Group 28 CS2006 Haskell 2 Report

Overview

This practical required us to implement the game Gomoku (aka five in a row) in Haskell, including a GUI and AI player.

Usage

To use the program:

- cd into the src directory
- install cabal directory, random, and gloss if necessary
- run “cabal build” to build the project
- run “./gomoku” to run the program (by default on a 6x6 grid with a target size of 3)
    - dist/build/gomoku – arguments are either board size and target or save file

Functionality

I will now describe how to use the features of our program.

- Basic specification: see “Usage”
- AI – press “A” at the GUI start page to play against the AI (or “P” as PVP)
- Left click to place moves
- Right-click to undo moves
- Press p to pause/unpause the move time limit (timer)
- Use the up/down keys to change the board size
- Use the right/left keys to change the target size
- Press s to save the current game to saveME.txt
    - When loading a game, when starting Gomoku, give the argument saveME.txt
        - “./gomoku saveME.txt”

Design

The overall structure of the project is largely unchanged from the given start code. The only difference in this aspect is the splitting of the AI module into two separate nodules; one handles the logical process by which the AI functions (tree generation and minimax branch selection), and the second provides a layer of abstraction to allow the logic of the AI to be written at a high level (in the former).

We did not further change the module structure of the practical as we deemed the given structure and extent of modularisation to be sufficient and reasonable to allow maintainability, error identification, the ability to allow logical understanding of the control flow of the program, task distribution and parallel development (of different modules simultaneously by different people), and to make code reusable where possible.

Implementation

The main method takes in the arguments for the creation of the starting world, detecting if these are present, for the size and target of the board or a saved game file to be read in. This then runs the playIO method from gloss so that it can handle input and output.

Board representation

The Board.hs file contains the data types for the board and world and the main implementation of the rules, accessed from the checkWon method, which works with boards of any size. It uses a series of checks over rows, columns, and diagonals that use filter and sort operations to analyse pieces on the board. The lists of pieces generated are checked to find if there are any adjacent pieces, and if so if there are enough to win the game (and without there being too many in a row). There are also rules for three and three, again for any target size, adding extra stipulations onto the black player.

GUI

Draw.hs is designed to draw a title screen before play begins so that extra options can be selected. The board is drawn to a specific size; even when there are more pieces on the board, it will stay a constant width as the lines will simply be drawn closer together. We believe this improves the aesthetic appeal of the GUI. After the board is drawn, pieces can be added to chosen positions correctly with mouse clicks. The pieces drawn are scaled to match the size of the board.

Input.hs is very similar to the board when it comes to scaling because the positions it detects are matched exactly to the lines displayed by the board, which is why they can return the correct piece position to be added to the world. Input.hs also implements the ability to undo moves with right clicks, removing the most recent move, as well as growing and shrinking both size and target, as well as saving and pausing the game.

The updateWorld method in AILogic.hs is dependent on the type of game being played and returns the correct alterations to the world on each update. This is particularly useful for timing, as since the refresh rate is known, a timer can be created by decrementing a value in the world representation every time the game is refreshed. This is then displayed by draw and can be seen counting down on the screen. On a timeout, the players will switch or will stop counting if the game state is paused.

AI

The AI logic is uncomplicated, just being a combination of abstracted methods used and combining to form high-level logic (including operations on lists of positions as potential moves, for instance). The process of it is a minimax tree searching future moves and evaluating them based on their scores in the evaluate method. The maximum depth to look forward to in the tree and the number of (the best possible) moves on each level of the tree to examine are adjustable from constants set in AILogic.hs.

AIImplementation.hs contains methods to perform the abstract operations on lists of positions, boards, etc. The operations needed to run the AI are broken down into as many small methods as possible to allow for a logical flow for the AI, to allow maintenance, understanding, etc. The getMaxEvalScoreForMove method is probably the most important in the module; it controls the majority of tree traversal and move assessment to find the maximum possible evaluation score for each possible initial move (so that with the largest can then be played by the AI).As I have mentioned, the methods are as broken up and small as possible, so the remainder of them in the module are fairly simple to understand when read.

I would argue that equally important as the methods implementing the minimax tree traversal and node selection is the evaluation method for the AI. It checks to see if a move will cause a player to win immediately or at the next move, then returning a positive or negative fraction of the maximum possible score, depending on which player will win and if they would win at that or the next move. Otherwise, it will return the sum of: (num * l) * [2 ^ (l – 1)], where 0 < l < target length, and num is the number of times a given colour has a combination of a given length.

Provenance

Source material

- studres start code (for this practical)
- Gloss examples found online

Contributions

taeh

- Adapting start code from studres and setting up the structure of practical
- Initial prototype implementations of several modules which were scrapped due to them not working (and redone by other team members)
- Implementing the evaluate method in Board, and the AI
- Some refactors and adjustments to feb7's code during AI implementation to allow compatibility
- Most contents and control of the group report

feb7

- Refactoring and/or rewriting initial prototype code from taeh to complete the basic specification (with the exception of the AI)
- Implementing command-line inputs to choose board and target size
- Implementing saving and loading sgf formats
- Implementing time limits for moves and pausing/unpausing them
- Implementing the Undo functionality to rollback moves
- Implemented the key presses to alter the board and target size
- Implemented alternate rules for three and three

Testing and Examples

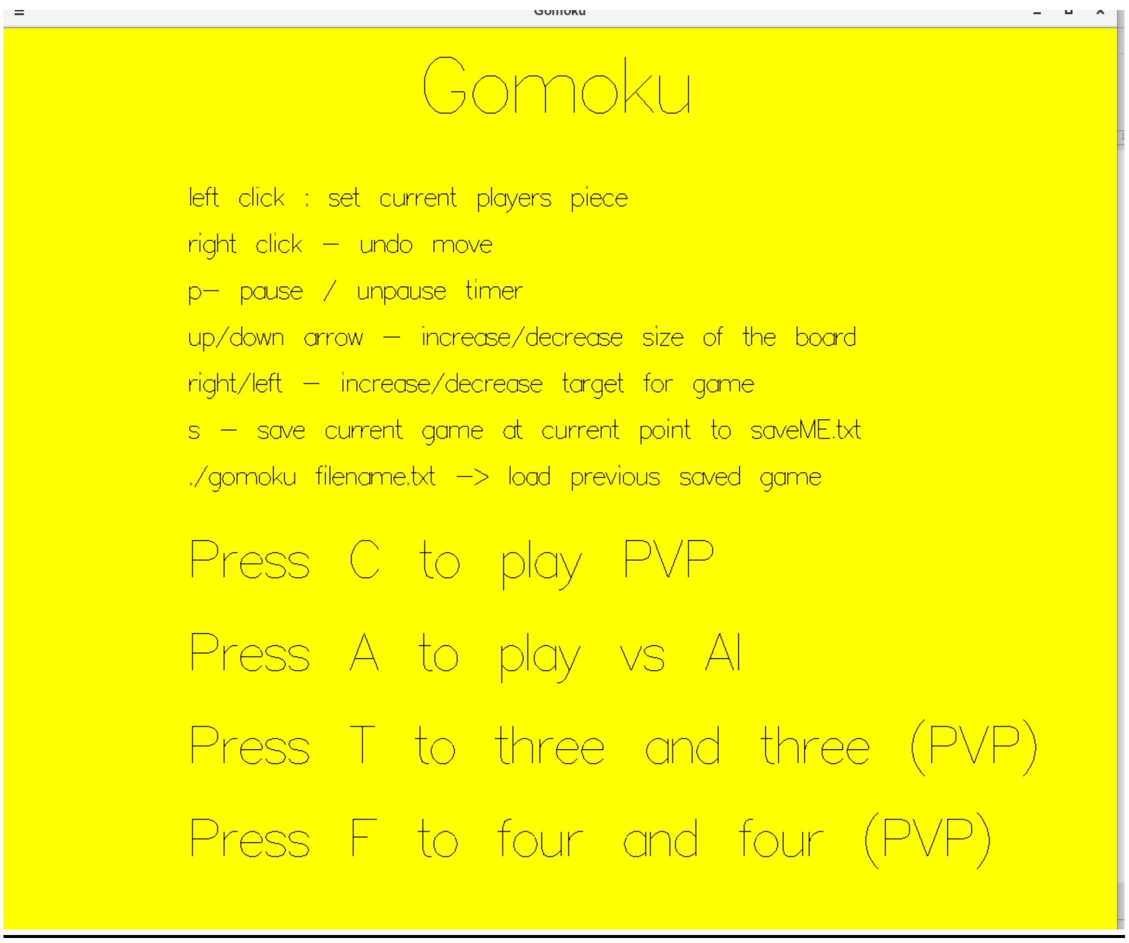
Testing

The program was largely tested with trial and error. For example, we considered the time taken by the AI to choose and execute a move, and the quality of that move. We repeatedly tested and then adjusted the AI, and in this manner, we were able to ensure it performed reasonably as a result. Specifically, we tested that it always (or the vast majority of the time) will win with a target of 3 and that at various board and target sizes it continues to execute moves reasonably sensibly and rapidly.

A similar process followed for the other features of the project. Other examples include that we tested that the GUI displayed the board and allowed interaction correctly, that additional features such as undoing moves worked correctly, etc. After implementing features we fixed any bugs, issues, or unexpected functionality found in their testing.

Examples

I will now include screenshots displaying the implementations of various features of our program (also available in the screenshots directory).


Gomoku

left click : set current players piece
right click – undo move
p– pause / unpause timer
up/down arrow – increase/decrease size of the board
right/left – increase/decrease target for game
s – save current game at current point to saveME.txt
./gomoku filename.txt –> load previous saved game

Press C to play PVP
Press A to play vs AI
Press T to three and three (PVP)
Press F to four and four (PVP)

Figure 1: The GUI Menu

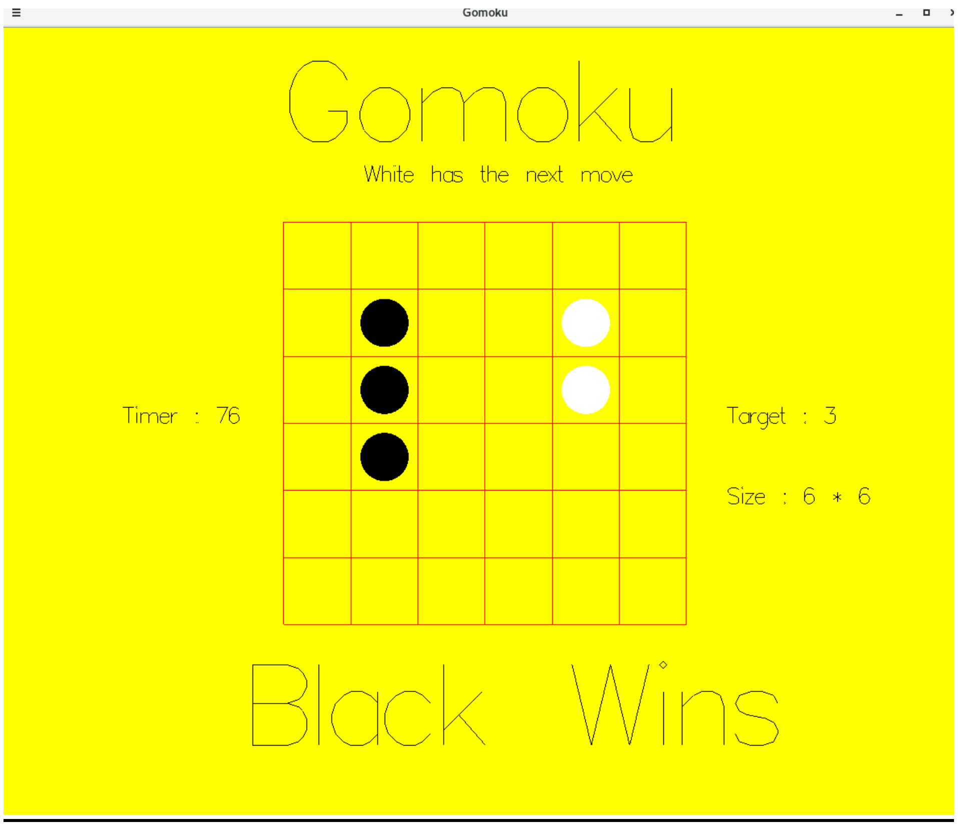


Figure 2: A win state from a vertical combination

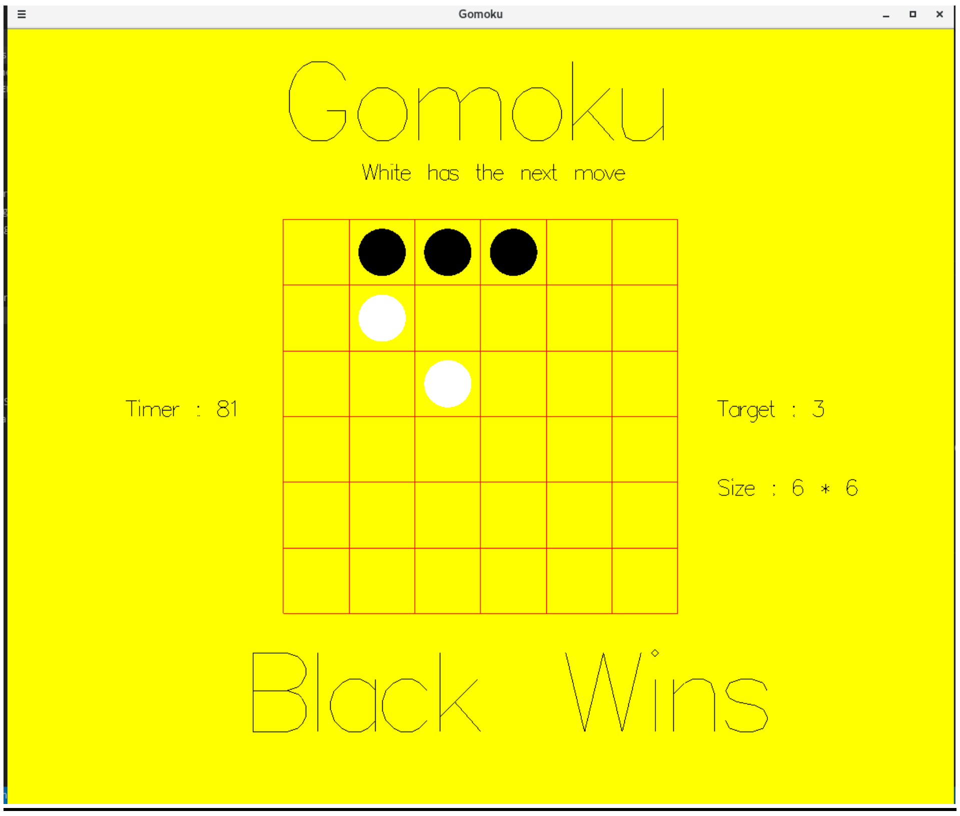


Figure 3: A win state from a horizontal combination

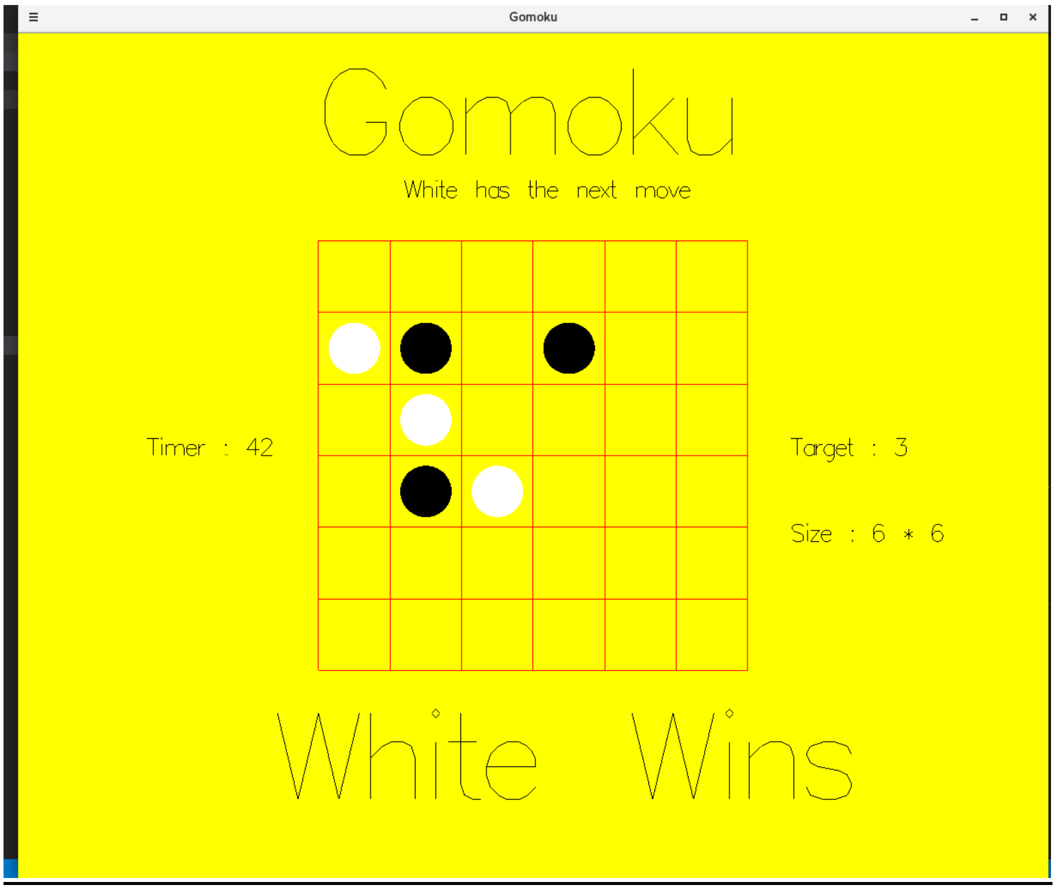


Figure 4: A win state from a diagonal combination

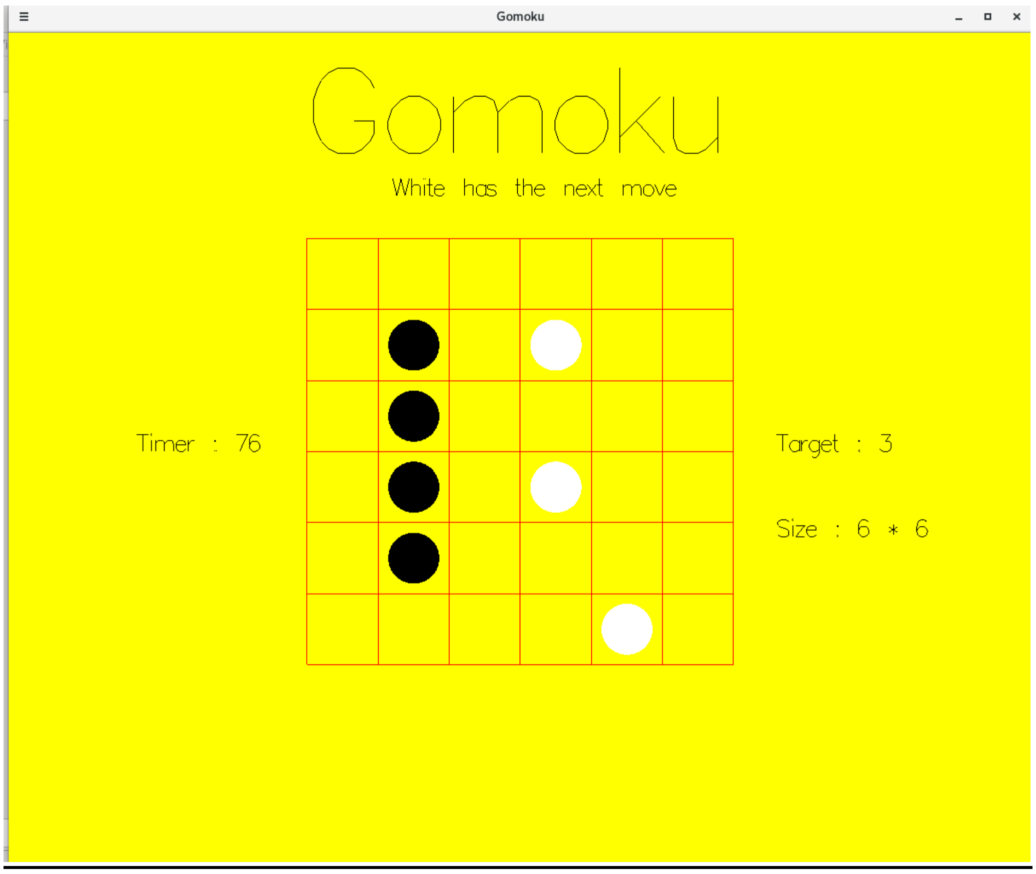


Figure 5: No win state when a combination with a length larger than the target is formed

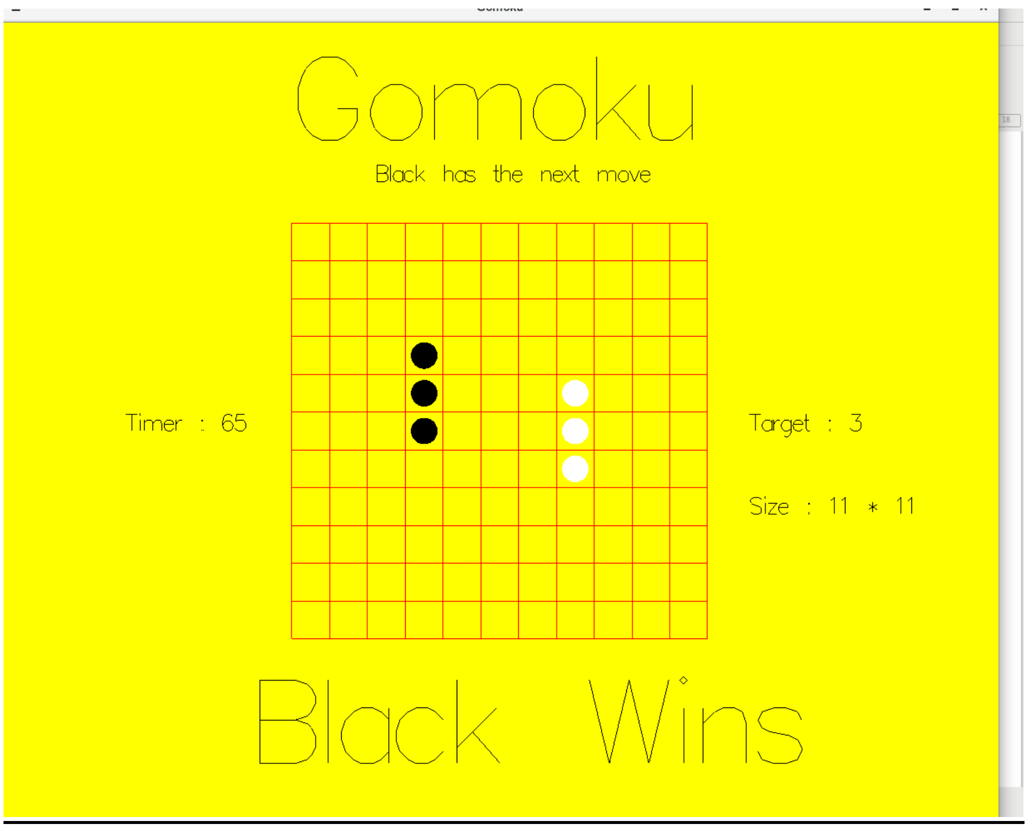


Figure 6: A board of altered size

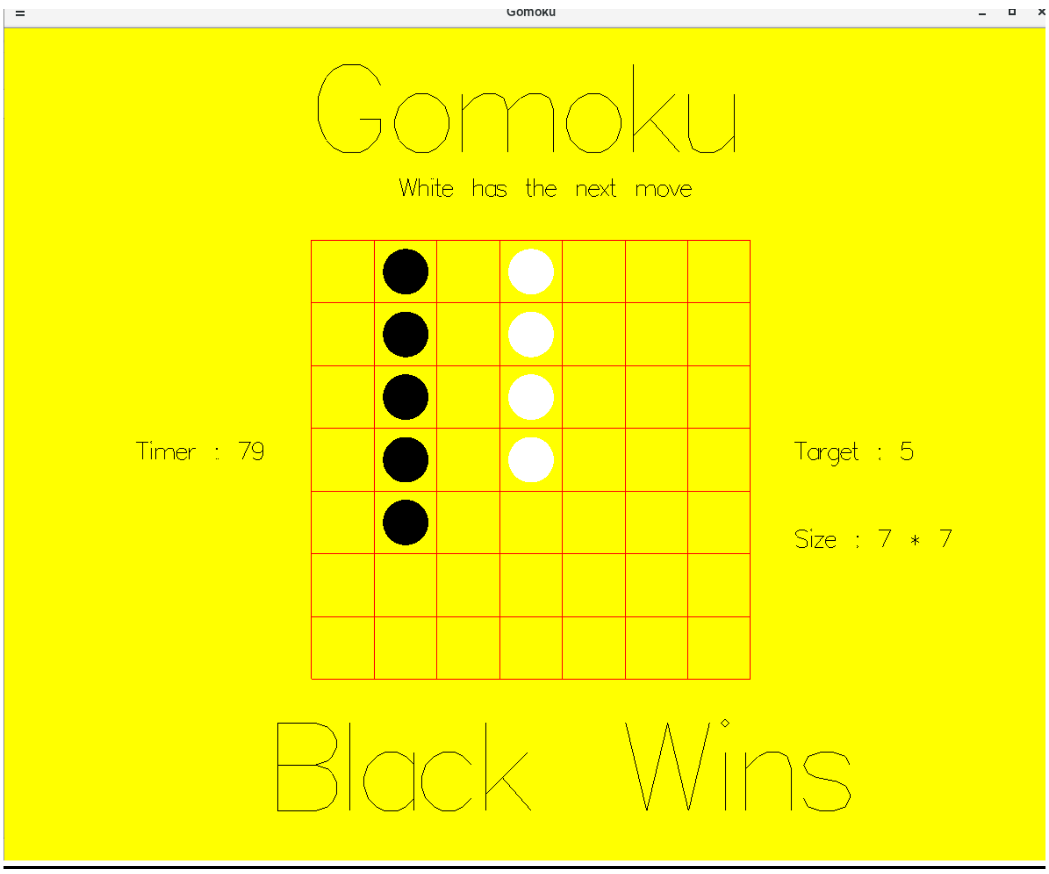


Figure 7: A (win state working with) an altered target size

Evaluation

As previously discussed, we have implemented the basic specification for this practical as well as multiple advanced features. For each feature, we tested it to ensure it works correctly and as expected and required. Our additional implemented features include undoing moves, a pausable move timer, choosing the board and target size, and saving and loading games.

Due to the number and quality of our extensions, and the tests we performed on them, we believe it is safe and fair to conclude that our program sufficiently and successfully meets the requirements of the practical.

Conclusion

We have shown that we have implemented the basic requirements, as well as several extra criteria, such as undoing moves and saving and loading games, and explained how they were implemented. We discussed their functionality in more depth in the Evaluation section.

We initially found understanding how to use cabal and gloss difficult, though once the basics were set up and understood the pace of progress increased rapidly. In addition to this, in a broader sense, our group is weaker with Haskell as a whole as compared to Python.

Given more time, we would like to implement network play for players against both other players and AIs, as well as AIs to play each other. We would also be interested in implementing a machine learning element with the AI using libraries such as TensorFlow or grenade, or even abstracting the ML/AI element out of Haskell and using a web-based AI such as Amazon Machine Learning, IBM Watson, Microsoft Azure, etc.