CS141: Snake Game In Haskell

Name: Luke Guppy
Student Number: 2117364

The Program

Overview

The program I made is a simple snake game with basic graphics and a constant speed of 10 cells per second. The size of the game is variable simply by changing the number in either of the rows, columns or cell size functions.

Conventional snake games usually either do or do not have borders: with borders, you die if you hit them; without borders, the snake passes through to the opposite side of the screen. I wanted to incorporate both of these options so on the home screen you can change between having borders or not.

In terms of aesthetics, the graphics design is simple using only square cell blocks for the edges, target and the snake. This gives the game a nice and simple look however I could have imported images to give a higher graphical quality.

Controls

- Movement - Arrow keys are used to control the movement of the snake where the up arrow means moving towards the top of the screen etc.
- Starting Game - The space bar starts the game when on the game over screen
- Activating Edges - Pressing ‘e’ enables/disables the edges changing the game style as previously described

Architectural choices

I decided to split the project into three Haskell files to separate the different processes:

- The main file imports the other two files and runs the gloss play function on the relevant functions from the two of them.
- The game file handles all the logic of the game including the coordinate positions of the snake and target, the game state, key inputs...
- The render file imports the game file and translates each game state into a list of pictures for the gloss function to use.

I decided to use functions for the rows, columns, cell size and different colours so they can act as constants rather than hand coding specific values every time they are used. This makes it a lot easier to adjust the scale of the game without having to scour through every function changing the necessary values.

I created a new type called cell which is a pair of integer coordinates representing where on the ‘grid’ a given cell block is.
The snake is represented by a list of cells in order where the head is the first cell and new cells are ‘added’ to the back.

The target is a single cell and is randomly generated using a stdGen from the random library which changes every game state and also between generating the x and y coordinate of the new target. It is used to generate random positions for the new target when the snake eats it.

Each game state is comprised of the snake, the current direction, a boolean for if the game is over, the target, the current random stdGen, the current score and whether the game is using edges or not.

Cells are filled in using a 'setCellColour' function which takes a colour and a cell and sets that cell on the 'grid' to the given colour returning a picture. The snake cells and edges are created by mapping this function to each of the relevant cells and the target is simply the application of this function to the target cell.

The score and writing for the game over messages are simply translation of gloss texts to the relative positions scaled accordingly by the number of columns and the cell size.

Libraries Used

Gloss

This program is entirely based on the gloss library which I used for its play function. This takes a generic world type, a function to convert it to the next world, a function to convert the world to a picture (a gloss type), an initial world state, a function for converting the world state based on input events, a refresh rate and a background colour. With all of these functions a functioning game can run.

The gloss library also allowed me to use its built in pictures to create the full game picture. In this snake game, this only consisted of using the rectangle shapes and text and the whole graphical display is based on these two pictures scaled and translated differently.

Random

I also used the random library to generate 'StdGen's. This is a way of producing random integers (paired with a new 'StdGen' value). Each game state had a stdGen value so that a new stdGen value is produced based on the value of the previous game state stdGen. I used this to generate new random positions for the target each time the snake hits it.

My Experience

As a first experience with gloss to create a program, I found it really easy to use. I found that the use of the library and the play function hid all the complexities: I only needed to provide relatively simple functions to transition between game states without having to worry about actually applying this transition at a given time interval. It was also easy to use simple shape and text pictures without needing to know how they work only colouring, scaling and translating them as needed

The stdGen type and related functions (in the random library) were initially a little confusing but after understanding it, it became intuitive how to use and was a relatively simple solution to generating random numbers.

Overall, I found this whole project a lot simpler than I had imagined it would be and it highlighted the strengths of functional programming. I am satisfied with the resulting program as it meets all the criteria I set out to achieve.

Resources Used

https://hackage.haskell.org/package/gloss
https://hackage.haskell.org/package/random-1.2.1/docs/System-Random.html