Project Report

## Mobile Computing - 2013/04

Course: MEIC

Campus: Alameda

Group: 7

Name: André Bispo Number: 66941 E-mail: andrefsbispo@hotmail.com

Name: Diogo Pinto Number: 69905 E-mail: diogo.reis.pinto@gmail.com

Name: Ricardo Leitão Number: 69632 E-mail: ricardo.l.1992@gmail.com

## 

## 1. Achievements

To keep track of the development of the CMov project some features were requested. The following table contains all these features and their implementation state: fully, partially or not implemented.

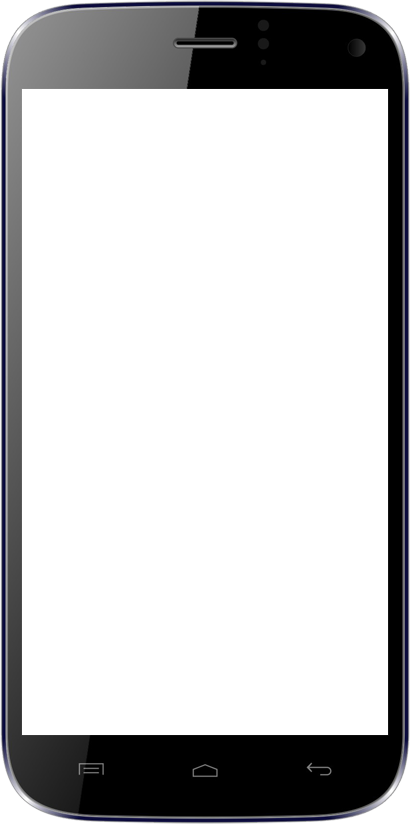
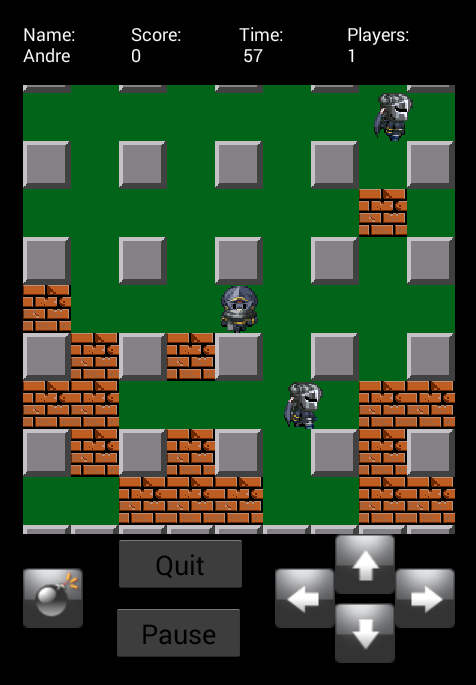
|  |  |
| --- | --- |
| **Feature** | **Implemented (Fully / Partially / Not implemented)?** |
| Game scene | Fully |
| Movement and life cycle of players | Fully |
| Bomb drop off and explosion | Fully |
| Movement and life cycle of robots | Fully |
| Collision detection | Fully |
| Score and game duration | Fully |
| Pausing / resuming the game | Fully |
| Handling of relevant activity lifecycle events (e.g., pressing home button) | Fully |
| Level selection | Fully |
| Multiplayer support | Fully |
| Clients leaving / joining the game | Fully |
| Server hand-over | Fully |
| Group Splitting | Fully |
| In WDSim mode peers can be waiting for the Group Owner to start a game. | Fully |
| Recursive Group Splitting | Fully |
| When GO leaves the group, the game running on the remaining peers is pause until a GO is assigned. | Fully |
| In Single Player and Multiplayer Centralized, players and robots have animations while moving, smoothing the gameplay. | Fully |
| When the game ends an alertDialog pops-up showing the game winner. | Fully |
| When joining a centralized mode game, a client waits in a lobby screen for other players, receiving information about them. | Fully |

## 

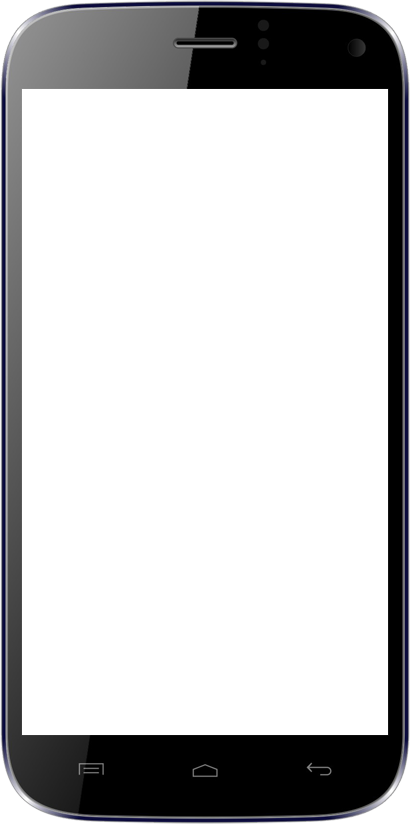
## 2. Specification

The following wireframe shows the different interactions the user can make when playing CMov’s Bomberman. Starting from the *Menu* if a user presses the settings button a new activity shows up. While the user is in the *Settings* state it can change many of the game properties, such as game duration, bomb duration and blast radius, player name, etc. If the user desires to save the settings just presses the save button, if changes were to be discarded the button cancel should be pressed. Either buttons send the player back to the *Menu.* If the user intends to go single player, by pressing the single player button on the main *Menu*, game settings are loaded and the transition to the *In Game* state is performed. When starting a centralized new game, the user is forwarded to the *Centralized Lobby,* there he can wait for other players to join and then start the game by pressing the corresponding start game button, being moved to the *In Game* state afterwards. Lastly, if a player desires to play a game with others using Wi-Fi direct, he is moved to the *WDS Lobby* state, there he can find a button to connect to the group owner and join his game. When accepted a transition to the *In Game* is executed.

Single Player New Game



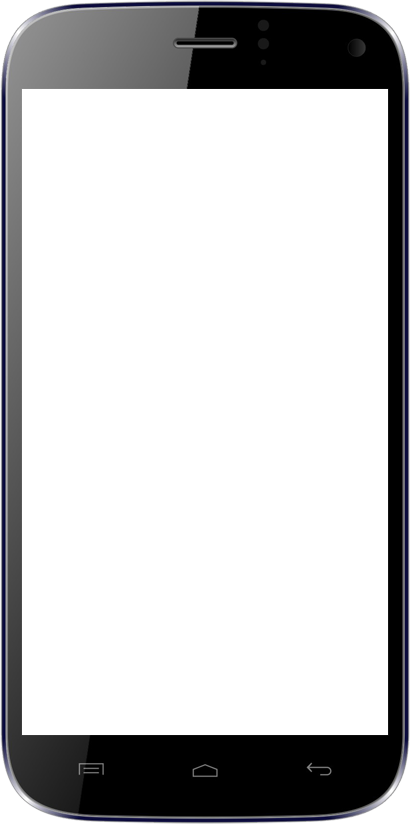
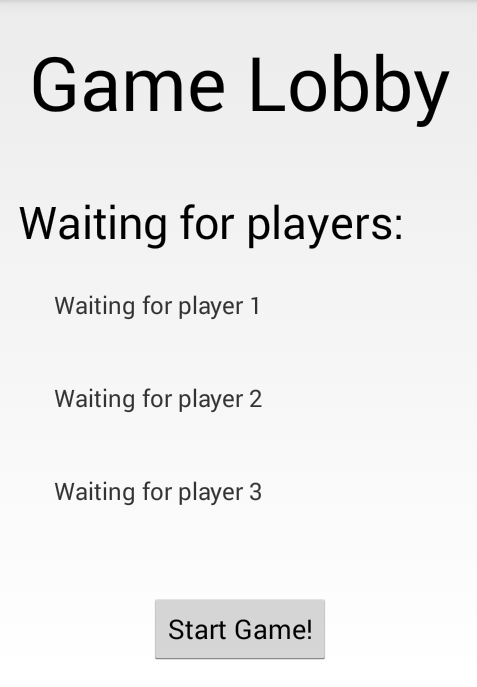
In Game



Menu

Centralized New Game

Centralized Start Game



Centralized Lobby

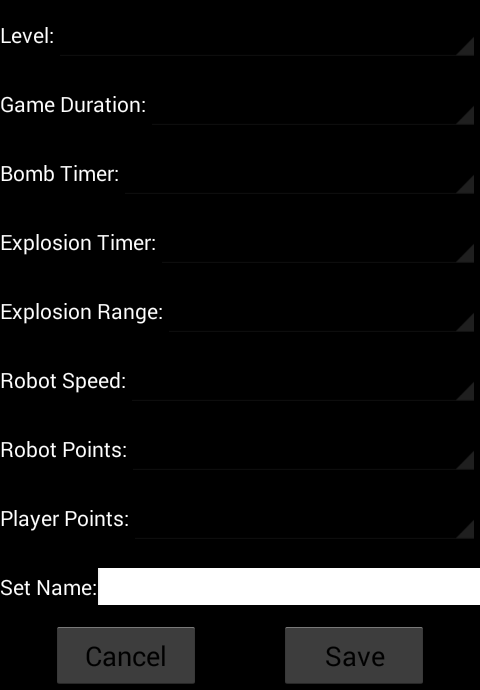
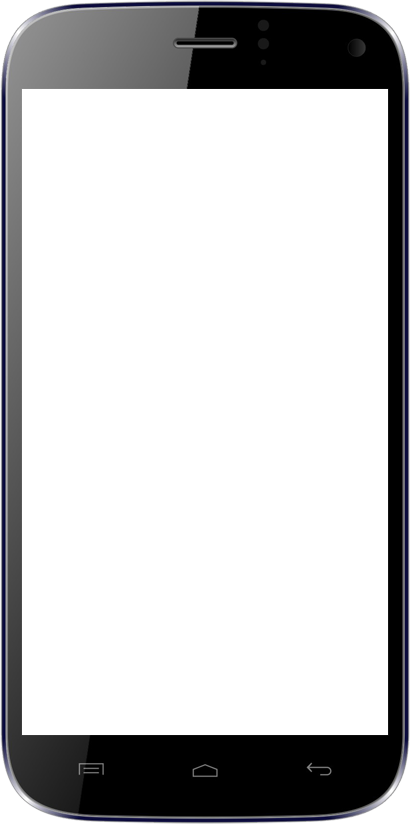
WDSim

New Game

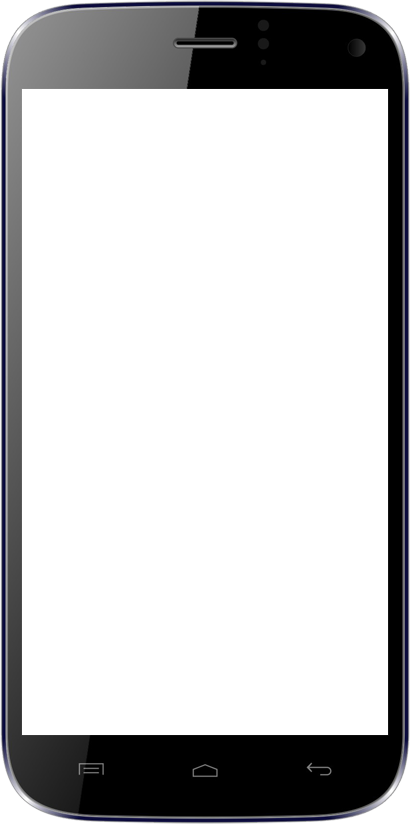
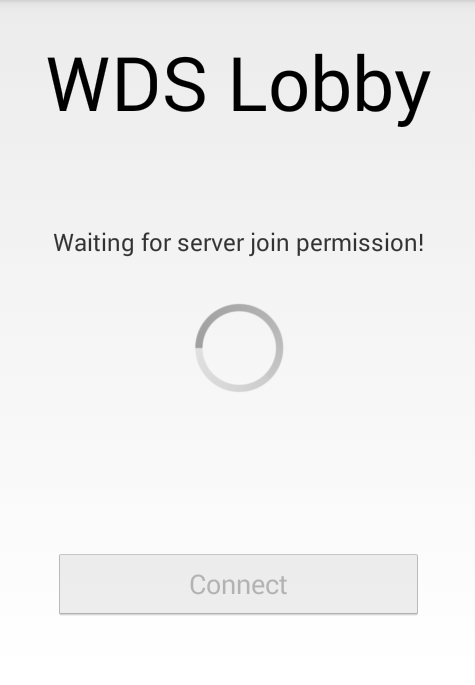
WDSim Join Game

## 

Settings



Settings



WDS Lobby

## 2. Design

The single player mode is a pretty straight forward game, consisting on an update game logic/redraw loop. Regarding android structure, there are just two activities launched when running a single player game, the menu (created when the application starts) and the main activity. This activity contains the HUD (buttons and text views) and the game canvas (Surface View). In order to keep the game running, a loop thread is started. This thread is responsible for the game logic, taking care of events like moving robots and player, placing bombs, explosions and removing destroyed objects. When all the game logic is computed in that loop, the game thread redraws the canvas with the updated information.

The multiplayer centralized mode consists on a java server running on a local machine, orchestrating the game among the participating clients. When the server is started, a socket is created to listen to player login requests until a logged player issues a game start. To maintain a consistent game the server is responsible for the robot’s movement and relaying incoming player messages (like player died or player quit game), to the other players. These tasks are possible because three threads are running on the server while a game is on course, the EndGameThread, MoveRobotsThread and UpdateMatrixThread. The EndGameThread, like the name suggests, keeps track of the number of players who are active in order to terminate the game if none are alive. The MoveRobotsThread computes the next position for the robots and sends to all players the direction in which they are going to move, waiting for the players’ ack response when their matrixes update. The UpdateMatrixThread listens to players messages and relays them to all other players in the game, performing on its matrix the required updates. For example, when a player places a bomb the server updates his matrix, this action has to be done because, when computing the next position of a robot, it has to be taken in account that a new bomb has appeared, therefore the robot cannot move into that cell. The protocol used between client and server is pretty simple. When logging in, a client issues a login request to the server receiving in response the name of the other players already waiting to start the game. After the login phase, one of the players sends a start game message, triggering on the server a broadcast of the game settings and which character the players are going to play. When the game is already going on, the server receives messages, broadcasts them to all players and updates itself. On the client side, there is an activity acting as a lobby where players wait for others to start a game. Unlike single player mode, to support the ability of receiving messages and updating the game state, clients run a thread to listen to server messages. These messages are then processed and the game state is updated accordingly.

The multiplayer decentralized mode takes advantage of P2P, allowing clients to communicate with each other, to remove the centralized server from the architecture. Activity wise, the only difference from the centralized mode is a lobby activity in which non group owner peers enter in order to join a game. When communicating with each other, peers use Wi-Fi direct technology and to process the signals from the WDSim API a service is bound to the application context. The WDSim version of Bomberman contains two game loop threads, one to process the messages sent from other peers and another, only ran by the group owner, to move the robots, broadcast the moving information among the peers playing the game and to check if the game has ended. By assigning these tasks to the group owner, the peers’ game state remain consistent, making the group splitting feature easy to implement. When the group formation changes, a new group owner is assign to continue the labor of the previous one. This design makes it possible for a group without a group owner, to be paused and wait for one to join. Communication protocol is pretty much message broadcasting to all peers playing the game, for example, player1 move left, then a message containing this information is broadcasted to all the peers, keeping everyone in the group consistent.

## 

## 3. Implementations Choices

The project consisted in 3 phases aimed for android 4.0 or above, but since phase1 and phase 2 did not required Wi-Fi direct, they were developed to support the available devices in the project work group (android below 4.0), making debugging and testing much easier. To develop phase 3, since the devices at disposal did not support Wi-Fi direct, a branch was created to isolate the code changes needed in order to work with emulators and the WDSim API. So the project done is separated in two projects, one running on real mobile devices, capable of single playing and multi playing with a centralized master, and another running on emulators capable of single playing and multi playing using the WDSim capabilities.

## 4. Conclusions

Taking in account the course objectives regarding learning criteria, the Bomberman project was a good way to put the theory in practice and implement an available, scalable and context aware application. But since a game requires good performance to be enjoyed, maybe it wasn’t such a good approach concerning the subject’s practical component. Our work group, as stated before, did not have at disposal android devices with Wi-fi direct capabilities, making the development of phase 3 quite painful, since the android emulators are very sluggish and it was required a minimum of 3 to test the project functionality properly. So to conclude and present some suggestions, a good way to apply the concepts of the course and still make the project a fun factor, it would be to purpose as project a less performance reliant application.