personal - Tetris Game Python Code

create a downloadable tetris game for windows

Here’s a minimal Tetris you can run on Windows using Python + Pygame.

1. Install prerequisites

```bash
pip install pygame
```

2. Save this as `tetris.py`:

```python
import pygame, random

# config
cols, rows, block = 10, 20, 30
width, height = cols*block, rows*block
shapes = [
    [['.....',
      '.....',
      '..00.',
      '.00..',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']],
    [['.....',
      '.....',
      '.000.',
      '..0..',
      '.....'],
     ['..0..',
      '.00..',
      '..0..',
      '.....',
      '.....'],
     ['..0..',
      '.000.',
      '.....',
      '.....',
      '.....'],
     ['..0..',
      '..00.',
      '..0..',
      '.....',
      '.....']],
    [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']],
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y = x, y
        self.shape = shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (j,i), color in locked.items(): grid[j][i] = color
    return grid

def convert(p):  # piece → grid positions
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c=='0': positions.append((p.y+i-2, p.x+j-2))
    return positions

def valid(p, grid):
    for i,j in convert(p):
        if  i<0 or i>=rows or j<0 or j>=cols or grid[i][j]!=(0,0,0):
            return False
    return True

def clear_rows(grid, locked):
    inc = 0
    for i in range(rows-1, -1, -1):
        if (0,0,0) not in grid[i]:
            inc += 1
            for j in range(cols):
                locked.pop((i,j))
    if inc:
        for key in sorted(list(locked), key=lambda x: x[0])[::-1]:
            r, c = key
            if r < i:
                locked[(r+inc,c)] = locked.pop(key)
    return inc

def draw(window, grid):
    window.fill((0,0,0))
    for i in range(rows):
        for j in range(cols):
            pygame.draw.rect(window, grid[i][j],
                             (j*block, i*block, block, block), 0)
    for i in range(rows+1):
        pygame.draw.line(window, (50,50,50), (0,i*block),(width,i*block))
    for j in range(cols+1):
        pygame.draw.line(window, (50,50,50), (j*block,0),(j*block,height))
    pygame.display.update()

def main():
    locked = {}
    grid = create_grid(locked)
    change, run = False, True
    clock = pygame.time.Clock()
    fall_time, speed = 0, 0.5
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    pygame.init()
    win = pygame.display.set_mode((width, height))
    while run:
        dt = clock.tick()
        fall_time += dt/1000
        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                change = True
        for ev in pygame.event.get():
            if ev.type==pygame.QUIT: run=False
            if ev.type==pygame.KEYDOWN:
                if ev.key==pygame.K_LEFT:  current.x -=1
                if ev.key==pygame.K_RIGHT: current.x +=1
                if ev.key==pygame.K_DOWN:  current.y +=1
                if ev.key==pygame.K_UP:    current.rot +=1
                if not valid(current,grid):
                    current.x, current.y, current.rot = getattr(ev,'orig', (current.x, current.y, current.rot)) if False else (current.x, current.y, current.rot)
        shape_pos = convert(current)
        for i,j in shape_pos:
            if i>=0: grid[i][j] = current.color
        if change:
            for pos in shape_pos: locked[pos] = current.color
            current, next_p = next_p, Piece(cols//2,0,random.choice(shapes))
            change = False
            clear_rows(grid, locked)
        grid = create_grid(locked)
        draw(win, grid)
    pygame.quit()

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

3. Run it

```bash
python tetris.py
```

4. Package as a standalone Windows exe

```bash
pip install pyinstaller
pyinstaller --onefile tetris.py
```

- your `tetris.exe` will appear in `dist/` and is ready to distribute.

the blocks aren't visible while they're falling, only when they hit the bottom

The issue is that you’re rebuilding the grid **after** you draw the falling piece, so your overlay gets immediately wiped out. Move your `grid = create_grid(locked)` call to the top of the loop (and drop the one at the bottom). For example, in `main()`:

```diff

     while run:
-        dt = clock.tick()
-        fall_time += dt/1000
+        dt = clock.tick()
+        fall_time += dt/1000

+        # rebuild from locked before drawing piece
+        grid = create_grid(locked)
```

```
         if fall_time > speed:
             fall_time = 0
             current.y += 1
             if not valid(current, grid):
                 current.y -= 1
                 change = True

         for ev in pygame.event.get():
             …

-        shape_pos = convert(current)
-        for i,j in shape_pos:
-            if i>=0: grid[i][j] = current.color
+        # overlay falling piece onto fresh grid
+        shape_pos = convert(current)
+        for i,j in shape_pos:
+            if i>=0: grid[i][j] = current.color

         if change:
             for pos in shape_pos: locked[pos] = current.color
             current, next_p = next_p, Piece(cols//2,0,random.choice(shapes))
             change = False
             clear_rows(grid, locked)

-        grid = create_grid(locked)
         draw(win, grid)
```

Now your falling blocks will stay visible, since you rebuild the grid **before** you draw the active piece.

IndexError: list index out of range

Replace your piece-overlay loop with a bounds check on both row and column. For example, in `main()` do:

```diff
-        shape_pos = convert(current)
-        for i,j in shape_pos:
-            if i>=0: grid[i][j] = current.color
```

```
+        shape_pos = convert(current)
+        for i,j in shape_pos:
+            if 0 <= i < rows and 0 <= j < cols:
+                grid[i][j] = current.color
```

give me the whole code

```python
import pygame, random

# config
cols, rows, block = 10, 20, 30
width, height = cols * block, rows * block
shapes = [
    [['.....',
      '.....',
      '..00.',
      '.00..',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']],
    [['.....',
      '.....',
      '.000.',
      '..0..',
      '.....'],
     ['..0..',
      '.00..',
      '..0..',
      '.....',
      '.....'],
     ['..0..',
      '.000.',
      '.....',
      '.....',
      '.....'],
     ['..0..',
      '..00.',
      '..0..',
      '.....',
      '.....']],
    [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']],
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y, self.shape = x, y, shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (r,c), color in locked.items():
        grid[r][c] = color
    return grid
```

```
def convert(p):
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c == '0':
                positions.append((p.y + i - 2, p.x + j - 2))
    return positions

def valid(p, grid):
    for r, c in convert(p):
        if r < 0 or r >= rows or c < 0 or c >= cols or grid[r][c] != (0,0,0):
            return False
    return True

def clear_rows(grid, locked):
    cleared = 0
    for i in range(rows-1, -1, -1):
        if (0,0,0) not in grid[i]:
            cleared += 1
            for j in range(cols):
                locked.pop((i,j), None)
    if cleared:
        for (r,c) in sorted(list(locked), key=lambda x: x[0])[::-1]:
            if r < i:
                locked[(r+cleared, c)] = locked.pop((r,c))
    return cleared

def draw(win, grid):
    win.fill((0,0,0))
    for r in range(rows):
        for c in range(cols):
            pygame.draw.rect(win, grid[r][c],
                             (c*block, r*block, block, block), 0)
    for i in range(rows+1):
        pygame.draw.line(win, (50,50,50), (0, i*block), (width, i*block))
    for j in range(cols+1):
        pygame.draw.line(win, (50,50,50), (j*block, 0), (j*block, height))
    pygame.display.update()

def main():
    pygame.init()
    win = pygame.display.set_mode((width, height))
    clock = pygame.time.Clock()
    locked = {}
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    fall_time, speed = 0, 0.5
    run = True

    while run:
        grid = create_grid(locked)
        dt = clock.tick()
        fall_time += dt / 1000
        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                for r, c in convert(current):
                    if 0 <= r < rows and 0 <= c < cols:
                        locked[(r,c)] = current.color
                current, next_p = next_p, Piece(cols//2, 0, random.choice(shapes))
                clear_rows(grid, locked)

        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                run = False
            if ev.type == pygame.KEYDOWN:
                orig = (current.x, current.y, current.rot)
                if ev.key == pygame.K_LEFT:  current.x -= 1
                if ev.key == pygame.K_RIGHT: current.x += 1
                if ev.key == pygame.K_DOWN:  current.y += 1
                if ev.key == pygame.K_UP:    current.rot += 1
                if not valid(current, grid):
                    current.x, current.y, current.rot = orig

        for r, c in convert(current):
            if 0 <= r < rows and 0 <= c < cols:
                grid[r][c] = current.color

        draw(win, grid)
    pygame.quit()

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

fix this tetris program. the pieces sit and overlap at the top

Update your `valid` function so it ignores any block positions above the visible grid (r<0) instead of treating them as invalid. For example, replace:

```python
def valid(p, grid):
    for r, c in convert(p):
        if r < 0 or r >= rows or c < 0 or c >= cols or grid[r][c] != (0,0,0):
            return False
    return True
```

with:

```diff
 def valid(p, grid):
-    for r, c in convert(p):
-        if r < 0 or r >= rows or c < 0 or c >= cols or grid[r][c] != (0,0,0):
-            return False
+    for r, c in convert(p):
+        if c < 0 or c >= cols or r >= rows:    # out-of-bounds side/bottom
+            return False
+        if r >= 0 and grid[r][c] != (0,0,0):     # collide only inside grid
+            return False
     return True
```

This lets new pieces spawn partly above the top without immediately locking, so they'll fall into view instead of stacking at row 0.

give me the completed program script

Just swap in this `valid` function in your existing `tetris.py`—the rest of the script stays the same:

python

```
def valid(p, grid):
    for r, c in convert(p):
        if c < 0 or c >= cols or r >= rows:      # side- or bottom-bounds
            return False
        if r >= 0 and grid[r][c] != (0,0,0):      # only collide inside the grid
            return False
    return True
```

The completed visual line isn't processed until the next piece fully drops. Additionally, the removed line doesn't collapse. Give me the full fixed program.

python

```
import pygame, random

# config
cols, rows, block = 10, 20, 30
width, height = cols * block, rows * block
shapes = [
    [['.....',
      '.....',
      '..00.',
      '.00..',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']],
    [['.....',
      '.....',
      '.000.',
      '..0..',
      '.....'],
     ['..0..',
      '.00..',
      '..0..',
      '.....',
      '.....'],
     ['..0..',
      '.000.',
      '.....',
      '.....',
      '.....'],
     ['..0..',
      '..00.',
      '..0..',
      '.....',
      '.....']],
    [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']],
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y, self.shape = x, y, shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (r,c), color in locked.items():
        grid[r][c] = color
    return grid

def convert(p):
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c == '0':
                positions.append((p.y + i - 2, p.x + j - 2))
    return positions

def valid(p, grid):
    for r, c in convert(p):
        if c < 0 or c >= cols or r >= rows:
            return False
        if r >= 0 and grid[r][c] != (0,0,0):
            return False
    return True

def clear_rows(grid, locked):
    rows_to_clear = [i for i in range(rows) if (0,0,0) not in grid[i]]
    for r in rows_to_clear:
        for c in range(cols):
            locked.pop((r,c), None)
    if rows_to_clear:
        rows_to_clear.sort()
        new_locked = {}
        for (r,c), color in locked.items():
            shift = sum(1 for cleared in rows_to_clear if r < cleared)
            new_locked[(r+shift, c)] = color
        locked.clear()
        locked.update(new_locked)
    return len(rows_to_clear)

def draw(win, grid):
    win.fill((0,0,0))
    for r in range(rows):
        for c in range(cols):
            pygame.draw.rect(win, grid[r][c], (c*block, r*block, block, block), 0)
    for i in range(rows+1):
        pygame.draw.line(win, (50,50,50), (0, i*block), (width, i*block))
    for j in range(cols+1):
        pygame.draw.line(win, (50,50,50), (j*block, 0), (j*block, height))
    pygame.display.update()

def main():
    pygame.init()
    win = pygame.display.set_mode((width, height))
    clock = pygame.time.Clock()
    locked = {}
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    fall_time, speed = 0, 0.5
    run = True

    while run:
        grid = create_grid(locked)
        dt = clock.tick()
        fall_time += dt / 1000
        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                for r, c in convert(current):
                    if 0 <= r < rows and 0 <= c < cols:
                        locked[(r,c)] = current.color
                clear_rows(grid, locked)
                current, next_p = next_p, Piece(cols//2, 0, random.choice(shapes))

        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                run = False
            if ev.type == pygame.KEYDOWN:
                orig = (current.x, current.y, current.rot)
                if ev.key == pygame.K_LEFT:  current.x -= 1
                if ev.key == pygame.K_RIGHT: current.x += 1
                if ev.key == pygame.K_DOWN:  current.y += 1
                if ev.key == pygame.K_UP:    current.rot += 1
                if not valid(current, grid):
                    current.x, current.y, current.rot = orig

        for r, c in convert(current):
            if 0 <= r < rows and 0 <= c < cols:
                grid[r][c] = current.color

        draw(win, grid)

    pygame.quit()

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

The completed rows now collapse correctly however the collapse operation doesn't occur on the move when the row is completed but instead after the next piece drops

python

```python
import pygame, random

# config
cols, rows, block = 10, 20, 30
width, height = cols * block, rows * block
shapes = [
    [['.....',
      '.....',
      '..00.',
      '.00..',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']],
    [['.....',
      '.....',
      '.000.',
      '..0..',
      '.....'],
     ['..0..',
      '.00..',
      '..0..',
      '.....',
      '.....'],
     ['..0..',
      '.000.',
      '.....',
      '.....',
      '.....'],
     ['..0..',
      '..00.',
      '..0..',
      '.....',
      '.....']],
    [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']],
```

```python
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y, self.shape = x, y, shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (r,c), color in locked.items():
        grid[r][c] = color
    return grid

def convert(p):
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c == '0':
                positions.append((p.y + i - 2, p.x + j - 2))
    return positions

def valid(p, grid):
    for r, c in convert(p):
        if c < 0 or c >= cols or r >= rows:
            return False
        if r >= 0 and grid[r][c] != (0,0,0):
            return False
    return True

def clear_rows(grid, locked):
    rows_to_clear = [i for i in range(rows) if (0,0,0) not in grid[i]]
    for r in rows_to_clear:
        for c in range(cols):
            locked.pop((r,c), None)
    if rows_to_clear:
        rows_to_clear.sort()
        new_locked = {}
        for (r,c), color in locked.items():
            shift = sum(1 for cleared in rows_to_clear if r < cleared)
            new_locked[(r+shift, c)] = color
        locked.clear()
        locked.update(new_locked)
    return len(rows_to_clear)

def draw(win, grid):
    win.fill((0,0,0))
    for r in range(rows):
        for c in range(cols):
            pygame.draw.rect(win, grid[r][c], (c*block, r*block, block, block))
    for i in range(rows+1):
        pygame.draw.line(win, (50,50,50), (0, i*block), (width, i*block))
    for j in range(cols+1):
        pygame.draw.line(win, (50,50,50), (j*block, 0), (j*block, height))
    pygame.display.update()

def main():
    pygame.init()
    win = pygame.display.set_mode((width, height))
    clock = pygame.time.Clock()
    locked = {}
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    fall_time, speed = 0, 0.5
    run = True

    while run:
        grid = create_grid(locked)
        dt = clock.tick()
        fall_time += dt / 1000
        piece_locked = False

        # automatic drop
        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                piece_locked = True

        # input
        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                run = False
            if ev.type == pygame.KEYDOWN:
                orig = (current.x, current.y, current.rot)
                if ev.key == pygame.K_LEFT:
                    current.x -= 1
                    if not valid(current, grid):
                        current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_RIGHT:
                    current.x += 1
                    if not valid(current, grid):
                        current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_UP:
                    current.rot += 1
                    if not valid(current, grid):
                        current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_DOWN:
                    current.y += 1
                    if not valid(current, grid):
                        current.y -= 1
                        piece_locked = True

        # lock, clear, spawn
        if piece_locked:
            for r, c in convert(current):
                if 0 <= r < rows and 0 <= c < cols:
                    locked[(r,c)] = current.color
            clear_rows(grid, locked)
            current, next_p = next_p, Piece(cols//2, 0, random.choice(shapes))
            grid = create_grid(locked)
            piece_locked = False

        # draw current piece
        for r, c in convert(current):
            if 0 <= r < rows and 0 <= c < cols:
                grid[r][c] = current.color

        draw(win, grid)

    pygame.quit()
```

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

the first time I completed a row during game play it correctly collapsed and disappeared however the next complete row I got didn't collapsed until the following piece dropped.

```python
import pygame, random

# config
cols, rows, block = 10, 20, 30
width, height = cols * block, rows * block
shapes = [
    [['.....',
      '.....',
      '..00.',
      '.00..',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']],
    [['.....',
      '.....',
      '.000.',
      '..0..',
      '.....'],
     ['..0..',
      '.00..',
      '..0..',
      '.....',
      '.....'],
     ['..0..',
      '.000.',
      '.....',
      '.....',
      '.....'],
     ['..0..',
      '..00.',
      '..0..',
      '.....',
      '.....']],
    [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']],
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y, self.shape = x, y, shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (r,c), color in locked.items():
            grid[r][c] = color
    return grid

def convert(p):
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c == '0':
                positions.append((p.y + i - 2, p.x + j - 2))
    return positions

def valid(p, grid):
    for r, c in convert(p):
        if c < 0 or c >= cols or r >= rows:
            return False
        if r >= 0 and grid[r][c] != (0,0,0):
            return False
    return True

def clear_rows(grid, locked):
    rows_to_clear = [i for i in range(rows) if (0,0,0) not in grid[i]]
    if not rows_to_clear:
        return 0
    for r in rows_to_clear:
        for c in range(cols):
            locked.pop((r,c), None)
    rows_to_clear.sort()
    new_locked = {}
    for (r,c), color in locked.items():
        shift = sum(1 for cleared in rows_to_clear if r < cleared)
        new_locked[(r+shift, c)] = color
    locked.clear()
    locked.update(new_locked)
    return len(rows_to_clear)

def draw(win, grid):
    win.fill((0,0,0))
    for r in range(rows):
        for c in range(cols):
            pygame.draw.rect(win, grid[r][c], (c*block, r*block, block, block))
    for i in range(rows+1):
        pygame.draw.line(win, (50,50,50), (0, i*block), (width, i*block))
    for j in range(cols+1):
        pygame.draw.line(win, (50,50,50), (j*block, 0), (j*block, height))
    pygame.display.update()

def main():
    pygame.init()
    win = pygame.display.set_mode((width, height))
    clock = pygame.time.Clock()
    locked = {}
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    fall_time, speed = 0, 0.5
    run = True

    while run:
        grid = create_grid(locked)
        dt = clock.tick()
        fall_time += dt / 1000
        piece_locked = False

        # automatic drop
        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                piece_locked = True

        # input
        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                run = False
            if ev.type == pygame.KEYDOWN:
                orig = (current.x, current.y, current.rot)
                if ev.key == pygame.K_LEFT:
                    current.x -= 1
                    if not valid(current, grid):
                        current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_RIGHT:
                    current.x += 1
                    if not valid(current, grid):
```

```
                        current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_UP:
                    current.rot += 1
                    if not valid(current, grid):
                        current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_DOWN:
                    current.y += 1
                    if not valid(current, grid):
                        current.y -= 1
                        piece_locked = True

        # lock, clear, spawn
        if piece_locked:
            for r, c in convert(current):
                if 0 <= r < rows and 0 <= c < cols:
                    locked[(r,c)] = current.color
            grid = create_grid(locked)           # rebuild with new locks
            clear_rows(grid, locked)             # detect & clear now
            current, next_p = next_p, Piece(cols//2, 0, random.choice(shapes))

        # draw current piece
        for r, c in convert(current):
            if 0 <= r < rows and 0 <= c < cols:
                grid[r][c] = current.color

        draw(win, grid)

    pygame.quit()

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

the game doesn't detect end when a piece comes to rest and touches the top of the container

python

```
import pygame, random

# config
cols, rows, block = 10, 20, 30
```

```
width, height = cols * block, rows * block
shapes = [
    [['.....',
      '.....',
      '..00.',
      '.00..',
      '.....'],
     ['.....',
      '..0..',
      '..00.',
      '...0.',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '..00.',
      '.....'],
     ['.....',
      '..0..',
      '.00..',
      '.0...',
      '.....']],
    [['.....',
      '.....',
      '.000.',
      '..0..',
      '.....'],
     ['..0..',
      '.00..',
      '..0..',
      '.....',
      '.....'],
     ['..0..',
      '.000.',
      '.....',
      '.....',
      '.....'],
     ['..0..',
      '..00.',
      '..0..',
      '.....',
      '.....']],
    [['..0..',
      '..0..',
      '..0..',
      '..0..',
      '.....'],
     ['.....',
      '0000.',
      '.....',
      '.....',
      '.....']],
    [['.....',
      '.....',
      '.00..',
      '.00..',
      '.....']],
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y, self.shape = x, y, shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (r,c), color in locked.items():
        grid[r][c] = color
    return grid

def convert(p):
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c == '0':
                positions.append((p.y + i - 2, p.x + j - 2))
    return positions

def valid(p, grid):
    for r, c in convert(p):
        if c < 0 or c >= cols or r >= rows:
            return False
        if r >= 0 and grid[r][c] != (0,0,0):
            return False
    return True

def clear_rows(grid, locked):
    rows_to_clear = [i for i in range(rows) if (0,0,0) not in grid[i]]
    if not rows_to_clear:
        return 0
    for r in rows_to_clear:
        for c in range(cols):
            locked.pop((r,c), None)
    rows_to_clear.sort()
    new_locked = {}
    for (r,c), color in locked.items():
        shift = sum(1 for cleared in rows_to_clear if r < cleared)
        new_locked[(r+shift, c)] = color
    locked.clear()
    locked.update(new_locked)
    return len(rows_to_clear)

def draw(win, grid):
    win.fill((0,0,0))
    for r in range(rows):
        for c in range(cols):
            pygame.draw.rect(win, grid[r][c], (c*block, r*block, block, block))
    for i in range(rows+1):
        pygame.draw.line(win, (50,50,50), (0, i*block), (width, i*block))
    for j in range(cols+1):
        pygame.draw.line(win, (50,50,50), (j*block, 0), (j*block, height))
    pygame.display.update()

def main():
    pygame.init()
    win = pygame.display.set_mode((width, height))
    clock = pygame.time.Clock()
    locked = {}
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    fall_time, speed = 0, 0.5

    while True:
        grid = create_grid(locked)
        dt = clock.tick()
        fall_time += dt / 1000
        piece_locked = False

        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                piece_locked = True

        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                pygame.quit()
                return
            if ev.type == pygame.KEYDOWN:
                orig = (current.x, current.y, current.rot)
                if ev.key == pygame.K_LEFT:
                    current.x -= 1
                    if not valid(current, grid): current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_RIGHT:
                    current.x += 1
                    if not valid(current, grid): current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_UP:
                    current.rot += 1
                    if not valid(current, grid): current.x, current.y, current.rot = orig
                elif ev.key == pygame.K_DOWN:
                    current.y += 1
                    if not valid(current, grid):
                        current.y -= 1
                        piece_locked = True

        if piece_locked:
            for r, c in convert(current):
                if 0 <= r < rows and 0 <= c < cols:
                    locked[(r,c)] = current.color
            grid = create_grid(locked)
            clear_rows(grid, locked)
            current, next_p = next_p, Piece(cols//2, 0, random.choice(shapes))
            if not valid(current, grid):
                break

        for r, c in convert(current):
            if 0 <= r < rows and 0 <= c < cols:
                grid[r][c] = current.color

        draw(win, grid)

    pygame.quit()
    print("Game Over")

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

the game works perfectly however the window closes immediately when the game ends. instead, the game should display a colourful "game over" message and the user should be given the option to start a new game with any keystroke

```python
import pygame, random, sys

# config
cols, rows, block = 10, 20, 30
width, height = cols * block, rows * block
shapes = [
    [['.....','.....','..00.','.00..','.....'],
     ['.....','..0..','..00.','...0.','.....']],
    [['.....','.....','.00..','..00.','.....'],
     ['.....','..0..','.00..','.0...','.....']],
    [['.....','.....','.000.','..0..','.....'],
     ['..0..','.00..','..0..','.....','.....'],
     ['..0..','.000.','.....','.....','.....'],
     ['..0..','..00.','..0..','.....','.....']],
    [['..0..','..0..','..0..','..0..','.....'],
     ['.....','0000.','.....','.....','.....']],
    [['.....','.....','.00..','.00..','.....']],
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y, self.shape = x, y, shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (r,c), color in locked.items():
        grid[r][c] = color
    return grid

def convert(p):
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c == '0':
                positions.append((p.y + i - 2, p.x + j - 2))
    return positions

def valid(p, grid):
    for r, c in convert(p):
        if c < 0 or c >= cols or r >= rows: return False
        if r >= 0 and grid[r][c] != (0,0,0): return False
    return True

def clear_rows(grid, locked):
    rows_to_clear = [i for i in range(rows) if (0,0,0) not in grid[i]]
    if not rows_to_clear: return 0
    for r in rows_to_clear:
        for c in range(cols):
            locked.pop((r,c), None)
    rows_to_clear.sort()
    new_locked = {}
    for (r,c), color in locked.items():
        shift = sum(1 for cleared in rows_to_clear if r < cleared)
        new_locked[(r+shift, c)] = color
    locked.clear(); locked.update(new_locked)
    return len(rows_to_clear)

def draw(win, grid):
    win.fill((0,0,0))
    for r in range(rows):
        for c in range(cols):
            pygame.draw.rect(win, grid[r][c], (c*block, r*block, block, block))
    for i in range(rows+1):
        pygame.draw.line(win, (50,50,50), (0, i*block), (width, i*block))
    for j in range(cols+1):
        pygame.draw.line(win, (50,50,50), (j*block, 0), (j*block, height))
    pygame.display.update()

def run_game(win):
    locked = {}
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    fall_time, speed = 0, 0.5
    clock = pygame.time.Clock()
    while True:
        grid = create_grid(locked)
        fall_time += clock.tick() / 1000

        piece_locked = False
        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                piece_locked = True

        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                pygame.quit(); sys.exit()
            if ev.type == pygame.KEYDOWN:
                orig = (current.x, current.y, current.rot)
                if ev.key == pygame.K_LEFT:  current.x -= 1
                if ev.key == pygame.K_RIGHT: current.x += 1
                if ev.key == pygame.K_UP:    current.rot += 1
                if ev.key == pygame.K_DOWN:
                    current.y += 1
                    if not valid(current, grid):
                        current.y -= 1
                        piece_locked = True
                if not valid(current, grid):
                    current.x, current.y, current.rot = orig

        if piece_locked:
            for r,c in convert(current):
                if 0 <= r < rows and 0 <= c < cols:
                    locked[(r,c)] = current.color
            grid = create_grid(locked)
            clear_rows(grid, locked)
            current, next_p = next_p, Piece(cols//2, 0, random.choice(shapes))
            if not valid(current, grid):
                return  # game over

        for r,c in convert(current):
            if 0 <= r < rows and 0 <= c < cols:
                grid[r][c] = current.color

        draw(win, grid)

def game_over_screen(win):
    font = pygame.font.SysFont(None, 72)
    msg = font.render('GAME OVER', True, random.choice(colors))
    rect = msg.get_rect(center=(width//2, height//2 - 30))
    info = pygame.font.SysFont(None, 36).render('Press any key to restart', True,
(255,255,255))
    info_rect = info.get_rect(center=(width//2, height//2 + 30))
    win.fill((0,0,0))
    win.blit(msg, rect); win.blit(info, info_rect)
    pygame.display.update()
    while True:
        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                pygame.quit(); sys.exit()
            if ev.type == pygame.KEYDOWN:
                return

def main():
    pygame.init()
    win = pygame.display.set_mode((width, height))
    pygame.display.set_caption('Tetris')
    while True:
        run_game(win)
        game_over_screen(win)

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

Everything works perfectly, keep the game logic the same but allow space bar to drop the piece immediately

Just add this inside your `for ev in pygame.event.get():` KEYDOWN block in `run_game`:

```diff

             if ev.type == pygame.KEYDOWN:
+                if ev.key == pygame.K_SPACE:    # hard drop
+                    while True:
+                        current.y += 1
+                        if not valid(current, grid):
+                            current.y -= 1
+                            piece_locked = True
+                            break
                 orig = (current.x, current.y, current.rot)
                 if ev.key == pygame.K_LEFT:  current.x -= 1
                 if ev.key == pygame.K_RIGHT: current.x += 1
                 if ev.key == pygame.K_UP:    current.rot += 1
                 if ev.key == pygame.K_DOWN:
                     current.y += 1
                     if not valid(current, grid):
                         current.y -= 1
                         piece_locked = True
```

That lets Space drop the piece all the way and lock it immediately.

give me the full program to copy & paste

Here’s the full program with space-bar hard drop:

```python

import pygame, random, sys

```

```python
# config
cols, rows, block = 10, 20, 30
width, height = cols * block, rows * block
shapes = [
    [['.....','.....','..00.','.00..','.....'],
     ['.....','..0..','..00.','...0.','.....']],
    [['.....','.....','.00..','..00.','.....'],
     ['.....','..0..','.00..','.0...','.....']],
    [['.....','.....','.000.','..0..','.....'],
     ['..0..','.00..','..0..','.....','.....'],
     ['..0..','.000.','.....','.....','.....'],
     ['..0..','..00.','..0..','.....','.....']],
    [['..0..','..0..','..0..','..0..','.....'],
     ['.....','0000.','.....','.....','.....']],
    [['.....','.....','.00..','.00..','.....']],
]
colors = [(0,255,255),(0,255,0),(255,0,0),(255,255,0),(128,0,128)]

class Piece:
    def __init__(self, x, y, shape):
        self.x, self.y, self.shape = x, y, shape
        self.color = random.choice(colors)
        self.rot = 0

def create_grid(locked):
    grid = [[(0,0,0) for _ in range(cols)] for _ in range(rows)]
    for (r,c), color in locked.items():
        grid[r][c] = color
    return grid

def convert(p):
    positions = []
    fmt = p.shape[p.rot % len(p.shape)]
    for i, line in enumerate(fmt):
        for j, c in enumerate(line):
            if c == '0':
                positions.append((p.y + i - 2, p.x + j - 2))
    return positions

def valid(p, grid):
    for r, c in convert(p):
        if c < 0 or c >= cols or r >= rows: return False
        if r >= 0 and grid[r][c] != (0,0,0): return False
    return True

def clear_rows(grid, locked):
    rows_to_clear = [i for i in range(rows) if (0,0,0) not in grid[i]]
    if not rows_to_clear: return 0
    for r in rows_to_clear:
        for c in range(cols):
            locked.pop((r,c), None)
    rows_to_clear.sort()
    new_locked = {}
    for (r,c), color in locked.items():
        shift = sum(1 for cleared in rows_to_clear if r < cleared)
        new_locked[(r+shift, c)] = color
    locked.clear(); locked.update(new_locked)
    return len(rows_to_clear)

def draw(win, grid):
    win.fill((0,0,0))
    for r in range(rows):
        for c in range(cols):
            pygame.draw.rect(win, grid[r][c], (c*block, r*block, block, block))
    for i in range(rows+1):
        pygame.draw.line(win, (50,50,50), (0, i*block), (width, i*block))
    for j in range(cols+1):
        pygame.draw.line(win, (50,50,50), (j*block, 0), (j*block, height))
    pygame.display.update()

def run_game(win):
    locked = {}
    current = Piece(cols//2, 0, random.choice(shapes))
    next_p = Piece(cols//2, 0, random.choice(shapes))
    fall_time, speed = 0, 0.5
    clock = pygame.time.Clock()
    while True:
        grid = create_grid(locked)
        fall_time += clock.tick() / 1000
        piece_locked = False

        if fall_time > speed:
            fall_time = 0
            current.y += 1
            if not valid(current, grid):
                current.y -= 1
                piece_locked = True

        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                pygame.quit(); sys.exit()
            if ev.type == pygame.KEYDOWN:
                if ev.key == pygame.K_SPACE:
                    while True:
                        current.y += 1
                        if not valid(current, grid):
                            current.y -= 1
                            piece_locked = True
                            break
                    continue
                orig = (current.x, current.y, current.rot)
                if ev.key == pygame.K_LEFT:  current.x -= 1
                elif ev.key == pygame.K_RIGHT: current.x += 1
                elif ev.key == pygame.K_UP:    current.rot += 1
                elif ev.key == pygame.K_DOWN:
                    current.y += 1
                    if not valid(current, grid):
                        current.y -= 1
                        piece_locked = True
                if not valid(current, grid):
                    current.x, current.y, current.rot = orig

        if piece_locked:
            for r,c in convert(current):
                if 0 <= r < rows and 0 <= c < cols:
                    locked[(r,c)] = current.color
            grid = create_grid(locked)
            clear_rows(grid, locked)
            current, next_p = next_p, Piece(cols//2, 0, random.choice(shapes))
            if not valid(current, grid):
                return

        for r,c in convert(current):
            if 0 <= r < rows and 0 <= c < cols:
                grid[r][c] = current.color
```

```python
        draw(win, grid)

def game_over_screen(win):
    font = pygame.font.SysFont(None, 72)
    msg = font.render('GAME OVER', True, random.choice(colors))
    rect = msg.get_rect(center=(width//2, height//2 - 30))
    info = pygame.font.SysFont(None, 36).render('Press any key to restart', True, (255,255,255))
    info_rect = info.get_rect(center=(width//2, height//2 + 30))
    win.fill((0,0,0))
    win.blit(msg, rect); win.blit(info, info_rect)
    pygame.display.update()
    while True:
        for ev in pygame.event.get():
            if ev.type == pygame.QUIT:
                pygame.quit(); sys.exit()
            if ev.type == pygame.KEYDOWN:
                return

def main():
    pygame.init()
    win = pygame.display.set_mode((width, height))
    pygame.display.set_caption('Tetris')
    while True:
        run_game(win)
        game_over_screen(win)

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