Programming I

Project 3  
Given on Tuesday, 6 November 2014  
Phase 1 demo by Monday, 17 November 2014 at 7:00pm  
Phase 2 demo by Monday, 24 November 2014 at 7:00pm  
Final version due through upload on Thursday, 4 December 2014 at 11:00pm

The Problem

For this project, you will implement a version of the Microsoft’s Minesweeper as described in http://en.wikipedia.org/wiki/Microsoft_Minesweeper using the graphics package.

Concepts and the Initial Set Up

The number of rows, columns, and mines will be provided by the player. With that information, and the constants that you will see shortly, the underlying screen will look like Figure 1. This is a 9 x 9 field, which is considered a beginner-size minesweeper. In general, the field is not necessarily a square. The size of the tiles (cells) and the location of the field are driven by the following constants.

\[
\begin{align*}
\text{TILE\_IMAGE} & = 'tile.gif' \\
\text{FLAG\_IMAGE} & = 'flag.gif' \\
\text{MINE\_IMAGE} & = 'mine.gif' \\
\text{LOSE\_IMAGE} & = 'lose.gif' \\
\text{SMILEY\_IMAGE} & = 'smiley.gif' \\
\text{BLANK\_CELL} & = 0 \\
\text{EXPOSED\_CELL} & = 10 \\
\text{MINE\_CELL} & = 13 \\
\text{MAX\_ADJACENT\_MINES} & = 8 \\
\text{WIDTH\_OF\_IMAGES} & = 32 \\
\text{HEIGHT\_OF\_IMAGES} & = 32 \\
\text{LEFT\_OFFSET} & = 100 \\
\text{RIGHT\_OFFSET} & = 100 \\
\text{TOP\_OFFSET} & = 120 \\
\text{BOTTOM\_OFFSET} & = \text{LEFT\_OFFSET} // 2 \\
\text{X\_OFFSET} & = \text{LEFT\_OFFSET} \\
\text{Y\_OFFSET} & = \text{TOP\_OFFSET}
\end{align*}
\]

The mines are randomly distributed on the above field. The cells that are adjacent to the mines each will contain the count of mines that are adjacent to them. The definition of adjacency is similar to that of your lab 10. That is, suppose there is a mine at grid[i][j]. The cells that are adjacent to it are: grid[i-1][j], grid[i+1][j], grid[i][j-1], grid[i][j+1], grid[i-1][j-1], grid[i-1][j+1], grid[i+1][j-1], and grid[i+1][j+1]. Of
course, some of these cells may be out of bounds, in which case, we do not take them into consideration. Figure 2 illustrates the idea.

![Image of game field with mines](image1.png)

**Figure 2:** The game field with mines

![Image of initial tiles](image2.png)

**Figure 3:** Initially, the tiles cover everything

In the figure, the mine at grid[1][3] is adjacent to grid[0][3], grid[2][3], grid[1][2], grid[1][4], grid[0][2], grid[0][4], grid[2][2], and grid[2][4]. As mentioned above, the cells that are adjacent to mines contain the number of mines that are adjacent to them. Therefore, the mine at grid[1][3] contributes one to the count of each of these cells. That is why grid[0][3], grid[1][2], grid[1][4], grid[0][2], grid[0][4], and grid[2][4] each contain one (1) and that grid[2][2] and grid[2][3] each contain two as they are also adjacent to the mine at grid[3][2].

The tiles are graphics image-objects that get created using image-files and get displayed in the center of cells. You will be given these image-files. As the player clicks on the tiles, we use specific rules to remove one or more tiles to reveal what is behind them or to replace them with the images of mines.

**Details**

As you have observed so far, objects that we draw in a graphics window are based on some internal data structure. For example, in project 2, we had a list of palette-circles and when it was time for a palette color to be selected, we took the player’s mouse-click and searched this list to determine which circle, if any, was clicked. In this project, we also will keep the state of the screen in the internal data structures, mainly two dimensional arrays. We use one such array, let’s call it `game_board_markers`, to represent the type of each cell on the screen. In addition, we use another one, `game_board_images`, to store the actual images that we have displayed on the screen. Therefore, `game_board_markers` is the programmer’s view of the game while `game_board_images` represents part of the player’s view of the game. For example, the user will see the graphics window as depicted in Figure 5. However, this figure is drawn based on the contents of `game_board_markers`, as shown in Figure 4.
In each of these two figures, the cells fall into three categories.

**Blank-cells:** in Figure 4, zeros represent blank-cells (the symbolic constant `BLANK_CELL` as shown earlier). However, in Figure 5, these cells are simply left blank.

**Number-cells:** these cells are adjacent to mines and in either figure, they give us the count of the number of mines that are adjacent to them.

**Mine-cells:** In Figure 4, the value of `MINE_CELL`, which is 13, is used to represent these cells. On the other hand, in Figure 5, the images of the mines are drawn.

`game_board_images` does not contain any images for the blank-cells and the number-cells that are exposed. On the other hand, `game_board_images` contains mine-images at locations `[1][3]`, `[0][4]`, `[1][1]`, `[2][3]`, etc.

As shown in Figure 3, initially all cells are covered with tiles and therefore, each of the elements of `game_board_images` contains an image of a tile. What we do in response to user’s click on these tiles depends on the cell that the tile is covering.

**Blank-cells:** if the user clicks on a tile that covers a blank-cell, we uncover it and all blank-cells that are adjacent to it, by undrawing the tiles that cover them. You will be given an algorithm for this later.

**Number-cells:** clicking on a tile that covers a number-cell will reveal the number by undrawing the tile.

**Mine-cells:** if a tile that covers a mine is clicked, we undraw all the tiles that are placed on top of the mine-cells. In the real game, this will stops the game. However, in this project, you should continue to allow the player to click and uncover tiles. This makes the testing of your programs easier.

As the user clicks on the tiles, we reveal what is under them and we change our markings in `game_board_markers` to represent the now uncovered cell(s). The following two figures show the idea. Compare Figures 4 and 5 with Figures 6 and 7.
In Figure 7, a number of underlying squares have been exposed because the player has clicked on certain tiles. Each exposed tile in Figure 7 is represented by value 10, \texttt{EXPOSED\_CELL}, in Figure 6. The cells in \texttt{game\_board\_images} that correspond to the cells that contains 10 in \texttt{game\_board\_markers} no longer have tile images in them. Here is an algorithm that summarizes the actions that your program should take when the player clicks on the screen.

\begin{verbatim}
  click_point = win.getMouse()
  row, column = convert_click_to_row_column(click_point)
  if row is None:
    ignore this click (continue to the next iteration of the loop)
  elif game_board_markers[row][column] == EXPOSED\_CELL
    ignore this click (continue to the next iteration of the loop)
  elif game_board_markers[row][column] is a number between 1 and 8
    undraw the tile at game_board_images[row][column]
    game_board_markers[row][column] = EXPOSED\_CELL
    game_board_images[row][column] = None
  elif game_board_markers[row][column] is a mine
    expose_all_mines(game_board_markers, game_board_images, win)
  elif game_board_markers[row][column] is an empty cell
    expose_empty_cells(game_board_markers, game_board_images, row, column, win)
\end{verbatim}

Creating and drawing images in the graphics window is simple. For example, the following code-segment displays a tile on the screen with its center at $(x, y)$. \texttt{TILE\_IMAGE} is a constant literal that you saw earlier.

\begin{verbatim}
  tile_image = Image(Point(x, y), TILE\_IMAGE)
  tile_image.draw(win)
\end{verbatim}
Input

The only input to the program is a string representing the level of the game. The acceptable values are beginner, intermediate, and expert. These levels correspond to the following number of rows, columns, and mines.

# Beginner game
rows = 9
columns = 9
num_mines = 10

# Intermediate
rows = 16
columns = rows
num_mines = 40

# Expert
rows = 16
columns = 30
num_mines = 99

Phase 1 — 50 points

For phase 1, you should be able to write the necessary code to accept the player’s input and use it to generate a game board similar to Figure 5. Recall that the board in Figure 5 represents a beginner level game. In addition, you should print the image of your game_board_markers as shown above. The following is a list of functions that you may find useful for this phase.

- Given the number of rows and columns, the following function create a two dimensional array and returns it.

  def create_minesweeper_matrix(rows, columns):

- The following function print the contents of the game_board_markers as depicted in Figure 4.

  def print_game_board(game_board_markers):

- All cells of the game_board_markers that is passed to the following function contain zeros. This function randomly selects number_of_mines cells of this two dimensional array and designates them as mine-cells. On return, number_of_mines cells of game_board_markers will contain MINE_CELL.

  def populate_with_mines(game_board_markers, number_of_mines):

- game_board_markers[row][column] contains a mine. This function adds one to the count of each of the neighbors of game_board_markers[row][column]. Of course, if any of the neighbors is a mine, that element will not be affected by this function.

  def update_neighbor_count(game_board_markers, row, column):
• The `game_board_markers` comes to this function with the locations of the mines marked (by `populate_with_mines` function). This function, using `update_neighbor_count`, counts and stores the number of mines that are adjacent to each element of `game_board_markers`.

```python
def add_mine_counts(game_board_markers):
```

• Draws the lines that are shown in Figure 1.

```python
def draw_the_grid(rows, columns, win):
```

• After having called `add_mine_counts`, this function draws the number-cells on the graphics window.

```python
def draw_board_numbers(game_board_markers, game_board_images, win):
```

Phase 2 — 100 points
For this phase, you should start the game as shown in Figure 3. That is, you will draw the grid-lines on the graphics window, add the mines, populate the cells that are adjacent to the mines, and cover them all with tiles. Then, you accept the player’s mouse-clicks and determine the row and the column of the cell on which the mouse was clicked. Finally, if the click is in one of the cells that is not exposed, expose that cell. After this, you print the contents of `game_board_markers` as shown above.

Complete Project — 100 points
In the complete version of the project, if the player was to click on a mine, all mines get exposed. In addition, if the player was to click on a tile that covers a blank space, all blank spaces adjacent to it will be exposed. The final version will also correctly identify when the game is over.

How to Submit the Final Version
You should submit the final version of this project using the upload link that is available to you through this course’s projects web-page. Before submitting it, be sure that you have the right version, your name is record at the top of the file that you are about to submit, and you choose the correct label on the web-site.

Grading
The most important part of your grade is the correctness of your final program. Your program will be tested numerous times, using different inputs, to be sure that it meets the specifications. A correct program will receive 85% of the grade for this project. The remaining 15% is for programming style and design. This includes good design, use of spaces between operators and operands, comments, and the use of descriptive variable-names.
Collaboration policy

Programming projects must be your own work, and academic misconduct is taken very seriously. You may discuss ideas and approaches with other students and the course staff, but you should work out all details and write up all solutions on your own. **The following actions will be penalized as academic dishonesty:**

- Copying part or all of another student’s assignment.
- Copying old or published solutions.
- Looking at another student’s code or discussing it in great detail. You will be penalized if your program matches another student’s program too closely.
- Showing your code or describing your code in great detail to anyone other than the course staff or tutor.