Documentation Assignment 5

Martin Boers (5676401) en Florian van Strien (5632757)

Basic framework

We used lenses from `Control.Lens` and a state monad to maintain the world state. This allowed us to easily update and retrieve "variables".

We created different data types for the different elements of the game, like the player, enemies and bonuses. These data types are then used in a `World` data type which contains the full state of the world. We then used the `makeLenses` function to create lenses from these data types. We also created some initialization functions to be able to easily create each data type.

The lenses, along with the state monad, allow to program quite imperatively in some places (using the do notation), while still profiting from the advantages of Haskell and executing everything in a specified order.

We kept the Model/View/Controller separation that was used in the original framework. Our lenses are created in Model, updated in the Controller, and drawn in View.

We added some helper functions we use throughout the project to the Helper module.

Collider Class

To help us with collisions between bullets and other objects, we have also added a Collider class, with instances for bullet collision with enemies and with bonusses, these can be found in the Helper module.

playIO

We used `playIO` from Gloss instead of the standard play function. This allows us to use IO to save and retrieve highscores. We use a stateTransformer (StateT) to change the world with IO where needed and a MonadState World to change the world on places where we do not need IO to modify the state using the state monad.

Extras

Highscore

The game contains a highscore function. When the game starts, the highscore will be loaded from a text file using the IO monad. Then, when the player reaches a higher score than their current highscore, this new score will be written to the text file.

The high score is always visible while playing. It's also shown in the main menu when the player dies.

The highscores are handled in the Highscores module.

Menu

The first thing you'll see when you open the game is the menu. The player can choose to start or quit the game here. The menu is also easily expandable, so more buttons could be added if needed. When the player dies, they will be returned to the menu. However, it'll now show some different texts (like "Play Again" instead of just "Play"). It'll also show the score the player got and the current highscore.

The menu is updated in the MenuUpdate module and drawn in the MenuView module.

Additional enemy type

The game contains two enemy types:

- An enemy which moves in the direction of the player when it's created, but doesn't ever change course. This enemy uses a randomly generated sprite, using an algorithm to always create an asteroid-like shape. It also moves at a random speed.
- An enemy which keeps following the player. This enemy has a single sprite. It always moves at the same speed.

Both enemies are based on the same Enemy lens with different initialization functions, using the movementType lens to keep them apart. Players will need to take the different behavior of the enemies into account.

Asteroids spawning smaller asteroids

Large asteroid enemies spawn four smaller asteroids when they are destroyed, which all move in random directions. This makes destroying these asteroids an extra challenge. Before you know it, the whole playing field is filled with asteroids!

Particles

A single lens is used for all particles in the game. This makes it possible to easily add particles for different situations. We added the required particles, but we also added a particle effect that is shown whenever an enemy is destroyed. This particle effect is based on the shape of the enemy, resulting in a unique explosion animation for each asteroid.

Lives

The player has three lives. The player only loses when these lives are all expended. However, losing a life is never without consequence, as the score multiplier is always reset. The player does get a second in which they are invincible after each death to recompose. During this time, the player ship is drawn transparently.

Movement wrapping

Whenever the player ship moves outside of the screen, it is wrapped to the other side of the screen. The same happens for enemies. We chose not to do this for bullets, as that wasn't very clear for the player.

Moving backwards (backthrust)

The player can move backwards by using the down arrow key. We added this based on our experience playing the game; this allows the player to escape from some dangerous situations. However, it's much slower than moving forwards. A special particle effect is used for this action.

HUD

A simple head up display is used, displaying the score, score multiplier and current highscore. The amount of lives is also shown.