COSC411 HW #1 Report

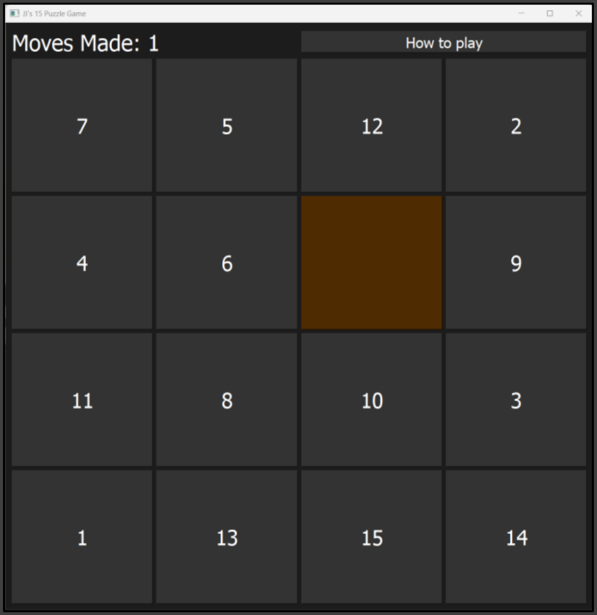
By JJ McCauley

Program Description

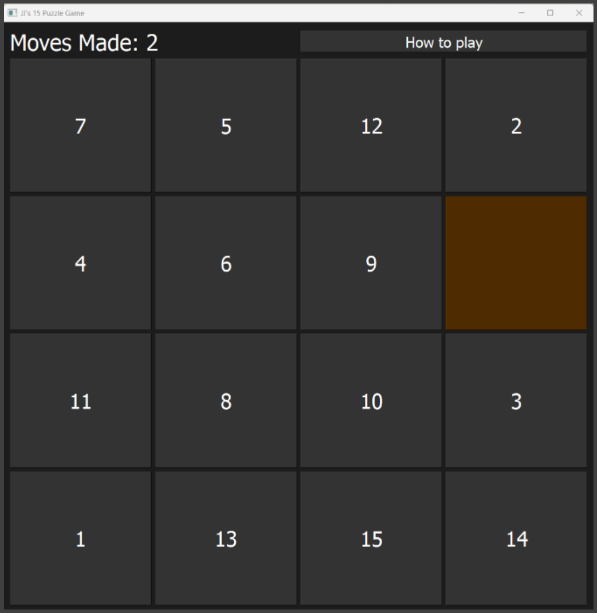
This program utilizes the PYQT5 library to implement a graphical interface for the popular 15-sliding puzzle game. This is primarily done through a 4x4 button grid, which swaps upon click. Once the board reads in the correct order (1-15, with a blank at the bottom right space), a winner is declared. The move count is visibly tracked and displayed at the end. Additional features include a "How to Play" button, dynamic button size, highlighted empty boxes, and the option to play again.

Demo

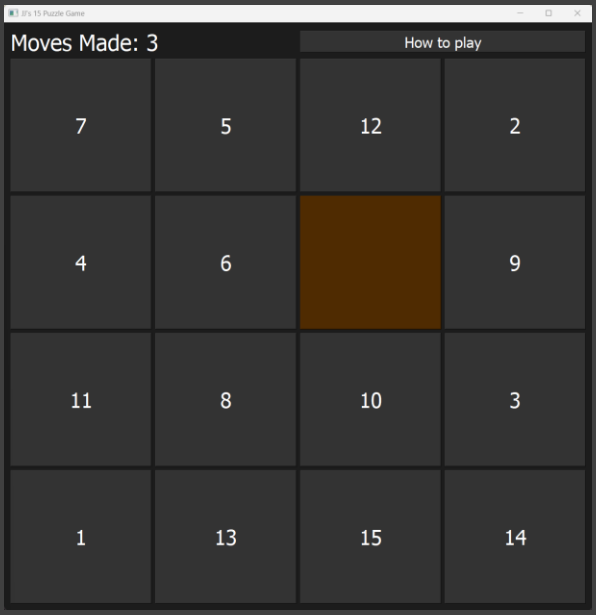
a) Initial Configuration



b) Move one cell left (9)



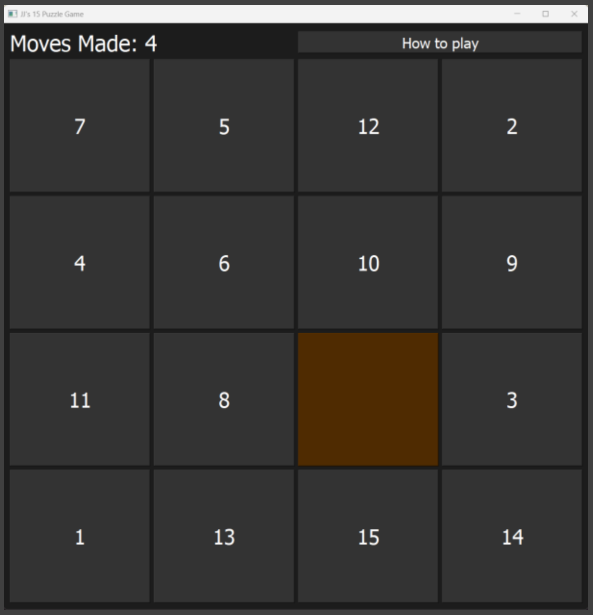
c) Move one cell right (9)



Moves Made: 3

7	5	12	2
4	6		9
11	8	10	3
1	13	15	14

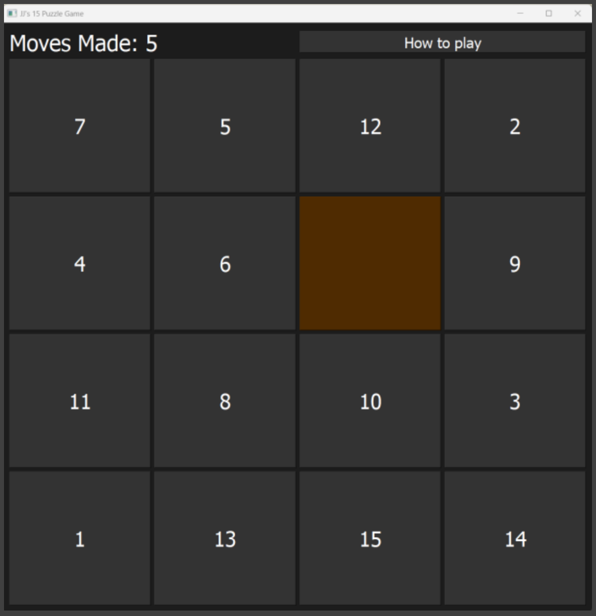
d) Move one cell up (10)



Moves Made: 4

7	5	12	2
4	6	10	9
11	8		3
1	13	15	14

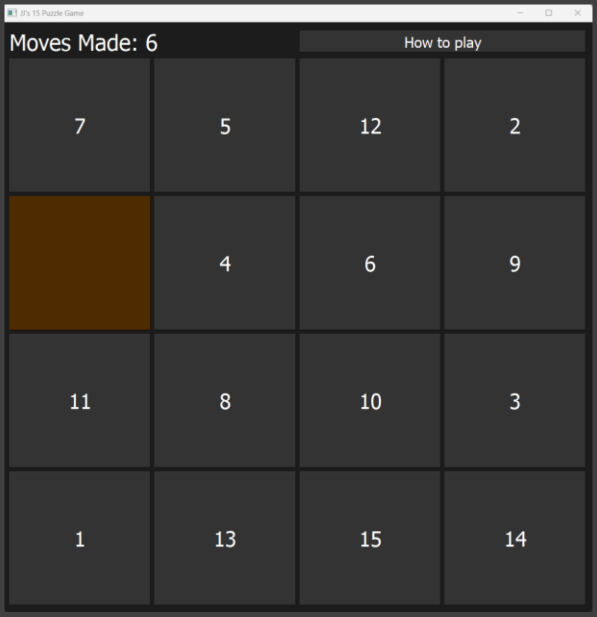
e) Move on cell down (10)



Moves Made: 5

7	5	12	2
4	6		9
11	8	10	3
1	13	15	14

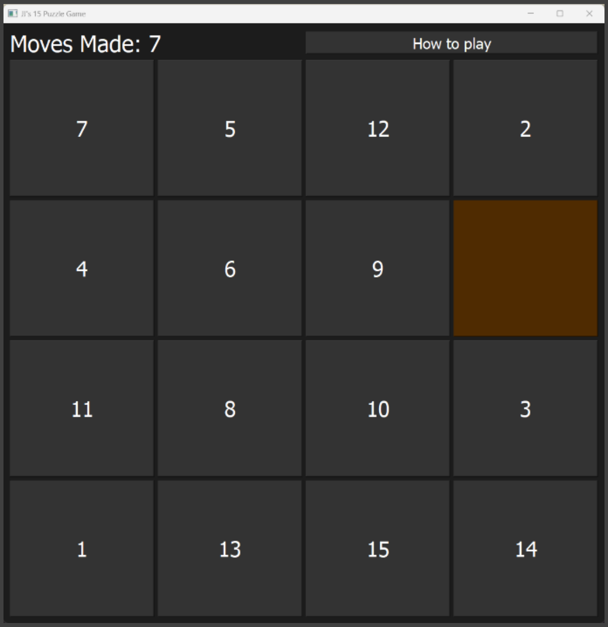
f) Move 2 cells right (4, 6)



Moves Made: 6

7	5	12	2
	4	6	9
11	8	10	3
1	13	15	14

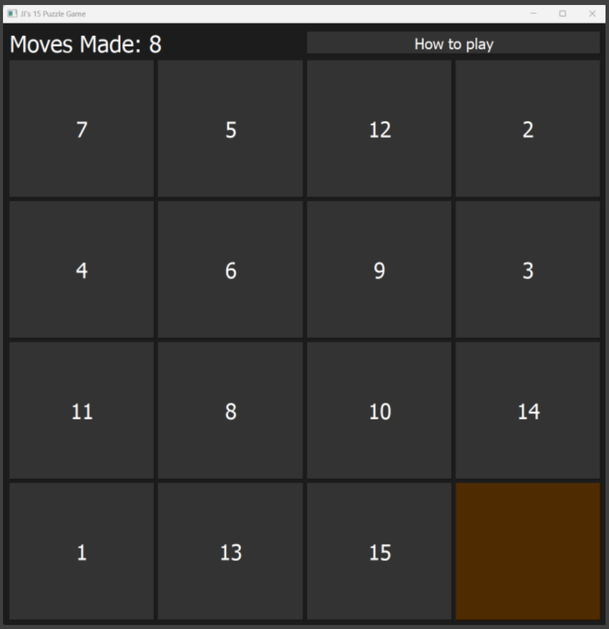
g) Move 3 cells left (4, 6, 9)


JJ's 15 Puzzle Game

Moves Made: 7

7	5	12	2
4	6	9	
11	8	10	3
1	13	15	14

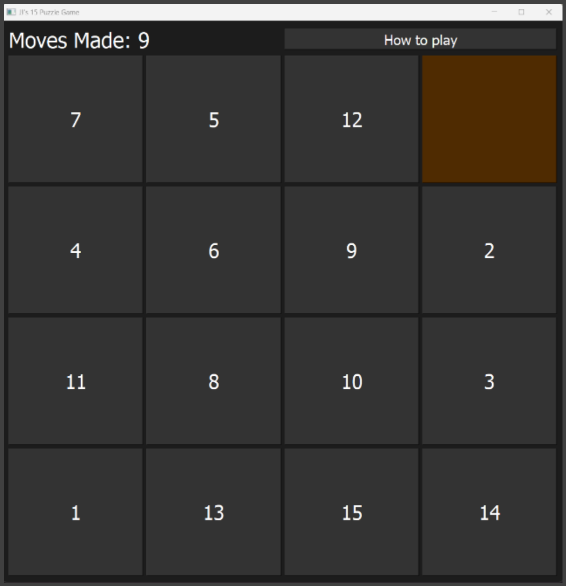
h) Move 2 cells up (3, 14)


JJ's 15 Puzzle Game

Moves Made: 8

7	5	12	2
4	6	9	3
11	8	10	14
1	13	15	

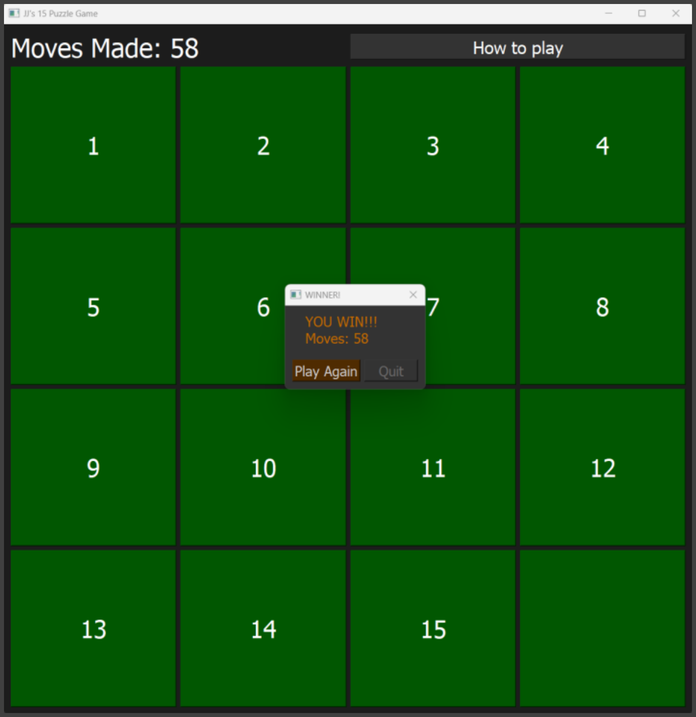
i) Move 3 cells down (2, 3, 14)


JJ's 15 Puzzle Game

Moves Made: 9

7	5	12	
4	6	9	2
11	8	10	3
1	13	15	14

j) Final Configuration (*Uses hardcoding test case)


JJ's 15 Puzzle Game

Moves Made: 58

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

WINNER!

YOU WIN!!!
Moves: 58

Python Source Code

```python
''' 15-puzzle (sliding puzzle game) - JJ McCauley - Last Update 9/24/24 '''

from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
from PyQt5.QtCore import Qt
from functools import partial  # Passing extra parameters through clicked buttons
import sys
import random as rand  # Shuffling board

# Global Constants
BOARD_SIZE = 4
# Global Variables
buttons = [[None for _ in range(BOARD_SIZE)] for _ in range(BOARD_SIZE)] # Empty
2d list that will hold the buttons
b_pos = (-1, 1)  # Current space of the blank, removes need for finding during
every successful swap
move_count = 0  # Move counter to be displayed
grid = QGridLayout()  # Grid which will hold the buttons

'''Determine whether board is solvable
Parameters: The board list
Returns: Whether the board is not solvable (True)'''
def not_solvable(b):
    empty_index = b.index(0)

    #Calculating inversions
    inversions = 0
    for i in range(0, BOARD_SIZE):
        for j in range(0, i):
            if b[j] > b[i]:
                inversions += 1

    # Returning Solvability
    if empty_index % 2 == 1 and inversions % 2 == 0:
        return False  # board is solvable
    elif empty_index % 2 == 0 and inversions % 2 == 1:
        return False  # board is solvable
    else:
        return True  # board is not solvable


'''Helper function that checks if the board is solved
Returns: Whether the board was solved (True if solved, False if not)'''
```

```python
def is_solved(board):
    return board == [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15,
0]]


'''Helper function that returns if the provided index is 4-way adjacent to the
blank
Parameters: i, j
Returns: Wether the given index is adjacent to blank'''
def is_directly_adjacent(i, j, b_pos):
    if i == b_pos[0] and abs(j - b_pos[1]) == 1:  # In the same row and 1 tile
apart
        return True
    elif j == b_pos[1] and abs(i - b_pos[0]) == 1:  # In the same column and 1
tile apart
        return True
    else:
        return False

'''Helper function that returns if the provided index is in the same row or col
as adjacent blank
Parameters: i, j, b_pos (blank position)
Returns: Wether the given index is in the same row or col to blank'''
def in_adjacent_row_col(i, j, b_pos):
    if i == b_pos[0] and j == b_pos[1]:  # Base case, clicking on the blank
shouldn't do anything
        return False
    elif i == b_pos[0] or j == b_pos[1]:  # In the same row or col
        return True
    else:
        return False


'''Makes the board, ensuring that it is solvable
Parameters: None
Returns: The solvable board as a 2-d list'''
def make_board():
    board = [num for num in range(0, 16)]  # 16 spots, 0-15
    rand.shuffle(board)  # Randomly generate spots
    while not_solvable(board):
        rand.shuffle(board)

    cols = 4
    n_board = [board[i:i+cols] for i in range(0, len(board), cols)]  # Convert 1d
to 2d
```

```
    return n_board


''' GUI interface (PYQT5) & Intra-game Logic '''
def main():
    global b_pos, move_count, grid  # Declare b_pos as global for correct
referencing

    # Getting the solvable board
    #board = make_board()
    board = [[1, 0, 2, 3], [5, 6, 7, 4], [9, 10, 11, 8], [13, 14, 15, 12]]  #
Hardcoded test case

    # Creating grid layouts
    top_grid = QGridLayout()  # Top grid for displaying moves made and
instructions button
    top_grid.setRowStretch(0, 1)  # Allow for row to stretch, improving GUI

    # Creating master layout to store on root window
    root_layout = QVBoxLayout()
    root_layout.addLayout(top_grid, 2)
    root_layout.addLayout(grid, 1)

    # Configuring root window
    app = QApplication([])
    window = QWidget()
    icon = QIcon("15-puzzle-game-image.png")  # Custom icon for window
    window.setWindowIcon(icon)
    window.setWindowTitle("JJ's 15 Puzzle Game")
    window.setGeometry(100, 100, 1200, 1200)
    window.setStyleSheet("background-color: #1c1c1c;")

    # Add label displaying move count
    moves_label = QLabel()
    moves_label.setText("Welcome!")
    moves_label.setStyleSheet("color: white; font-size: 45px; font-family:
Tahoma;")
    top_grid.addWidget(moves_label, 0, 0)

    # Adding a button that displays a pop-up with instructions
    instructions_button = QPushButton("How to play")
    instructions_button.setStyleSheet("background-color: #333333; color: white;
font-size: 30px; font-family: Tahoma;")
    instructions_button.clicked.connect(on_help_button_click)
    top_grid.addWidget(instructions_button, 0, 1)

    # Adding numbers to grid
    for i in range(0, BOARD_SIZE):
        for j in range(0, BOARD_SIZE):
            if board[i][j] == 0:
                button = QPushButton(' ')
                button.setStyleSheet("background-color: #4f2b01; color: black;
font-size: 42px;")  # Set blank button formatting
                b_pos = (i, j)
            else:
                button = (QPushButton(f'{board[i][j]}'))
                button.setStyleSheet("background-color: #333333; color: white;
font-size: 42px;")  # Set Number buttons formatting

            buttons[i][j] = button
            button.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Expanding)  #
Allow for buttons to expand
            button.clicked.connect(partial(on_num_button_click, i, j, board,
moves_label))  # Pass positional paramters to function
            grid.addWidget(button, i, j)

    # Show the window
    window.setLayout(root_layout)
    window.show()
    sys.exit(app.exec_())

'''Number Button Click Method - Checks if the current button is adjacent to the
blank button.
If it is, swaps values in board and updates GUI
Parameters: i (row of button clicked), j (column of button clicked), board, b_pos
(blank button position)'''
def on_num_button_click(i, j, board, moves_label):
    global b_pos, move_count
    if in_adjacent_row_col(i, j, b_pos):
        # Get the current button & it's number
        clicked_button_number = board[i][j]

        # Swap the number values on the grid
        if is_directly_adjacent(i, j, b_pos):  # Directly adjacent
            buttons[b_pos[0]][b_pos[1]].setText(str(clicked_button_number))  #
Move number to blank
            buttons[b_pos[0]][b_pos[1]].setStyleSheet("background-color: #333333;
color: white; font-size: 42px;")  # Give new num button formatting
            buttons[i][j].setText(' ')  # Move blank to last clicked tile
            buttons[i][j].setStyleSheet("background-color: #4f2b01; color: black;
font-size: 42px;")  # Give new button blank formatting
            # Swap the number values on the board
            board[i][j] = 0
            board[b_pos[0]][b_pos[1]] = int(clicked_button_number)
            b_pos = (i, j)
        else:  # Adjacent by row or col, must slide peices over
            shift_board(i, j, board)  # Call helper function
            update_buttons(board)

        # Update the move count
        move_count += 1
        moves_label.setText(f"Moves Made: {move_count}")

        # Check for win
        if is_solved(board):
            p_again = is_winner(moves_label)

'''Shift Board - Shifts the board at the given index
 Parameters: i (row of click), j (col of click), b_pos (tuple of blankl
position), board'''
def shift_board(i, j, board):
    global b_pos
    if i == b_pos[0]:  # Shift row
        if j < b_pos[1]:  # Shift right
            for num in range(b_pos[1], j, -1):
                board[i][num] = board[i][num-1]
            board[i][j] = 0

        else:  # Shift left
            for num in range(b_pos[1], j):
                board[i][num] = board[i][num+1]
            board[i][j] = 0

    else:  # Shift col
        if i > b_pos[0]:  # Shift up
            for num in range(b_pos[0], i):
                board[num][j] = board[num+1][j]
            board[i][j] = 0

        else:  # Shift down
            for num in range(b_pos[0], i, -1):
                board[num][j] = board[num-1][j]
            board[i][j] = 0
```

```python
    b_pos = (i, j)  # Update blank space


'''Update Buttons - Syncs the buttons with the board
Parameters: board 2d list'''
def update_buttons(board):
    print(board)
    for i in range(BOARD_SIZE):
        for j in range(BOARD_SIZE):
            if board[i][j] == 0:
                buttons[i][j].setText(' ')  # Move blank to last clicked tile
                buttons[i][j].setStyleSheet("background-color: #4f2b01; color:
black; font-size: 42px;")  # Give new button blank formatting
            else:
                buttons[i][j].setText(str(board[i][j]))
                buttons[i][j].setStyleSheet("background-color: #333333; color:
white; font-size: 42px;")  # Give num button formatting


'''Help Button Click Method - Displays directions for the game'''
def on_help_button_click():
    # Help instructions string
    m = " ---------------- Instructions ---------------- \n \
        Goal: Slide the puzzle pieces together to order the numbers from 1-15,
with the bottom right tile being an empty space. \n \
        How to play: Start by clicking o an adjacent number to swap with the
empty space. Once the game is complete and you won, a message will be displayed.
\n \
        Move Counter: This will display how many moves you have made in the
current game. An optional goal is to minimize these moves."
    # Making a message box and displaying it
    message = QMessageBox()
    message.setStyleSheet("background-color: #333333; color: white; font-family:
Tahoma; font-size: 24px;")
    message.setText(m)
    message.setWindowTitle("15-Game Instructions")
    message.show()
    #message.resize(800, 600)
    message.exec_()


'''Win Dialouge Box - Shows win message and asks the user to play again
Parameters: moves_label (will be changed in nextgame() function if user decides
to play again)
Returns: Whether to player would like to play again'''
```

```
def is_winner(moves_label):
    # Set all Cells to green, indicating winner
    for b_list in buttons:
        for b in b_list:
            b.setStyleSheet("background-color: #015701; color: white; font-
family: Tahoma; font-size: 42px;")

    # Win Dialouge Box
    m = f"YOU WIN!!!\nMoves: {move_count}"

    # Making a message box and displaying it
    win_message = QMessageBox()
    win_message.setStyleSheet("background-color: #333333; color: #c46a00; font-
family: Tahoma; font-size: 24px;")
    win_message.setText(m)
    win_message.setWindowTitle("WINNER!")

    # Make "Play Again" and "Quit" Buttons
    play_again_button = QPushButton("Play Again")
    play_again_button.setStyleSheet("background-color: #4f2b01; color: #cfcccc;
font-family: Tahoma; font-size: 24px;")
    quit_button = QPushButton("Quit")
    quit_button.setStyleSheet("background-color: #333333; color: #6b6a6a; font-
family: Tahoma; font-size: 24px;")

    # Add buttons to message box
    win_message.addButton(play_again_button, QMessageBox.YesRole)
    win_message.addButton(quit_button, QMessageBox.NoRole)

    # Wait for input from the user
    response = win_message.exec()

    # Assign roles for each button (responses will be passed into next_game
helper function)
    if win_message.clickedButton() == play_again_button:
        next_game(moves_label, True)
    elif win_message.clickedButton() == quit_button:
        next_game(moves_label, False)

    # Display the window
    win_message.show()
    win_message.resize(800, 600)


'''Next Game Function - Begins new game or exits program, based on user's input
```

```
Parameters: moves_label (updates counter), p_again (whether user would like to
play again)'''
def next_game(moves_label, p_again):
    global grid, b_pos
    if p_again:  # Set up the board and reset counter for next game
        move_count = 0
        moves_label.setText(str(move_count))
        board = make_board()  # Generate new, solvable board
        # Adding numbers to grid (unsure if I need to re-declare properties)
        for i in range(0, BOARD_SIZE):
            for j in range(0, BOARD_SIZE):
                if board[i][j] == 0:
                    button = QPushButton(' ')
                    button.setStyleSheet("background-color: #4f2b01; color:
black; font-size: 42px;")
                    b_pos = (i, j)
                else:
                    button = (QPushButton(f'{board[i][j]}'))
                    button.setStyleSheet("background-color: #333333; color:
white; font-size: 42px;")

                buttons[i][j] = button
                button.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding)
                button.clicked.connect(partial(on_num_button_click, i, j, board,
moves_label))
                grid.addWidget(button, i, j)
    else:
        QApplication.quit()  # End event loop and close application


# Good practice
if __name__ == '__main__':
    main()
```

Source References

Working with button & button grid - https://www.youtube.com/watch?v=D4J1fAFZu8s

Styling buttons (CSS-like styling sheets): https://doc.qt.io/qtforpython-6/overviews/stylesheet-examples.html

Partial Library: Cannot find link but I believe it was Stack Overflow

Expanding Buttons: https://python-forum.io/thread-19742.html

Grid layouts: https://python-forum.io/thread-19742.html

Discussed with/ asked help from

N/A (None)