# Design Document

Our game, going by the name of “Metal Dog 3”, will be a Shoot‘em up type of game in which the player controls a dog inside of a spaceship shooting all the evildoers of doghoodness. It’s an open-end arcade game in which the goal is to destroy as many enemies as possible and survive as long as possible. A striving for the highscore.

# Minimal Requirements:

## Player

Metal Dog 3 will have a controllable dog inside of a spaceship. His 2D movement will be controlled with the arrow keys and shooting will be actioned by clicking the space bar.

## Enemies

The enemies come in various shapes and sizes ranging from Cars to Vacuum-Cleaners. They can damage the player by colliding into him. The different enemies have different movement patterns, movement speeds and reward a different amount of points each. The “minimal intelligence” was implemented by having heat-seeking fireworks, which move directly towards the player once spawned.

## Randomness

The spawn locations of the enemies and the type of enemy being spawned will be randomly generated.

## Animation

The projectiles shot by the player character will leave yellow trails behind the bullet, which rapidly lose opacity and enhance the look and feel of firing a round.

## Pause

The game can be paused by pressing the ‘p’ key (It can also be restarted by pressing the ‘r’ key).

## Interaction with the file system

The achieved highscores will be saved to a json file alongside the players name. If a file does not exist, one will be created.

# Optional Requirements:

# Different enemies

Our game has 5 different enemy types, being cars, cats, fireworks, postmen and vacuum cleaners. These enemies have different appearances and different movementspeeds and -patterns. All enemies are equally easy to kill, though the difficulty ramps up the longer the game progresses, meaning that more enemies will be spawned.

# Use JSON to save information

Our highscore saves are handled via the JSON format, using the Aeson library.

## Schedule

|  |  |  |
| --- | --- | --- |
| Week |  |  |
| 1 (22 – 28 October) | -Make a moveable player, which can shoot.  -Implement pausing of the game | -Learn how to draw graphics on screen  -Learn how to convert user input to movement of an object on screen  -Learn how to create projectiles  -Learn how to pause a game. |
| 2 (29 – 4 November) | -Implement enemies  -Implement highscore  -Implement end of game | -Make enemies appear at random intervals and locations  -If an enemy dies, add his reward to the score.  -When player dies, stop the game.  -If the players’ score is higher than the highscore, save the new highscore to a file.  -When the game is started, read a file containing the highscore. |
| 3 (5 – 11 November) | -Implement extra’s  -Bug fixing  -Polishing | -Learn about writing design documents |

# Modules, data types and type classes:

# Modules:

Our game consists of twelve modules. The modules Config, GameTypes and GenericTypes contain all the base game information and data types alongside the type classes being used. The only modules which handle impure operations is our Main and our Controller module. The View and Model modules handle the rendering of the game objects and the game calculations. All the other modules represent data types in our game.

## Data types:

The data types are used to modularize and describe the games objects, such as projectiles, the player, enemies, particles and weapons. Some default data types are implemented in the GenericTypes module to help with the calculations.

## Type classes:

Three type classes have been implemented into our game, in order to allow unified operation on the different data types, these being Moveable, Damageable and Mathable. Moveable allows for retrieval of position, a move function and out-of-bounds checks. The Damageable type class is not currently in use, but would allow for unified handling of health subtraction. Mathable implements some basic mathematical functions like addition and multiplication for our computational data types like Score, Vector and Speed.

# Implementation Process:

To make it easier to collaborate on the game code we used git as our version control tool, and Github as our repository server. We made use of the issue and milestone functionalities provided by Github to better structure our work and always have a clear goal in mind. The repository can be found under this link: https://github.com/RobertMiriuta/Metal-Dog-3

## Known bugs

Cabal has some trouble on one of our machines with building/running the application. But building through GHC and using the .exe does work well. Main.hs is the file to be build, which can also be seen in the cabal config.