Game of Life

Team number:

103

Team members:

Yanjuan Li 001497203

Jingyi Cui 001493484

1. Problem Description

- Genetic Algorithm is a method to search the optimal solution by simulating the natural evolution process.
- Game of Life is a cellular automation which imposes fixed rules on the cells and “played” on an infinite two-dimensional discrete grid. Its successive generations are dependent only on the previous generation
- In our project, we use Genetic Algorithm to find the best pattern. Then using this pattern as an input to start our game of life(20*20). After successive generations, we try to find a stable pattern so that the game will last forever.

2.Rules of Game of Life

- Each “live” cell will die if it has fewer than two or more than three neighbors.
- Each “dead” cell will come alive if it has exactly three neighbors.

3.Implementation Details

- Initialization

Initial (“seed”) population: 1,000

Proportion of organisms that survive and breed: 0.5

Maximum number of generations: 10,000

Note: It may take you at least eight minutes to get the result.

3.Implementation Details

- Genotype and Phenotype

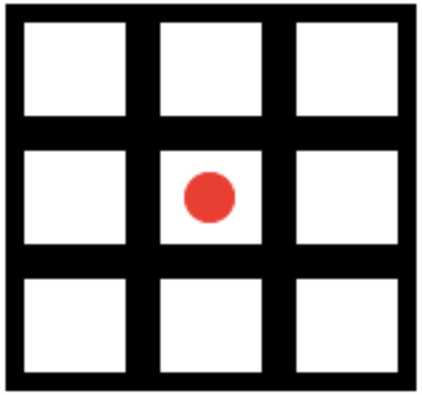
In our project, we use one bit(0 or 1) to represent each gene. If the genotype of the gene is 0, the phenotype of that is dead. If the genotype of the gene is 1, the phenotype of that is live.

About genotype generation, we assign a random integer to each gene, if the random number of that gene less than “cellChanceToLive” (a parameter, set up the chance for cell survival), we will assign 1 to that gene and that gene will live. if the random number of that gene larger than “cellChanceToLive”, we will assign 0 to that gene and that gene will die.

3.Implementation Details

- Next Generation

We consider nine cells as a group, and the next generation of each cell will depend on the status of the other eight cells around it.



3.Implementation Details

- Mutation

We assign a random integer to each gene, if this number is less than “mutationChance” (a parameter, set up the chance for cell mutation), then the status of next generation will be generated randomly (true or false).

- Fitness

It is max generation that our pattern lasts for.

3.Implementation Details

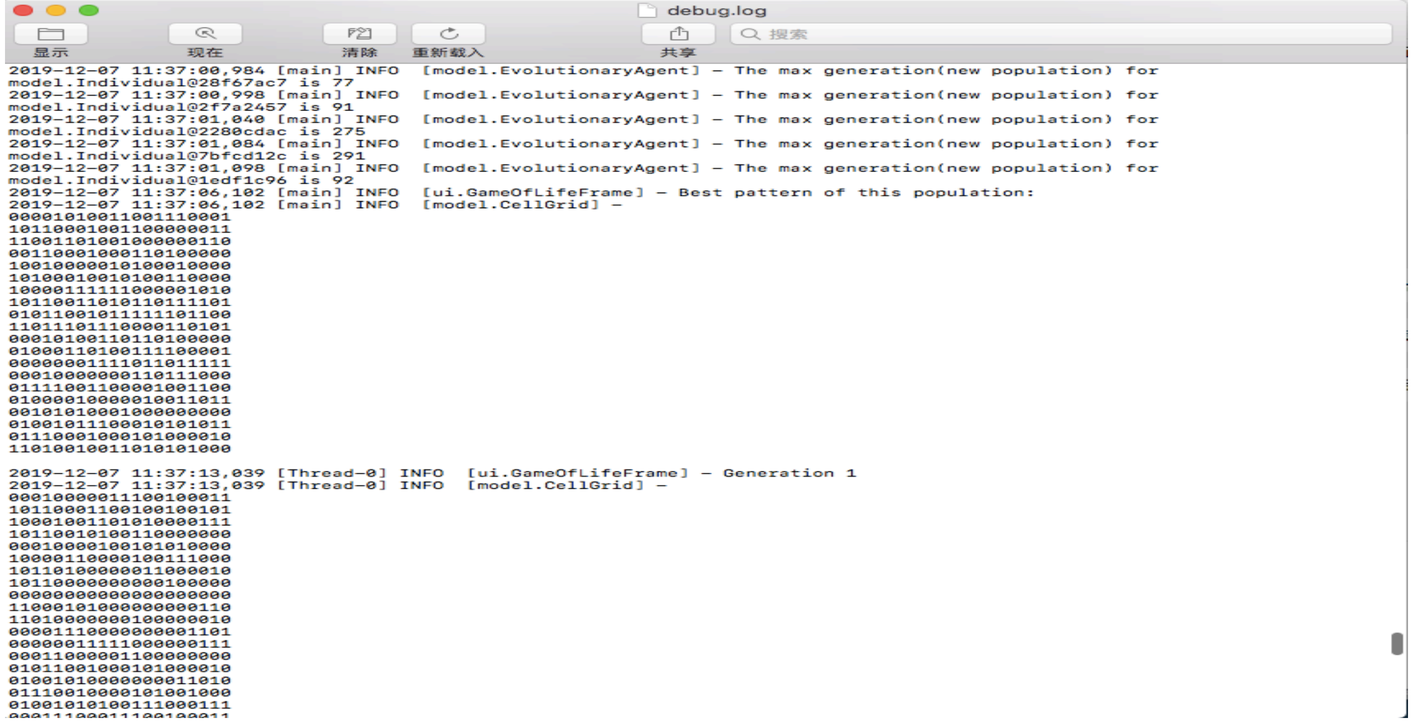
- Survival Function

During our selection, we choose the top20 individuals based on the fitness rules for next new population and the rest part of the new population will be selected randomly.

3.Implementation Details

- Log

We add a logging function to keep track of the progress of the evolution, including the best candidate from the final generation.

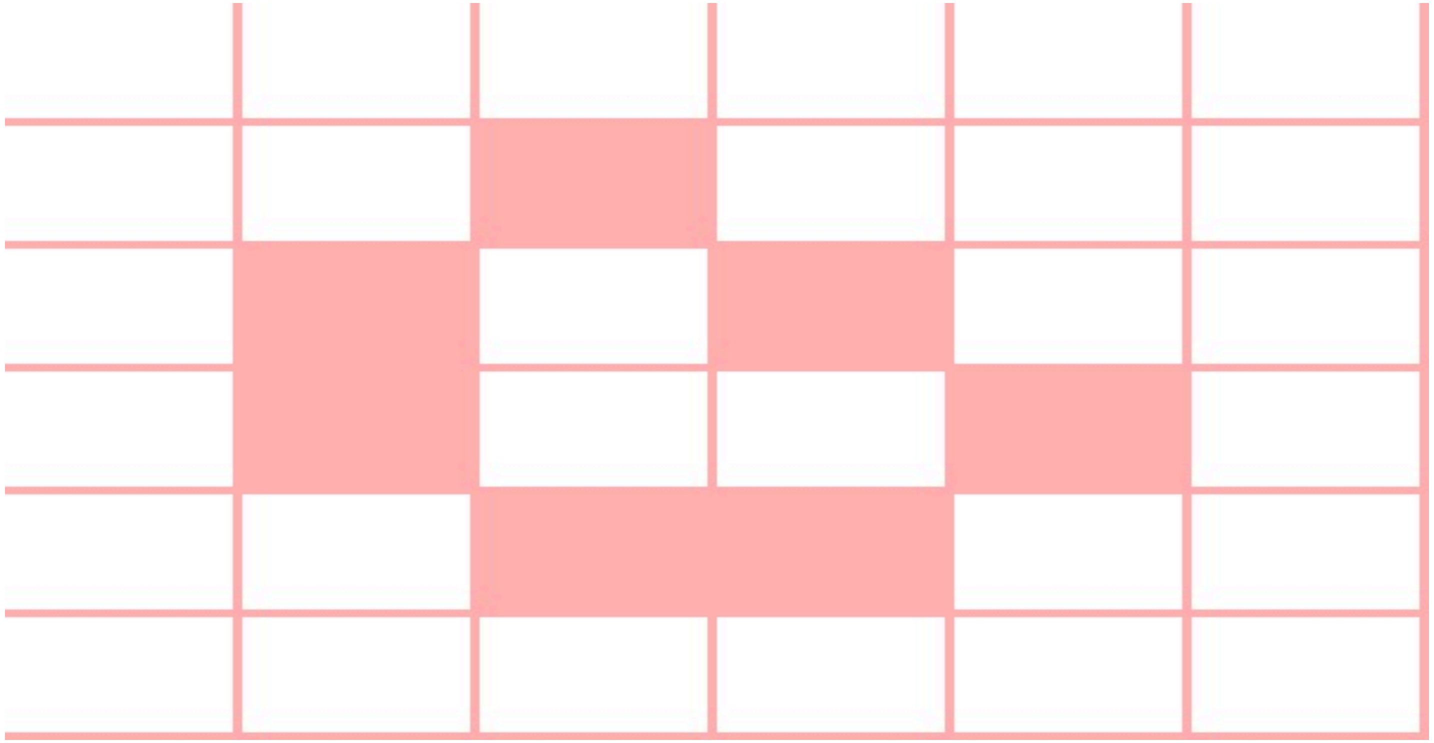


```
2019-12-07 11:37:00,984 [main] INFO  [model.EvolutionaryAgent] - The max generation(new population) for
model.Individual@28f67ac7 is 77
2019-12-07 11:37:00,998 [main] INFO  [model.EvolutionaryAgent] - The max generation(new population) for
model.Individual@2f7a2457 is 91
2019-12-07 11:37:01,040 [main] INFO  [model.EvolutionaryAgent] - The max generation(new population) for
model.Individual@2280cdac is 275
2019-12-07 11:37:01,084 [main] INFO  [model.EvolutionaryAgent] - The max generation(new population) for
model.Individual@7bfcd12c is 291
2019-12-07 11:37:01,098 [main] INFO  [model.EvolutionaryAgent] - The max generation(new population) for
model.Individual@1edf1c96 is 92
2019-12-07 11:37:06,102 [main] INFO  [ui.GameOfLifeFrame] - Best pattern of this population:
2019-12-07 11:37:06,102 [main] INFO  [model.CellGrid] -
00001010011001110001
10110001001100000011
11001101001000000110
00110001000110100000
10010000010100010000
10100010010100110000
10000111111000001010
10110011010110111101
01011001011111101100
11011101110000110101
00010100110110100000
01000110100111100001
00000001111011011111
00010000000110111000
01111001100001001100
01000010000010011011
00101010001000000000
01001011100010101011
01110001000101000010
11010010011010101000

2019-12-07 11:37:13,039 [Thread-0] INFO  [ui.GameOfLifeFrame] - Generation 1
2019-12-07 11:37:13,039 [Thread-0] INFO  [model.CellGrid] -
00010000011100100011
10110001100100100101
10001001101010000111
10110010100110000000
00010000100101010000
10000110000100111000
10110100000011000010
10110000000000100000
00000000000000000000
11000101000000000110
11010000000100000010
00001110000000001101
00000011111000000111
00011000001100000000
01011001000101000010
01001010000000011010
01110010000101001000
01001010100111000111
00011100011100100011
```

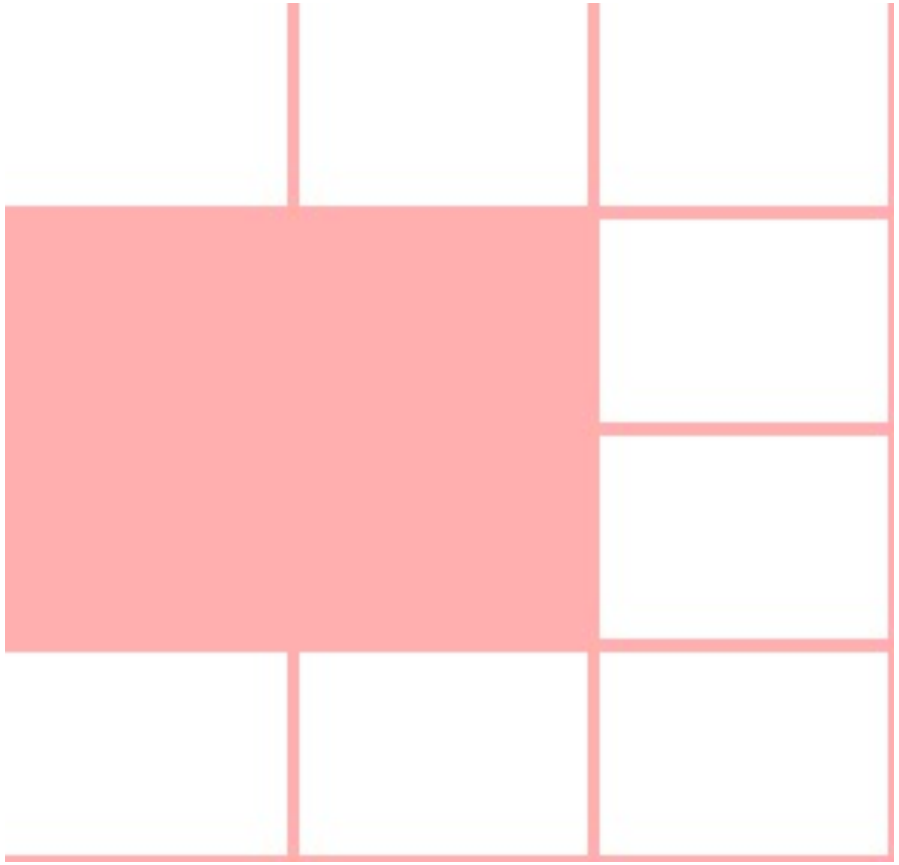
4.Results and Conclusion

- We did many experiments with our project and we found several patterns that can result in a steady state of the game.

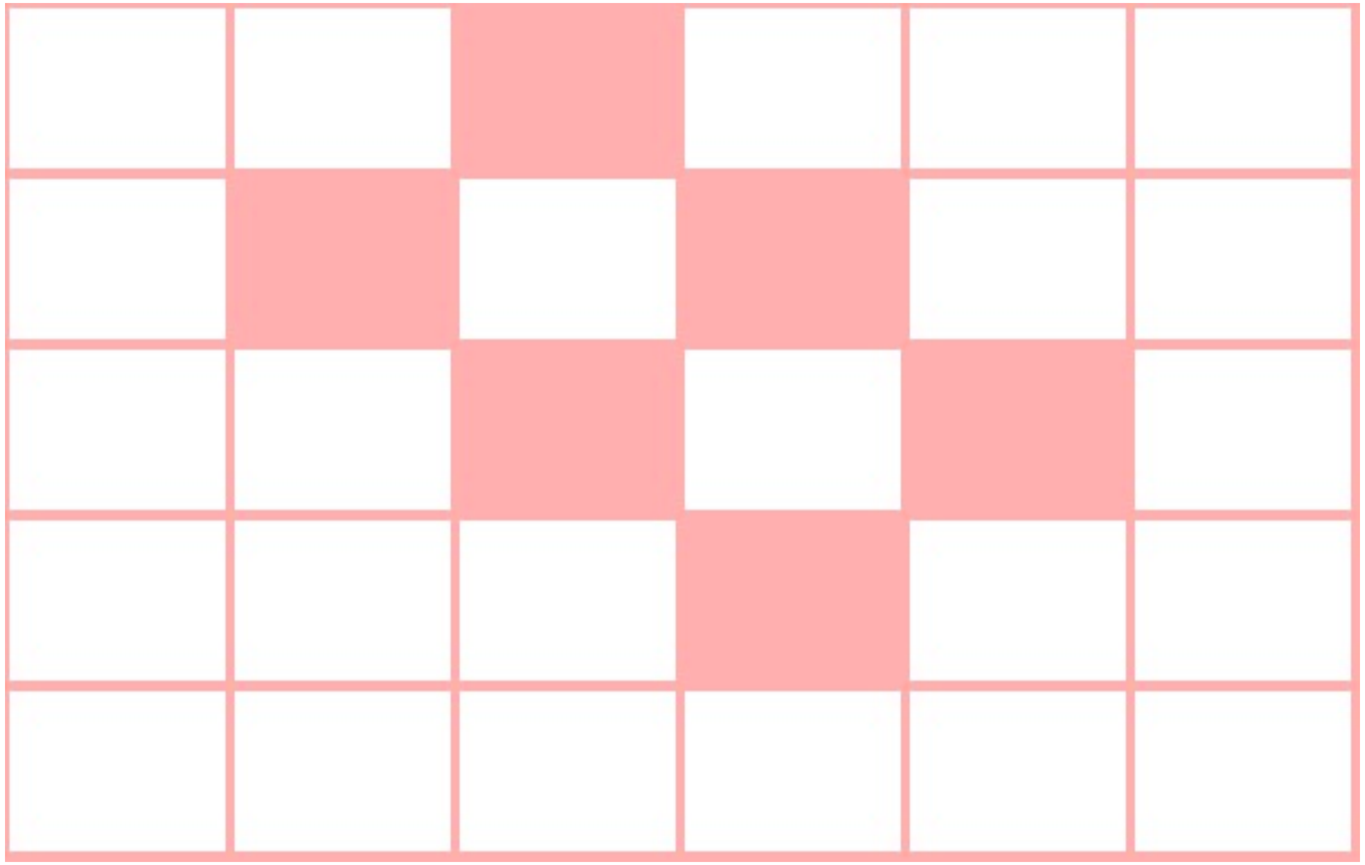


Picture 1

4.Results & Conclusion

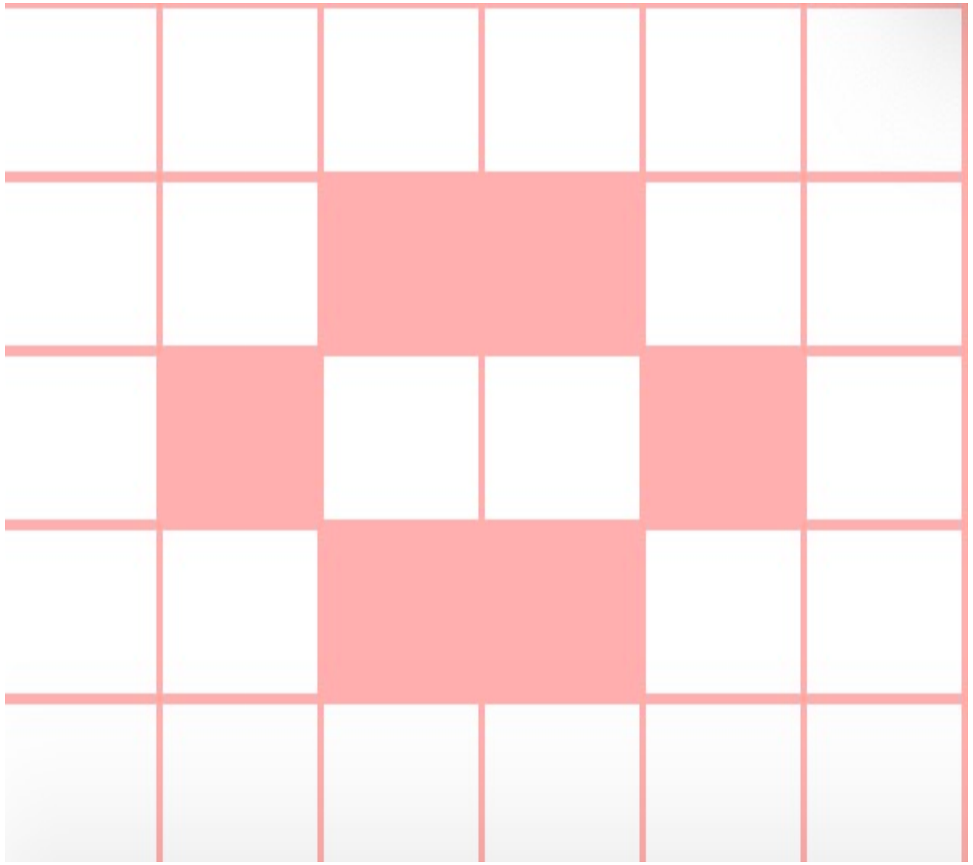


Picture 2

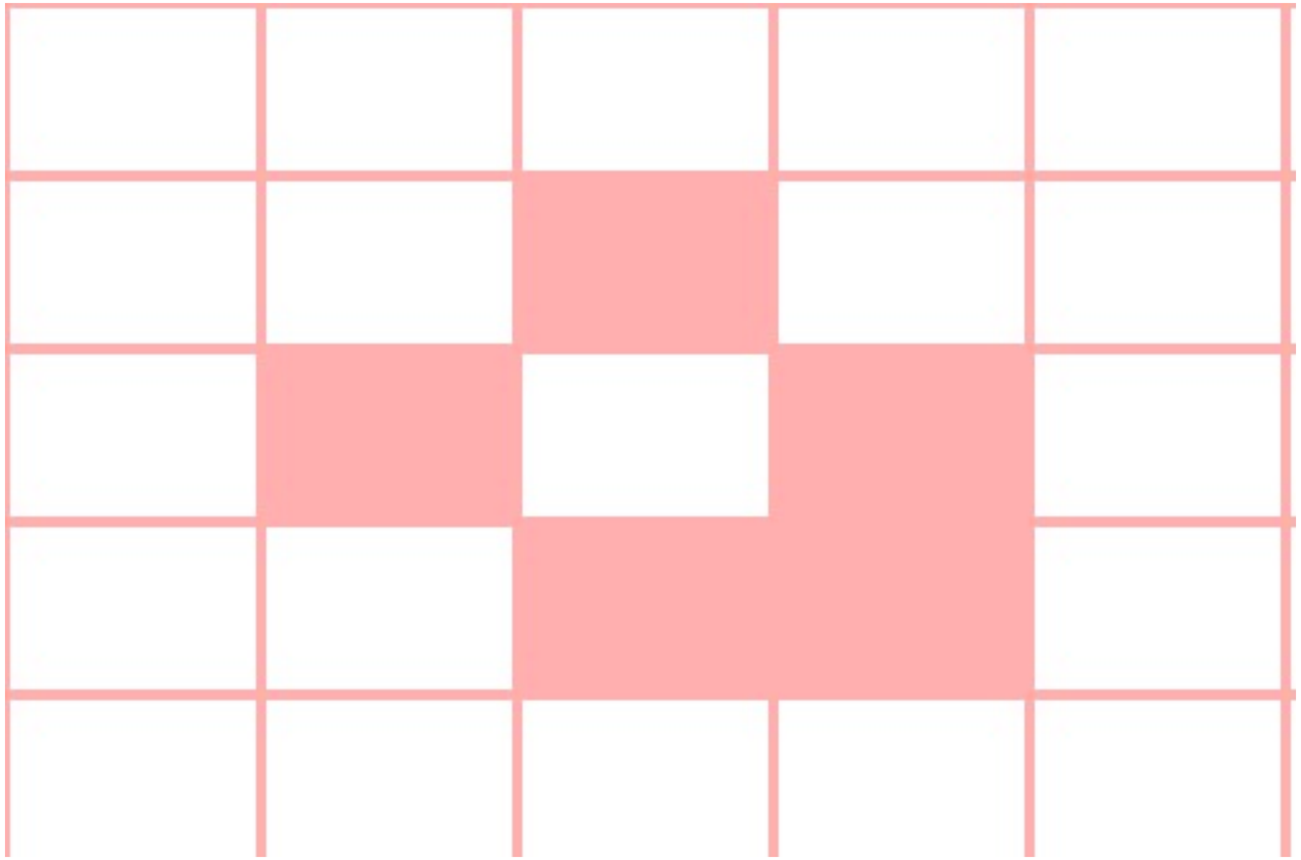


Picture 3

4.Results & Conclusion



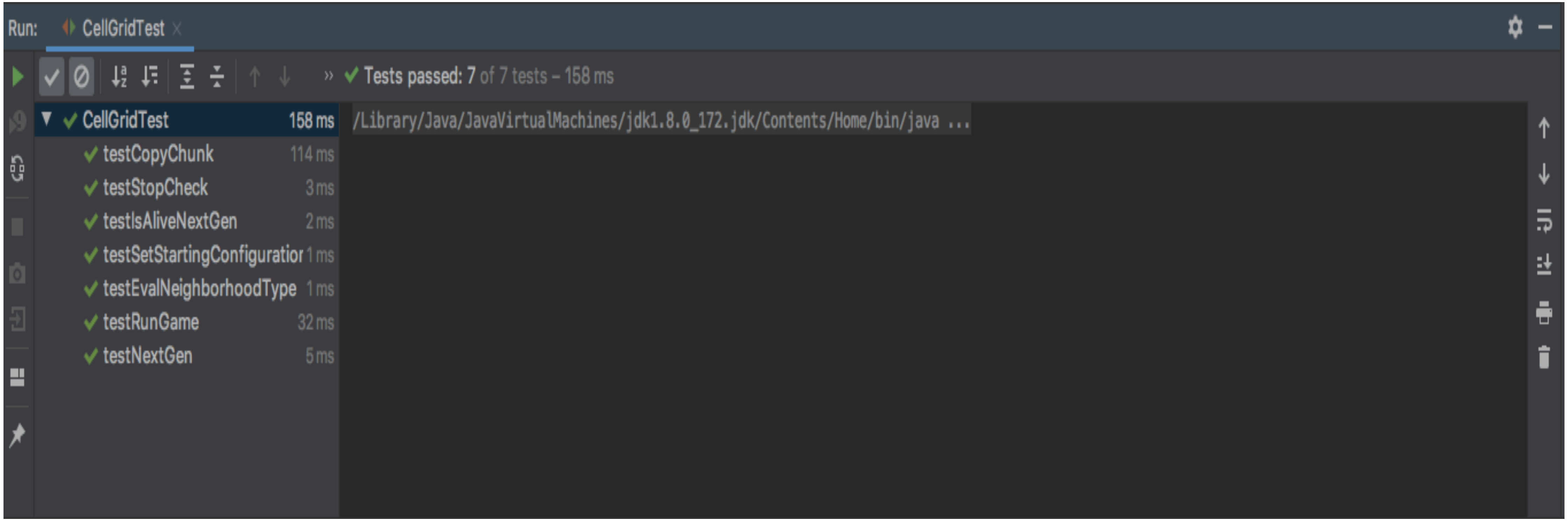
Picture 4



Picture 5

5.Unit Test

- CellGridTest

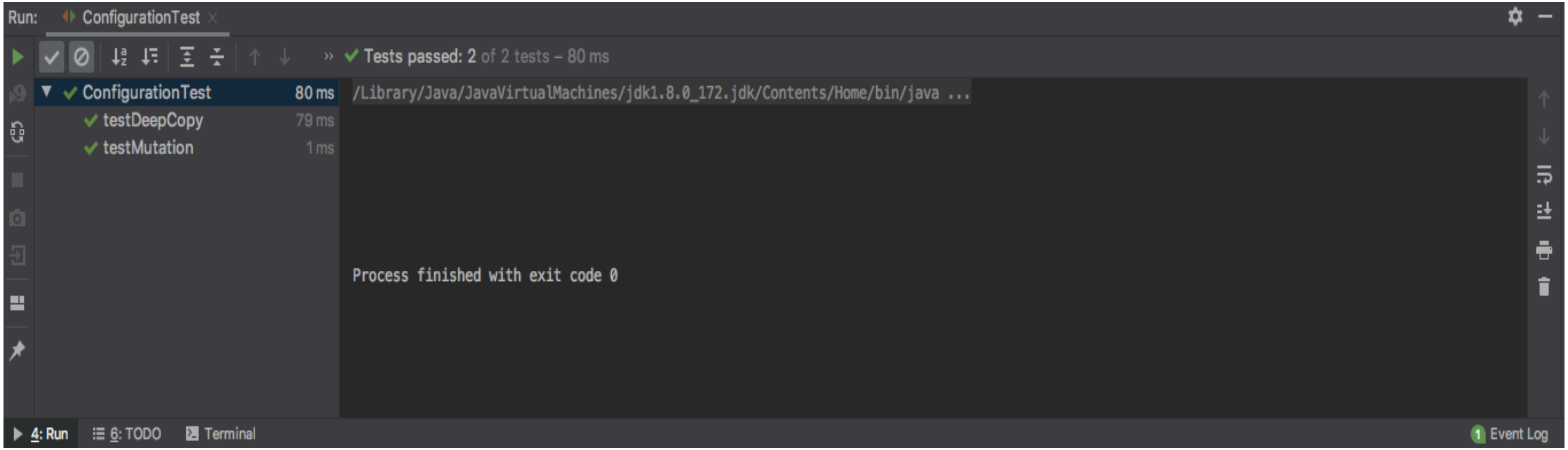


```
Tests passed: 7 of 7 tests – 158 ms

▼ ✓ CellGridTest                          158 ms   /Library/Java/JavaVirtualMachines/jdk1.8.0_172.jdk/Contents/Home/bin/java ...
    ✓ testCopyChunk                       114 ms
    ✓ testStopCheck                         3 ms
    ✓ testIsAliveNextGen                    2 ms
    ✓ testSetStartingConfiguratior          1 ms
    ✓ testEvalNeighborhoodType              1 ms
    ✓ testRunGame                          32 ms
    ✓ testNextGen                           5 ms
```

5.Unit Test

- ConfigurationTest



✓ Tests passed: 2 of 2 tests – 80 ms

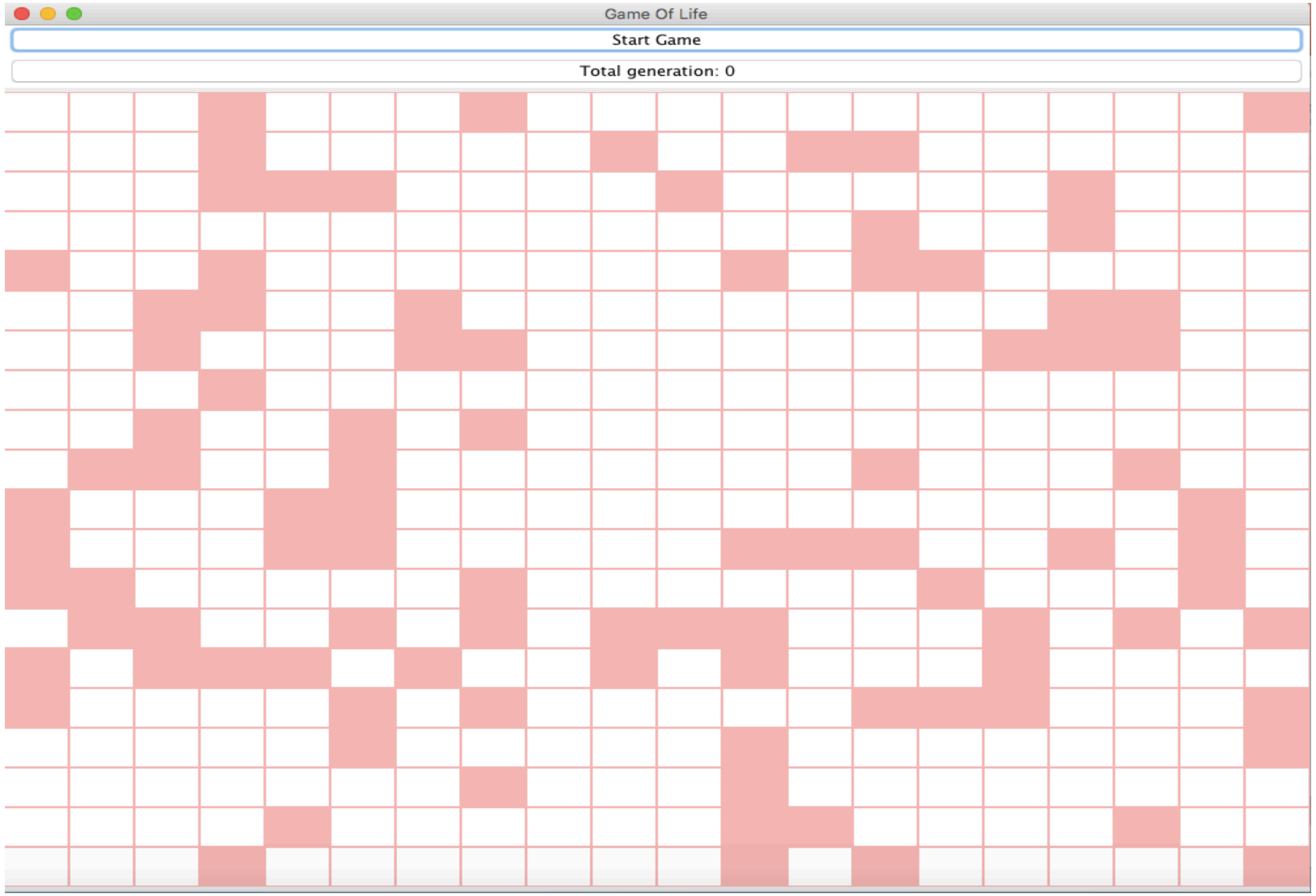
✓ ConfigurationTest 80 ms
✓ testDeepCopy 79 ms
✓ testMutation 1 ms

```
/Library/Java/JavaVirtualMachines/jdk1.8.0_172.jdk/Contents/Home/bin/java ...


Process finished with exit code 0
```

6.Bonus Part

- We build a UI for our project.


Game Of Life
Start Game
Total generation: 0