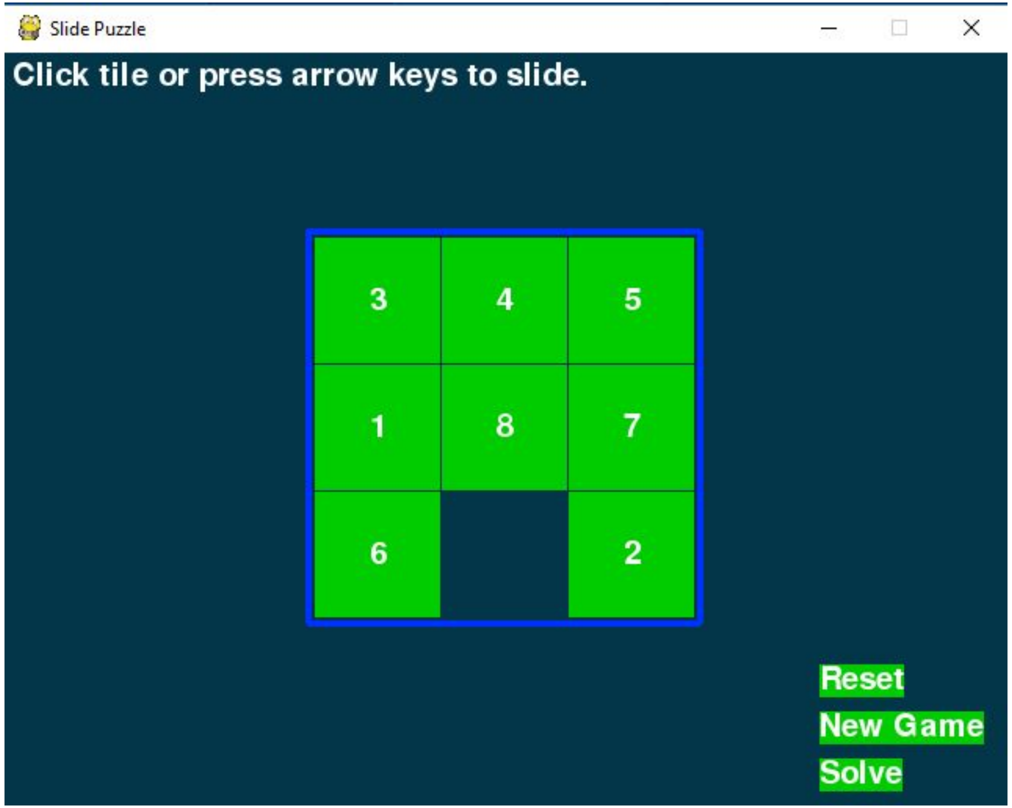
Sliding Puzzle

Making Game with Python
3/31/2019

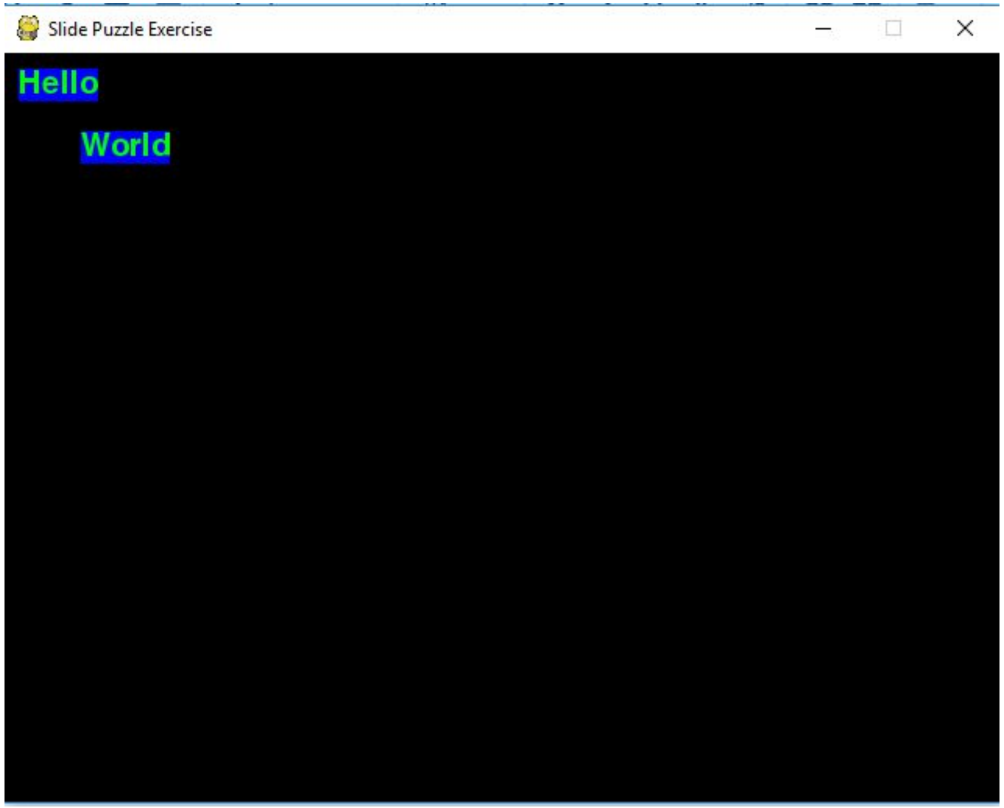

Slide Puzzle

Click tile or press arrow keys to slide.

Features of the Game

- Quit: ‘X’ sign, ESC Key
- Words: Reset, New Game, Solve, Message
- Board:
  - Mouse
  - Arrow Keys: left, right, up, down
  - Letter Keys: A W S D

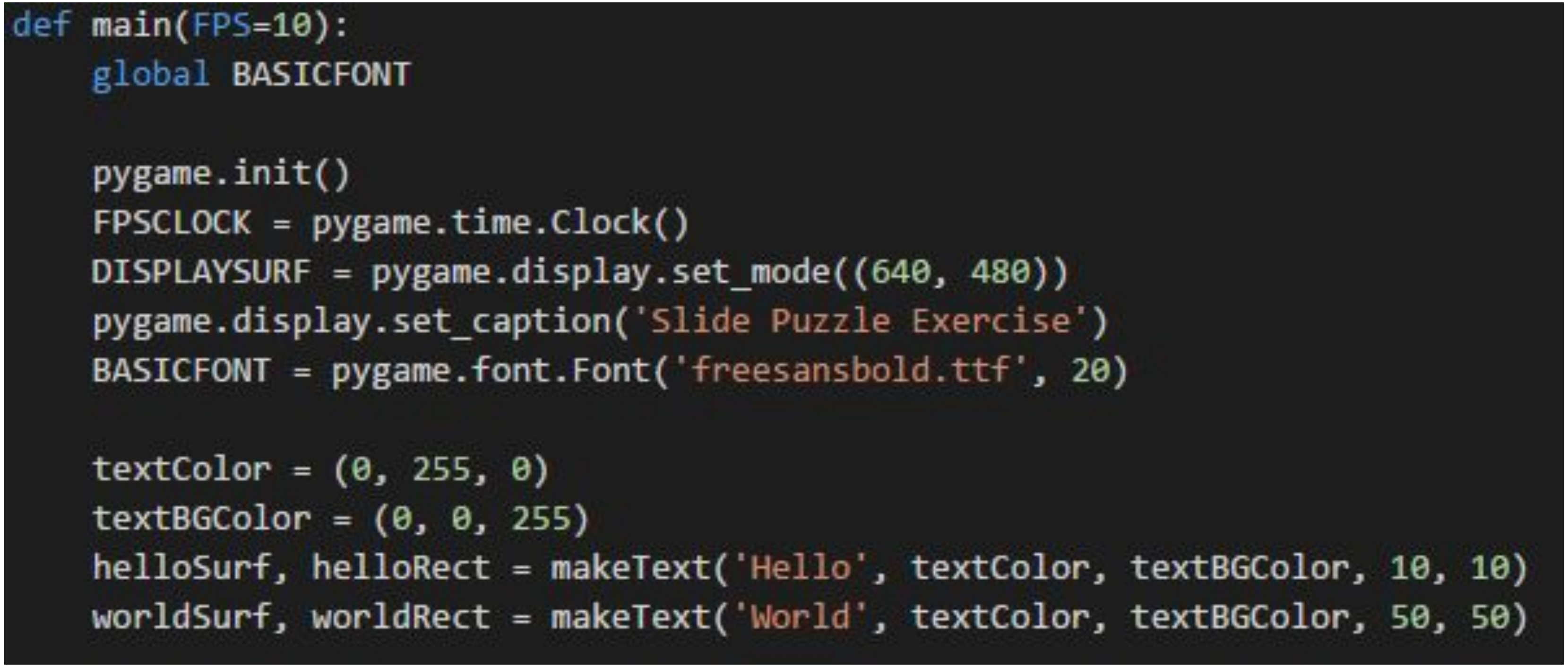
Project 1: Word Game



Hello

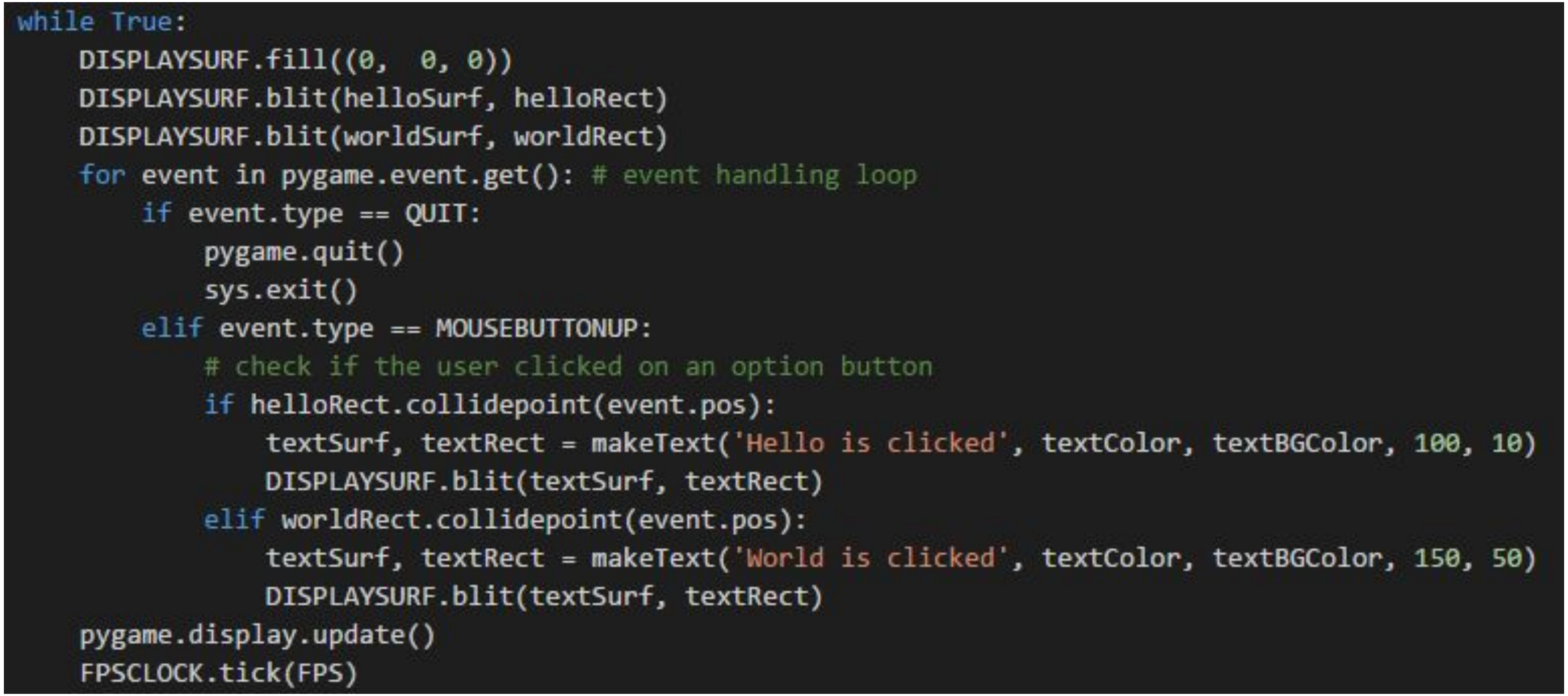
World

Main function (1)



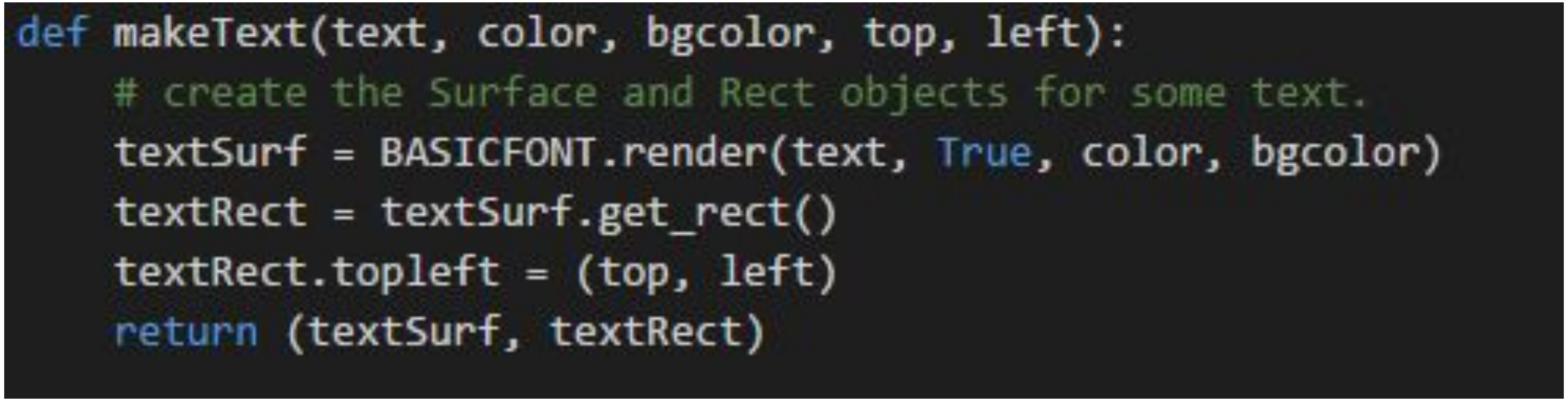
```
def main(FPS=10):
    global BASICFONT

    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((640, 480))
    pygame.display.set_caption('Slide Puzzle Exercise')
    BASICFONT = pygame.font.Font('freesansbold.ttf', 20)

    textColor = (0, 255, 0)
    textBGColor = (0, 0, 255)
    helloSurf, helloRect = makeText('Hello', textColor, textBGColor, 10, 10)
    worldSurf, worldRect = makeText('World', textColor, textBGColor, 50, 50)
```

Main function (2)



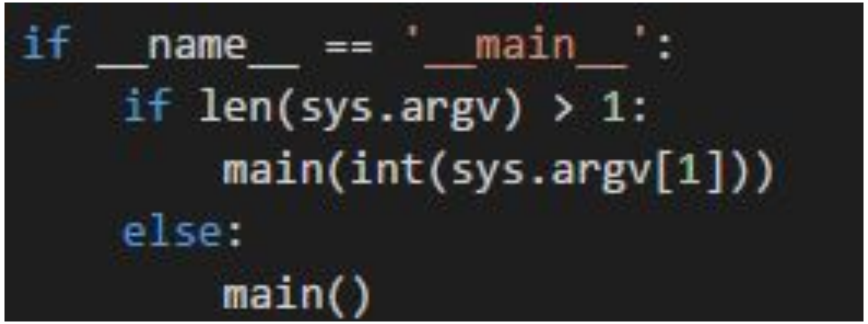
```python
while True:
    DISPLAYSURF.fill((0,  0, 0))
    DISPLAYSURF.blit(helloSurf, helloRect)
    DISPLAYSURF.blit(worldSurf, worldRect)
    for event in pygame.event.get(): # event handling loop
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == MOUSEBUTTONUP:
            # check if the user clicked on an option button
            if helloRect.collidepoint(event.pos):
                textSurf, textRect = makeText('Hello is clicked', textColor, textBGColor, 100, 10)
                DISPLAYSURF.blit(textSurf, textRect)
            elif worldRect.collidepoint(event.pos):
                textSurf, textRect = makeText('World is clicked', textColor, textBGColor, 150, 50)
                DISPLAYSURF.blit(textSurf, textRect)
    pygame.display.update()
    FPSCLOCK.tick(FPS)
```

makeText function



```
def makeText(text, color, bgcolor, top, left):
    # create the Surface and Rect objects for some text.
    textSurf = BASICFONT.render(text, True, color, bgcolor)
    textRect = textSurf.get_rect()
    textRect.topleft = (top, left)
    return (textSurf, textRect)
```

Entry Point



```
if __name__ == '__main__':
    if len(sys.argv) > 1:
        main(int(sys.argv[1]))
    else:
        main()
```

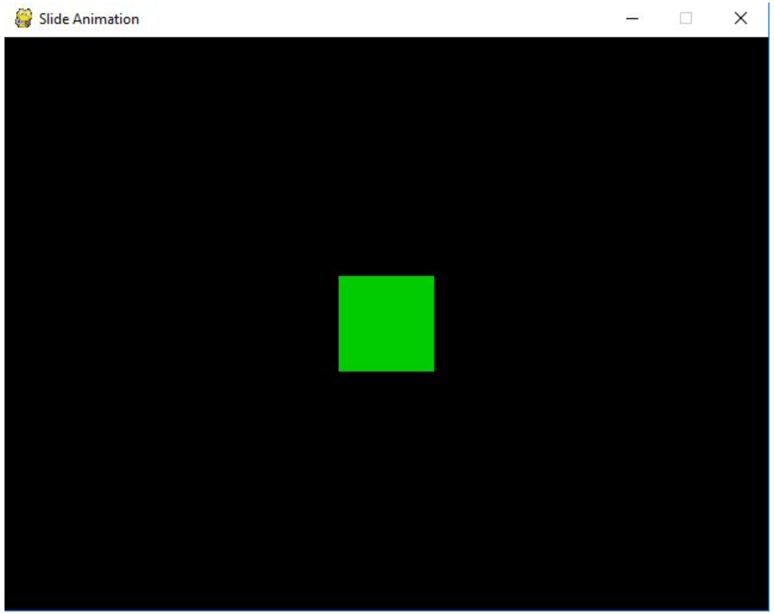
Windows: py slidepuzzle_exercise.py 10

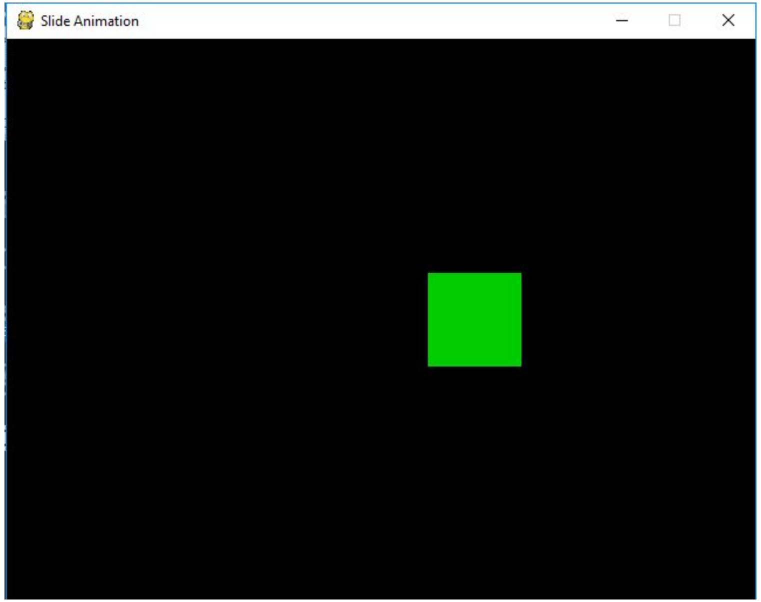
Mac: python3 slidepuzzle_exercise.py 10

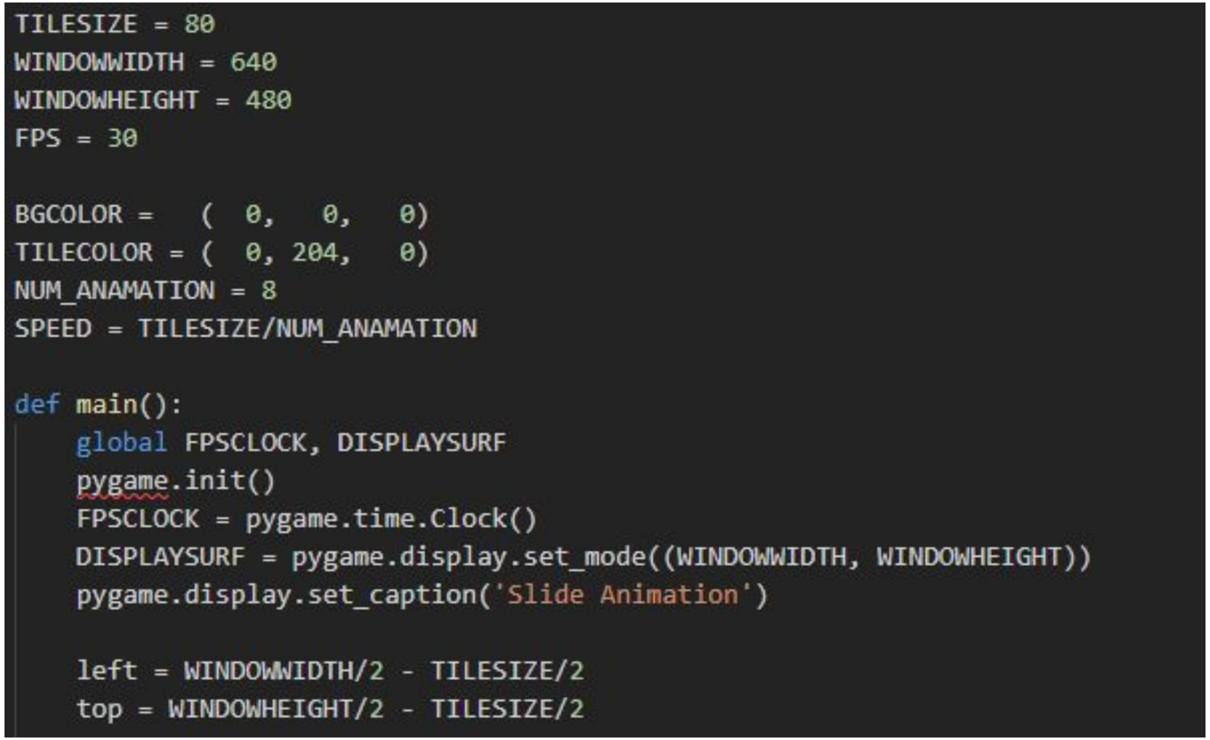
Python 3.7 IDE

```
Import os

os.chdir(your_working_directory)

os.popen(‘py slidepuzzle_exercise.py 10’).read()
```

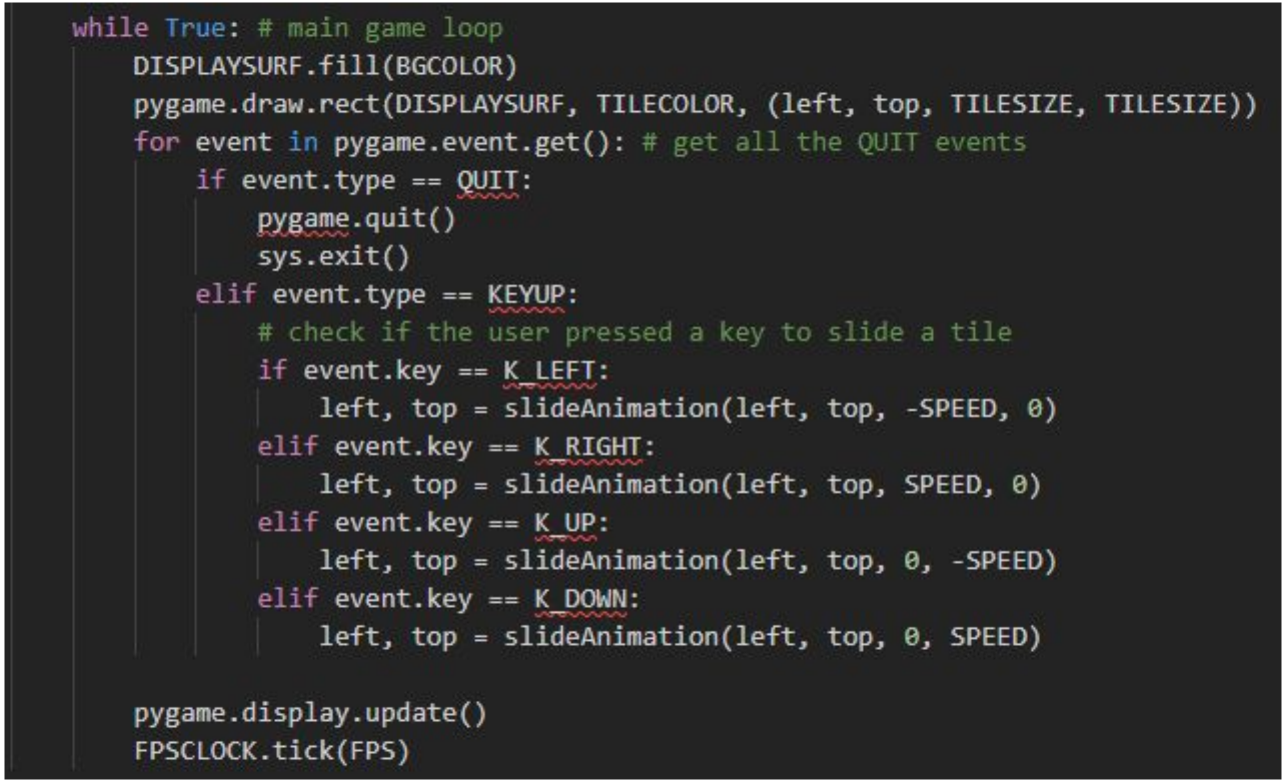
Project 2: Slide Animation



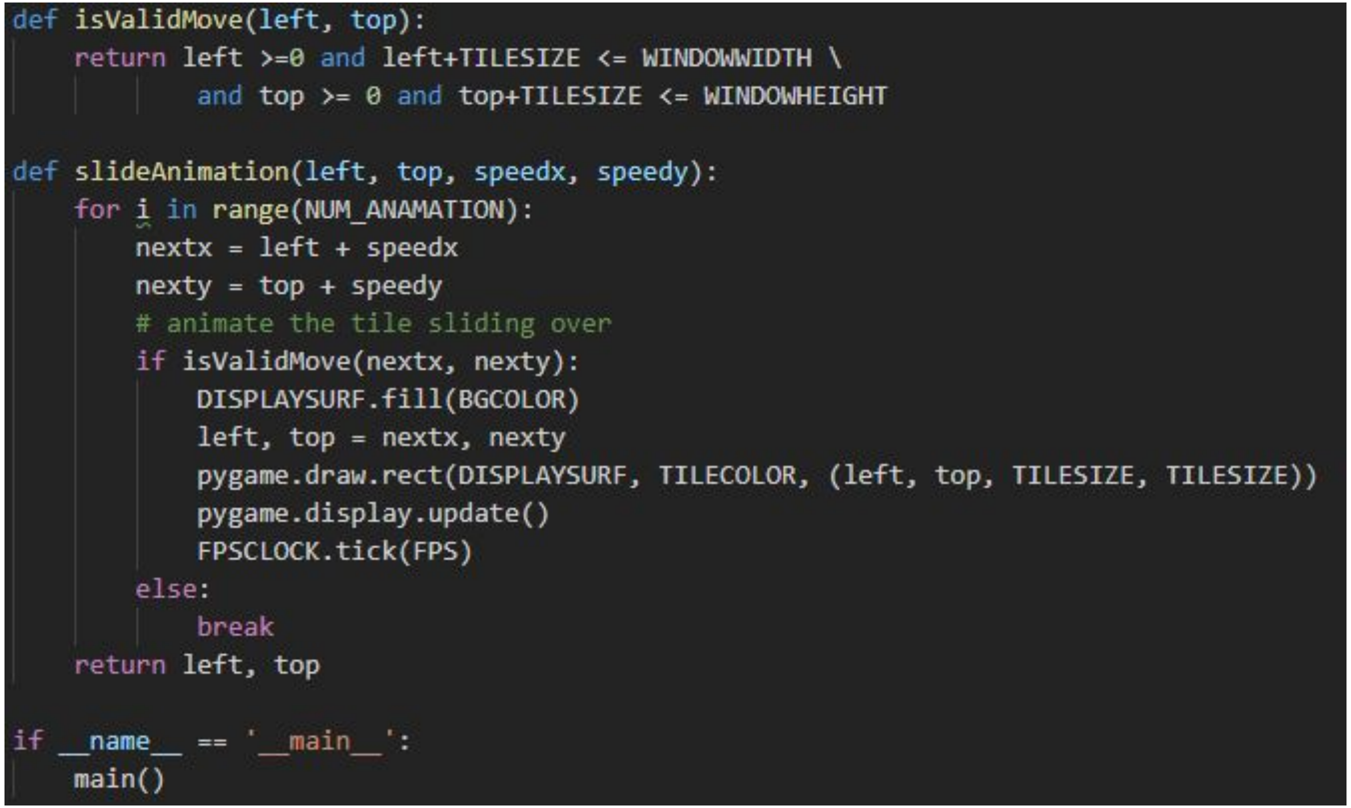




```
TILESIZE = 80
WINDOWWIDTH = 640
WINDOWHEIGHT = 480
FPS = 30

BGCOLOR =   (  0,   0,   0)
TILECOLOR = (  0, 204,   0)
NUM_ANAMATION = 8
SPEED = TILESIZE/NUM_ANAMATION

def main():
    global FPSCLOCK, DISPLAYSURF
    pygame.init()
    FPSCLOCK = pygame.time.Clock()
    DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
    pygame.display.set_caption('Slide Animation')

    left = WINDOWWIDTH/2 - TILESIZE/2
    top = WINDOWHEIGHT/2 - TILESIZE/2
```

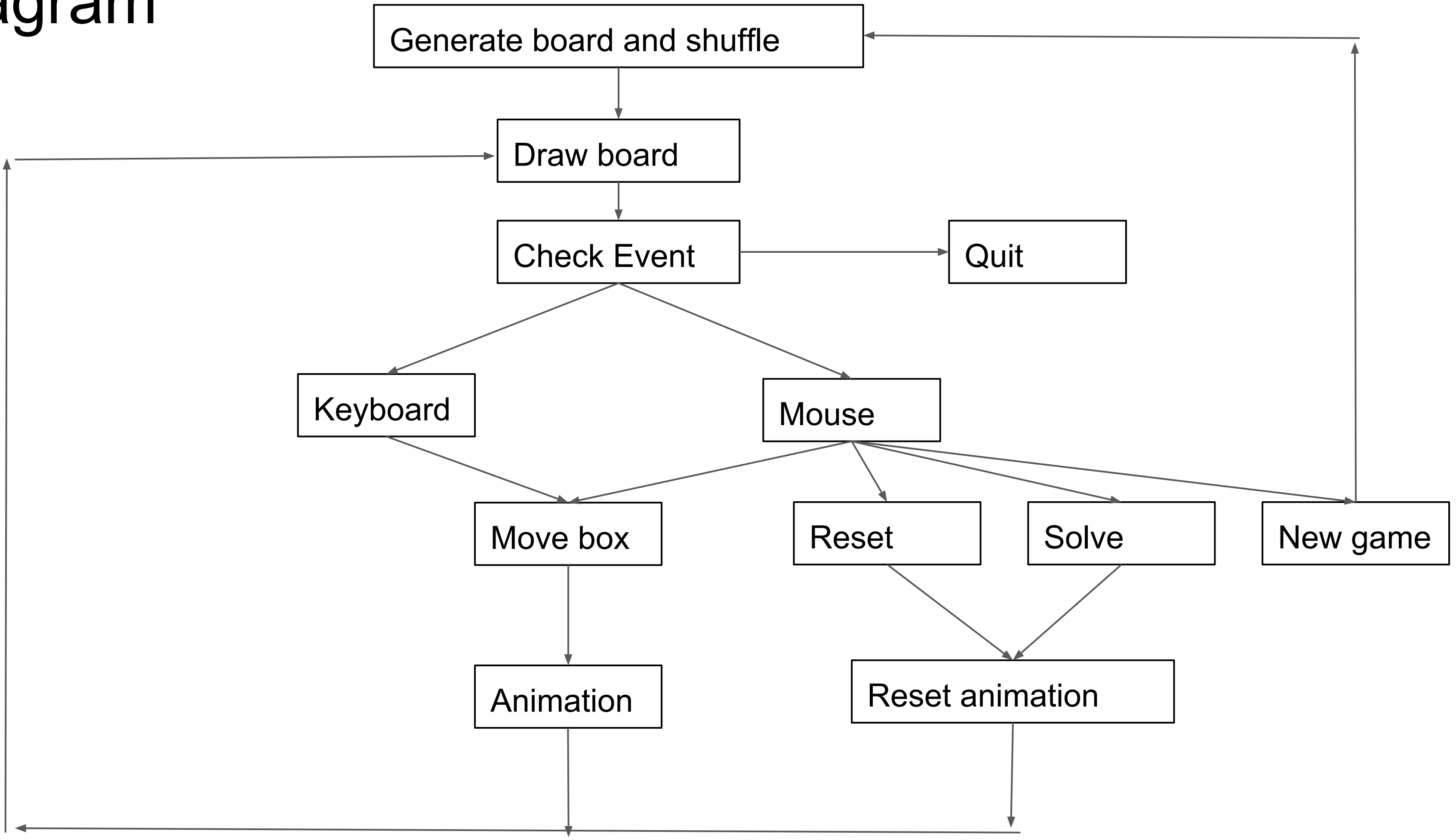


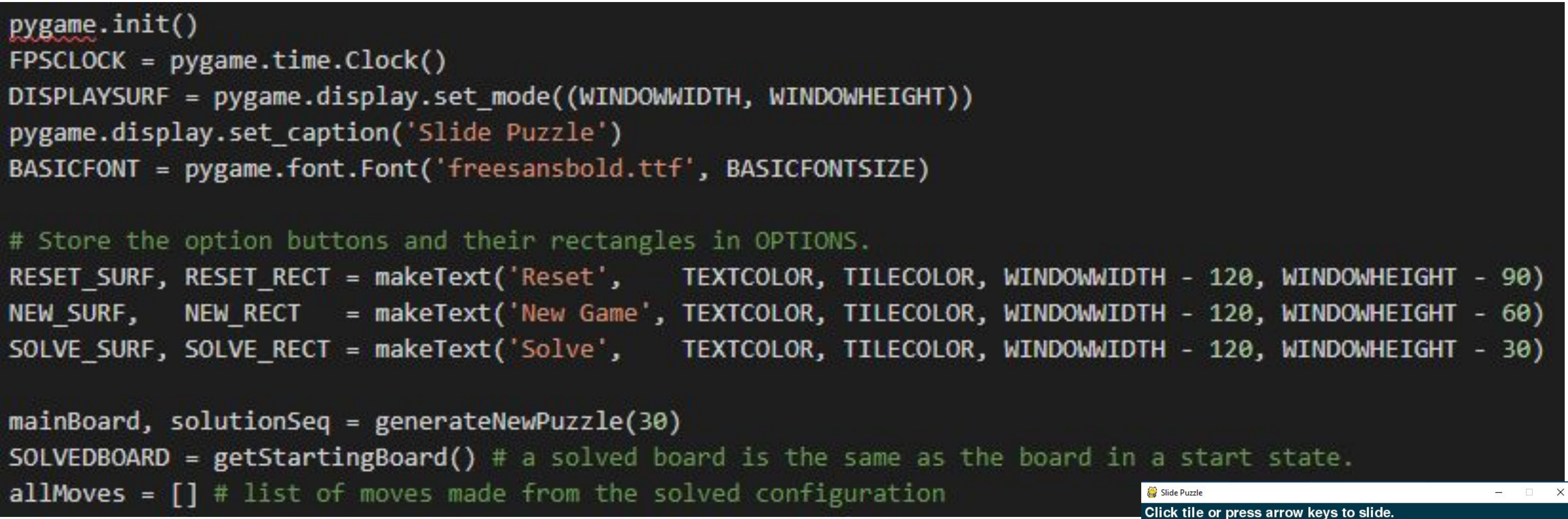
```python
while True: # main game loop
    DISPLAYSURF.fill(BGCOLOR)
    pygame.draw.rect(DISPLAYSURF, TILECOLOR, (left, top, TILESIZE, TILESIZE))
    for event in pygame.event.get(): # get all the QUIT events
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        elif event.type == KEYUP:
            # check if the user pressed a key to slide a tile
            if event.key == K_LEFT:
                left, top = slideAnimation(left, top, -SPEED, 0)
            elif event.key == K_RIGHT:
                left, top = slideAnimation(left, top, SPEED, 0)
            elif event.key == K_UP:
                left, top = slideAnimation(left, top, 0, -SPEED)
            elif event.key == K_DOWN:
                left, top = slideAnimation(left, top, 0, SPEED)

    pygame.display.update()
    FPSCLOCK.tick(FPS)
```



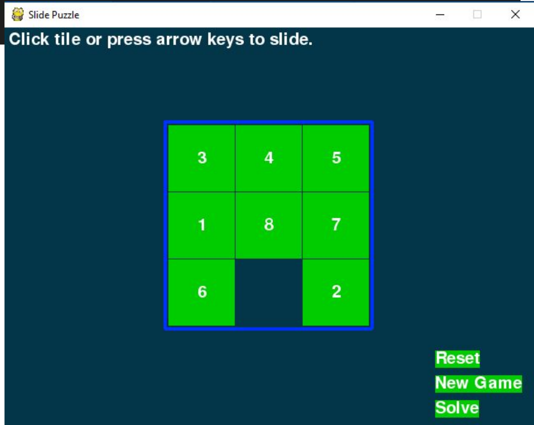
```python
def isValidMove(left, top):
    return left >=0 and left+TILESIZE <= WINDOWWIDTH \
            and top >= 0 and top+TILESIZE <= WINDOWHEIGHT

def slideAnimation(left, top, speedx, speedy):
    for i in range(NUM_ANAMATION):
        nextx = left + speedx
        nexty = top + speedy
        # animate the tile sliding over
        if isValidMove(nextx, nexty):
            DISPLAYSURF.fill(BGCOLOR)
            left, top = nextx, nexty
            pygame.draw.rect(DISPLAYSURF, TILECOLOR, (left, top, TILESIZE, TILESIZE))
            pygame.display.update()
            FPSCLOCK.tick(FPS)
        else:
            break
    return left, top

if __name__ == '__main__':
    main()
```

Diagram

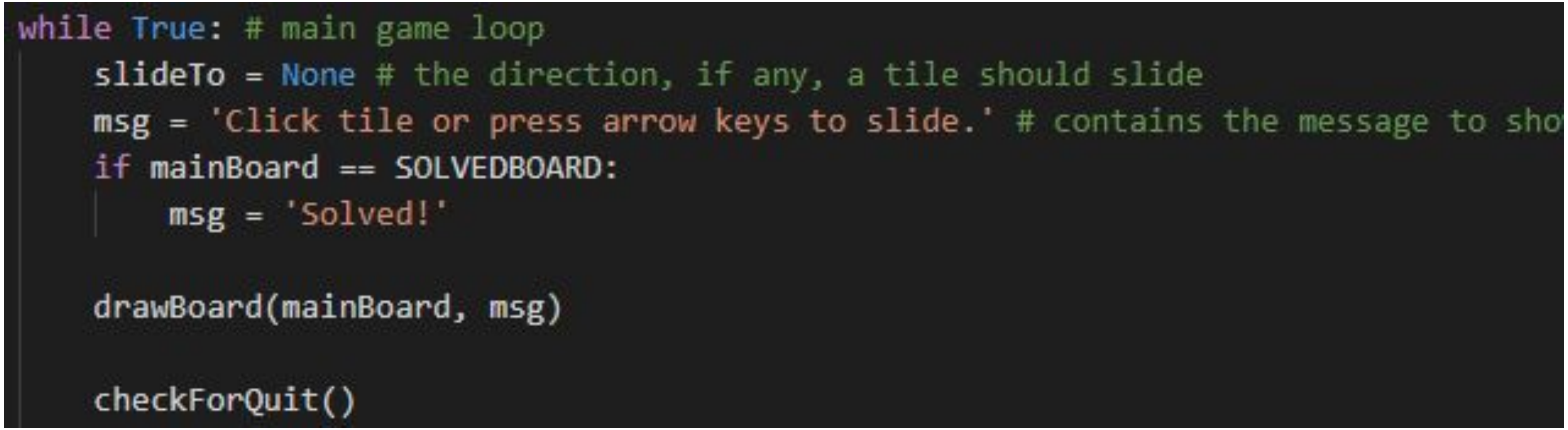

Generate board and shuffle
Draw board
Check Event
Quit
Keyboard
Mouse
Move box
Reset
Solve
New game
Animation
Reset animation



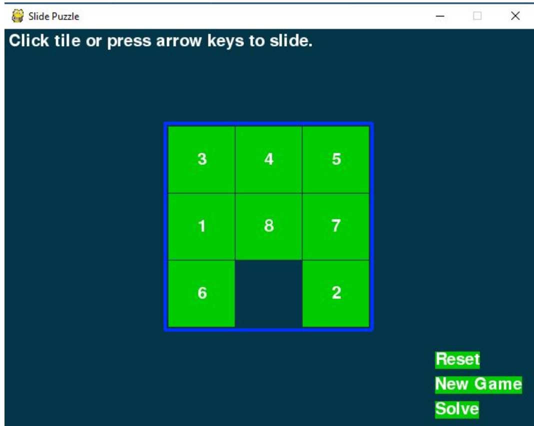
```
pygame.init()
FPSCLOCK = pygame.time.Clock()
DISPLAYSURF = pygame.display.set_mode((WINDOWWIDTH, WINDOWHEIGHT))
pygame.display.set_caption('Slide Puzzle')
BASICFONT = pygame.font.Font('freesansbold.ttf', BASICFONTSIZE)

# Store the option buttons and their rectangles in OPTIONS.
RESET_SURF, RESET_RECT = makeText('Reset',    TEXTCOLOR, TILECOLOR, WINDOWWIDTH - 120, WINDOWHEIGHT - 90)
NEW_SURF,   NEW_RECT   = makeText('New Game', TEXTCOLOR, TILECOLOR, WINDOWWIDTH - 120, WINDOWHEIGHT - 60)
SOLVE_SURF, SOLVE_RECT = makeText('Solve',    TEXTCOLOR, TILECOLOR, WINDOWWIDTH - 120, WINDOWHEIGHT - 30)

mainBoard, solutionSeq = generateNewPuzzle(30)
SOLVEDBOARD = getStartingBoard() # a solved board is the same as the board in a start state.
allMoves = [] # list of moves made from the solved configuration
```

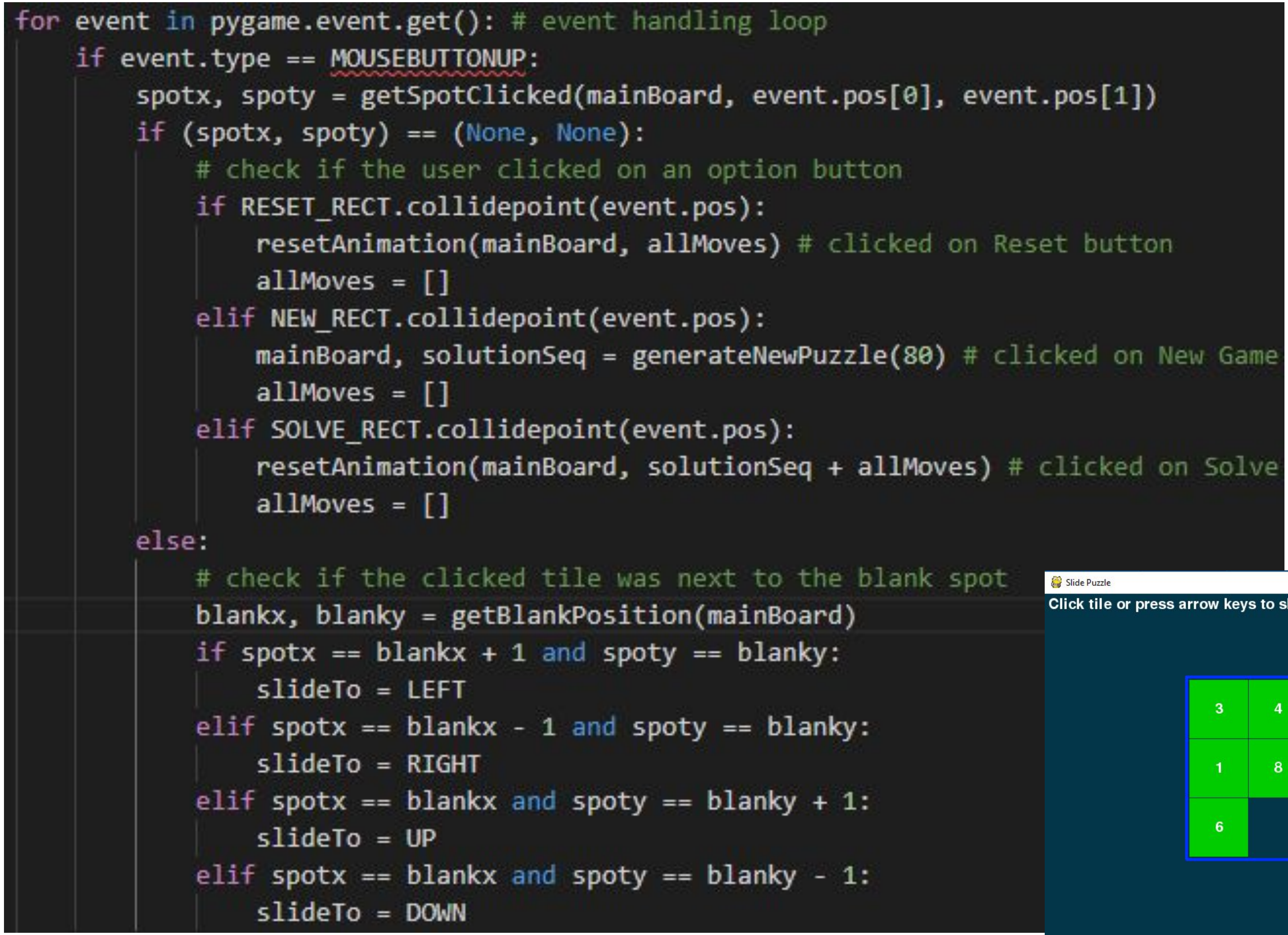


Click tile or press arrow keys to slide.

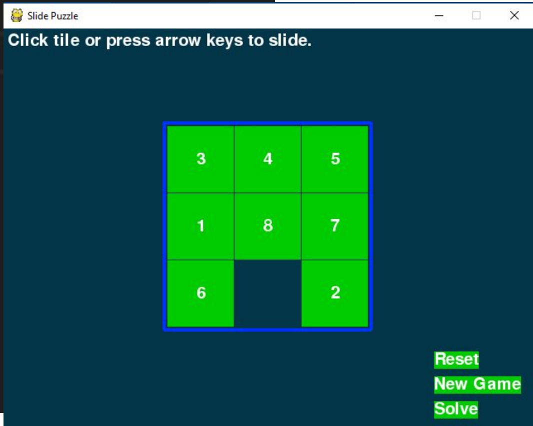


```
while True: # main game loop
    slideTo = None # the direction, if any, a tile should slide
    msg = 'Click tile or press arrow keys to slide.' # contains the message to sho
    if mainBoard == SOLVEDBOARD:
        msg = 'Solved!'

    drawBoard(mainBoard, msg)

    checkForQuit()
```

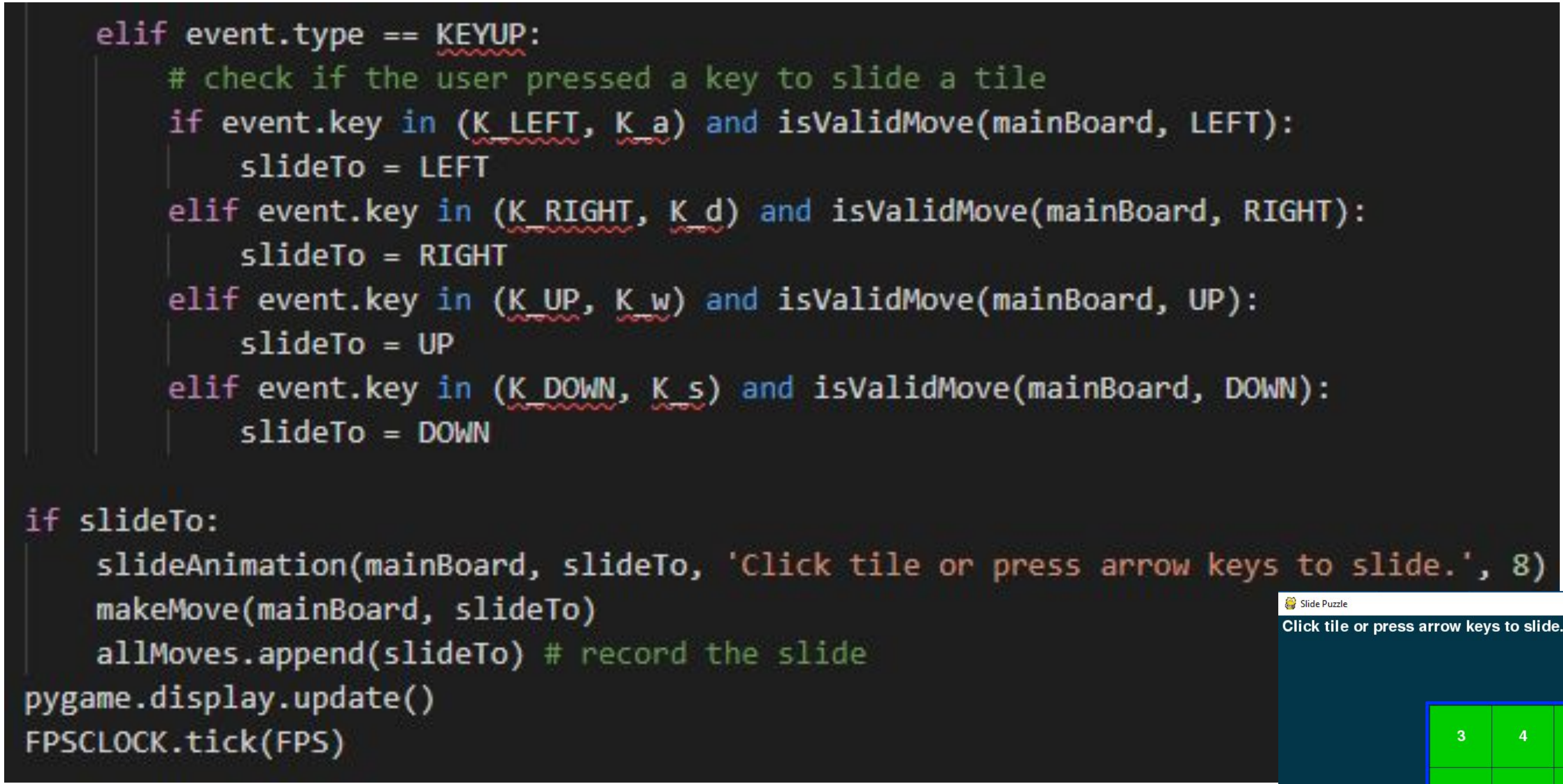




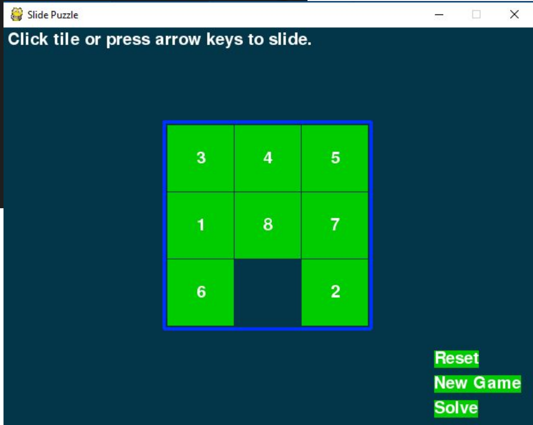
```python
for event in pygame.event.get(): # event handling loop
    if event.type == MOUSEBUTTONUP:
        spotx, spoty = getSpotClicked(mainBoard, event.pos[0], event.pos[1])
        if (spotx, spoty) == (None, None):
            # check if the user clicked on an option button
            if RESET_RECT.collidepoint(event.pos):
                resetAnimation(mainBoard, allMoves) # clicked on Reset button
                allMoves = []
            elif NEW_RECT.collidepoint(event.pos):
                mainBoard, solutionSeq = generateNewPuzzle(80) # clicked on New Game
                allMoves = []
            elif SOLVE_RECT.collidepoint(event.pos):
                resetAnimation(mainBoard, solutionSeq + allMoves) # clicked on Solve
                allMoves = []
        else:
            # check if the clicked tile was next to the blank spot
            blankx, blanky = getBlankPosition(mainBoard)
            if spotx == blankx + 1 and spoty == blanky:
                slideTo = LEFT
            elif spotx == blankx - 1 and spoty == blanky:
                slideTo = RIGHT
            elif spotx == blankx and spoty == blanky + 1:
                slideTo = UP
            elif spotx == blankx and spoty == blanky - 1:
                slideTo = DOWN
```

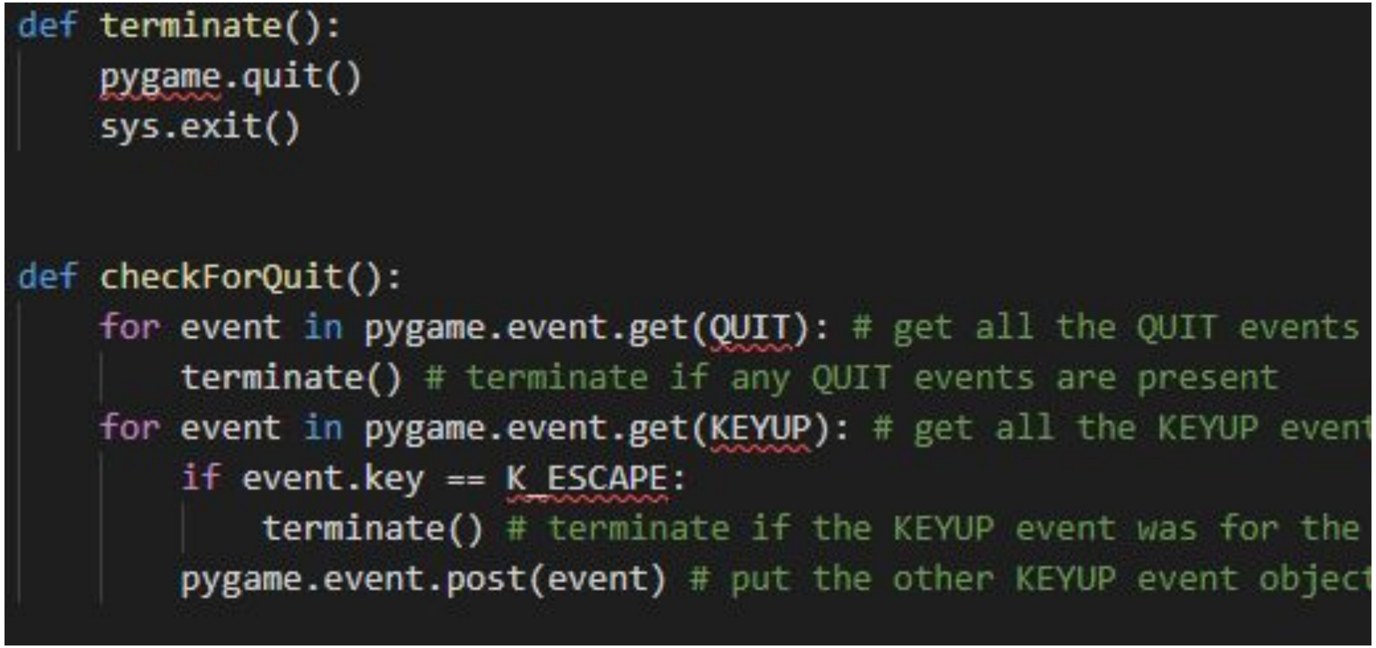


Click tile or press arrow keys to slide.

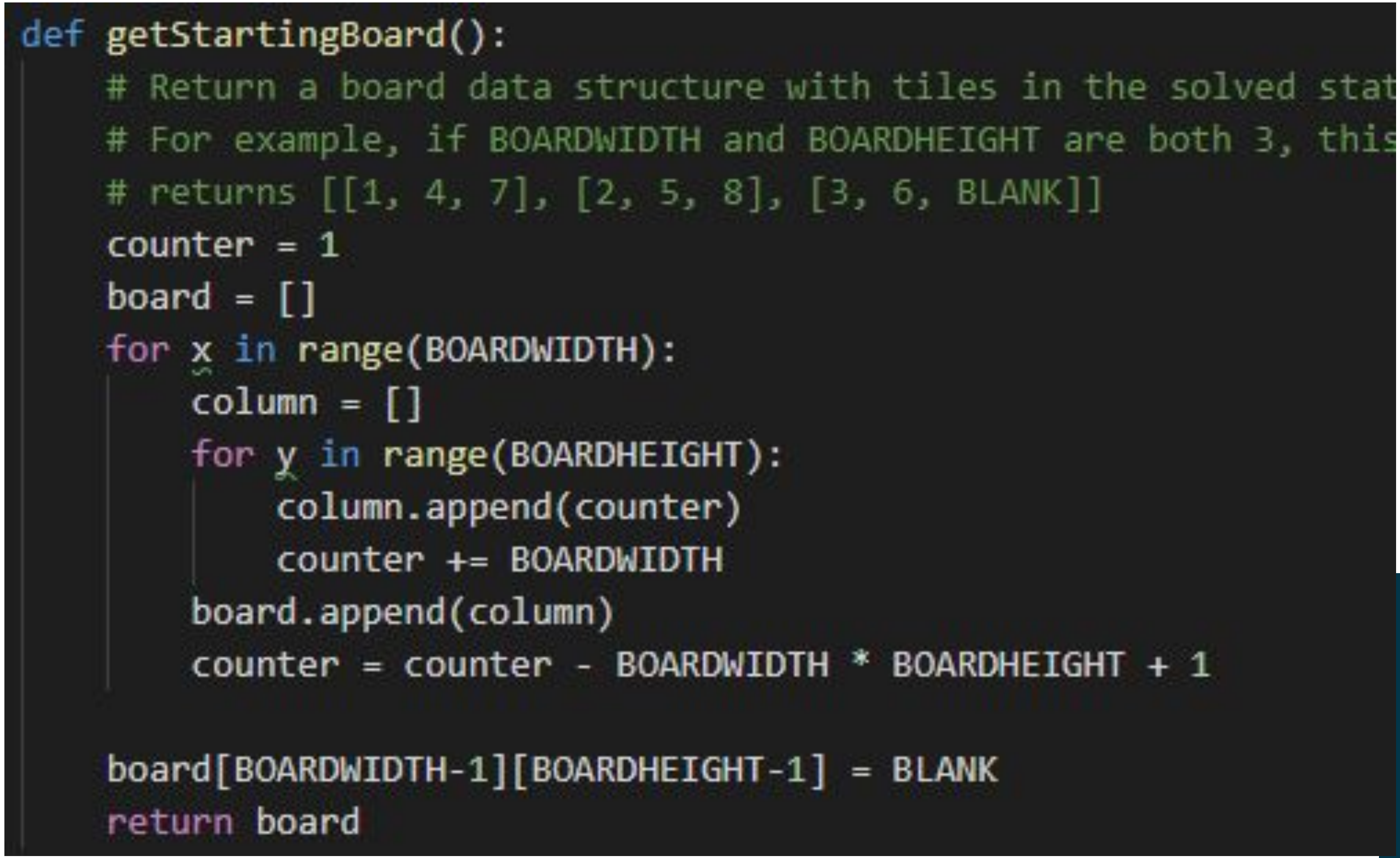


```
    elif event.type == KEYUP:
        # check if the user pressed a key to slide a tile
        if event.key in (K_LEFT, K_a) and isValidMove(mainBoard, LEFT):
            slideTo = LEFT
        elif event.key in (K_RIGHT, K_d) and isValidMove(mainBoard, RIGHT):
            slideTo = RIGHT
        elif event.key in (K_UP, K_w) and isValidMove(mainBoard, UP):
            slideTo = UP
        elif event.key in (K_DOWN, K_s) and isValidMove(mainBoard, DOWN):
            slideTo = DOWN

if slideTo:
    slideAnimation(mainBoard, slideTo, 'Click tile or press arrow keys to slide.', 8)
    makeMove(mainBoard, slideTo)
    allMoves.append(slideTo) # record the slide
pygame.display.update()
FPSCLOCK.tick(FPS)
```

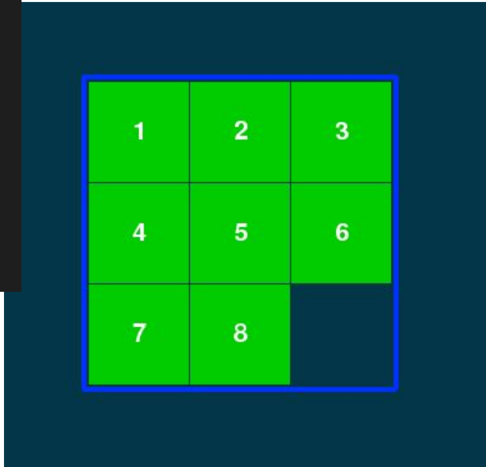


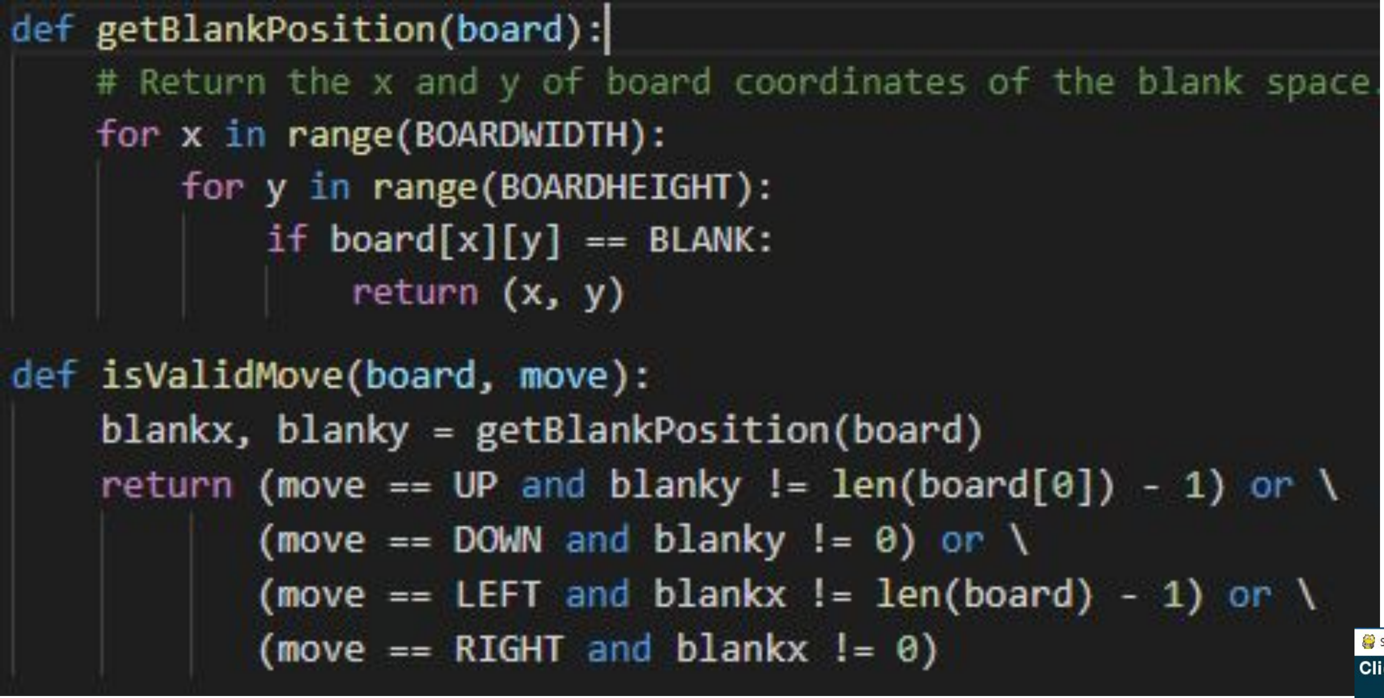


```
def terminate():
    pygame.quit()
    sys.exit()


def checkForQuit():
    for event in pygame.event.get(QUIT): # get all the QUIT events
        terminate() # terminate if any QUIT events are present
    for event in pygame.event.get(KEYUP): # get all the KEYUP event
        if event.key == K_ESCAPE:
            terminate() # terminate if the KEYUP event was for the
        pygame.event.post(event) # put the other KEYUP event object
```

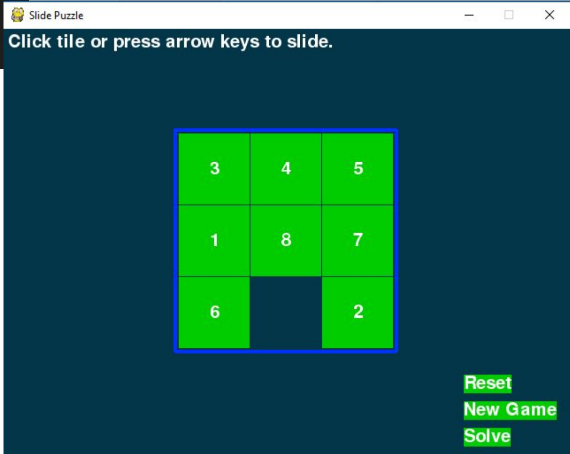


```python
def getStartingBoard():
    # Return a board data structure with tiles in the solved stat
    # For example, if BOARDWIDTH and BOARDHEIGHT are both 3, this
    # returns [[1, 4, 7], [2, 5, 8], [3, 6, BLANK]]
    counter = 1
    board = []
    for x in range(BOARDWIDTH):
        column = []
        for y in range(BOARDHEIGHT):
            column.append(counter)
            counter += BOARDWIDTH
        board.append(column)
        counter = counter - BOARDWIDTH * BOARDHEIGHT + 1

    board[BOARDWIDTH-1][BOARDHEIGHT-1] = BLANK
    return board
```

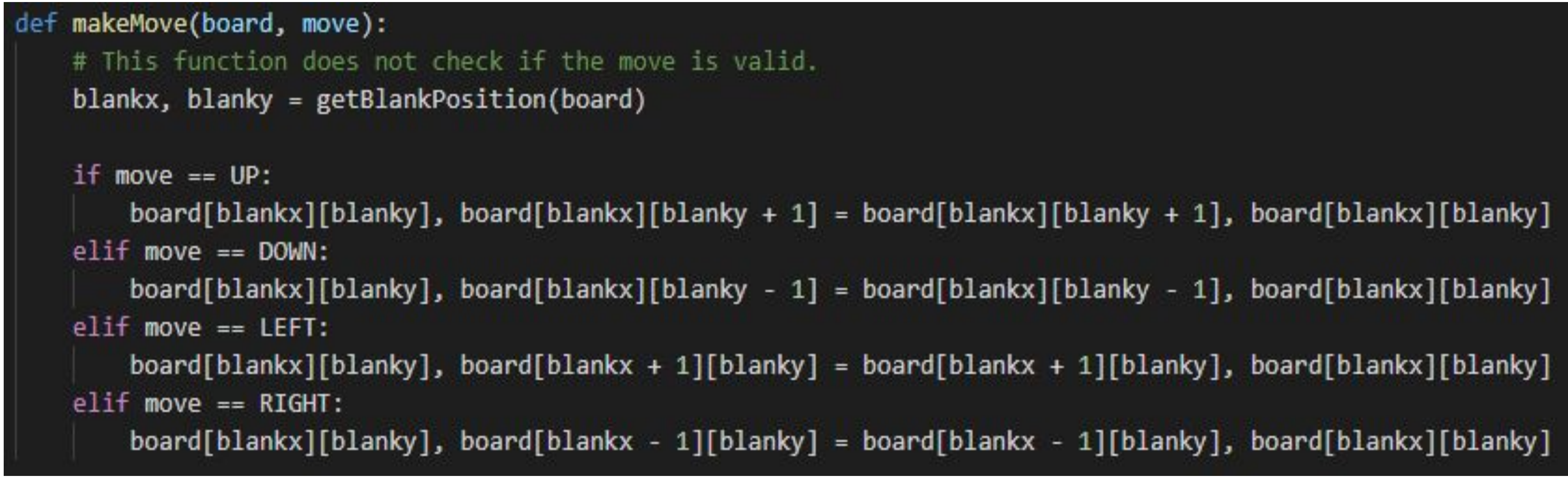




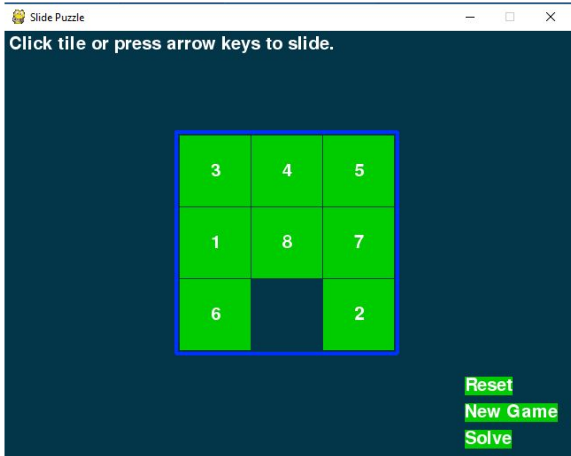
```python
def getBlankPosition(board):
    # Return the x and y of board coordinates of the blank space
    for x in range(BOARDWIDTH):
        for y in range(BOARDHEIGHT):
            if board[x][y] == BLANK:
                return (x, y)

def isValidMove(board, move):
    blankx, blanky = getBlankPosition(board)
    return (move == UP and blanky != len(board[0]) - 1) or \
           (move == DOWN and blanky != 0) or \
           (move == LEFT and blankx != len(board) - 1) or \
           (move == RIGHT and blankx != 0)
```

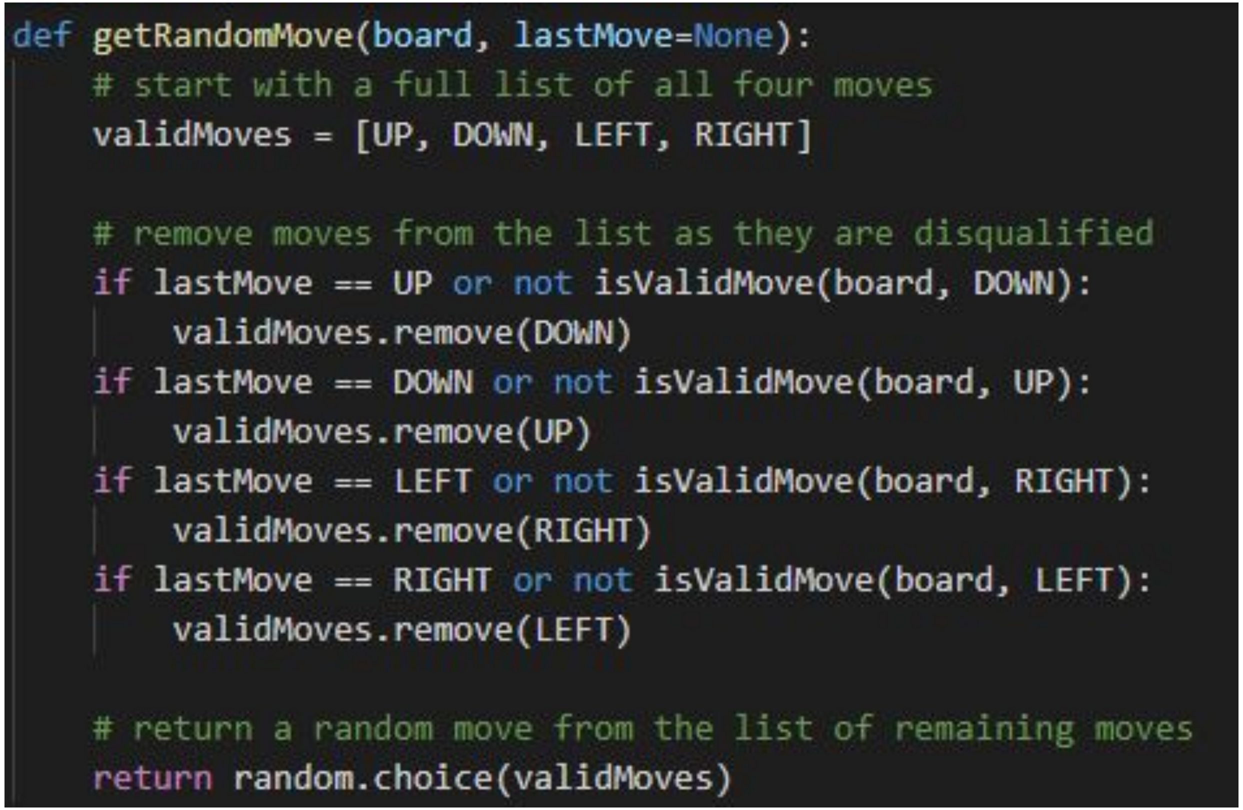


Click tile or press arrow keys to slide.

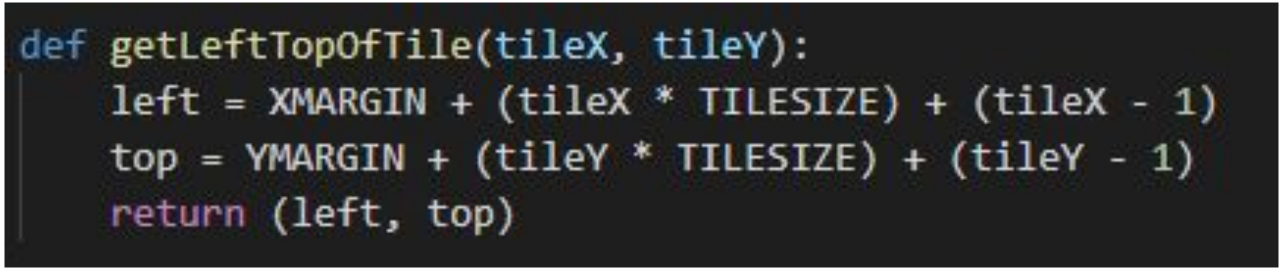


```python
def makeMove(board, move):
    # This function does not check if the move is valid.
    blankx, blanky = getBlankPosition(board)

    if move == UP:
        board[blankx][blanky], board[blankx][blanky + 1] = board[blankx][blanky + 1], board[blankx][blanky]
    elif move == DOWN:
        board[blankx][blanky], board[blankx][blanky - 1] = board[blankx][blanky - 1], board[blankx][blanky]
    elif move == LEFT:
        board[blankx][blanky], board[blankx + 1][blanky] = board[blankx + 1][blanky], board[blankx][blanky]
    elif move == RIGHT:
        board[blankx][blanky], board[blankx - 1][blanky] = board[blankx - 1][blanky], board[blankx][blanky]
```

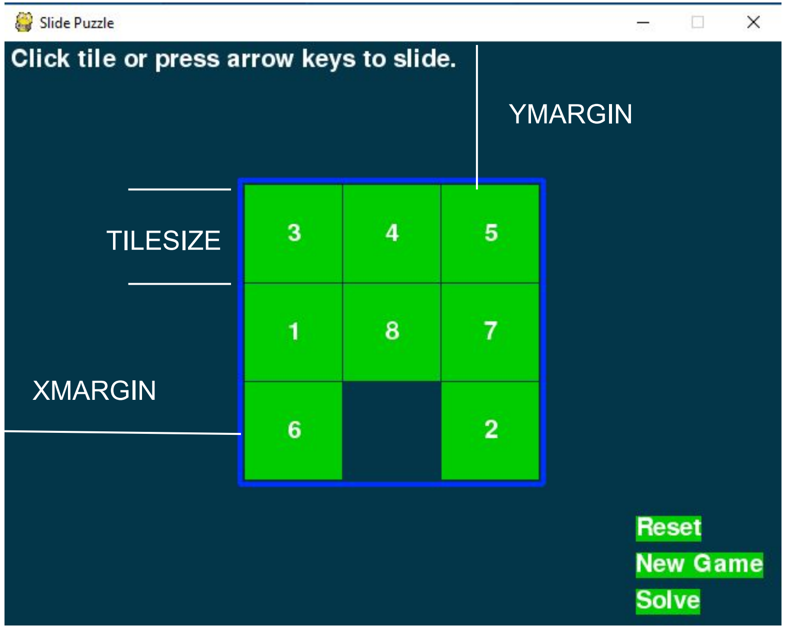


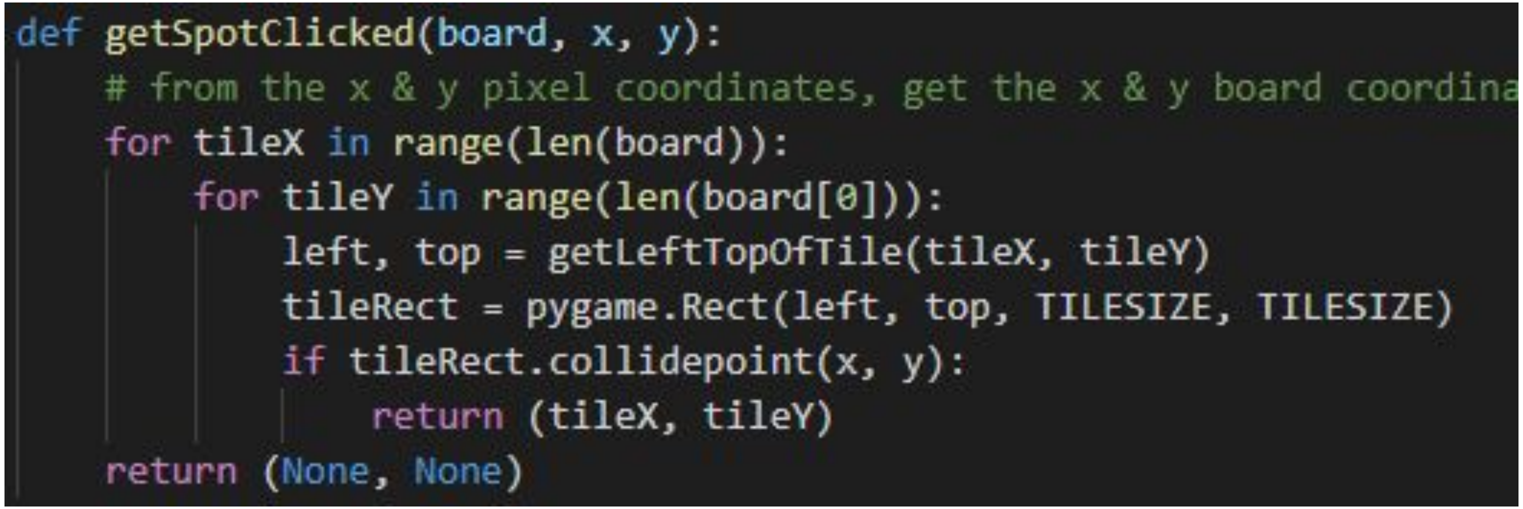


```python
def getRandomMove(board, lastMove=None):
    # start with a full list of all four moves
    validMoves = [UP, DOWN, LEFT, RIGHT]

    # remove moves from the list as they are disqualified
    if lastMove == UP or not isValidMove(board, DOWN):
        validMoves.remove(DOWN)
    if lastMove == DOWN or not isValidMove(board, UP):
        validMoves.remove(UP)
    if lastMove == LEFT or not isValidMove(board, RIGHT):
        validMoves.remove(RIGHT)
    if lastMove == RIGHT or not isValidMove(board, LEFT):
        validMoves.remove(LEFT)

    # return a random move from the list of remaining moves
    return random.choice(validMoves)
```

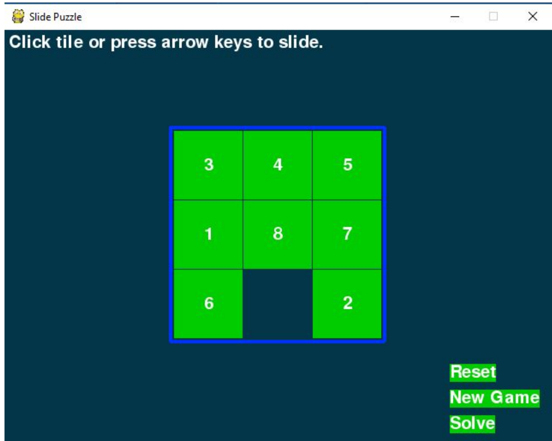


```
def getLeftTopOfTile(tileX, tileY):
    left = XMARGIN + (tileX * TILESIZE) + (tileX - 1)
    top = YMARGIN + (tileY * TILESIZE) + (tileY - 1)
    return (left, top)
```

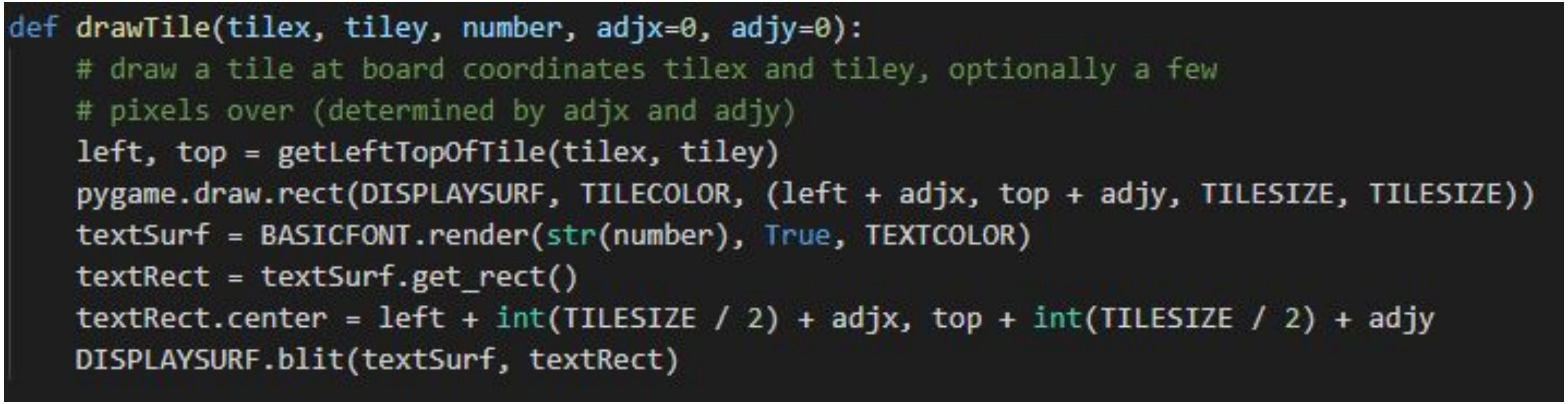




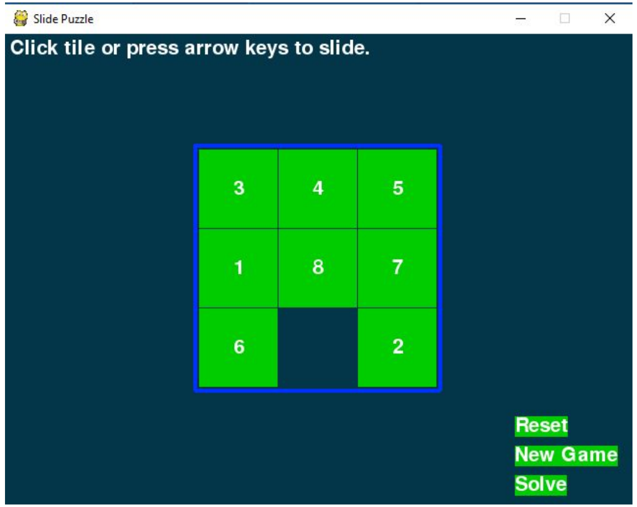
```
def getSpotClicked(board, x, y):
    # from the x & y pixel coordinates, get the x & y board coordina
    for tileX in range(len(board)):
        for tileY in range(len(board[0])):
            left, top = getLeftTopOfTile(tileX, tileY)
            tileRect = pygame.Rect(left, top, TILESIZE, TILESIZE)
            if tileRect.collidepoint(x, y):
                return (tileX, tileY)
    return (None, None)
```



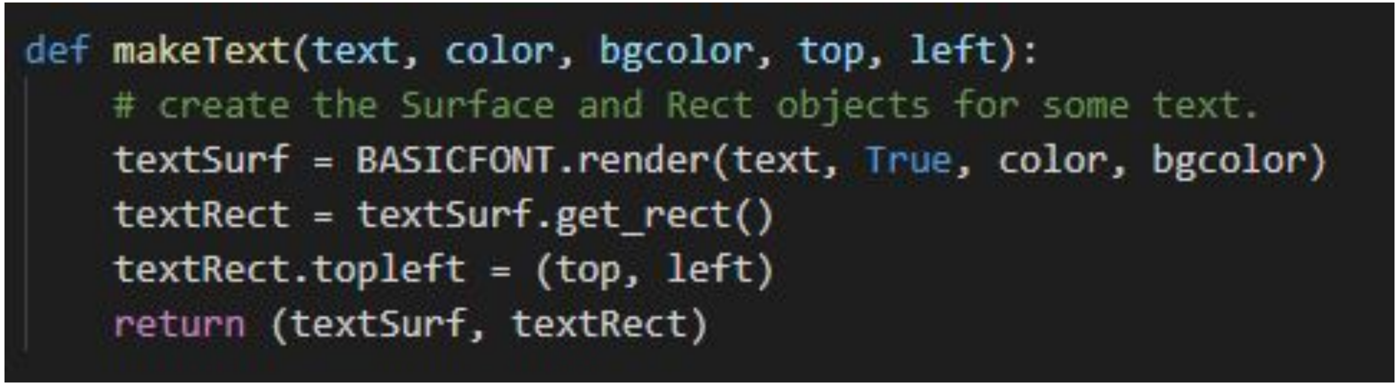
Click tile or press arrow keys to slide.



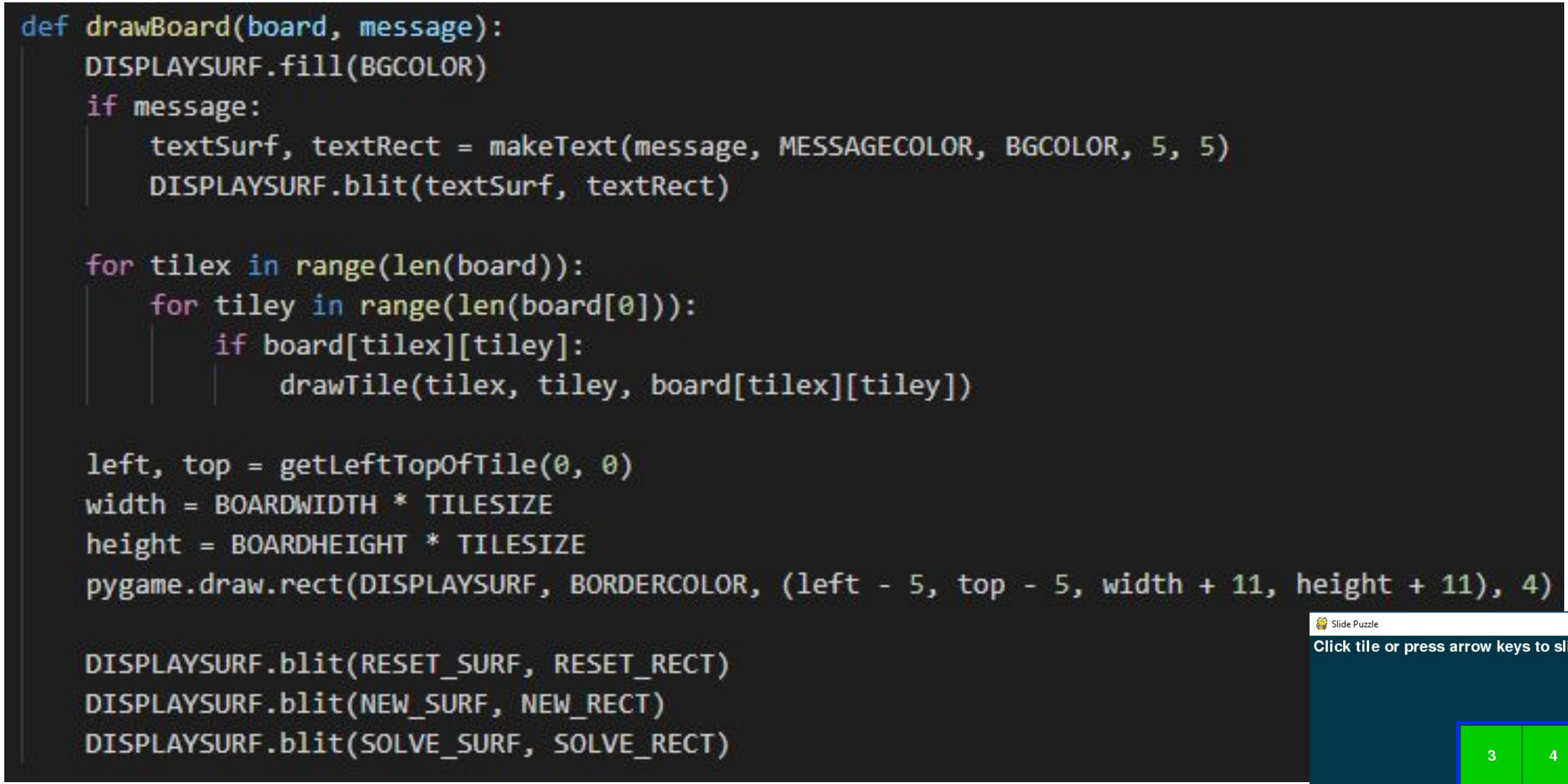
```python
def drawTile(tilex, tiley, number, adjx=0, adjy=0):
    # draw a tile at board coordinates tilex and tiley, optionally a few
    # pixels over (determined by adjx and adjy)
    left, top = getLeftTopOfTile(tilex, tiley)
    pygame.draw.rect(DISPLAYSURF, TILECOLOR, (left + adjx, top + adjy, TILESIZE, TILESIZE))
    textSurf = BASICFONT.render(str(number), True, TEXTCOLOR)
    textRect = textSurf.get_rect()
    textRect.center = left + int(TILESIZE / 2) + adjx, top + int(TILESIZE / 2) + adjy
    DISPLAYSURF.blit(textSurf, textRect)
```



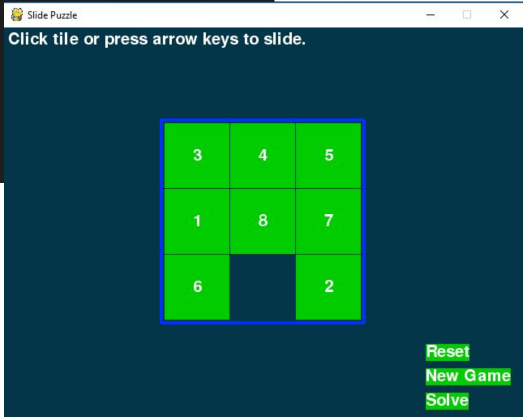
Click tile or press arrow keys to slide.

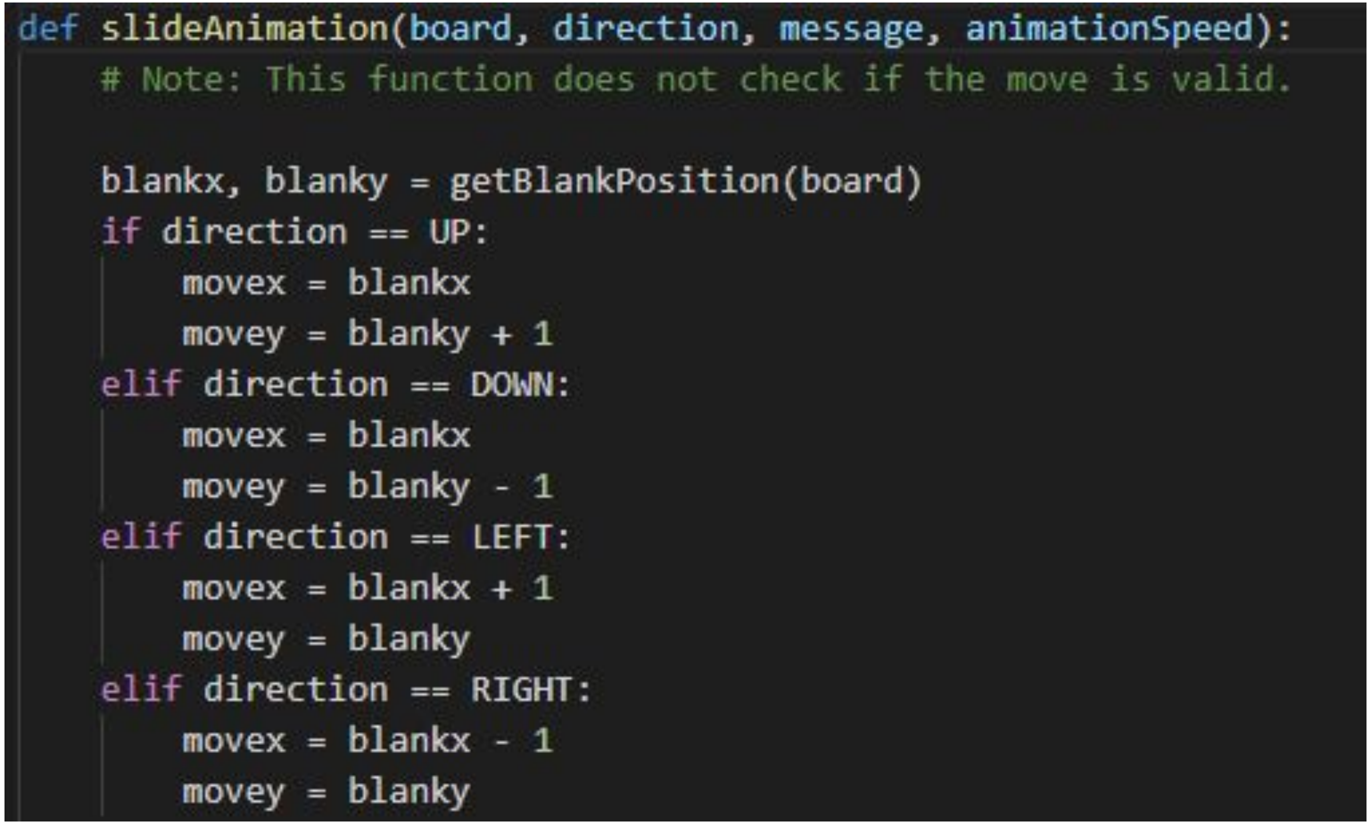


```python
def makeText(text, color, bgcolor, top, left):
    # create the Surface and Rect objects for some text.
    textSurf = BASICFONT.render(text, True, color, bgcolor)
    textRect = textSurf.get_rect()
    textRect.topleft = (top, left)
    return (textSurf, textRect)
```

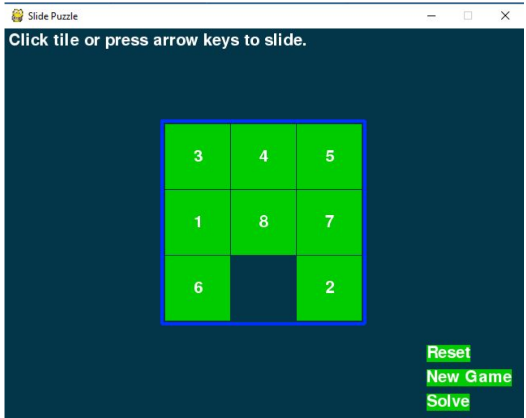


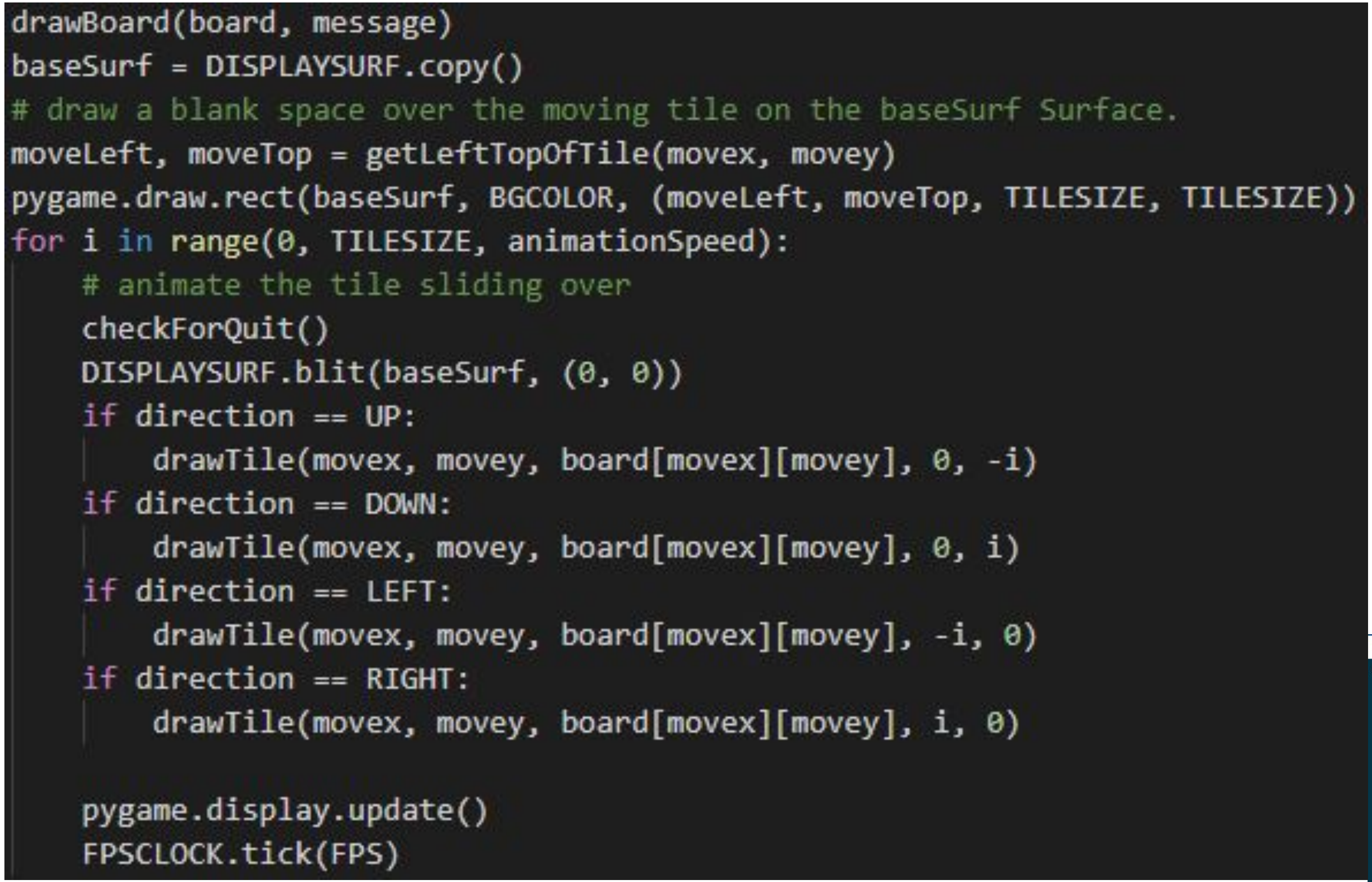
```
def drawBoard(board, message):
    DISPLAYSURF.fill(BGCOLOR)
    if message:
        textSurf, textRect = makeText(message, MESSAGECOLOR, BGCOLOR, 5, 5)
        DISPLAYSURF.blit(textSurf, textRect)

    for tilex in range(len(board)):
        for tiley in range(len(board[0])):
            if board[tilex][tiley]:
                drawTile(tilex, tiley, board[tilex][tiley])

    left, top = getLeftTopOfTile(0, 0)
    width = BOARDWIDTH * TILESIZE
    height = BOARDHEIGHT * TILESIZE
    pygame.draw.rect(DISPLAYSURF, BORDERCOLOR, (left - 5, top - 5, width + 11, height + 11), 4)

    DISPLAYSURF.blit(RESET_SURF, RESET_RECT)
    DISPLAYSURF.blit(NEW_SURF, NEW_RECT)
    DISPLAYSURF.blit(SOLVE_SURF, SOLVE_RECT)
```



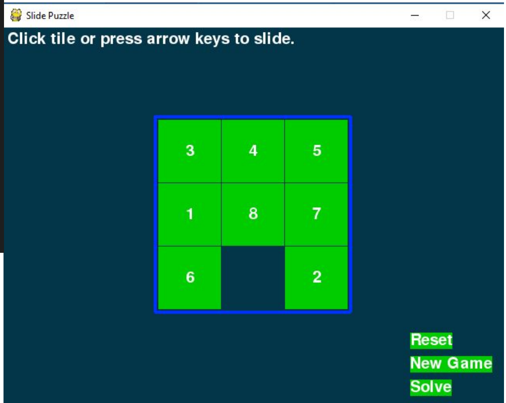


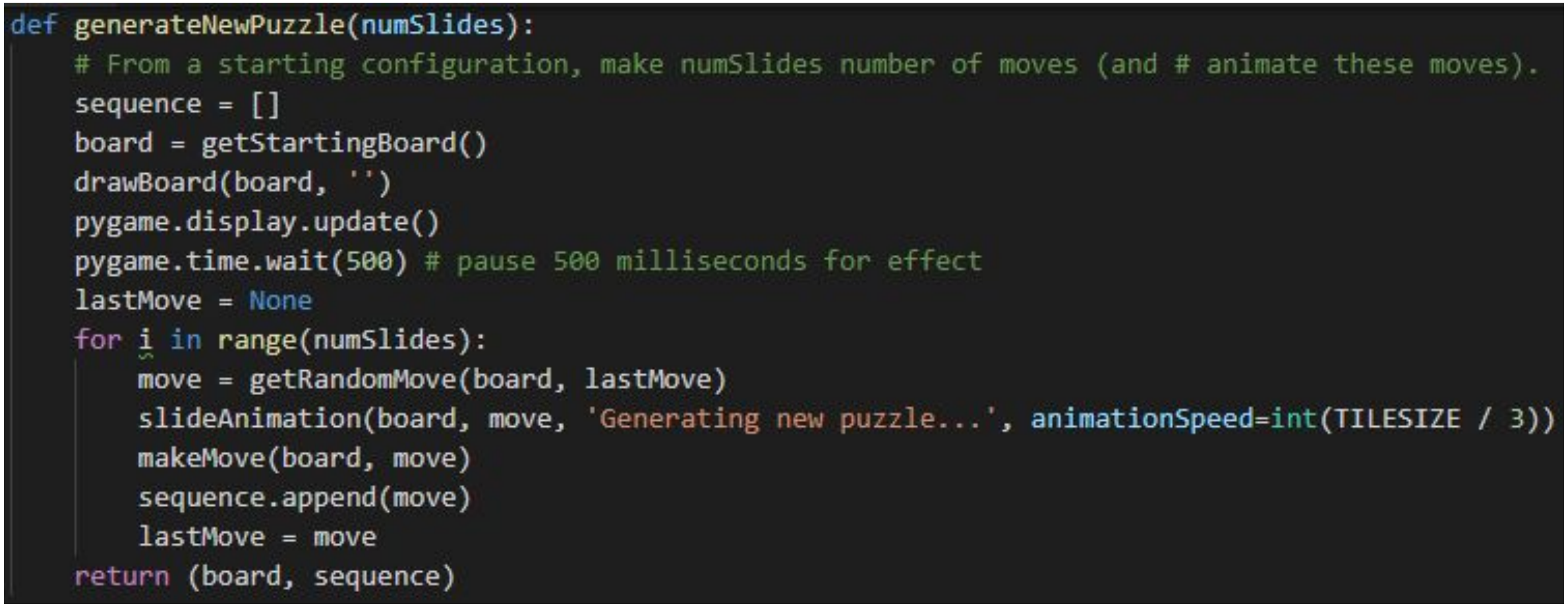
```python
def slideAnimation(board, direction, message, animationSpeed):
    # Note: This function does not check if the move is valid.

    blankx, blanky = getBlankPosition(board)
    if direction == UP:
        movex = blankx
        movey = blanky + 1
    elif direction == DOWN:
        movex = blankx
        movey = blanky - 1
    elif direction == LEFT:
        movex = blankx + 1
        movey = blanky
    elif direction == RIGHT:
        movex = blankx - 1
        movey = blanky
```



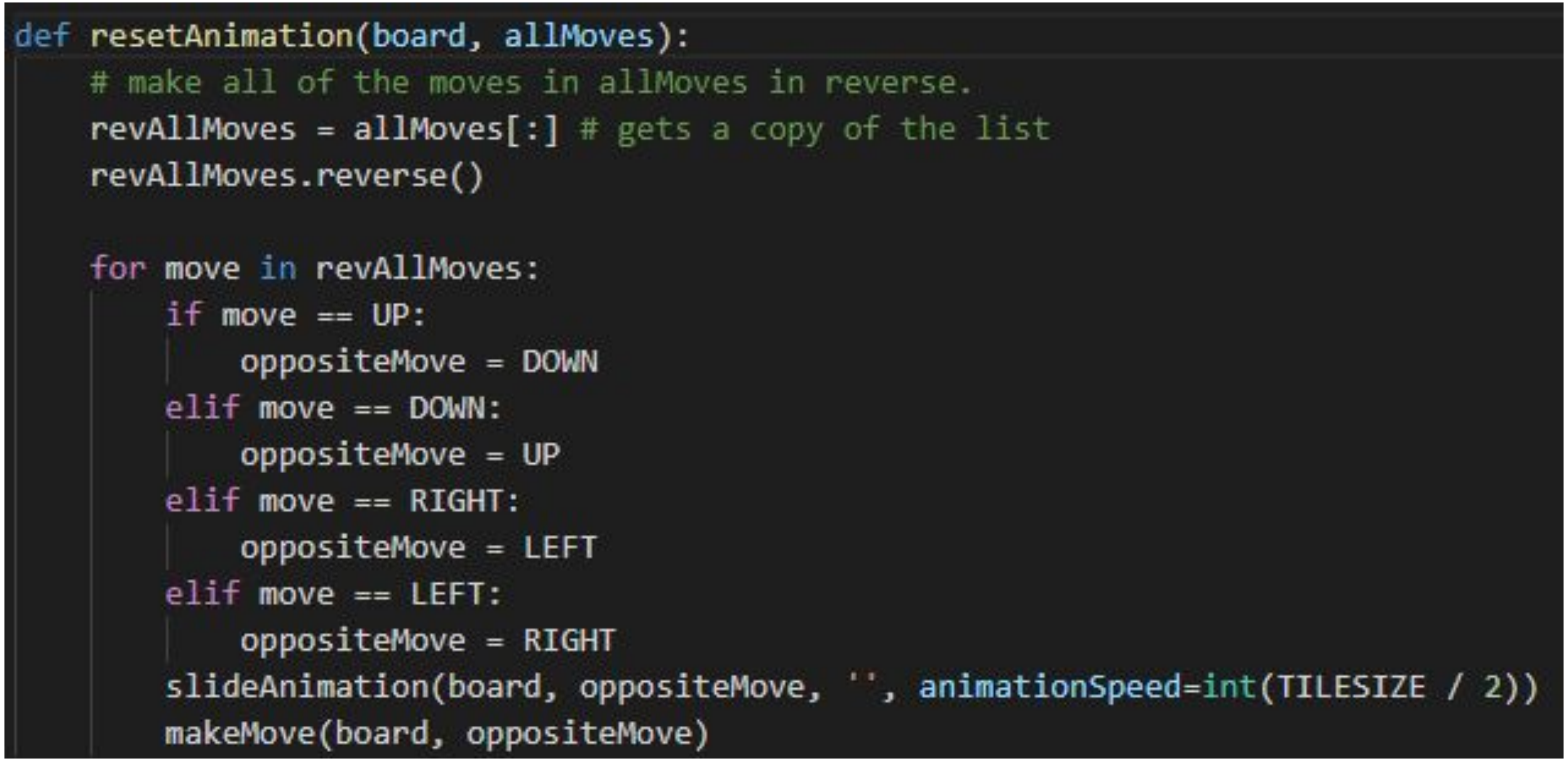


```
drawBoard(board, message)
baseSurf = DISPLAYSURF.copy()
# draw a blank space over the moving tile on the baseSurf Surface.
moveLeft, moveTop = getLeftTopOfTile(movex, movey)
pygame.draw.rect(baseSurf, BGCOLOR, (moveLeft, moveTop, TILESIZE, TILESIZE))
for i in range(0, TILESIZE, animationSpeed):
    # animate the tile sliding over
    checkForQuit()
    DISPLAYSURF.blit(baseSurf, (0, 0))
    if direction == UP:
        drawTile(movex, movey, board[movex][movey], 0, -i)
    if direction == DOWN:
        drawTile(movex, movey, board[movex][movey], 0, i)
    if direction == LEFT:
        drawTile(movex, movey, board[movex][movey], -i, 0)
    if direction == RIGHT:
        drawTile(movex, movey, board[movex][movey], i, 0)

    pygame.display.update()
    FPSCLOCK.tick(FPS)
```

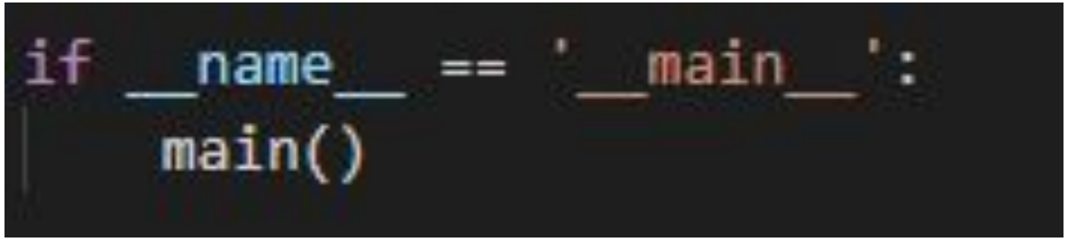




```python
def generateNewPuzzle(numSlides):
    # From a starting configuration, make numSlides number of moves (and # animate these moves).
    sequence = []
    board = getStartingBoard()
    drawBoard(board, '')
    pygame.display.update()
    pygame.time.wait(500) # pause 500 milliseconds for effect
    lastMove = None
    for i in range(numSlides):
        move = getRandomMove(board, lastMove)
        slideAnimation(board, move, 'Generating new puzzle...', animationSpeed=int(TILESIZE / 3))
        makeMove(board, move)
        sequence.append(move)
        lastMove = move
    return (board, sequence)
```

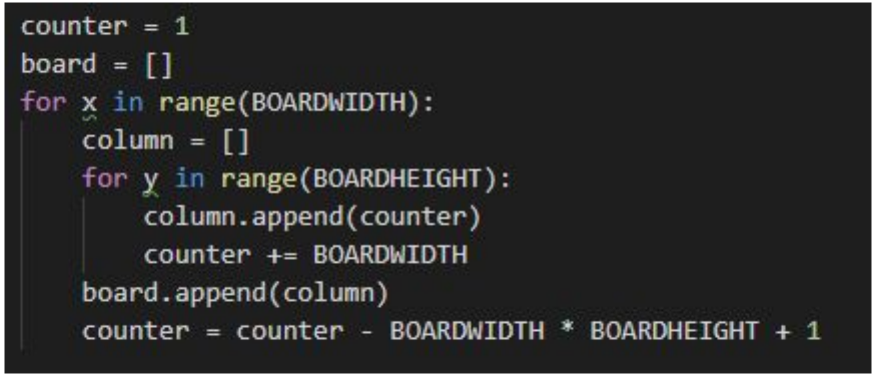


```python
def resetAnimation(board, allMoves):
    # make all of the moves in allMoves in reverse.
    revAllMoves = allMoves[:] # gets a copy of the list
    revAllMoves.reverse()

    for move in revAllMoves:
        if move == UP:
            oppositeMove = DOWN
        elif move == DOWN:
            oppositeMove = UP
        elif move == RIGHT:
            oppositeMove = LEFT
        elif move == LEFT:
            oppositeMove = RIGHT
        slideAnimation(board, oppositeMove, '', animationSpeed=int(TILESIZE / 2))
        makeMove(board, oppositeMove)
```

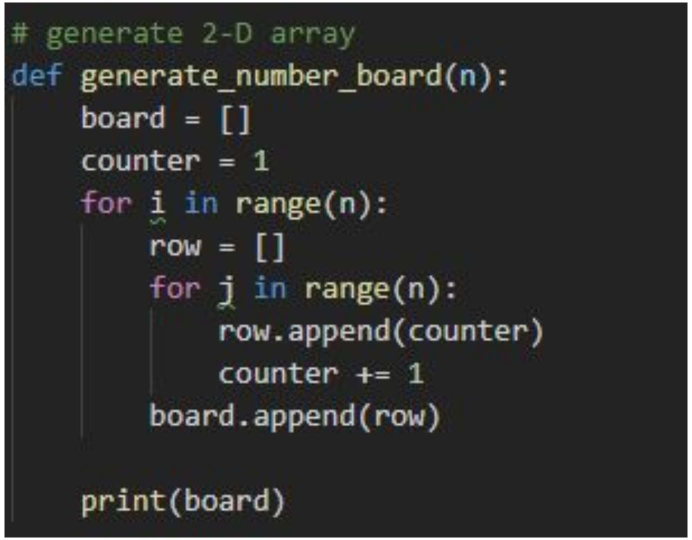


```python
if __name__ == '__main__':
    main()
```

Another Way to generate board :



```
counter = 1
board = []
for x in range(BOARDWIDTH):
    column = []
    for y in range(BOARDHEIGHT):
        column.append(counter)
        counter += BOARDWIDTH
    board.append(column)
    counter = counter - BOARDWIDTH * BOARDHEIGHT + 1
```



```
# generate 2-D array
def generate_number_board(n):
    board = []
    counter = 1
    for i in range(n):
        row = []
        for j in range(n):
            row.append(counter)
            counter += 1
        board.append(row)

    print(board)
```