CS141 Hurtle Report

0.1 What is Hurtle

Hurtle is a basic parser and interpeter for ”Hogo” code which is a variant on Logo code.

0.2 How to use Hurtle

Hurtle is operated via the command-line with a visual window when a program is executed. It will run Hogo programs which typically have the extension ”.Hogo” however any file type will work.

- Run the program using ”stack run”
- You should be prompted with ”Enter path to a HogoCode file (or q/quit):”. At this point, you can either enter a valid file path or quit by typing ”q” or ”quit”
- If your file path was invalid then an IO exception will occur and you will have to run ”stack run” again
- If your file path was valid but the program could not be parsed then there should be an appropriate parsing error in the terminal. At this point, you can enter another Hogo program to execute.
- If your program is valid, then it will execute and a gloss window will appear and your program will be animated.

You could try the following file paths: examples/passing/03-worm.hogo, examples/passing/08-penup-pendown.hogo, examples/passing/hask.hogo, examples/failing/00-keysmash.hogo, examples/failing/02-missing-number.hogo

0.3 Overarching Design

To abide by separation of concerns, the program is split into modules of code with each module concerned with a major part of the process.

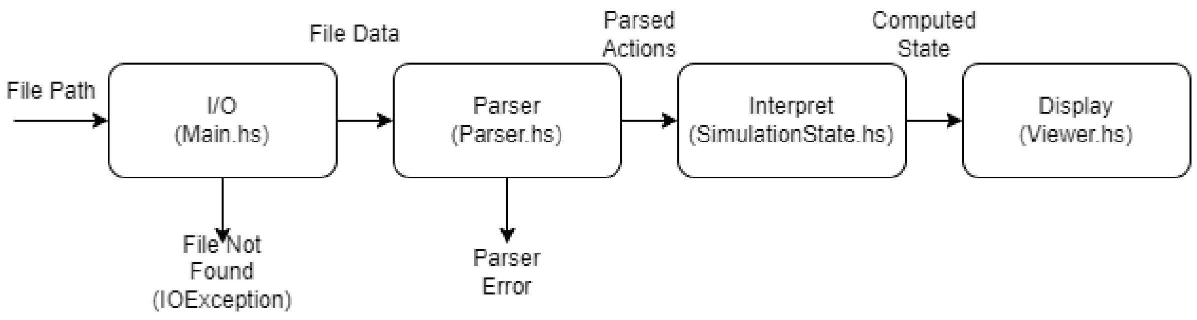

Figure 1: Program Flow

Raw text is obtained from a .hogo file and is then converted into a list of HogoCode actions via the parser. These actions are then converted into a Simulation object through the interpreter. The data inside the Simulation object is displayed by the viewer.

0.4 Libraries Used

Control.Monad is used for combinators (unless, when).
System.IO is used for dealing with IO; in particular, flushing the output buffer (to display text).
Control.Monad.State is used for stateful computation (SimulationState.hs) and is the backbone behind the interpreting of Hogo code.
Gloss is used for viewing the animation (Viewer.hs) and is how the user gets feedback for their Hogo programs.
Megaparsec and Megaparsec's Lexer is used for parsing Hogo programs; they break the raw text into actions which are then interpreted by the state handler.

0.5 Note on Test Cases

Additional test cases were added. Two test cases were edited (02-missing-number.hogo and 03-good-then-bad.hogo) because they used Haskell comments at the start so they failed due to invalid characters rather than what they were meant to be testing. Hence their Haskell comments were replaced with Hogo comments.

0.6 Personal Experience

I usually really enjoy CS141, however the timing of the deadline made it a very stressful experience. I prioritised the CS126 coursework more because it was worth more of the grade, and so I started this coursework after the deadline of CS126 (and a number of other assignments due near the same time).

Unfortunately, this meant that I could not put as much effort as I'd have liked into this coursework and I am quite dissatisfied with the final product. I would have liked to implement more interesting language features such as subroutines, and finished features such as the penwidth action. There was also a lot of room for improving the visualisation.

On the positive side, I've learned that I really need to work on my time management. In the future, I will try to manage my time better for coursework deadlines. I've realised that it's the difference between a project being really enjoyable (as it should be!), and mentally draining (due to time pressure and highly caffeinated all-nighters).

I'm glad that I learned this now, instead of later in the degree (or career).

0.7 Resources Used

Documentation for Megaparsec Lexer:
`https://hackage.haskell.org/package/megaparsec-9.6.1/docs/Text-Megaparsec-Char-Lexer.html`

A guide through an imperative language parser (used as a guide for the Lexer on top of the Megaparsec Lexer documentation): `https://wiki.haskell.org/Parsing_a_simple_imperative_language`