

Texas A&M University Kingsville
Department of EECS
CSEN 5303 Foundations of Computer Science
Project 1 Conway's Game of Life

Ameya Khot
Mengxiang Jiang
Professor Habib Ammari

September 23, 2022

Contents

1	Introduction	2
2	Design	3
3	Code	5
4	Tests	19
5	Lessons Learned	22
6	Bibliography	23

Chapter 1

Introduction

In 1968, John Conway, an English mathematician at Cambridge University, tried to simplify a machine with a set of complicated rules to replicate itself on a 2D grid of squares by John von Neumann. He didn't use a computer to simulate his rules, but rather "shuffled poker chips, foreign coins, cowrie shells, Go stones or whatever came to hand, until there was a viable balance between life and death." [3] When he finally succeeded, his game would remain mainly an academic curiosity until he showed it in 1970 to Martin Gardner, a great popularizer of recreational mathematics. Gardner put the game into his "Mathematical Games" column of *Scientific American*, and it became the most popular column he had ever written as well as "made Conway an instant celebrity. The game was written up in *Time*." [2] And the rest is history as the saying goes.

Well, I'm sure you're very excited to learn about the simple rules of Conway's Game of Life. And it is actually very simple. They are:

1. A live cell lives if it has two or three neighbors and dies otherwise.
2. A dead cell becomes alive if it has exactly three live neighbors, otherwise it stays dead.

Despite how simple these rules are, the complexity of the behavior of these cells is extremely difficult to predict. In fact, this seemingly simple game has been proven to be equivalent to a universal Turing machine, which basically means that it can be used to program any software that your favorite programming language is capable of programming (assuming your favorite programming language is Turing complete of course). [4] This paper will not be delving into any depth of this complexity.

Instead, the main purpose of this paper is to explain an implementation of the game using an object oriented approach in Python, and utilizing the Pygame library to display the cells from generation to generation. Some of the optimization ideas from a C++ implementation by the legendary game programmer Michael Abrash in his famous *Michael Abrash's Graphics Programming Black Book*. [1]

Chapter 2

Design

The main data structure we used to capture the 2D grid of squares for the cells is, fairly obviously, a 2D array. However, rather than simply use the raw 2D array, we decided to create a class wrapper around it called *cellmap*. There are two main advantages to this. One, higher level users do not need to know the implementation details and can treat it as a black box of extraction, so if a more efficient data structure or implementation is found, the higher level users will not need to change their code. Two, some of the functionality should not be modified by higher level users, so using a wrapper class allows us to encapsulate the functionality I want private. The basic functionality of a *cellmap* object is detailed below:

```
class cellmap
  private width
  private height
  private cells
  private changed
  private generation
  private steady state

  public procedure new(width, height, rand, file)
    // initialize variables of the cellmap
    // if rand is true, randomly assign cells as live or dead
    // if file is present, use file to assign cells
  endprocedure

  private procedure turn_cell_on(x, y)
    // sets the state of the cell at coord (x, y) to be alive
  endprocedure

  private procedure turn_cell_off(x, y)
    // sets the state of the cell at coord (x, y) to be dead
  endprocedure

  private procedure count_on_neighbors(x, y)
    // counts live neighbors of the cell at coord (x, y)
```

```

endprocedure

public procedure cell_state(x, y)
    // returns the state of the cell at coord (x, y)
    // if (x, y) is out of bounds return dead
endprocedure

public procedure next_generation()
    // evaluates the next generation of cells in a new cellmap
    // returns the new cellmap if at least one cell changed
endprocedure

public procedure write_to_file(file)
    // write the states of the cell to the given file
    // used for testing/debugging purposes
endprocedure

endclass

```

Most of the functionality is very trivial, so we will go over the two parts that are slightly more complex, namely the *count_on_neighbors* and *next_generation*:

For *count_on_neighbors*, rather than use three different cases (cells at the corners, edges, and center), we generalized the cases to just consider all cells as if they were in the center and therefore always having 8 neighbors. There are two ways to do this, one as stated in class is the pad the 2D array on the outside with dead cells. The other, which is the way we did it here, is to have the getter method (*cell_state*), return the state of out of bounds cells as dead.

For *next_generation*, first the old *cellmap* has its private variable *steady_state* checked. If it's true, then it simply returns the old *cellmap* since it has reached steady state. Else, a new *cellmap* is allocated, each cell of the old *cellmap* has its neighbors counted. This count determined whether the corresponding cell in the new *cellmap* lives or dies, based on the rules. Whenever a new cell state changes (from live to dead or vice versa), it is added to the *changed* stack of the new *cellmap* (*changed* is initialized to be empty on every newly constructed *cellmap*). If *changed* is empty at the end of checking all the old cells, the old *cellmap* changes the *steady_state* to true and returns itself. Otherwise, the new *cellmap* has the *generation* counter incremented by one, and is returned.

This is all well and good, but how do we view the *cellmap* and successive generations of it? Well, we need some form of drawing/displaying of the cells on a continuous loop, and this sounds very much like a game engine. It just so happens that Python has a fairly decent game engine library called Pygame, which we imported rather than implement our own from scratch.[5]

Chapter 3

Code

```
1 """
2 cell map class for keeping track of cells for game of life
3 """
4 # Import random library since I need to randomly generate a cell map at
   the start
5 import random
6
7 class cellmap:
8     ADJACENT = [[-1, -1], [-1, 0], [-1, 1], [0, -1], [0, 1], [1, -1], [1,
9         0], [1, 1]]
10
11     # cellmap constructor
12     def __init__(self, width, height, rand=False, file=None):
13         self.width = width
14         self.height = height
15         self.cells = [[0]*self.height for i in range(self.width)]
16         self.changed = []
17         self.generation = 0
18         self.steady_state = False
19         if rand:
20             for x in range(width):
21                 for y in range(height):
22                     if random.random() > 0.5:
23                         self.turn_cell_on(x, y)
24         if file:
25             f = open(file, 'r')
26             self.width = int(f.readline())
27             self.height = int(f.readline())
28             self.cells = [[0]*self.height for i in range(self.width)]
29             for y in range(self.height):
30                 for x, c in enumerate(f.readline()):
31                     if c == '0':
32                         self.turn_cell_off(x, y)
33                     if c == '1':
34                         self.turn_cell_on(x, y)
35             f.close()
36
37     # writes cellmap to file
```

```

37 def write_to_file(self, file):
38     f = open(file, 'w')
39     f.writelines([f'{self.width}', '\n', f'{self.height}', '\n'])
40     for y in range(self.height):
41         for x in range(self.width):
42             f.write(str(self.cell_state(x, y)))
43             f.write('\n')
44     f.close()
45
46 # turns cell on
47 def turn_cell_on(self, x, y):
48     self.cells[x][y] = 1
49
50 # turns cell off
51 def turn_cell_off(self, x, y):
52     self.cells[x][y] = 0
53
54 # returns the cell state
55 def cell_state(self, x, y):
56     if 0 <= x < self.width and 0 <= y < self.height:
57         return self.cells[x][y]
58     else:
59         return 0
60
61 # count the number of neighbors on of a given cell
62 def count_on_neighbors(self, x, y):
63     neighbor_count = 0
64     for i, j in self.ADJACENT:
65         neighbor_count += self.cell_state(x + i, y + j)
66     return neighbor_count
67
68 # returns the next generation cell map
69 def next_generation(self):
70     if self.steady_state:
71         return self
72     else:
73         next_map = cellmap(self.width, self.height)
74
75         for x in range(self.width):
76             for y in range(self.height):
77                 neighbor_count = self.count_on_neighbors(x, y)
78                 # if the cell is on, turn it off if it has too many or
too few on neighbors
79                 if self.cell_state(x, y):
80                     if neighbor_count < 2 or neighbor_count > 3:
81                         next_map.turn_cell_off(x, y)
82                         next_map.changed.append((x, y))
83                     else:
84                         next_map.turn_cell_on(x, y)
85                 # if the cell is off, turn it on if it has enough on
neighbors
86                 else:
87                     if neighbor_count == 3:
88                         next_map.turn_cell_on(x, y)

```

```

89         next_map.changed.append((x, y))
90     if len(next_map.changed) == 0:
91         self.steady_state = True
92         return self
93     else:
94         next_map.generation = self.generation + 1
95         return next_map

```

Listing 3.1: cell_map.py

```

1 # https://www.jagregory.com/abrash-black-book/#chapter-17-the-game-of-life
2
3 """
4 Basic Game of Life program in Python using the Pygame library to draw
5 """
6
7 # Import libraries
8 from email import message
9 from tkinter import filedialog
10 import pygame
11 import pygame_menu
12 import random
13 from cell_map import cellmap
14 from tkinter import *
15 from tkinter import messagebox
16 import win32gui
17
18 root = Tk()
19 root.withdraw() #to hide the main window
20
21 pygame.init()
22
23 # font for pygame
24 pygame_font = pygame.font.Font('WHITRABT.ttf', 30)
25
26 # States of the game:
27 MAIN_MENU = 0
28 PLAYING = 1
29
30 # Window constants
31 WINDOW_WIDTH = 800
32 WINDOW_HEIGHT = 600
33
34 # Text constants
35 WIDTH_TEXT_X = 520
36 WIDTH_TEXT_Y = 50
37 HEIGHT_TEXT_X = 520
38 HEIGHT_TEXT_Y = 100
39 GENERATION_TEXT_X = 520
40 GENERATION_TEXT_Y = 150
41 LIVE_TEXT_X = 520
42 LIVE_TEXT_Y = 200
43 DEAD_TEXT_X = 520
44 DEAD_TEXT_Y = 250

```



```

45 STEADY_TEXT_X = 520
46 STEADY_TEXT_Y = 300
47 TEXT_COLOR = (200, 200, 200)
48
49 # Button constants
50 BUTTON_WIDTH_LARGE = 200
51 BUTTON_WIDTH_SMALL = 100
52 BUTTON_HEIGHT = 50
53 NEXT_BUTTON_X = 50
54 NEXT_BUTTON_Y = 520
55 AUTO_BUTTON_X = 200
56 AUTO_BUTTON_Y = 520
57 WRITE_BUTTON_X = 450
58 WRITE_BUTTON_Y = 520
59 BACK_BUTTON_X = 600
60 BACK_BUTTON_Y = 520
61 BUTTON_LIGHT = (170, 170, 170)
62 BUTTON_DARK = (100, 100, 100)
63 BUTTON_TEXT_COLOR = (0, 255, 0)
64 BUTTON_TEXT_OFFSET_X = 10
65 BUTTON_TEXT_OFFSET_Y = 10
66
67 # Initial state should be on the main menu
68 game_state = MAIN_MENU
69
70 # How quickly should the next generation be created? Time in milliseconds
71 GEN_INTERVAL = 100
72
73 # Create a custom event for next cell generation
74 NEXTGEN = pygame.USEREVENT + 1
75 pygame.time.set_timer(NEXTGEN, GEN_INTERVAL)
76
77 # cell map displaying variables
78 cellmap_width = random.randint(1, 100)
79 cellmap_height = random.randint(1, 100)
80 cell_pixel_size = 4
81
82 # cell colors (Official Blue and Gold from TAMUK graphics standards)
83 # https://www.tamuk.edu/marcomm/\_images\_MARCOMM/branding/graphic\_standards.pdf
84 live_color = (255, 196, 37)
85 dead_color = (0, 93, 170)
86
87 # Set up the drawing window
88 screen = pygame.display.set_mode((WINDOW_WIDTH, WINDOW_HEIGHT))
89 pygame.display.set_caption('Game of Life')
90
91 # generates a random cell map
92 current_map = cellmap(cellmap_width, cellmap_height, rand=True)
93
94 # writing the current_map to a file for debugging purposes
95 current_map.write_to_file('current_map.txt')
96
97 about_menu = pygame_menu.Menu('About', WINDOW_WIDTH, WINDOW_HEIGHT,

```

```

98         theme=pygame_menu.themes.THEME_DARK)
99 about_menu.add.label('This is a programming project for the course')
100 about_menu.add.label('Foundations of Computer Science at TAMUK.')
101 about_menu.add.label('The Game of Life is a 2D simulation of cells')
102 about_menu.add.label('that live or die in successive generations.')
103 about_menu.add.label('Only the starting state of the cells is needed')
104 about_menu.add.label('to produce the next generation, so this is not')
105 about_menu.add.label('very interactive beyond setting initial conditions.'
106 )
107
108 about_menu.add.label('')
109 about_menu.add.button('Back', pygame_menu.events.BACK)
110
111 settings_menu = pygame_menu.Menu('Settings', WINDOW_WIDTH, WINDOW_HEIGHT,
112     theme=pygame_menu.themes.THEME_DARK)
113 def check_cellmap_width(value):
114     global cellmap_width
115     if value < 0:
116         messagebox.showerror('Invalid width', 'value must be positive')
117     elif value > 100:
118         messagebox.showerror('Invalid width', 'value must be less than 100')
119     else:
120         cellmap_width = value
121
122 settings_menu.add.text_input(
123     'Width: ',
124     default=cellmap_width,
125     onchange=check_cellmap_width,
126     input_type=pygame_menu.locals.INPUT_INT,
127     textinput_id='cellmap_width'
128 )
129
130 def check_cellmap_height(value):
131     global cellmap_height
132     if value < 0:
133         messagebox.showerror('Invalid height', 'value must be positive')
134     elif value > 100:
135         messagebox.showerror('Invalid height', 'value must be less than
136 100')
137     else:
138         cellmap_height = value
139
140 settings_menu.add.text_input(
141     'Height: ',
142     default=cellmap_height,
143     onchange=check_cellmap_height,
144     input_type=pygame_menu.locals.INPUT_INT,
145     textinput_id='cellmap_height'
146 )
147
148 def check_live_color(value):
149     global live_color

```

```

149     global dead_color
150     if value == dead_color:
151         messagebox.showerror('Invalid live color', 'color must be
different from dead color')
152     else:
153         live_color = value
154
155 settings_menu.add.color_input(
156     'Live Color (R,G,B): ',
157     default=live_color,
158     color_type=pygame_menu.widgets.COLORINPUT_TYPE_RGB,
159     onchange=check_live_color,
160     color_id='live_color'
161 )
162
163 def check_dead_color(value):
164     global live_color
165     global dead_color
166     if value == live_color:
167         messagebox.showerror('Invalid dead color', 'color must be
different from livecolor')
168     else:
169         dead_color = value
170
171 settings_menu.add.color_input(
172     'Dead Color (R,G,B): ',
173     default=dead_color,
174     color_type=pygame_menu.widgets.COLORINPUT_TYPE_RGB,
175     onchange=check_dead_color,
176     color_id='dead_color'
177 )
178
179 def load_from_file():
180     f = filedialog.askopenfilename()
181     hwnd = pygame.display.get_wm_info()['window']
182     win32gui.SetFocus(hwnd)
183     start_the_game()
184     global current_map
185     global cellmap_width
186     global cellmap_height
187     current_map = cellmap(cellmap_width, cellmap_height, file=f)
188     cellmap_width = current_map.width
189     cellmap_height = current_map.height
190
191 settings_menu.add.button('Load from file', load_from_file)
192
193 settings_menu.add.button('Back', pygame_menu.events.BACK)
194
195 main_menu = pygame_menu.Menu('Conway\'s Game of Life', WINDOW_WIDTH,
WINDOW_HEIGHT,
196                             theme=pygame_menu.themes.THEME_DARK)
197
198 def start_the_game():
199     global game_state

```

```

200     global main_menu
201     global current_map
202     global cellmap_width
203     global cellmap_height
204     game_state = PLAYING
205     main_menu.disable()
206     screen.fill((0, 0, 0))
207     current_map = cellmap(cellmap_width, cellmap_height, rand=True)
208
209     main_menu.add.button('Play', start_the_game)
210     main_menu.add.button('About', about_menu) # Add about submenu
211     main_menu.add.button('Settings', settings_menu) # Add settings submenu
212     main_menu.add.button('Quit', pygame_menu.events.EXIT)
213
214
215     # state for checking whether the user wants the next generation to be
216     # generated automatically
217     auto_state = False
218
219     def draw_next_button(mouse):
220         if NEXT_BUTTON_X <= mouse[0] <= (NEXT_BUTTON_X + BUTTON_WIDTH_SMALL) \
221             and NEXT_BUTTON_Y <= mouse[1] <= (NEXT_BUTTON_Y + BUTTON_HEIGHT):
222             pygame.draw.rect(screen, BUTTON_LIGHT, [NEXT_BUTTON_X,
223                 NEXT_BUTTON_Y, BUTTON_WIDTH_SMALL, BUTTON_HEIGHT])
224         else:
225             pygame.draw.rect(screen, BUTTON_DARK, [NEXT_BUTTON_X,
226                 NEXT_BUTTON_Y, BUTTON_WIDTH_SMALL, BUTTON_HEIGHT])
227             text = pygame_font.render('NEXT', True, BUTTON_TEXT_COLOR)
228             screen.blit(text, (NEXT_BUTTON_X + BUTTON_TEXT_OFFSET_X, NEXT_BUTTON_Y
229                 + BUTTON_TEXT_OFFSET_Y))
230
231     def draw_auto_button(mouse):
232         if AUTO_BUTTON_X <= mouse[0] <= (AUTO_BUTTON_X + BUTTON_WIDTH_LARGE) \
233             and AUTO_BUTTON_Y <= mouse[1] <= (AUTO_BUTTON_Y + BUTTON_HEIGHT):
234             pygame.draw.rect(screen, BUTTON_LIGHT, [AUTO_BUTTON_X,
235                 AUTO_BUTTON_Y, BUTTON_WIDTH_LARGE, BUTTON_HEIGHT])
236         else:
237             pygame.draw.rect(screen, BUTTON_DARK, [AUTO_BUTTON_X,
238                 AUTO_BUTTON_Y, BUTTON_WIDTH_LARGE, BUTTON_HEIGHT])
239             text = pygame_font.render(f'AUTO: {auto_state}', True,
240                 BUTTON_TEXT_COLOR)
241             screen.blit(text, (AUTO_BUTTON_X + BUTTON_TEXT_OFFSET_X, AUTO_BUTTON_Y
242                 + BUTTON_TEXT_OFFSET_Y))
243
244     def draw_write_button(mouse):
245         if WRITE_BUTTON_X <= mouse[0] <= (WRITE_BUTTON_X + BUTTON_WIDTH_SMALL) \
246             and WRITE_BUTTON_Y <= mouse[1] <= (WRITE_BUTTON_Y + BUTTON_HEIGHT):
247             pygame.draw.rect(screen, BUTTON_LIGHT, [WRITE_BUTTON_X,
248                 WRITE_BUTTON_Y, BUTTON_WIDTH_SMALL, BUTTON_HEIGHT])
249         else:
250             pygame.draw.rect(screen, BUTTON_DARK, [WRITE_BUTTON_X,
251                 WRITE_BUTTON_Y, BUTTON_WIDTH_SMALL, BUTTON_HEIGHT])
252             text = pygame_font.render('WRITE', True, BUTTON_TEXT_COLOR)

```

```

243     screen.blit(text, (WRITE_BUTTON_X + BUTTON_TEXT_OFFSET_X,
244                        WRITE_BUTTON_Y + BUTTON_TEXT_OFFSET_Y))
245
246 def draw_back_button(mouse):
247     if BACK_BUTTON_X <= mouse[0] <= (BACK_BUTTON_X + BUTTON_WIDTH_SMALL) \
248     and BACK_BUTTON_Y <= mouse[1] <= (BACK_BUTTON_Y + BUTTON_HEIGHT):
249         pygame.draw.rect(screen, BUTTON_LIGHT, [BACK_BUTTON_X,
250 BACK_BUTTON_Y, BUTTON_WIDTH_SMALL, BUTTON_HEIGHT])
251     else:
252         pygame.draw.rect(screen, BUTTON_DARK, [BACK_BUTTON_X,
253 BACK_BUTTON_Y, BUTTON_WIDTH_SMALL, BUTTON_HEIGHT])
254     text = pygame_font.render('BACK', True, BUTTON_TEXT_COLOR)
255     screen.blit(text, (BACK_BUTTON_X + BUTTON_TEXT_OFFSET_X, BACK_BUTTON_Y
256 + BUTTON_TEXT_OFFSET_Y))
257
258 def draw_width_text():
259     text = pygame_font.render(f'Width: {cellmap_width}', True, TEXT_COLOR)
260     screen.blit(text, (WIDTH_TEXT_X, WIDTH_TEXT_Y))
261
262 def draw_height_text():
263     text = pygame_font.render(f'Height: {cellmap_height}', True,
264 TEXT_COLOR)
265     screen.blit(text, (HEIGHT_TEXT_X, HEIGHT_TEXT_Y))
266
267 def draw_generation_text():
268     text = pygame_font.render(f'Generation: {current_map.generation}',
269 True, TEXT_COLOR)
270     screen.fill((0, 0, 0), [GENERATION_TEXT_X, GENERATION_TEXT_Y, 300,
271 300])
272     screen.blit(text, (GENERATION_TEXT_X, GENERATION_TEXT_Y))
273
274 def draw_live_color():
275     text = pygame_font.render('Live Color: ', True, TEXT_COLOR)
276     pygame.draw.rect(screen, live_color, [LIVE_TEXT_X + 200, LIVE_TEXT_Y,
277 20, 20])
278     screen.blit(text, (LIVE_TEXT_X, LIVE_TEXT_Y))
279
280 def draw_dead_color():
281     text = pygame_font.render('Dead Color: ', True, TEXT_COLOR)
282     pygame.draw.rect(screen, dead_color, [DEAD_TEXT_X + 200, DEAD_TEXT_Y,
283 20, 20])
284     screen.blit(text, (DEAD_TEXT_X, DEAD_TEXT_Y))
285
286 def draw_steady_state():
287     text = pygame_font.render(f'Finished: {current_map.steady_state}',
288 True, TEXT_COLOR)
289     screen.fill((0, 0, 0), [STEADY_TEXT_X, STEADY_TEXT_Y, 300, 100])
290     screen.blit(text, (STEADY_TEXT_X, STEADY_TEXT_Y))
291
292 # Run until the user asks to quit
293 running = True
294 while running:
295     # gets the current mouse position
296     mouse = pygame.mouse.get_pos()

```

```

287
288 # Did the user click the window close button?
289 for event in pygame.event.get():
290     if event.type == pygame.QUIT:
291         running = False
292
293 # Should the next generation be displayed?
294 elif event.type == NEXTGEN:
295     if game_state == PLAYING and auto_state:
296         current_map = current_map.next_generation()
297         # Stop the previous timer by setting the interval to 0
298         pygame.time.set_timer(NEXTGEN, 0)
299         # Start a new timer
300         pygame.time.set_timer(NEXTGEN, GEN_INTERVAL)
301 elif event.type == pygame.MOUSEBUTTONDOWN:
302     if game_state == PLAYING:
303         if NEXT_BUTTON_X <= mouse[0] <= (NEXT_BUTTON_X +
304     BUTTON_WIDTH_SMALL) \
305             and NEXT_BUTTON_Y <= mouse[1] <= (NEXT_BUTTON_Y +
306     BUTTON_HEIGHT):
307             current_map = current_map.next_generation()
308             elif AUTO_BUTTON_X <= mouse[0] <= (AUTO_BUTTON_X +
309     BUTTON_WIDTH_LARGE) \
310                 and AUTO_BUTTON_Y <= mouse[1] <= (AUTO_BUTTON_Y +
311     BUTTON_HEIGHT):
312                 auto_state = not auto_state
313                 elif WRITE_BUTTON_X <= mouse[0] <= (WRITE_BUTTON_X +
314     BUTTON_WIDTH_SMALL) \
315                     and WRITE_BUTTON_Y <= mouse[1] <= (WRITE_BUTTON_Y +
316     BUTTON_HEIGHT):
317                     f = filedialog.askopenfilename()
318                     hwnd = pygame.display.get_wm_info()['window']
319                     win32gui.SetFocus(hwnd)
320                     current_map.write_to_file(f)
321                     elif BACK_BUTTON_X <= mouse[0] <= (BACK_BUTTON_X +
322     BUTTON_WIDTH_SMALL) \
323                         and BACK_BUTTON_Y <= mouse[1] <= (BACK_BUTTON_Y +
324     BUTTON_HEIGHT):
325                         game_state = MAIN_MENU
326                         main_menu.enable()
327
328 if game_state == MAIN_MENU:
329     main_menu.mainloop(screen)
330
331 if game_state == PLAYING:
332     draw_next_button(mouse)
333     draw_auto_button(mouse)
334     draw_write_button(mouse)
335     draw_back_button(mouse)
336     draw_width_text()
337     draw_height_text()
338     draw_generation_text()
339     draw_live_color()
340     draw_dead_color()

```

```

333     draw_steady_state()
334
335     # Check every cell of the entire cell map if it's the first 10
iterations
336     if current_map.generation == 0:
337         for x in range(current_map.width):
338             for y in range(current_map.height):
339                 if current_map.cell_state(x, y):
340                     pygame.draw.rect(screen, live_color,
341 [(cell_pixel_size + 1) * (x + 1), (cell_pixel_size
+ 1) * (y + 1), cell_pixel_size, cell_pixel_size])
342                 else:
343                     pygame.draw.rect(screen, dead_color,
344 [(cell_pixel_size + 1) * (x + 1), (cell_pixel_size
+ 1) * (y + 1), cell_pixel_size, cell_pixel_size])
345     # Else only draw cells that have changed
346     else:
347         for x, y in current_map.changed:
348             if current_map.cell_state(x, y):
349                 pygame.draw.rect(screen, live_color,
350 [(cell_pixel_size + 1) * (x + 1), (cell_pixel_size +
1) * (y + 1), cell_pixel_size, cell_pixel_size])
351             else:
352                 pygame.draw.rect(screen, dead_color,
353 [(cell_pixel_size + 1) * (x + 1), (cell_pixel_size +
1) * (y + 1), cell_pixel_size, cell_pixel_size])
354
355     # Flip the display to make everything appear
356     pygame.display.flip()
357
358 # Done! Time to quit.
359 pygame.quit()

```

Listing 3.2: game_of_life.py

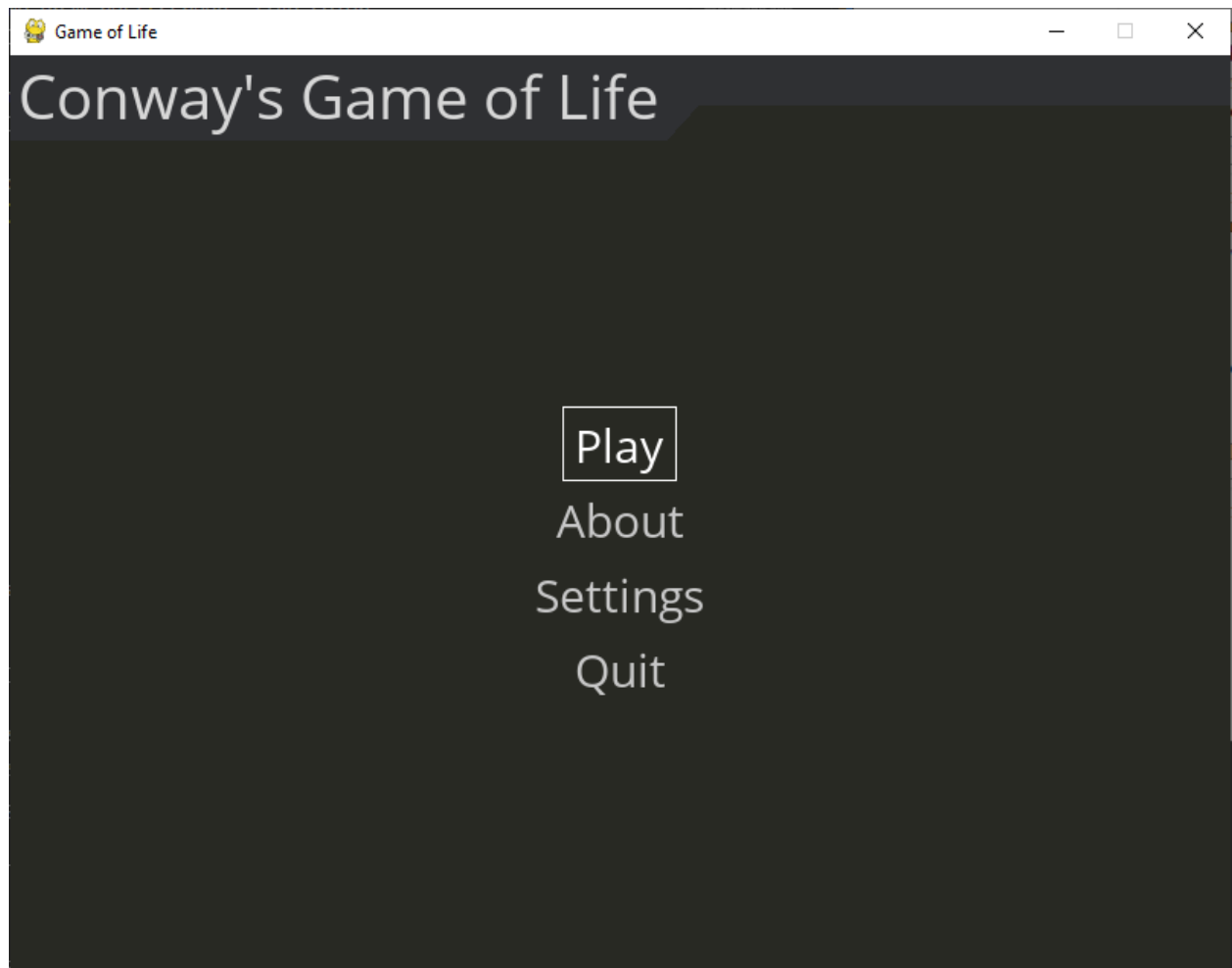


Figure 3.1: main menu

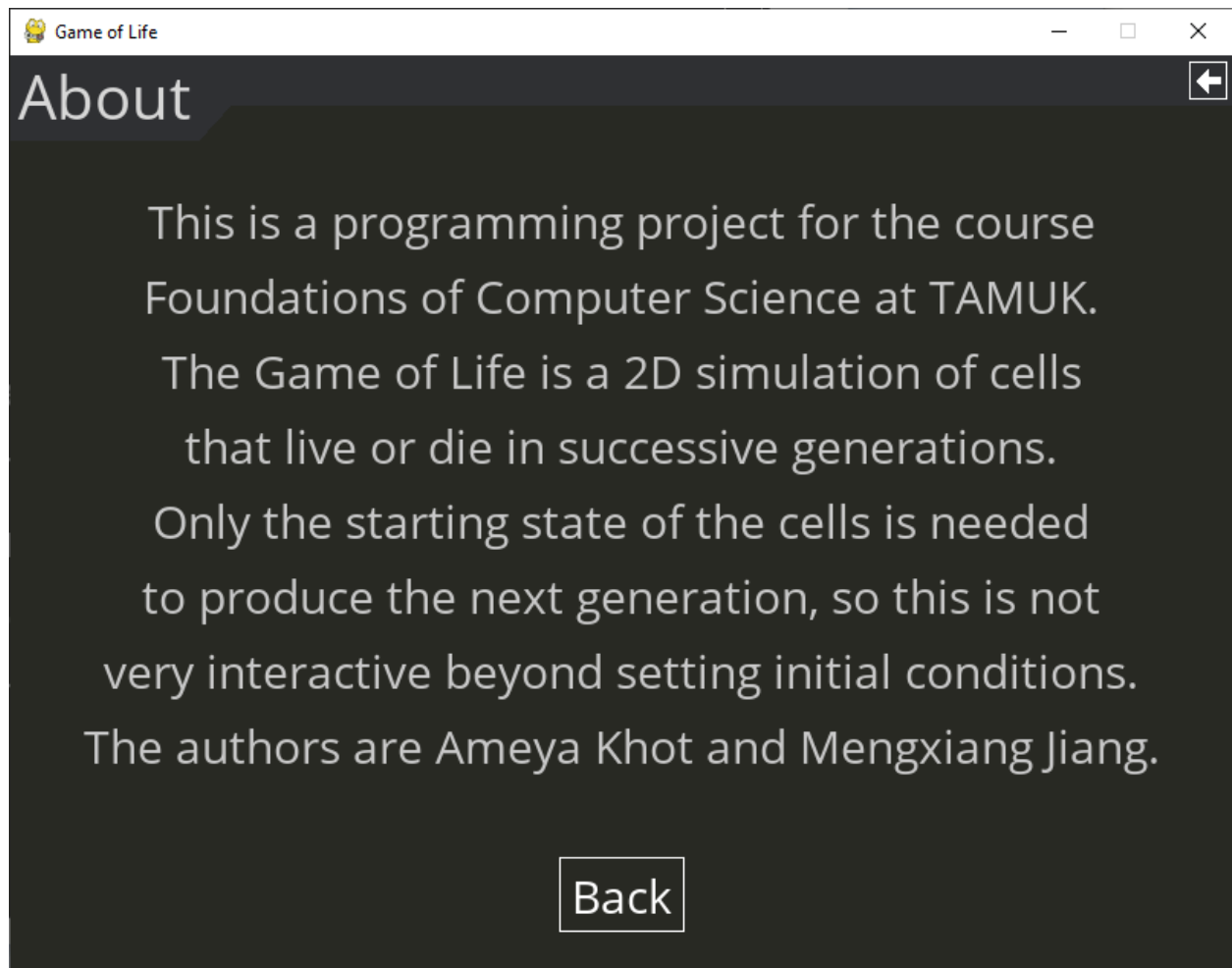


Figure 3.2: about

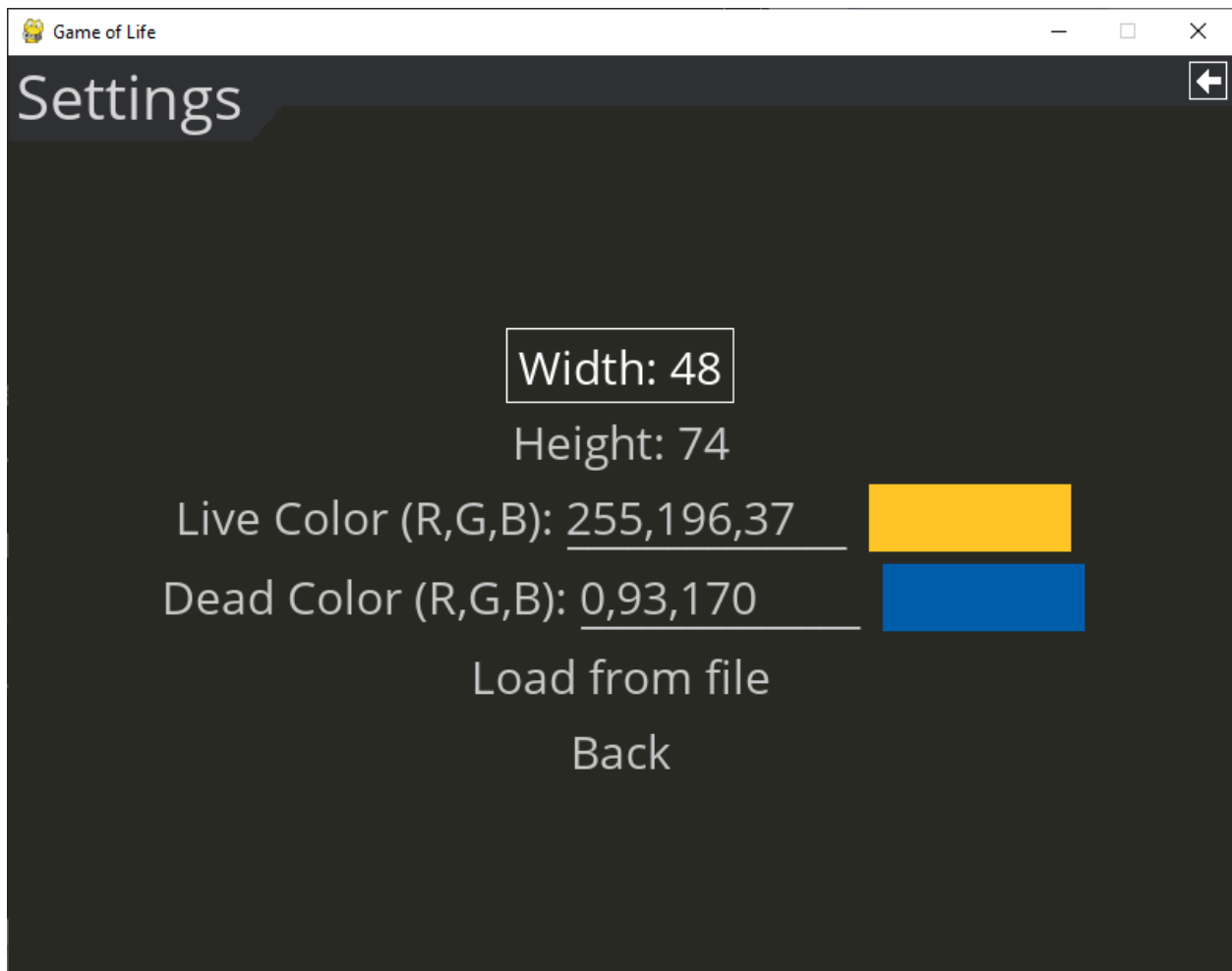


Figure 3.3: settings



Figure 3.4: playing

Chapter 4

Tests

```
1 import unittest
2 import filecmp
3 from cell_map import cellmap
4
5 class TestCellMap(unittest.TestCase):
6     # empty cellmap will not change in the next generation
7     def test_empty(self):
8         current_map = cellmap(3, 3, file='tests/empty.txt')
9         next_map = current_map.next_generation()
10        next_map.write_to_file('tests/next.txt')
11        self.assertTrue(filecmp.cmp('tests/empty.txt', 'tests/next.txt'))
12
13    # cellmap with a 2x2 square will not change in the next generation
14    def test_square(self):
15        current_map = cellmap(3, 3, file='tests/square.txt')
16        next_map = current_map.next_generation()
17        next_map.write_to_file('tests/next.txt')
18        self.assertTrue(filecmp.cmp('tests/square.txt', 'tests/next.txt'))
19
20    # cellmap with a 1x3 horizontal line will change to a 3x1 vertical
    line
21    def test_horizontal(self):
22        current_map = cellmap(3, 3, file='tests/horizontal.txt')
23        next_map = current_map.next_generation()
24        next_map.write_to_file('tests/next.txt')
25        self.assertTrue(filecmp.cmp('tests/vertical.txt', 'tests/next.txt'))
26
27    # cellmap with a 3x1 vertical line will change to a 1x3 horizontal
    line
28    def test_vertical(self):
29        current_map = cellmap(3, 3, file='tests/vertical.txt')
30        next_map = current_map.next_generation()
31        next_map.write_to_file('tests/next.txt')
32        self.assertTrue(filecmp.cmp('tests/horizontal.txt', 'tests/next.
    txt'))
33
34    # a glider in the top left corner of a 5x5 matrix
```

```

35     # will glide down to the bottom right corner and become a square
36     def test_glider(self):
37         current_map = cellmap(5,5, file='tests/glider0.txt')
38         next_map = current_map.next_generation()
39         next_map.write_to_file('tests/next.txt')
40         for i in range(1, 11):
41             self.assertTrue(filecmp.cmp(f'tests/glider{i}.txt', 'tests/
next.txt'))
42             next_map = next_map.next_generation()
43             next_map.write_to_file('tests/next.txt')
44         self.assertTrue(filecmp.cmp('tests/square2.txt', 'tests/next.txt')
)
45
46 unittest.main()

```

Listing 4.1: cell_map_tests.py

```

1 3
2 3
3 000
4 000
5 000

```

Listing 4.2: empty.txt

```

1 5
2 5
3 01000
4 00100
5 11100
6 00000
7 00000

```

Listing 4.3: glider0.txt

```

1 5
2 5
3 00000
4 10100
5 01100
6 01000
7 00000

```

Listing 4.4: glider1.txt

```

1 5
2 5
3 00000
4 00100
5 10100
6 01100
7 00000

```

Listing 4.5: glider2.txt

glider3 to glider10 omitted for brevity

```
1 3
2 3
3 000
4 111
5 000
```

Listing 4.6: horizontal.txt

```
1 3
2 3
3 000
4 110
5 110
```

Listing 4.7: square.txt

```
1 5
2 5
3 00000
4 00000
5 00000
6 00011
7 00011
```

Listing 4.8: square2.txt

```
1 3
2 3
3 010
4 010
5 010
```

Listing 4.9: vertical.txt

Chapter 5

Lessons Learned

The logic for evaluating the game is very simple and straightforward to implement. However, for large grid sizes, it becomes very slow very quickly, since the number of cells grows quadratically as the number of rows and columns grows linearly. There are actually quite a large number of optimizations, many of them detailed in *Michael Abrash's Graphics Programming Black Book*, but many of the ones listed there are specialized for assembly and/or specialized memory access, which unfortunately is either very difficult or impossible to do in Python.[1] One of the optimizations I did implement, however, was keeping track of what cells changed and only drawing those rather than drawing every cell every generation. Keeping track of the changed cells also came in handy when I needed to figure out when the cells reached steady state, since that only happens when the number of changed cells becomes 0. Back to the lesson learned, I guess the big lesson here is that if I want to write a very optimized game of life, I would need to write it in a lower level language like C, C++, Rust, etc.

The second big lesson I learned was that the presentation and user interface is as hard if not harder than the core logic and evaluation code of the game. Creating the menus, buttons, and colors look half decent took about five times the effort and time as the core logic.

The third lesson was that I learned a lot of new ways to do things, since it was my first time using Pygame and the Python unit testing framework. I also haven't written a long report in L^AT_EX, so learning how to do bibliography and table of contents was pretty cool.

Chapter 6

Bibliography

- [1] M. Abrash. *Michael Abrash's Graphics Programming Black Book*. Coriolis, 1997.
- [2] Colm Mulcahy. The top 10 martin gardner scientific american articles. <https://blogs.scientificamerican.com/guest-blog/the-top-10-martin-gardner-scientific-american-articles/>, October 2014.
- [3] J J O'Connor and E F Robertson. John horton conway. <https://mathshistory.st-andrews.ac.uk/Biographies/Conway/>, June 2004.
- [4] Paul Rendell. A universal turing machine in conway's game of life. In *2011 International Conference on High Performance Computing & Simulation*, pages 764–772, 2011.
- [5] Pete Shinnars and the Pygame Community. Pygame. <https://pygame.org/>, September 2022.