Developer manual

Course: DAT255 HT-13

Group 10

Getting started

What you need:

- Java 7 (JDK)
- Android developer tools
- Android SDK
    - Minimum SDK: 5
    - Target SDK: 17
- An Android device or emulator

How to:

- Run:
    - git clone https://github.com/larsson152/dat255
- Build and run the application

Build process

Open Android Developer Tools and import *projectWood*, *projectWood-android* and *projectWood-server* from the Git repository.

- Run *projectWood-android* as an android application to build, install and start the game client on your device/emulator.
- To run the server, right-click on *projectWood-server* and run it as a Java application. Remember to change the IP-address in the client to run your own server, more information below.

Change IP address in client application

When you want to run your own high score server you have to change the IP address that the client will connect to. Go to GameClient.java and locate the connect-method. Change the string to the IP address of your server.

External dependencies

Libgdx - http://libgdx.badlogicgames.com/

Libgdx is a development framework with an API that provides many useful classes and methods. Libgdx enables the use of OpenGL 1.x and 2.0 and helps with graphic rendering through many built in features.

Kryonet - http://code.google.com/p/kryonet/

Kryonet is a library that provides an API for TCP/UDP client/server networking. It uses the Kryo serialization library for quick and easy transfer across the net. One of the most outstanding features of Kryonet is its speed and efficiency, making it very suitable for games.

Universal Tween Engine - http://code.google.com/p/java-universal-tween-engine/

Universal Tween Engine lets us interpolate any attribute of any object in our OpenGL-based project. We use this to animate some of our screens, making them fade in and out.

Major Components

LevelController:

This class is a large controller class that gets information from the Level and Player classes. It checks whether a level has been completed, the game has been paused or if a game over has occurred. It also checks how the player will interact with different kinds of blocks on the level, such as movable blocks or dangerous blocks or if he can move to them. The class also processes the input of the user so the player moves accordingly.

SoundHandler:

This class is used for the sounds in the game. It loads sounds from files for the class to use. The playing of a certain sound is used with a respective method.

LevelRenderer:

This class represents the rendering of a level in the game. It gets information from the Level class to know what kind of block to put where on the screen. The class also loads the textures and draws the player.

Block:

This class represents a block in the game. A level is made of several different kinds of blocks. It can have a variety of different attributes. Such as if its slippery or not, hazardous, solid or movable.

Player:

This class represents the player and his attributes. Such as speed, position, state and the number of keys he has.

Level:

This model class represents a level in the game. A level contains a player and blocks on different layers. A level is read from a file to help identify the locations of different kinds of blocks and the player starting location. For example, blocks that the player collides with are placed upon the collision layer and walkable blocks are placed upon the ground layer.

GameScreen:

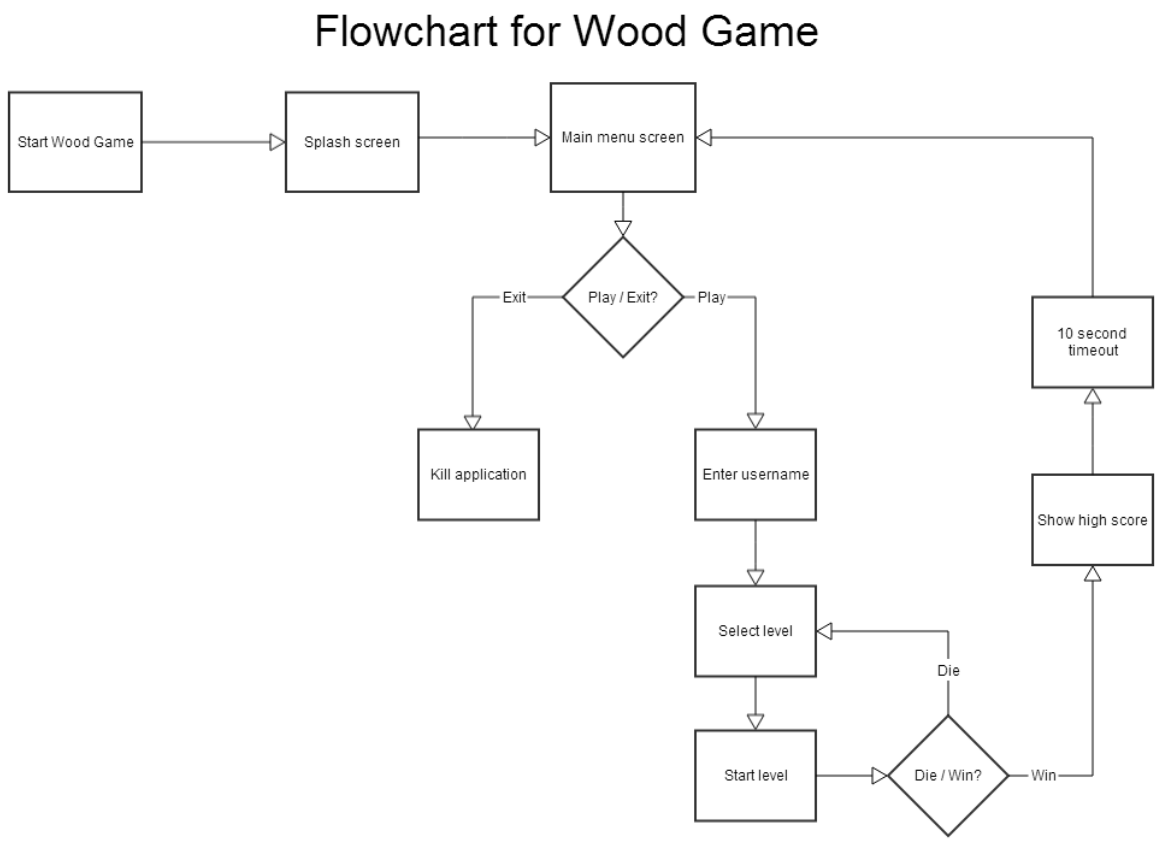
This libgdx class represents a game screen. It uses a LevelRenderer to draw the level on the screen and also sets up the directional pad and score display.

GameClient:

This class handles connections with a server and sends a HighScore object (a small serializable class, used for the score in the game) to the server so the high score gets properly added to the high score screen. It uses the methods of a class named NetworkListener to handle the connections.

Flowchart

The flowchart describes the general flow of the application. The flowchart have been worked out with user simplicity in mind allowing the user, with minimal number of steps, to play any level of the game.



Picture 1 - The Wood game flowchart

Protocol

The Kryonet API is used to set up a connection between the Android client and the server. Kryonet allows any serializable class to be sent over the network and today we have only two classes that will be exchanged, the HighScore class and an ArrayList<HighScore>:

- **Package**
  com.dat255.Wood.model.HighScore
  Sent to
  Server (www.larsson152.com)
  Response
  If "isGetter" boolean is true**: None**
  If "isGetter" boolean is false: **ArrayList<HighScore>**

Only the HighScore class will be sent by the client to the server. The server returns an ArrayList of all the highscores if the boolean "isGetter" in the HighScore object is set to true, indicating that the client wants the highscore list.