Asteroids Game

Universiteit Utrecht

By

Li-Wei Yeh, 0540013
Maher Mussa, 1787284

January 3, 2022

1 Introduction and Rules

This document describes the implementation we have made of the Asteroids Game for the project of the course Functional Programming at Utrecht University. Asteroids is a single-player space-themed game. The player controls a spaceship in a field of flying asteroids. Periodically, there will be asteroids spawning all over the map. The player should try to simultaneously evade the asteroids whilst also trying to destroy them. The player gains points as he destroys the asteroids.

1.1 Controls

The player can move forward by pressing the "w" key. The player can rotate left and right by using the keys "a" and "d". Shooting bullets can be done by both using the Space key and the left mouse button. When a game has ended, you can press "c" to play again. Some hidden keys are the "q" and "f" keys. Pressing the "q" adds 10 points to the player's current score. Pressing the "f" key ends the game early.

2 Design

2.1 Data Structures

2.1.1 The Game State

The full state of our game is represented using a GameState data type. The GameState holds the entire state of the game, including the time, the player, the enemy objects, as well as playstates of the game and the score.

```
data GameState = GameState {
          player             :: SpaceShip
        , asteroids          :: [Asteroid]
        , explodedAsteroids  :: [ExplodedAsteroid]
        , score              :: Int
        , elapsedTime        :: Float
        , spawnTimerAst      :: Float
        , spawnTimerUfo      :: Float
        , shootTimerUfo      :: Float
        , playstate          :: PlayState
        , highscore          :: Int
}
```

By using a PlayState datatype, we can keep track of the state of the game.

```
data PlayState =
  IsPlaying
| IsPaused
| IsFinished
```

```
| IsEnded
deriving (Show, Eq)
```

2.1.2 The Player

The player is a SpaceShip. The SpaceShip can move around and shoot asteroids to destroy them. The player needs to know it's position, as well as his velocity to move around. The rotation of the player decides which way the bullets are fired. This is not to be confused with the direction field.

```
data SpaceShip = SpaceShip {
          rotation           :: Rotation
        , playerPosition     :: Position
        , playerVelocity     :: Velocity
        -- the size of the player
        , playerRadius       :: Float
        , direction          :: Direction
        , bullets            :: [Bullet]
}
```

The direction field decides whether it should rotate or move forward. The direction type goes as follows:

```
type IsLeft = Bool
type IsRight = Bool
type IsForward = Bool
type Direction = (IsLeft, IsRight, IsForward)
```

As mentioned earlier, the objective of the game is to score points. This can be done by shooting down asteroids. The bullet is a separate data type as well. Bullets need a position, as well as velocity to know which way to go and how fast.

```
data Bullet = MkBullet {
          bulletPosition :: Position
        , bulletVelocity :: Velocity
        -- the size of the bullets
        , bulletRadius   :: Float
}
```

2.1.3 Player interaction

The player can only move forward. However, the player can tilt his spaceship, thus changing the direction. We have one function that moves the player. However, this function makes use of extra functions that are needed to control the player to it's fullest. The player can use the "w" "a" "d" keys to move the player.

```
-- the function that enables it all
movePlayer   :: GameState -> GameState

rotateLeft   :: GameState -> GameState
rotateRight  :: GameState -> GameState
moveForward  :: GameState -> GameState

-- velocity should be updated based on the player's rotation
updateVelocityRight :: Rotation -> Velocity -> Velocity
updateVelocityLeft  :: Rotation -> Velocity -> Velocity

-- helper functions to keep the rotation within 0 and 360
-- this is important for the updateVelocity functions
moddedRotation              :: Rotation -> Int
increaseTillBetween0And360  :: Float -> Float
```

The space key can be used to shoot bullets, as well as the left mouse button.

```
addBullet     :: Position -> Velocity -> [Bullet] -> [Bullet]
moveBullets   :: GameState -> GameState
-- helper func
moveBullets'  :: [Bullet] -> [Bullet]
```

2.1.4 The enemies

The enemies of the player are the asteroids that spawn randomly, and ufos that can move and shoot at the player.

```
data Asteroid = MkAsteroid {
          asteroidPosition   :: Position
        , asteroidVelocity   :: Velocity
        , asteroidRadius     :: Float
}

data UFO = MkUFO { ufoPosition :: Position
        , ufoVelocity :: Velocity
        , ufoRadius   :: Float
        , ufoBullets  :: [Bullet] }
```

When an asteroid or a ufo is shot down, an animation will play. The asteroid that is destroyed will be "moved" to an explodedAsteroid.

```
data ExplodedAsteroid =
MkExplodedAsteroid {
          explodedAsteroidPosition :: Position
        , explodedAsteroidRotation :: Rotation
        , explodedAsteroidSize     :: Float
}
```

The asteroids that spawn, are spawned randomly. Asteroids follow a straight line, so the velocity will never change. The asteroids' direction is always towards the player to abide by the requirement of minimal intelligence. The asteroid's velocity uses the player's position and a random margin to decide direction. The velocity is also dependent on the time. The longer the game goes on, the faster the asteroids get.

```
addAsteroid     :: Position -> Velocity -> Float -> [Asteroid]
                   -> [Asteroid]
-- gets a random asteroid position
getAsteroidPos  :: Float -> IO Position
-- gets a random asteroid velocity based on it's own
-- spawn position and the player's position and the time
getAsteroidVel  :: Position -> Position -> Float -> IO Velocity
```

2.1.5 Enemy movement

The asteroids move based on the asteroid velocity.

```
moveAsteroids :: GameState -> GameState
-- helper function
moveAsteroids :: [Asteroid] -> [Asteroid]
```

The ufos move diagonally. The ufo's velocity keeps getting updated when the ufo reaches a certain position.

```
updateUfoVel :: GameState  -> GameState
-- helper function
updateUfoVel' :: UFO -> UFO
```

When asteroids get shot down, an animation will occur.

```
explodedAsteroidsAnimation   :: GameState -> GameState
explodedAsteroidsAnimation'  :: [ExplodedAsteroid]
                                -> [ExplodedAsteroid]
```

2.2 Interface

The initial display is just a small window.

```
initialDisplay :: Display
```

The player will interact with the game through a window where the player and the enemies will be drawn on. It can be passed to the main function as an IO.

```
view      :: GameState -> IO Picture
viewPure  :: String -> GameState -> Picture
drawing   :: String -> GameState -> Picture
```

The drawing function uses gloss' pictures data type to format the elements into drawings on the screen. The elements must be pictures in order for it to be able to display on the screen.

```
bulletPictures           :: [Picture]
asteroidPictures         :: [Picture]
explodedAsteroidPictures :: [Picture]
ufoPictures :: [Picture]
showState                :: PlayState -> Picture
```

2.3 Implementation of the Minimum Requirements

Player The player can control the SpaceShip. The SpaceShip is able to move around and shoot players. The player gains score when it destroys an enemy

Enemies The enemies will try to take down the player. This can be done by the player bumping into the asteroids or ufos. The asteroids have been implemented in a way with certain intelligence, by knowing the position of the user. The ufo shoots down the player based on it's position.

Randomness The enemies spawn at random places. The enemies move towards the player with a random margin.

Animation When an asteroid or ufo is shot down, an animation will display to show the player that the asteroid has been destroyed.

Pause The player can pause/unpause the game by pressing "p".

IO The highscore can be viewed using IO. When the player gets a score higher than the highscore, the highscore will then be updated.

2.4 Implementation of the optional Requirements

Mouse input The player can shoot using the mouse button.

Different enemnies The player has two enemies, one is an asteroid. The asteroids move in a straight line to the player's position when it spawns. When the asteroid hits the player, the player dies. The other enemy is the ufo. The ufo moves diagonally, and aims to shoot at the player. If the bullet of the ufo collide with the player, the player dies. The player also dies if it collides with the ufo.