Institute for Advanced Studies
in Basic Sciences
Gava Zang, Zanjan, Iran

Name:

Parham Afsharnia

Student ID:

14024106

Professor Last Name:

Dr. Ansari – Dr Narimani

Course:

Advance Artificial Intelligence

Homework number:

5

Season: 1

Fall 2023

A block for each square

Nested for loop used for fill the board row by row with blocks that is from row 1 - (0,0), (0,1) ... (0, width)

last row - (height,0), ..., (height, width) in order. From left to right.

```
for x in range(self.width):
    for y in range(self.height):
            self.block_list[(x, y)] = Block(pos=Point(x, y), color='green')
```

this blocks added without changing anything else related to status.

Adding specific tiles

As the seed function gets list of coordinations, for each block, if block coordination is in the list, it sets to be live.

```
# coordinations involved
for block in self.block_list.values():
    if block.get_coords() in block_coords:
             block.set_live(self.canvas)
```

Finding out neighbors

First we get blocks coordination that we want its neighbor.

block_x, block_y = block.get_coords() # this is blocks coordination that we want its neighbor with this position we can reach to the neighbors by neighbors position dictionary below.

imagine blocks position between key 4 and 5. relatively, keys 1 to 8 are positional neighbors of the block as it shown in table 1. on a grid line surface

```
neighbors_pos = {1: Point(-1, -1), 2: Point(-1, 0), 3: Point(-1, 1),
                 4: Point(0, -1),     block here!    5: Point(0, 1),
                 6: Point(1, -1),  7: Point(1, 0),   8: Point(1, 1)}
```

1: Point(-1, -1)	2: Point(-1, 0)	3: Point(-1, 1)
4: Point(0, -1)	block	5: Point(0, 1)
6: Point(1, -1)	7: Point(1, 0)	8: Point(1, 1)

Table 1 neighbors position from a block

```
neighbors_list = [] # list of neighbors
```

by summing up blocks pos and positions in neighbor pos, we have all possible neighbors coordination but not all of the are valid (some of them are out of the board - for example let (0,0) be our block position, (-1,-1) is on possible neighbor for this block, but its not exist on the board

1: (-1, -1)	2: (-1, 0)	3: (-1, 1)
4: (0, -1)	Block (0,0)	5: (0, 1)
6: (1, -1)	7: (1, 0)	8: (1, 1)

Table 2 valid and invalid neighbors

This is true for not only corners but edges.

```
for pos in neighbors_pos.values()
# this condition check all possible neighbor and remove invalids
# if true -> ignore. else -> add to neighbors list
    if (BOARD_WIDTH - 1 < block_x + pos.x or block_x + pos.x < 0) or (
        BOARD_HEIGHT - 1 < block_y + pos.y or block_y + pos.y < 0):
            continue
    else:
     neighbors_list.append(self.block_list[(block_x + pos.x, block_y + pos.y)])
return neighbors_list
```

adding the rules

step 1

in this step for each block, after we got all its neighbors, we count neighbors state whether

It's alive or not then we apply rules on that block

rules are apply on new status, which is status of the block in next generation

```
for block in self.block_list.values():
            alives = 0
            neighbor_list = self.get_block_neighbors(block=block)
            for neighbor in neighbor_list:
                if neighbor.is_live():
                    alives += 1
            # rules applied
            if block.is_live():
                if alives < 2:
                    block.new_status = state[0]
                elif 2 == alives or alives == 3:
                    block.new_status = state[1]
                elif alives > 3:
                    block.new_status = state[0]
                else:
                    if alives == 3:
                        block.new_status = state[1]
```

step 2

reset blocks status. next generation status

```
for block in self.block_list.values():
        block.reset_status(self.canvas)
```

Done