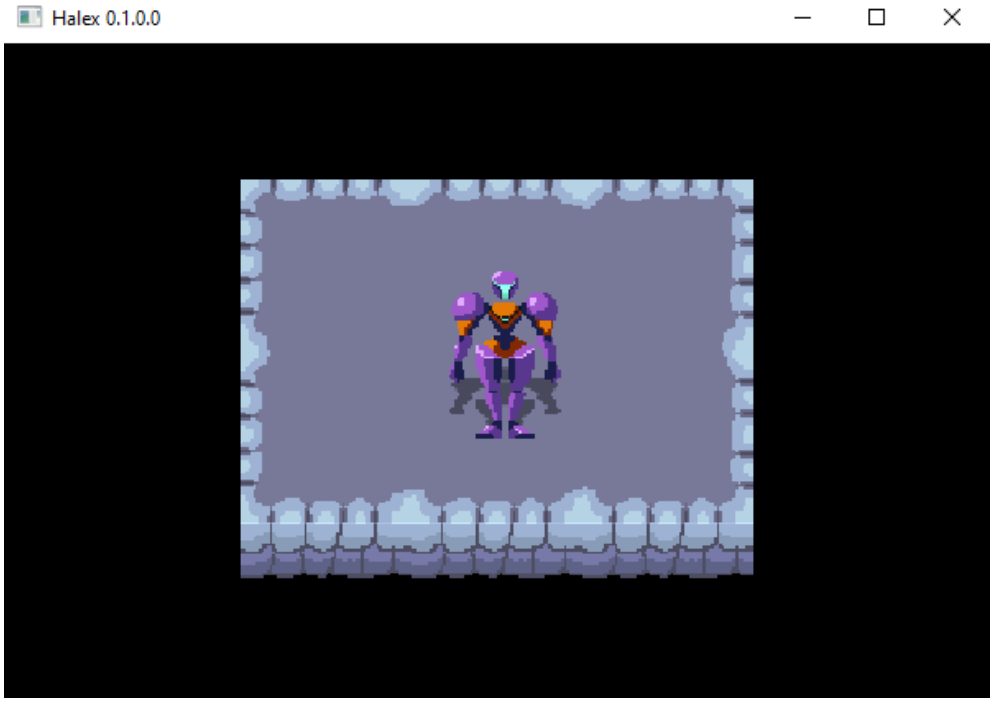
Jeffery Meverden
COMPSCI 657 – 203
Functional Programming
Final Project Motivation
4/12/2022

Final Project Motivation

1. Example Description



Our heroine stands resolute atop a set of ground sprites.

For my final project, I have proposed to develop a video game in Haskell. I have elected to craft a 2D top-down Metroidvania Soulslike that emulates classical arcade and home console games. For my example, I have taken a screenshot of my current progress. Above, we see the game window with several sprites rendered together to create a scene of the player character standing on a stone platform.

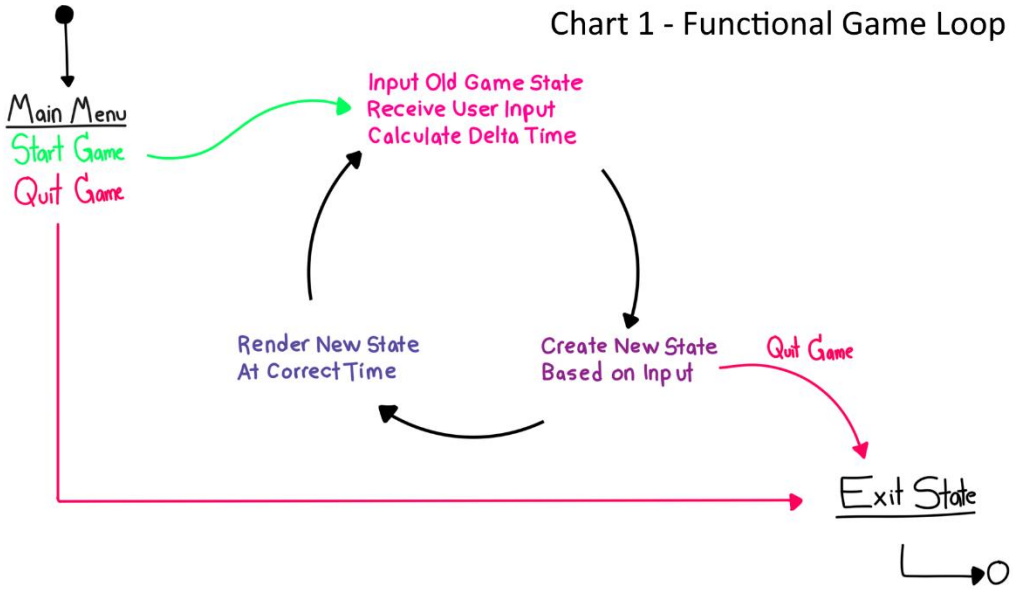
This example is more video than video game because it lacks any form of user agency. However, it helps demonstrate a powerful side-effect: the rendering of graphics. We can create sprites and draw them in a way that represents a game world and a player character to inhabit that world. My approach to solving the issue of building a functional game program is to create a function that accepts this entire world as the game state, consider user input, and then output a new world that reflects the changed state. This recursive creation of an updated world begets the illusion of life: the protagonist marching about in response to the player's orders.

My initial design employed the Simple DirectMedia Layer library (SDL2). However, after further research and numerous failed attempts to install the library on my machine, I have opted instead to use the Gloss library with JuicyPixels to process images and user input. Gloss offers a simpler API than SDL2, though it lacks support for playing sound. JuicyPixels, and its extension Gloss-Juicy, will allow me to load PNG files for sprites that I create.

In my investigation of the Gloss library, I have discovered more Haskell games that I was previously unaware of. There is a pong clone, a platformer game, and even a fan-made recreation of Sonic 2, all developed in Haskell with Gloss. These titles will offer abundant examples for me to follow when developing my own game.

However, I maintain that my approach is still novel because I have not yet discovered any top-down, sprite-based Metroid clones such as the project I plan to release. I also hope that my example will be useful to other game developers interested in pursuing functional game development. I will release the finished project on GitHub and Hackage so that other Haskell programmers can easily download the source code for their own analysis.

2. Architectural Diagram



This diagram is based on the schematic of a functional game by Ashley Smith (see References).

In Chart 1 above, we can see this overarching game loop function illustrated. The game begins with the main menu, and can be launched with a Start Game option. Unlike an imperative game loop that simply updates variables whenever they are altered by certain actions in the game (such as the player character's x and y-coordinates when the player moves), the functional game loop creates a new world every frame that exhibits all of the changes to the old world conducted in the previous frame.

In Chart 2 to the right, we can see the composition of the game state. This data structure will be divided into sub-structures for entities in the game, the game world, and game logic.

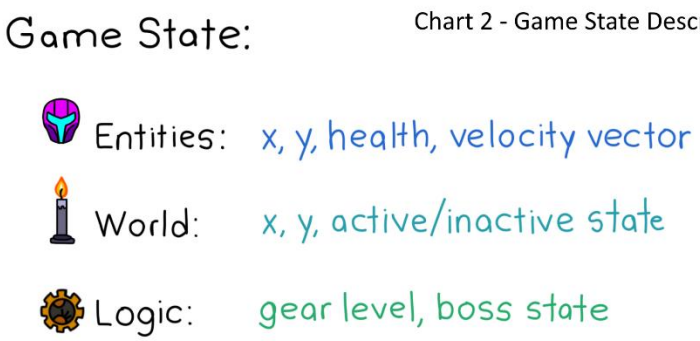


Chart 2 - Game State Description

Entities include the player character and enemies. These objects can move about the game world. The world category references ground objects and environmental scenery. I have included an active/inactive field to reference doors that can be open or closed. Logic envelops abstract states like the player's current level or the status of the final boss. Game logic is also necessary to determine if the player has achieved the win or loss condition.

3. Existing Approaches

The majority of the Haskell games that I have studied thus far have rendered images in a manner similar to the source code presented below. As part of the IO action of the main function, an image is loaded and takes on the type Maybe Picture. This data type may be a picture, or it may be nothing in the event that the file was not found. In the case that the program receives Just Picture, it can be displayed in the window. Multiple pictures can be drawn in the window and given translations to change their position onscreen. The dominant approach to creating game characters that can move around the game world is to translate the character sprite based on user input (or simple AI if the character is an enemy).

```haskell
module Main(main) where

import Graphics.Gloss
import Graphics.Gloss.Juicy

window :: Display
window = InWindow "Halex 0.1.0.0" (600, 400) (60, 40)

background :: Color
background = black

main :: IO ()
main = do

    demo <- loadJuicyPNG "Assets//Images//Demo//Demo.png"
    case demo of
        Just spr -> display window background (pictures [spr])
        Nothing -> print("Loading failed")
```

This code was used to render the example demo depicted on page 1.

The major limitation to this approach is that the game character will not have an animated sprite. Most of the Haskell game characters that I have seen are depicted as static images. In Claes-Magnus Berg's *Gloss Game*, the player character is a small sprite with no animation. Andrew Gibiansky's brilliant *Pong* tutorial loads no images and draws only primitive shapes. Morgen Thum's *Lambda Heights* humorously allows players to pilot the Haskell logo itself. The only games that I have found featuring true sprite animation are Incoherent Software's *Defect Process* and Brian McKenna's *Sonic 2* recreation, both of which are restricted to 2-directional platformer sprites.

4. Functional Solution

To achieve an animated player character, I will draw a series of sprites that can be strung together to form idle, walking, and shooting animations for the character. These animations will be drawn in the four cardinal directions so the player can walk north, south, east, and west. Each sprite must be loaded into the program using the Maybe Picture type to handle any potential file error. The program will feature data structures similar to arrays, one array for each type of animation, with each element of an array containing one sprite (or one frame of that animation).

In the game loop, we will draw the character using the current frame of the animation the character is currently performing. For instance, if the player presses a key to move right, the drawing function will pull frames from the Walk Right array. Each time the game loop function recurs, the following frame in the Move Right array will be rendered, causing an animation to occur. If the last frame in the Move Right array is drawn, then the next game update will reset the loop to the start of the array. This animation will continue looping until the player input changes.

5. Solution Comparison

It seems that the market of games written in Haskell is evolving as a microcosm of the video game industry at large. Pong and Pacman have been created. I have seen a roguelike implementation. Platformers have been introduced. I hope to emulate a classic game style so that I can contribute to this evolution. One of the major milestones in the road these Haskell developers are retreading is the introduction of the animated character.

Although there is more to making a game than just animating characters, I believe that my solution for animation will be a boon to the immersive nature of the game and allow for a greater range of player expression and agency than what is achievable in games that do not feature animation. A bounty of information about the state of the game world can be afforded to the player who can appreciate such animations as enemies reacting to being struck, water flowing to indicate a dangerous current, or the principal character striding about in response to the user's commands.

References

1. LibSDL.org. SDL2 Library
https://www.libsdl.org/

2. Haskell.org. Gloss Library
https://hackage.haskell.org/package/gloss

3. Haskell.org. JuicyPixels Library
https://hackage.haskell.org/package/JuicyPixels

4. Haskell.org. Gloss-Juicy Library
https://hackage.haskell.org/package/gloss-juicy-0.1

5. Ashley Smith. “Of Boxes and Threads: Game Development in Haskell” (Referenced in Chart 1)
https://aas.sh/blog/of-boxes-and-threads/

6. Claes-Magnus Berg. “Making a Small Game with Gloss”
https://blog.jayway.com/2020/11/01/making-a-small-game-with-gloss/

7. Andrew Gibiansky. “Making Your First Haskell Application (with Gloss)”
https://andrew.gibiansky.com/blog/haskell/haskell-gloss/

8. Morgen Thum. *Lambda Heights*
https://github.com/morgenthum/lambda-heights/blob/master/screenshot.png

9. Incoherent Software, LLC. *Defect Process*
https://store.steampowered.com/app/1136730/Defect_Process/

10. Brian McKenna. “Sonic 2 in Haskell: Sprite Animation!”
https://www.youtube.com/watch?v=ZpgnAWsuhV4

11. John Slap. “Make You A Roguelike in Haskell for Greater Good”
https://www.youtube.com/watch?v=5URsWAlkT_0