Exercise 1 - The Core

1) CRC Cards

The CRC (Classes, Responsibilities, Collaborations) cards for our classes.

Classes:

- Point: has coordinates
- Abstract class GridUnit: has intersection method
- Movable interface
- GridUnit extends Point
- MovableGridUnit extends GridUnit implements Movable
- Wall extends GridUnit
- Stix extends GridUnit has array of Points
- Area extends GridUnit has attribute PointState
- Draw interface
- abstract class SpriteUnit extends gridUnit and implements interface draw

- levelFactory makes Levels
- GridFactory makes GridHandler
- GridHandler has all the points with there PointState, has attribute AreaTracker
- CollisionHandler handles collision between GridUnits
- Level has attribute GridHandler and CollisionHandler
- Gamescene has ScoreHandler, SoundHandler, DrawHandler, InputHandler and Level
- PointState enum
- ScoreHandler
- SoundHandler
- DrawHandler has attribute canvas, , handles drawing of the grid
- AreaTracker calculates the areas
- InputHandler

GridUnit	
Superclass(es): Point	
SubClasses: MovableGridUnit , Wall and Stix.	
Returning collision points	Points

SpriteUnit	
Superclass(es): Point, GridUnit, MovableGridUnit	
SubClasses: Player, Sparx and Fuse.	
Drawing the unit	Drawhandler, Sprites/Image and canvas

GridFactory	
Superclass(es): none.	
SubClasses: none.	
Create a grid	GridUnit, PointState

LevelFactory	
Superclass(es): none.	
SubClasses: none.	
Create a level	Level, GridHandler and CollisionHandler

Returning a gridHandler	gridHandler and areatracker

Specifying different levels(amount of units and level difficulty etc.)	Level

GridHandler	
Superclass(es): none.	
SubClasses: none.	
Adds and deletes GridUnits	GridUnit
Changes states of GridUnits	GridUnit and PointStates

CollisionHandler	
Superclass(es): none.	
SubClasses: none.	
Checks for collision between units	GridUnits

Level	
Superclass(es): none.	
SubClasses: none.	
Handles all the elements that are different within a level	GridHandler and CollisionHandler

GameScene	
Superclass(es): none.	
SubClasses: none.	
Displays score	ScoreHandler and Level
Plays sound	SoundHandler and Level
Draws the grid	DrawHandler and Level

ScoreHandler	
Superclass(es): none.	
SubClasses: none.	
Calculates score	Level

AreaTracker	
Superclass(es): none.	
SubClasses:none.	
Calculates covered	Grid

area	

SoundHandler	
Superclass(es): none.	
SubClasses: none.	
Plays audio	3PMs

DrawHandler	
Superclass(es): none.	
SubClasses: none.	
Draw the game on the canvas	Canvas and grid

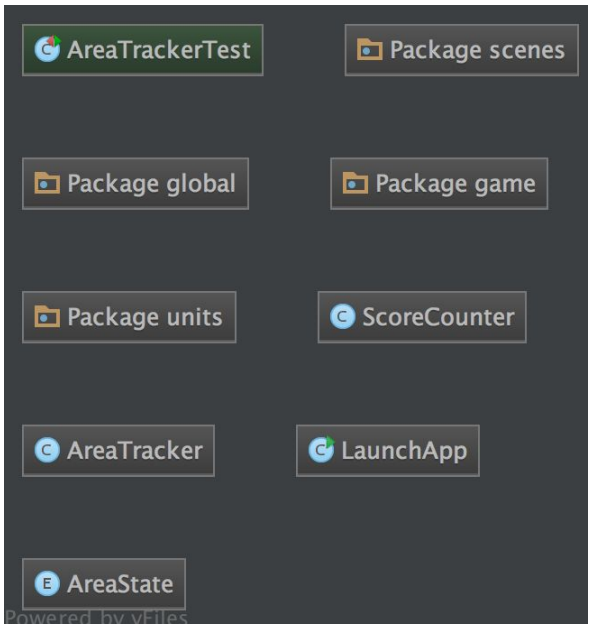
InPutHandler	
Superclass(es): none.	
SubClasses: none.	
Takes input from user	User input/keyboard
Moves player	player

MovableGridUnit	
Superclass(es): Point, GridUnit	
SubClasses: Qix and SpriteUnit	
Moves GridUnit in the grid	GridUnit and GridHandler

2) Main Classes & Responsibility Driven Design

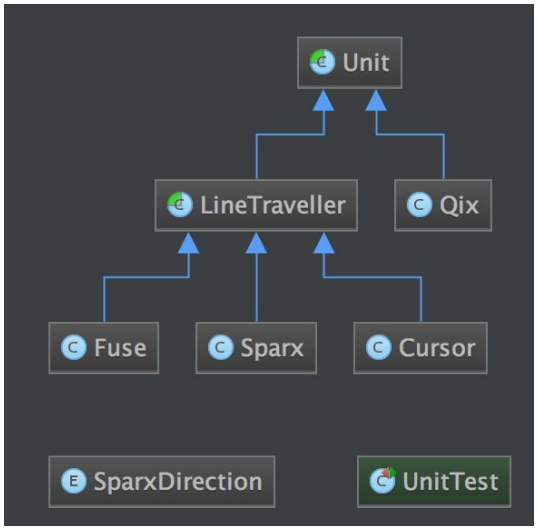
The following contains a description of all responsibilities and collaborations per class in the system. The classes are ordered per package.

nl.tudelft.sem.group2



	Responsible for...	Collaborates with...
AreaTracker	- Marking all points on the map with an AreaState - Calculating scores when new areas are claimed - Updating score counter - Removing Stix when areas are completed - Checking move directions of Stix - Performing FloodFill algorithm - Printing board grid state to console for debugging	- Contains a list of Points which describe the borders - Has a 2d area of AreaStates
LaunchApp	- Launching the game window containing a GameScene and ScoreScene	- Sets a GameScene on the primary stage

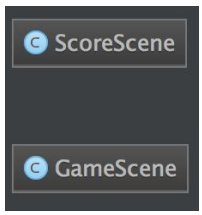
nl.tudelft.sem.group2.units



	Responsible for...	Collaborates with...
Unit	- Positioning on the screen - Collision calculations - Tracks width and height	- Contains an AreaTracker - Handles collisions with Qix and
LineTraveller	- Checks for innerborders, outerborders and uncovered areas - Drawing sprites on the screen - Tracks which sprite to draw on what frame	- Extends Unit - Uses AreaTracker to obtain AreaStates
Qix	- Movement - Changing colors - Drawing itself on the screen	- Extends Unit - Checks for collisions with Line
Fuse	- Movement with a certain set speed - Movement - Drawing Fuse sprites - Updating speed (moving/standing still)	- Extends LineTraveller
Sparx	- Movement in a certain direction	- Extends LineTraveller

	- Drawing Sparx sprites	- Direction to move at creation is one of SparxDirection
Cursor	- Responding to up/down/left/right/X/Z keycodes from player - Drawing Cursor sprite - Movement (left/right/up/down) - Calculating line coordinates - Showing an animation at start of the game - Travelling at slow or high speed	- Extends LineTraveller - GameScene passes the keycodes to the Cursor - Asks LineTraveller which way it may move

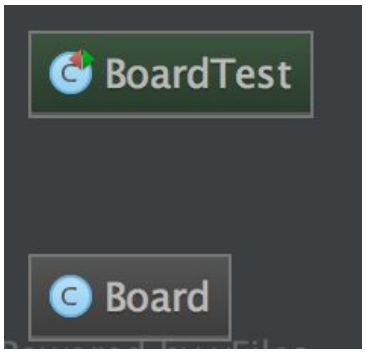
nl.tudelft.sem.group2.scenes



	Responsible for...	Collaborates with...
ScoreScene	- Displaying current score percentage - Displaying current score amount - Displaying title logo of the game -	- Extends Canvas
GameScene	- Displaying all Units of the Board - Showing game progress (game over/game start/game won) - Adding Sparx, Qix, Cursor to the Board - Playing game start/game over	- Extends Canvas - Has a list of Units - Has a Qix - Has a Board - Has a ScoreScene - Has a ScoreCounter

	sounds - Registering keyboard event handlers - Moving all Units every frame - Drawing all Units every frame - Calculating new claimed area percentages	

nl.tudelft.sem.group2.game



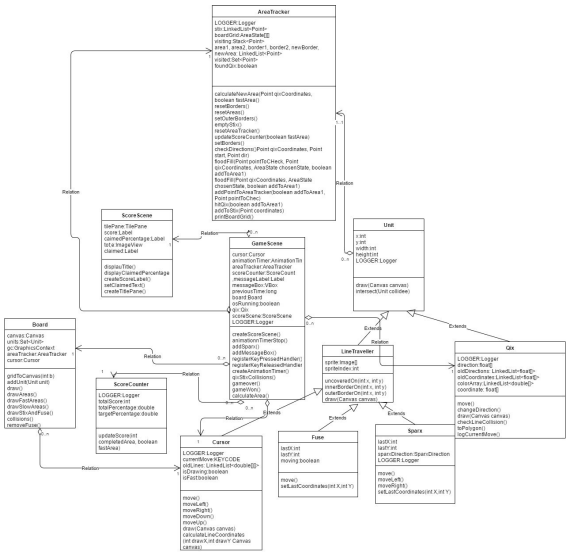
	Responsible for...	Collaborates with...
Board	- Keeping a set of all Units which are currently in the game - Having only one Fuse at maximum - Drawing all Units on the canvas - Drawing all Stix, fast areas, slow areas on the canvas - Setting background color on the canvas	- Has an AreaTracker - Has a Cursor - Has a set of Units - Has a Canvas

3) Other classes

The Enums AreaState and SparxDirection are not marked as main classes. These Enums only describe the states an area in the game can have and the directions the sparx can go to. The Enum AreaState is used in many different classes and brings consistency throughout them. SparxDirection however, is only used by the Sparx classes and will in the future most

likely not be shared by other classes. For this reason, we have decided to merge the SparxDirection Enum into the Sparx class.
The Globals class, as part of the nl.tudelft.sem.group2.globals package, is not marked as a main class because it contains no functionality. It only contains static definitions for other classes, like the width and height of the Board, or the start position coordinates of the Qix. In the future, when this Globals class becomes too large, it may be a good idea to split its definitions into multiple of these classes. For the moment, it works fine for us. It keeps the codebase more maintainable.
The ScoreCounter is not marked as a main class either. The reason for this is that it has similar functionality as defined in ScoreScene. In the next sprint, we plan to better define and split the functionalities between ScoreCounter and ScoreScene.

4) Class Diagram



For bigger view It’s is the folder Diagrams

5) Sequence Diagram

These are also in the folder Diagrams. We made each different scenario. This document would be to long to have all picture of them.

Exercise 2 - UML in practice

1) Aggregation & Composition

Explanation of difference between aggregation and composition
Imagine there are three classes, A, B and C. Class A contains B and C as attributes. In composition, when any instance of class A gets destroyed, all instances of B and C as attributes of instance of class A, will be destroyed. In composition, those instances will not be destroyed.
In our game

In GameScene, mainly composition is used. All the attributes of GameScene share the same lifespan as the GameScene. Without GameScene, none of its containing objects can “live on”. The Board class works differently. The Board class contains a set of units, which is simply a reference. The units can live on their own if an instance of the Board class gets destroyed. Therefore, this uses aggregation.

2) Parametrized Classes in source code

These classes are parametrized:

```
ArrayList<Unit> unitsList =  new ArrayList<Unit>() ;
```

This is a list of units but it accepts all subclasses of Unit. This is parametrized because then we can loop through all units and call the move() and draw() functions of the units with one for-loop. This can be done because all units share these two functions. The same thing can be done with the collisions between the units.

3) Class diagrams for all the hierarchies

The hierarchies are allready show in the class diagram exercise 1.4.

Exercise 3 - Logging

1) Logging Requirements

It is possible to choose the level of logging, level of logging: Detailed, Normal and Off.
Logging to external file
Level logging: Off nothing should be logged
Level logging: Normal the actions which trigger logging
- Collisions
- Completing a area
- The starting and ending of the game

Level logging: Detailed the actions which trigger logging
- The actions of Normal logging
- Moving player/cursor