Functional Programming Game: Asteroids

Design document

Hugo Krul & Koen Vermeulen

11-10-2023

Table of Contents

Game Type

We are going to implement the Asteroids game. This game has a single player, which is controlled directly by the player. The gameplayer has to dodge asteroids which will be spawned at random points in space and will fly in a random direction from the spawn point to an arbitrary point in space. Once an asteroid collides with the gameplayer the gameplayer will lose a life. Once all lives have been lost, the game is over and the option to restart the game will appear. The gameplayer is able to fire bullets that destroy the asteroids. Using this tactic the gameplayer can defend itself and have a greater chance at dodging the asteroids. Once an asteroid is broken there will be a score added to the current gamescore and if the asteroid is big enough, it will break in 2 smaller asteroids that spawn at the exact point of breaking of the bigger asteroid and will fly in a random direction. For this, the already existing spawn function can be used. Enemies will be introduced later in the game. These enemies will shoot in a general direction of the player attempting to kill the player. Once a bullet hits the player, the player will lose a life.

Twist

We will not be implementing any twist, we will be completely focusing on the minimal requirements and the two optional requirements stated below. This way we can assure a good quality of our code and learn more from the assignment. If there is a lot of time left for before the deadline, we can always look to find some twist we can implement, but we will not think about that for now.

Datatypes

```
data World = Play [Asteroid] [Planet] [Player] Enemy [Bullet]
           | Pause [Asteroid] [Planet] [Player] Enemy [Bullet]
           | GameOver
                  deriving (Eq,Show)


type Velocity     = (Float, Float)
type Acceleration = (Float, Float)
type AsteroidSize =  Float
type BulletAge    =  Float


-- Define Ship, Bullet and Rock --
data Player  = Player  PointInSpace Velocity Acceleration
    deriving (Eq,Show)
data Bullet  = Bullet   PointInSpace Velocity BulletAge
    deriving (Eq,Show)
data Rock    = Rock     PointInSpace AsteroidSize Velocity
    deriving (Eq,Show)
-- Our UFO --
data Enemy   = Enemy  PointInSpace Velocity Acceleration
    deriving (Eq, Show)
```

Minimal Requirements

- Player:
  The player will be movable with the Up, Down, Left and Right arrow and will shoot with the spacebar. Once we implement two players, the second player will be movable with W, A, S, D and shoot with Alt. Because the player glides around like you're in space, we will try to use vectors for position, speed and acceleration. This will be implemented in the update. It will also constantly check if it collides with a rock, bullet (or planet). If so the ship will be destroyed and if there are still lives a new ship will spawn.
- Enemies:
  We will have three different types of enemies.
  - An Astroid which will fly around randomly and split in two if the player shoots it. The random velocity will be set in the spawn function and will never change. The update function will also detect if it collides with a bullet. If so, then the astroid will be destroid and two new smaller astroids will spawn. Until the smallest astroid has been destroid
  - A space ship which will fly around targeting the player and try to shoot it. It does so by seeking the player: it calculates a "force" by making a vector from the ship to the player and that force will be limited by a maximum and then it will be added to the acceleration force of the space ship. It will also shoot bullets to the position of the player.
  - If we implement our twist we will also have a big planet with a slow velocity which will be set on initiation. It cannot be destroyed so it won't have a colliding function. It will have a timer to see if it lived long enough.
- Randomness:
  There are two types of randomness which will be implemented:
  - The randomness of the velocity of the astroids and planets.
  - The randomness of where an enemy will spawn. We will implement this in the main function of the game that checks if all enemies are gone, after it will spawn new enemies with random positions and velocities.
- Animation:
  To implement animations we will need to import an animation library. Information about this is not yet known to us, which means we cannot accurately predict the implementation. The animation library we would likely use is Reanimate.
- Pause:
  When a player chooses to pause the game by pressing a predefined key, the game world should be toggled from a Play state to a Pause state with all the attributes of the current game state and thus GameWorld. In the Pause state no calculations will be made and thus the game is perceived as paused. Once the player decides to toggle the game back to Play, all the calculations continue as Play will get all the arguments used in Pause. In code it will roughly look like this:

```
data World = Play [Asteroid] [Planet] Player Enemy [Bullet]
           | Pause [Asteroid] [Planet] Player Enemy [Bullet]
           | GameOver
                  deriving (Eq,Show)
```

- Interaction with the file system:
  The GameWorld can be saved in as a text file to be read later. This will be built on all the elements and it's attributes in the World. To save this to text we need to implement or inherit a show method on each data type we implement. To load a file we need to reverse this show function to make new instances of the datatypes we load into the gamestate.

Optional Requirements

- Multiplayer
    - We could implement a dual playing option where you can either play against eachother with two ships, or with eachother to destroy more astroids. It will be as simple as add a new player to the world with different controls and a value which will check if it needs to check for collision with other player bullets or not.
- Custom Enemies:
    - There will be planets that can’t be broken and are bigger than asteroids and thus have to be dodged instead of simple shooting. This means we need to add a new datatype only with pointInSpace and Velocity, it doesn’t need a collision detector for the bullets, but only a time value after a max it will destroy itself.
    - Stars that overheat the ship and thus disable the gameplayer to shoot for a moment. This will also be a new datatype. It will destroy all bullets from the playership immediately after shooting. That way we don’t have to add a new value to the player which checks if there is a star or not.

Seperation Pure & Impure

The impure functions we will handle in the main function, which will call the update and render functions. This update function itself is considered to be impure. This update function will call the pure Haskell functions. These functions are considered pure as they are pure functional programming implementations. The Impure functions are considered more to be imperative programming functions.

Opportunities abstraction

We think there are options for abstraction.

- Player and Enemies all have a position in space and a velocity. Some player and enemies also have an acceleration and some also can shoot bullets.
- There will also be options for higher order functions. For example the collision detection can have three inputs and a Bool for outputs:

```
type BottomLeft = (Float, Float)
type BottomRight = (Float, Float)
type Border = (BottomLeft, BottomRight)

collisionDetection :: Border -> Border -> Bool
{- Check if the second border is inside the first border -}
```