1 Motivation

Games are commonly programmed in imperative languages, especially in object oriented style. Object oriented programming is a great paradigm to model the world of objects interacting with each other. However, there may be much overhead in adding any new class of objects to the game, and the code quickly becomes more difficult to maintain.

Functional programming has become more popular in the recent decade, but is still limited in its applications. Functional programming offers a new way to think about software in general. In a purely functional language like Haskell, we de-emphasize code that modifies data, rather focus on functions that take immutable values as input and produce new values as output. This makes it easier to organize, reason and test the code. (RWH09)

This project investigates the merits of functional programming in Haskell when applied to developing a 3D network multiplayer game. There are a few examples of games written in Haskell, notably Frag, a 3D game. To our best knowledge, HamsterBall is only 3D Network game written in Haskell, written by David Constanzo, Alexander Thompson, Matthew Sills and myself. The game suffered from some drawbacks and poor design decisions. For this project, I evaluate the efficiency of the game, rewrite the new version from scratch, and add more features to make the game more playable.

2 Hamster Ball v1.0

Hamster Ball v1.0 which was written in Spring 2009 as part of a final project in a Computer Graphics course at Yale University. Each player controls a hamster armed with a 10-terawatt laser and a 100-terawatt protective hamster ball. Mouse-look and standard A-S-D-W controls are used to aim and move. Left mouse-click shoots and mouse-wheel triggers flying. The game’s objective is to shoot and destroy other players. Every laser hit a player takes reduces that player’s protective hamster ball strength by the wattage of the laser fired, and when a laser hit takes a hamster’s protection down to 0 strength, the hamster dies a gory death.
and is respawned at its original start location. The scoreboard keeps track of each player’s achievements by the number of kills and deaths.

Currently there are various bugs and drawbacks with the game:

- Other clients crash when a client quits out of the game, cutting off connection with the server.
- The game is slow and rather skippy, especially when more than 4 people are join the game.
- It only runs on the zoo machines.

This project will take the game designs and functionalities to a whole new level. It will be called Hamster Ball v2.0.

3 Issues with dependencies in Hamster v1.0

The architecture of Hamster Ball v1.0 centres around Functional Reactive Programming (FRP). (EH97) For this, it uses Yampa, a domain-specific embedded language (DSL) for the programming of hybrid systems along the concepts of FRP. However, Yampa is no longer maintained and is not well optimized to run efficiently. (CNP03)

Hamster Ball v1.0 also uses an outdated OpenGL 2.2.3.0 Haskell binding and does not work with the newest OpenGL 2.4.0.1 library.

Another external library used is GLFW module for Haskell which is used for creating OpenGL contexts and managing input, including keyboard, mouse, joystick and time. GLFW doesn’t work well with GHC threads, forkIO and threadDelay. Unfortunately, we used threads and forkIO in our network code. One major challenge in this upcoming rewrite is to separate graphic rendering and network codes better.

4 Promises in Hamster v2.0

In this project I will actively analyze the v1.0 code and plan for the rewrite of v2.0. The new version will be able to run on any platform. It will gracefully handle players entering and exiting the game. It will be less skippy. There will be AI to play with.

Below is a rough timeline.

2/15: Benchmarking to find out the bottlenecks in the game. Some potential bottlenecks are graphic rendering, client-server frequency of communication, and the amount of client-server messages.
3/1: Overall re-design of the game on paper
3/5 - 3/22: Spring break
3/22: Hamster ball floating around
3/29: Terrain and surroundings
4/5: Hamster shooting lasers and collision detection
4/12: Multiplayer support: a remote player can join and exit – server should handle client exiting gracefully
4/19: AI floating around
4/26: Scoring mechanism
5/3: Benchmarking
5/10: Writeup

5 Deliverables

- A final paper that explains the high level design of the game, benchmarking of the old version and improvements in the new version.
- Source code and executables. The game should run on any computer, not just zoo machines
- Benchmarking results
- A webpage containing all the above

References


[EH97] Conal Elliott and Paul Hudak, in ICFP 97. Functional Reactive Animation
