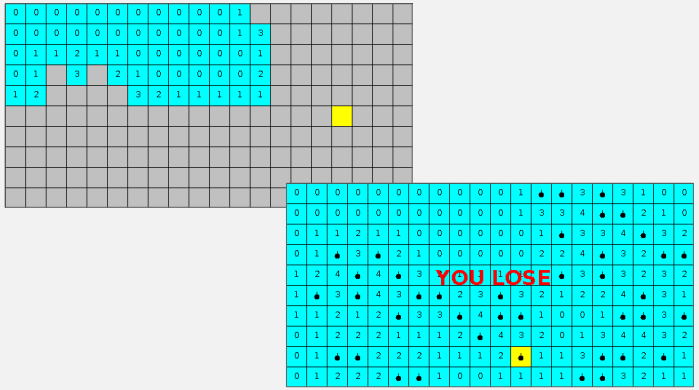
Minesweeper – Demo and Game Play



Move the yellow square (the current position) using the arrow keys (make sure the mouse is within the program's window before you start pressing keys); press the space bar to uncover the square at the current position.

Minesweeper – Dealing With 2D Arrays

- how do you access one slot of a 2D array?

```
arr[row][col]
```

- how do you go through every slot of a 10x20 2D array?

```
for ( int row = 0 ; row < 10 ; row++ ) {
  for ( int col = 0 ; col < 20 ; col++ ) {
    // do something with arr[row][col]
  }
}
```

think:

```
repeat for each row 0..numrows-1
  // go through all the columns on that row
  repeat for each col 0..numcols-1
    …
```


Minesweeper – Dealing With 2D Arrays

To keep track of the necessary information in order to play the game, you will need to have several variables:

- a 2D array storing whether or not each square has been uncovered
- a 2D array storing whether or not a square contains a mine
- a 2D array storing the number of adjacent mines for each square
- the row and column of the player's current position on the board

Each of the 2D arrays should have 10 rows and 20 columns.

- initialization
  - whether or not each square has been uncovered
    - boolean – start all false (not uncovered)
  - whether or not a square contains a mine
    - boolean – start all false (no mine), then place mines
  - number of adjacent mines
    - int – start 0 (no adjacent mines), then count once the mines have been placed

Minesweeper – Dealing With 2D Arrays

To keep track of the necessary information in order to play the game, you will need to have several variables:

- a 2D array storing whether or not each square has been uncovered
- a 2D array storing whether or not a square contains a mine
- a 2D array storing the number of adjacent mines for each square
- the row and column of the player's current position on the board

Each of the 2D arrays should have 10 rows and 20 columns.

- update
  - whether or not each square has been uncovered
    - set to true for the current position when the player presses the space bar
  - whether or not a square contains a mine
    - not updated after initialization
  - number of adjacent mines
    - not updated after initialization

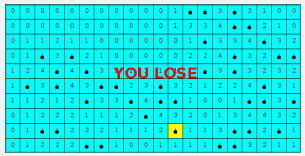
Minesweeper – Placing Random Mines

- There should be 40 hidden mines, placed randomly on the board. (A square can contain at most one mine, so you should be careful not to put a mine in a square that already has one when you are distributing the mines.)

- is there repetition here?
  - repeat attempt to place a mine until 40 mines have been placed
- how do you choose a random position on the board?
  - choose a random row and a random column
- how do you know if there's already a mine in that position?
  - there's an array for that...
- how do you place the mine in a position if there's not a mine there already?
  - update the mines array only if there's not a mine there already

Minesweeper – Determining Adjacent Mines

- Pressing the space bar should uncover the current square, revealing the number of mines adjacent to the square if it isn't a mine.
  - a 2D array storing the number of adjacent mines for each square

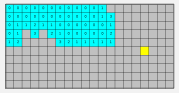


- is there repetition here?
  - need to compute for each square
- how do you count the number of adjacent mines for a particular square?
  - look at all the neighboring squares and count if they have a mine
- how do you know the positions of the adjacent squares?
  - given (row,col), think about what is above, above and right, right, below and right, below, etc
- how do you not fall off the edge of array?
  - check that the neighbor is within the array before looking to see if the neighbor has a mine

Minesweeper – Graphics

- how to make the board correspond to the arrays?

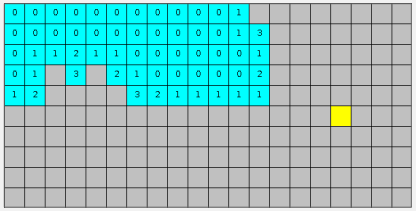
You can draw the game board by going through each slot of the 10x20 grid and drawing the corresponding square in the drawing window — use the information stored in that slot in each of the two arrays to determine how to draw the square (covered or uncovered, number of mines or bomb). You can use `fillRect` to draw the square (and `strokeRect` to draw an outline around it, to distinguish one square from another), however, you will need to know the position of the upper left corner of each rectangle in order to draw it. You can work out a formula for converting from the row and column of the grid cell to the *y* and *x* coordinates for the rectangle (keep in mind that the *x* coordinate corresponds to the column and the *y* coordinate corresponds to the row — this is easy to mix up!), but a simpler option is to treat both the row and column of the grid cell and the *x* and *y* coordinates of the rectangle as things that change in the loops and initialize/update them accordingly. (The `ColorSpots` example from class illustrates this — `i` is the counter going through the array indexes and `x` is the *x* coordinate for the current oval.)



  - go through each square of the grid
    - this is the going through every slot of a 2D array pattern…
  - to start: draw a gray box with a black outline for each square
    - in addition to the row and col loop variables, also have x and y loop variables that have the right value for the upper left corner of the square for position (row,col)
    - be careful to avoid mixups – row corresponds to y, col corresponds to x
  - then: if the square is uncovered, draw a cyan box with a black outline and the number of adjacent mines, otherwise draw a gray box with a black outline for each square
    - how do you know if a square is uncovered, or the number of adjacent mines?
      - for square (row,col), look at that slot in the appropriate array

Minesweeper – Graphics

- it seems tedious to draw 200 squares…
  - loops!
- how to visually distinguish the current position?
  - when you are drawing squares, consider another alternative for what is drawn – if the square is the current position, …



Minesweeper – Graphics

- how to handle the animation?
  - there is no animation – draw the board when state changes and you need to update the display
  - when to update the display?
    - after setup and after the user's key press has been handled

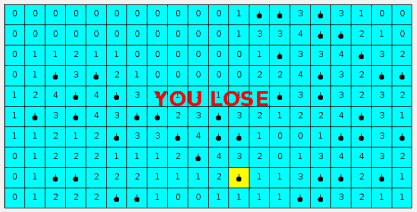
```
// set up the game

// play the game
// -- repeat until the game is over
// -- .. get the user's key press
// -- .. handle the user's key press
// -- display win or lose message
```

draw the board

- how to handle boxes disappearing?
  - nothing disappears – draw the entire board each time

Minesweeper – Graphics

- it seems like it will be time-consuming to come up with a symbol for the uncovered mines
  - it doesn't have to be complicated – the demo's is just a circle and a line, but even just a single shape is fine



- how to make redrawing the board after each turn elegant?
  - it happens after every turn but the code is only written once within the main loop
  - can create a subroutine to avoid repetition after setup and within loop, but that isn't required

Minesweeper – User Input

- how do we get the user's key presses?

Your program will need to repeatedly get input specifying the player's next move. Since this is a graphical program, you won't use `Scanner`; instead, the provided `Minesweeper` template includes `getKeyCode()` to support getting the user's key presses. (Normally GUIs utilize what is known as *event-driven programming*, but that is beyond the scope of this course, so `getKeyCode()` is a workaround.)

Like the `Scanner` operations, call `getKeyCode()` when you want user input; it will waits until a key is pressed and then return a value indicating which key was the one pressed. As with `Scanner`, you'll probably want to store that value in a variable so you can work with it:

```
KeyCode key = getKeyCode();
```

Then, to determine what key was pressed:

```
if ( key == KeyCode.LEFT ) {     // left arrow pressed
  ...
}
```

Checking for the other possibilities (`KeyCode.RIGHT`, `KeyCode.UP`, `KeyCode.DOWN`, or `KeyCode.SPACE`) is similiar.

(If you do the "flag" option for extra credit, you'll need to check another key: `KeyCode.F`)

Minesweeper – User Input

- how do the arrow keys select certain positions?

- Make a copy of `Minesweeper.java` and name the new program `InputTrial`.
- Write code for the part of the game that deals with reading key presses and updating the current position. (This will go after the `// *** drawing goes here!` comment even though there's not any actual drawing going on.) Utilize incremental development and do this in three stages:
  - Write code to repeatedly get the user's key press, check which key was pressed, and print out "left arrow!", "right arrow!", "down arrow!", "up arrow!", or "space!" accordingly.
  - Declare two variables for the current row and column and initialize them to zero. Update them when arrow keys are pressed — left arrow decreases the column by 1, down arrow increases the row by 1, and so forth. Print out the current row and column after each update.
  - Address robustness: in the game, the user shouldn't be able to move the current position off the edges of the board. That means if the current column is column 0, pressing the left arrow shouldn't change the current column; if the current row is row 0, pressing the up arrow shouldn't change the current row; and so forth. The board is 10 rows by 20 columns.

Minesweeper – User Input

- it seems tedious to have to move arrow one square at a time, is there a way to jump several squares at once?
  - it's not really that bad, as often the next square uncovered is near the last
  - a possible extra credit option...

Minesweeper – Robustness

- how do we keep the current position from going off the board?
  - Make a copy of `Minesweeper.java` and name the new program `InputTrial`.
  - Write code for the part of the game that deals with reading key presses and updating the current position. (This will go after the `// *** drawing goes here!` comment even though there's not any actual drawing going on.) Utilize incremental development and do this in three stages:
    - Write code to repeatedly get the user's key press, check which key was pressed, and print out "left arrow!", "right arrow!", "down arrow!", "up arrow!", or "space!" accordingly.
    - Declare two variables for the current row and column and initialize them to zero. Update them when arrow keys are pressed — left arrow decreases the column by 1, down arrow increases the row by 1, and so forth. Print out the current row and column after each update.
    - Address robustness: in the game, the user shouldn't be able to move the current position off the edges of the board. That means if the current column is column 0, pressing the left arrow shouldn't change the current column; if the current row is row 0, pressing the up arrow shouldn't change the current row; and so forth. The board is 10 rows by 20 columns.

Minesweeper – Robustness

- there are a million things the user could do to mess with the program, how do we deal with all of those things?
  - there aren't really that many things…
    - moving the current position off the edge
    - pressing a key other than an arrow or space bar
    - ??

Minesweeper – Game Over

- The game should end if the player uncovers a mine (the player loses) or if the player has uncovered all of the squares not containing a mine (the player wins).

```
// set up the game

// play the game
// -- repeat until the game is over
// -- .. get the user's key press
// -- .. handle the user's key press
// -- display win or lose message
```

- when do you detect that a mine has been uncovered?
  - when handling the space bar
- what should happen in that case?
  - exit the main game play loop and print "you lose!"
- when would whether or not all of the non-mine squares have been uncovered possibly change status?
  - after uncovering a square i.e. when handling the space bar
- how do you know whether all the non-mine squares are uncovered?
  - go through every square in the grid, checking whether there is an uncovered position with a mine, or
  - you know how many non-mine squares there are, so keep a count of how many are uncovered
- what should happen in that case?
  - exit the main game play loop and print "you win!"

Minesweeper – Managing a Larger Program

- start with pseudocode
- incremental development – one piece at a time, testing after each
- implement simpler versions first
  - e.g. just draw squares on the board, then add uncovered status, then …
  - e.g. a single shape for a mine

Minesweeper – Debugging

- incremental development – one piece at a time, testing after each