Multiplayer Pac-Man

Distributed Systems – Project Proposal

Stefan Oancea, Johannes Beck
ETH ID-1 14-944-912, ETH ID-2 14-805-279
oanceas@student.ethz.ch, beckjoh@student.ethz.ch

Markus Hauptner, Florian Morath
ETH ID-3 13-939-871, ETH ID-4 14-931-968
markuhau@student.ethz.ch, fmorath@student.ethz.ch

Linus Fessler, Tiziano Zamaroni
ETH ID-5 14-924-203, ETH ID-6 05-915-947
fesslerl@student.ethz.ch, ztiziano@student.ethz.ch

1. INTRODUCTION

Since 1980, the game Pac-Man has fascinated players around the world. Starting as an arcade game, it was adapted for many platforms while technology was improving and is still played today on mobile devices. While graphics have improved significantly over time and new features were added to the original idea of the game, there is still a major drawback to most of its versions as the game only provides single player user experience.

Our goal is therefore to exploit the opportunities of modern mobile devices in order to create a new user experience. This new user experience will consist of the well known Pac-Man game combined with a distributed multiplayer approach. The idea will be implemented as an Android application allowing the user to create a new instance of the game by starting a server on an Android device. Other users can then connect to this server with their own devices.

2. SYSTEM OVERVIEW

Figure 1 gives an overview of the Pacman app Activities. The app starts in the Main activity. Here, the user can choose to either edit the game settings, host or join a game by pressing corresponding buttons.

If the user presses the "Settings" button, the app will switch to the Settings Activity, where the device's local IP address is listed and the following properties can be changed:

1. The username that will be used in the game.
2. The port on which the local server would listen to.
3. The IP address and port on which the app would try to connect to a server.

If the user chooses the "Host" button, the app will switch to the Lobby Activity (Host). Here, the user can await new clients and see which ones are currently connected (identified by usernames). Furthermore, the user can decide to start the game by pressing a "Start" button. This will launch the Game Activity and start the actual game.

If the user chooses the "Join" button, the app will switch to the Lobby Activity (Client). Here, the user can see the host's username and which clients are currently connected (identified by username). In contrast to the host's view, the client can not see a "Start" button, but instead has to wait for the host to start the game.

Once the game started, a Score activity keeps track of the current performance of the players. The game will be played in rounds, such that every player plays as Pac-Man once. If Pacman manages to take all the points that are positioned on the map or looses all his available lifes, the game ends and the player roles (who plays which figure) will be distributed anew. Furthermore, a timer will keep track of the time played and end the game if a certain time has been reached.

Independent from the device's role as host or client, the app will automatically switch to the Score Activity after the game has ended. This Activity displays each users score (determined by the gameplay performance). The only following action is to return to the Main Activity by pressing a button.

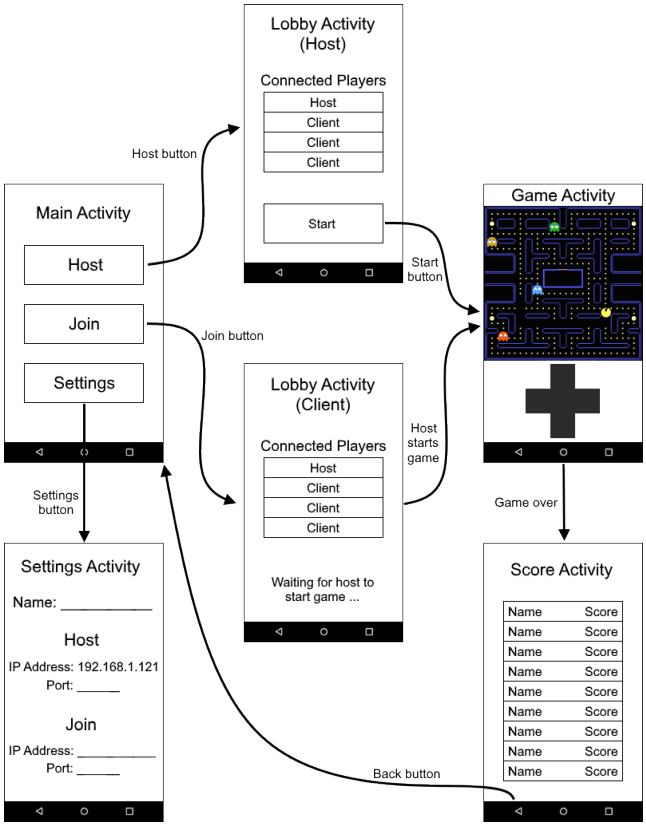

Figure 1: Activity Overview

Figure 2 provides a raw draft of the game's architecture. The following three threads will be used:

1. Communication Thread:
   To prevent blocking of the UI thread, all network communication will be outsourced to a separate communication thread. If the device works as server (host), this communication thread will be a server thread and needs to deal with receiving and distributing network messages from multiple clients. If the device works as client, the communication thread only needs to send outgoing commands and receive game state.

2. Model Thread:
   The model thread keeps track of the game state and is responsible for state updates as well as computations on the state. Furthermore this thread receives user commands provided by either the communication thread or the user interface thread.

3. UI Thread:
   The UI thread is responsible for displaying the current game state to the user.

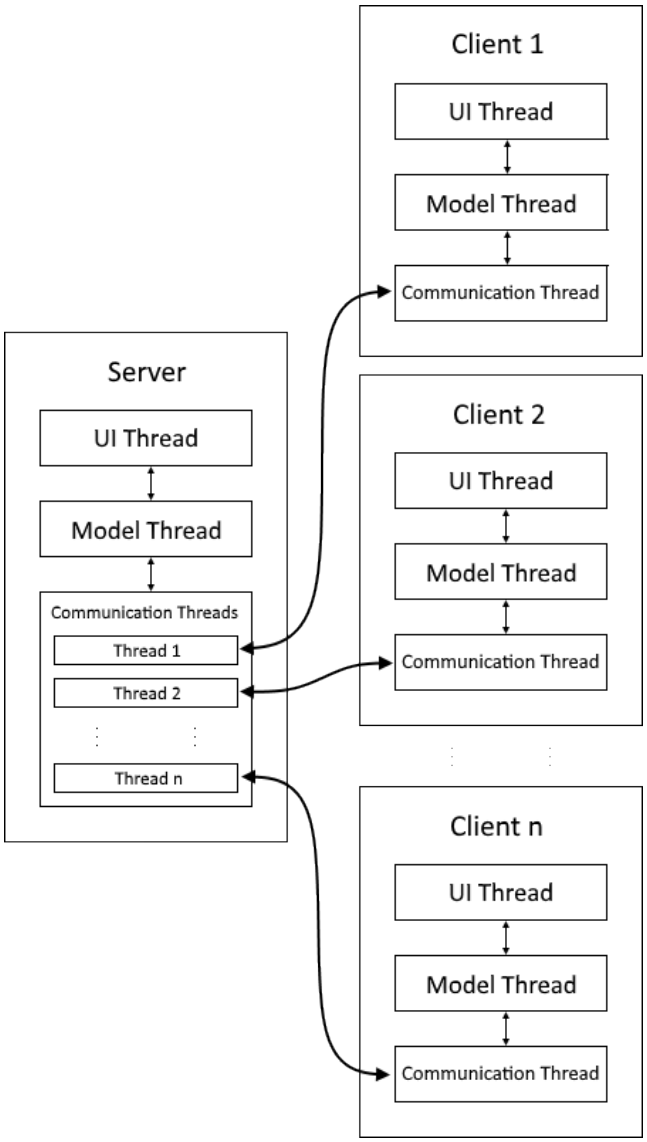

Figure 2: Threads

3. REQUIREMENTS

1. The game can be played on multiple Android devices (at least two).
2. The gameplay should work as follows:
   (a) One player plays as Pac-Man.
   (b) One or multiple other players play as ghosts.
   (c) Each player figure constantly moves forward unless the movement is blocked by a wall element. This means that the only way to completely halt movement for a player is to run into a wall without changing direction.
   (d) In case it helps balancing the gameplay, power-ups (objects that provide special abilities to Pac-Man or the ghosts) spawn randomly on the map. Those power-ups might improve or worsen the performance of the player figures. Power-Ups can be collected by simply moving over them.
   (e) The amount of rounds played corresponds to the amount of players connected.
   (f) Pac-Man starts with a predefined amount of lifes.
   (g) When Pac-Man is being captured by a ghost, he loses one life and all ghosts will return to their spawn.
   (h) The round ends if either all coins have been collected, Pac-Man has lost all his lifes, or the round-timer (predefined and the same for each round) reaches zero.
   (i) Each player plays as Pac-Man for one round and as ghost for the remaining rounds.
   (j) After all rounds have been played, the game ends and the players can see the amount of points achieved by each player.
   (k) The host will start as Pac-Man in the initial round.
   (l) Clients will start as ghosts in the initial round.
3. Each player must use one Android device in order to control his figure (Pac-Man or ghost).
4. The map (board) on which the players move should provide the following features:
   (a) Pac-Man starts on a predefined location (Pac-Man spawn).
   (b) The ghosts (one ore multiple) start on predefined locations (ghost spawns).
   (c) Player figures can only move up, down, left and right.
   (d) The only structuring elements of the map are walls.
   (e) Walls have either horizontal or vertical orientation.
   (f) Player figures can not move through walls.
   (g) The map has rectangular shape and is limited by walls at its borders (horizontal walls along the left and right border, vertical walls along upper and bottom border). On two ore more locations the wall on the border of the map is open and players can teleport to the other side of the map. Teleporting means that players exiting the board on the right side will spawn again on the left side and vice versa.
   (h) Coins are distributed evenly on the game map (board).
   (i) Every coin on the map must be reachable by all player figures.
   (j) When Pac-Man collects a coin by moving over it, the coin disappears from the map and Pac-Man receives points.
   (k) The player with the most points wins the game. In case of a draw, the player that completed his round the fastest is the winner.
5. The communication among the devices will be implemented on top of TCP (or UDP).
6. The minimum SDK required is API 21: Android 5.0 (Lollipop). The target SDK version is API 24: Android 7.0 (Nougat).
7. On the hosting player's device, a service will be started to run the server which the clients can connect to.
8. In order to ease certain game development aspects, libGDX[1], AndEngine[2], or similar java game development frameworks might be used.

4. WORK PACKAGES

- **WP1**: Define appropriate model for game situation (state) and methods to change the state.
- **WP2**: Design graphical representation for Pac-Man player figure.
- **WP3**: Design graphical representation for ghost player figure.
- **WP4**: Design graphical representation for map elements (coins, walls, unoccupied positions).
- **WP5**: Implement Game activity that connects model and graphical representation. It should also allow the user to control its figure.
- **WP6**: Define a communication protocol to synchronize the game's state across devices. Choose appropriate transport protocol.
- **WP7**: Implement the communication protocol - part 1: Server.
- **WP8**: Implement the communication protocol - part 2: Client.
- **WP9**: Combine Game activity with the communication protocol.
- **WP10**: MainActivity, LobbyActivity, SettingsActivity, ScoreActivity design.
- **WP11**: MainActivity functionality:
  1. Host button starts LobbyActivity with boolean intent extra "is host" set to true to then enable Start button and disable "Waiting for host to start game" text in LobbyActivity.
  2. Join button starts LobbyActivity with boolean intent extra "is host" set to false to then disable Start button and enable "Waiting for host to start game" text in LobbyActivity.
  3. Settings button starts SettingsActivity.
- **WP12**: LobbyActivity functionality:
  1. ListView to list all connected player (might needs to be scrollable depending on how many player we allow in one game).
  2. Host can press Start button to start game for all players.
- **WP13**: SettingsActivity functionality:
  1. Name EditText: default name is "Anonymous" or device name.
  2. Host IP Address TextView: Shows the device's LAN IP Address.
  3. Host Port Address EditText: Allows to change the server's port.
  4. Join IP Address EditText: Allows to change the IP Address to connect to when joining a game.
  5. Join Port Address EditText: Allows to change the port at which to connect when joining a game.
- **WP14**: ScoreActivity functionality:
  1. ListView (possibly scrollable) to show Highscores list. When the game ends, every player will land in the ScoreActivity, receive the list from the host and display it.
  2. The Back button returns to the MainActivity.

5. MILESTONES

Work package distribution:

- **Group**: WP1, WP6
- **Stefan Oancea**: WP5, WP13
- **Johannes Beck**: WP7-8
- **Linus Fessler**: WP2-4
- **Markus Hauptner**: WP9, WP14
- **Tiziano Zaramoni**: WP10, WP12
- **Florian Morath**: WP5, WP11

Schedule:

- Week 1: In the first week, the general program structure should work. This includes menu navigation, the settings menu, the lobby, and a class for the game activity. Furthermore WP1 should be finished.
- Week 2: In the second week, the game map should be drawn and the player can move Pac-Man around the map and collect coins.
- Week 3: In the third week, the basics of the game should already work. This includes movemet of Pac-Man and the ghosts. Collision detection and networking.
- Week 4: The last week should be used for bug fixing, gameplay balancing and optimizations.

6. REFERENCES

[1] libGDX `https://libgdx.badlogicgames.com/` Accessed on 18 Nov 2016

[2] AndEngine. `http://www.andengine.org/`. Accessed on 16 Nov 2016.