ECE-C301 – Advanced Programming for Engineers

Contact

Dr. James Shackleford
shack@drexel.edu
Bossone 211

Office Hours: 3 – 4 pm (Wednesday)
Course Website: http://learn.dcollege.net

Textbook (1st half of course)

Think Python
by Allen Downey
O'Reilly Press, 2015
ISBN-13: 978-1449330729
(Freely available in PDF format, check course website)

Grading

(subject to change)

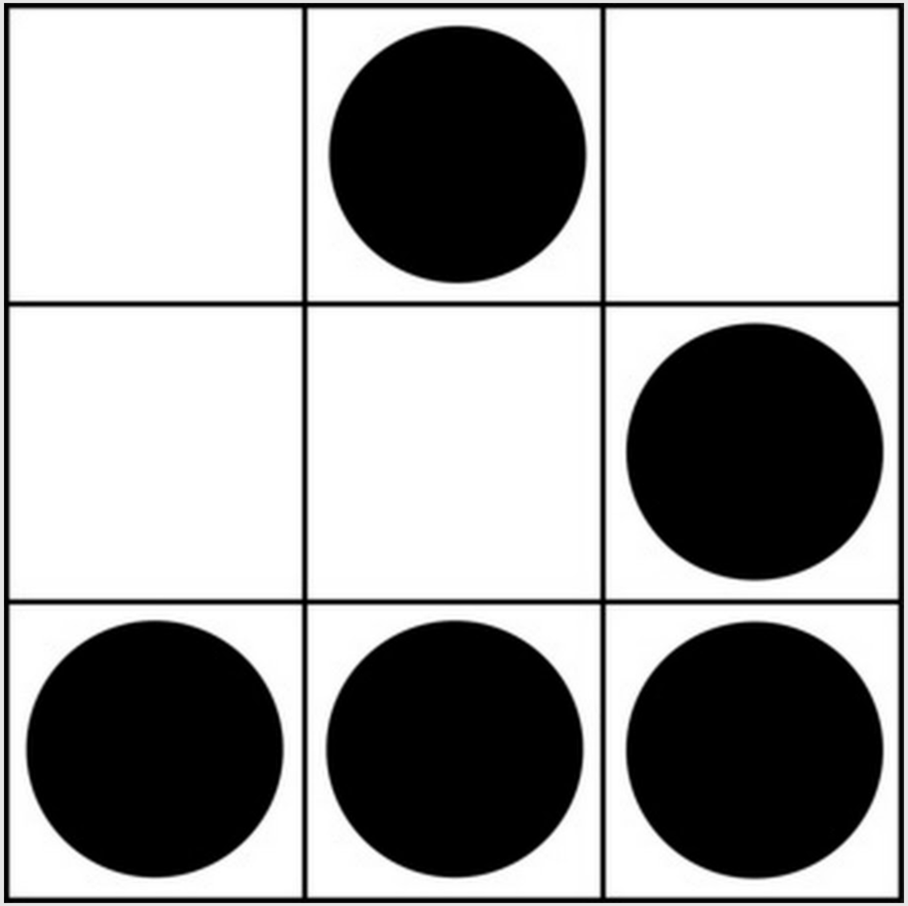
- 30% In-lab Programming Assignments
- 30% Take-Home Programming Assignments
- 40% Programming Projects

Course Structure

1st Half	Advanced Python
2nd Half	Introduction to C Used in: – Microcontrollers – Embedded Systems – System Programming

Conway's Game of Life

An introduction to programming simulations



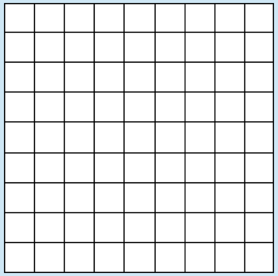
Conway's Game of Life

What is it?

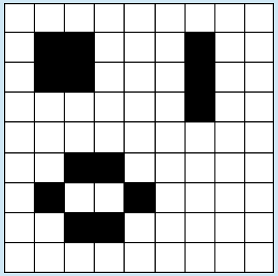
A cellular automata simulation.

Setup

Create the **World**
(a unifomly spaced grid)



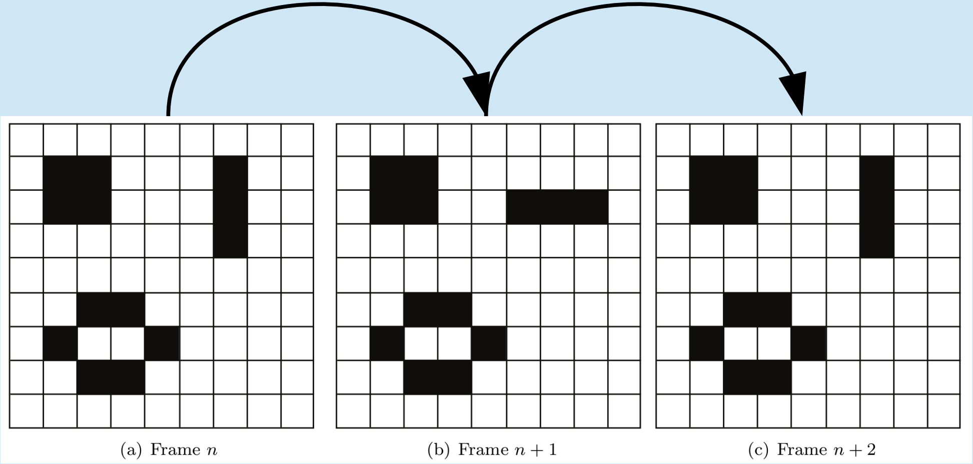
Initialize **World State**
– live cells (black)
– dead cells (white)



Run Simulation

Apply **Update Rule**
(over and over and over again)

update()   update()

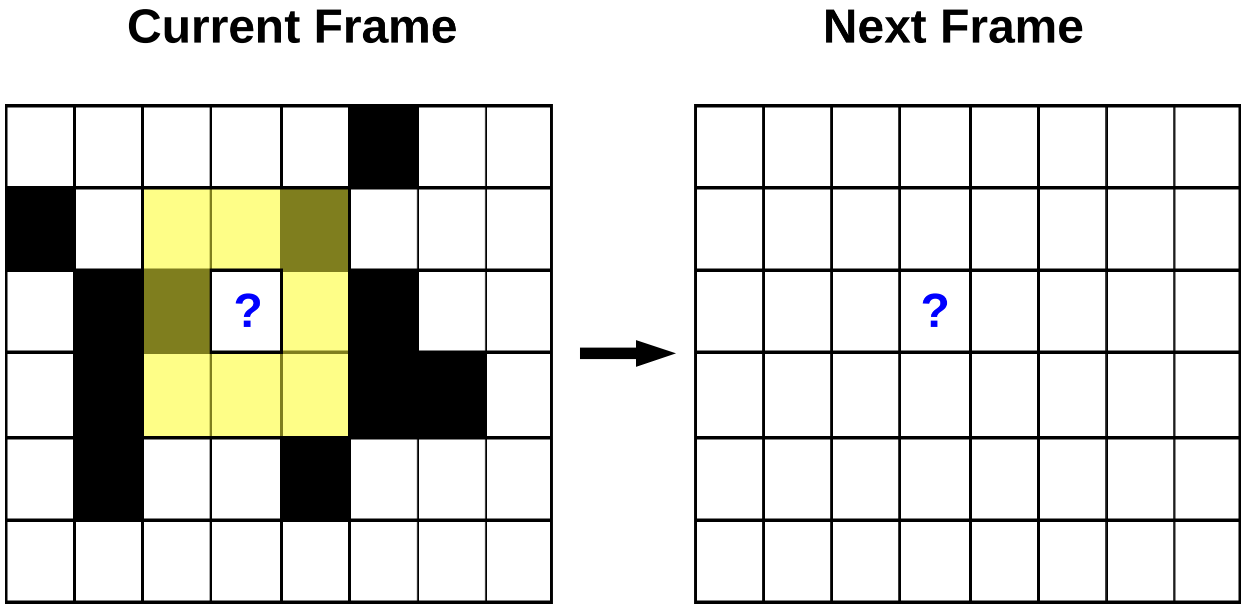


(a) Frame n (b) Frame $n+1$ (c) Frame $n+2$

Conway's Game of Life

Update Rule?

Determines if a **cell** in the **next frame** should be **black** or **white** based on the **state** of its **neighbors**



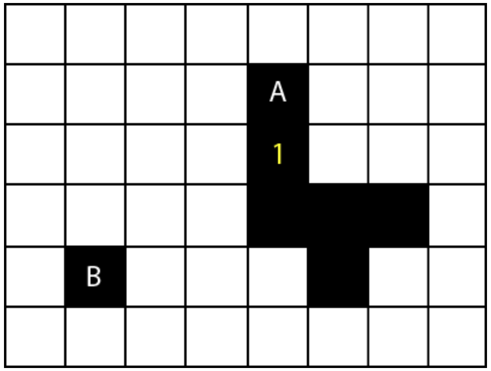
look at 8 neighbors in current frame

→ **determines state of cell in next frame** ←

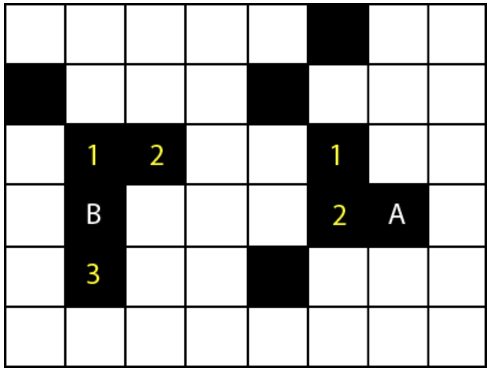
Update Rule?

Determines if a **cell** in the **next frame** should be **black** or **white** based on the **state** of its **neighbors**

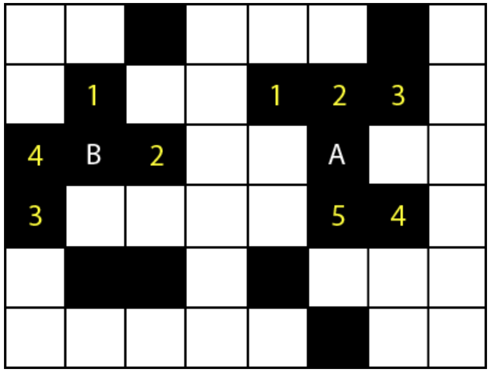
There are 4 cases



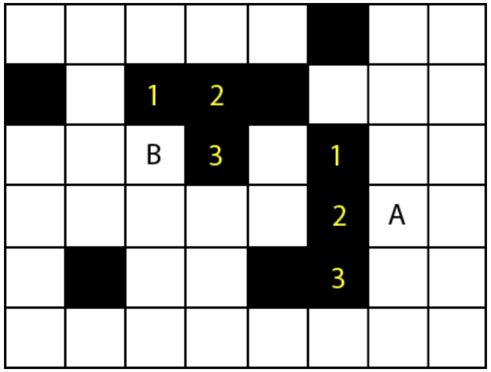
Under Population



Stable Population



Over Population



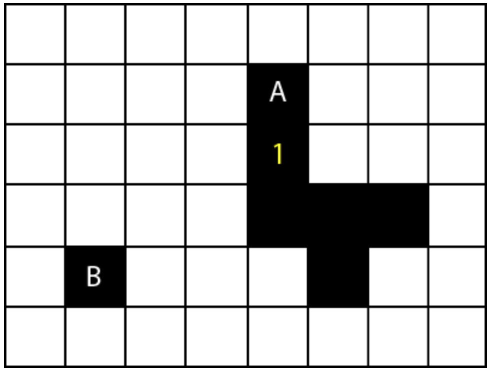
Reproduction

Conway's Game of Life

Update Rule?

Determines if a **cell** in the **next frame** should be **black** or **white** based on the **state** of its **neighbors**

There are 4 cases



Under Population

★ **Applies to living cells**

★ Dies if living neighbors < 2

– **Cell A**: Has 1 neighbor → **DEAD** in next frame

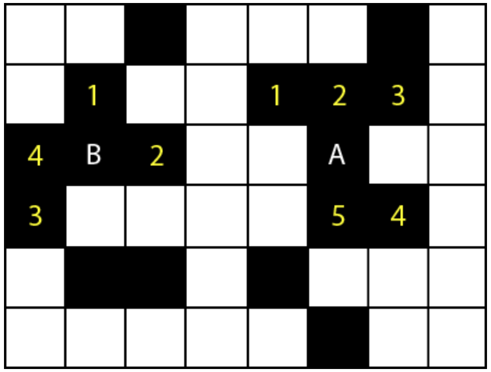
– **Cell B**: Has 0 neighbors → **DEAD** in next frame

Conway's Game of Life

Update Rule?

Determines if a **cell** in the **next frame** should be **black** or **white** based on the **state** of its **neighbors**

There are 4 cases



Over Population

★ **Applies to living cells**

★ Dies if living neighbors > 3

– **Cell A**: Has 5 neighbors → **DEAD** in next frame

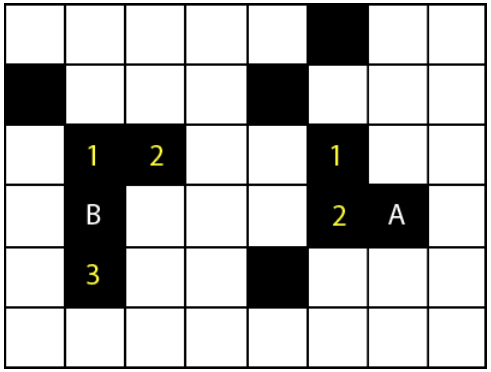
– **Cell B**: Has 4 neighbors → **DEAD** in next frame

Conway's Game of Life

Update Rule?

Determines if a **cell** in the **next frame** should be **black** or **white** based on the **state** of its **neighbors**

There are 4 cases



Stable Population

★ **Applies to living cells**

★ Lives if living neighbors
= 2 **or** = 3

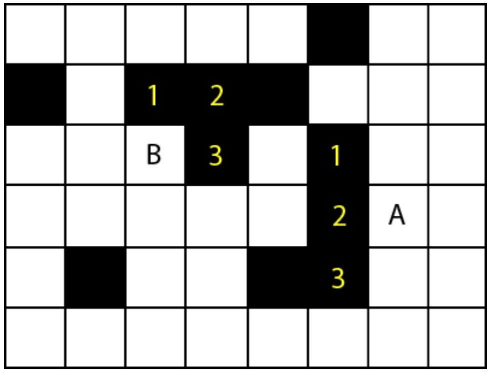
– **Cell A**: Has 2 neighbors → **ALIVE** in next frame

– **Cell B**: Has 3 neighbors → **ALIVE** in next frame

Update Rule?

Determines if a **cell** in the **next frame** should be **black** or **white** based on the **state** of its **neighbors**

There are 4 cases



Reproduction

★ **Applies to DEAD cells**

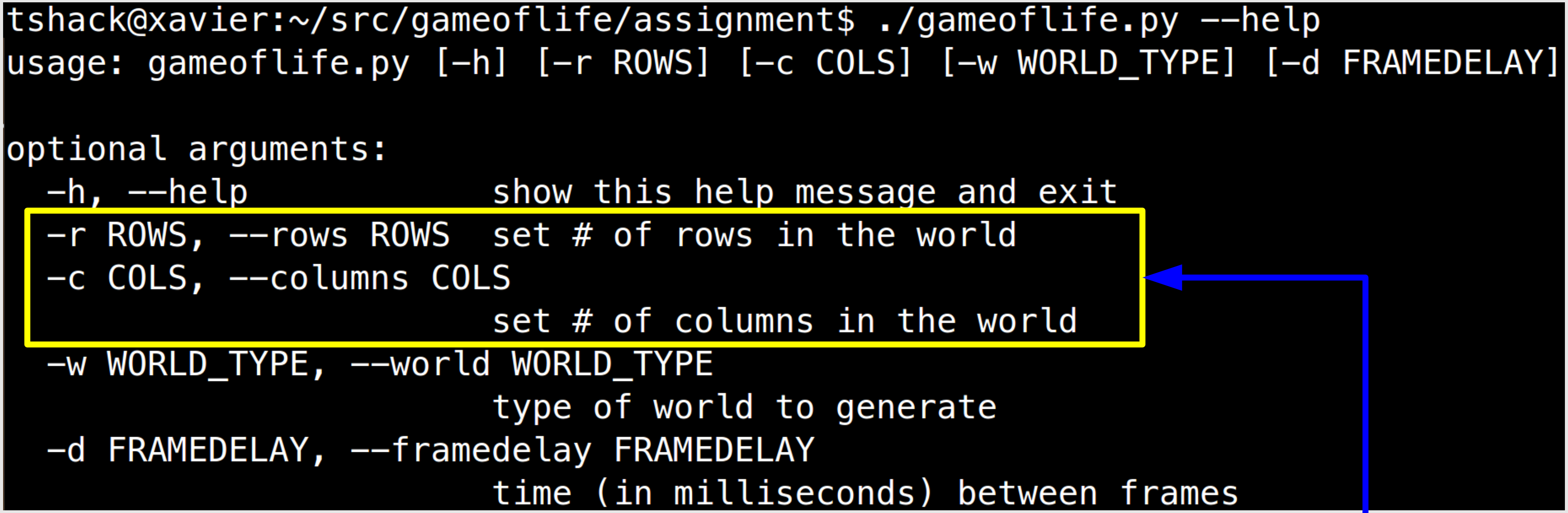
★ Lives if living neighbors EXACTLY = 3

– **Cell A**: Has 3 neighbors → **ALIVE** in next frame

– **Cell B**: Has 3 neighbors → **ALIVE** in next frame

ONLY WAY FOR A DEAD CELL TO COME BACK TO LIFE

Program Specifications

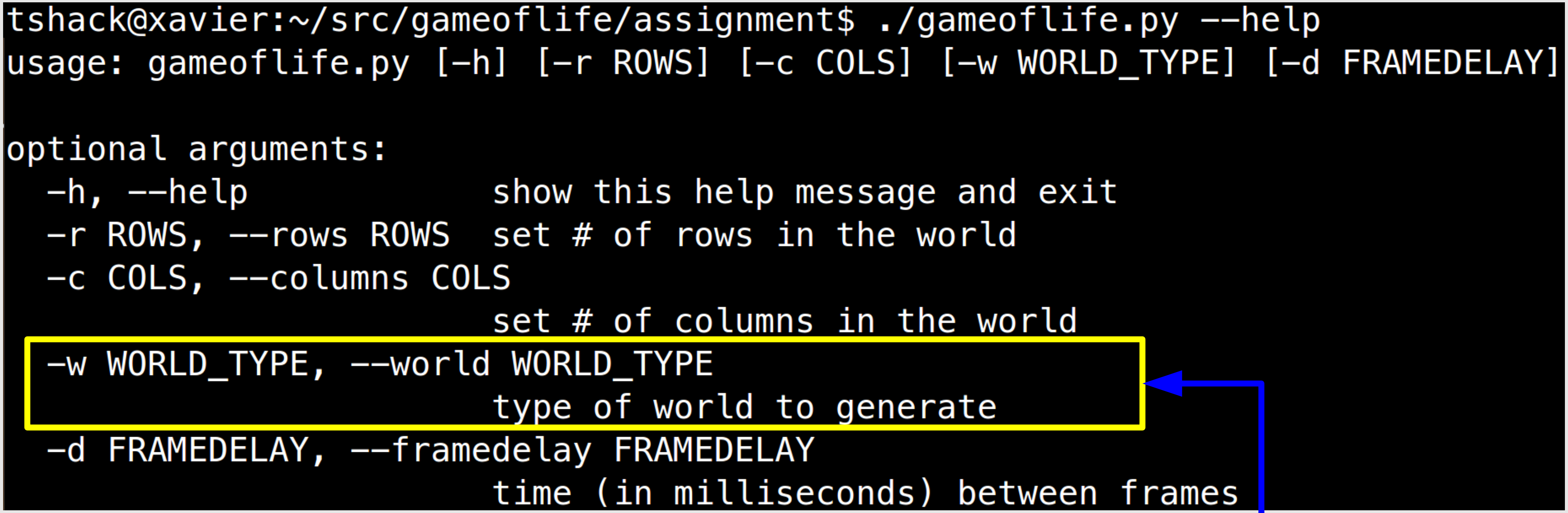


```
tshack@xavier:~/src/gameoflife/assignment$ ./gameoflife.py --help
usage: gameoflife.py [-h] [-r ROWS] [-c COLS] [-w WORLD_TYPE] [-d FRAMEDELAY]

optional arguments:
  -h, --help            show this help message and exit
  -r ROWS, --rows ROWS  set # of rows in the world
  -c COLS, --columns COLS
                        set # of columns in the world
  -w WORLD_TYPE, --world WORLD_TYPE
                        type of world to generate
  -d FRAMEDELAY, --framedelay FRAMEDELAY
                        time (in milliseconds) between frames
```

From command prompt:

★ can specify world size
★ can set world type
  – random (10% alive)
  – empty

Program Specifications



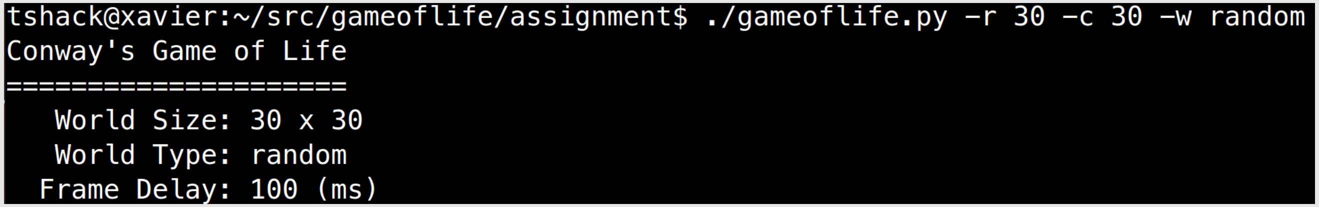
```
tshack@xavier:~/src/gameoflife/assignment$ ./gameoflife.py --help
usage: gameoflife.py [-h] [-r ROWS] [-c COLS] [-w WORLD_TYPE] [-d FRAMEDELAY]

optional arguments:
  -h, --help            show this help message and exit
  -r ROWS, --rows ROWS  set # of rows in the world
  -c COLS, --columns COLS
                        set # of columns in the world
  -w WORLD_TYPE, --world WORLD_TYPE
                        type of world to generate
  -d FRAMEDELAY, --framedelay FRAMEDELAY
                        time (in milliseconds) between frames
```

From command prompt:

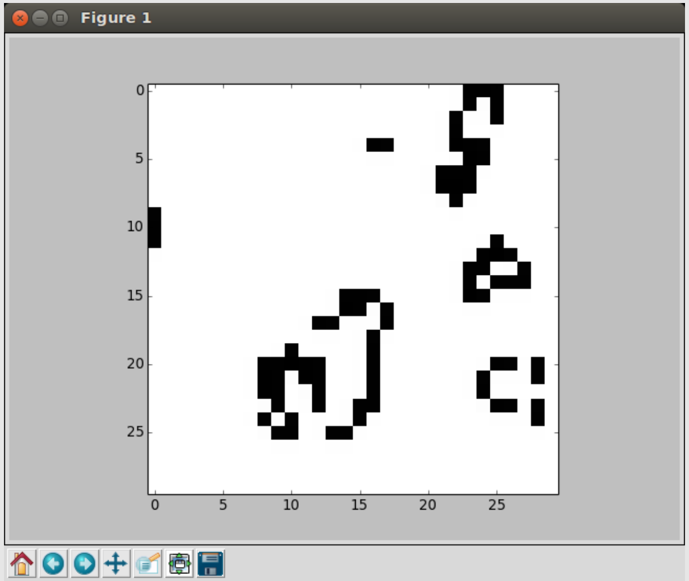
★ can specify world size
★ can set world type
  – random (10% alive)
  – empty

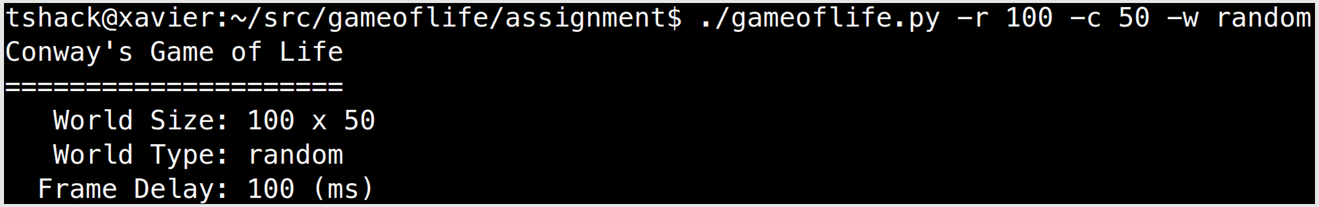
Conway's Game of Life

Program Specifications

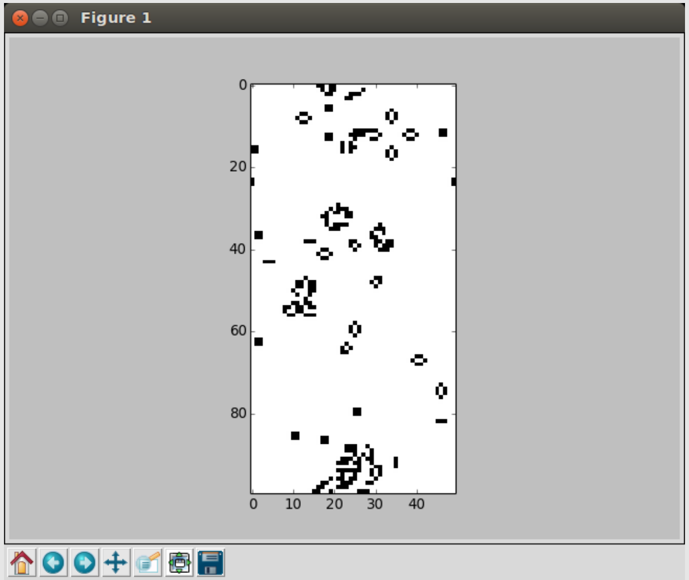


```
tshack@xavier:~/src/gameoflife/assignment$ ./gameoflife.py -r 30 -c 30 -w random
Conway's Game of Life
=====================
   World Size: 30 x 30
   World Type: random
  Frame Delay: 100 (ms)
```

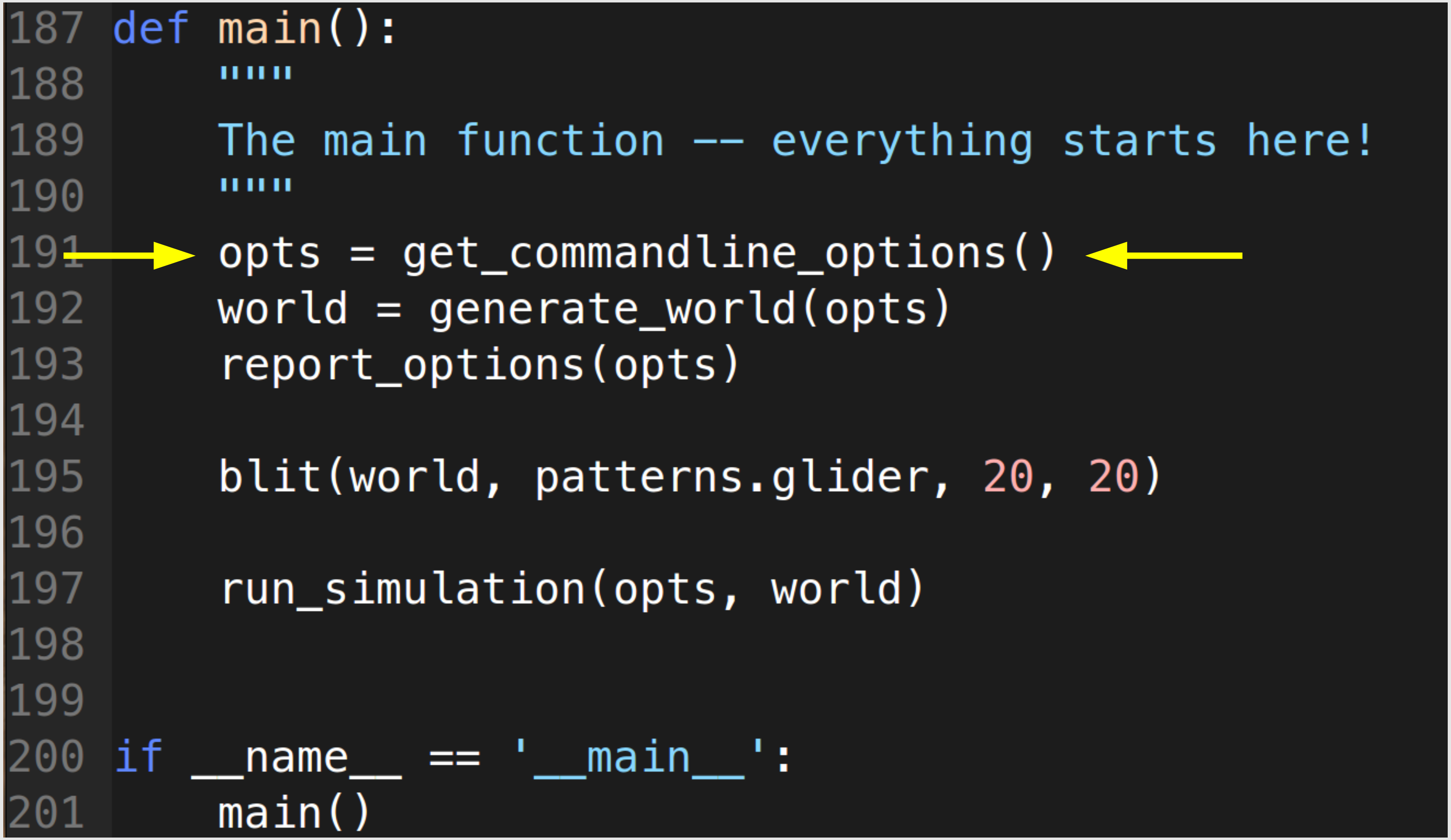




```
tshack@xavier:~/src/gameoflife/assignment$ ./gameoflife.py -r 100 -c 50 -w random
Conway's Game of Life
=====================
   World Size: 100 x 50
   World Type: random
  Frame Delay: 100 (ms)
```

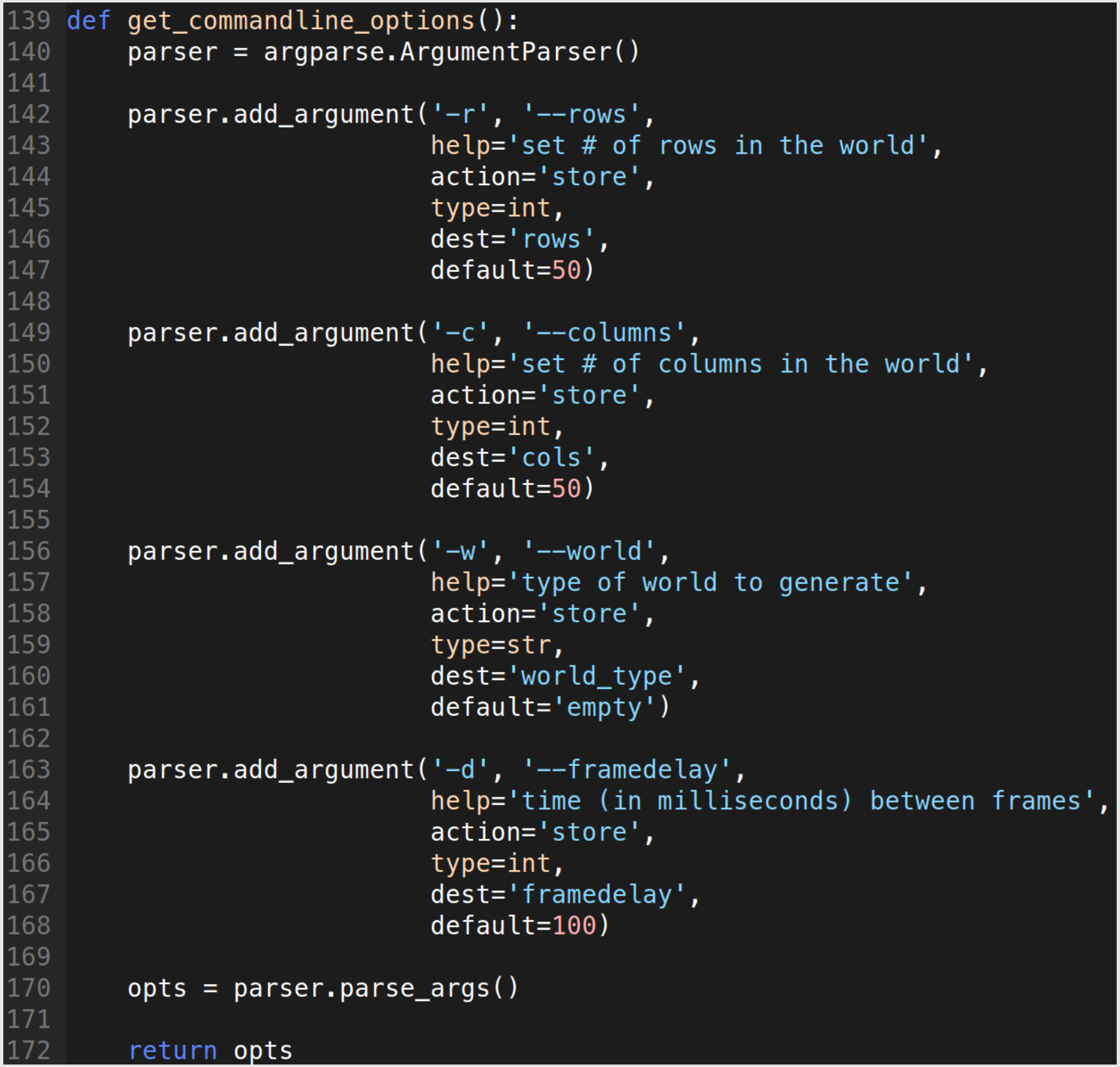


The Skeleton Code



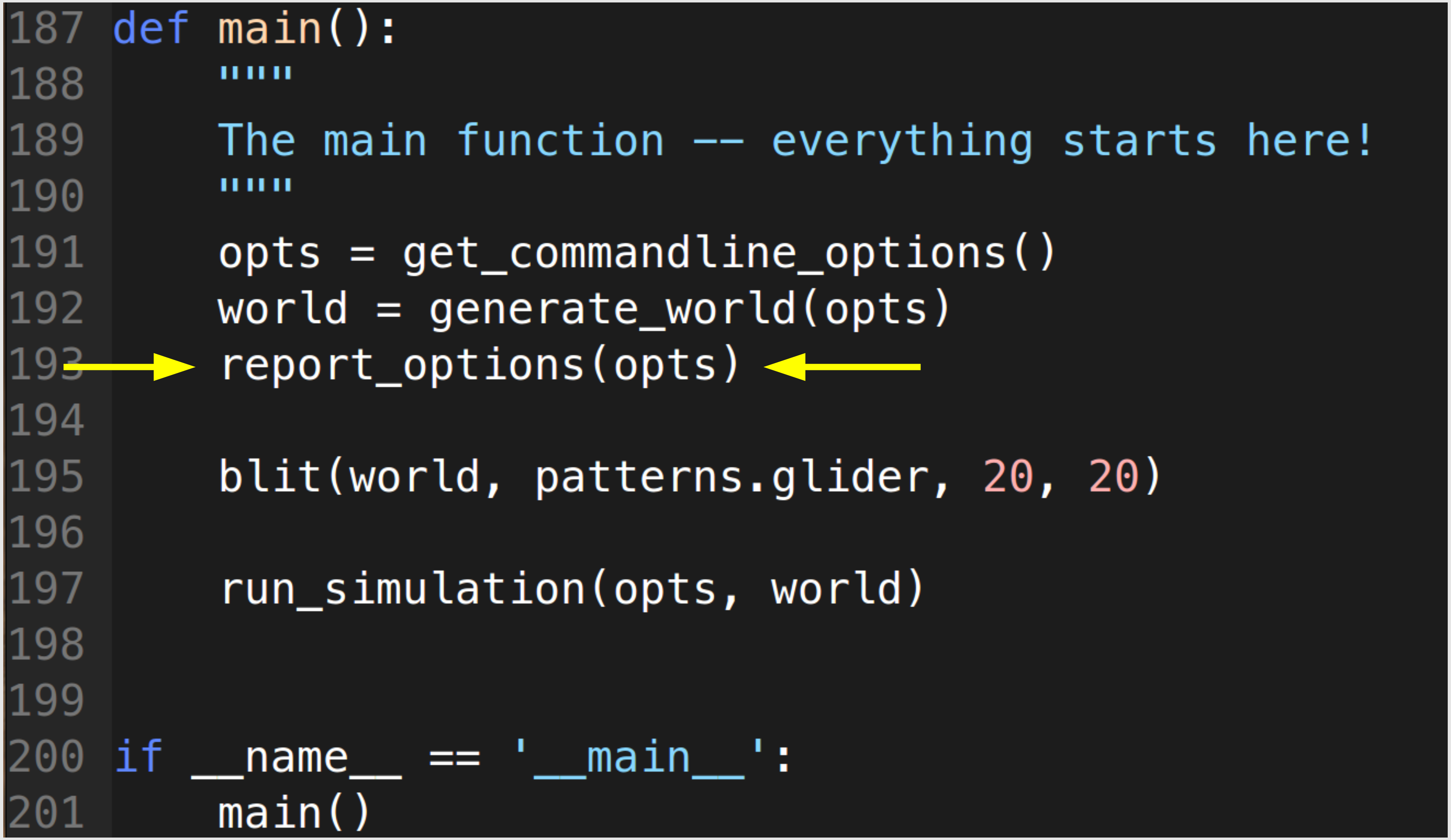
```python
def main():
    """
    The main function -- everything starts here!
    """
    opts = get_commandline_options()
    world = generate_world(opts)
    report_options(opts)

    blit(world, patterns.glider, 20, 20)

    run_simulation(opts, world)


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

The Skeleton Code



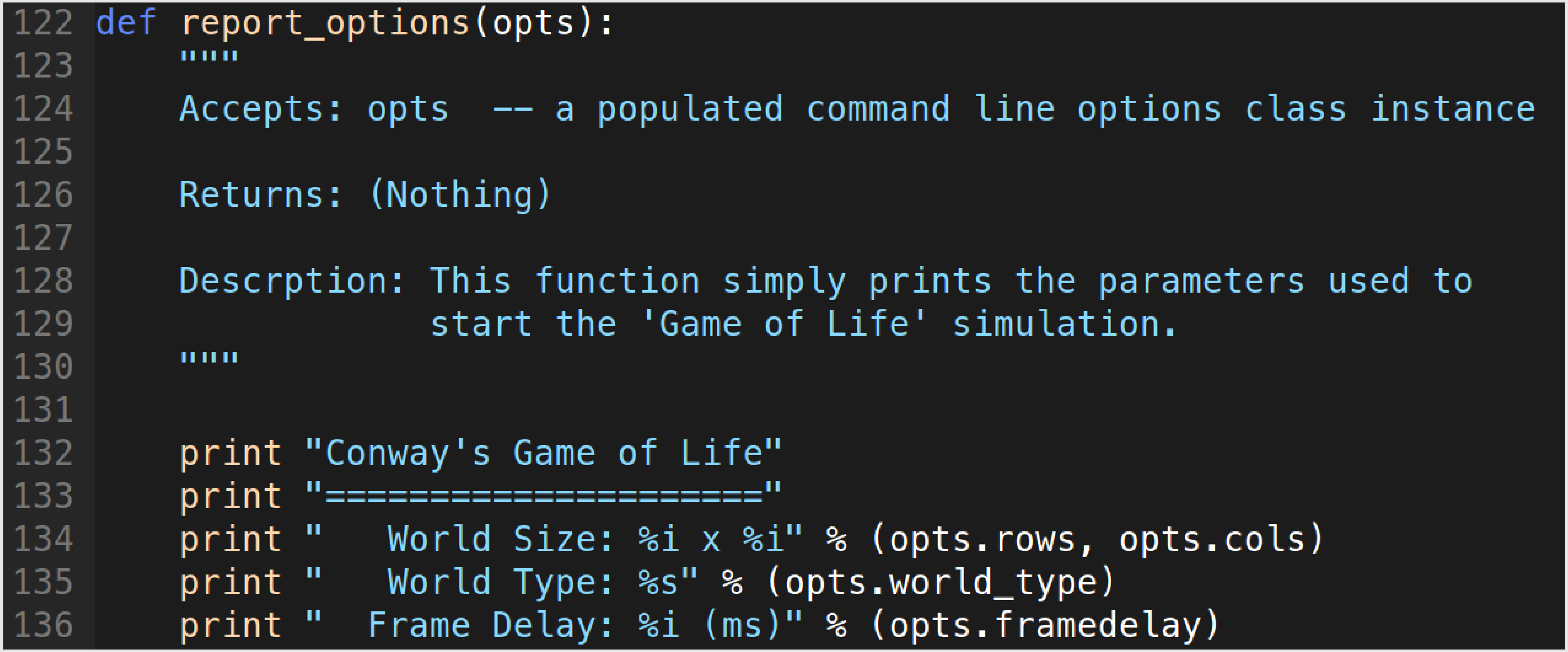
```python
def get_commandline_options():
    parser = argparse.ArgumentParser()

    parser.add_argument('-r', '--rows',
                        help='set # of rows in the world',
                        action='store',
                        type=int,
                        dest='rows',
                        default=50)

    parser.add_argument('-c', '--columns',
                        help='set # of columns in the world',
                        action='store',
                        type=int,
                        dest='cols',
                        default=50)

    parser.add_argument('-w', '--world',
                        help='type of world to generate',
                        action='store',
                        type=str,
                        dest='world_type',
                        default='empty')

    parser.add_argument('-d', '--framedelay',
                        help='time (in milliseconds) between frames',
                        action='store',
                        type=int,
                        dest='framedelay',
                        default=100)

    opts = parser.parse_args()

    return opts
```

The Skeleton Code



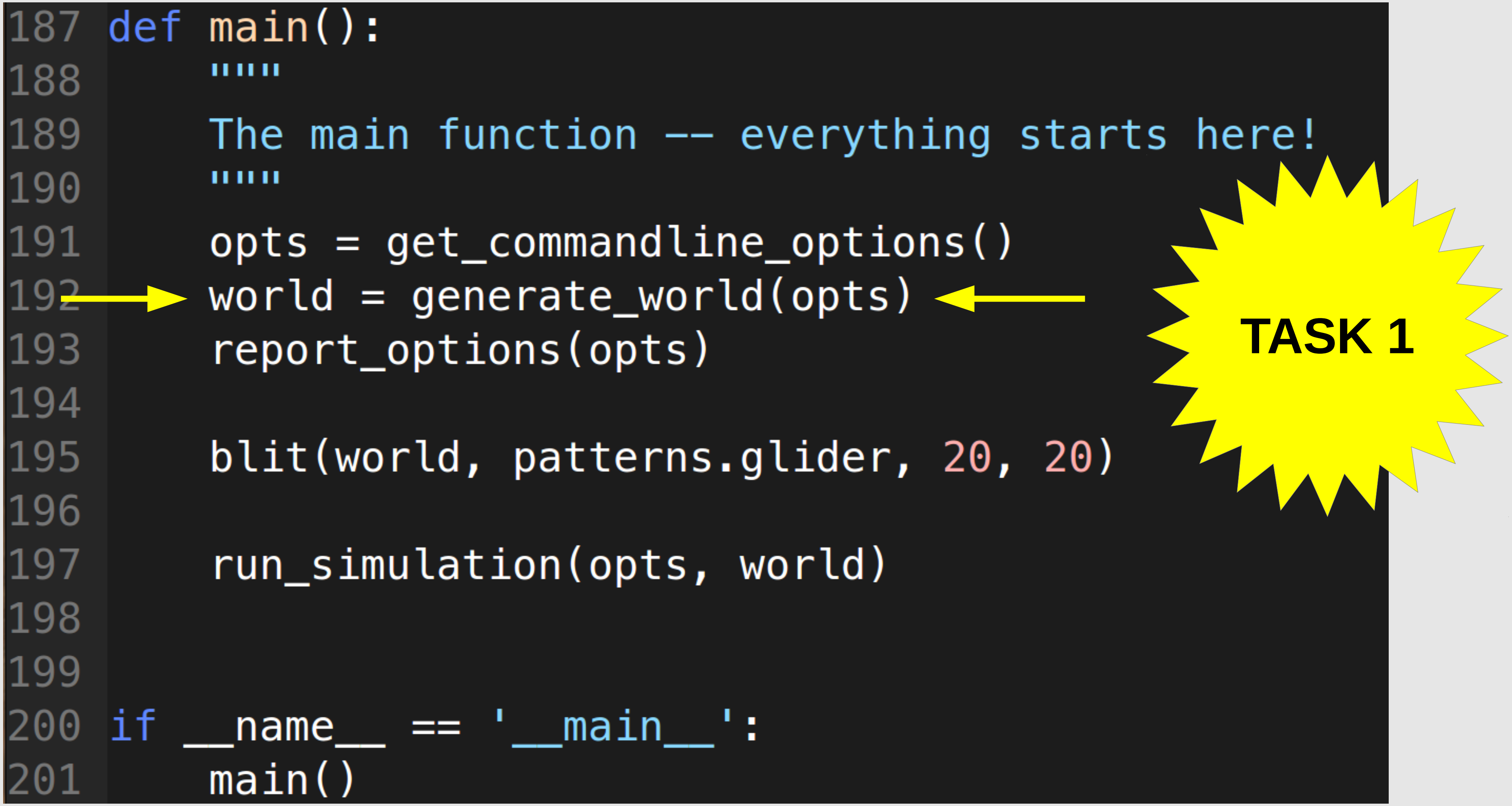
```python
def main():
    """
    The main function -- everything starts here!
    """
    opts = get_commandline_options()
    world = generate_world(opts)
    report_options(opts)

    blit(world, patterns.glider, 20, 20)

    run_simulation(opts, world)


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

The Skeleton Code



```python
def report_options(opts):
    """
    Accepts: opts  -- a populated command line options class instance

    Returns: (Nothing)

    Descrption: This function simply prints the parameters used to
                start the 'Game of Life' simulation.
    """

    print "Conway's Game of Life"
    print "====================="
    print "   World Size: %i x %i" % (opts.rows, opts.cols)
    print "   World Type: %s" % (opts.world_type)
    print "  Frame Delay: %i (ms)" % (opts.framedelay)
```

The Skeleton Code

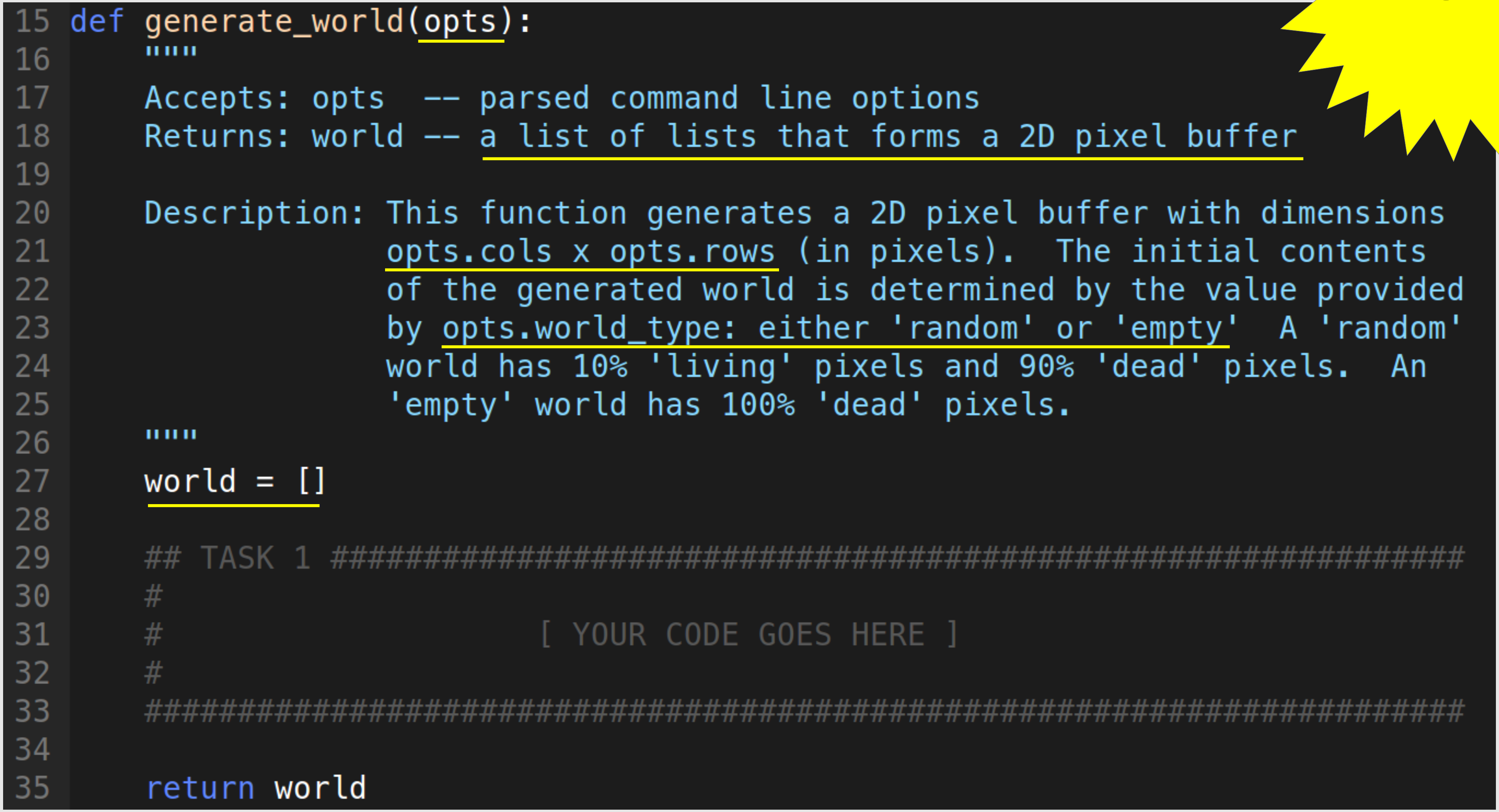


```python
def main():
    """
    The main function -- everything starts here!
    """
    opts = get_commandline_options()
    world = generate_world(opts)
    report_options(opts)

    blit(world, patterns.glider, 20, 20)

    run_simulation(opts, world)


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

TASK 1

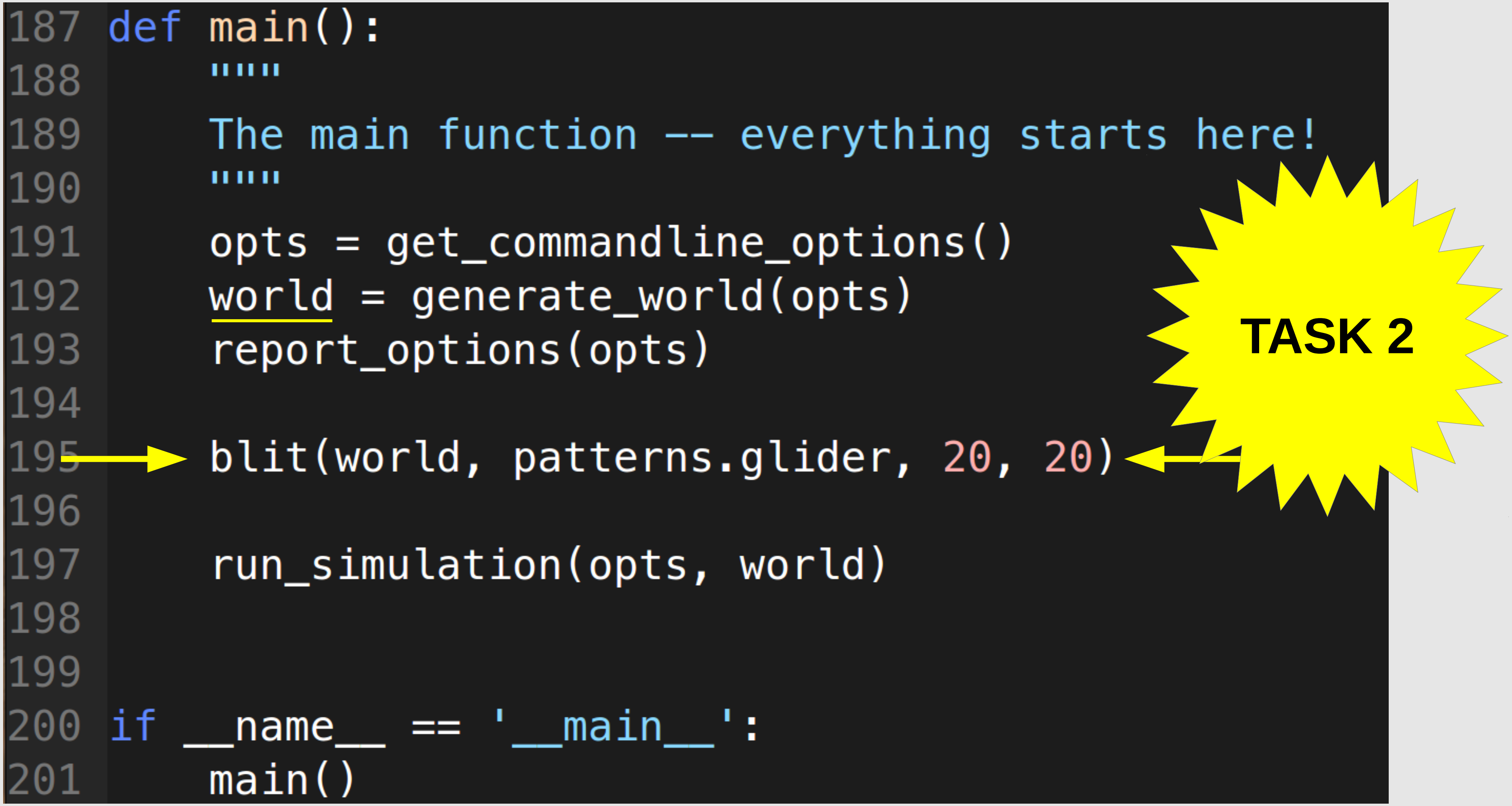
The Skeleton Code

TASK 1



```
def generate_world(opts):
    """
    Accepts: opts  -- parsed command line options
    Returns: world -- a list of lists that forms a 2D pixel buffer

    Description: This function generates a 2D pixel buffer with dimensions
                 opts.cols x opts.rows (in pixels).  The initial contents
                 of the generated world is determined by the value provided
                 by opts.world_type: either 'random' or 'empty'  A 'random'
                 world has 10% 'living' pixels and 90% 'dead' pixels.  An
                 'empty' world has 100% 'dead' pixels.
    """
    world = []

    ## TASK 1 #############################################################
    #
    #                    [ YOUR CODE GOES HERE ]
    #
    #######################################################################

    return world
```

The Skeleton Code

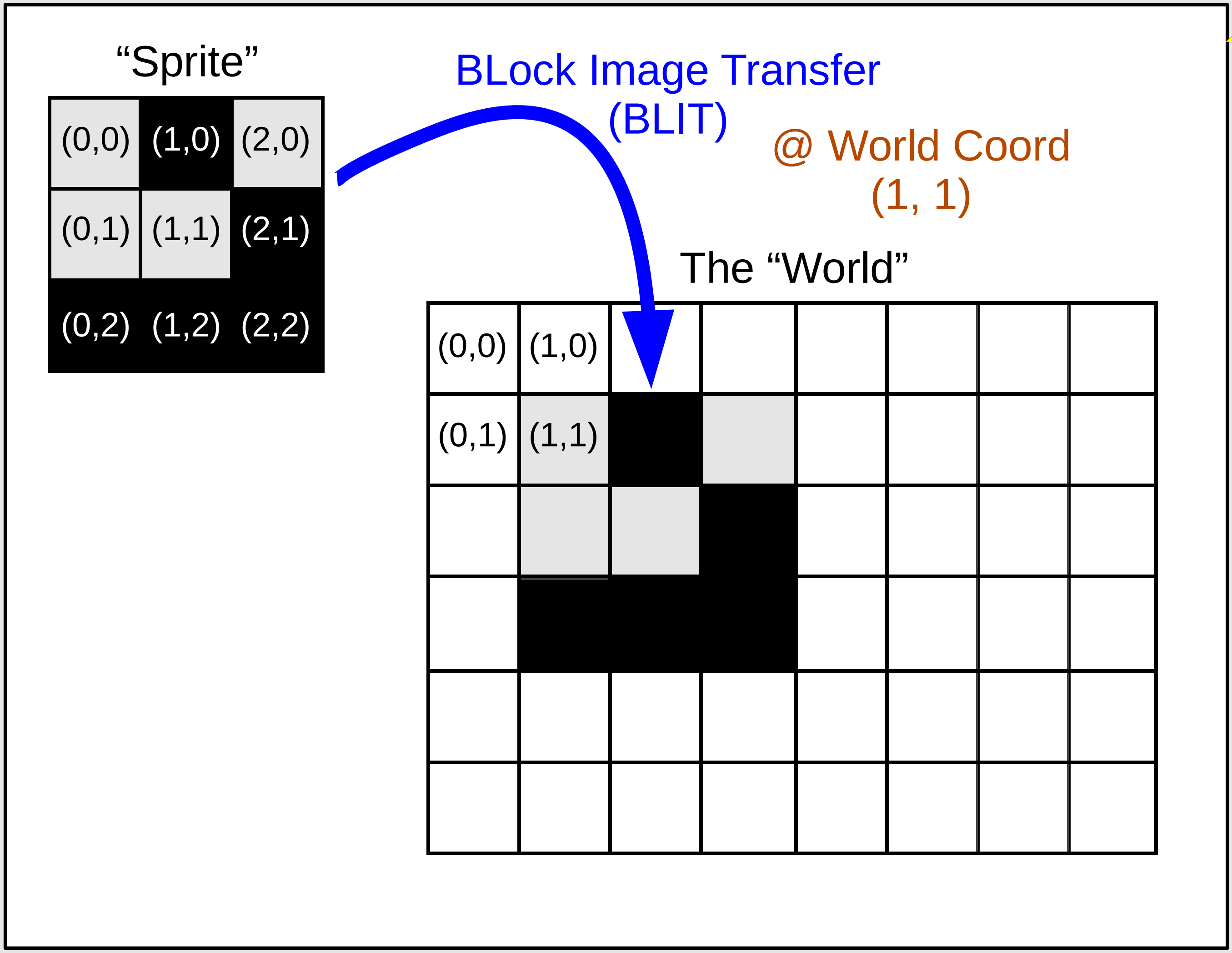


```
def main():
    """
    The main function -- everything starts here!
    """
    opts = get_commandline_options()
    world = generate_world(opts)
    report_options(opts)

    blit(world, patterns.glider, 20, 20)

    run_simulation(opts, world)


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

TASK 2

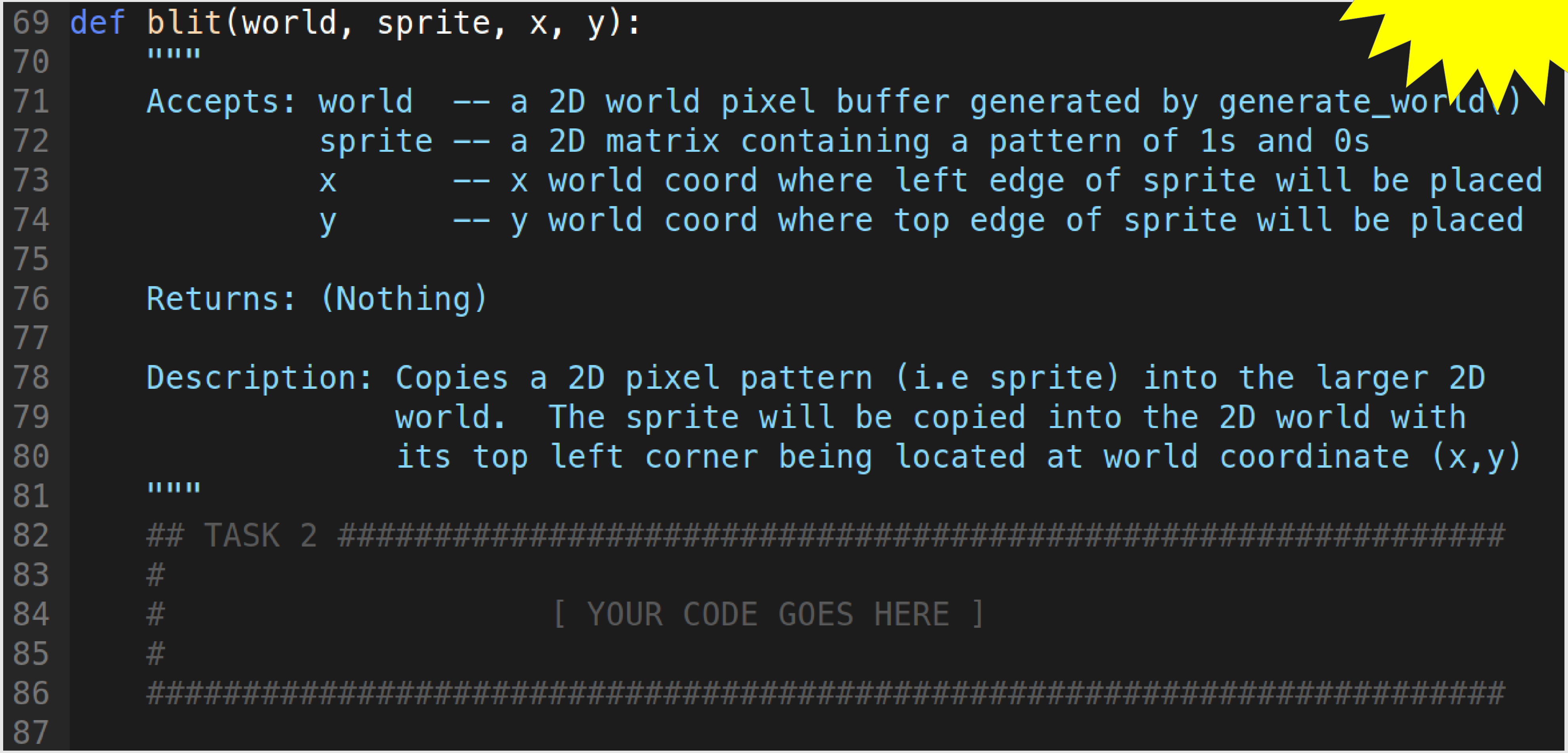
The Skeleton Code

TASK 2



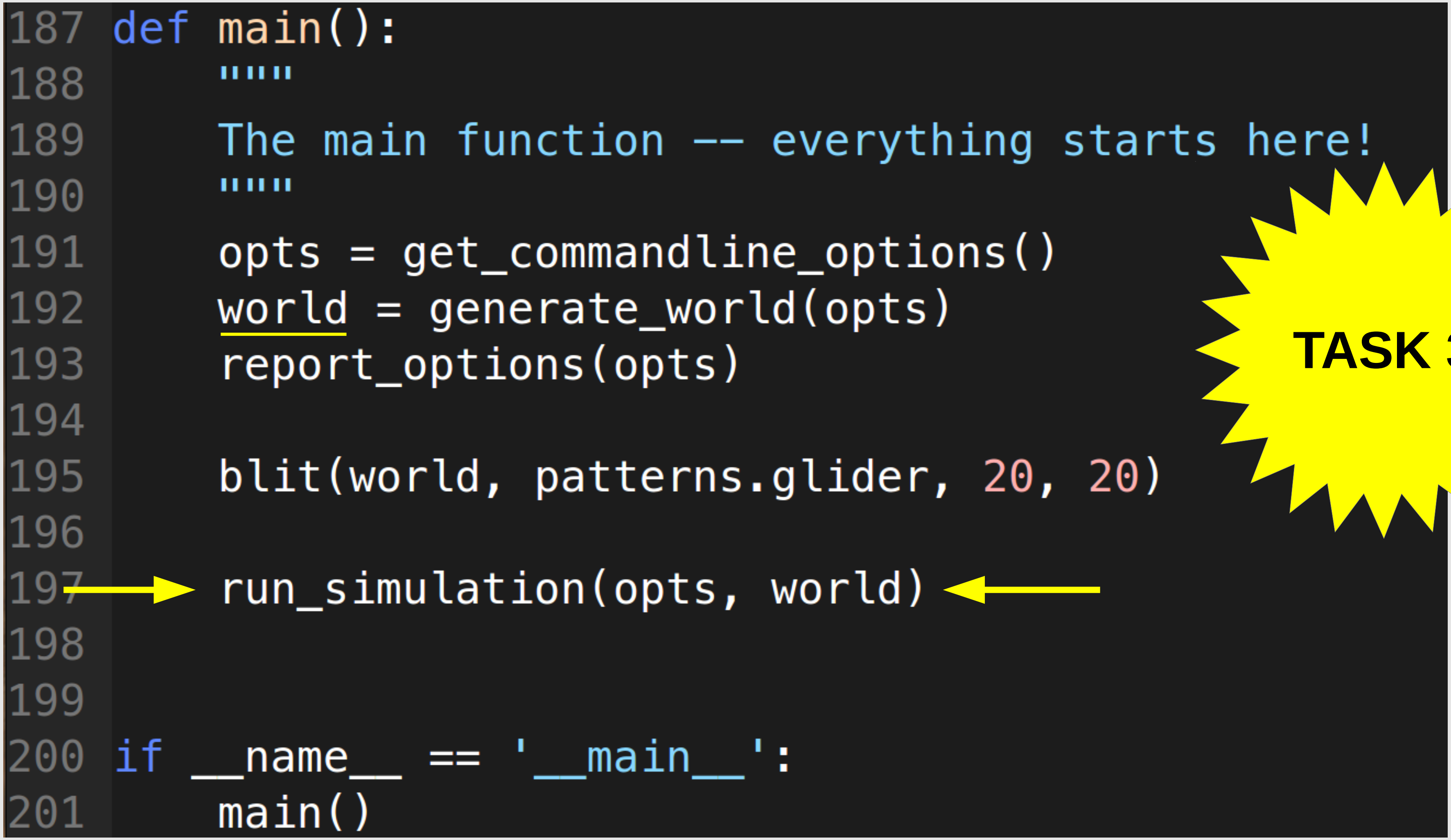
The Skeleton Code

TASK 2



```python
def blit(world, sprite, x, y):
    """
    Accepts: world  -- a 2D world pixel buffer generated by generate_world()
             sprite -- a 2D matrix containing a pattern of 1s and 0s
             x      -- x world coord where left edge of sprite will be placed
             y      -- y world coord where top edge of sprite will be placed

    Returns: (Nothing)

    Description: Copies a 2D pixel pattern (i.e sprite) into the larger 2D
                 world.  The sprite will be copied into the 2D world with
                 its top left corner being located at world coordinate (x,y)
    """
    ## TASK 2 ##############################################################
    #
    #                    [ YOUR CODE GOES HERE ]
    #
    ########################################################################

```

The Skeleton Code

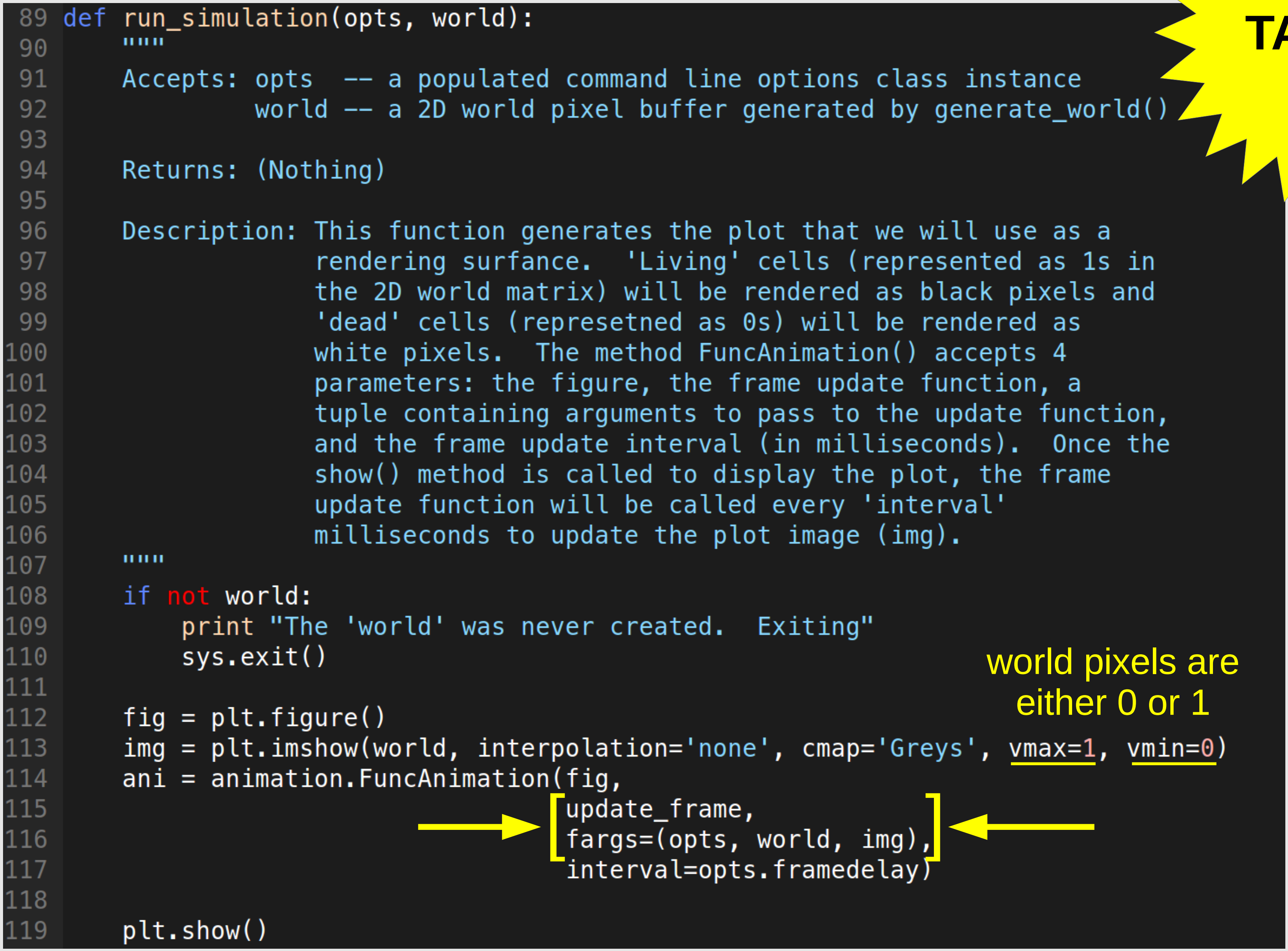


```
def main():
    """
    The main function -- everything starts here!
    """
    opts = get_commandline_options()
    world = generate_world(opts)
    report_options(opts)

    blit(world, patterns.glider, 20, 20)

    run_simulation(opts, world)


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

TASK 3

The Skeleton Code

TASK 3



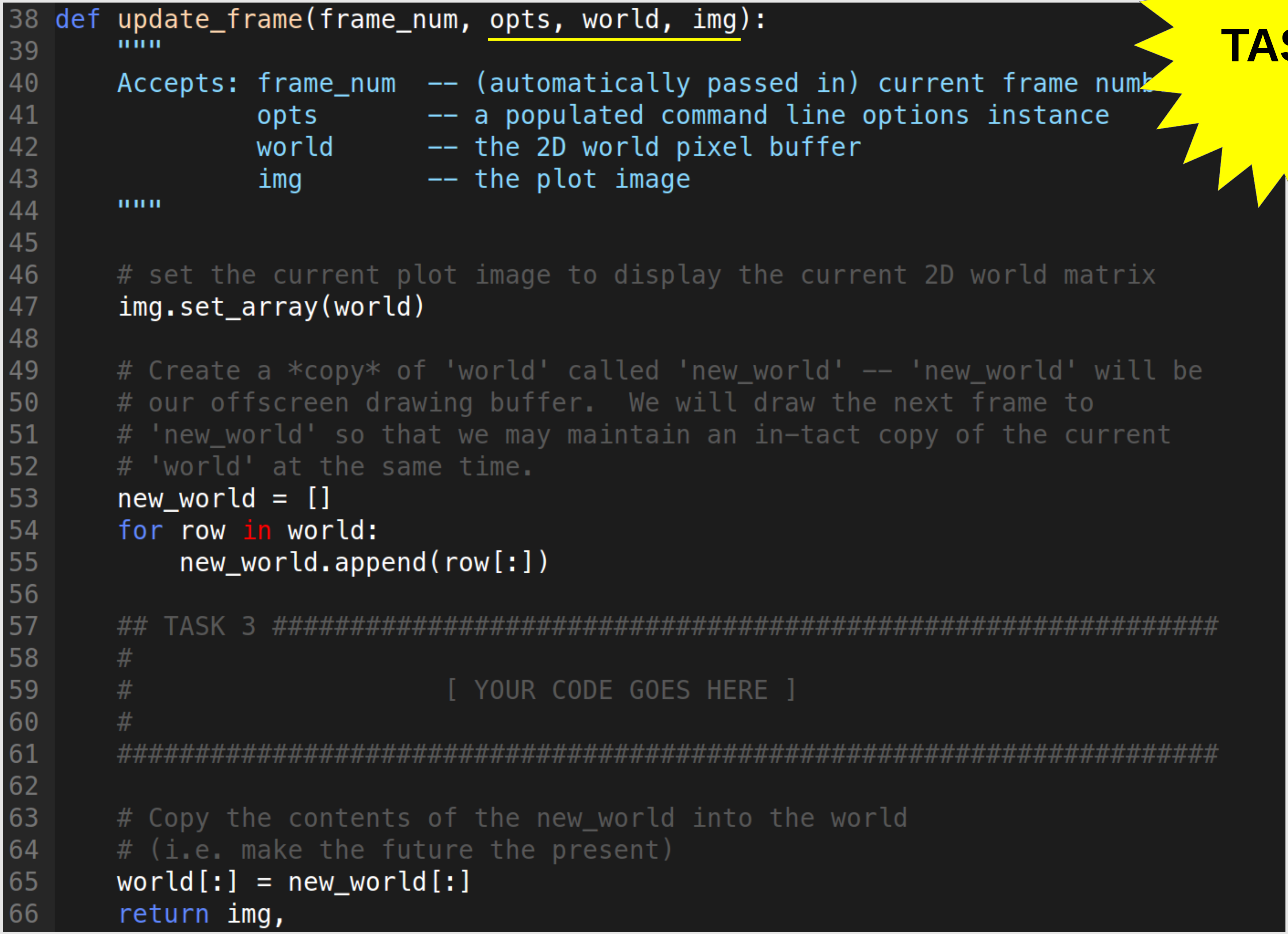
```
def run_simulation(opts, world):
    """
    Accepts: opts  -- a populated command line options class instance
             world -- a 2D world pixel buffer generated by generate_world()

    Returns: (Nothing)

    Description: This function generates the plot that we will use as a
                 rendering surface.  'Living' cells (represented as 1s in
                 the 2D world matrix) will be rendered as black pixels and
                 'dead' cells (represetned as 0s) will be rendered as
                 white pixels.  The method FuncAnimation() accepts 4
                 parameters: the figure, the frame update function, a
                 tuple containing arguments to pass to the update function,
                 and the frame update interval (in milliseconds).  Once the
                 show() method is called to display the plot, the frame
                 update function will be called every 'interval'
                 milliseconds to update the plot image (img).
    """
    if not world:
        print "The 'world' was never created.  Exiting"
        sys.exit()

    fig = plt.figure()
    img = plt.imshow(world, interpolation='none', cmap='Greys', vmax=1, vmin=0)
    ani = animation.FuncAnimation(fig,
                                  update_frame,
                                  fargs=(opts, world, img),
                                  interval=opts.framedelay)


    plt.show()
```

world pixels are either 0 or 1

Conway's Game of Life

TASK 3

The Skeleton Code

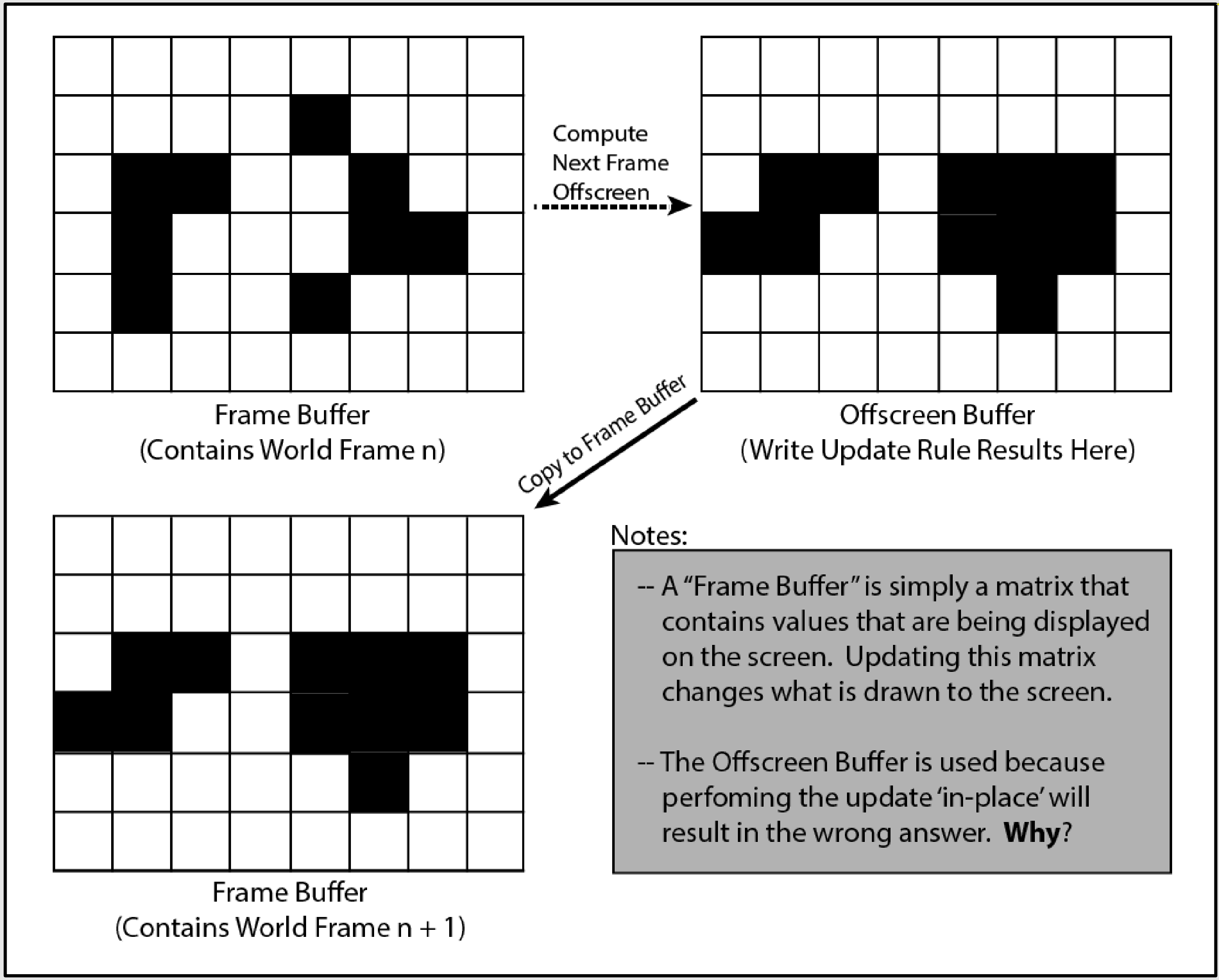


```
def update_frame(frame_num, opts, world, img):
    """
    Accepts: frame_num  -- (automatically passed in) current frame numb
             opts       -- a populated command line options instance
             world      -- the 2D world pixel buffer
             img        -- the plot image
    """

    # set the current plot image to display the current 2D world matrix
    img.set_array(world)

    # Create a *copy* of 'world' called 'new_world' -- 'new_world' will be
    # our offscreen drawing buffer.  We will draw the next frame to
    # 'new_world' so that we may maintain an in-tact copy of the current
    # 'world' at the same time.
    new_world = []
    for row in world:
        new_world.append(row[:])

    ## TASK 3 ##############################################################
    #
    #                     [ YOUR CODE GOES HERE ]
    #
    ########################################################################

    # Copy the contents of the new_world into the world
    # (i.e. make the future the present)
    world[:] = new_world[:]
    return img,
```

Conway's Game of Life

The Skeleton Code

TASK 3



Compute Next Frame Offscreen

Frame Buffer
(Contains World Frame n)

Offscreen Buffer
(Write Update Rule Results Here)

Copy to Frame Buffer

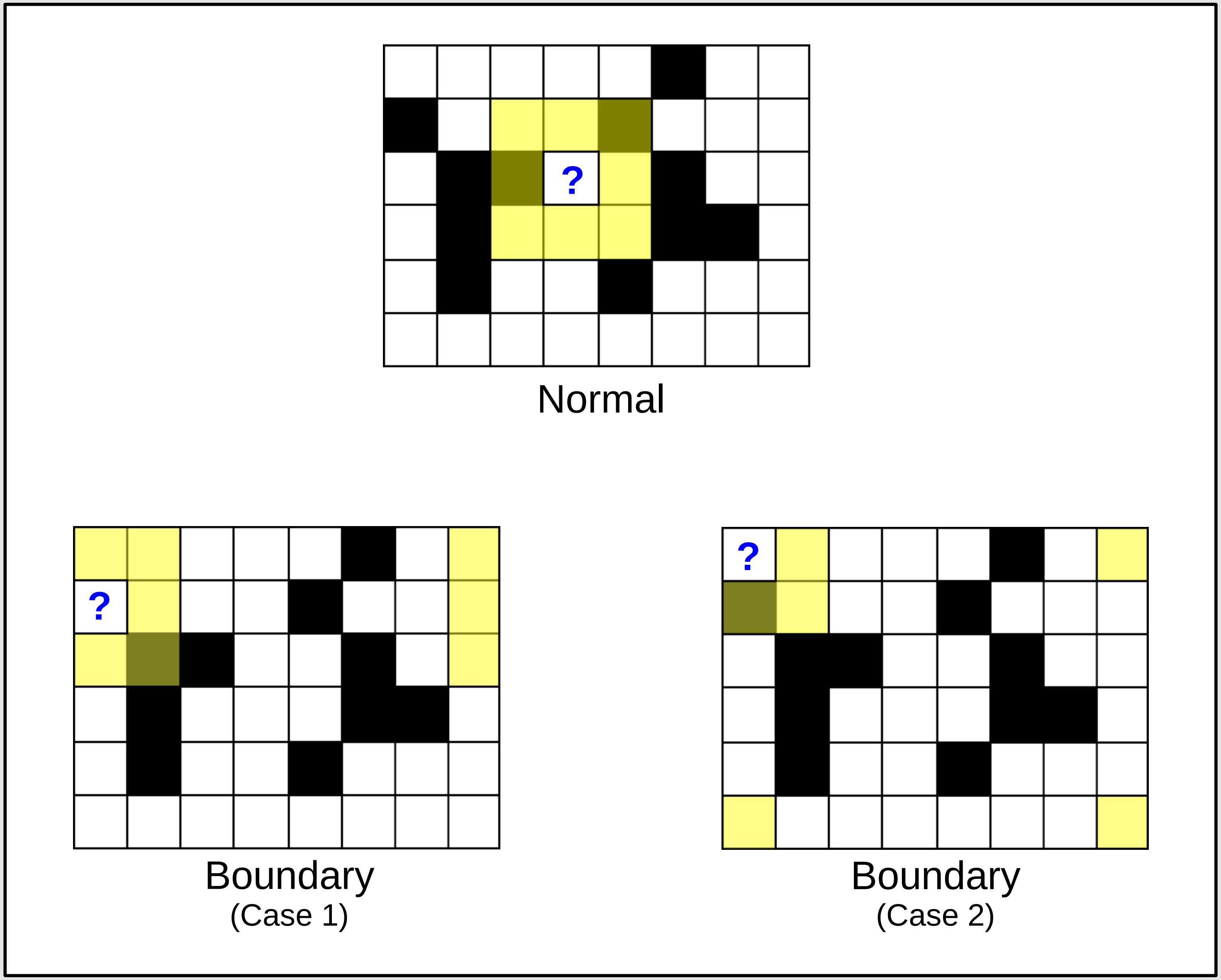
Frame Buffer
(Contains World Frame n + 1)

Notes:

-- A "Frame Buffer" is simply a matrix that contains values that are being displayed on the screen. Updating this matrix changes what is drawn to the screen.

-- The Offscreen Buffer is used because perfoming the update 'in-place' will result in the wrong answer. **Why**?

Boundary Conditions



Conway's Game of Life

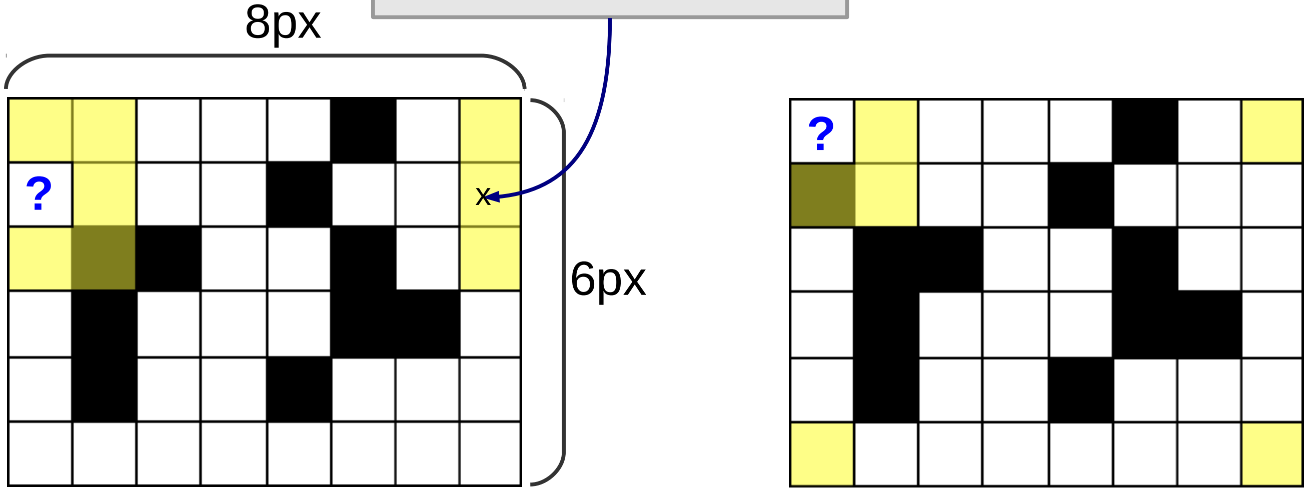
Boundary Conditions

Easily Handled Using Modular Arithmetic

Neighbor *x* coord → (pixel *x* coord) **mod** (world width)

Neighbor *y* coord → (pixel *y* coord) **mod** (world height)

```
0 - 1 mod 8 = 7
```

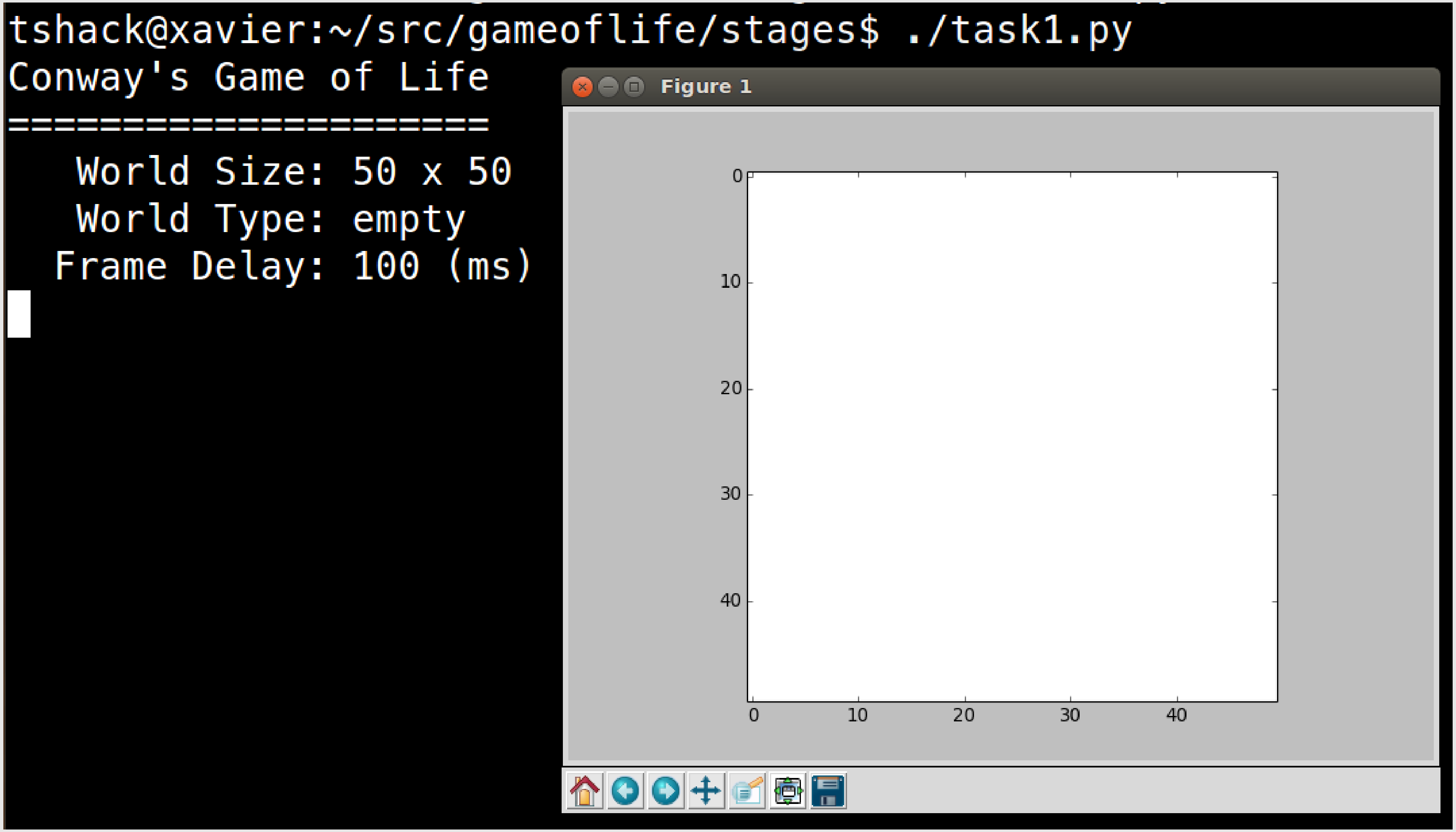


Boundary (Case 1)

Boundary (Case 2)

Making Progress

After Successfully Completing Task 1

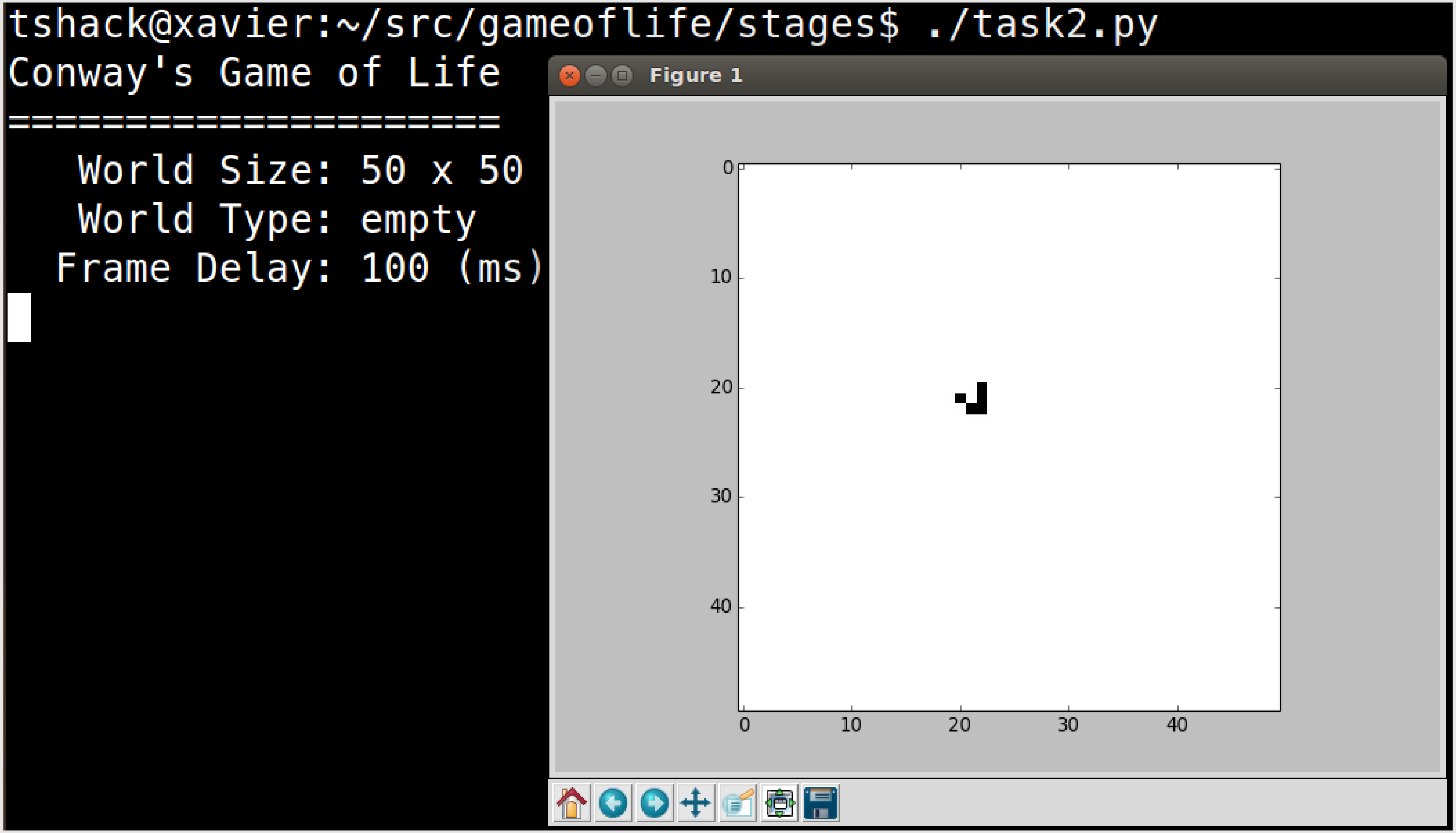


```
tshack@xavier:~/src/gameoflife/stages$ ./task1.py
Conway's Game of Life
=====================
   World Size: 50 x 50
   World Type: empty
  Frame Delay: 100 (ms)
```

Conway's Game of Life

Making Progress

After Successfully Completing Task 2

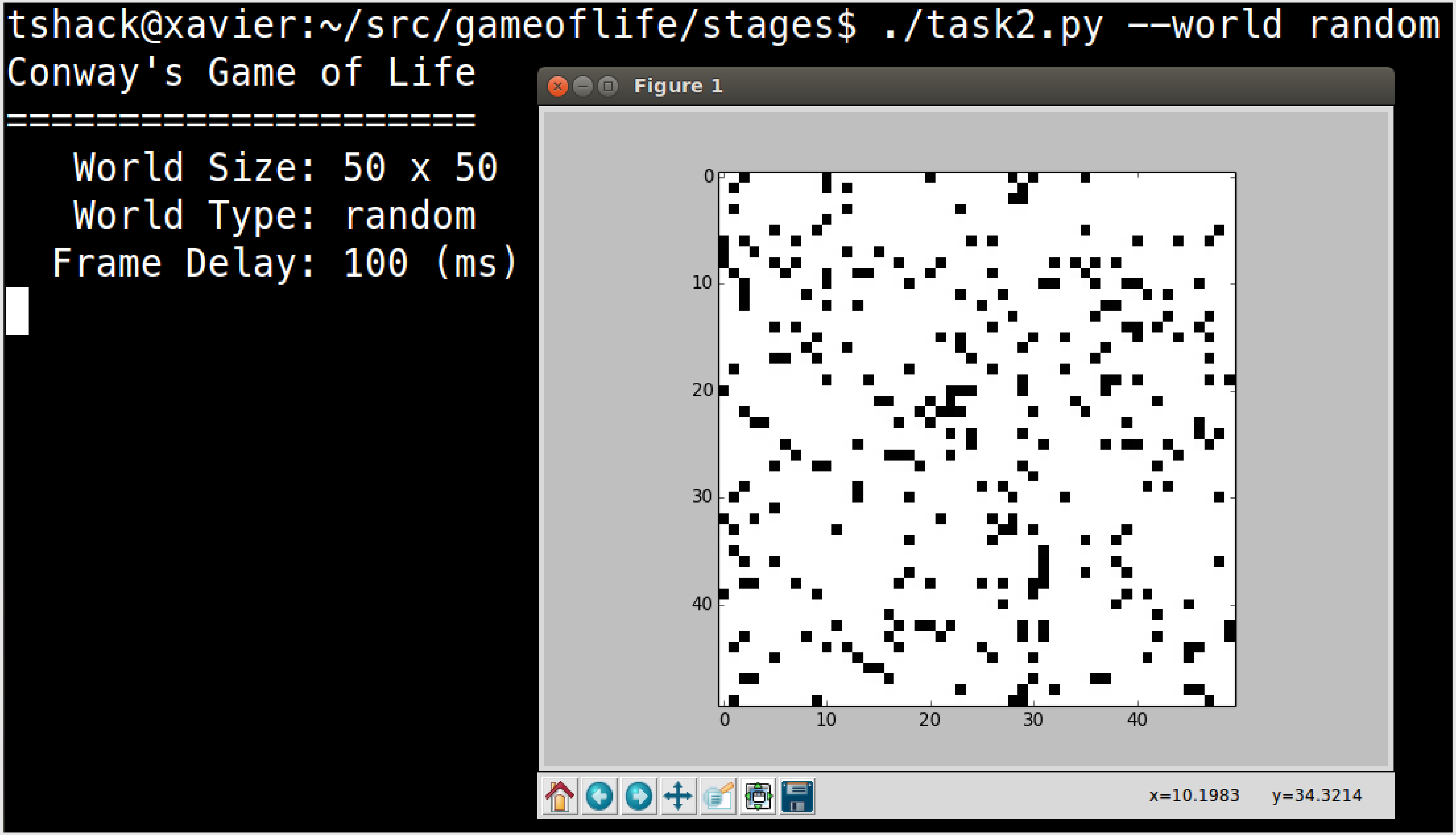


```
tshack@xavier:~/src/gameoflife/stages$ ./task2.py
Conway's Game of Life
=====================
   World Size: 50 x 50
   World Type: empty
  Frame Delay: 100 (ms)
```

Conway's Game of Life

Making Progress

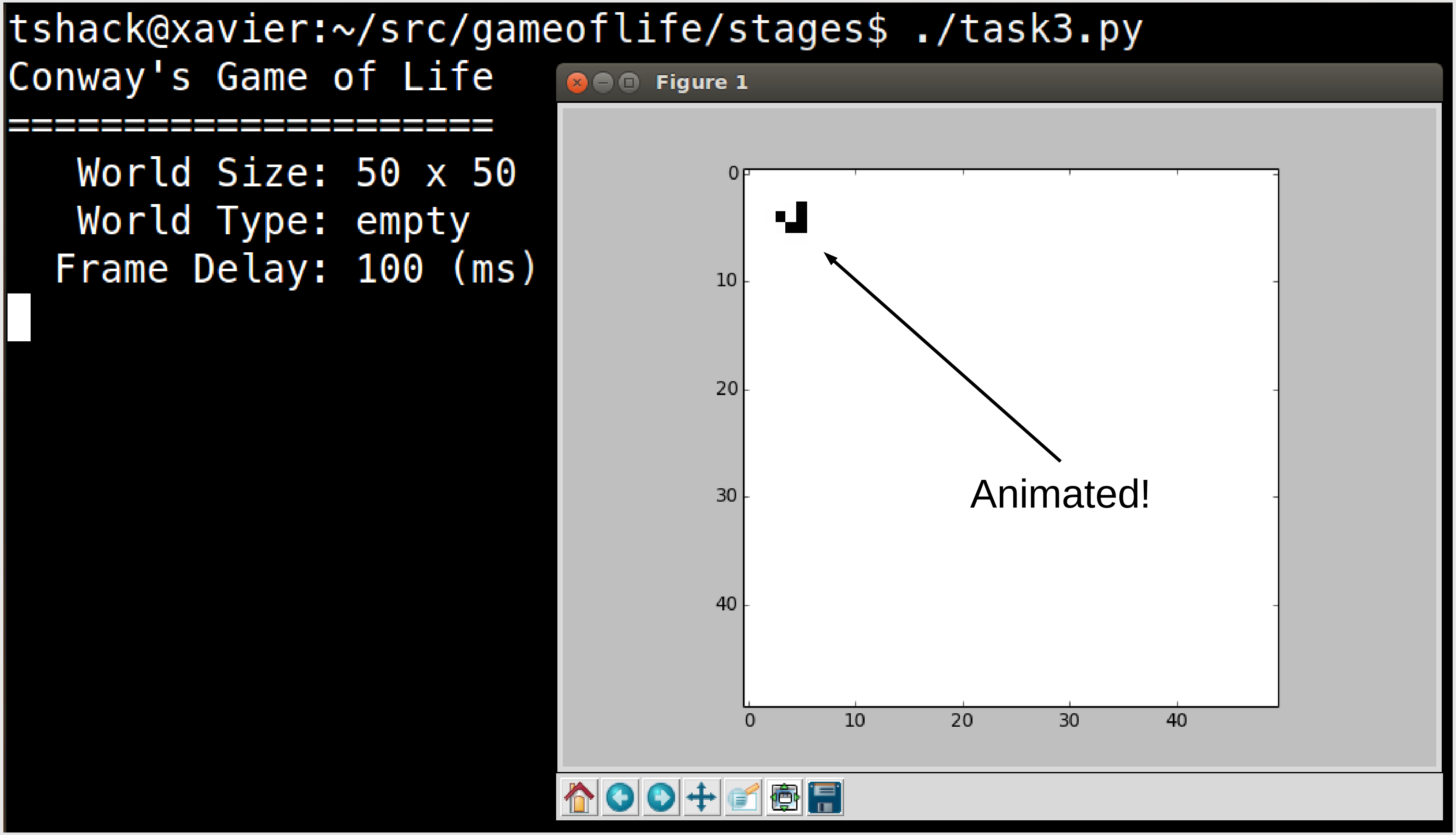
After Successfully Completing Task 2



```
tshack@xavier:~/src/gameoflife/stages$ ./task2.py --world random
Conway's Game of Life
=====================
   World Size: 50 x 50
   World Type: random
  Frame Delay: 100 (ms)
```

Making Progress

After Successfully Completing Task 3



```
tshack@xavier:~/src/gameoflife/stages$ ./task3.py
Conway's Game of Life
=====================
   World Size: 50 x 50
   World Type: empty
  Frame Delay: 100 (ms)
```

Animated!

Conway's Game of Life

Now Go Have Some Fun!

(check out other cool patterns in patterns.py)