TouchDeck: Developer Manual

Dependencies

- Java 6 SE development environment
- Android SDK
- A (virtual) Android device
- Junit
- Robotium

Android SDK targets

- Minimum SDK: 15
- Target SDK: 15

Building and installing

In the project root directory there is an ant build xml-file that can be used to build the project without using eclipse. By default, the output directory is the bin folder in the project root. To build the project, open a command prompt, navigate to the project root and run the following command:

```
$ ant clean debug
```

To build and install the TouchDeck.apk file on a connected Android device, run the following command:

```
$ ant clean debug install
```

To uninstall the application from the device:

```
$ ant uninstall
```

To view all ant targets, run:

```
$ ant -p
```

Developing

Using git

We are using git as revision control and the repo is located at:

git://github.com/sebiva/TouchDeck.git

We have decided that we will only be using git from the command-line and not from git inside eclipse. This was in order to maintain more control over the repo.

Branching

When developing a new feature we always create a new separate branch to develop in. The new branch should always be branched out from the "dev" branch and only in exceptional cases, which are discussed with and accepted by the configuration manager, from other branches.

Conditions for merging back a feature to "dev" see the section "Definition of done".

The master branch is only merged to when releasing a new version.

Setup and using eclipse

We are using Eclipse as IDE. The complete and updated installation guide is located in the wiki on github:

https://github.com/sebiva/TouchDeck/wiki/Setup-eclipse

Writing code

We try to write our code as much as possible according to the android "Code Style Guidelines for Contributors". When this is not possible we try to write the code as clear and readable as possible so future programmers, such as you, will understand it.

After we have finished a feature we always try to run three static analysis tools:

- FindBugs(http://findbugs.sourceforge.net)
- CheckStyle(http://checkstyle.sourceforge.net)
- PMD(http://pmd.sourceforge.net)

A future goal is that when we have a good build server this will run automatically every time the code is built.

We also have a wiki page on github for writing code:

https://github.com/sebiva/TouchDeck/wiki/Writing-code

Issues

Bugs and other issues is tracked in github.

Definition of Done

In the "docs" folder in project root there is a document named "DefinitionOfDone.doc" that describes exactly what needs to be completed before a feature is considered done. This should apply to every feature we develop. This definition is not meant to be static but rather evolve dynamically alongside the project.

Release procedure

The following steps should always be done for every major release of TouchDeck.

Merging
Merge dev into master and tag the commit with the corresponding release version.

Building TouchDeck
You are free to build the .apk file either with Eclipse or with ant by running:

```
$ ant clean release
```

Create a new folder in the distribution folder
There is a folder called “dist” which is located in the project root. In this distribution folder we place all major releases in a separate subfolder.

Write a change log
For every major release there should always be a change log provided. The change log should be in PDF format and contain, at least, the following information:

- Introduced features
- Changes since the previous release
- Bug fixes since the previous release
- Known bugs and limitations

A good practice is to use an old one as template

Test report
A test report with the result from a complete run of the test suite should also be provided in the release folder.

Testing

There is a separate test project for our automatic tests. It’s located in the “TouchDeckTest” folder in the project root and it contains both unit tests and GUI tests.

Unit tests
Where it’s applicable to the code we write JUnit test cases. The tests should test both positive cases and negative ones.

GUI tests
We are using Robotium (http://code.google.com/p/robotium/) as test framework.

Currently there are a few GUI tests that runs but we don’t have any stable platform to run it on (for instance, the tests won't run in sequence) so for now we don’t develop any new GUI tests. To maximize feature development we also decided to write GUI test as manual tests and don’t spend a lot of time on troubles from developing automated GUI tests. Read more about future plans for automatic GUI tests under “Future development” section.

Manual tests
In the docs folder we have a document named “TestCases.doc”. This is our main document for manual tests and should always be updated with the latest tests. This type of test is more of end-user type and should make sure that all features works but also that a good user experience is kept, such as the speed and quality.

Architecture

The fundamental architecture
TouchDeck is divided into three major packages; game, network and misc.

Game
The game package has two major sub-packages; server and client. The names are quite self-explanatory and contain classes accordingly. The main class in the server packages is “GameController” and for the client it’s “GuiController”.

Network
In this package all classes related to network and synchronisation.

Misc
Here we have other types of classes that are used application-wide, such as enums and constants.

Network

Every distribution of the app has the possibility to be both host and client. An important thing is that the client always connects to the host by network; this means that when you are both host and client you are playing over network on ip-address 127.0.0.1. This is done by design and results in few to no special cases at all for local play.

The network part is quite simple. It doesn't use well known network protocols, instead we send serializable objects.

The client part sends an object containing which action a user has performed and the necessary card and pile information to perform it.

When a user has performed an action the server pushes out the new game state to all connected devices. The information that is pushed contains all information about the game and not only deltas. The down side with this is that more data is sent over the network but the amount of data is still very small and the biggest advantage of this is that all devices are always in sync. It also enables users to leave and enter a game as many times as and whenever, they wish.

Activities

Currently we only have 3 activities

- Start screen – This is where you choose if you want to create or join a game
- Table view – An overview over all piles
- Pile view - Displays all cards in a pile.

Past design decisions

All large design decisions that have been taken are documented in the document "DesignDecisions.pdf" in the "docs/reviewcandidates" folder.

Future development

Future goals for the app
Implement the option to have game templates in order to automatically set up piles and deal cards according to well-known

games. A further goal would be to let users create game templates by themselves.

We want to keep adding features and optimize the GUI to improve the user experience and make the game smoother and faster to use.

Future goals for the testing

Setup a stable test environment for Robotium and convert all manual tests that are possible to convert. We want to be able to check the test coverage and have all tests are run automatically on every commit.