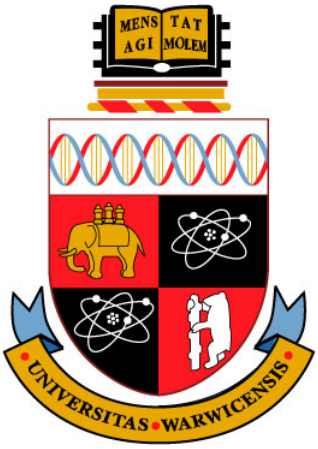

Gridlock

CS141 Coursework 2 Report 2024/25

Oluwaferanmi Akodu

CS141 Coursework 2 Gridlock

Department of Computer Science

University of Warwick

Word Count: 972

Contents

1 Aims

The overall aim of the project was to "Write a Haskell program that allows the user to explore individiual games of Gridlock by parsing written records of Gridlock games." (as specified in the course work specification). This goal can be broken into two main stages:

- Parsing the Gridlock File
- Displaying the Gridlock Game to the User

As an extension to the initial specification, players should also be able to:

- Play Gridlock Games
- Save Gridlock Games to a file once the Game has been played

Adding these features would result in a final application that feels more "complete"

2 Parsing Gridlock Games

When parsing Gridlock game files there are 2 possible ways the data can be invalid:

- **Syntactically invalid** : The structure / format for the data of the file is incorrect. For Example
  - "GRIDLOCK" is missing from the start of the file
  - A full-stop is missing from the end of a line
- **Semantically invalid** : The structure and format of the data is correct but does not make logical sense. For Example:
  - A player tries to play a colour which doesnt exist
  - A player tries to play a colour in a square which has already been filled

Instead of trying to verify that the file is semantically and syntactically correct in one step, it was more manageable to split the verification of these two factors into separate stages.

2.1 Syntactic Validation

Using the Megaparsec library to parse the Gridlock file automatically ensures that the file is syntaxically correct, as the library fails when the contents of the file do not align with the expected format.

2.2 Semantic Validation

2.2.1 Abstract Syntax Trees

- Many compilers use Abstract Syntax Trees during semantic analysis, where the compiler checks for correct usage of the elements of the program and the language.
- A complete traversal of the tree allows verification of the correctness of the program.
- Using a similar approach, my parser validates that the Gridlock file is correct by traversing the Abstract Syntax Tree of the Gridlock game file.

2.2.2 Gridlock AST design

- As the Gridlock file does not have any nested elements and is essentially just a sequence of statements, the AST for the Gridlock file will resemble a Linked List rather than a more typical tree
- The AST was implemented as a recursive data type.

```
data GridlockASTNode
    = Gridlock GridlockASTNode
    | Players (Player, Player) GridlockASTNode
    | GridSize (Int, Int) GridlockASTNode
    | Colours (Set Colour) GridlockASTNode
    | Move GridlockMove GridlockASTNode
    | Lose Player GridlockASTNode
    | Win Player GridlockASTNode
    | End
    deriving (Eq, Ord, Read, Show)
```

- We build the AST as we parse the Gridlock File using the Megaparsec library

2.2.3 Validation via the AST

- An error state was added to the GameRecord data type. This allows for errors to be reported while the Game Record is being generated.
- (Note: In hindsight "Either String GameRecord" could have been used instead of creating a new data type

```haskell
data GameRecordExt = GameRecordExt
    {
        isValid :: Bool,
        errorMsg :: String,
        lastPlayer :: Player,
        gameRecord :: GameRecord
    }
    deriving (Eq, Ord, Read, Show)
```

- A function called "makeGameRecord" takes in a "GameRecordExt" and a "GridlockASTNode" and returns an updated "GameRecordExt".
- This function is called recursively, with recursion ending when either:
  - The "GridlockASTNode" uses the constructor "End"
  - The input "GameRecordExt" has isValid set to false
- When "makeGameRecord" reaches a "GridlockASTNode" that uses the constructor "Move", a function called "playMove" is called which attempts to play the move.
- If the move is invalid then we set the isValid field in the "GameRecordEXT" to false
- If the "makeGameRecord" runs on the AST without isValid being set to False then the Gridlock File was valid and we can return the "Game Record"

3 Displaying Gridlock Games

- Inspired by Halatro, it was decided that a Terminal UI application would be used to display the game to the user.
- The "Brick" library was used for the Terminal UI as it meant that Halatro could be used as a reference / guide
- I struggled to get the Brick File Browser widget to work so i implemented my own file browser UI
- To make the code easier to manage the different menus were written in different haskell files with functions that are common between menus being written in a "Utils" File
- I wanted to also have the colours used in the output of "drawGrids" in the game log.
  - However, I could not this as "drawGrids" uses ANSI escape sequences to achieve this.
  - ANSI escape sequences interfere with the layout of brick widgets so brick does not support them
  - Further research led me to find this workaround (https://stackoverflow.com/questions/63662243/rgb-terminal-colors-with-haskell-and-brick).
  - I chose not to use this solution as it seemed like a hack rather than an actual solution

4 Playing Gridlock

Bricks event handling system was used to call the "playMove" function when an empty cell was clicked on

5 Saving Files

- One option for saving files is to create a AST after each move is played and to write the AST to a file when the user wants to save
- Another Approach would be to iterate through the list of Grids and look for cells which have changed. From this we can then deduce which move was played.
- I chose the second approach, as it would prevent additional data having to be stored and updated when playing the game.

6 Error Handling

- Some user actions, such as loading files, may give errors
- It would not be good if the program crashed after every error
- To overcome this I decided to add error screens for cenarios where errors might occur

7 Testing

- The parser was tested using the unit tests provided in addition to some extra tests
- Some of the tests passed as a result of other decisions made when writing the parser rather than me explicitly writing code to ensure that test case passed

8 Instructions

To Run the program run the command "stack run" while in the project directory.

9 Resources

- Brick UI Library Documentation (https://hackage.haskell.org/package/brick-2.8.3/docs/)
- MegaParsec Library Documantation (https://hackage.haskell.org/package/megaparsec-9.7.0/docs/)
- CS141 Lecture notes
- Terminal UI Code from Halatro (CS141 Coursework 1)
- Additional tests were provided by Niel Martey-Botchway (coursemate)

A Appendix A: Videos of my Gridlock Implementation

- **Link to Loading Files Demonstration**
- **Link to Error Handling Files Demonstration**
- **Link to Gameplay Demonstration**