**Gomoku**

Project documentation

2016

Juraj Kubala

# Obsah

[Introduction 3](#_Toc466232074)

[Rules 4](#_Toc466232075)

[Objectives 5](#_Toc466232076)

[Use Cases 6](#_Toc466232077)

[Mockups 7](#_Toc466232078)

[Components planning 9](#_Toc466232079)

[Board 9](#_Toc466232080)

[Button 10](#_Toc466232081)

[ScoreBoard 10](#_Toc466232082)

[Tile 10](#_Toc466232083)

[Database planning 11](#_Toc466232084)

[Mobile 11](#_Toc466232085)

[Backend database – server side 11](#_Toc466232086)

[Backend – server side 12](#_Toc466232087)

[Class diagram 14](#_Toc466232088)

[Time planning 15](#_Toc466232089)

# Introduction

Gomoku is a strategy board game. It also could be called Gobang or Five in a Row.

It is traditionally played with special 'Go' white and black stones on a board. You can also play it as a paper game. In our case it will be virtual game board on the mobile screen.

Main purpose of this project is to build mobile app, which will be able to provide at least 3 game modes: multiplayer, single player, friendly match. In the case, you don’t have an internet access there will be single player mode. If you have an internet access you can be more competitive and you can choose between multiplayer with random player or friendly match with you friend.

# Rules

There are many variations of rules around the world. Base rule is same everywhere, you have to have 5 your stones in a row. Player, who is starting, is randomly chosen at the beginning of game. Than players alternate in placing stones to empty fields. The winner is the first player to get an unbroken row of five stones horizontally, vertically, or diagonally.

# Objectives

1. Create mobile game which can be played offline
2. Create artificial intelligence
3. Make competitive environment where can players play against each other
4. Allow friends to play friendly game
5. Make this app for Android, but with the possibility to make it for iOS in the future

This app is created in React native, which allows creating multiplatform app while you are writing same code. Main goal is to do app for Android, but in future if it will be reasonable make version also for iOS.

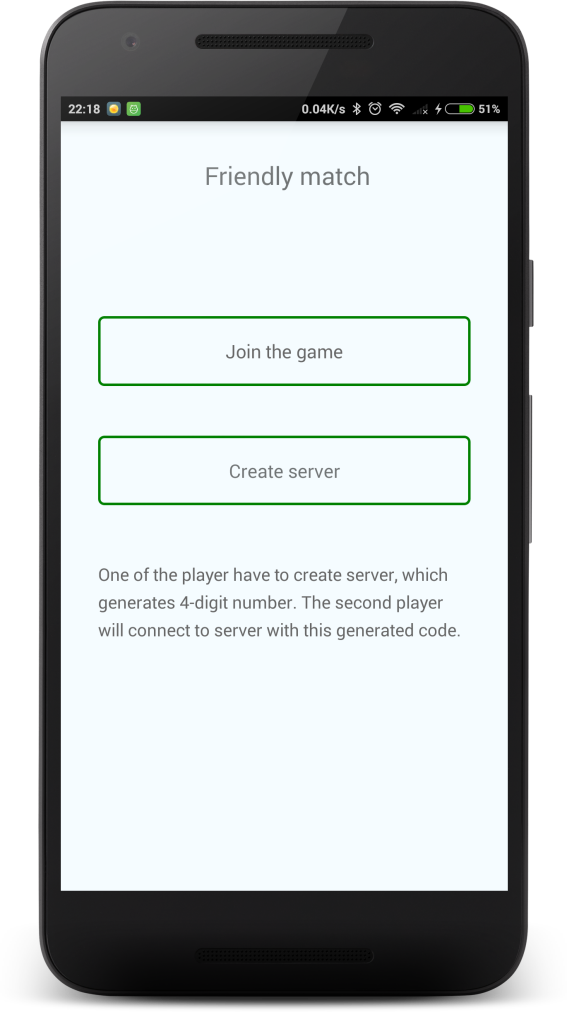
All stated requirements were fulfilled. Application is finally written in Typescript, which was good choice because of supporting types and static type checking. Different stages of development is described in the end of this document.

# Screenshots and UI functionality

## **Home screen**

From home screen we can go to:

* Play multiplayer game
* Play single player game
* Play friendly match
* Manage application settings

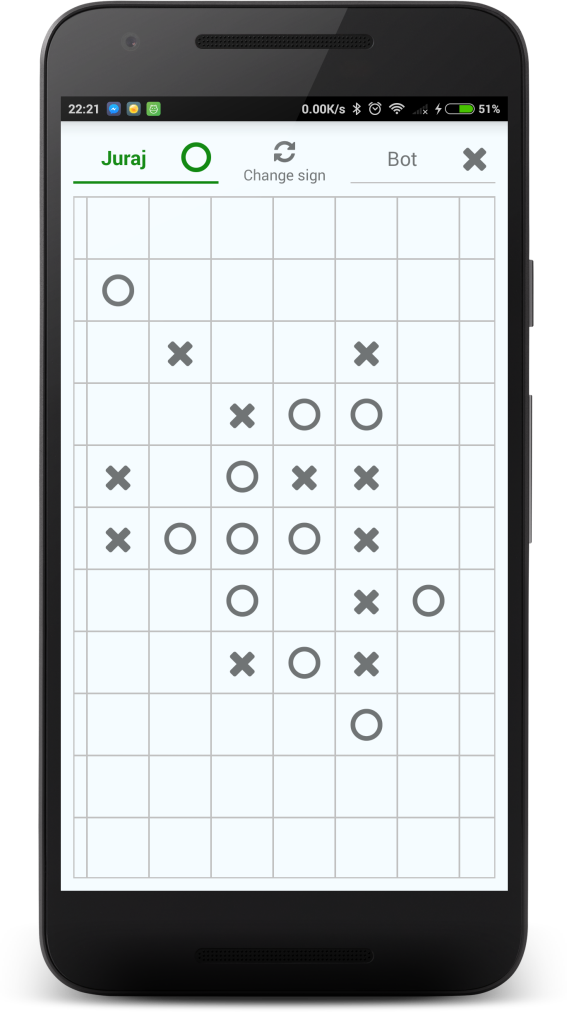


**Friendly match**

If we want to play match with friend, we need to do a bit settings. One player has to create server. It will generate server code which the second player has to enter this code to join the right game. After this setting, players can play

## **Loading screen**

Loading screen appears before showing game board. It shows progress of connecting to server or just preparing single player game.

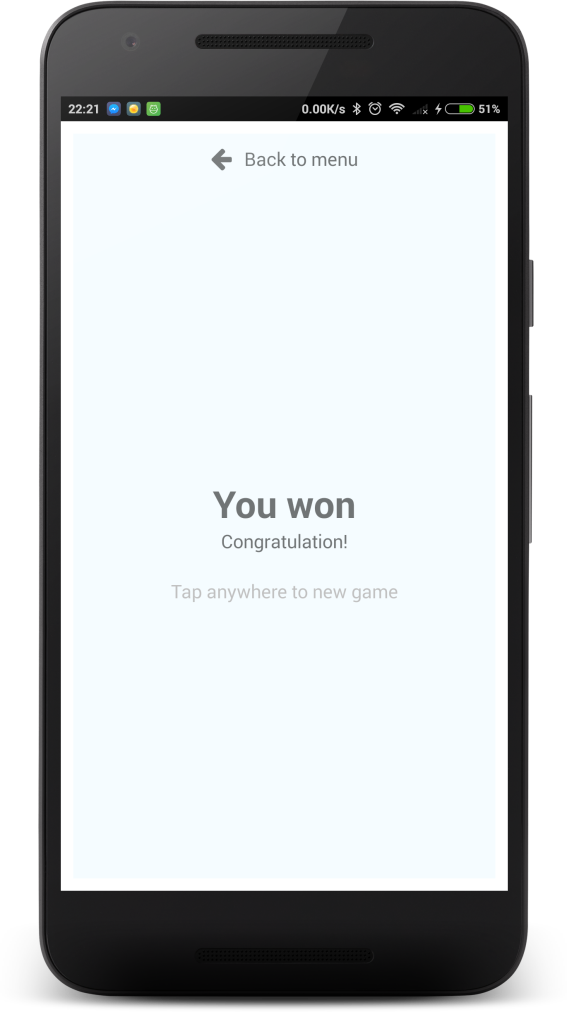


## **Game**

This is most important screen. Here whole game passing off. We can see opponent’s name. In this case, we are playing against bot in single player mode. We can also change players sign by pressing button in the middle of top bar.

## **Settings**

Here is one input field in this time. More settings may come in future.



## **After game screen**

This screen appears after game or when opponent leave game. We have to possible ways from this screen. One is to play new game and the other is to go to main screen.

# Interfaces

Application is written in Typescript, which allows us to create types via interfaces. These makes app more consistent and leads us to using same name. In application each component has interface of its properties, but these are descripted later. Here are common interfaces used across application.

interface **IUser** {

nick: string;

}

interface **IPlayer** extends **IUser** {

id: string;

}

interface **IMove** {

player: IPlayer;

row: number;

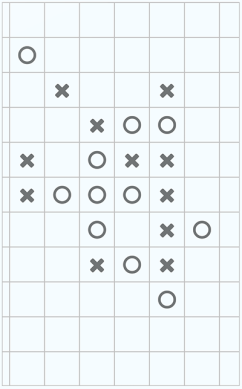
column: number;

}

# Components description

Creating components are major part of building React native app. Components can be composed of other smaller components. This allows us to reuse a lot of code and save tons of time. Behavior and look of each component can be change or managed by passing properties to component from parent component.

## **Board**

****Component which renders game board.

Props:

* lastMove: IMove

Object of last move. It is used for moving to last move if opponent makes move which you can’t see, board will move to position when you will be able to see it. It is passed to PanResponderEnhancer.

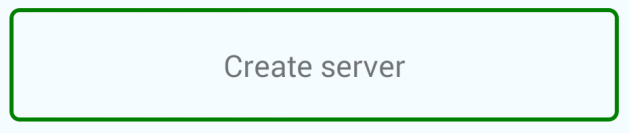
* onTouch = (row: number, column: number) => void

Function which is called when user touch the board. It is passed to PanResponderEnhancer.

* mappedMoves: {[key: number]: {[key: number]: number}}

Moves prepared in structure which is fast for rendering signs of players at the board.

## **Button**

****Wrapper for React button. Its style is customized for app design consistency.

Props:

* style?: ViewStyle
* innerStyle?: ViewStyle
* borderWidth?: number

Optional attribute for defining width of border. Default value is 2.

* onPress: () => void

Function which is called when user press button.

## **PanRespondEnhancer**

Component which wraps functionality of board movements. It checks if the game board is on screen and corrects movements of the board.

Props:

* width: number

Width of wrapper component. It is important for computing boundaries of movements.

* height: number

Height of wrapper component. It is important for computing boundaries of movements.

* lastMove: IMove

Object of last move. It is used for moving to last move if opponent makes move which you can’t see, board will move to position when you will be able to see it

* onTouch: (x: number, y: number) => void

Function which is called when user touch the board.

## **ScoreBar**

****In this bar are shown names of the players and theirs signs. There is also button for changing signs.

Props:

* me: IPlayer
* opponent: IPlayer
* playerInTurn: IPlayer
* onSingChangeClick: () => void

Function which is called when user press button for changing signs of players.

* playerToSignMapping: IPlayerToSignMapping

Mapping which defines which player has which sign.

## **Tile**

One tile of game board.

Props:

* sign: number

Attribute which defined how the tile will be rendered. This attribute can have value for empty field, for cross or circle sign. Use constants TILE\_ZERO, TILE\_CIRCLE, TILE\_CROSS.

# Database and managing state of app

When app gains complexity, it is important to set up one way of how we manage state of whole app. Each component could have own state, but this approach leads to raising mess in project. Other, very common approach is to set up one global state for whole app and manage this state with predefined functions called “actions”, not directly. This approach is called “single source of truth”. Creating and managing global state is provided by **Redux** library.

It is possible to attach listeners for calling certain actions, and this functionality I’ve used for saving important information in mobile phone via React library **AsyncStorage**. Previously planned class for managing saving data in mobile was reduced to one callback function in listener.

Global state:

{

net: {

status: INTERNET\_STATUS.\*

},

user: {

nick: string

},

game: {

id: string,

me: IPlayer,

opponent: IPlayer,

playerInTurn: IPlayer,

moves: IMove[],

status: GAME\_STATUS.\*,

winner: IPlayer

},

websocket: {

status: WEBSOCKET\_STATUS.\*

},

playerToSignMapping: {

circle: IPlayer,

cross: IPlayer

}

}

Action for updating state:

Game related actions:

* initGame(me: IPlayer, opponent: IPlayer, playerInTurn: IPlayer = null, gameId: string = '')
* makeMove(row: number, column: number, player: IPlayer)
* addMove(row: number, column: number, player: IPlayer)
* changeGameStatus(status: string, winner: IPlayer = null)
* changeSigns()
* resetGame()

User related actions:

* updateUserData(user: IUser)

Websocket related actions:

* updateWebsocketStatus(status: WEBSOCKET\_STATUS)

## 

# Backend – server side

For server side app I have used node.js server. Communication with mobile phones is provided via WebSocket which is supported by React-native. Communication is going through sending string messages with specific structure. String message is made via serialization of JSON object. Messages has strict structure:{“type”: string, “payload”: object}

* **fetch-game (phone -> server)**

Message send from mobile phone to server when user is looking for new multiplayer or friendly game.

**payload**: {

player: IPlayer, serverCode: number

}

* **fetch-game-response (server -> phone)**

Message send to mobile phone as a response for fetch-game message.

**payload**: {

opponent: IPlayer, playerInTurn: IPlayer, gameId: string

}

* **new-move (phone -> server -> phone)**

Message is send from mobile phone to server when user makes new move, and this move is send to opponent’s mobile phone.

**payload**:{ move: IMove }

* **opponent-left** **(phone -> server -> phone)**

Message is send from mobile phone to server when user leave game and is resend to opponent’s mobile phone.

**payload**: {}

# Time planning

* 1.week – making of shared components, small components (button, tile, scoreboard, board)
* 2.week – making of algorithm for robot which will be used in single player game
* 3.week – preparing server side app, start with multiplayer game
* 4.week – finish multiplayer game, make friend game mode, final touches, make presentation