Project Report

## Mobile Computing - 2013/04

Course: MEIC

Campus: Alameda

Group: 1

Name: Daniel Bali Number: 79534 E-mail: janos.bali@tecnico.ulisboa.pt

Name: Gayana Chandrasekara Number: 79529 E-mail: gayana.withanage@ist.utl.pt

Name: Seçkin Savaşçı Number: 79503 E-mail: seckin.savasci@tecnico.ulisboa.pt

*(PAGE LIMIT: 5 pages – including the cover)*

## 

## 

## 1. Achievements

*Describe which features of the project specification were implemented by filling out the following table. For each feature, indicate its implementation state. If partially implemented, describe what was achieved. Feel free to add any other features you have incorporated into the project.*

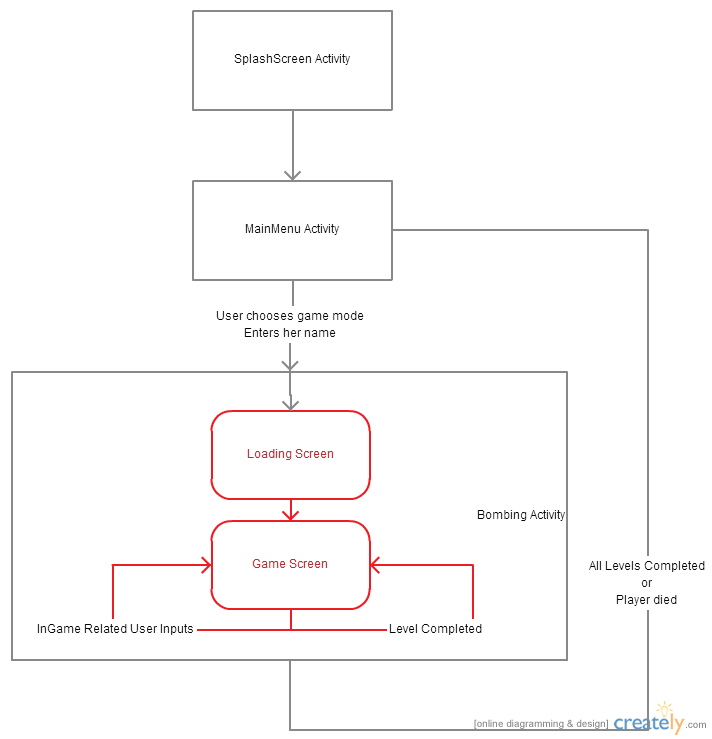
|  |  |
| --- | --- |
| **Feature** | **Implemented (Fully / Partially / Not implemented)?** |
| Game scene | Fully Implemented. |
| Movement and life cycle of players | Fully Implemented. |
| Bomb drop off and explosion | Fully Implemented. |
| Movement and life cycle of robots | Fully Implemented. |
| 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 | Not implemented. Players need to complete the current level to advance the next one |
| Multiplayer support | // TODO |
| Clients leaving / joining the game | //TODO |
| Server hand-over | //TODO |
| Group merging (or group spitting) | //TODO |
| … |  |

## 2. Specification

*Draw and describe the activity wireframe of your program. Describe any other additional relevant details regarding the program behavior that have not been specified by the faculty.*

In this project, we have implemented a multiplayer bomberman clone which runs on a custom game framework we wrote during this project. We aim to achieve a solid and smooth gaming experience which led to implement our own game framework and use OpenGL for game graphics.

In our current implementation, we have three activities. SplashScreen activity is responsible for loading and initializing general game resources. MainMenu is the activity where users chooses the game mode (multiplayer or not) and also inputs her player name. Bombing activity is our core activity which the actual game runs.



## 2. Design

*Describe the design of your program. This includes: the architecture of the program (single player, multiplayer centralized, multiplayer decentralized), the respective internal structure in terms of Android components (activities, services, and broadcast receivers) and threads, description of the game loop (i.e., handling input events, updating game state, and refreshing the display), description of consistency and network protocols (e.g., for formation and disaggregation of groups, handling failures, hand-over the server role, merging or splitting groups), etc. While describing the program design, indicate to what extent you took into account relevant issues to mobile computing, such resource efficiency, performance, fault tolerance, and usability. Discuss how the involved trade-offs affected your design decisions (e.g., in the game synchronization protocols, game scene refreshing algorithms, etc.).*

We have implemented //TODO architecture till the deadline of the project.

Our custom framework introduces “Screens” which are sub activities tailored for game development. Bombing activity contains several screens which are responsible for game logic and presentation flow. We have one main thread, one UI thread which is responsible for drawing and updating user interface, and another thread which is responsible for OpenGL. Except from activity and screen transitions, our core game logic can be described as follows:

while True:  
 call current screen’s update method  
 call current screen’s present method

In this scenario, the screen’s update method is responsible for updating the game state and handling game related inputs (Game unrelated inputs are handled by the encapsulating activity). The screen’s present method draws the current game world using OpenGL.

In our first iteration (which is evaluated in project checkpoint), we didn’t have animations. Our game update code blocks were tightly coupled with code blocks presenting the updates. In addition to this, we didn’t have clear separation between managing game resources and using them. To solve such problems, we implemented our custom game framework.

In earlier iterations using our framework, we have tried RelativeLayout and SurfaceView to present the game. Yet in both cases, we stuck 15-25 frames per second. This was unacceptable since we aimed to have smooth game experience. Finally we have switched to OpenGL ES 1.0. With OpenGL, our game runs around 60 frames per second. We assume we have done a good job while implementing the game framework, since 60 frames per second is the practical limit and we achieved it even with our custom framework overhead.

## 3. Implementations Choices

*Describe any relevant implementation choices, e.g., the targeted testing platform (WDSim or real devices), external libraries used, etc.*

//TODO mention GenyMotion, Android Studio etc

At the beginning of the project we mainly worked with the Eclipse and used the real devices for the testing. When it becomes the multiple player development phase we need to find a fast and reliable emulator in order to make the development life easier and it is experienced fact that the android in built bundled emulator is too slow. As a solution we moved to GenyMotion which is fast and reliable emulator which matched exactly to our requirements.

Moreover, at the same time we move to new emulator, we moved to the Android Studio which has much better capabilities integrated as an Android development IDE. Integrating OpenGL as an enhancement for the gaming screen was a challenging task but we thought of implementing that because it was vital in the end user experience point of view when we develop a game.

At the final phase where the multiplayer connectivity was based on Wifi Direct, we initially started development using WDSim but sooner we realized it is much faster and easier to move to the real devices because real devices provided more flexibility in device discovery and connectivity rather than WDSim. Obviously, the testing was much faster with real devices rather than using emulators at this phase.

## 4. Conclusions

*State the conclusions of this work. Please provide some input on how the practical component of the course could be improved in future editions.*