# Assignment 1

**Exercise 1.1**

## Finding classes

Guidelines:

| | |
|---|---|
| Physical objects<br>Conceptual entities<br>One word for one concept<br>Adjectives<br>Check missing or misleading subjects<br>Interfaces<br>Attributes values not attributes | |

## Requirement specification and noun phrases

The space invaders **game** should start with an empty field and a **spaceship** at the bottom middle, after that 20 aliens should appear on the top in rows of 10, aliens can only move left or right and descend when reaching **the border of the screen**, multiple **types of aliens** should be available with different **score value** based on its type, including a **bonus alien**, only one **alien** can shot at the same time, alien's shoot must be only downwards, automatically and randomly. Aliens must disappear when hit and **the player** should earn points per each alien hit. The spaceship should be able to move left or right and should only **shoot** upwards (controlled by the player). When the spaceship is hit, the player should lose a live. **Protection barriers** shall appear between the spaceship and the aliens with the purpose of block alien bullets. Every time a protection barrier, a piece of it should disappear.

The player should start the game with multiple lives and multiple levels, the player should move to **the next level** when there are no aliens left to be destroyed and such level should increase its difficulty, the game should end when the player has reached the last level, when the spaceship is hit by an alien shoot and there are no more lives left or when at least one alien reaches the bottom of the screen.  The player should be able to **restart**, **pause** and **stop** the current game.

The game should show a **score board** which shows the amount of points earned by the player. The game should display **the game statistics** after losing showing the amount of aliens killed, the overall score and the highest level achieved. After completing a level, the game should display the **level statistics** showing the amount of aliens killed in such level, the total score and the current level. The game should also have a **list of high scores**. The game should not necessarily have a multiplayer game mode.

**Conceptual entities:**

- Game ~ restart, pause, stop
- Spaceship
- Alien ~ types of aliens, bonus alien
- Protection barriers
- Level

**Concepts:**

- Shoot ~ Alien shoot, spaceship shoot
- Score ~ score value, high score
- Statistics ~ game statistics, level statistics.

**Candidate classes:**

| | | | |
|---|---|---|---|
| Game | Spaceship | Alien | Protection barriers |
| Level | Shoot | Score | Statistics |

| | | |
|---|---|---|
| Physical objects<br>Conceptual entities<br>One word for one concept<br>Adjectives<br>Check missing or misleading subjects<br>Interfaces<br>Attributes values not attributes | Game<br>Spaceship<br>Alien<br>Protection barriers | Level<br>Shoot<br>Score<br>Statistics |

**CRC Cards**

| | |
|---|---|
| Class Name: Game | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Start game | |
| Move to next level | |
| End game | |
| Restart game | |
| Pause game | |
| Stop game | |

| | |
|---|---|
| Class Name: Spaceship | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Move left | |
| Move right | |
| Shoot upwards | Shoot |
| Have multiple lives (spaceship) | |
| Decrease amount of lives (spaceship) | |

| | |
|---|---|
| Class Name: Alien | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Shoot randomly | |
| Shoot downwards | Shoot |
| Disappear when hit | |
| Have different types | |
| Have bonus alien | |

| | |
|---|---|
| Class Name: Protection Barriers | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Block alien bullets | |
| Protect spaceship | |
| Disappear gradually when hit | |

| | |
|---|---|
| Class Name: Shoot | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Create alien bullet | |
| Create spaceship bullet | |

| | |
|---|---|
| Class Name: Level | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Create levels | |
| Increase level difficulty | |

| | |
|---|---|
| Class Name: Score | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Record score | |

| | |
|---|---|
| Class Name: Statistics | |
| Superclass(es): | |
| Subclasses: | |
| Purpose | Collaborators |
| Create game statistics | Score, level, alien |
| Create level statistics | Score, level, alien |

Discussion of results.

From the 8 resulting classes (the class Shoot was renamed as Bullet and the class Protection barrier as Barrier), only 2 have been partially implemented (Game, Aliens), and 3 fully implemented (spaceship, bullet, Barrier), and 2 more classes were added to deal with the sprite sheet and the images, however the other three classes haven’t been discussed for implementation as they are not part of the “Must have” requirements and were not delivered in the first working version of the game.

## Exercise 1.2:

The main classes of our project are: Game, Alien, Spaceship, Bullet, Spritesheet, BufferImageLoader.

The game class has 2 responsibilities. The game class has the responsibility of starting the game with an empty field with a spaceship in the bottom middle and 20 aliens in rows of 10 at the top. The game class also has the responsibility of ending the game when the necessary demands have been fulfilled.

The alien class has 6 responsibilities. Firstly the alien class has the responsibility of moving the aliens horizontally from left to right and vice versa. Secondly the alien class has the responsibility of when it hits a horizontal border to move vertically down and inverse the horizontal movement direction(e.g. from left to right becomes from right to left and vice versa). Thirdly the alien class has the responsibility of selecting a random alien object each game cycle. After selection the alien object has the responsibility to fire a single bullet. Fourthly the when an Alien is hit with a bullet from the spaceship it has the responsibility to die and be removed from the field. Fifthly when an Alien reaches the bottom of the screen it has a responsibility to notify the game class to end the game. Sixthly when there are no Aliens are left on the field the Alien class has the responsibility to notify the game class to end the game.

The Spaceship class has 4 responsibilities. Firstly when the Player presses left on his/her keyboard the spaceship object has the responsibility to move towards the left. Secondly when the player presses right on his/her keyboard the spaceship object has the responsibility to move towards the right. Thirdly when the player presses the spacebar on his/her keyboard the spaceship has the responsibility to fire a bullet. Fourthly when a spaceship is hit by an alien bullet it has the responsibility to notify the game class to end the game.

The Bullet class has 4 responsibilities. When the bullet is being fired by an Alien the bullet has the responsibility to move vertically down. Secondly when the bullet is being fired by a spaceship object it has the responsibility to move vertically upward. Thirdly when a bullet leaves the screen(aka the area that is viewable by the player) the bullet has the responsibility to be removed from the field. Fourthly when the bullet hits an object(either a spaceship bullet hitting an alien or an alien bullet hitting a spaceship) the bullet has the responsibility to be removed from the field.

The Spritesheet class has 1 responsibility. When the coordinates of an image are send to the spritesheet class the spritesheet will use those coordinates and return the exact image specified by these coordinates.

The BuffereImageLoader class has 1 responsisbility. When the file path of an Image(A string is used to carry this information) to the BuffereImageLoader class. The bufferImageLoaders responsibility is to use to grab the image specified in the file path and return it.

The collaborations between the main classes. In our project the collaborations are between the game class and every other class. This is because the game class contains the game loop so the game class tells the other classes when they should fulfil their responsibilities. These responsibilities are mostly able to be fulfilled by the class themselves however there are a few exceptions.

The collaboration between the game class and the alien class. There are 4 collaborations between alien and game. The first one is that like previously mentioned the game class is the class in control of the main game loop. Therefore it tells every other class when they should execute their responsibilities. So every cycle the game class tells the alien class to fulfil its horizontal movement responsibility. The execution of this responsibility is entirely done by the alien class itself. The second

collaboration is done in a similar method. Each game cycle the game class tells the alien class to select a random alien which will shoot. The random selection is done by the alien class and this information is returned to the game class, the game class will then give this information to the bullet class so it can fulfil its responsibility. The third responsibility is about the when the alien is hit. Every cycle the game class tells the alien class to check which alien objects have been hit by space ship bullets. The alien class tells the alien objects that have been hit to the game class and the game class then promptly removes these alien objects from the field. The final collaboration is when the aliens have reached the bottom of the screen, when this happens the game should end but the alien class itself does not have the ability to end the game(only the game class itself has that ability). So when the aliens have hit the bottom of the screen Alien gives this information to the game class, the game class then promptly ends the game.

The collaboration between game and spaceship. There are 4 collaborations between game and spaceship. The first three collaborations are about user input. The spaceship class relies on user input to know when to fulfil certain responsibilities. However the spaceship class itself can’t read user input but the game class can. So each cycle if the player gives a certain user input and this information is picked up by the game class and sent to the spaceship class so it can fulfil its responsibility. This is the case for left, right and spacebar. However in the case of spacebar the spaceship fires a bullet and returns this information to the game class. The fourth collaboration is about when the spaceship is hit. Every cycle the game tells the spaceship class to check and see if the spaceship was hit by a bullet. The spaceship class checks this and returns an answer. If that answer was yes than the game class decides to end the game to fulfil the spaceship class responsibility.

The collaboration between game and bullet. There are 3 collaborations between game and bullet. The first two are about the movement. When either an alien or a spaceship shoots the information is being send to the game class. So the game class ultimately keeps track of which bullets are fired from the spaceship and which are fired by aliens. It sends this information each cycle to the bullet class so the bullet class knows which bullets should move upwards and which should move downwards. The third collaboration is about when the bullets leave the viewable area. Each cycle the game class tells the bullet class when to check for the bullets to have left the viewable area. The bullet then sends the information back to the game class and then game promptly removes the bullet objects in question from the field.

The collaboration between bullet and spaceship and the collaboration between bullet and Alien. When the alien or spaceship has to calculate if it is hit by a bullet they need the coordinates of said bullet. The Alien/Spaceship class don't have that information so they gain that information from the bullet class(using the game class as a mediator).

The collaboration between Spritesheet and Alien/Spaceship/Bullet. Alien, Spaceship and Bullet are all the classes that are projected on the field and for that they need an image. To gain this Image they collaborate with the Spritesheet class. They send the Spritesheet the necessary information(the spritesheet image and coordinates) and the sprite sheet class will return the requested image that will be projected on the game field.

The collaboration between BuffereImageLoader and SpriteSheet. Whenever a class needs an image it calls the Spritesheet class. However the spritesheet class doesn't retrieve the original image it merely adjusts it so it can be used as a sprite on the game field. So to be able retrieve the image it collaborates with the BuffereImageLoader. It sends the string to BuffereImageloader and in return BuffereImageLoader returns the image specified.

## Exercise 1.3:

The non-main classes are Barrier and LogFile. LogFile is not a main class because its purpose is not directly integral to the function of the game. The responsibility of a LogFile class is to write the functional details of the game to an txt file which the developer will use to gain information about the functioning of the system. But it is not Integral to the functioning of the game therefore it is not considered a main class. It is not possible to merge LogFile with another class so it will not be changed. Barrier is the second class that is not considered a main class. The reason that it is not considered a main class is the same reason as LogFile because the barrier class is not integral to the basic functioning of the project. The Barrier class has the responsibility to add barriers on the field which will absorb the bullet class and this adds another level of complexity however it is not essential to the core gameplay. So the Barrier class will not be merged or changed but it will remain a non-main class.

**Exercise 2.1**

What is the difference between aggregation and composition? Where are composition and aggregation used in your project? Describe the classes and explain how these associations work.

Aggregation: A is a part of B, but A can exist without B.

Composition: A is a part of B, but A cannot exist without B.

Aggregations in our project:

-

Compositions in our project:

- Spaceship is a part of Game; The Spaceship only exist in the Game, not on itself.
- Alien is a part of Game; The Alien only exist in the Game, not on itself.
- Bullet is a part of Game; The Bullet only exist in the Game, not on itself.
- Barrier is a part of Game; The Barrier only exist in the Game, not on itself.

**Exercise 2.2**

We do not have any parametrized classes in our project. They can be useful to make instances of containers be able to store objects different types or for classes that appear in different types. We could integrate some kind of a parametrized class by modifying the Bullet class. There are currently two different types of bullets: the spaceship bullets that move up and are able to shoot over the barriers and the alien bullets that move down and are stopped when they hit a barrier. If this class would be parametrized, the class itself would know what move function to use (up or down). There is also the possibility to have more bullet types with different speed or damage if the game expands and different aliens are added.

UML: BulletParam.png in assignment1 folder

## Exercise 2.3

The game class is the class that communicates with the most other classes. First of all it needs a spaceship. There is only one spaceship per game and every spaceship belongs to only one game. The game also needs at least one spritesheet to create its screen and at least one alien. A spritesheet can belong to zero or more games but each alien belongs to only one game. The game class has vectors containing bullets. These vectors can be empty. To meet the requirements of exercise 3, the game has exactly one logfile. The classes game, spaceship, barrier, bullet and alien are able to write to this log file. To buffer images, the game class uses the BufferImageLoader class. Finally, the game can also have zero or more barriers that all belong to only one game.

It is not mandatory that the aliens and spaceship shoot bullets but when they do, a bullet object gets created and belongs to either an alien or a spaceship. (This is a contsraint but the UML program was not able to draw contraint lines).

To draw the graphics on the screen the spritesheet class was needed. Every other class except for BufferImageLoader and LogFile needs at least one (part of a) spritesheet to be visible for the player to see. A spritesheet does not need to belong to a certain object of a class.

UML: ClassHierarchies.png in assignment1 folder