Asteroids: Data & Architecture Design

November 19, 2022

1 Introduction and Rules

The focus of this paper is to explore game development, particularly the Asteroids game, in Haskell in the context of small-scale 2-D games. The intention will be to use the Gloss library for graphics which provides a nice interface to build small 2-D games. The following sections of this document describe the implementation and design of Asteroids. Asteroids is a space-themed single player arcade game designed by Atari in 1979. [1]The player controls a spaceship which can move around the screen also known as the Asteroids field. The spaceship has to avoid the asteroids that appear on the field, and can destroy them by shooting at them. After some time, the game becomes harder as the number of Asteroids increases or different kinds of enemies might pop up with a bit of more intelligence making the game more challenging. The goal of the game in short is to shoot the asteroids while not being hit. The game is finished when the spaceship collides with the asteroids or is shot by the saucers (depending on the life of the spaceship in the initial state, the game might end with one collusion/shot or more).[2]

2 Design

The process of the game can be briefly described as: "The game has **one player** using **keyboard buttons,** specifically "w", "a", "d", and "s" will be used for the **movement** of the **spaceship** and the arrow keys for **shooting.** When one **rock (Asteroid)** is shot by **bullets,** it is destroyed and an **explosion** occurs. As the game progresses, the Asteroids become more **intelligent** and instead of randomly wondering around they will move towards the spaceship. The ship is trying to survive and gain **points** by shooting down asteroids. By shooting down each Asteroid 100 points will be achieved. The game will **end** when the spaceship and asteroid collide. These are all the elements that constitute our **world.**

There are **three** main data structures in this game: 1- The world state which encapsulates the characteristics of the world 2- The phase of the world which indicates whether the game is paused, playing, over, or at the start. 3- The entities (spaceship, asteroids, bullet, particles).

2.1 Data Structures

2.1.1 The Game State & Phase

The **phase** of the game at the highest level consists of four general states: 1- the game is still running. 2- the game is paused 3- the game is over 4- the game is at the start. Every **state** of the world will need to consider the phase of the game, state of the spaceship, the state of the asteroids, the state of the bullets fired by the spaceship, the explosions, the time elapses and finally the score of the game which is gained by shooting the asteroids.

[1] https://en.wikipedia.org/wiki/Asteroids
[2] http://www.freeasteroids.org/

Note that the Ship will always be controlled by a human and the Asteroids will be controlled by the computer. Furthermore, at each state of the game there can be several Asteroids and several bullets. It is worth mentioning that the game consists of two kinds of collisions: 1) asteroid and bullet 2) asteroid and ship. When asteroids and the bullets collide, there will be an explosion but when the asteroids and spaceship collide the game will finish.

2.1.2 The Player and Playing field

What does the state of the ship, bullets, and Asteroids mean? The Ship data type will be represented by an enumeration type with two constructors and the following fields: 1- The location of the ship at each state is important. 2- The velocity of the ship at each state must be considered. 3- The size of the ship must be defined (this will also consider the shape of the entity which will be used for detecting collision). In this case the size has been defined by a radius.

The same goes for bullets and asteroids as well; but the bullets and asteroids only have one constructor. This is because that in each state we only need a list of these entities and if an asteroid is destroyed for example, then we just remove it from the list. Take note that we obviously do not want the bullets to wonder around forever, thus, from the firing, their velocity will decrease with each **timestep** of the world.

2.1.3 Attributes

The Location, Velocity, Radius of the asteroids, bullets and the spaceship are the attributes of the entities. We have encapsulated the Radius and Location inside a new data type called Collision. The Score data type is for keeping the score of the game.

2.1.4 Constants

With the mentioned details at the beginning of the game it is necessary to define the initial state of the world. Thus we can define

initialWorld :: GameWorld

initiallWorld = GameWorld X

where X will be the traits of the initial world. Also the Velocity of all objects will be constant. Moreover, the first four asteroids will always be the same. The time frame is constant as well.

2.1.5 Player Movement

Asteroids is a game where all the moving particles are independent. Likewise, movement is continuous (non-discrete). In other words, there is smooth movement inside the entities. There are four moving entities: 1) spaceship 2) bullets 3) asteroids 4) particles (the explosion result). There are, therefore, four movement functions each of which will take the object they are correlated with and change its state. These functions are pure and only when operated the impure part needs to be considered. That will be done by a step function which transforms one world state to the other.

Furthermore, there are defined functions for checking collision (the two kinds) for the movement of bullets, asteroids, spaceship, and particles.

2.1.6 Event Handling

The events are collision (two kinds), firing bullets, and explosion. For each of these events there is a function allocated to describe what the event will do. As an example, the collision functions takes the two entities it is correlated with - as input – and outputs the same two entities. However, if collision between the two objects did not happen; in other words collision was False then the output entities will be in the same state as they were in the input. Also, the location of the collision will be passed on as the input to create the explosion in that location. Ultimately firing bullets will be done when the user inputs the arrow keys and the bullets will be created from the location of the spaceship.

Probably a helper function will be needed with a Bool data type as the output. In this way the state of the entity will be updated. In the case of Asteroids, if any collision has occurred the Asteroid will be updated (will be smashed to pieces) otherwise it will remain unchanged. Pause takes a world as input. If the state of the World is in the Play state it will give the World in a Pause state as output. If the World is in a Pause state it will give the World in the Play state. Otherwise, it will do nothing. (that is if the World is in the GameOver state or start state). Restart if the game is in GameOver state restarts the game from scratch.

2.2 World transition & Interface

Ultimately there is one function to take a World data type and a set of events and IO as input and return a World data type as the output where the state of the world will change. This is done by the step function. We will then turn the world into a *Picture*, which is displayed in the screen.

step :: Float -> GameState -> IO GameState

view :: [Picture] -> GameState -> IO Picture

The float in the state indicates the timestep.

2.3 Randomness

Randomness has been applied in the initial Location of the Asteroids. That is in the production of the asteroids there is a randomness in where they will be created.

2.4 Abstraction

I have divided my code in the model, view, and controller modules. However, I have not used any type classes and have only boxed by data using various data types.