World of Everbies

1. Introduction:
    1.1 What is World of Everbies:
    1.2 General characteristics of application
2. Code Structure
    2.1 Class description
    2.2 MVC
    2.3 Classes with Dependencies
    2.4 Source Code
3. Test environment
    3.1 Ugly Tests
    3.2 Manual tests
4. Known bugs and limitations
5. Test report
    5.1 JUnit
    5.2 Manual tests

1. Introduction:

1.1 What is World of Everbies:

The reason we decided to take on this project was the opportunity to create our very own version of the classic game Tamagotchi. But we didn't want to create an exact replica, we simply wanted to use the concept and add our own ideas and features. So the main idea of the game became that of having a pet and raising it through training, fighting and working. Then came the idea to add a level system to the game, which led to that your Everbie, your pet, gains levels through the different activities in the game and also develop certain attributes depending on the level and the way it has been treated.

1.2 General characteristics of application

The application utilises a very basic GUI with heavy focus on the "log" function, displaying the most recent events that have occurred regarding your Everbie.

2. Code Structure

2.1 Class description

Main package:
- Everbie; the main class in the “model”-part, describes an Everbie and follows the Singleton pattern.
- MainActivity; this is the main controller that has control of most of the xml classes.
- Log; log is the class responsible for generating Strings to be displayed in the textarea in the activity_main class by taking in a object of the type that is going to add a new String to the textarea. Follows the Singleton Pattern.
- Database; the class responsible for saving and loading. Uses a SQLight database.
- NewEverbieActivity; the controller class for creating a new Everbie.
- Use; a utility class for activating food, work, training and the likes.
- Combat; the class that holds the logic for fights.
- Occupation; a thread class that starts when an object of type Occupationable is commenced, and takes care of what should happen when the Occupationable is finished.
- StartscreenActivity; the controller for the start screen that handles the initiation of the other activities.

Util package:
- Occupationable; an interface implemented by all events that will take time.
- SimpleGestureFilter; a class copied from android-journey.blogspot.se to enable Swipe.

- Race package: This package contains information on how the different races of Everbies should be initialised.

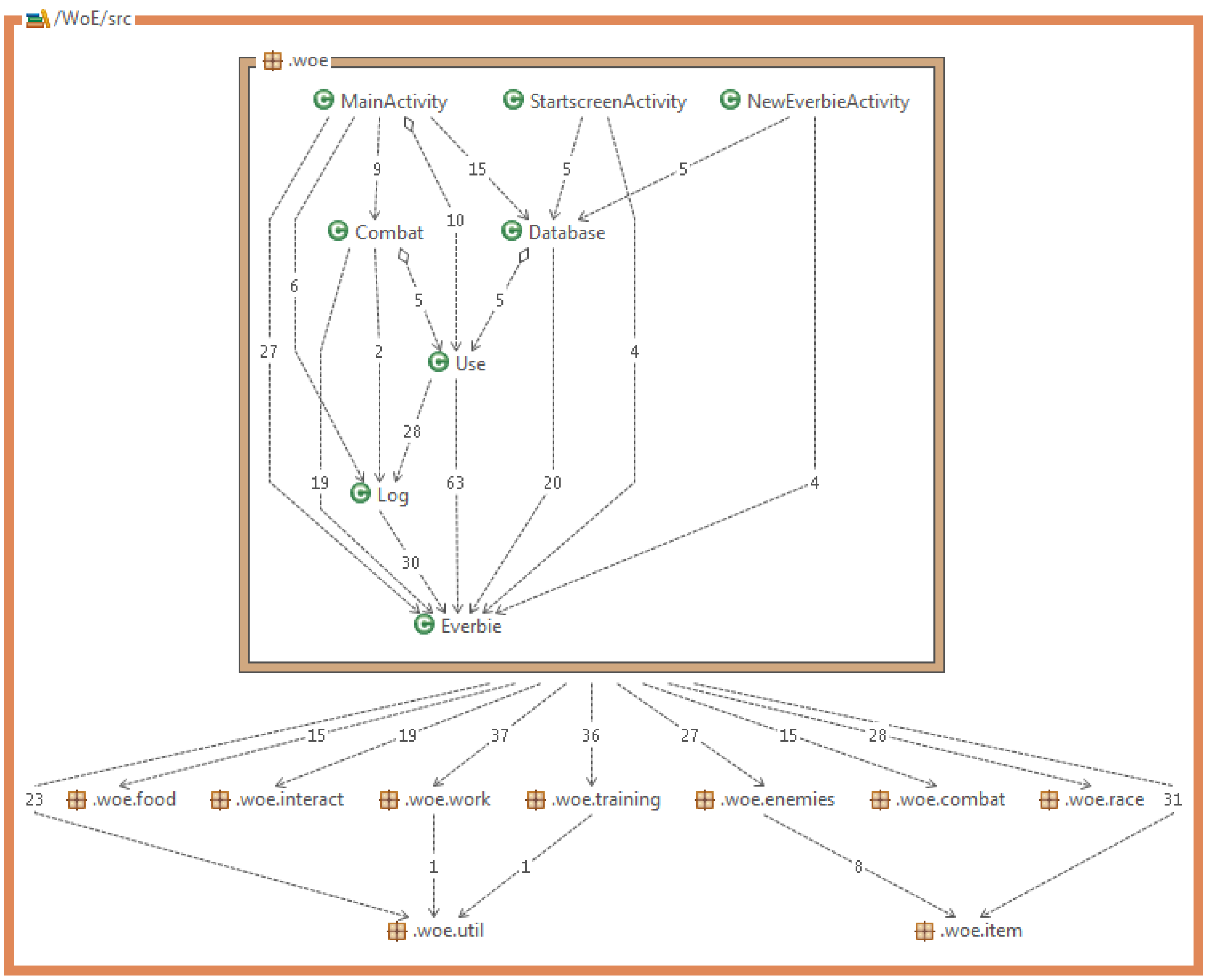
- Combat package: This contains the different approaches you can have in combat. These styles will slightly affect how the battle will turn out.

- enemies, food, interaction, item, training and work: All these packages contain an abstract class plus classes that extend this class and are shown in the main menu.

2.2 MVC

All classes that ends with ...Activity.java are considered “controllers” if it would be the classical MVC-concept. .xml files are “views” or part of a view, and the plain .java classes are mostly “model” classes. The model includes some abstract classes and a few Singletons.

2.3 Classes with Dependencies



The packages food, interact, work, training, enemies, combat, race and item are all structured with an abstract class on top and a number of classes extending it. The package util contains loose utility classes without connections.

2.4 Source Code

Our source code is available from github at user: Mackro, repository: WoE.

3. Test environment

3.1 Ugly Tests

Due to the inability of JUnit Android to work, at all, we have written “ugly tests” that are located in the “Tests”-folder next to src. They have similar structure to JUnit though instead of Asserting booleans, we just use if-statements to print correctly in the log.

3.2 Manual tests

We will perform manual tests for the GUI and look-and-feel of the application, this is also how we look for bugs in the application.

4. Known bugs and limitations

Bugs:
1. Log doesn't update automatically when work or training is done.
2. Toxicity doesn't decrease over time only melon lowers it.

Limitations:
1. In the "Feed"-menu, the effects of the different foods are not shown.

5. Test report

5.1 JUnit

No JUnit tests will work, though they appear to be correctly coded. Neither our coach nor the teacher knows why they aren't accepted by Eclipse.

5.2 Manual tests

We've written rough classes that test the model functionality in an extreme loss of JUnit tests. They are located in a source folder next to src called Tests.
Further we have texts that describe how to test the Gui and other functionalities, not testable by code alone. These texts are located in the same folder.