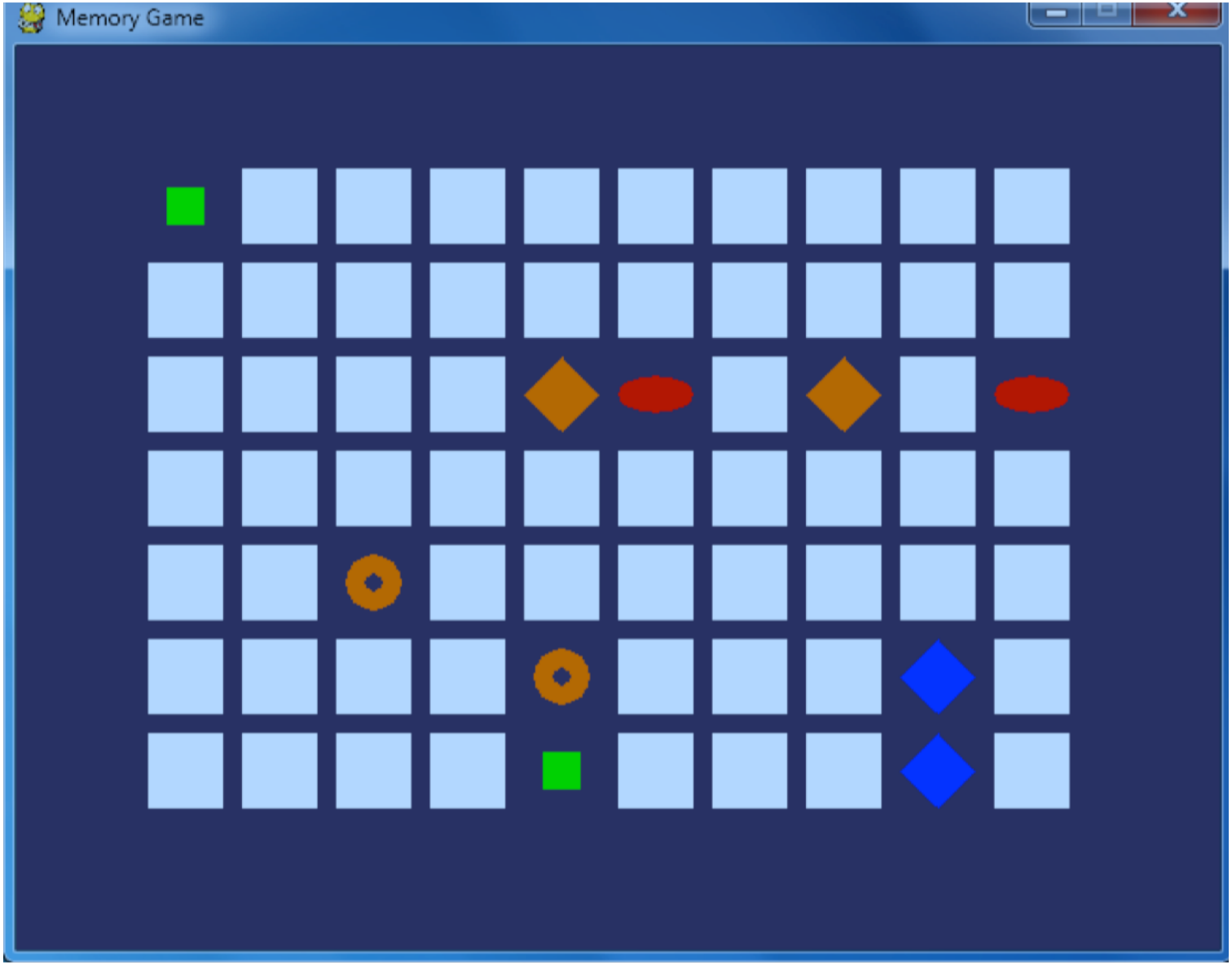
Memory Puzzle


Memory Game

How to Play Memory Puzzle

- In the Memory Puzzle game, several icons are covered up by white boxes. There are two of each icon. The player can click on two boxes to see what icon is behind them. If the icons match, then those boxes remain uncovered. The player wins when all the boxes on the board are uncovered. To give the player a hint, the boxes are quickly uncovered once at the beginning of the game.

Nested for Loops

```
>>> for x in [0, 1, 2, 3, 4]:
...     for y in ['a', 'b', 'c']:
...         print(x, y)
```

vs

```
>>> for y in ['a', 'b', 'c']:
...     for x in [0, 1, 2, 3, 4]:
...         print(x, y)
```

Credits and Imports

```
1. # Memory Puzzle
2. # By Al Sweigart al@inventwithpython.com
3. # http://inventwithpython.com/pygame
4. # Released under a "Simplified BSD" license
5.
6. import random, pygame, sys
7. from pygame.locals import *
```

- At the top of the program are comments about what the game is, who made it, and where the user could find more information.
- This program makes use of many functions in other modules, so it imports those modules on line 6. Line 7 is also an import statement in the from (module name) import * format, which means you do not have to type the module name in front of it. There are no functions in the pygame.locals module, but there are several constant variables in it that we want to use such as MOUSEMOTION, KEYUP, or QUIT. Using this style of import statement, we only have to type MOUSEMOTION rather than pygame.locals.MOUSEMOTION.

Magic Numbers

```
 9. FPS = 30 # frames per second, the general speed of the program
10. WINDOWWIDTH = 640 # size of window's width in pixels
11. WINDOWHEIGHT = 480 # size of windows' height in pixels
12. REVEALSPEED = 8 # speed boxes' sliding reveals and covers
13. BOXSIZE = 40 # size of box height & width in pixels
14. GAPSIZE = 10 # size of gap between boxes in pixels
```

- There are two reasons to use constant variables.
  - First, if we ever wanted to change the size of each box later, we would have to go through the entire program and find and replace each time we typed 40.
  - Second, it makes the code more readable.

```
XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE + GAPSIZE))) / 2)
```

VS

Magic numbers

```
XMARGIN = int((640 - (10 * (40 + 10))) / 2)
```

Sanity Checks with assert Statements

```
15. BOARDWIDTH = 10 # number of columns of icons
16. BOARDHEIGHT = 7 # number of rows of icons
17. assert (BOARDWIDTH * BOARDHEIGHT) % 2 == 0, 'Board needs to have an even
number of boxes for pairs of matches.'
18. XMARGIN = int((WINDOWWIDTH - (BOARDWIDTH * (BOXSIZE + GAPSIZE))) / 2)
19. YMARGIN = int((WINDOWHEIGHT - (BOARDHEIGHT * (BOXSIZE + GAPSIZE))) / 2)
```

- The assert statement on line 15 ensures that the board width and height we've selected will result in an even number of boxes (since we will have pairs of icons in this game). There are three parts to an assert statement: the assert keyword, an expression which, if False, results in crashing the program. The third part (after the comma after the expression) is a string that appears if the program crashes because of the assertion.
- The assert statement with an expression basically says, —The programmer asserts that this expression must be True, otherwise crash the program.

Crash Early and Crash Often!

```
 Traceback (most recent call last):
  File "C:\book2svn\src\memorypuzzle.py", line 292, in <module>
    main()
  File "C:\book2svn\src\memorypuzzle.py", line 58, in main
    mainBoard = getRandomizedBoard()
  File "C:\book2svn\src\memorypuzzle.py", line 149, in getRandomizedBoard
    columns.append(icons[0])
IndexError: list index out of range
```

- If the values we chose for BOARDWIDTH and BOARDHEIGHT that we chose on line 15 and 16 result in a board with an odd number of boxes (such as if the width were 3 and the height were 5), then there would always be one left over icon that would not have a pair to be matched with. This would cause a bug later on in the program, and it could take a lot of debugging work to figure out that the real source of the bug is at the very beginning of the program. In fact, just for fun, try commenting out the assertion so it doesn't run, and then setting the BOARDWIDTH and BOARDHEIGHT constants both to odd numbers. When you run the program, it will immediately show an error happening on a line 149 in memorypuzzle.py, which is in getRandomizedBoard() function!

Crash Early and Crash Often!

- The assertion makes sure that this never happens. If our code is going to crash, we want it to crash as soon as it detects something is terribly wrong, because otherwise the bug may not become apparent until much later in the program. Crash early!
- You want to add assert statements whenever there is some condition in your program that must always, always, always be True. Crash often!

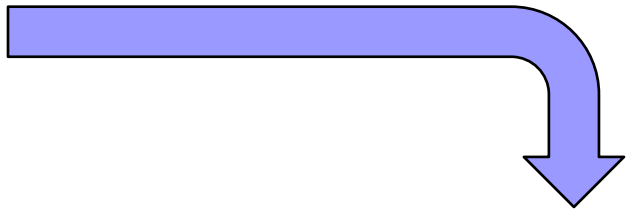
Making the Source Code Look Pretty

```
21. #                R    G    B
22. GRAY      = (100, 100, 100)
23. NAVYBLUE  = ( 60,  60, 100)
24. WHITE     = (255, 255, 255)
25. RED       = (255,   0,   0)
26. GREEN     = (  0, 255,   0)
27. BLUE      = (  0,   0, 255)
28. YELLOW    = (255, 255,   0)
29. ORANGE    = (255, 128,   0)
30. PURPLE    = (255,   0, 255)
31. CYAN      = (  0, 255, 255)
32.
33. BGCOLOR = NAVYBLUE
34. LIGHTBGCOLOR = GRAY
35. BOXCOLOR = WHITE
36. HIGHLIGHTCOLOR = BLUE
```

- In Python the indentation (that is, the space at the beginning of the line) needs to be exact, but the spacing in the rest of the line is not so strict.

Using Constant Variables Instead of Strings

```
38. DONUT = 'donut'
39. SQUARE = 'square'
40. DIAMOND = 'diamond'
41. LINES = 'lines'
42. OVAL = 'oval'
```



```
if shape == DONUT:
```

```
if shape == DUNOT:
```

vs

```
if shape == 'dunot':
```

Making Sure We Have Enough Icons

```
44. ALLCOLORS = (RED, GREEN, BLUE, YELLOW, ORANGE, PURPLE, CYAN)
45. ALLSHAPES = (DONUT, SQUARE, DIAMOND, LINES, OVAL)
46. assert len(ALLCOLORS) * len(ALLSHAPES) * 2 >= BOARDWIDTH * BOARDHEIGHT,
"Board is too big for the number of shapes/colors defined."
```

- Make a tuple that holds all of the values for every possible color and shape to be able to create icons
- Assertion on line 46 makes sure that there are enough color/shape combinations for the size of the board we have.

Tuples vs. Lists, Immutable vs. Mutable

```
>>> listVal = [1, 1, 2, 3, 5, 8]
>>> tupleVal = (1, 1, 2, 3, 5, 8)
>>> listVal[4] = 'hello!'
>>> listVal
[1, 1, 2, 3, 'hello!', 8]
>>> tupleVal[4] = 'hello!'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> tupleVal
(1, 1, 2, 3, 5, 8)
>>> tupleVal[4]
5
```

- Lists - **mutable** (they can be changed)
- Tuples - **immutable** (they cannot be changed)

Tuples vs. Lists, Immutable vs. Mutable

- Lists - **mutable** (they can be changed)
- Tuples - **immutable** (they cannot be changed)
- Silly benefit – the code that uses tuples is slightly faster than code that uses lists (Python optimizations)
- Important benefit - it's a sign that the value in the tuple will **never change**, and the value in the list **could be modified**

"If I see a list value, I know that it could be modified at some point in this program. Otherwise, the programmer who wrote this code would have used a tuple."

"I can expect that this tuple will always be the same. Otherwise the programmer would have used a list."

Tuples vs. Lists, Immutable vs. Mutable

- A new tuple value can be assigned to a variable - overwriting the old tuple value
- Strings are also an immutable data type. You can use the square brackets to read a single character in a string, but you cannot change a single character in a string:

```
>>> strVal = 'Hello'
>>> strVal[1]
'e'
>>> strVal[1] = 'X'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
```

One Item Tuples Need a Trailing Comma

- In the case of a single element tuple, the trailing comma is required:

```
oneValueTuple = (42, )
```

```
variableA = (5 * 6)  vs  variableB = (5 * 6, )
```

- Blank tuple values do not need a comma in them, they can just be a set of parentheses by themselves: ()
- In general, the trailing comma for tuples, lists, or function arguments is good style

Converting Between Lists and Tuples

- list() function
- tuple() function

```
>>> spam = (1, 2, 3, 4)
>>> spam = list(spam)
>>> spam
[1, 2, 3, 4]
>>> spam = tuple(spam)
>>> spam
(1, 2, 3, 4)
>>>
```

The global statement, and Why Global Variables are Evil

- Rules:
  - If there is a global statement for a variable at the beginning of the function, then the variable is global.
  - If the name of a variable in a function has the same name as a global variable and the function never assigns the variable a value, then that variable is the global variable.
  - If the name of a variable in a function has the same name as a global variable and the function does assign the variable a value, then that variable is a local variable.
  - If there isn't a global variable with the same name as the variable in the function, then that variable is obviously a local variable.

The global statement, and Why Global Variables are Evil

- Example:

```
48. def main():
49.     global FPSCLOCK, DISPLAYSURF
50.     pygame.init()
51.     FPSCLOCK = pygame.time.Clock()
52.     DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
53.
54.     mousex = 0 # used to store x coordinate of mouse event
55.     mousey = 0 # used to store y coordinate of mouse event
56.     pygame.display.set_caption('Memory Game')
```

The global statement, and Why Global Variables are Evil

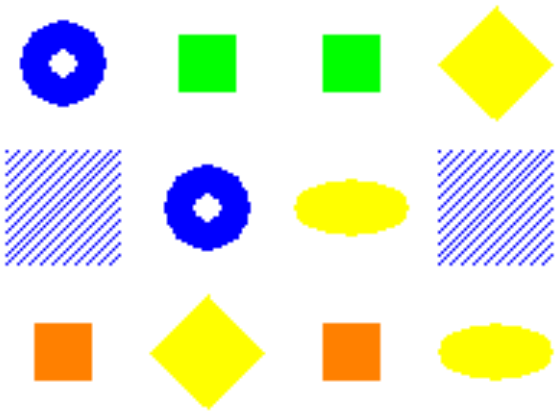
- Avoid using global variables inside functions
  - Since the global variable could have been modified in many places before the function was called, it can be tricky to track down a bug involving a bad value set in the global variable.
  - It also makes changing the code in a function easier

Data Structures and 2D Lists

```
58.     mainBoard = getRandomizedBoard()
59.     revealedBoxes = generateRevealedBoxesData(False)
```

- getRandomizedBoard() - returns a data structure that represents the state of the board (2D list)
- generateRevealedBoxesData() function returns a data structure that represents which boxes are covered (2D list)

Data Structures and 2D Lists



```
mainBoard = [[(DONUT, BLUE), (LINES, BLUE), (SQUARE, ORANGE)], [(SQUARE,
GREEN), (DONUT, BLUE), (DIAMOND, YELLOW)], [(SQUARE, GREEN), (OVAL, YELLOW),
(SQUARE, ORANGE)], [(DIAMOND, YELLOW), (LINES, BLUE), (OVAL, YELLOW)]]
```

- mainBoard[x][y] will correspond to the icon at the (x, y) coordinate on the board
- mainBoard[4][5] - get the icon on the board at the position (4, 5)

The “Start Game” Animation

```
61.     firstSelection = None # stores the (x, y) of the first box clicked.
62.
63.     DISPLAYSURF.fill(BGCOLOR)
64.     startGameAnimation(mainBoard)
```

- firstSelection - When the player clicks on an icon on the board, the program needs to track if this was the first icon of the pair that was clicked on or the second icon
- Line 63 fills the entire surface with the background color
- startGameAnimation() - at the beginning of the game, all of the boxes are quickly covered and uncovered randomly to give the player a sneak peek at which icons are under which boxes.

The Game Loop

```
66.     while True: # main game loop
67.         mouseClicked = False
68.
69.         DISPLAYSURF.fill(BGCOLOR) # drawing the window
70.         drawBoard(mainBoard, revealedBoxes)
```

- The game loop handles events, updates the game state, and draws the game state to the screen.

The Game Loop

- The game state for the Memory Puzzle program is stored in the following variables:
  - mainboard
  - revealedBoxes
  - firstSelection
  - mouseClicked
  - Mousex
  - mousey
- the mouseClicked variable stores a Boolean value that is True if the player has clicked the mouse during this iteration through the game loop

The Game Loop

```
66.     while True: # main game loop
67.         mouseClicked = False
68.
69.         DISPLAYSURF.fill(BGCOLOR) # drawing the window
70.         drawBoard(mainBoard, revealedBoxes)
```

- The surface is painted over with the background color to erase anything that was previously drawn on it. The program then calls drawBoard() to draw the current state of the board based on the board and “revealed boxes” data structures that we pass it. (These lines of code are part of drawing and updating the screen.)

The Event Handling Loop

```
72.         for event in pygame.event.get(): # event handling loop
73.             if event.type == QUIT or (event.type == KEYUP and event.key ==
K_ESCAPE):
74.                 pygame.quit()
75.                 sys.exit()
76.             elif event.type == MOUSEMOTION:
77.                 mousex, mousey = event.pos
78.             elif event.type == MOUSEBUTTONUP:
79.                 mousex, mousey = event.pos
80.                 mouseClicked = True
```

- Line 72 executes code for every event that has happened since the last iteration of the game loop. This loop is called the **event handling loop** and iterates over the list of pygame.Event objects returned by the pygame.event.get() call.

The Event Handling Loop

- QUIT event or a KEYUP event for the Esc key - the program should terminate
- MOUSEMOTION event or MOUSEBUTTONUP event - the position of the mouse cursor should be stored in the **mousex** and **mousey** variables
- MOUSEBUTTONUP event - **mouseClicked** should also be set to **True**

Checking Which Box The Mouse Cursor is Over

```
82.         boxx, boxy = getBoxAtPixel(mousex, mousey)
83.         if boxx != None and boxy != None:
84.             # The mouse is currently over a box.
85.             if not revealedBoxes[boxx][boxy]:
86.                 drawHighlightBox(boxx, boxy)
```

- The getBoxAtPixel() function will return the XY board coordinates of the box that the mouse coordinates are over
  - If the mouse cursor was not over any box (for example, if it was off to the side of the board or in a gap in between boxes) then the tuple (None, None) is returned by the function

Checking Which Box The Mouse Cursor is Over

- The if statement on line 85 checks if the box is covered up or not by reading the value stored in revealedBoxes[boxx][boxy]
- Whenever the mouse is over a covered up box, we want to draw a blue highlight around the box to inform the player that they can click on it (drawHighlightBox() function). This highlighting is not done for boxes that are already uncovered.

Checking Which Box The Mouse Cursor is Over

- The getBoxAtPixel() function will return the XY board coordinates of the box that the mouse coordinates are over
  - If the mouse cursor was not over any box (for example, if it was off to the side of the board or in a gap in between boxes) then the tuple (None, None) is returned by the function

Checking Which Box The Mouse Cursor is Over

```
87.                 if not revealedBoxes[boxx][boxy] and mouseClicked:
88.                     revealBoxesAnimation(mainBoard, [(boxx, boxy)])
89.                     revealedBoxes[boxx][boxy] = True # set the box as
"revealed"
```

- Line 87 - check if the mouse cursor is not only over a covered up box but if the mouse has also been clicked. In that case, we want to play the “reveal” animation for that box by calling revealBoxesAnimation() function - only draws the animation of the box being uncovered
- Line 89 - the data structure that tracks the game state is updated.

Handling the First Clicked Box

```
 90.                 if firstSelection == None: # the current box was the first
box clicked
 91.                     firstSelection = (boxx, boxy)
 92.                 else: # the current box was the second box clicked
 93.                     # Check if there is a match between the two icons.
 94.                     icon1shape, icon1color = getShapeAndColor(mainBoard,
firstSelection[0], firstSelection[1])
 95.                     icon2shape, icon2color = getShapeAndColor(mainBoard,
boxx, boxy)
```

- If line 90's condition is True - this is the first of the two possibly matching boxes that was clicked
  - play the reveal animation for the box
  - keep that box uncovered
  - set the firstSelection variable to a tuple of the box coordinates for the box that was clicked

Handling the First Clicked Box

- If this is the second box the player has clicked on
  - play the reveal animation for that box
  - then check if the two icons under the boxes are matching - the getShapeAndColor() function retrieves the shape and color values of the icons (one of the values in the ALLCOLORS and ALLSHAPES tuples)

Handling a Mismatched Pair of Icons

```
 97.                     if icon1shape != icon2shape or icon1color !=
icon2color:
 98.                         # Icons don't match. Re-cover up both selections.
 99.                         pygame.time.wait(1000) # 1000 milliseconds = 1 sec
100.                         coverBoxesAnimation(mainBoard,
[(firstSelection[0], firstSelection[1]), (boxx, boxy)])
101.                         revealedBoxes[firstSelection[0]][firstSelection
[1]] = False
102.                         revealedBoxes[boxx][boxy] = False
```

- Line 97 - if either the shapes or colors of the two icons don’t match
  - pause the game for 1000 by calling pygame.time.wait(1000) so that the player has a chance to see that the two icons don’t match
  - “cover up” animation plays for both boxes
  - update the game state to mark these boxes as not revealed (covered up).

Handling If the Player Won

```
103.                    elif hasWon(revealedBoxes): # check if all pairs found
104.                        gameWonAnimation(mainBoard)
105.                        pygame.time.wait(2000)
106.
107.                        # Reset the board
108.                        mainBoard = getRandomizedBoard()
109.                        revealedBoxes = generateRevealedBoxesData(False)
110.
111.                        # Show the fully unrevealed board for a second.
112.                        drawBoard(mainBoard, revealedBoxes)
113.                        pygame.display.update()
114.                        pygame.time.wait(1000)
115.
116.                        # Replay the start game animation.
117.                        startGameAnimation(mainBoard)
118.                    firstSelection = None # reset firstSelection variable
```

Handling If the Player Won

- hasWon() function - returns True if the board is in a winning state (all of the boxes are revealed)
  - play the “game won” animation by calling gameWonAnimation()
  - pause slightly to let the player revel in the victory
  - reset the data structures in mainBoard and revealedBoxes to start a new game
  - play the “start game” animation again
- No matter if the two boxes were matching or not, after the second box was clicked line 118 will set the firstSelection variable back to None

Drawing the Game State to the Screen

```
120.         # Redraw the screen and wait a clock tick.
121.         pygame.display.update()
122.         FPSCLOCK.tick(FPS)
```

- The game state has been updated depending on the player's input, and the latest game state has been drawn to the DISPLAYSURF display Surface object.
- pygame.display.update() - draw the DISPLAYSURF Surface object to the computer screen.

Drawing the Game State to the Screen

- FPS constant to the integer value 30 (line 9)
  - The game will run (at most) at 30 frames per second.
  - increase this number - the program will run faster
  - decrease this number - the program will run slower (0.5 - half a frame per second)
- to run at 30 fps, each frame must be drawn in 1/30th of a second.
  - => pygame.display.update() and all the code in the game loop must execute in under 33.3 ms
- To prevent the program from running too fast, we call the tick() method of the pygame.Clock object in FPSCLOCK to have to it pause the program for the rest of the 33.3 ms

Creating the “Revealed Boxes” Data Structure

```
125. def generateRevealedBoxesData(val):
126.     revealedBoxes = []
127.     for i in range(BOARDWIDTH):
128.         revealedBoxes.append([val] * BOARDHEIGHT)
129.     return revealedBoxes
```

- generateRevealedBoxesData() function needs to create a list of lists of Boolean values determined by the val parameter.
- Result: revealedBoxes[x][y] structure
- The inner lists represent the vertical columns of the board and not the horizontal rows. Otherwise, the data structure will have a revealedBoxes[y][x] structure.

Creating the Board Data Structure: Step 1 – Get All Possible Icons

```
132. def getRandomizedBoard():
133.     # Get a list of every possible shape in every possible color.
134.     icons = []
135.     for color in ALLCOLORS:
136.         for shape in ALLSHAPES:
137.             icons.append( (shape, color) )
```

- The board data structure - a list of lists of tuples, each tuple has a two values: icon's shape and icon's color.
- Create a list with every possible combination of shape and color (nested for loops on lines 135 and 136 go through every possible shape (ALLSHAPES) for every possible color (ALLCOLORS)

Step 2 – Shuffling and Truncating the List of All Icons

```
139.     random.shuffle(icons) # randomize the order of the icons list
140.     numIconsUsed = int(BOARDWIDTH * BOARDHEIGHT / 2) # calculate how many
icons are needed
141.     icons = icons[:numIconsUsed] * 2 # make two of each
142.     random.shuffle(icons)
```

- There may be more possible combinations than spaces on the board!!!
- BOARDWIDTH * BOARDHEIGHT = the number of spaces on the board
- E.g. On a board with 70 spaces, 35 different icons are required (there will be two of each icon) - numIconsUsed

Step 2 – Shuffling and Truncating the List of All Icons

- Line 139 - shuffle the list of icons
- Line 141 - list slicing to grab the first numIconsUsed number of icons in the list (two of each by list replication using the * operator)
- Line 142 – shuffle the list elements again because the first half of the new list is identical to the last half

Step 3 – Placing the Icons on the Board

```
144.     # Create the board data structure, with randomly placed icons.
145.     board = []
146.     for x in range(BOARDWIDTH):
147.         column = []
148.         for y in range(BOARDHEIGHT):
149.             column.append(icons[0])
150.             del icons[0] # remove the icons as we assign them
151.         board.append(column)
152.     return board
```

- Create a list of lists data structure for the board.
- For each column on the board, we will create a list of randomly selected icons. As we add icons to the column (line 149) we will then delete them from the front of the icons list (line 150)

Splitting a List into a List of Lists

```
155. def splitIntoGroupsOf(groupSize, theList):
156.     # splits a list into a list of lists, where the inner lists have at
157.     # most groupSize number of items.
158.     result = []
159.     for i in range(0, len(theList), groupSize):
160.         result.append(theList[i:i + groupSize])
161.     return result
```

- The splitIntoGroupsOf() function (which will be called by the startGameAnimation() function) splits a list into a list of lists, where the inner lists have groupSize number of items in them.
- E.g. If the length of the list is 20 and the groupSize parameter is 8, then range(0, len(theList), groupSize) evaluates to range(0, 20, 8). This will give the i variable the values 0, 8, and 16 for the three iterations of the for loop.

Splitting a List into a List of Lists

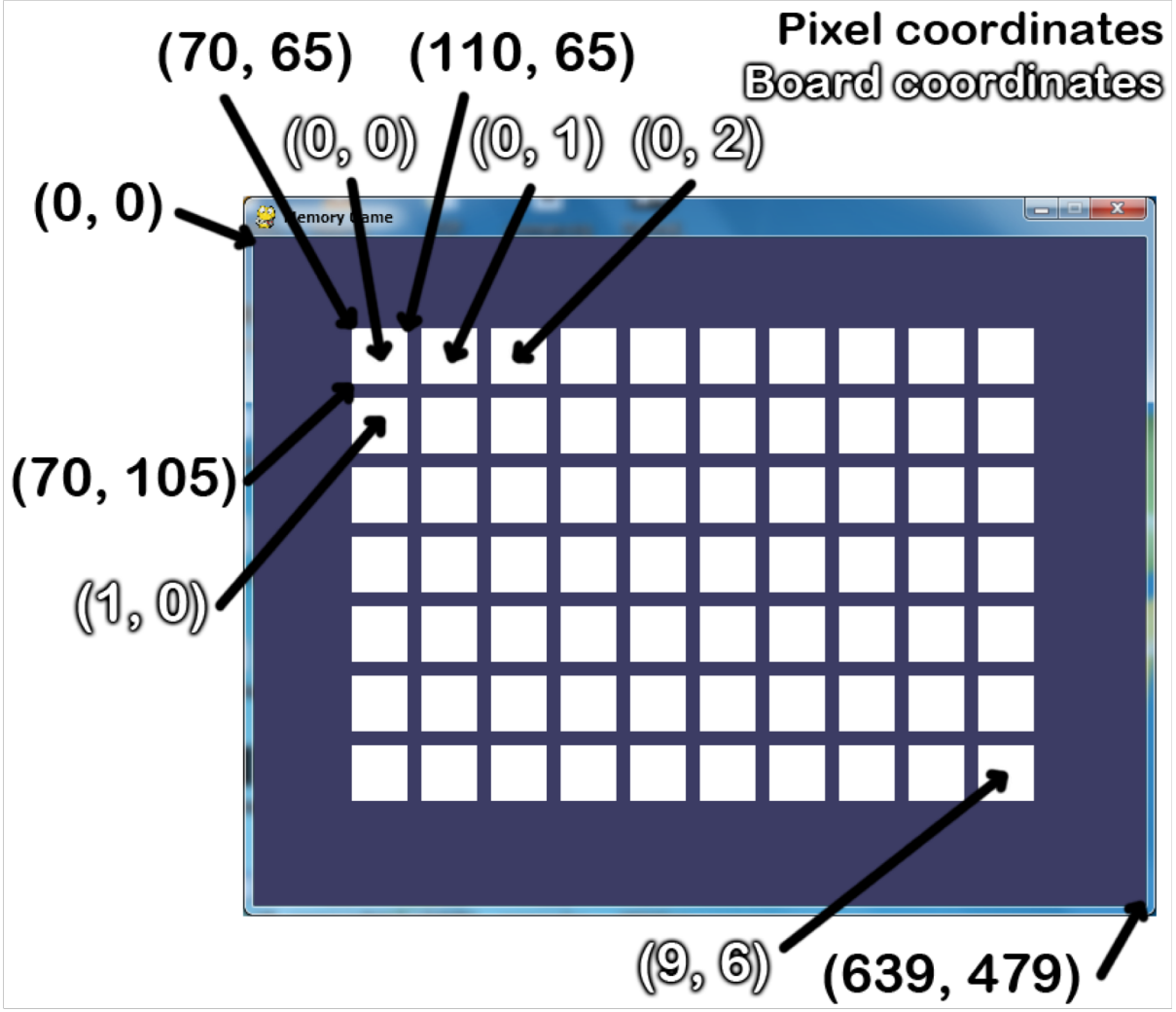
- The list slicing on line 160 with theList[i:i + groupSize] creates the lists that are added to the result list. On each iteration where i is 0, 8, and 16 (and groupSize is 8), this list slicing expression would be theList[0:8], then theList[8:16] on the second iteration, and then theList[16:24] on the third iteration.
  - The last one won't raise an IndexError error even though 24 is larger than 19, but will create a list slice with the remaining items in the list.
- Result: a list of lists.

Different Coordinate Systems

```
164. def leftTopCoordsOfBox(boxx, boxy):
165.     # Convert board coordinates to pixel coordinates
166.     left = boxx * (BOXSIZE + GAPSIZE) + XMARGIN
167.     top = boxy * (BOXSIZE + GAPSIZE) + YMARGIN
168.     return (left, top)
```

- Cartesian Coordinate system for the pixel or screen coordinates.
- Cartesian Coordinate system for the boxes.
- The leftTopCoordsOfBox() function will take box coordinates and return pixel coordinates. Because a box takes up multiple pixels on the screen, we will always return the single pixel at the top left corner of the box (two-integer tuple).

Different Coordinate Systems



Converting from Pixel Coordinates to Box Coordinates

```
171. def getBoxAtPixel(x, y):
172.     for boxx in range(BOARDWIDTH):
173.         for boxy in range(BOARDHEIGHT):
174.             left, top = leftTopCoordsOfBox(boxx, boxy)
175.             boxRect = pygame.Rect(left, top, BOXSIZE, BOXSIZE)
176.             if boxRect.collidepoint(x, y):
177.                 return (boxx, boxy)
178.     return (None, None)
```

- Convert from pixel coordinates (which the mouse clicks and mouse movement events use) to box coordinates (so we can find out over which box the mouse event happened).
- Rect objects have a collidepoint() method that you can pass X and Y coordinates too and it will return True if the coordinates are inside (that is, collide with) the Rect object's area.

Drawing the Icon, and Syntactic Sugar

```
181. def drawIcon(shape, color, boxx, boxy):
182.     quarter = int(BOXSIZE * 0.25) # syntactic sugar
183.     half =    int(BOXSIZE * 0.5)  # syntactic sugar
184.
185.     left, top = leftTopCoordsOfBox(boxx, boxy) # get pixel coords from
board coords
```

- drawIcon() function - draw an icon (with the specified shape and color) at the space whose coordinates are given in the boxx and boxy parameters.
- Each possible shape has a different set of Pygame drawing function calls for it (lines 187 to 198)
- many of the shape drawing function calls use the midpoint and quarter-point of the box
- Syntactic sugar

Drawing the Icon, and Syntactic Sugar

```
186.     # Draw the shapes
187.     if shape == DONUT:
188.         pygame.draw.circle(DISPLAYSURF, color, (left + half, top + half),
half - 5)
189.         pygame.draw.circle(DISPLAYSURF, BGCOLOR, (left + half, top +
half), quarter - 5)
190.     elif shape == SQUARE:
191.         pygame.draw.rect(DISPLAYSURF, color, (left + quarter, top +
quarter, BOXSIZE - half, BOXSIZE - half))
192.     elif shape == DIAMOND:
193.         pygame.draw.polygon(DISPLAYSURF, color, ((left + half, top), (left
+ BOXSIZE - 1, top + half), (left + half, top + BOXSIZE - 1), (left, top +
half)))
194.     elif shape == LINES:
195.         for i in range(0, BOXSIZE, 4):
196.             pygame.draw.line(DISPLAYSURF, color, (left, top + i), (left +
i, top))
197.             pygame.draw.line(DISPLAYSURF, color, (left + i, top + BOXSIZE
- 1), (left + BOXSIZE - 1, top + i))
198.     elif shape == OVAL:
199.         pygame.draw.ellipse(DISPLAYSURF, color, (left, top + quarter,
BOXSIZE, half))
```

Syntactic Sugar with Getting a Board Space's Icon's Shape and Color

```
202. def getShapeAndColor(board, boxx, boxy):
203.     # shape value for x, y spot is stored in board[x][y][0]
204.     # color value for x, y spot is stored in board[x][y][1]
205.     return board[boxx][boxy][0], board[boxx][boxy][1]
```

- A function with only one line - it improves the readability of the code

Drawing the Box Cover

```
208. def drawBoxCovers(board, boxes, coverage):
209.     # Draws boxes being covered/revealed. "boxes" is a list
210.     # of two-item lists, which have the x & y spot of the box.
211.     for box in boxes:
212.         left, top = leftTopCoordsOfBox(box[0], box[1])
213.         pygame.draw.rect(DISPLAYSURF, BGCOLOR, (left, top, BOXSIZE,
BOXSIZE))
214.         shape, color = getShapeAndColor(board, box[0], box[1])
215.         drawIcon(shape, color, box[0], box[1])
216.         if coverage > 0: # only draw the cover if there is an coverage
217.             pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top, coverage,
BOXSIZE))
218.     pygame.display.update()
219.     FPSCLOCK.tick(FPS)
```

- When the coverage parameter is 0, there is no coverage at all. When the coverage is set to 20, there is a 20 pixel wide white box covering the icon. The largest size we'll want the coverage set to is the number in BOXSIZE, where the entire icon is completely covered.

Handling the Revealing and Covering Animation

```
222. def revealBoxesAnimation(board, boxesToReveal):
223.     # Do the "box reveal" animation.
224.     for coverage in range(BOXSIZE, (-REVEALSPEED) - 1, - REVEALSPEED):
225.         drawBoxCovers(board, boxesToReveal, coverage)
226.
227.
228. def coverBoxesAnimation(board, boxesToCover):
229.     # Do the "box cover" animation.
230.     for coverage in range(0, BOXSIZE + REVEALSPEED, REVEALSPEED):
231.         drawBoxCovers(board, boxesToCover, coverage)
```

- revealBoxesAnimation() and coverBoxesAnimation() only need to draw an icon with a varying amount of coverage by the white box.

Drawing the Entire Board

```
234. def drawBoard(board, revealed):
235.     # Draws all of the boxes in their covered or revealed state.
236.     for boxx in range(BOARDWIDTH):
237.         for boxy in range(BOARDHEIGHT):
238.             left, top = leftTopCoordsOfBox(boxx, boxy)
239.             if not revealed[boxx][boxy]:
240.                 # Draw a covered box.
241.                 pygame.draw.rect(DISPLAYSURF, BOXCOLOR, (left, top,
BOXSIZE, BOXSIZE))
242.             else:
243.                 # Draw the (revealed) icon.
244.                 shape, color = getShapeAndColor(board, boxx, boxy)
245.                 drawIcon(shape, color, boxx, boxy)
```

- The drawBoard() function makes a call to drawIcon() for each of the boxes on the board. The nested for loops on lines 236 and 237 will loop through every possible X and Y coordinate for the boxes, and will either draw the icon at that location or draw a white square instead (to represent a covered up box).

Drawing the Highlight

```
248. def drawHighlightBox(boxx, boxy):
249.     left, top = leftTopCoordsOfBox(boxx, boxy)
250.     pygame.draw.rect(DISPLAYSURF, HIGHLIGHTCOLOR, (left - 5, top - 5,
BOXSIZE + 10, BOXSIZE + 10), 4)
```

- To help the player recognize that he can click on a covered box to reveal it, we will make a blue outline appear around a box to highlight it. This outline is drawn with a call to pygame.draw.rect() to make a rectangle with a width of 4 pixels.

The “Start Game” Animation

```
253. def startGameAnimation(board):
254.     # Randomly reveal the boxes 8 at a time.
255.     coveredBoxes = generateRevealedBoxesData(False)
256.     boxes = []
257.     for x in range(BOARDWIDTH):
258.         for y in range(BOARDHEIGHT):
259.             boxes.append( (x, y) )
260.     random.shuffle(boxes)
261.     boxGroups = splitIntoGroupsOf(8, boxes)
```

- The animation that plays at the beginning of the game gives the player a quick hint as to where all the icons are located. In order to make this animation, we have to reveal and cover up groups of boxes one group after another.
- To change up the boxes each time a game starts, we will call the random.shuffle() function to randomly shuffle the order of the tuples in the boxes list. Then when we reveal and cover up the first 8 boxes in this list (and each group of 8 boxes afterwards), it will be random group of 8 boxes.

Revealing and Covering the Groups of Boxes

```
263.     drawBoard(board, coveredBoxes)
264.     for boxGroup in boxGroups:
265.         revealBoxesAnimation(board, boxGroup)
266.         coverBoxesAnimation(board, boxGroup)
```

- drawBoard() will end up drawing only covered up white boxes
- The for loop will go through each of the inner lists in the boxGroups lists. We pass these to revealBoxesAnimation(), which will perform the animation of the white boxes being pulled away to reveal the icon underneath. Then the call to coverBoxesAnimation() will animate the white boxes expanding to cover up the icons. Then the for loop goes to the next iteration to animate the next set of 8 boxes.

The “Game Won” Animation

```
269. def gameWonAnimation(board):
270.     # flash the background color when the player has won
271.     coveredBoxes = generateRevealedBoxesData(True)
272.     color1 = LIGHTBGCOLOR
273.     color2 = BGCOLOR
274.
275.     for i in range(13):
276.         color1, color2 = color2, color1 # swap colors
277.         DISPLAYSURF.fill(color1)
278.         drawBoard(board, coveredBoxes)
279.         pygame.display.update()
280.         pygame.time.wait(300)
```

- When the player has uncovered all of the boxes by matching every pair on the board, we want to congratulate them by flashing the background color. The for loop will draw the color in the color1 variable for the background color and then draw the board over it. However, on each iteration of the for loop, the values in color1 and color2 will be swapped with each other on line 276. This way the program will alternate between drawing two different background colors.

Telling if the Player Has Won

```
283. def hasWon(revealedBoxes):
284.     # Returns True if all the boxes have been revealed, otherwise False
285.     for i in revealedBoxes:
286.         if False in i:
287.             return False # return False if any boxes are covered.
288.     return True
```

- The player has won the game when all of the icon pairs have been matched. Since the “revealed” data structure gets values in it set to True as icons have been matched, we can simply loop through every space in revealedBoxes looking for a False value.

Why Bother Having a main() Function?

```
291. if __name__ == '__main__':
292.     main()
```

- Lets you have local variables whereas otherwise the local variables in the main() function would have to become global variables. Limiting the number of global variables is a good way to keep the code simple and easier to debug.

Why Bother Having a main() Function?

- This also lets you import the program so that you can call and test individual functions.

```
>>> import memorypuzzle
>>> memorypuzzle.splitIntoGroupsOf(3, [0,1,2,3,4,5,6,7,8,9])
[[0, 1, 2], [3, 4, 5], [6, 7, 8], [9]]
>>> memorypuzzle.getBoxAtPixel(0, 0)
(None, None)
>>> memorypuzzle.getBoxAtPixel(150, 150)
(1, 1)
```

- When a module is imported, all of the code in it is run. If we didn't have the main() function, and had its code in the global scope, then the game would have automatically started as soon as we imported it, which really wouldn't let us call individual functions in it.