Ex. 1 Implementing multiplayer.

There are three different modes we have implemented: survivalmode, coopmode and duelmode. The requirements we specified can be found in Appendix A.

Coopmode and duelmode:

CRC

The CRC cards can be found at appendix C.
We came up with a number of changes we thought we would have to make to our code to implement multiplayer. First, some options have to be added to the menu. To achieve this most menu, options and highscore controllers have to be modified. Furthermore, we came up with different game classes for the different multiplayer game-modes. We chose not to implement those because it was highly unnecessary. We now support multiplayer games only by changing some hardcoding in more dynamic coding and moving some stuff to classes it actually belongs. For example the key-codes for moving a player, which should be in Player or a dedicated class and not in level. The only difficulty we had was to get other classes like player to know which game-mode is active. We did this by making the current game-mode public. The UML for this exercise is not changed and can be found at Appendix B.

Survival mode:

CRC

Survival mode is basically single player with the only differences being some handling of events. Instead of counting down the time, survival counts up from 0 and when the balls have all disappeared, instead of winning the level, new balls will spawn.
The difference in handling what to do when all balls have disappeared can be handled by adding an if-statement in the collisionhandler to check if it is dealing with a survival game.
The difference in the handling of time could be handled in two ways.

1. Making survivalLevel a child-class of level and overriding the update class while adding the classes for spawning the balls.
2. Make a private boolean in Level and put if-statements in the update class while adding the classes for spawning the balls.

Option 1 would be inheritance for code reuses, which is bad practise, so we chose for option 2.

Since now new classes were made for this exercise, the UML hasn't changed.

Ex. 2

Name	Cumulative Severity
LevelBuilder	6
Level	3
CollisionResolver	2

The result file of inCode is located at Doc/inCode_before_refactoring.result.

2.1 LevelBuilder is a data class

a. The LevelBuilder class had two main problems following from the InCode report, combined with severity 6. Both problems follow from our design choice to create a LevelBuilder class, to build a new instance of the complex Level class, which is implicitly tightly coupled with Level.
The first problem was that LevelBuilder exposes a lot of data in its public interface via setters. Specifically, this is because LevelBuilder has five setters to set the data LevelBuilder needs to build a Level.
The second problem was that LevelBuilder calls a lot of external accessor methods of Level, which is also understandable because LevelBuilder has to build the Level class entirely.

b. We refactored the LevelBuilder class to be a static class inside Level, and renaming it to “Builder”, so it can be referenced as Level.Builder. This resolves the second problem, as the external accessor methods are now included in the same class and are therefore internal accessors now.
This does not resolve the first problem, however, we believe it is justified to ignore this issue. Firstly, because the builder pattern in Java is adequate to use in this situation: The need to create a complex object, Level. Secondly, because the data that is set is really necessary, and this is the best way to do this. Another way to pass data can be via the constructor, but would be quite ugly and not very flexible. Using setters allows for chaining, which improves code readability and syntax throughout the whole project. Because Level.Builder now exists inside the Level class, we believe it is suitable to ignore the issue of having too much data in its public interface, because the functionality is needed and allows for good syntax. The Level.Builder class now has severity 1.

2.2 Level is a god class

a. The Level class has many functionalities and dependencies. It can be thought of as the main class. Thus, it uses many external classes to control many aspects of the game, and methods can become a bit complex and highly nested.

Level controls movement of objects, animations objects and its own states. This is the reason the class gets so complex and large.

b. In order to fix the god class design flaw, we moved the drawing and moving objects responsibilities to an external ObjectDrawer class. Level is now only responsible for the things it should be responsible, such as handling state changes. The large amount of attributes can however nog be reduced, as there still a few classes dependent on Level that have to be dependent on Level. Also, it has to use a few other classes in order to act as a controller. That is why the god class design flaw is not gone as far as inCode thinks. However, we consider the Level class to have the right amount of responsibility to not be called a god class.

2.3 Method CollisionResolver.playerVersusWall() has feature envy

a. The error was caused by the fact that the method does calculations with getters and setters of the Player and Wall class, to calculate the new position of the Player. The new position is outside of the Wall, so Player stops colliding with Wall, and, more importantly, does not go through Wall.The problem results from a decision to split responsibility over classes. More specifically, we moved the responsibility of resolving (acting on) collisions in the CollisionResolver class. The CollisionResolver now took care of handling this player wall collision.

b. As a solution, we revisited the player wall collision, came up with a better way of handling things and divided the method into some smaller ones. Now, the method(s) no longer call too many outside attributes and the design flaw is resolved.

Appendix A

Requirements Multiplayer

Must have:

- There must be an option to play a multiplayer game in the menu.
- There must be two possible modes to play in multiplayer: Coop mode and duel mode.
- In the coop mode, the players work together to beat the levels and the points are counted together.
- In the duel mode the players try to beat each other. Points are counted separately, as is for the lives.
- When a duel multiplayer game starts, both players have 0 points and 5 lives.
- When a coop multiplayer game starts, the players have 0 points and 10 lives together.
- When a player dies, only lives from him are substracted unless in coop mode.
- When a player hits a ball with a projectile, only that player will receive points unless in coop mode.
- Both players keep their points after a player dies.

Should have:

- Players must be controlled by separate controls.
- When one player is dead but the other is not, the multiplayer game should continue with one player.
- When a certain player picks up a power-up which counts for all players, all players should receive it (like freeze).
- Different graphics for different players.
- Special levels for multiplayer

Could have:

- When the user tries to apply the same controls to both players, an error message should appear.
- An option to set the color of a player.
- Different highscores for a multiplayer game than for a single-player game.

Won’t have:

- Online multiplayer via LAN.

Requirements Survival Mode

Must haves:

- The level should spawn new balls once enough balls have been cleared.
- The player must have only one life.
    - The life powerup must be disabled
- The level must not have a time limit and should theoritically continue forever.
    - The time powerup must be disabled.

Should haves:

- The balls spawned should be random.

Could haves:

- The player could receive point-multipliers the longer it can survive.

Won’t haves:

- The level won't be playable as a multi-player game.

Appendix B: UML

Appendix C: CRC cards

Multiplayer CRC

Requirements specification

HighscoreMenuController	
Superclass: -	
Subclasses: -	
Display the singleplayer highscores	HighscoreController
Display the multiplayer highscores	HighScoreController

MenuController	
Superclass: -	
Subclasses: -	
Start a singleplayer game	Gamecontroller
Start a multiplayer game	MultiPlayerMenuController

MultiPlayerMenuController	
Superclass: -	
Subclasses: -	
Start a multiplayer game in coop mode	GameController
Start a multiplayer game in duel mode	GameController

OptionsController	
Superclass: -	
Subclasses: -	
Set the controls of a player	Player
Set the color of a player	Player

MultiplayerGame	
Superclass: -	
Subclasses: CoopMPGame, DuelMPGame	
Manage common multiplayer functionality	Gamecontroller

CoopMpGame	
Superclass: MultiplayerGame	
Subclasses: -	
Start a COOP mode multiplayer game	Gamecontroller

DuelMPGame	
Superclass: MultiplayerGame	
Subclasses: -	
Start a Duel mode multiplayer game	Gamecontroller

Survival CRC

Requirements specification

HighscoreMenuController	
Superclass: -	
Subclasses: -	
Display the survival mode highscores	HighscoreController

MenuController	
Superclass: -	
Subclasses: -	
Start a survival game	Gamecontroller