Functional Project Proposal: *Game of Life*

Bayard Walsh

October 2023

1 Introduction

For the final project I am proposing to build a recreation of Conway's Game of Life. This simulation was one of the initial creations that inspired me to begin programming, and I haven't had the opportunity to recreate it (yet!). What better language to do it in than Haskell?

2 Easy

The fundamental behavior of the game should be relatively straightforward to program. At its core, this is the game logic that decides if a given cell should die, maintain or repopulate for a given epoch, based on the cells surrounding it, as described by Conway in the original game. In addition to coding the passive cell behavior, I want to create a function that populates a given amount of cells on the board. Options for this could be to follow a certain distribution (say create one cell every 3 rows), or a random distribution (say $\frac{1}{5}$ overall cells become populated). For the random distribution, I could use the `randomR` library from **PA7**. Additionally, I also want to give the user control over manually updating the game through the command line or letting it automatically update on an epoch timer, and giving the user the control to alternate between the two. The logic for this is the clock could go too fast and result in the game board dying out or reaching equilibrium before the user can properly interact with it. Therefore, manual updating will function as a pause. I can implement a terminal command line-focused MVC, where the user can see a terminal visualization of the board, while having control over generating more cells, (randomly, specifically, in groups, or individually), or just hitting the `next` command for however many epochs and letting the cells continue with their automatic behavior.

3 Medium

A slightly harder challenge is the visual representation of the image. While we have seen visual output with both png files and the command line in homework, I found that generating larger png files took very long to load and I don't want

the illusion of the game animation to be lost by buffering, so I plan on visualizing the board in the command line instead. I want to give the user control over the size of the board (and ideally the option to generate boards big enough to fill the terminal) and the size of each cell, for example, a 16x16 board with either a 1x1 or 2x2 sized pixel based on user choice. These commands will be from the command line and an introduction would give the user the needed information about controlling the game from initialization. With different-sized pixels and game boards, there will be some difficulty in manipulating the terminal output so the board is consistent. Also, internally, I want to use a more efficient way of representing the 2D game board than the default choice of a list of lists, so I will use the following library setup and data type: `import Data.Array.Repa` and `type Grid a = Array U DIM2 a`. This grid type will help with larger test cases in terms of efficiency compared to the list of lists. I also will implement the `State` Monad as a way to maintain the board for the game. Because I am updating cells based on the previous state for the next state, in the updating process I will have to have two copies of the game board, and the `State` Monad will help me achieve this through using `get` and `put` calls to maintain two copies of the board with ease.

4 Challenge

The first and biggest challenge is speed, especially for larger boards. I see the biggest potential bottleneck as the automatic updating phase, and initially, I plan to check each cell and update it based on its surroundings, however, if time permits I intend to keep a hash map of only the living cells (to speed up the updating query), as those are the only ones that impact the board. Because this is nonessential to the game functionality and the main upside is speed, I will save this for a later stage implementation. Another challenging aspect would be to code the various Game of Life configurations discovered by users (which move across the screen and repeat their patterns), such as gliders, tubs, and beehives (to name a few), and positioning them in a set spot on the board based on a command line input. While these are not inherently difficult once I have the functionality to set a group of cells to live, it requires the game to already be complete, meaning that I would add this functionality last to the code. Another hard, but essential task will be testing my code, which I should be able to do holistically, but I want to automate. A way of achieving this will be to run various commands and states on the board and then see how the board develops over a set amount of epochs. Because the game is deterministic, this approach will work for testing. Finally, I want to give the option to remove sections inside the board as potential population spots or modify the board beyond its square shape. While the game is traditionally a square, giving the option to create nonsquare boards would offer more interesting gameplay for the user. Missing holes would also add an interesting aspect to updating cells, and I could implement a Maybe Monad case to check if the surrounding area is viable for a cell, and return nothing if not.