# System design document of ‘Tendu’

*Version:* *0.5*
*Date:* *October 27, 2013*

*Authors:* *Hampus Forsvall, Markus Norén, Roy Nard, Johanna Hartman, John Petersson, Christian Meijner, Ludwig Lindberg.*



## Quick Start

Git Repository: https://github.com/tortal/IT2-LP1-Tendu.git
https://github.com/tortal/IT2-LP1-Tendu

To start Tendu, install the binary file (Tendu-android.apk) from the root folder of the repository.

Tendu is meant to be played four players over Bluetooth. Start the application on all devices. Press “Host game” on one and “Join Game” on the rest of the devices.
(There is also an experimental WiFi implementation: Tendu-android-wifi.apk)

The project structure is built around that of any Libgdx project and is divided by platform-specific code.

Folders:

- **Tendu-android**: Android-specific code.
- **Tendu-desktop**: Skeleton folder.
- **Tendu**: Main project folder.
- **Doc**: Project documentation.

# 1 Introduction

**The purpose of this document is to describe the outline of Tendu’s system design, programming architecture, software dependencies and other technical details that might be relevant to anyone developing Tendu.**

**Tendu** is a fast-paced, time-attack, real-time multiplayer game primarily developed for Android hand-held devices.
Tendu marks the re-introduction of gaming in local local-area-network by enforcing users to be in physical proximity in order to play.

Tendu is developed in the Java programming language and constitutes the namespace ‘*it.chalmers.tendu’*.

## 1.1 Design Goals

- To create a local-area-network multiplayer game for 2-4 players that does not require internet-connectivity.
- To make use of simple but effective close proximity RFCOMM such as Bluetooth as a primary implementation.
- With libgdx library as dependency, cover graphical, audio-visual feedback and user input. Even though the current implementation is tightly coupled to their dependencies, there is no restrictions on partly swapping one current implementation with others, as long as they will provide a functional implementation of the specified interfaces.
- For Android devices with Android OS version 4.1 and above additionally support WiFi connectivity. WiFi support however is not yet merged into the master branch.
- To avoid platform-specific restrictions.

Tendu is developed with the philosophy of platform-independency, modularity of OOP concepts, compromised with native application user-end experience.

### 1.2 Nomenclature

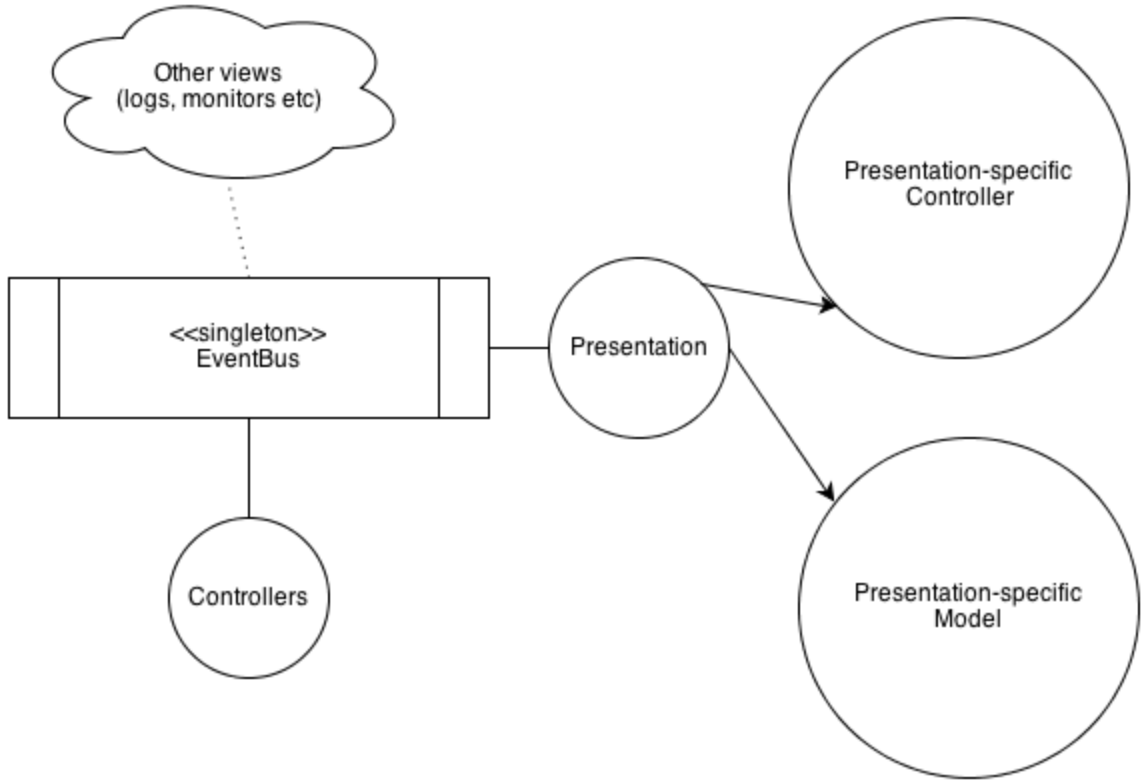
- **Libgdx:** Cross-platform game development framework with a high-level API for OpenGL and relevant technologies. (http://github.com/libgdx/libgdx/)
- **Kryo:** “Fast, efficient” Java object serialization. (http://code.google.com/p/kryo/)
- **KryoNet:** TCP/UDP client/server library for Java built on Kryo.

# 2 System Design

Tendu is in its core a framework for “mini games”. These mini-games populate a game session in which players participate, plays through a set of mini games, then finishes with some quantifiable result.
There are Java classes representing these cases:

- *MiniGame*, an abstract class to be extended by any game model.
- *GameSession*, the active model of a game session.
- *GameResult*, a passive model representing the result of a game session.

Developing in the Libgdx framework restricts the need for structural abstraction in the sense that the developer may use the libraries’ interfaces and supertypes in favor of any self-manufactured solution (Same actually applies to the Android core framework). Adding abstraction over the already provided layer is cumbersome however also conceptually correct. This mean that any protocol must be replaced by an independent one - any given OOP structure must be replaced by our own - and ultimately, any dependency should be replaceable.



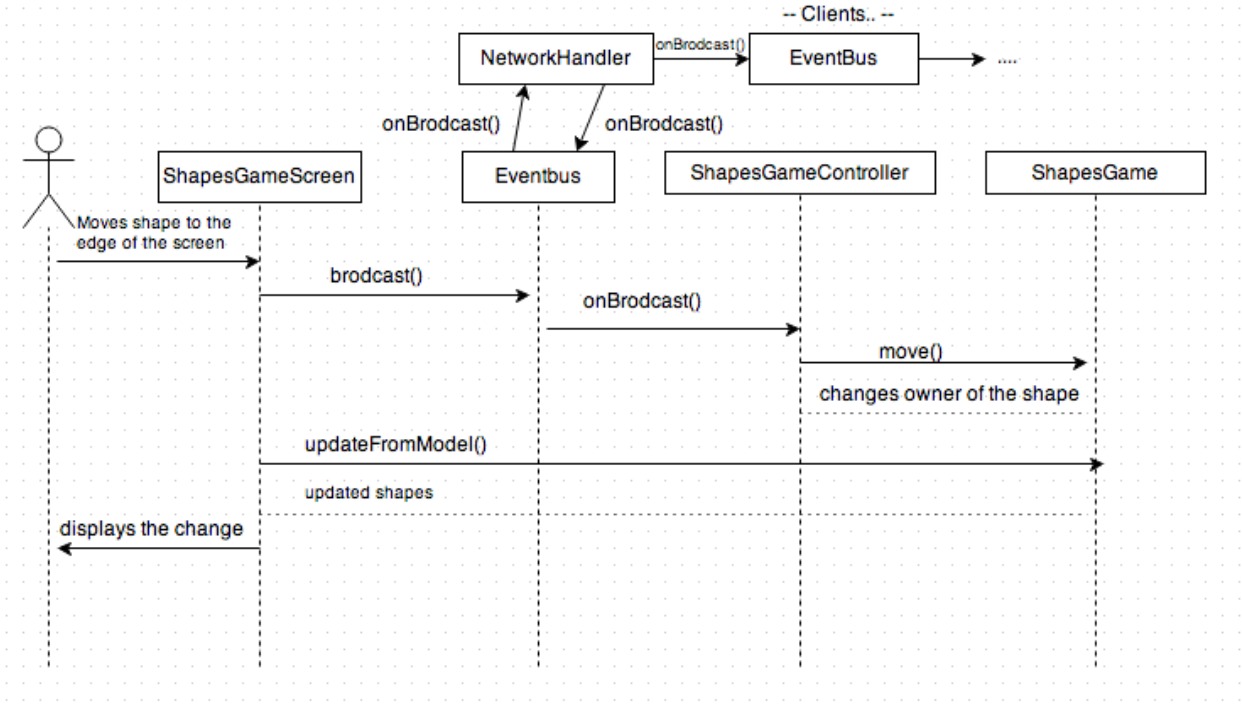

Since the application is designed to run concurrently and on multiple threads, an “event bus” or “event manager” pattern is used, which to and from model, views and controllers may inter-operate and exchange data (internally on the device and externally through a network). Any class that implements the *EventBusListener* interface and are registered may receive *broadcasts*, and since EventBus is a Java Enum singleton, it can be accessed from all objects and threads without delegation. The *EventBus*, and its related classes, represents the core communication framework in Tendu. Controllers may broadcast *EventMessages*, and depending on situation, any listener may react to the invocation.

The application uses a modified Model-View-Presenter pattern and is fundamentally built in a similar way of how a framework would be structured. The interface *Screen* represents the presentation-layer and handles both audio-visual feedback as well as local controller mechanisms. *Screen* objects may be bundled with a Model and an additional Controller. For instance, *LobbyScreen* is the presentation-layer of the connection lobby when connecting devices. The *Screen* handles any graphical representation and any local controller events, *LobbyController* handles the network events and communicates these through the event bus and *LobbyModel* is the passive model representation of the lobby state.

The main entry class of Tendu (*it.chalmers.tendu.Tendu*) implements the libgdx-specific interface *ApplicationListener*. This is by nature of libgdx and is meant to abstract any platform-specific code from the cross-platform code. Tendu is a local multiplayer game but does

not require a stand-alone server. Instead, a device participating in a game of Tendu will act as a “Servent”, which means that it is both server for other clients as well as being a client to itself.

Tendu 0.5 has two mini-games implemented; “Shape Game” and “Number Game”.
Number Game is implemented as a passive model while Shape game is active.
Shape Game is a puzzle game where players starts with a set of geometric shapes and a set of slots to place these shapes. Every player is not guaranteed to have all shapes to fill their own slots - and must therefore send and receive shapes to complete their slot sequence. Sending shapes is done by moving the respective shape to the edge of the screen:




When a player moves a shape to send it to another player the Screen-object broadcasts the event to the event bus. The ShapeGameController-object then receives the broadcast and depending on if the player’s device is acting as server or not, either updates its own model and pushes the change to other devices (server), or requests that the model update is done (client). The model state of the device acting as server is considered authoritative; when the server receives a request it first validates if the update was legal, before pushing it to all clients.

## 2.1 Dependencies

**Libgdx**: Cross-platform java game development framework.

Motivation: The Android Core Framework does not provide a high-level API for OpenGL ES. Libgdx also offers high-level APIs for audio playback and physical user input. Moreover, Libgdx is cross-platform and simplifies any future implementation of other platforms.

**Kryo/KryoNet**: Networking framework and serialization.

Motivation: The Kryo/Kryonet library provide high-level API for TCP communication - eliminating the need of manually writing low-level parsers and serializers. Kryo is a speedy automatic serializer and deserializer that uses Java's Reflection API.
.

## 2.2 Event Handling/Event Message Protocol

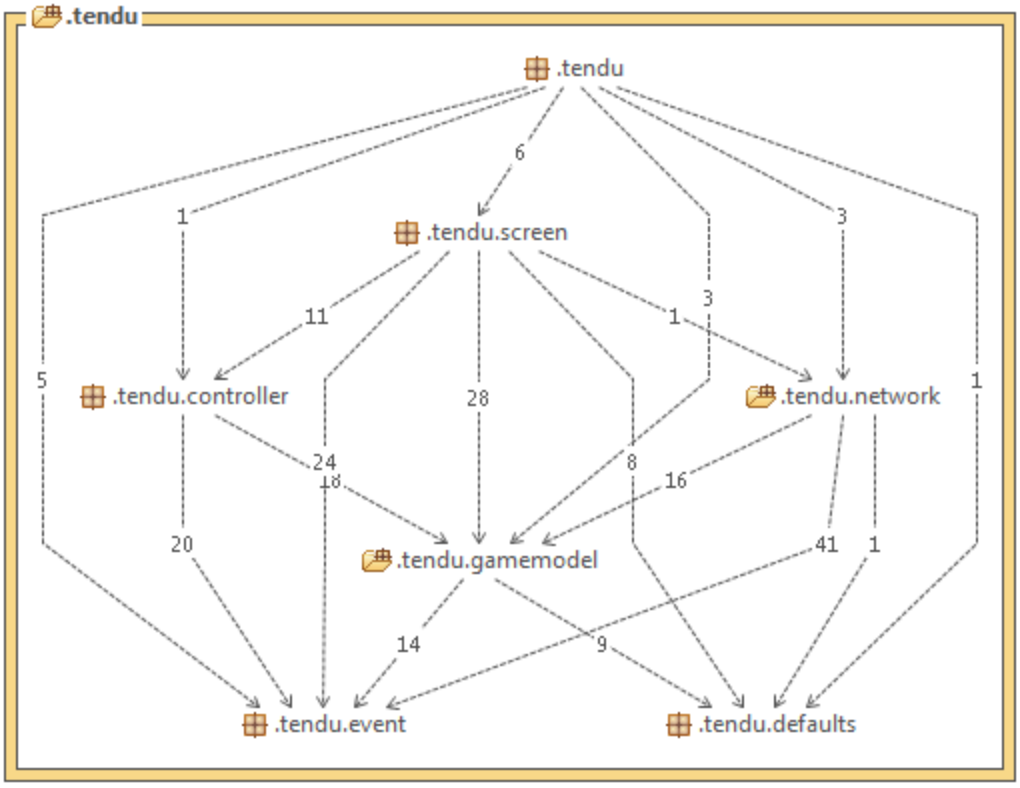
The EventBus protocol defines the core workings of Tendu and also covers its network protocol. *EventMessages*, when sent over the network, are serialized by reflection with the Kryo library. Models are inserted into messages and pushed to clients by the device acting as server when starting a game or game session. While playing, incremental updates are made to clients by sending deltas rather than pushing the whole model. Conceptually, incremental updates are more performant, but no profiling specific to this method in Tendu has been made.

## 2.3 Resources

Tendu has audio and fonts as only resources. These are placed in the Android-project's "assets" folder. All other graphic is described programmatically.

# 3 Software decomposition

## 3.1 Layering and Dependency analysis




## 3.2 Concurrency issues

The OpenGL context runs on one single thread - this is also a constraint of the Android Core Framework. Network controllers (such as a Bluetooth connection) are on separate threads; one thread for every connection. By date, there are no known concurrency issues.

## 3.3 Persistent data management

Tendu does not implement any data persistence. However, the passive POJO *GameResult* is intended to be serialized manually (e.g. to JSON) in future versions of Tendu and is constructed in such way that it eventually may be persisted either locally or remotely to a Server.

## 3.4 Access control and security

Running the application in *Dalvik-VM* on Android OS gives it the least access privilege possible. However, Tendu does not implement a cynical network architecture and therefore trusts other devices' code to be non-malicious. By date, Tendu does not validate connected devices and is prone to intended corruption and hacking.

## 3.5 Boundary conditions

The application will start, initialize, work and stop as expected by a typical user on the Android device. However, a bug in libgdx will incorrectly render the screen if the connection breaks during gameplay - lock the device screen and unlock to re-render).
Tendu will request the user to enable Bluetooth at startup. Tendu is not intended for single-player use, version 0.5 however will not enforce the user to connect in order to start the game. The game models are designed to dynamically support 2-4 players.