!!Currently a JAR of our system doesnt work, It should however work on Linux And Windows through the IDE, we are working on a fix.

Exercise 1 - Anonymous peer suggestions (30 pts)
Last week you were asked to understand, read, and analyze the code of another anonymous group. In addition, you had to propose meaningful enhancements to the other's group codebase, for a total of 30 points.

1. Ask your TA for the peer feedback that he received about your group and implement the proposed enhancements. Your TA will make sure that all the enhancements are relevant to you current codebase. (30 pts)

If we had to implement a way to only update mouse- or key-driven events when pressing a button or clicking instead of just always checking if something is pressed we would do it in the following way:

We would have an class that observes if there are mouse-clicks or key-presses. It also has lists that hold all objects that do something on a mouse-click or key-press. (Depending on the size of the system, as lists for each key if there are a lot of key-driven events or one list with all key-press events to make it easier to handle in a smaller system.) When a key-press or mouse-clicks is observed it will call onClick() and onKeyPress() respectively on all objects in the appropriate list. These lists and their contents would use the observer pattern with the class that observes for mouse-clicks and key-presses as subject.

You do have to handle correctly that items are added to and removed from these lists

Exercise 2 - Software Metrics (45 pts)
The inCode tool uses software metrics to detect a number of design flaws. In this exercise you will use it to have guidelines to improve your implementation, from a code quality perspective.

1. Use inCode to compute software metrics on your project, then upload the resulting analysis file4 to your git repository. Write in the explanation PDF file where the analysis file is located (3 pts).

The analysis file is located in "docs\files\SEM-Project_1477296186160.result"

2. Consider the 'System summary' view (see Figure 1, Point 1) regarding the analysis of your project. You can see the design flaws that seem affect your system.
Pick the first three design flaws (in order of severity, see Figure 1, Point 2) that affect your software, and for each flaw complete the following points:
a) Explain the design choices or errors leading to the detected design flaw (4 pts).

All seven design flaws detected are caused by the design choice to make power-ups very customizable on the run, we didn't end up using this, and because of this a lot of variables are present. This would have been different if we had used the decorater design

pattern, because then for example a powerup wouldn't need all the parameters for a drop too, but this was not possible.

It would also have helped if we put X and Y values in pairs instead of single variables.

b) Fix the design flaw or extensively and precisely explain why this detected flaw is not an error and, thus, should not be fixed (10 pts).

We will fix this by making these powerups far less customizable (a feature we didn't use anyway), which should remove a lot of the duplicate variables and make the dropRandomizer less messy. By removing customizable textures for power-ups, horizontal movement, and being able to define a start position (which got set to the ball location anyway and was useless), the Drops now have far less variables and are not Data Clumps anymore.

In the future it would be a good idea to make sure that when using coordinates to default to using a Point instead of making two separate variables.

If your project has less than three design flaws, congratulations! In this case, consider other design flaws (to reach a total of three with the previous ones) that inCode could detect, and explain in detail where each of these design flaws could have probably affected your system and how you managed to avoid it (10 pts per design flaw).

One of the design flaws we purposefully avoided was external duplicating, especially visible in the player. We managed to avoid it by adding a playerScheme to assign which key should be considered rightPressed() etc, instead of creating two players with different keys. This greatly improved our maintainability.

We also avoided sibling duplication in the Drops by extending from one common Drop class for all power-ups, and thus sharing shared behaviour instead of duplicating. And also avoided sibling duplication in all our Visual objects by making the abstract GameObject which handles the actual drawing of objects, from which we extended to not duplicate this code. We managed to avoid this by properly applying Inheritance.

Exercise 3 - Teaming up (15 pts)

1. In this exercise, you decide together with your TA, during the group meeting on Monday, new game features to add to your game. After you decide on the tasks, write a requirements document, which will be evaluated in the same way as for the requirements document of the initial version. Afterwards you must implement the requirements. (11 pts).

2. During the analysis and design phases of this extension use responsibility driven design and UML (push to the repository the single PDF file including all the documents produced) (4 pts).

Requirements Pang!:

RETURN BUTTON:

Group: -1

Jurriaan den Toonder: 4431324
Erik Wiegel: 4476328
Govert de Gans: 4491955
Jaap-Jan van der Steeg: 4456130
Tim Rietveld: 4472926

Functional Requirements:

Must have:

- **Return button must return to mainmenu on click.**

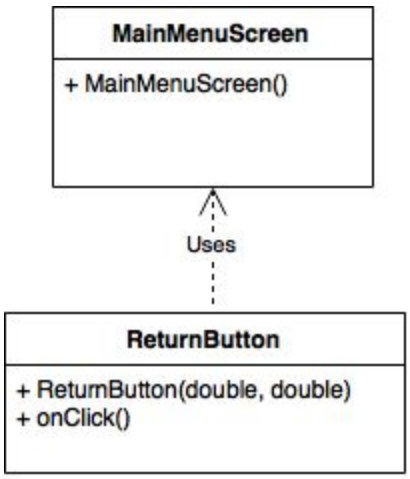
Should have:

- **Return button's effect should trigger when esc is pressed.**

Non-functional Requirements:

- **Return button should have the same graphical style as the rest of the game**
- **The design of the return button should immediately suggest that it is a return button**

UML:



Requirements Pang!:

Gameover-/Winscreen Delay:

Group: -1

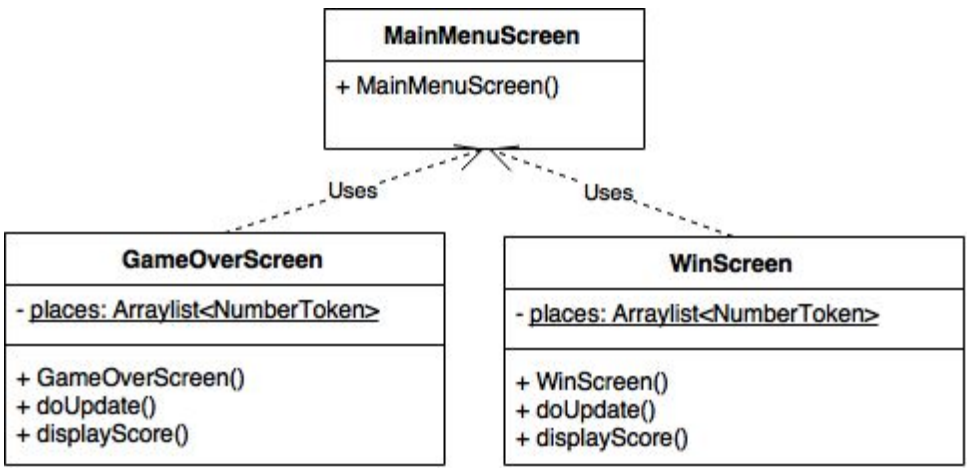
Jurriaan den Toonder: 4431324
Erik Wiegel: 4476328
Govert de Gans: 4491955
Jaap-Jan van der Steeg: 4456130
Tim Rietveld: 4472926

Functional Requirements:

Must have:

- **Gameover and winscreen must not change on every keypress, only on enter**
- **After enter is pressed, the screen must change to the mainmenu**
- **The user must be notified that he/she should press enter after winning/losing a level.**

UML:



Requirements Pang!:

Countdown for Level:

Group: -1

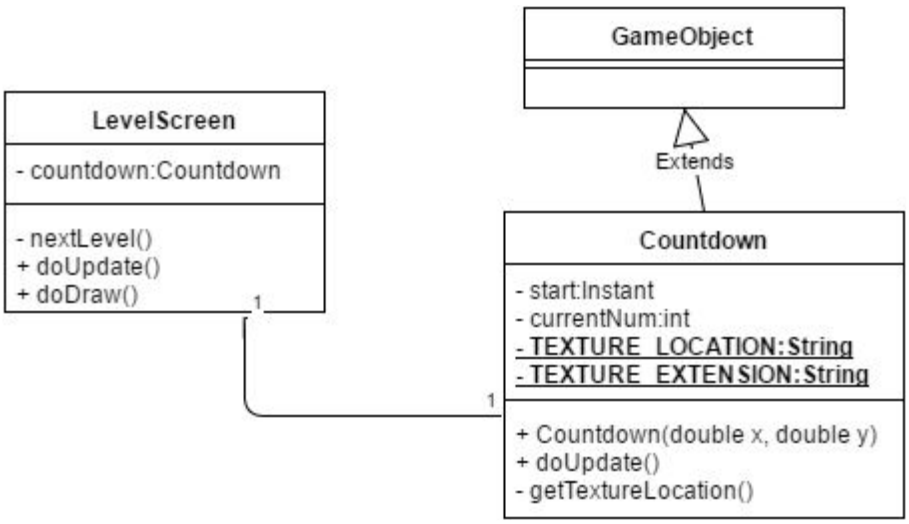
Jurriaan den Toonder: 4431324
Erik Wiegel: 4476328
Govert de Gans: 4491955
Jaap-Jan van der Steeg: 4456130
Tim Rietveld: 4472926

Functional Requirements:

Must have:

- **Levels must not start immediately after one another, but must have a wait period of 3 seconds before the next level actually begins.**
- **The 3 second countdown must be visible to the user.**

UML:



The constructor of LevelScreen creates a new Countdown, thats starts counting as soon as that level is the current screen. LevelScreen deletes the countdown and unpauses the game when the countdown reaches 0.

Requirements Pang!:

More Levels:

Group: -1

Jurriaan den Toonder: 4431324
Erik Wiegel: 4476328
Govert de Gans: 4491955
Jaap-Jan van der Steeg: 4456130
Tim Rietveld: 4472926

Functional Requirements:

Must have:

- Levels must be unique.

Should have:

- Levels should be ordered in a order with (generally) increasing difficulty
- Levels should be possible to win

Could have:

- Levels could have a different background

Won't have:

- There won't be a difference between levels for singleplayer and multiplayer

Non-functional Requirements

- Levels should be fun to play