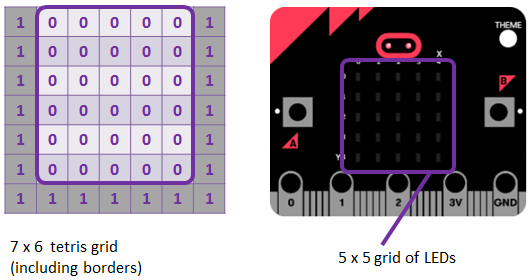
Tetris

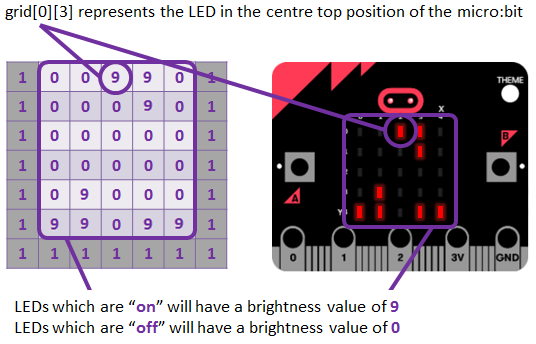
The game will use the LED screen which consists of a 5×5 grid of 25 LEDs.

Each LED can be on (value: 9 for maximum brightness) or Off (value: 0)

The side/borders of the grid will note be displayed.

The Python code will use 2-dimension arrays (list of lists in Python) to store the main grid (7×5) and the current brick (2×2)





**0. Import `microbit` module and `random`**

from microbit import \*

from random import choice

**1. Let’s now create the (7 x 6) tetris grid including the borders**

grid=[[1,0,0,0,0,0,1], \

[1,0,0,0,0,0,1], \

[1,0,0,0,0,0,1], \

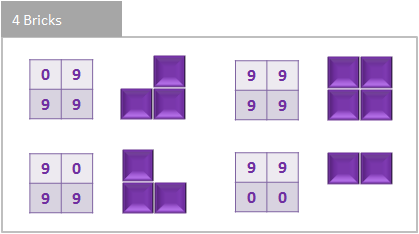
[1,0,0,0,0,0,1], \

[1,0,0,0,0,0,1], \

[1,1,1,1,1,1,1]]

**2. Store all the 4 bricks, each in a 2x2 grid**

bricks = [[9,9],[9,0]],[[9,9],[0,9]],[[9,9],[9,9]],[[9,9],[0,0]]



**3. Now, we need to select a brick randomly and position it at the centre/top of the grid (x = 3, y = 0). We set a frameCount to move the brick down. We will move a brick every 15 framecounts.**

brick = choice(bricks)

x=3

y=0

frameCount=0

**4. Define a function to hide the 2x2 brick on the LED screen. If the brick is visible we move the brick to the left or right of the LED screen so as to hide it.**

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])

**5. Define a function to return a maximum of 2 values.**

def max(a,b):

if a>=b:

return a

else:

return b

**6. Define a function to show the 2x2 grid on the screen, if the brick is hidden. It should be placed within the LED screen, so check against the maximum value before displaying the brick**

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]))

**7. Move or translate brick when possible and check for collisions**

def moveBrick(delta\_x,delta\_y):

global x,y

move=False

# Check if the move if 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

**8. Function to check for or remove completed lines**

def checkLines():

global score

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 (grid[i][1]+grid[i][2]+grid[i][3]+grid[i][4]+grid[i][5])==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

**9. Function to start Game by showing a brick**

gameOn=True

score=0

showBrick()

**9a. Go through the loop every half a second. If button a is pressed move brick to left by 1 and button b is pressed move brick to right by 1.**

#Main Program Loop - iterates every 50ms

while gameOn:

sleep(50)

frameCount+=1

#Capture user inputs

elif button\_a.is\_pressed():

moveBrick(-1,0)

elif button\_b.is\_pressed():

moveBrick(1,0)

**9b. Every 15 frames try to move the brick down. Check if it is possible to move the brick down, if not the brick stays in position.**

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

gameOn=False

else:

#select a new brick randomly

x=3

y=0

brick = choice(bricks)

showBrick()

if frameCount==15:

frameCount=0

**9c. At the end of game sleep for 2 seconds and display the score.**

#End of Game

sleep(2000)

display.scroll("Game Over: Score: " + str(score))

**10. Let’s now try to rotate 2x2 brick. We need to check if rotation is possible before doing so**

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()

**11. Add rotate brick command when a + b are pressed together. Add this to the main loop above button\_a.is\_pressed()**

#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)