Developer Manual

Import the project

1. Open Eclipse.
2. Import the project.

If Eclipse complains about the compiler version, do:

1. Right-click the project in the Package Explorer.
2. Choose Properties from the drop-down list.
3. Go to the Java Compiler tab in the new window.
4. Set Compiler compliance level to 1.6.

How to run the tests

Import the tests project

1. Open Eclipse.
2. Choose to import a new project.
3. Navigate into the tests folder in UltraExtreme.
4. Import the project UltraExtremeTest.

It's possible that you have to change this project's Java compiler compliance level to 1.6 too.

Run the tests

1. Right-click the tests project.
2. Choose Run As > JUnit Test.
3. Choose Android Junit Test Launcher in the new window and press OK.
4. Eclipse should now open the JUnit view and run through all the tests.

Architecture, 7/10 2012

Model

The GameModel keeps most of its game data in containers called -Manager, such as BulletManager, PickupManager and EnemyManager. It also has a Player and an EnemySpawner.

The EnemyManager has a list of Enemies, and those have much in common with the Player class. The biggest difference is that Player is controlled by the actual player and an Enemy is controlled by the game.

A Player has an Itembar which contains Weapons, and a Player also has a PlayerShip which is an Entity that is shown on the screen.

An Enemy has a single Weapon and an EnemyShip, which is an Entity.

All objects that appear in the game extends the class AbstractEntity. These are bullets, ships and items that can be picked up.

The EnemySpawner has WaveLists which describes when waves of enemies should start.
A Wave has a number of enemies, and the enemies in the wave fly in a pattern, such as a V pattern or a horizontal line.

The PickupManager contains the items that are dropped from some of the enemy ships and it keeps them until the player's ship picks them up.

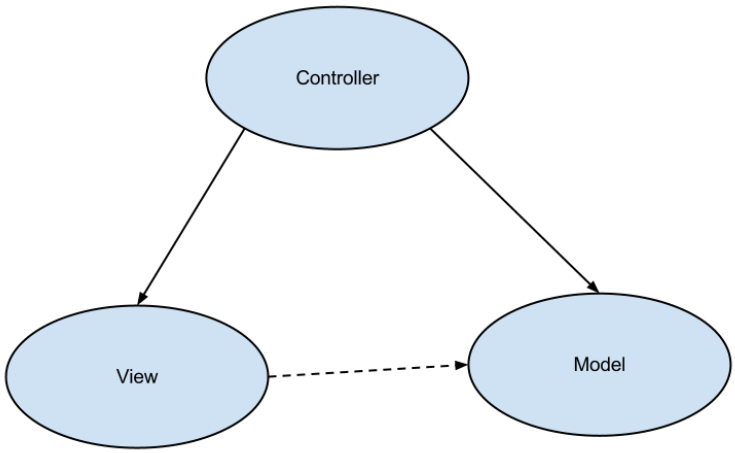
Controller & view

MainActivity extends AndEngine's SimpleBaseGameActivity and it has a number of scenes, which extends AndEngine's Scene class. The scenes we currently have are MainMenuScene, GameScene and GameOverScene. Each scene also has a controller.

The controller we have worked the most on so far is the GameController. It manages the GameScene, it also has a reference to the GameModel, and to regularly update the model and the scene it uses a class called GameLoop which implements AndEngine's interface IUpdateHandler.

Package structure

The project uses a strict MVC model, where the model is completely oblivious to its surroundings.
The controller makes changes to the model by giving it an instance of a standardised input-class. The controller also handles the view and hands it listeners.
The View listens to the model and makes changes to itself accordingly.


Controller
View
Model