until user quit program:
    – Draw initial display with cover image. Set each cell (rect type object - cells[x][y]) as Cell object. Initialize value = 0, flag = false, display = false, bomb = false
    – Distribute mines randomly in grid. Change cells[x][y].bomb to True for each mine.
    – Check each mines neighbors ( 8 neighbors) and if they are not bomb, add 1 to their value.
    – While loop until user wins or loses (while till there is no more non-bomb cell left):
        – Wait for events from user:
            – If the program starts: set a title in queue for every one second (timer; remaining cells; win or lose situation)
            – If user clicked on display
                    – If left bottom get pressed
                        – get the mouse position, find the cell, uncover the cell using the uncover() function inside of class: if it is bomb quit the first loop, if value is in [1,8] show the value, if value is zero:
                            – neighbor check function inside the Cell class will check all 8 neighbors of clicked zero cell to connect all zeros
                    – If right bottom get pressed
                        – flag-unflag the cell

        – If user used quit bottom of screen
            – exit the program
    – After win or lose situations:
    – If user wins:
        – show all the mine positions,  change the title in queue
    – If user loses:
        – specify clicked mine and show all mine positions
    – wait for user to either quit the game or press space
        – if space get pressed: continue infinite while loop  \
```

Initial Condition:



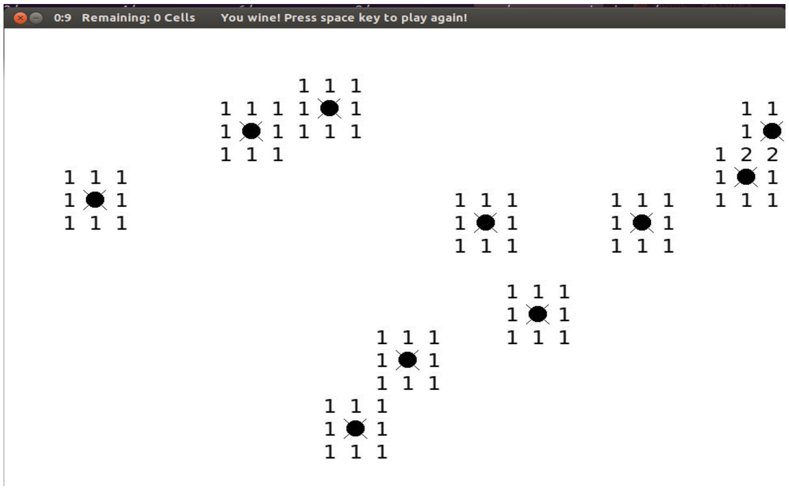
0:14 Remaining: 540 Cells

After clicking on a mine: (red cell)



2:20 Remaining: 133 Cells   You lose! Press space key to play again!

Win situation:



0:9   Remaining: 0 Cells   You wine! Press space key to play again!

Python Code:

```
import sys
from random import *
from pygame import *
from pygame.locals import *
init()

##########################################################################

class Cell(Rect):
        "defining a cell object for each element of matrix"
        def __init__(self,rect):
                                                # Rect class has attributes for storing rectangular coordinates.
                Rect.__init__(self, rect)       # It is a pre-defined class in pygame library
                self.value = 0                  # value : number of mines in neighborhood of cell
                self.flag = False               # if cell is flagged or not
                self.disp = False               # if cell is revealed or not
                self.bomb = False               # if cell is mine or not

        def uncover(self):                      # called after clicking on a cell in grid

                if self.bomb == True:
                        self.disp = True
                        screen.blit(ubomb,self) # copy function in pygame, for copying ubomb on screen
                        display.update()
                        return False

                elif 0 < self.value < 9:
                        self.disp = True
                        screen.blit(numbers[self.value],self)


                elif (self.value == 0 and self.bomb == False):  # when user clicked on zero program should find all
                                                                # connected zeros to the clicked one
                        screen.blit(numbers[self.value],self)   # first it will uncover the clicked one
                        self.disp = True        # the check its neighbors with neighbour_check() function
                                                # if they are values from 0 to 8,uncover them,set disp = True for them
                        display.update()        # add cells with zero values to a list
                        time.wait(100)          # check neghbors of those zeros again

                        source = self.neighbour_check()

                        for item in source:
                                if item.value == 0:
                                        source.extend(item.neighbour_check())
                display.update()

        def neighbour_check(self):

                List = []
                Neighbours = [(-1,-1),(0,-1),(1,-1),(-1,0),(1,0),(-1,1),(0,1),(1,1)]
                a = (self.centerx)/30           # centerx and centery are attributes of pre-defined Rect class
                b = (self.centery)/30
                for (m,n) in Neighbours:
                        if 0<=a+m<x and 0<=b+n<y:
                                if cells[a+m][b+n].disp == False:
                                        if 0 <= cells[a+m][b+n].value < 9:
                                                if cells[a+m][b+n].bomb == False:
                                                        obj = cells[a+m][b+n]
                                                        screen.blit(numbers[obj.value],obj)
                                                        obj.disp = True
                                                        display.update()
                                                        if obj.value == 0:
                                                                List.append(obj)

                return List

######################################## defining display by X x Y cell
# Size of Screen ######################################################
x = 10                                                  # number of vertical cells
y = 10                                                  # number of horizontal cells


screen = display.set_mode((x*30,y*30))                  # output type >> Surface


def  MinesNumber(s):                                    # difficulty (number of mines)

  if s == 'easy':
    return 0.1*x*y
  elif s == 'medium':
    return 0.2*x*y
  elif s == 'hard':
    return 0.3*x*y

####################################### Importing images which are used

numbers = []
for num in range(9):
        name = str('%d.jpg' % num)
        pic     = image.load(name)
        numbers.append(pic)                             # >> surface type will be appended

cover = image.load('cover.jpg')
flag = image.load('flag.jpg')
bomb = image.load('bomb.jpg')
ubomb = image.load('ubomb.jpg')

########################################### Main Program (infinit loop)

while True:

#       level = raw_input("choose level of difficulty as easy, medium and hard:")
        mines = int(MinesNumber('easy'))
        time0 = time.get_ticks()                # get current time when program starts
        time.set_timer(USEREVENT,1000)


        cells = []                              # >> draw initial screen blitting(copying) cover to each cell
                                                # and creating a list of list of Cell objects which
                                                # their type is Rect (out put of blit function is rect type)
        for a in range(x):
                column = []
                for b in range(y):
                        column.append(Cell(screen.blit(cover, (30*a,30*b))))
                cells.append(column)


        MinePos = []                            # assigning random places for mines, change the
        i = 0                                   # value of object rect.bomb to true
        while i < mines:
                xm = randint(0,x-1)
                ym = randint(0,y-1)
                if (xm,ym) not in MinePos:
                        MinePos.append((xm,ym))
                        cells[xm][ym].bomb = True
```

```
            i+=1
                                                    # assigning value to each cell considering their
neighbors
                                                    # for each mine, add 1 to value of its neighbors
                                                    # after this step our rigid initialization will be
finished
    Neighbours = [(-1,-1),(0,-1),(1,-1),(-1,0),(1,0),(-1,1),(0,1),(1,1)]
    for (X,Y) in MinePos:
        for (a,b) in Neighbours:
            if 0<=X+a<x and 0<=Y+b<y:
                if cells[X+a][Y+b].bomb == False:
                    if cells[X+a][Y+b].value < 9:
                        cells[X+a][Y+b].value += 1

    display.update()
    remain = x*y - mines

    while remain:                           # till user can play, if remaining cell equals to zero
                                            # user will win

        move = event.wait()                 # waiting for user to create an event

        if move.type == USEREVENT:          # this will happen continously when program starts

            dt = (time.get_ticks() - time0)/1000
            sec = str(dt%60)
            min = str(dt/60)

            r = 0                           # calculate remainig non-bomb cells
            for i in range(len(cells)):
                for j in range(len(cells[0])):
                    if cells[i][j].disp == False:
                        r +=1
            remain = r - mines

            title = "%.2s:%.2s            Remaining: %d Cells" % (min,sec,remain)
            display.set_caption(title)


        elif move.type == MOUSEBUTTONUP:    # when user clicks on a cell

            (a0,b0) = mouse.get_pos()
            (a,b) = (a0/30,b0/30)

            if cells[a][b].disp == False:   # if it is first time clicking on that cell
                if move.button == 1:        # if user pressed left bottom of mouse
                    cells[a][b].uncover()
                    if cells[a][b].uncover() == False:
                        break

                elif move.button == 3:                      # if user pressed right bottom of mouse
                    if cells[a][b].flag == False:
                        cells[a][b].flag = True
                        display.update(screen.blit(flag, cells[a][b]))
                    else:
                        cells[a][b].flag = False
                        display.update(screen.blit(cover, cells[a][b]))


        if move.type == QUIT:
            quit()
            sys.exit()

    for (m,n) in MinePos:
        if (m,n) != (a,b):
            display.update(screen.blit(bomb, cells[m][n]))
        else:
            display.update(screen.blit(ubomb, cells[m][n]))
    display.set_caption(title+'\tYou lose! Press space key to play again!')

    if remain == 0:
        display.set_caption(title+'\tYou wine! Press space key to play again!')

    while not key.get_pressed()[K_SPACE]:
        if event.wait().type == QUIT: exit()

########################################################################
```