ISTANBUL TECHNICAL UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

BLG 458E FUNCTIONAL PROGRAMMING

TERM PROJECT

PREPARED BY:

150220096 : CEYDA AKIN

SPRING 2025

Contents

1 Introduction

This project investigates the dynamic behavior of Conway's Game of Life, a well-known cellular automaton that exhibits complex global behavior through simple local rules. The focus of this assignment is on five oscillator patterns introduced in the paper by Brown et al. [1], each demonstrating periodic behavior over a number of generations.

2 Problem Definition

The objective is to implement Conway's Game of Life in Haskell and visualize selected oscillator patterns. An oscillator is a finite configuration that returns to its original state after a fixed number of generations, known as its period. The system must:

- Correctly implement Game of Life update rules.
- Initialize with distinct oscillator patterns.
- Simulate the evolution of each pattern across multiple generations.
- Export each generation as a PNG image for visualization.

3 Background: Game of Life

Conway's Game of Life is based on a grid of cells that can be either alive or dead. At each step, the state of each cell is determined by its eight immediate neighbors, including diagonals. The update rules are as follows:

- A dead cell with exactly three live neighbors becomes alive.
- A live cell with two or three live neighbors survives; otherwise, it dies.

These simple rules can lead to emergent complexity, including stationary patterns, moving objects (gliders), and oscillators. Oscillators are particularly valuable in understanding the structured behavior of the system.

4 Selected Oscillators

The following five oscillator patterns were selected from Brown et al. [1]:

- **Blinker** – Period 2
- **Toad** – Period 2

- **Beacon** – Period 2
- **Pulsar** – Period 3
- **Pentadecathlon** – Period 15

Each pattern was simulated for **period + 1** iterations to fully demonstrate its cyclical behavior.

5 Implementation in Haskell

The project was implemented using functional programming paradigms in Haskell. Key principles include:

- Use of immutable data structures and pure functions.
- Recursive definitions and lazy evaluation.
- Modular design: separate functions for pattern generation and evolution logic.
- PNG generation using the `JuicyPixels` library.

5.1 Simulation Logic

Each cell is updated based on its eight neighbors according to the Game of Life rules. A functional approach is used to compute the next generation as a pure transformation of the current state.

5.2 Pattern Initialization

Oscillator patterns are defined as 2D lists of integers and embedded in the center of the simulation grid using a custom placement function. Each pattern is encapsulated in its own function.

5.3 Image Generation

Each iteration of each pattern is rendered as a PNG image using the JuicyPixels library. Images are saved as `frameX.png`, where `X` represents the generation number.

6 Results

Each oscillator pattern was simulated for **period + 1** generations to visualize both the initial state and one full cycle. The number of iterations for each pattern is as follows:

- Blinker: 3 iterations
- Toad: 3 iterations
- Beacon: 3 iterations
- Pulsar: 4 iterations
- Pentadecathlon: 16 iterations

Example Frames

Figure 1: Blinker – Initial State

Figure 2: Blinker – Final State (Iteration 2)

Figure 3: Pentadecathlon – Initial State

Figure 4: Pentadecathlon – Final State (Iteration 15)

7 Discussion

Oscillators demonstrate how simple deterministic rules can lead to highly structured behavior. This project has shown:

- The relationship between oscillator period and the number of visible states.
- The importance of precise pattern placement to preserve symmetry and behavior.
- That Haskell's declarative and immutable nature is well-suited for such simulations.

8 Conclusion

This project successfully:

- Simulated five well-known oscillator patterns from academic literature.
- Visualized their behavior over time.
- Applied functional programming concepts to model and transform grid-based state.

Conway's Game of Life remains a powerful illustration of how local interactions can produce global order. The use of Haskell allowed for clean, maintainable, and expressive modeling of this emergent behavior.

REFERENCES

[1] Nico Brown, Carson Cheng, Tanner Jacobi, Maia Karpovich, Matthias Merzenich, David Raucci, and Mitchell Riley. *Conway's Game of Life is Omniperiodic*. arXiv preprint arXiv:2312.02799, 2023.

[2] Martin Gardner. *Mathematical Games: On Cellular Automata, Self-Reproduction, the Garden of Eden and the Game Life*. Scientific American, Vol. 224, pp.112–117, 1971.