Tetris with PyGame

# Game structure

**Define variables and objects**

**Game loops:** Checks for collision and object updates

# Setup

**Display surface :** Canvas

Import pygame 🡪 import the game

Pygame.init() 🡪 initialize pygame

Set windoe title name 🡪 pygame.set\_caption(“ “)

Setting display 🡪 pygame.display.set\_mode(*tuple\_width,height*)

We will use (x,y) to draw in this screen

Here origin is at top left



Origin

**Clock object:**

clock=pygame.time.Clock()

>>Tell show fast the game should run using *tick()* method

# Game Loop

1. Event Handling: check for event
2. Updating positions based on events
3. Drawing objects by using pygame graphics

To keep running till end we have while loop, that runs till the end and there will be loops inbetween too.

>> should also ensure the way to stop while loop

To get the events in the game we use **pygame.event.get()** 🡪 gives a list of events that can be recognized by pygame and in the list we check for QUIT event if present we break the while loop

Checking 🡪 event.type === pygame.QUIT

Update display 🡪 pygame.display.update()

Setting rate or speed of game 🡪 clock.tick(60) 🡪60 frames per second

>> if not set lead to inconsistencies -> code in while runs 60 times per second

# Drawing elements

Colors ->tuple of 3 in pygame ->(r,g,b) 🡪 0 to 255

Putting color for background ==🡺 screen.fill(\_\_*colot\_tuple\_\_*)

>>Here for tetris we should treat the play area as grids

Where the origin is the top left cell -> (0,0)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

Origin

## Code logic to place objects

Play space 🡺 2d grid / array

Empty space 🡺 0

Occupied by block 🡺 number of its color

Block that is descending or controlled by player 🡺 not reflected in grid

Here we use OOPs to organize our objects 🡪 created in separate files for better organizing

### Pygame rendering/drawing on screen

🡪Display Surface: where we see all the game object – 1 per game – blank canvas

🡪Surface: can have multiple surface – can draw on it

🡪Rect: Rectangle area for positioning , collision detection and to draw objects

**Steps:**

1. Create a Rect 🡪 invisible but helps draw on the screen 🡺 create only object not draws
   1. Drawn in each cell
   2. Takes x,y of top left of the obj, width and height
   3. Cell\_rect=pygame.Rect(col\*self.cell\_size,row\*self.cell\_size,self.cell\_size,self.cell\_size)
2. To draw the rect
   1. Needs 3 params 🡪 surface to draw the object, color and rect object
   2. Pygame.draw.rect(surface,color,rect)

### Class Grid

**Members:**  Number of rows , number of columns, size of each cell side, 2d array for grid, list of colors where the index will be the integer representing the color

7 shapes 🡪 7 colors

**Functions:**

Print the grid in console

To return the list of colors 🡪 to initialize the self.colors in init function

Draw 🡪draw each cell using pygame.draw.rect(screen,self.colors[cell\_value],cell\_rect) 🡪screen is received as argument

IsInside(row,col)🡪does boundary checking for the given row and col index

isEmpty(self,row,col)🡪 checks if the row and col grid position is empty (no block present) 🡪 used to check for valid key press movement

### Blocks

7 tetrominoes – can spin about center – but center may vary with the shape

To make rotation uniform, we imagine a square grid that contains the block in all 4 rotation state and the top 1st cell is origin

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Storing whole grid- more memory – store only the cells occupied by the block

So we use **state variable** to indicate the current **rotation state** of the block and modify the occupied cells according to it (no calculation – time consuming and overflow issues)

To create **different shapes of blocks we use inheritance 🡺** In Separate file

L block , J block I Block , O block, S block , T block, Z Block

### Class Block

**Members:**

Id to distinguish blocks – argument to init

Dictionary of cells to show occupied cells

Cell\_size

Row offset and column offset 🡪 showing the position of the block on the screen

Rotation state -> initially 0

Color of the block -> since it may need duplicating colors code (avoidable) we will create a class called colors and use it instead in grid as well as in blocks

### Class color

Here we declare the colors as the members and use **@classmethod** 🡺 python decorator that allows the method to be called on a class instead on an instance

Done as follows:

@classmethod

def get\_cell\_colors(cls):

cls🡪 reference to the class itself giving access to class level attributes and methods

In grid class we modify the colors initialization into

Self.colors=Colors.get\_cell\_colors()

>>remove the method in grid that returns colors list

**Methods:**

Draw 🡪draws the blocks and takes screen as argument and similarly creats Rect object and draws the rect

### Class Position

We need to store the position of each object which will be easier with class

To represent the position in 2d grid with single object with 2 attributes row and col

To make positioning easier , we create separate class for each block containing position for each rotation state that is inherited from the Block class

### Child Blocks:

Each will call the parent function with its own id and has a dictionary that has the cell position wrt to each rotation state --- for O block alone (square) the shape will be same for all the state

Move-> takes the movement in row and columns and updates the offsets

Get\_positions -> to modify the positions of the block tile by tile and returns the list of new position of each tile 🡺 used in the draw method to initialize tiles to be drawn

The blocks in tetris game appear from the middle of the grid (not from top left – origin) 🡺 modified in the init method

Rotate and undoRotate methods

## Game class

Becomes a container for all elements like grid , block etc

Better code organization and structure

Centralizes game functionality , maintaining, readability

This will be called in main.py instead of each individual element

**Members:**

grid, current block, next block, blocks(list)

**Methods:**

Get\_random\_blocks

Draw

Move\_left, Move\_right, move\_down

Rotate

IsInside-> sees if all tile of current block lies inside the grid space

Lock\_block🡪locks the block on reaching bottom and spawns new block

blockFits🡪Check if every cell of grid corresponding to the block is empty

## Controls

Requires detecting of clicks of keys 🡪 check done in game loop🡺check is event.type==pygame.KEYDOWN

To get the exact key pressed , we check by if event.key==pygame.K\_LEFT:

To keep the block moving down continuously we call the move\_down in the loop 🡺 but the block will fall fast for 60fps speed

So it needs delaying 🡺 Create a user defined event GAME\_UPDATE and Timer object created using timer that executes user defined event for each interval

We check for the GAME\_UPDATE event.type in the game loop and execute move\_down each time

🡺Need to lock the block once it reaches the bottom 🡪 need to change grid value and lock it and generate next block

🡺Game ends when new block overlaps on locked block 🡪 check if new block fits in the grid else the game is over – use a gameover variable (game class) to check it and update the game loop only if the variable is valid

## UI

Have to set font 🡪 created by pygame.font.Font(*FontFamily,size*)

To create a surface or a div/card kinda thing 🡪 title\_font.render(“Score”,True,white)🡺 white added to colors class

🡺We need to call block image transfer method inside game loop to display the game loop 🡪 screen.blit(score\_surface,(*xposition,yposition,height,width*))

Score system

100 pts – 1 line clear

300 pts – 2 line clear

500 pts – 3 line clear

1 pt for each block moved down by player