# CS141 - A Functional Program

Cameron Ramsell, u2147599

## 1 What the program is and how to use it

### 1.1 What the program is

For the coursework, I have developed a simple game that a player can interact with through Discord. The premise of the game is simple, a random number between 1 and 1000 is generated, and you gain points by guessing the number. The bot is also capable of showing a leader-board, in the form of a Discord rich-embed

The user must first create a profile with the command ”create” (@BotHandle create), which will create an account in the database for them. Then they can attempt to guess the number with the command ”guess” (ex: @BotHandle guess 13, if required for testing: the generated numbers are outputted to console), the amount of points gained depends on how long the number has lasted - every 10 minutes the reward doubles, up to a maximum of 32x

A user can also view their profile using the ”profile” command, users can provide a second mention to view another’s profile (ex: @BotHandle profile @dave). Alternatively a user can view a leaderboard using the ”leaderboard” command. There are two types of leaderboard: a single global leaderboard and a second guild leaderboard for each guild the bot is in, these can be viewed with @BotHandle leaderboard global, or @BotHandle leaderboard server (when not specified, the command defaults to the global leaderboard)

### 1.2 How to use it

The bot connects to both Discord and a MongoDB server. Hence, the application requires both a Discord bot user (including it’s API key) and a running MongoDB server. The API key of the bot, and the host of the mongo server should be entered in the configuration file - located in src/BotConfig.hs. After this, simply build  execute with ”stack run”

## 2 Architectural choices made

### 2.1 Monad transformer stack

The entire bot is built around a monad transformer stack (defined in src/BotHandler.hs). The transformer stack has a reader at it’s head, which contains 3 TVars: the first holds the current randomly generated number; the second holds the time which this number was generated (stored as seconds since epoch, since this is what I am familiar with); and the third is to a Map that acts as a cache for the leaderboards maintained by the bot. I chose to use a Reader object containing TVars for two reasons: discord-haskell spawns a new thread for each request, and I know TVars to be thread safe; and secondly, TVars seemed a lot simpler than other stores of state, such as StateT. I used a data object so that I had room for expansion - if there were more time, I could have implemented a cache for player profiles, or perhaps a rate-limiting system for command execution. This reader is stacked on top of the DiscordHandler monad transformer stack, which is provided by discord-haskell. This is used for the obvious reason - to interact with the discord API.

### 2.2 Command handler

The entry point for most messages is the command handler, which is located in src/EventHandler.hs (originally I planned to listen for more events, but it turns out deadlines exist). The command handler does 3 things: validates the user should be able to execute commands; finds the command a user is trying to execute; and routes command executions to their respective handlers. My original intent with the handler was to be less restrictive with how commands are formatted - unlike other bots, the bot doesn't require arguments to be in a specific order. Instead the user must include the bot's handle in the message, as well as a specific phrase (such as 'guess' or 'leaderboard').

### 2.3 Leaderboards

My leaderboard implementation is extremely basic: Every time a player gains points, we check every possible leaderboard they can be on to see whether they fit: if they do, we insert them - we only store information about players that are currently in the top 10. This however means that we can't develop any system that removes score from a player (for example, a shop). If we wanted functionality like this, we could instead use a structure like a tree.

## 3 Personal experience

I originally intended the project to be a stat aggregation bot for the game 'League of Legends' - but creating a new set of models for a secondary API would of took a considerable amount of time, which I did not have. So, I pivoted to a simple discord game that could use the already developed systems (leaderboard and profile store)

## 4 Libraries used

### 4.1 discord-haskell

I used discord-haskell to interact with the Discord gateway/rest API. It is very poorly documented, which made it very difficult to use

### 4.2 mongoDB

I used the mongoDB driver available on hackage to connect to the Mongo server. It was also very poorly documented.

### 4.3 time, text, mtl, random, containers

time: used to pull the current time from IO; text: used to manipulate utf-8 text (used by discord); mtl: transformer stack stuff; random: used to generate random numbers; containers: I used the Map structure from this for the leaderboard cache

## 5 Resources used

The only resources used, excluding documentation of dependencies, are example programs provided by both discord-haskell and mongoDB. These were:

- discord-haskell/examples/gateway.hs
- mongodb-haskell/test/QuerySpec.hs

I also viewed the source code of discord-haskell directly, as the documentation provided was not sufficient