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 test 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

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 contains all Enemies that are present in the game. The Enemies 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 her Weapons. The Itembar has a maximum limit of Weapons it can contain. The Weapons also work as “extra lives”. If the ship gets hit, the player loses some Weapons instead of losing an actual life. Apart from the ItemBar 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. There are three different implemented enemies called BasicEnemy, HitAndRunEnemy and ParabolaEnemy. The main difference between enemies are their different movement and shooting patterns.

All objects that appear in the game extends the class AbstractEntity. These are bullets, ships and items that can be picked up. This makes it easier to check for collisions between them.

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 or until they disappear from screen.

WeaponFactory

To avoid having to pass the BulletManager to all classes that need to be able create Enemies or Weapons we have a WeaponFactory. It’s a singleton that needs to be initialized with a reference to a BulletManager before it can be used. Then the WeaponFactory can be used to easily create new Weapons, which you can give to Enemies.

Spawning enemies

To spawn new enemies we have the class EnemySpawner. It 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 WaveList we have implemented is called RandomWaveList. It spawn random waves indefinitely.

To get the enemies that EnemySpawner spawn, you have to add EnemyManager as a listener to it. Then the only thing that is needed is to call EnemySpawner’s update method in each update of the GameModel.

Difficulty

It’s possible to set the game’s difficulty through the GameModel’s constructor. The chosen Difficulty adjusts how fast the time between spawning waves will decrease, making the game harder faster with a higher Difficulty level.

Controller & view

All different views in the game, such as the main menu, options view, high score view and the game view, all extends AndEngine's Scene class. All scenes are controlled by a controller. The controllers are maintained by MainActivity.

MainActivity is the main component of our application, it extends AndEngine's SimpleBaseGameActivity, and it's where the thread starts in.

Only one Controller and Scene can be active at a given time, and MainActivity handles switching between the views.

The controller that has gotten the most attention is GameController. It's here that the actual game is played out. It manages the GameScene, it 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.

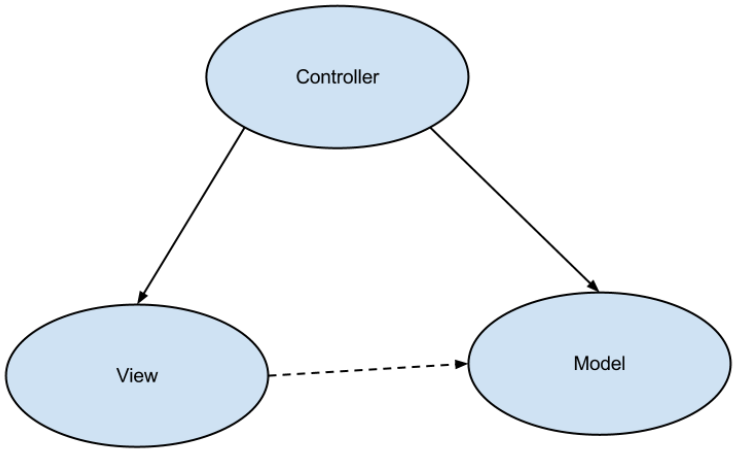
Databases

To store the high scores list and which difficulty setting the player has chosen, we use a couple of local SQLite databases.

When you die in a game, you are sent to the GameOverScene. Here you are able to type a name you want to associate with your score, and when you press on the Save button the score is stored in the local database. The scores can then be viewed on the high scores view of the application.

Package structure

The project uses a strict MVC model, where the model is completely oblivious to its surroundings.
The GameController 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.



Build procedure

To build the project we generally do the following steps:

1. Have the project imported in Eclipse.
2. Use Eclipse's compile and run feature to build the project (and install in on a device).
3. Copy the .apk file from the bin folder inside the UltraExtreme project directory.

An alternate way to build the project can be done through ant:

1. First you have to create a machine-specific local.properties file containing a path to where the android sdk can be found. That can be done by running **android update project -p** inside the UltraExtreme project directory.

   An example of a local.properties file is the following:

```
# This file is automatically generated by Android Tools.
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file must *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
# location of the SDK. This is only used by Ant
# For customization when using a Version Control System, please read
the
```

```
# header note.
sdk.dir=/home/matachi/android-sdk-linux
```

2. Then you can build an .apk file with the following command: **ant -f build.xml release**
3. Finally you can find a generated .apk file in the bin folder.

Release procedure

When releasing a new version we do the following things:

1. Compile and build an .apk file of the project, which we then put in the doc folder. The apk package can be downloaded here: https://github.com/DAT255-group20/UltraExtreme/blob/develop/doc/UltraExtreme.apk
2. Update the Release Notes document in the doc folder with a list of new features found in the new version and also with known bugs.
3. Merge the develop branch into master.
4. Tag the merge commit it with the version number of the release. The tags can be found here: https://github.com/DAT255-group20/UltraExtreme/tags