Aaron Schurman
Alec de Graaf
ECEN 2703
4/25/23

Final Project Report

I. Introduction

For our final project we created a version of Conway's Game of Life with added functionality for the user. Created in 1970 by British mathematician John Horton Conway, this game is "played" by the user creating an initial state for the machine, which then uses specific rules to generate the next frames. The game is Turing complete, making it theoretically as powerful as a computer with unlimited memory and no time constraints, although we would highly recommend *not* trying to treat our code as one with unlimited memory and no time constraints. The game operates on a set of rules and has interesting emergent properties as the population of cells becomes more complex. These properties could be better investigated by using a SAT solver to determine what variables lead to these properties.

II. Motivation

The motivation behind creating this game was from its history as one of the first games used on a computer and relation to ideas learned in this class. At first the scope of our project was just to recreate the game itself. This was expedited by the reference to an existing project by Jubnzv found here https://github.com/jubnzv/gol-solver. This code uses a SAT solver z3 to find the next step in a deterministic sequence of events based on an initial frame. "This is like using a sledgehammer to crack a peanut" - The Greatest Discrete Professor. A better use of the SAT solver would be to work backwards from a desired end state to find one of many possible initial states. Therefore we decided to put certain constraints on the initial state, such as the number of cells allowed to be alive, and constrain the desired end state, using the SAT solver to find an initial state that satisfies the problem.

III. Scope

After building up the base game in python we began to investigate different ways to increase the scope of the project. We first added functionality to give it an initial and final frame and see whether the final frame could be reached from the first in a given number of frames. It is important to note here that no algorithm or program exists that would be able to tell if a given initial pattern will eventually result in a later pattern. This would solve the Halting problem and

is therefore impossible. However given a small number of steps to constrain the problem we are able to create a seemingly accurate test for whether an input frame would be able to get to the output frame in the desired steps. Although interesting, we felt that we could go further.

Because of the deterministic nature of the program the beginning frame cannot be fully enumerated. Instead the solver must be given constraints for a possible start frame. Thus the number of alive cells in the start frame and the amounts of transitions before the end frame can be constrained by the user. The program then determines if it is possible to complete the final state and then shows the step-by-step process to achieve it. This will then allow the user to investigate emergent properties and narrow down the necessary variables needed for them to emerge.

IV. Investigation

A. Basic Operational Testing

During the investigation it was found that the program works best when using a small grid space. During our testing and implementation of the program we chose to use a 5x5 grid for it. This resulted in fast instantaneous runtimes for the code when a small number of transitions were requested. During testing the largest grid we used was 10x10. This size proved to be near unusable as a run with a simple final state, few alive cells at the start, and five or fewer frames used would still result in a runtime of multiple minutes each. It is posited that this is because there is some exponential time complexity when using the SAT solver.

B. Emergent Properties Testing

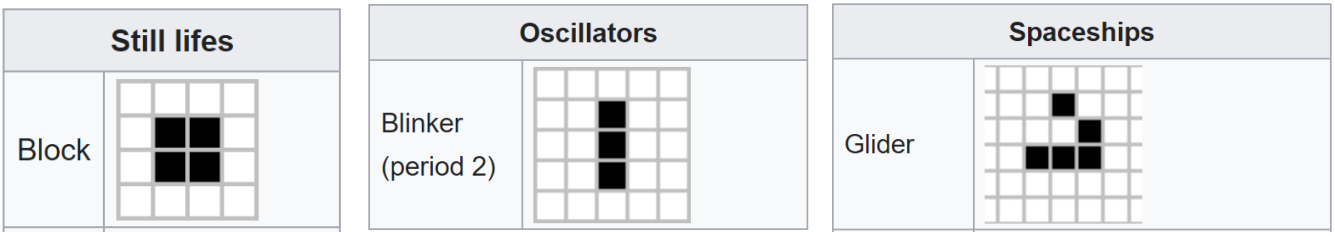

Figure 1 - Simple Emergent Properties of The Game of Life

1. Still Lifes

The most simple element that emerges out of the game of life is still life. Seen in figure 1. There are multiple forms that this element can take but the square was the one chosen for testing. Using an input file as seen in Index A and 4 alive cells requested at the start, our assumption was the SAT solver would give a solution in a minimum number of steps or 0 steps and make the initial frame the same as the final frame. This was not the case and the program took 3 frames to get to the solution. This can be seen in figure 2. Multiple runs with the same input gave dramatically different results along with different run times. This can be seen in figure 3. We

believe this behavior has to do with how the SAT solver only responds to the constraints set and has no reason to solve the problem in a fewer amount of frames or the same way every time.

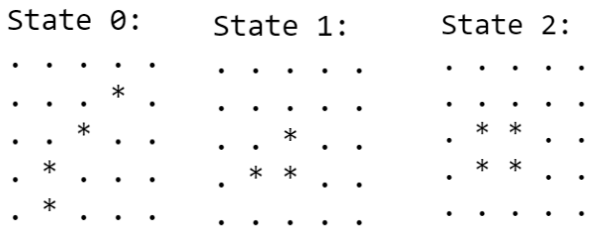

Figure 2 - First 3 frames of the output from first still life test

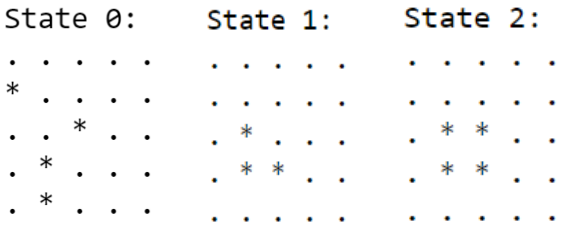

Figure 3 - First 3 frames of the output from the second still life test

2. Oscillators

The second behavior investigated was how the SAT solver dealt with oscillators. As seen in figure 1. Given the input file seen in Index B a 10 requested alive cells in the initial frame the output seen in Figure 4 is achieved.

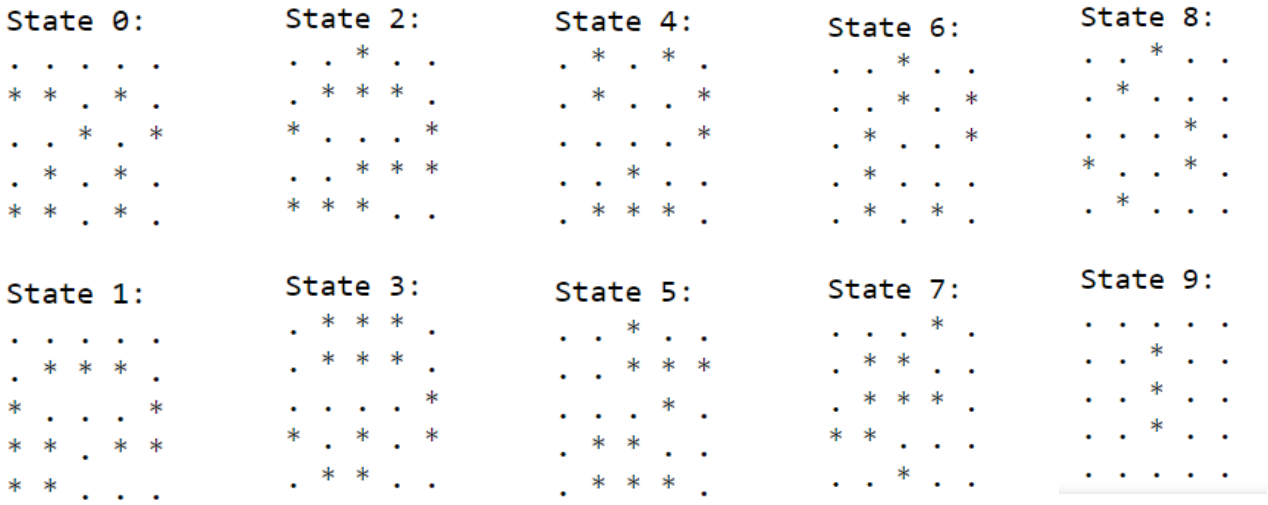

Figure 4 - All 10 frames of the output from the oscillator test

As seen in Figure 4 the program was able to accurately get to the final state from the given inputs. As expected it took longer to get to a more complex state. Interestingly on further runs of the same input with 5 fewer transitions requested the program is able to come up with a solution much faster than previously. This again shows how z3 will spend longer in states of chaos then finding an oscillation that leads to the requested state of the oscillator.

3. Spaceships

Lastly the most dynamic emergent property was tested: the Spaceship. The input was a glider set at the south east corner of the grid in hopes of seeing some glider movement, and 8 cells requested alive at the start. The input can be seen in Index C. The output can be seen in Figure 5.

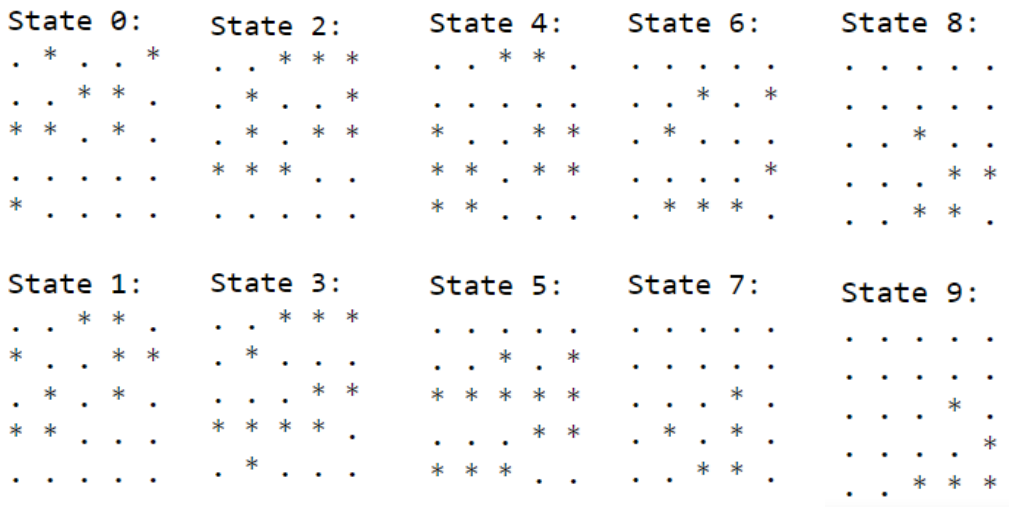

Figure 5 - All 10 frames of the output from the glider test

It is interesting to look at the last 3 frames of the output showing the glider going through one half of a full movement.

V. Achievements

It is clear that the program is able to fulfill the goals enumerated in the scope of the project. It can accurately determine an input that will lead to a desired output frame when both the input and steps are constrained. This is a good starting point. Future iterations of this program could implement defined positions of live and dead cells in each frame making an application that transitions between images at a large scale. This could have interesting applications in media, design, and investigations into how and why SAT solvers make the decisions they do.

Much further investigation must be done into how the SAT solver is actually getting to a solution and why it might choose some inputs over others. However, this was an incredibly interesting challenge that led to a lot of learning discoveries, the least of which being just how cool SAT solvers really are.

VI. Index

A. Input file contents for first emergent property test

5 5 10
. . . . .
. . . . .
. * * . .
. * * . .
. . . . .

This .txt file represents this grid and specifies 10 requested steps from the initial frame to the final frame

B. Input file for second emergent property test

5 5 10
. . . . .
. . * . .
. . * . .
. . * . .
. . . . .

C. Input file for third emergent property test

5 5 10
. . . . .
. . . . .
. . . * .
. . . . *
. . * * *