

# 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 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. Though 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

All classes are currently part of the same package.

the classes that exists at the moment are:

- Everbie; the main class in the "model"-part,describes an Everbie. Follows the Singleton pattern.
- MainActivity; this is the main controller that has control of the most xml classes.
- Food; the abstract class that all food-related items extends.
- 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.
- NewEverbieActivity; the controller class for creating a new Everbie.

-Use; a utility class for activating food, work, workouts and the likes.

## 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 woe.work, .training, .item and .interact are internally structured exactly as woe.food is.

## 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**

1. Unable to save and load correctly.
2. Unable to scroll in StatsView.

## **5. Test report**

### **5.1 JUnit**

No JUnit tests will work, though they appear to be correctly coded and neither our

### **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 sourcefolder next to src called Tests.