1.1

Fish	
Superclass(es): -	
Subclasses: LaserShark, Fishbot	
Contains the information about all Fish	

Following from the must-have requirements is the Fish class. This class forms the basis for both the LaserShark and the Fishbot classes which we will need for our characters in the game.

LaserShark	
Superclass(es): Fish	
Subclasses: -	
Contains the information about the shark	
Knows the position of the shark	Position

The LaserShark class is the class that contains our LaserShark object. It inherits from the Fish superclass.

Fishbot	
Superclass(es): Fish	
Subclasses: -	
Contains the information about the Fishbots	
Knows the position of the fishbot	Position

The Fishbot class is the class that contains our Fishbot object. It inherits from the Fish superclass.

GUI	
Superclass(es): -	
Subclasses: -	
Showing the game to the player	Screencontroller
Contains information about the level	Level
Knows the position of everything on the board	Position
Knows everything about the fishbots	Fishbot
Knows everything about the shark	LaserShark
Updates accordingly to the user input	KeyBoardController

The player needs to see the board and everything that happens there. The player controls the LaserShark and must be able to see the updates in position of both the Fishbots and the LaserShark.

KeyBoardController	
Superclass(es): -	
Subclasses: -	
Accepts user input	EventHandler

The keyboardcontroller class is the class that accepts the user inputs so that they can be used in the GUI class. This follows directly from the requirement that the player has to be able to move his shark.

Level	
Superclass(es): -	
Subclasses: -	
Contains information about the level	Position
Contain information about the shark	LaserShark
Contain information about the fishtbots	Fishbot

This class is needed to keep track on everything that is on the board and to bring all the classes together.

Position	
Superclass(es): -	
Subclasses: -	
Contain The position that objects on the board can have	
Has a position x	Integers
Has a position y	Integerss

The position class returns an object containing the x and y coordinate of the position of the fishes on the board. This position is used to keep track of the fishes and determine possible collisions.

Differences between our actual implementation:

Although implementing the classes in the way as shown above will satisfy all the must-haves of our requirements, our actual implementation is a bit different. In the actual implementation there are some additional classes. Two of these classes are the fishcontroller and the screencontroller. These classes are used to ‘control’ the fishes and the screen. As the name suggests this controlling includes how the objects from the controlled classes interact with each other. On first sight these classes might not seem necessary, but in practise these classes proved very useful. There was also the direction class which did not seem that obvious from this point of view. The class contains the possible directions of the Shark and the updates in those positions.

1.2

The main classes are the Fish, FishBot, FishController, KeyboardController, LaserShark, Position and ScreenController classes.

Responsibilities and collaborations of the Fish class:
The Fish class provides and maintains knowledge about the position, size, speed and direction of a fish and it says whether or not the fish is alive. The Fish class is responsible for actions like increasing the size of a fish, moving the fish, and checking for collisions with other fishes.
The fish class is used by the ScreenController class to add fishes to the controller. The class has two subclasses, the FishBot class and LaserShark class.

Responsibilities and collaborations of the FishBot class:
The FishBot class generates fishes with random values and is used to spawn fishes on either the left or right side of the screen. The FishBot class is a subclass of the Fish class and has a Is-Kind-Of relationship with the LaserShark class.

Responsibilities and collaborations of the LaserShark class:
The LaserShark class is also a subclass of the Fish class and as described above, this class has a Is-Kind-Of relationship with the FishBot class. The LaserShark class has actions for increasing the size of the shark when eating a fish and a method for moving the shark.

Responsibilities and collaborations of the FishController class:
This class provides and maintains knowledge about a list of fishes, a random value for generating fishes and a float value for the chance of spawning fishes. The actions this class performs are adding fishes to the list of fishes, update the positions of all the fishes, it checks the information in every cycle and this class checks for collision between the lasershark and other fishes.

Responsibilities and collaborations of the KeyboardController class:
This class is responsible for all the key events. When a key is pressed in a certain direction, this class will give that direction, so that the shark will swim in the direction which is given.

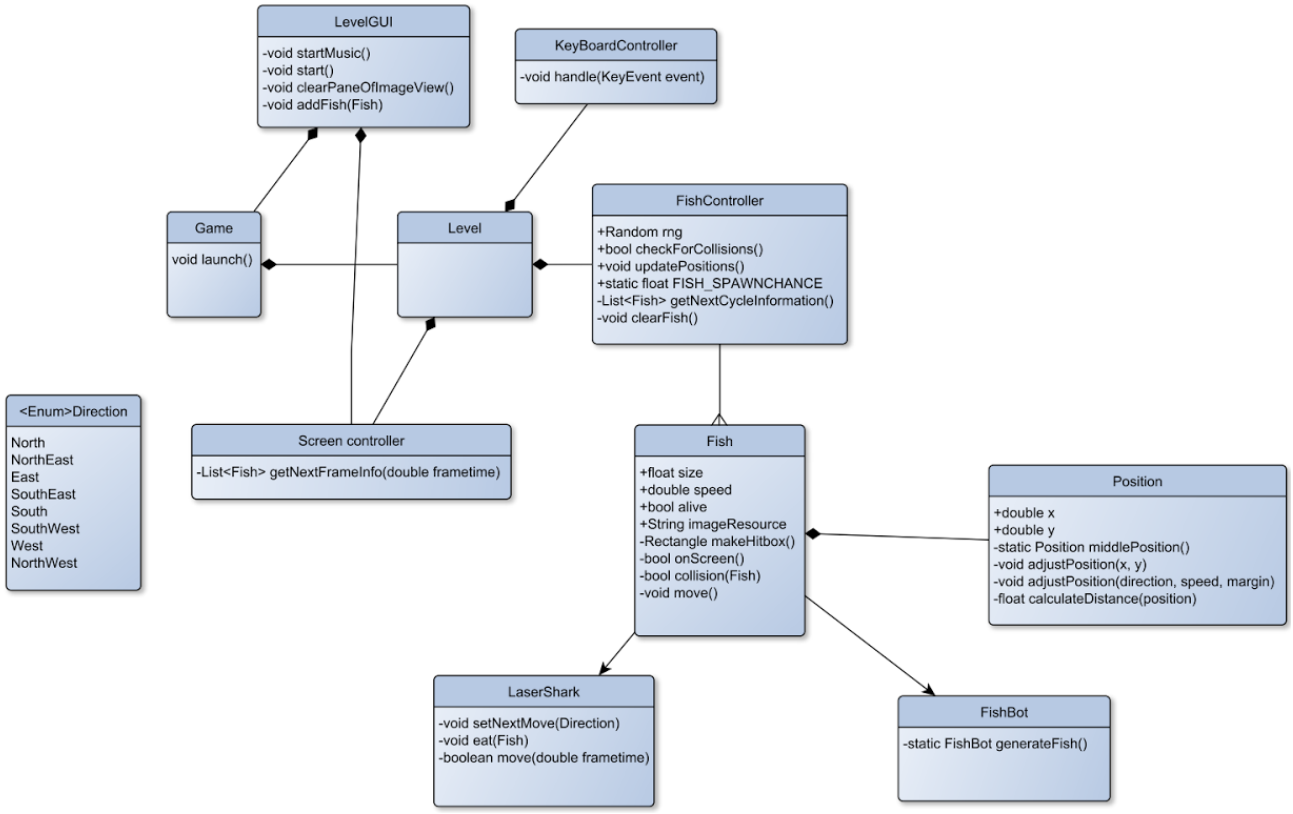
Responsibilities and collaborations of the Position class:
This class provides and maintains the knowledge of a position x and y. Every fish has a position and this position is used to display the fish on the screen. The position is also used for calculating the distance between fishes, so that it can check for collisions.

Responsibilities and collaborations of the ScreenController class:
This class provides and maintains the knowledge about the level, the gui and the scene. It checks every cycle if the lasershark is alive and if it has reached its winning size. When the lasershark is not alive, it sets the lose screen in the gui to true. When the lasershark has reached its winning size, it sets the winning screen to true in the gui.

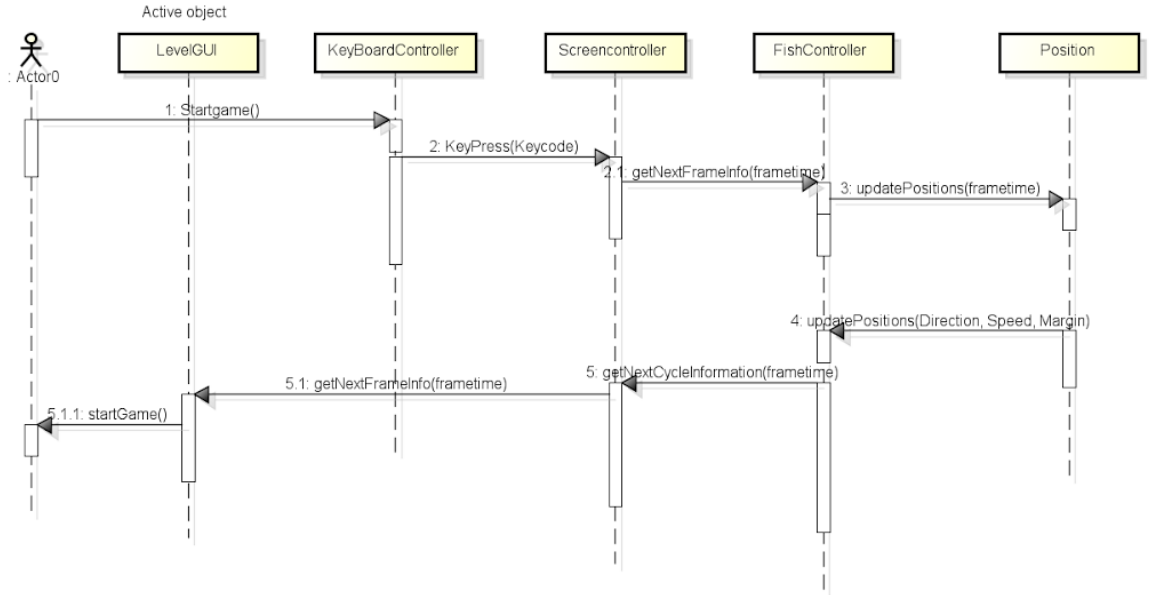
1.3

The less important classes are the Direction, Game and Level classes.
The Direction class is a class to store all the possible directions, it is especially used by the KeyboardController class and some other methods. The Direction class has a constructer with a delta x and y. The direction class is not a main class, but it gives more clarity in the implementation.
The game class is used to launch a level. It is important to launch a level, but it is possible to launch the level directly from the gui, so this class is redundant. We will merge and remove this class.
The Level class provides and maintains knowledge about the FishController, the LaserShark, the ScreenController and the KeyboardController. This class has, besides getters and setters, no actions. But we need the knowledge in this class for other classes, so this class cannot be removed.

1.4



1.5



2.1

In aggregation an object contains another object and the contained object doesn't need the containing object to exist. In composition an object also contains another object, but the contained object cannot exist without the containing object.

We use aggregation for the Position class, because any fish object needs to have a position, but there are also positions that are separate from fishes. Everything else is a composition, because a fish cannot exist without the fishController, the fishController cannot exist without the level, and the level cannot exist without the screenController which cannot exist without the levelGUI, the keyboardController also cannot exist without the level class.

2.2

We did not use any parameterized classes in our main package.
We could have used a parameterized class for position, with either a double or int as input. We didn't think this was necessary, as the functionality of integers and doubles is, for this class, the same.

2.3

The game class should be removed. We can clearly see in our UML how the game class is pretty useless as the main method is called from the LevelGUI class.
Doing this will not have big changes to our structure as there are few classes that rely on the Game class.