Project Report

Mobile Computing - 2013/04

Course: EMDC

Campus: Alameda

Group: 2

Name: **Bogdan Suvar** Number: 79502 E-mail: bogdan.suvar@tecnico.ulisboa.pt

Name: **David Daharewa Gureya** Number: 79533 E-mail: david.gureya@tecnico.ulisboa.pt

Name: **Srijeyanthan Kuganesan** Number: 79531 E-mail: kuganesan.srijeyanthan@tecnico.ulisboa.pt

1. Achievements

The following table shows the features of project specification that were implemented. For each feature we indicate its implementation state.

TABLE 1: PROJECT SPECIFICATION FEATURES

Feature	**Implemented (Fully / Partially / Not implemented)?**
Game scene	Fully Implemented
Movement and life cycle of players	Fully Implemented
Bomb drop off and explosion	Fully Implemented
Movement and life cycle of robots	Fully Implemented
Collision detection	Fully Implemented
Score and game duration	Fully Implemented
Pausing / resuming the game	Fully Implemented
Handling of relevant activity lifecycle events (e.g., pressing home button)	Fully Implemented
Level selection	Fully Implemented
Multiplayer support	Fully Implemented
Clients leaving / joining the game	Fully Implemented
Server hand-over	Fully Implemented
Group merging (or group spitting)	Fully Implemented

2. Specification

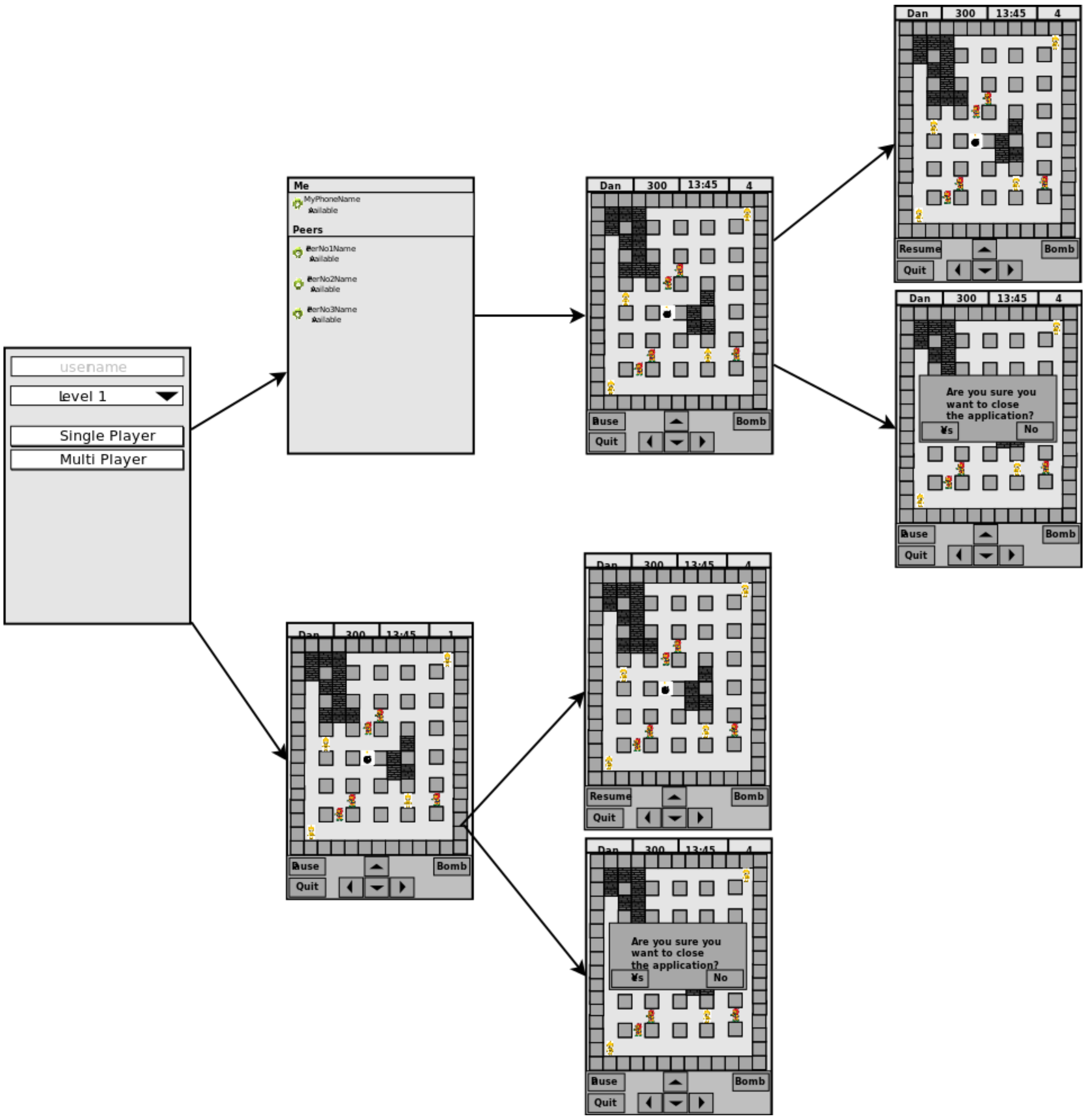

Figure 1: The Activity wireframe of the Bomberman Game

The user enters the preferred nickname (in both, standalone and multi-player versions), selects a difficulty level (1 – the easiest, 3 the hardest) and clicks on either “Single Player” or “Multi Player” button to start the game.

Once the game is started, depending on the game-mode selected, the next view is either the list of peers available for Multi Player game or the grid display populated with obstacles, walls, robots and the player itself for the Single Player game.

Pressing **Quit** has the same effect on both single and multi-player modes, pressing **Pause** results in different behavior on two player modes:

- In the *Multi-Player* mode, if a player pauses its presence will appears as an (immutable) wall to the other players. Thus bombs and robots won't affect the player's state. When the user resumes, its coordinates and life are preserved.

- In the *Single Player* mode, pausing game will preserve the state of the entire grid (robots, player, timers, etc).

The following scenarios will lead to the game termination: When the game duration expires and the player kills all the enemies (opponents and robots).

Quitting in the multi-player game will result in the (quitting) player being removed from the game state. Dying in the multi-player mode, will display the players' points earned and leave the player watching the game of the other players.

3. Design

3.1 Multi-player Centralized

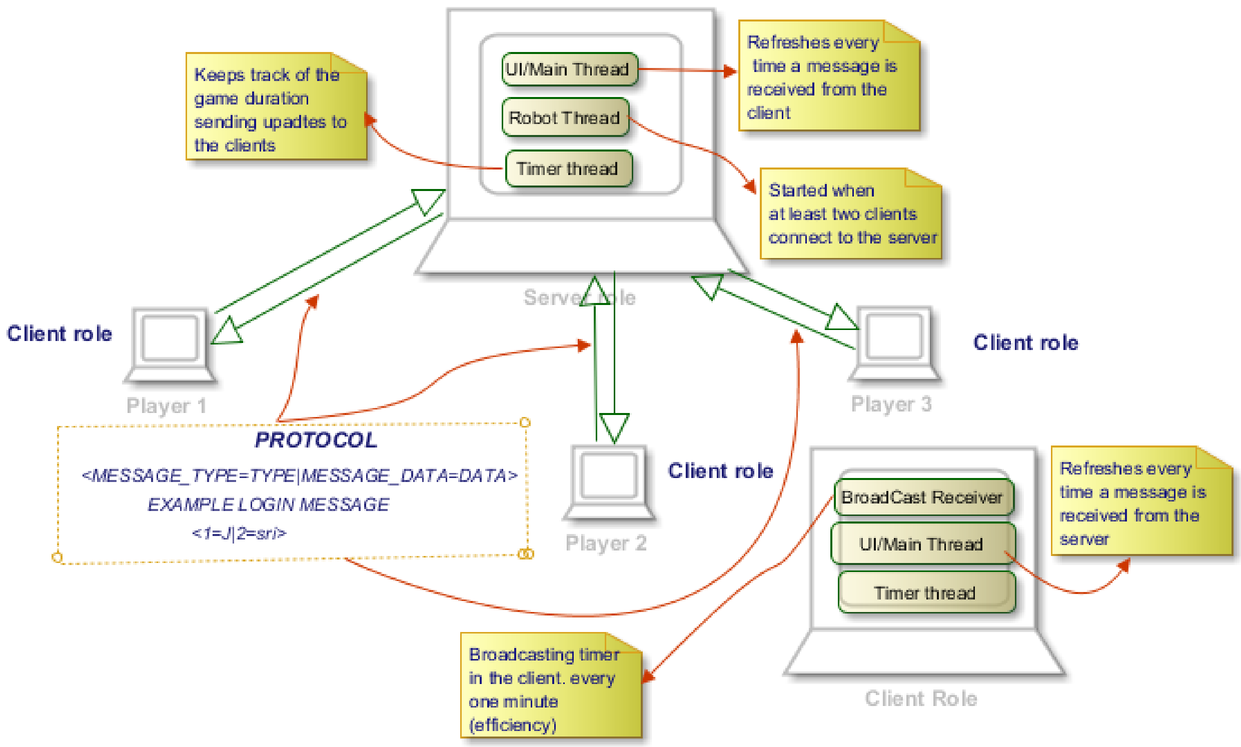
Keeps track of the game duration sending upadtes to the clients
UI/Main Thread
Robot Thread
Timer thread
Refreshes every time a message is received from the client
Started when at least two clients connect to the server
Server role
Client role
Player 1
Client role
Player 3
PROTOCOL
<MESSAGE_TYPE=TYPE|MESSAGE_DATA=DATA>
EXAMPLE LOGIN MESSAGE
<1=J|2=sri>
Client role
Player 2
BroadCast Receiver
UI/Main Thread
Timer thread
Refreshes every time a message is received from the server
Client Role
Broadcasting timer in the client. every one minute (efficiency)

Figure 2: The architecture of multiplayer centralized game

3.2 Multi-player decentralized (WiFi Direct)

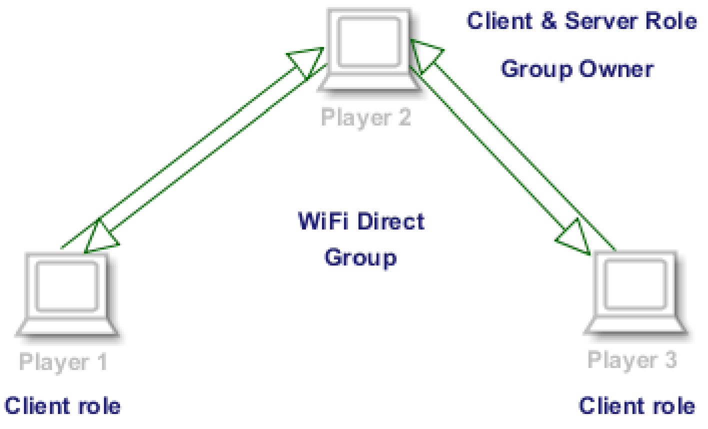

Figure 3: The architecture of multiplayer decentralized game

Figure 2 is explaining the whole architecture of the multiplayer centralized game in terms of connection establishment, protocol specification, running threads and activities. Our design of the whole Bomberman game is to separate the logic and GUI handling in a two different layers which will give more flexibility to developers. Any logic can be added into the logic layer with less modification in the GUI layer. Every thread inside the Bomberman game is not running continuously inside the while loop, instead it will be waiting for an external event to wakeup. This approach is saving more CPU resources and battery life. Our server has been designed to handle multiple clients request by using non blocking socket in Java, so every connection, read socket and write sockets events will handled asynchronously. It has given significantly higher throughputs when more clients are connected to server and performed more operations on their game panel.

Since our game logic is separated into different layers, all the movements from clients will be sent to server to decide effect. Upon successful movement, corresponding event will be broadcasted to all the connected clients. Even though it is hard to emulate all the standalone game operations in server side, we have managed to provide all the operations and some extra features such as score management, multiplayer game pause/resume and game duration timer.

All the server functionalities have been tested with maximum of 4 clients and gained desired performance with some smaller latency.

We have also tested our Wifi p2p implementation with maximum of 3 players (3 real Android devices). Initially, one of the devices will become an owner and p2p group will be established. After the connection establishment, server component and client component will be started up on the owner’s device. Other peers can be able to connect with owner as clients to start the multiplayer game. If an owner device is abandoning from the existing p2p group, owner’s server will inform all the connected peers to dumb their current view of the game in to temporary file for continuation in the next round. When the rest of peers start the game in next round, new owner will read the state information from the temporary file and remaining peers can continue with the game.

4. Implementations Choices

1) Our implementation targets the real android devices.

2) Our server has been implemented to support only PC in the centralized multiplayer game mode.

5. Conclusions

We have done lots of feasibility studies on non blocking asynchronous sockets, Android activities, handling application specific threads, passing messages to other background threads and finally Wi-Fi P2P. Lots of time has been spent on testing the application and fine tuning them. It will be great if we have some practical classes discussing about how to instrument an Android application and developing secure Android application.