Assignment 4 – Group 23

Exercise 1

1. **Requirements (Levels):**
   Must haves:
   - The game shall have multiple levels.
   - The game shall advance to the next level when there are no aliens left.
   - The game shall end when there are no more levels left.
   - Each level shall increase the difficulty.

2. **CRC Cards (Levels):**

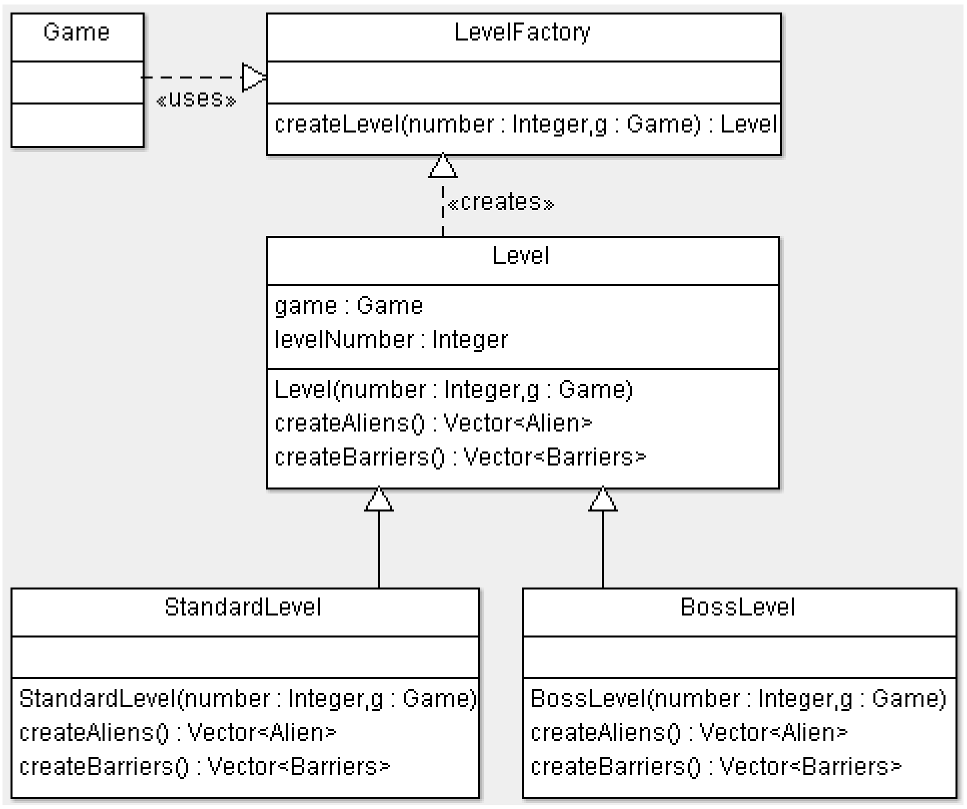
Class name: Level	
Superclass: /	
Subclasses: StandardLevel, BossLevel	
Purpose:	**Collaborators:**
Create level	Game
Create Aliens for in the vector	Alien
Create Barriers for in the vector	Barrier

Class name: LevelFactory	
Superclass: /	
Subclasses: /	
Purpose:	**Collaborators:**
Provide the Game class with levels	StandardLevel, BossLevel, Game

Class name: StandardLevel	
Superclass: Level	
Subclasses: /	
Purpose:	**Collaborators:**
Create level	Game
Create Aliens for in the vector	Alien
Create Barriers for in the vector	Barrier

Class name: BossLevel	
Superclass: Level	
Subclasses: /	
Purpose:	**Collaborators:**
Create boss level	Game
Create Boss Alien for in the vector	Alien
Create Barriers for in the vector	Barrier

3. UML (Levels):


Game
LevelFactory
«uses»
createLevel(number : Integer,g : Game) : Level
«creates»
Level
game : Game
levelNumber : Integer
Level(number : Integer,g : Game)
createAliens() : Vector<Alien>
createBarriers() : Vector<Barriers>
StandardLevel
StandardLevel(number : Integer,g : Game)
createAliens() : Vector<Alien>
createBarriers() : Vector<Barriers>
BossLevel
BossLevel(number : Integer,g : Game)
createAliens() : Vector<Alien>
createBarriers() : Vector<Barriers>

4. **Requirements (Boss Alien):**

Must haves:

- The game shall have a Boss alien.
- Boss alien shall have a higher score than the rest of the aliens.
- Boss alien shall disappear if hit multiple times by spaceship bullet.
- Boss alien shall go from left to right and down if they hit a border.
- Boss alien shall be the only alien in the screen for that level.

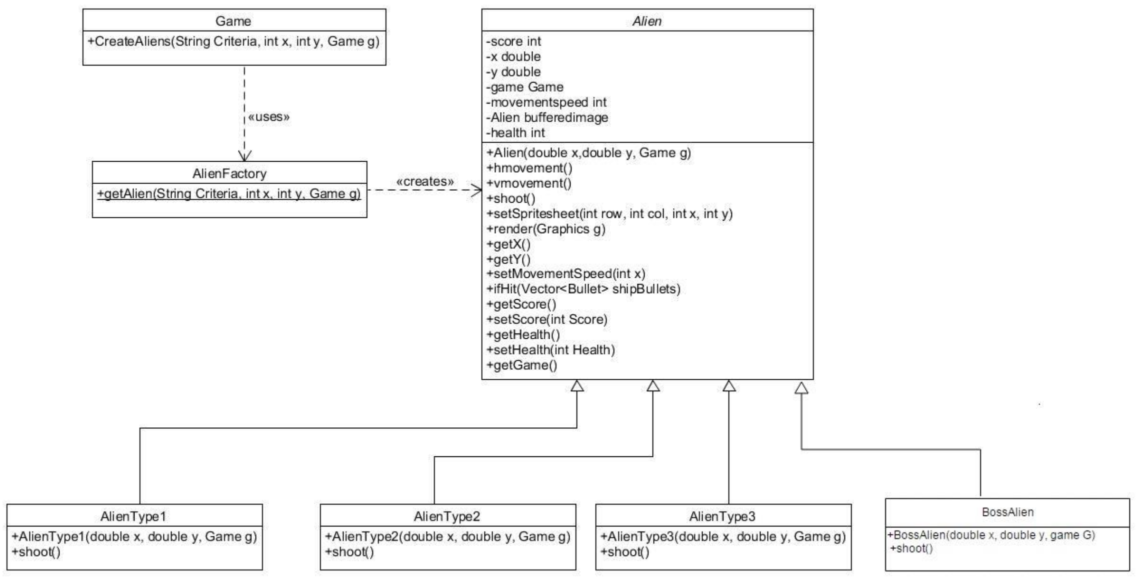
Should haves:

- Boss alien shall appear every five levels.
- Boss alien shall be able to shoot downwards.
- Boss alien shall be able to shoot different types of bullets.
- Boss alien shall be able to shoot multiple bullets at the same time.

5. **CRC Cards (Boss Alien):**

Class name: BossAlien	
Superclass: Alien	
Subclasses:	
Purpose:	**Collaborators**:
Superclass to provide the alien type	
Shoot bullets downwards	Bullet
Disappear when hit	Game
Move from left to right	Game
Appear every five levels	AlienFactory, Levels

6. **UML (Boss Alien):**


Game
+CreateAliens(String Criteria, int x, int y, Game g)
«uses»
AlienFactory
+getAlien(String Criteria, int x, int y, Game g)
«creates»
Alien
-score int
-x double
-y double
-game Game
-movementspeed int
-Alien bufferedimage
-health int
+Alien(double x,double y, Game g)
+hmovement()
+vmovement()
+shoot()
+setSpritesheet(int row, int col, int x, int y)
+render(Graphics g)
+getX()
+getY()
+setMovementSpeed(int x)
+ifHit(Vector<Bullet> shipBullets)
+getScore()
+setScore(int Score)
+getHealth()
+setHealth(int Health)
+getGame()
AlienType1
+AlienType1(double x, double y, Game g)
+shoot()
AlienType2
+AlienType2(double x, double y, Game g)
+shoot()
AlienType3
+AlienType3(double x, double y, Game g)
+shoot()
BossAlien
+BossAlien(double x, double y, game G)
+shoot()

Exercise 2

Flaw 1: God class

The Game class was a god class because it was extensively large, had a lot of (complex) methods and used a lot of external data. The purpose of the game class was to coordinate all the different aspects of the game but this got a little out of hand when everyone kept assigning it new responsibilities and more and more code was added to it every iteration without having any order. It was also a mistake to give it the responsibility of handling all the graphic aspects as well.

Flaw 2: Schizophrenic class

Because the game class was very large it also became very non-cohesive. It was not structured at all. This was, just like the god class issue, due to the fact that we chose the game class as the class that was going to coordinate the aspects of the game and therefore we gave it too easily too many responsibilities. This made the class very complex and did not follow the rules of encapsulation.

Solution:

A lot of responsibility has been taken away from the game class. A new class called Screen has been created to handle all the graphical aspects and key input. The main while loop and main function have been taken out and put in a class Main. External data is not often used anymore, only a few get method calls remain. A lot of methods have been simplified or merged with similar methods and methods with responsibilities that could be done by another class have been moved. This made the game class shrink from 900 to 450 lines where about 100 of them are methods only used for testing. It also caused much better encapsulation and less complexity throughout the whole project.

NOTE: This is all implemented but because this caused changes in almost every class and changed the structure of the whole project, there was no time left to merge it with the new functionalities that were added. They also changed a big part of the project and merging them would take a lot of time. I did not want to risk ending up with one non-functional branch full of mistakes and I thought it would be better to have two branches that actually work but separately.

Potential design flaw: Brain method

Brain methods are alike to God class and could have affected our project by having extremely long methods that centralized the functionality of the entire system and that are hard to understand and debug and practically impossible to reuse due to long methods tend to do more than one piece of functionality and they are therefore using many temporary variables and parameters, making them more error-prone. Also excessive branching is in most cases a clear symptom of a non-objected design which ignores polymorphism.

This problem was avoided by refactoring the methods to have a considerable size, and each one only serves a specific function using a few amount of parameters and by being override to eliminate the code duplication.