CS141 Coursework

Archie Harrodine
2127935

One Night at Freddy's

1 Project Description

When faced with the task at hand, there was only one suitable option for me to pick: Five Nights at Freddy's. Not only had this game been a favourite of mine for years, but the more I thought about it, the more it dawned on me that it was also built in a way that lent itself to Haskell. Five Nights at Freddy's (or FNAF) is a fairly simple game, in which you try to survive the night shift at a haunted pizzeria, where the animatronics: Freddy, Foxy, Bonnie and Chica, are after your life. The way you protect yourself is by monitoring the animatronics on the camera monitor you are given, and by shutting doors to stop them from getting to you. On top of all that you have a limited supply of power, and so coupled all together it becomes a hectic experience of micromanaging each animatronic.

Hence the reason this project lends itself so well to Haskell, is because its a game built on time based functions. The first introduction to Haskell was a labs in gloss, using time to determine the the way images changed on a screen. This was the logical, and meaningful extension of how far I had come since that first lab.

2 Architectural Choices

2.1 Running the Program

As you will have seen from looking at the code, there is a large facet of different areas and files that all contribute in different ways towards game. These can be broken down into the main sections of the primary monadic interface used: Gloss; specifically their play function.

Play is broken down into several arguments:

→ Color : Background color
→ Int : Frames per second (fps)
→ world : Initial GameState
→ (world → picture) : Render function
→ Event → world → world : Input handler
→ (Float → world → world) : GameState Update

The ones that really matter to us are the render function, the input handler and the GameState update. Starting with the render function, this takes all of our assets, and uses the current GameState to convert it into an image that reflects what we want to see on screen. This includes displaying the office, the content on the cameras, the jumpscares, and any other images present throughout FNAF. The assets used were taken straight from the original game, with the link to the assets and the game being in the resources.

Next we have the input handler, and this is how we tackle inputs such as clicking on different cameras, or opening and closing the doors using buttons. This made use of the Event handler built into gloss, allowing us to find where the mouse was on the screen, and if it was clicking. Additionally the cameras were implemented such that you bring up the monitor by pressing the space key.

Finally there is the update function, which is the most important as it explores how we change our game every single frame. This made heavy use of the State monad, in which it was used not only to parse changes on to the GameState in a way reminiscent to an imperative language; but it also allowed us to return useful data and update

the GameState simultaneously. The most prominent example of this how the State monad enables randomness, by updating the random seed each call of random, whilst returning the random value.

Currently to the point it is at, the features of: a fully functioning office, cameras, time and battery system, with Bonnie and Chica fully implemented, Freddy mostly implemented, and a plenty of ways to win and to lose the game!

2.2 Design

The way that the game was designed hence, was that the animatronics would operate as close as they would in the original game, their AI being found and explained in the video 'Ruining FNaF by Dissecting the Animatronics' AI — Tech Rules'. A graph of data would hold the directions to which an animatronic could traverse too, and every set number of frames, the animatronic would make an attempt to move. This move is dependent on a random generator, compared the aggression level of the animatronic. Bonnie and Chica would roam freely, and if coming close, would try to get into your office, whilst Freddy would always be heading in your direction, and the only way to stop him is to look at him on the cameras. Unfortunately, due to how different Foxy was, I was unable to implement them, similar to Golden Freddy; however the basis and their set up for their existence is there.

The other mechanics then to implement were the in game time progressing, as well as the power. The way FNAF power consumption works is that every 9.6 seconds you are guaranteed to lose 1% of your power. However, if you have one door close, then its every 4.8 seconds; another door, halves the time again to 2.4 seconds, and so on. If the usage is four, you will roughly lose power once every second, and when you run out, you enter blackout mode, where everything will go dark, you cannot pull up the cameras, all you can do is wait before Freddy jumpscares you.

3 Libraries

Dependencies: gloss, gloss-juicy, random, mtl
Default-Extensions: RecordWildCards, MultiWayIf

4 Personal Experience

My experience with this coursework has been a mix of incredible highs and bouts of extreme frustration. The opportunity to make something creative and fun without restraint is everything I enjoy, and making a game with Haskell's rudimentary yet deep systems was very entertaining. Once I had made the world - the office, the cameras, etcetera - much of its creation came quite naturally, especially designing graphs and systems around the animatronics. Moreover, being able to explore wider uses of Haskell and monads as a whole was incredibly useful.

However, there were areas which were - and still are - a pain. Certain points in my code seem to require the buffer to be refreshed to maintain randomness, and furthermore due to the severe lack of documentation or examples out there online. This made it difficult to code in some aspects, especially in regards to make monads and do notation work well and in order, with there being small errors with thr frameTrackers currently. Additionally, there was a many more features I wish I could have added: a static effect if an animatronic disappears on camera, a start screen/menu for ultimate custom Night, as well as a retry feature, multiple nights, Foxy, and most importantly Audio. Audio is the backbone of a horror game, and without i, it does dampen the experience quite a large amount; plus the lack of Foxy makes the game incredibly poorly balanced.

If I could redo any bits in particular, it would be in designing the locations, implementing them as an array of Animatronics in each room, then creating an instance of Eq that checks the name of the location, rather than all of the values. Another area of improvement would be the organisation of the assets; not all of them being downloaded at once meant they weren't properly organised into groups of images, which hurt some efforts of iterating value.

On the more positive side though, when the game is working, it is working! I have had a lot of fun ensuring Freddy doesn't get any where near my office, as I frantically search the cameras to find if Bonnie is sneaking up on me. The original plan of making my own assets would have been fun, but with the time that I had, the product is something I am happy with.

5 Resource List

Cawthon, S. (2014). Five Nights at Freddy's.

claes-magnus (2020). Making a small game with Gloss. [online] blog. Available at: https://blog.jayway.com/2020/11/01/making-a-small-game-with-gloss/ [Accessed 7 Mar. 2022].

claes-magnus (2022). Glossy Haskell Game. [online] GitHub. Available at: https://github.com/bergsans/glossy-haskell-game/blob/main/app/Main.hsL155 [Accessed 7 Mar. 2022].

Dropbox. (n.d.). Textures. [online] Available at: https://www.dropbox.com/sh/vxiigvz3e465xpt/AABIiEbOjZhjQ1U4oAe6NHb1[$Accessed 8 Mar.$2022].

Five Nights at Freddy's Wiki. (n.d.). Power Indicator. [online] Available at: https://freddy-fazbears-pizza.fandom.com/wiki/Pow$text = 1\%25\%20each\%204.8\%20seconds\%20with$[$Accessed 24 Mar.$2022].

hackage.haskell.org. (n.d.). Graphics.Gloss.Game. [online] Available at: https://hackage.haskell.org/package/gloss-game-0.3.3.0/docs/Graphics-Gloss-Game.html [Accessed 24 Mar. 2022].

Monday Morning Haskell (2019). Making a Glossy Game! (Part 1). [online] Monday Morning Haskell. Available at: https://mmhaskell.com/blog/2019/3/25/making-a-glossy-game-part-1 [Accessed 7 Mar. 2022].