Internet Of Things (UFCFVK -15 - 2)

# TETRIS GAME

Ahmed Ali

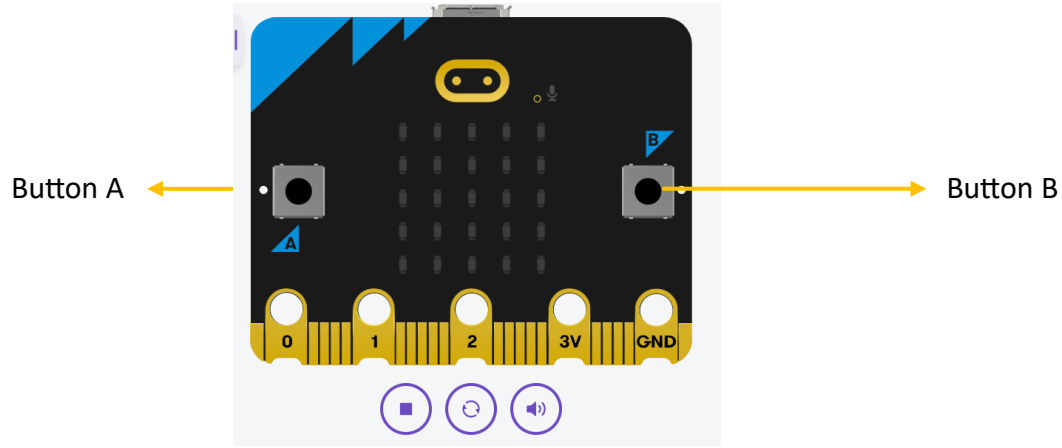
S2201026

# TETRIS GAME DOCUMENTATION

```
# Set up the tetris grid
grid = [[1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 1, 1]]

# Store a list of 4 bricks, each brick is a 2x2 grid
bricks = [[[9, 9], [9, 0]],
          [[9, 9], [0, 9]],
          [[9, 9], [9, 9]],
          [[9, 9], [0, 0]]]
```

1. Grid and Bricks: A 2D list (grid) representing a 6x7 grid is used for the game's playing surface. Each grid cell has two possible states: empty (0) or filled with a brick fragment (1 or 9). There are four different types of bricks, which are represented by 2x2 grids (bricks list).




2. Display: The game is shown on the Micro:bit using its LED matrix. Individual grid pixels can be managed using the display.set_pixel() function.

**3. Movement**: The player can move the current brick left (**button_a**), right (**button_b**), or rotate it (**button_a** + **button_b**). The **moveBrick()** function checks for collisions with other blocks or grid boundaries before allowing a movement.

```
# Function to update the game state
def update_game():
    global frameCount, x, y, brick, score, gameOver
    sleep(50)
    frameCount += 1
    # Capture user inputs
    if button_a.is_pressed() and button_b.is_pressed():
        rotateBrick()
    elif button_a.is_pressed():
        moveBrick(-1, 0)
    elif button_b.is_pressed():
        moveBrick(1, 0)
```

```
# Main Program Loop
while not gameOver:
    update_game()
```

**4. Gravity**: The bricks fall automatically at a regular interval (every 15 frames). The **update_game()** function tracks the frame count and moves the brick down if possible. If the brick cannot move down further, it becomes a part of the grid, and a new brick is selected.

5. Completing Lines: The checkLines() method determines whether any row in the grid has all available blocks filled in (9). In that case, it eliminates that line, raises the score, and moves the lines above it downward as shown below.

```
# Every 15 frames try to move the brick down
if frameCount == 15 and moveBrick(0, 1) == False:
    frameCount = 0
    # The move was not possible, the brick stays in position
    grid[y][x] = max(brick[0][0], grid[y][x])
    grid[y][x + 1] = max(brick[0][1], grid[y][x + 1])
    grid[y + 1][x] = max(brick[1][0], grid[y + 1][x])
    grid[y + 1][x + 1] = max(brick[1][1], grid[y + 1][x + 1])

    if checkLines() == False and y == 0:
        # The brick has reached the top of the grid - Game Over
        game_over()
    else:
        # Select a new brick randomly
        x = 3
        y = 0
        brick = choice(bricks)
        showBrick()
```

Game Over: If a newly placed brick reaches the top row (y == 0) and cannot descend any further, the game is over. The game_over() method is used, which outputs the result.

```
# Main Program Loop
while not gameOver:
    update_game()

# End of Game
sleep(2000)
display.scroll("Game Over: Score: " + str(score))
```

```
# Function to handle game over
def game_over():
    global gameOver
    gameOver = True
    display.scroll("Game Over: Score: " + str(score))
```

## The CODE for the TETRIS GAME

```
from microbit import *
from random import choice

# Set up the tetris grid
grid = [[1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 1, 1, 1]]

# Store a list of 4 bricks, each brick is a 2x2 grid
bricks = [[[9, 9], [9, 0]],
          [[9, 9], [0, 9]],
          [[9, 9], [9, 9]],
          [[9, 9], [0, 0]]]

# Select a brick randomly and position it at the center/top of the grid (y=0,x=3)
brick = choice(bricks)
x = 3
y = 0
frameCount = 0
score = 0
gameOver = False

# Function to return the maximum of two values
def max(a, b):
    if a >= b:
        return a
    else:
        return b

# Function to hide the 2x2 brick on the LED screen
def hideBrick():
    if x > 0:
        display.set_pixel(x - 1, y, grid[y][x])
    if x < 5:
        display.set_pixel(x + 1 - 1, y, grid[y][x + 1])
```

```
    if x > 0 and y < 4:
        display.set_pixel(x - 1, y + 1, grid[y + 1][x])
    if x < 5 and y < 4:
        display.set_pixel(x + 1 - 1, y + 1, grid[y + 1][x + 1])

# Function to show the 2x2 brick on the LED screen
def showBrick():
    if x > 0:
        display.set_pixel(x - 1, y, max(brick[0][0], grid[y][x]))
    if x < 5:
        display.set_pixel(x + 1 - 1, y, max(brick[0][1], grid[y][x + 1]))
    if x > 0 and y < 4:
        display.set_pixel(x - 1, y + 1, max(brick[1][0], grid[y + 1][x]))
    if x < 5 and y < 4:
        display.set_pixel(x + 1 - 1, y + 1, max(brick[1][1], grid[y + 1][x + 1]))

# Function to rotate the 2x2 brick
def rotateBrick():
    pixel00 = brick[0][0]
    pixel01 = brick[0][1]
    pixel10 = brick[1][0]
    pixel11 = brick[1][1]
    # Check if the rotation is possible
    if not ((grid[y][x] > 0 and pixel00 > 0) or (grid[y + 1][x] > 0 and pixel10 > 0) or
            (grid[y][x + 1] > 0 and pixel01 > 0) or (grid[y + 1][x + 1] > 0 and pixel11 >
0)):
        hideBrick()
        brick[0][0] = pixel10
        brick[1][0] = pixel11
        brick[1][1] = pixel01
        brick[0][1] = pixel00
        showBrick()

# Function to move/translate the brick
def moveBrick(delta_x, delta_y):
    global x, y
    move = False
    # Check if the move is possible: no collision with other blocks or borders of
the grid
    if delta_x == -1 and x > 0:
```

```
        if not ((grid[y][x - 1] > 0 and brick[0][0] > 0) or (grid[y][x + 1 - 1] > 0 and
brick[0][1] > 0) or
                (grid[y + 1][x - 1] > 0 and brick[1][0] > 0) or (grid[y + 1][x + 1 - 1] > 0
and brick[1][1] > 0)):
            move = True
    elif delta_x == 1 and x < 5:
        if not ((grid[y][x + 1] > 0 and brick[0][0] > 0) or (grid[y][x + 1 + 1] > 0 and
brick[0][1] > 0) or
                (grid[y + 1][x + 1] > 0 and brick[1][0] > 0) or (grid[y + 1][x + 1 + 1] > 0
and brick[1][1] > 0)):
            move = True
    elif delta_y == 1 and y < 4:
        if not ((grid[y + 1][x] > 0 and brick[0][0] > 0) or (grid[y + 1][x + 1] > 0 and
brick[0][1] > 0) or
                (grid[y + 1 + 1][x] > 0 and brick[1][0] > 0) or (grid[y + 1 + 1][x + 1] > 0
and brick[1][1] > 0)):
            move = True
    # If the move is possible, update x,y coordinates of the brick
    if move:
        hideBrick()
        x += delta_x
        y += delta_y
        showBrick()
    # Return True or False to confirm if the move took place
    return move

# Function to check for and remove completed lines
def checkLines():
    removeLine = False
    # Check each line one at a time
    for i in range(0, 5):
        # If 5 blocks are filled in (9) then a line is complete (9*5=45)
        if sum(grid[i][1:6]) == 45:
            removeLine = True
            # Increment the score (10 pts per line)
            score += 10
            # Remove the line and make all lines above fall by 1
            for j in range(i, 0, -1):
                grid[j] = grid[j - 1]
            grid[0] = [1, 0, 0, 0, 0, 0, 1]
```

```
    if removeLine:
        #Refresh the LED screen
        for i in range(0, 5):
            for j in range(0, 5):
                display.set_pixel(i, j, grid[j][i + 1])
    return removeLine

# Function to handle game over
def game_over():
    global gameOver
    gameOver = True
    display.scroll("Game Over: Score: " + str(score))

# Function to update the game state
def update_game():
    global frameCount, x, y, brick, score, gameOver
    sleep(50)
    frameCount += 1
    # Capture user inputs
    if button_a.is_pressed() and button_b.is_pressed():
        rotateBrick()
    elif button_a.is_pressed():
        moveBrick(-1, 0)
    elif button_b.is_pressed():
        moveBrick(1, 0)

    # Every 15 frames try to move the brick down
    if frameCount == 15 and moveBrick(0, 1) == False:
        frameCount = 0
        # The move was not possible, the brick stays in position
        grid[y][x] = max(brick[0][0], grid[y][x])
        grid[y][x + 1] = max(brick[0][1], grid[y][x + 1])
        grid[y + 1][x] = max(brick[1][0], grid[y + 1][x])
        grid[y + 1][x + 1] = max(brick[1][1], grid[y + 1][x + 1])

        if checkLines() == False and y == 0:
            # The brick has reached the top of the grid - Game Over
            game_over()
        else:
```

```
# Select a new brick randomly
            x = 3
            y = 0
            brick = choice(bricks)
            showBrick()

    if frameCount == 15:
        frameCount = 0

# Main Program Loop
while not gameOver:
    update_game()

# End of Game
sleep(2000)
display.scroll("Game Over: Score: " + str(score))
```

THE HEX CODE IS ATTACHED WITH THE FILE