```
# Slide Puzzle
# By Al Sweigart al@inventwithpython.com
# http://inventwithpython.com/pygame
# Released under a "Simplified BSD" license

import pygame, sys, random
from pygame.locals import *

# Create the constants (go ahead and experiment with different values)
BOARDWIDTH = 4  # number of columns in the board
BOARDHEIGHT = 4 # number of rows in the board
TILESIZE = 80
WINDOWWIDTH = 640
WINDOWHEIGHT = 480
FPS = 30
BLANK = None

#                 R    G    B
BLACK =         (  0,   0,   0)
WHITE =         (255, 255, 255)
BRIGHTBLUE =    (  0,  50, 255)
DARKTURQUOISE = (  3,  54,  73)
GREEN =         (  0, 204,   0)

BGCOLOR = DARKTURQUOISE
TILECOLOR = GREEN
TEXTCOLOR = WHITE
BORDERCOLOR = BRIGHTBLUE
BASICFONTSIZE = 20

BUTTONCOLOR = WHITE
BUTTONTEXTCOLOR = BLACK
MESSAGECOLOR = WHITE

XMARGIN = int((WINDOWWIDTH - (TILESIZE * BOARDWIDTH + (BOARDWIDTH - 1))) / 2)
YMARGIN = int((WINDOWHEIGHT - (TILESIZE * BOARDHEIGHT + (BOARDHEIGHT - 1))) / 2)

UP = 'up'
DOWN = 'down'
LEFT = 'left'
RIGHT = 'right'

def main():
    global FPSCLOCK, DISPLAYSURF, BASICFONT, RESET_SURF, RESET_RECT, NEW_SURF, NEW_RECT, SOLVE_SURF, SOLVE_RECT

    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(80)
SOLVEDBOARD = getStartingBoard() # a solved board is the same as the board in a start state.
allMoves = [] # list of moves made from the solved configuration

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 show in the upper left corner.
if mainBoard == SOLVEDBOARD:
    msg = 'Solved!'

drawBoard(mainBoard, msg)

checkForQuit()
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 button
            allMoves = []
        elif SOLVE_RECT.collidepoint(event.pos):
            resetAnimation(mainBoard, solutionSeq + allMoves) # clicked on Solve button
            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

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) # show slide on screen
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 events
if event.key == K_ESCAPE:
terminate() # terminate if the KEYUP event was for the Esc key
pygame.event.post(event) # put the other KEYUP event objects back


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

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

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


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 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]


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)


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 coordinates
    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)


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)


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)
```

```python


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)


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

    # prepare the base surface
    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)


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)


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)


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