CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS VII - UNIDADE TIMÓTEO - Engenharia da Computação

Sistemas Distribuídos
André Marcelino de Souza Neves e Leonam Teixeira de Vasconcelos

Trabalho 01:
Comunicação Entre Processos: sockets

1 Introdução

Nesse relatório, descreveremos como implementamos e como utilizamos o software que desenvolvemos para o trabalho de *socket* para a disciplina de Sistemas Distribuídos.

1.1 A proposta

Para esse trabalho, decidimos fazer um jogo simples. O jogo inicialmente era *singleplayer* e a nossa intenção era de transformá-lo em um game competitivo de dois jogadores. O jogo em questão é o game *drench*. Esse jogo consiste em uma matriz quadrada onde cada célula possui uma cor (pertencente a um conjunto de **n** cores). O jogador tem apenas uma ação: trocar a cor da célula localizada na extremidade superior esquerda da matriz – para evitar repetições e a escrita desnecessária de termos grandes, iremos nos referir à essa célula como **célula raiz**. Ao trocá-la, a cor é então propagada para todas as células adjacentes que possuem – em seu estado anterior – a mesma cor que a célula raiz. O objetivo do jogo, é cobrir toda a matriz com uma única cor, tendo um limite de trocas permitidas. Um exemplo online do *drench* pode ser encontrado **neste link**. Ainda mantivemos a possibilidade de um único jogador jogar, mas tal funcionalidade não é interessante para o presente trabalho.

A tecnologia utilizada para criar o game foi o **Flutter**, por motivos meramente pessoais: ambos os membros da equipe possuem certa experiência com o *framework*, e por mais que a ferramenta possa não ser a mais apropriada para tal tarefa, o desenvolvimento seria mais fluido, justamente pelo costume dos desenvolvedores com a ferramenta.

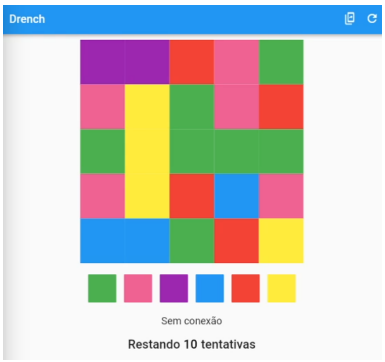

Figura 1: Print do jogo. Fonte: autores.

2 Desenvolvimento

2.1 A engine

Para desenvolver a *engine* do *drench*, fizemos a engenharia reversa da versão online que encontramos (**link**) baseando-se no nosso entendimento do jogo. Concluímos que um algoritmo recursivo seria a abordagem mais simples de se codificar. Após algumas versões, chegamos no algoritmo abaixo:

```
1   void paintFirstSquare(int colorIndex) {
2     if (colorIndex == _matrix[0][0]) {
3       return;
4     }
5
6     _paintsCount++;
7
8     this.propagateColorInMatrix(colorIndex, _matrix[0][0]);
9   }
10
11  void propagateColorInMatrix(int newValue, int oldValue, [x = 0, y = 0]) {
12    if (x >= size || y >= size || x < 0 || y < 0) {
13      return;
14    }
15
16    if (_matrix[x][y] != oldValue && (x != 0 || y != 0)) {
17      return;
18    }
19
20    _matrix[x][y] = newValue;
21
22    propagateColorInMatrix(newValue, oldValue, x, y + 1);
23    propagateColorInMatrix(newValue, oldValue, x, y - 1);
24    propagateColorInMatrix(newValue, oldValue, x + 1, y);
25    propagateColorInMatrix(newValue, oldValue, x - 1, y);
26  }
```

O código acima verifica se a célula raiz tem uma cor diferente da selecionada, e caso seja, incrementa o número de trocas do jogador, e propaga a cor na matriz. A propagação acontece de forma recursiva, para cada uma das células imediatamente acima, abaixo, direita e esquerda da célula raiz. Como a célula raiz não possui vizinhas à esquerda e acima, essas duas chamadas de função são ignoradas para essa célula em específico.

As condições de parada (casos base) para essa recursão são os caros onde as coordenadas extrapolam o tamanho da matriz ou quando as cores das células se diferem da cor antiga.

2.2 Execução

É possível executar o software tanto em desktop (testamos apenas no linux) e em mobile (testamos apenas em android). Como o *flutter* é um *framework cross-platform* podemos facilmente executar o sistema para diferentes dispositivos. No nosso caso, testamos no linux e no android. Abaixo temos a página de instrução para a instalação das dependências do flutter para desenvolvimento android e desenvolvimento desktop:

- **Instalação do flutter**
- **Flutter para desktop**

2.3 Sobre sockets

A comunicação entre os dois jogadores é feita utilizando *sockets*. A linguagem *dart* – linguagem utilizada para desenvolver aplicativos em flutter – possui suporte nativo a tal recurso [3] [4]. Como o trabalho requeria o uso de ambos TCP e UDP, o software permite que a conexão ocorra utilizando qualquer um dos dois protocolos.

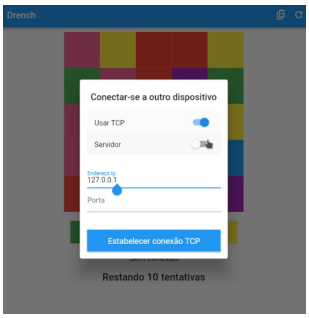

Figura 2: Menu de configuração. Fonte: autores.

2.4 Funcionamento

Abaixo, descreveremos o funcionamento geral do aplicativo (aquilo que é comum em ambos os protocolos) e também explicaremos aquilo que é diferente em cada um dos métodos. Para ver o código inteiro, vá na seção de Anexos (4) ou acesse o repositório públic no **github**

2.4.1 Funcionamento geral

Existem algumas trocas básicas de mensagens que acontecem entre os dispositivos que estejam utilizando esse aplicativo. Essas trocas foram elaboradas baseando-se na própria mecânica do jogo.

Para que a informação flua, precisamos primeiramente de estabelecer um padrão de envio e recebimento dessas mensagens. Chegamos na conclusão que enviar mensagens como **JSON** (Javascript Object Notation) seria a forma mais simples de realizar tal tarefa. O fluxo simples das mensagens acontecem da seguinte forma:

1. Remetente codifica a mensagem em formato **JSON** para uma string
2. Remetente envia a mensagem via um dos protocolos utilizados
3. Destinatário recebe a mensagem via um dos protocolos utilizads
4. Destinatário decodifica a string recebida para **JSON novamente**

Dois tipos de comandos trafegam de um dispositivo para outro: informações sobre a jogada do ultimo jogador (qual cor ele(a) selecionou) e sincronização entre dois jogadores junto com *reset* do jogo (Ao

iniciar o aplicativo, ambos os jogadores terão a matriz totalmente diferente, pois elas são geradas aleatoriamente). O padrão dos objetos transferidos é o seguinte:

```
{
    type : 'syncBoard'
    board : ...,
    reset : ...
},
{
    type : 'updateBoard'
    colorIndex : ...
}
```

Caso o objeto recebido pelo destinatário seja do tipo `syncBoard`, siginifica que a ação realizada é do tipo sincronização e que o tabuleiro do oponente está também nos dados recebidos (no campo `board`). Caso seja do tipo `updateBoard` a ação realizada é de atualização da matriz com a cor recebida no parâmetro `colorIndex`.

Caso o objeto recebido seja do tipo `syncBoard` e o parâmetro `reset` seja passado como verdadeiro, o contador do destinatário volta para o estado inicial (com o valor máximo de jogadas permitido).

O código que lida com o dado recebido está abaixo (esse código está no arquivo `drench_controller.dart`, que gerencia todo o estado do jogo):

```
1   void handleSocketData(value) {
2     print('===== data received');
3     print(value);
4
5     if (value['type'] == 'updateBoard') {
6       this.updateBoard(value['colorIndex']);
7       return;
8     }
9
10    if (value['type'] == 'syncBoard') {
11      List<List<int>> board = new List<List<int>>();
12
13      value['board'].forEach((vector) {
14        List<int> list = new List<int>();
15
16        vector.forEach((value) {
17          list.add(value as int);
18        });
19
20        board.add(list);
21      });
22
23      if (value['reset'] == true) {
24        this.newGame(false);
25      }
26
```

```
27        this.syncBoard(board);
28        return;
29      }
30    }
```

Os métodos para enviar os dados estão listados abaixo:

```
1     sendBoardSync(List<List<int>> board, bool reset) {
2       this
3           ._socketConnectionService
4           .sendData({'type': 'syncBoard', 'board': board, 'reset': reset});
5     }
6
7     sendBoardUpdate(int colorIndex) {
8       this._socketConnectionService.sendData({
9         'type': 'updateBoard',
10        'colorIndex': colorIndex,
11      });
12    }
```

Para armazenar todos os parâmetros de conexão, criamos uma classe chamada **ConnectionParams**:

```
1   class ConnectionParams {
2     bool isTcp;
3     bool isServer;
4     String ipAddress;
5     int port;
6     String remoteIpAddress;
7     int remotePort;
8
9     ConnectionParams(
10        {this.isTcp,
11        this.isServer,
12        this.ipAddress,
13        this.port,
14        this.remoteIpAddress,
15        this.remotePort});
16
17    Map<String, dynamic> toJson() => {
18          'isTcp': isTcp,
19          'isServer': isServer,
20          'ipAddress': ipAddress,
21          'port': port,
22          'remoteIpAddress': remoteIpAddress,
23          'remotePort': remotePort,
24        };
25  }
```

onde:

- **isTcp** diz se a conexão acontece via TCP
- **isServer** diz se o dispositivo é servidor (necessário apenas para conexão via TCP)
- **ipAddress** endereço de ip do dispositivo
- **port** porta onde será realizada a conexão do dispositivo
- **remoteIpAdress** endereço de ip do dispositivo remoto (destinatário)
- **remotePort** porta do dispositivo remoto

2.4.2 TCP

O primeiro teste foi realizado com o protocolo TCP, o qual foi implementado com a utilização da API nativa do dart *Socket* [4]. Foi utilizado o tutorial de James Slocum [1] como referência.

Utilizando esse método, precisamos distinguir entre cliente e servidor. Na figura (2), podemos selecionar o *switch* **Servidor** para que esse dispositivo, nessa instância do jogo se comporte como um servidor. O dispositivo que irá jogar contra este servidor, será um cliente. A conexão entre cliente é feita através dos campos **Endereço ip** e **Porta**. O cliente irá inserir o ip e porta do servidor e assim a conexão será estabelecida. Vale ressaltar que esse game foi testado somente em rede interna, e observamos alguns problemas em estabelecer conexão quando o *firewall* estava ativado (no caso do aplicativo desktop) e quando utilizamos portas abaixo de 1024.

Desenvolvemos um módulo específico para conexão TCP. O código deste se encontra abaixo:

```
1  import 'dart:io';
2  import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
3  import 'package:drench/features/multiplayer/socket/socket_connection_service.dart';
4
5  class TcpConnection {
6    SocketConnectionService socketConnectionService;
7    ServerSocket tcpServer;
8    Socket tcpClient;
9    Socket tcpRemoteClient;
10
11   TcpConnection({this.socketConnectionService});
12
13   void openConnection(ConnectionParams connectionParams) async {
14     if (connectionParams.isServer) {
15       openTcpServer(connectionParams); return;
16     }
17     connectWithTcpClient(connectionParams);
18   }
19
```

```
20    void openTcpServer(ConnectionParams connectionParams) async {
21      this.tcpServer = await ServerSocket.bind(InternetAddress.anyIPv4, connectionParams.port);
22      this.socketConnectionService.updateConnectionParams(connectionParams);
23      this.tcpServer.listen(handleClientConnectionInTcpServer);
24    }
25
26    handleClientConnectionInTcpServer(Socket client) {
27      print( 'Connection from ${client.remoteAddress.address}:${client.remotePort}', );
28      if (tcpRemoteClient != null) {
29        rejectClientConnection(client); return;
30      }
31      ConnectionParams connectionParams = this.socketConnectionService.getConnectionParams();
32
33      client.write( this.socketConnectionService.getInformationMessage('welcome-to-drench'), );
34      this.tcpRemoteClient = client;
35      this.listenDataReceiving(client);
36      connectionParams.remoteIpAddress = client.remoteAddress.address;
37      connectionParams.remotePort = client.remotePort;
38      this.socketConnectionService.updateConnectionParams(connectionParams);
39    }
40
41    rejectClientConnection(Socket client) {
42      print('Another client connected. Closing connection with
        ↪  ${client.remoteAddress.address}:${client.remotePort}');
43
44      client.write(this.socketConnectionService.getInformationMessage('another-client-connected'));
45      client.close();
46    }
47
48    void connectWithTcpClient(ConnectionParams connectionParams) async {
49      this.tcpClient = await Socket.connect(connectionParams.ipAddress, connectionParams.port);
50      this.listenDataReceiving(this.tcpClient);
51      this.socketConnectionService.updateConnectionParams(connectionParams);
52    }
53
54    void listenDataReceiving(Socket client) {
55      client.listen((event) {
56        print('---- Message from ${client.remoteAddress.address}:${client.remotePort}');
57        var data = new String.fromCharCodes(event).trim();
58        this.socketConnectionService.broadcastMessageReceived(data);
59      });
60    }
61
62    void sendMessage(String data) {
63      Socket client = getActiveTcpClient();
64      if (client == null) {
65        print('Inactive TCP client');
66        print(this.socketConnectionService.getConnectionParams().toJson());
67        this.socketConnectionService.updateConnectionParams(null);
68        return;
69      }
70      client.write(data);
71    }
72
73    Socket getActiveTcpClient() {
74      ConnectionParams connectionParams = this.socketConnectionService.getConnectionParams();
75      if (connectionParams.isServer) {
76        return this.tcpRemoteClient;
77      }
78      return this.tcpClient;
79    }
```

```
80
81    void closeActiveConnections() {
82      if (this.tcpClient != null) {
83        this.tcpClient.destroy();
84        print('destroy tcpClient');
85        this.tcpClient = null;
86      }
87
88      if (this.tcpRemoteClient != null) {
89        this.tcpRemoteClient.destroy();
90        print('destroy tcpRemoteClient');
91        this.tcpRemoteClient = null;
92      }
93
94      if (this.tcpServer != null) {
95        this.tcpServer.close();
96        print('close tcpServer');
97        this.tcpServer = null;
98      }
99    }
100 }
```

2.4.3 UDP

A implementação da comunicação via UDP foi feita com a utilização da API nativa do dart *RawDatagramSocket* [3]. Foi utilizado o tutorial de James Slocum [2] como referência.

Na implementação da comunicação via UDP, não precisamos distinguir entre servidor e cliente, pois a transmissão é realizada através do envio de datagramas em uma porta aberta em ambos os clientes. O método de conexão e envio de mensagens é idêntico ao TCP e o funcionamento do game também se manteve, apenas houve alteração na API utilizada.

2.5 Resultados e análise

Ao longo da descrição do procedimento metodológico foi mostrado os resultados do funcionamento obtido com a implementação realizada. As figuras 2 e 3 exibem as telas utilizadas para conexão com o outro host e execução do jogo em si, respectivamente.

Em ambos os experimentos, com conexão TCP e UDP, os resultados obtidos foram de acordo com o esperado, de forma que as ações realizadas em um host era refletida em seu oponente, o que possibilita uma experiência multiplayer.

A execução do jogo foi demonstrada nesse vídeo: **link para o youtube**

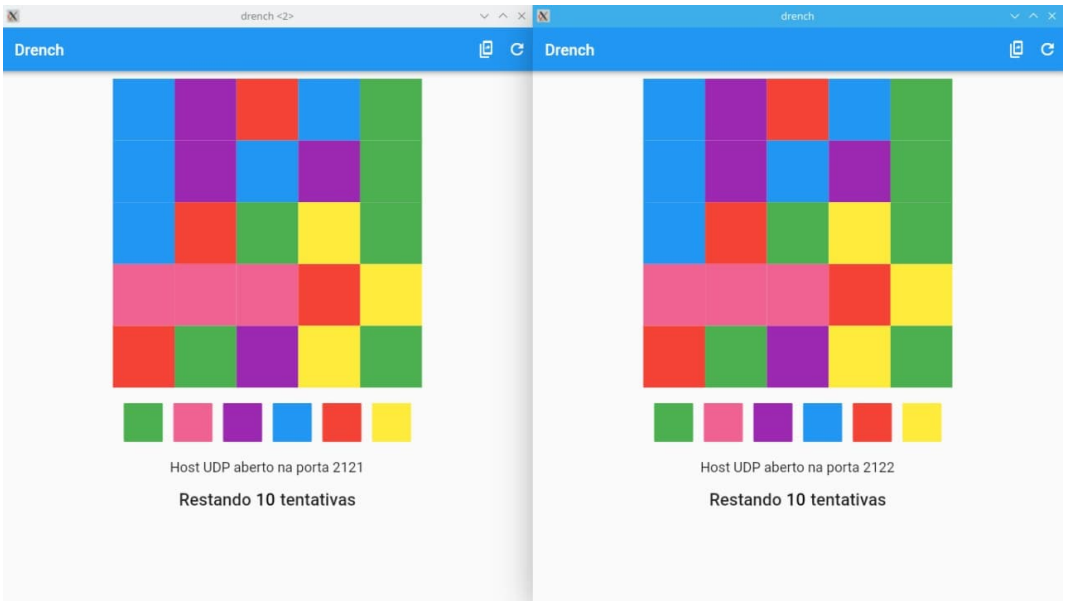

Figura 3: Print de dois simuladores executando via UDP. Fonte: autores.

3 Conclusão

A utilização de sockets permite que uma comunicação entre dois nós seja estabelecida de forma simplória e direta, não havendo a necessidade de criar uma entidade (servidor) centralizador de toda a comunicação como teríamos feito antes de termos o conhecimento de sockets.

Ao utilizar o protocolo TCP, com pouca configuração adicional, temos a confiabilidade do recebimento dos dados que o protocolo proporciona, já que o UDP simplesmente envia os dados na forma de datagrama e não garante que o recebimento ocorreu.

O desenvolvimento desse trabalho proporcionou, além da experiência de implementar um jogo simples e funcional, um entendimento mais profundo e prático sobre os protocolos TCP e UDP, e como podem ser utilizados em aplicações que podem ser comuns ao cotidiano, como esse jogo.

4 Anexos

4.1 Código utilizado

O código mostrado aqui pode ser acessado pelo link do **github**

4.1.1 features/drench_game/widgets/drench_connection_status.dart

```dart
import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
import 'package:flutter/widgets.dart';

class DrenchConnectionStatus extends StatelessWidget {
  final ConnectionParams connectionParams;

  final TextStyle textStyle = TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.w400,
  );

  DrenchConnectionStatus({this.connectionParams});

  @override
  Widget build(BuildContext context) {
    return Container(
      padding: const EdgeInsets.symmetric(vertical: 10),
      child: Center(child: _textsWidgets()),
    );
  }

  _textsWidgets() {
    if (this.connectionParams == null) {
      return _withoutConnection();
    }

    if (this.connectionParams.isTcp && this.connectionParams.isServer) {
      return _tcpServer();
    }

    return _tcpClientOrUpd();
  }

  _withoutConnection() {
    return Text(
      'Sem conexão',
      style: textStyle,
    );
  }

  _tcpServer() {
    return Column(
      children: <Widget>[
        Text(
```

```
45            'Servidor TCP aberto na porta ${this.connectionParams.port}',
46            style: textStyle,
47          ),
48          SizedBox(
49            height: 5,
50          ),
51          _clienteInfoInTcpServer(),
52        ].where((element) => element != null).toList(),
53      );
54    }
55
56    Widget _clienteInfoInTcpServer() {
57      if (this.connectionParams.remoteIpAddress == null) {
58        return null;
59      }
60
61      return Text(
62        'Cliente connectado: ${this.connectionParams.remoteIpAddress}:${this.connectionParams.remotePort}',
63        style: textStyle,
64      );
65    }
66
67    Widget _tcpClientOrUpd() {
68      return Column(
69        children: <Widget>[
70          Text(
71            getTcpClientOrUpdText(),
72            style: textStyle,
73          ),
74        ],
75      );
76    }
77
78    String getTcpClientOrUpdText() {
79      if (connectionParams.isTcp) {
80        return 'Cliente TCP conectado ao servidor
          ↪  ${this.connectionParams.ipAddress}:${this.connectionParams.port}';
81      }
82
83      return 'Host UDP aberto na porta ${this.connectionParams.port}';
84    }
85  }
```

4.1.2 features/drench_game/widgets/drench_control_menu.dart

```
1   import 'package:drench/features/drench_game/drench_game.model.dart';
2   import 'package:drench/features/drench_game/widgets/drench_connection_status.dart';
3   import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
4   import 'package:drench/pages/home_page/components/drench/drench_controller.dart';
5   import 'package:flutter/material.dart';
6   import 'package:flutter/widgets.dart';
7
8   class DrenchControlMenu extends StatelessWidget {
9     final bool gameOver;
10    final DrenchController controller;
11    final double controlMenuSize;
```

```
12    final DrenchGame drenchGame;
13    final ConnectionParams connectionParams;
14
15    DrenchControlMenu({
16      this.gameOver,
17      this.controller,
18      this.controlMenuSize,
19      this.drenchGame,
20      this.connectionParams,
21    });
22
23    @override
24    Widget build(BuildContext context) {
25      return SingleChildScrollView(
26        child: Column(
27          children: [
28            buildBottomMenu(),
29            buildBottomConnectionStatus(),
30            buildBottomStatus(),
31            buildBottomOption(),
32          ],
33        ),
34      );
35    }
36
37    Container buildBottomMenu() {
38      List<Container> buttons = [];
39
40      for (int i = 0; i < 6; i++) {
41        buttons.add(Container(
42          height: controlMenuSize / 8,
43          width: controlMenuSize / 8,
44          color: DrenchGame.getColor(i),
45          child: FlatButton(
46            color: DrenchGame.getColor(i),
47            onPressed: () {
48              this.controller.updateBoard(i);
49              this.controller.sendBoardUpdate(i);
50            },
51            child: SizedBox.shrink(),
52          ),
53        ));
54      }
55
56      return Container(
57        width: controlMenuSize,
58        padding: const EdgeInsets.only(top: 20, bottom: 10),
59        child: Row(
60          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
61          children: <Widget>[...buttons],
62        ),
63      );
64    }
65
66    buildBottomConnectionStatus() {
67      return DrenchConnectionStatus(connectionParams: this.connectionParams);
68    }
69
70    Container buildBottomStatus() {
71      if (gameOver) {
72        return _gameFinished();
```

```
73      }
74
75      return _remaingingPaints();
76    }
77
78    Widget _remaingingPaints() {
79      var remainingPaints = this.drenchGame.remainingPaints;
80
81      TextStyle textStyle = TextStyle(
82        fontSize: 22,
83        fontWeight: FontWeight.w500,
84      );
85
86      return Container(
87        padding: const EdgeInsets.symmetric(vertical: 5),
88        child: Wrap(
89          children: <Widget>[
90            Text('Restando ', style: textStyle),
91            Text(
92              remainingPaints.toString(),
93              style: textStyle.copyWith(
94                color: (remainingPaints > 5) ? Colors.black : Colors.red,
95              ),
96            ),
97            Text(
98              remainingPaints > 1 ? ' tentativas' : ' tentativa',
99              style: textStyle,
100           )
101         ],
102       ),
103     );
104   }
105
106   Widget _gameFinished() {
107     TextStyle textStyle = TextStyle(
108       fontSize: 25,
109       fontWeight: FontWeight.w500,
110       color: Colors.red,
111     );
112
113     return Container(
114       padding: const EdgeInsets.symmetric(vertical: 10),
115       child: Center(
116         child: Text(
117           'O jogo acabou ',
118           style: textStyle,
119         ),
120       ),
121     );
122   }
123
124   Container buildBottomOption() {
125     if (!gameOver) {
126       return Container(
127         child: SizedBox.shrink(),
128       );
129     }
130
131     return Container(
132       padding: EdgeInsets.fromLTRB(0, 10, 0, 15),
133       width: controlMenuSize,
```

```
134         child: FlatButton(
135           color: Colors.green,
136           onPressed: () {
137             this.controller.newGame(true);
138           },
139           child: Padding(
140             padding: const EdgeInsets.symmetric(vertical: 4),
141             child: Text(
142               'Novo Jogo',
143               style: TextStyle(
144                 fontSize: 20,
145                 fontWeight: FontWeight.w500,
146                 color: Colors.white,
147               ),
148             ),
149           ),
150         ),
151       );
152     }
153   }
```

4.1.3 features/drench_game/widgets/drench_matrix.dart

```
1   import 'package:drench/features/drench_game/drench_game.model.dart';
2   import 'package:flutter/widgets.dart';
3
4   class DrenchMatrix extends StatelessWidget {
5     final DrenchGame drenchGame;
6     final double widgetSize;
7
8     DrenchMatrix({this.drenchGame, this.widgetSize});
9
10    @override
11    Widget build(BuildContext context) {
12      List<Widget> result = [];
13
14      for (int i = 0; i < this.drenchGame.size; i++) {
15        result.add(
16          _row(i),
17        );
18      }
19
20      return Column(children: result);
21    }
22
23    _row(i) {
24      List<Widget> auxRow = [];
25
26      for (int j = 0; j < this.drenchGame.size; j++) {
27        auxRow.add(_square(i, j));
28      }
29
30      return Row(
31        crossAxisAlignment: CrossAxisAlignment.center,
32        mainAxisAlignment: MainAxisAlignment.center,
33        children: auxRow,
```

```
    );
  }

  _square(i, j) {
    return Container(
      height: this.widgetSize / this.drenchGame.size,
      width: this.widgetSize / this.drenchGame.size,
      color: DrenchGame.getColor(this.drenchGame.matrix[i][j]),
    );
  }
}
```

4.1.4 features/drench_game/widgets/drench.dart

```
import 'dart:math';
import 'package:drench/features/drench_game/drench_game.model.dart';
import 'package:drench/features/drench_game/widgets/drench_control_menu.dart';
import 'package:drench/features/drench_game/widgets/drench_matrix.dart';
import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
import 'package:drench/pages/home_page/components/drench/drench_controller.dart';
import 'package:flutter/material.dart';

class Drench extends StatefulWidget {
  final DrenchController controller;

  Drench({Key key, this.controller});

  @override
  _DrenchState createState() => _DrenchState(controller: controller);
}

class _DrenchState extends State<Drench> {
  final DrenchController controller;

  final double topWidgetHeight = 10;
  double get bottomWidgetHeight => min(
        0.5 * MediaQuery.of(context).size.height,
        275,
      );

  double get maxDrenchBoardHeight => max(
        MediaQuery.of(context).size.height -
            topWidgetHeight -
            bottomWidgetHeight -
            56,
        0,
      );

  double get drenchBoardSize => min(
        MediaQuery.of(context).size.width - 10,
        maxDrenchBoardHeight,
      );

  double get controlMenuSize => min(
        MediaQuery.of(context).size.width,
        max(
```

```
          drenchBoardSize,
          300,
        ),
      );

  DrenchGame drenchGame;
  List<Color> colors = DrenchGame.colors;

  ConnectionParams connectionParams;

  bool gameOver = false;

  _DrenchState({this.controller}) {
    controller.newGame = newGame;
    controller.updateBoard = updateBoard;
    controller.syncBoard = syncBoard;
    controller.setConnectionParams = setConnectionParams;

    this.setDrenchGame();
  }

  setDrenchGame() {
    this.drenchGame = DrenchGame(maxClicks: 10, size: 5);
  }

  void newGame(bool syncBoard) {
    setState(() {
      this.setDrenchGame();
      gameOver = false;
    });

    if (syncBoard && this.connectionParams != null) {
      this.controller.sendBoardSync(this.drenchGame.matrix, true);
    }
  }

  void updateBoard(int value) {
    if (gameOver == true) {
      return;
    }

    this.drenchGame.paintFirstSquare(value);

    if (this.drenchGame.isGameOver()) {
      gameOver = true;
    }

    setState(() {});
  }

  void syncBoard(List<List<int>> board) {
    setState(() {
      this.drenchGame.matrix = board;
    });
  }

  void setConnectionParams(ConnectionParams connectionParams) {
    setState(() {
      this.connectionParams = connectionParams;
    });
  }
```

```
104
105   @override
106   Widget build(BuildContext context) {
107     return SingleChildScrollView(
108       child: Column(
109         children: <Widget>[
110           _topWidget(),
111           DrenchMatrix(
112             drenchGame: this.drenchGame,
113             widgetSize: drenchBoardSize,
114           ),
115           _bottomWidget(),
116         ],
117       ),
118     );
119   }
120
121   Widget _topWidget() {
122     return ConstrainedBox(
123       constraints: BoxConstraints(
124         maxHeight: max(topWidgetHeight, 0),
125       ),
126       child: SizedBox(
127         height: max(topWidgetHeight, 0),
128       ),
129     );
130   }
131
132   Widget _bottomWidget() {
133     return ConstrainedBox(
134       constraints: BoxConstraints(
135         maxHeight: bottomWidgetHeight,
136       ),
137       child: DrenchControlMenu(
138         gameOver: this.gameOver,
139         controller: this.controller,
140         controlMenuSize: this.controlMenuSize,
141         drenchGame: this.drenchGame,
142         connectionParams: this.connectionParams,
143       ),
144     );
145   }
146 }
```

4.1.5 features/drench_game/drench_game.model.dart

```
1  import 'dart:math';
2
3  import 'package:flutter/material.dart';
4
5  class DrenchGame {
6    static final List<Color> colors = [
7      Colors.green,
8      Colors.pink[300],
9      Colors.purple,
10     Colors.blue,
```

```
11      Colors.red,
12      Colors.yellow,
13    ];
14
15    final int maxClicks;
16    final int size;
17
18    int _paintsCount;
19    List<List<int>> _matrix;
20
21    DrenchGame({@required this.maxClicks, @required this.size}) {
22      _paintsCount = 0;
23
24      _matrix = List.generate(size, (i) => List(size), growable: false);
25
26      var rng = new Random();
27      for (int i = 0; i < size; i++) {
28        for (int j = 0; j < size; j++) {
29          _matrix[i][j] = rng.nextInt(100) % 6;
30        }
31      }
32    }
33
34    List<List<int>> get matrix => this._matrix;
35    set matrix (List<List<int>> matrix) => this._matrix = matrix;
36
37    int get remainingPaints => this.maxClicks - this._paintsCount;
38
39    bool isGameOver() {
40      if (_paintsCount >= maxClicks) {
41        return true;
42      }
43
44      int val = _matrix[0][0];
45      int cont = 0;
46
47      for (int i = 0; i < size; i++) {
48        for (int j = 0; j < size; j++) {
49          if (_matrix[i][j] != val) {
50            return false;
51          }
52
53          cont++;
54        }
55      }
56
57      return cont == (size * size);
58    }
59
60    void paintFirstSquare(int colorIndex) {
61      if (colorIndex == _matrix[0][0]) {
62        return;
63      }
64
65      _paintsCount++;
66
67      this.propagateColorInMatrix(colorIndex, _matrix[0][0]);
68    }
69
70    void propagateColorInMatrix(int newValue, int oldValue, [x = 0, y = 0]) {
71      if (x >= size || y >= size || x < 0 || y < 0) {
```

```
72        return;
73      }
74
75      if (_matrix[x][y] != oldValue && (x != 0 || y != 0)) {
76        return;
77      }
78
79      _matrix[x][y] = newValue;
80
81      propagateColorInMatrix(newValue, oldValue, x, y + 1);
82      propagateColorInMatrix(newValue, oldValue, x, y - 1);
83      propagateColorInMatrix(newValue, oldValue, x + 1, y);
84      propagateColorInMatrix(newValue, oldValue, x - 1, y);
85    }
86
87    static Color getColor(int i) {
88      return colors[i];
89    }
90  }
```

4.1.6 features/multiplayer/components/connection_dialog/connection_dialog_form.dart

```
1   import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
2   import 'package:flutter/material.dart';
3   import 'package:flutter/services.dart';
4   import 'package:flutter/widgets.dart';
5
6   class ConnectionDialogForm extends StatefulWidget {
7     ConnectionDialogForm() {}
8
9     @override
10    _ConnectionDialogFormState createState() => _ConnectionDialogFormState();
11  }
12
13  class _ConnectionDialogFormState extends State<ConnectionDialogForm> {
14    bool _isTcp = true;
15    bool _isServer = false;
16    TextEditingController _ipAddressFieldController;
17    TextEditingController _portFieldController;
18    TextEditingController _remotePortFieldController;
19
20    _ConnectionDialogFormState() {
21      this._ipAddressFieldController = TextEditingController(text: '127.0.0.1');
22      this._portFieldController = TextEditingController(text: '2121');
23      this._remotePortFieldController = TextEditingController(text: '2122');
24    }
25
26    @override
27    Widget build(BuildContext context) {
28      return SingleChildScrollView(
29        child: Column(
30          mainAxisSize: MainAxisSize.min,
31          crossAxisAlignment: CrossAxisAlignment.start,
32          children: _getFields(),
33        ),
34      );
```

```
35    }
36
37    List<Widget> _getFields() {
38      List<Widget> list = [
39        _isTcpField(),
40        hasIsServerField() ? _isServerField() : null,
41        SizedBox(height: 20),
42        hasIpAddressField() ? _ipAddressField() : null,
43        _portField(),
44        hasRemotePortField() ? _remotePortField() : null,
45        SizedBox(height: 40),
46        _submitButton()
47      ];
48
49      return list.where((field) => field != null).toList();
50    }
51
52    Widget _isServerField() {
53      return SwitchListTile(
54        title: const Text('Servidor'),
55        value: _isServer,
56        onChanged: (bool value) {
57          setState(() {
58            _isServer = value;
59          });
60        },
61      );
62    }
63
64    Widget _isTcpField() {
65      return SwitchListTile(
66        title: const Text('Usar TCP'),
67        value: _isTcp,
68        onChanged: (bool value) {
69          setState(() {
70            _isTcp = value;
71          });
72        },
73      );
74    }
75
76    Widget _ipAddressField() {
77      return TextField(
78        controller: _ipAddressFieldController,
79        autofocus: true,
80        textInputAction: TextInputAction.next,
81        decoration: InputDecoration(
82          labelText: getIpAddressFieldLabel(),
83        ),
84      );
85    }
86
87    getIpAddressFieldLabel() {
88      if (!_isTcp || !_isServer) {
89        return 'Endereço Ip remoto';
90      }
91
92      return 'Endereço Ip';
93    }
94
95    Widget _portField() {
```

```
96      return TextField(
97        controller: _portFieldController,
98        textInputAction:
99            hasRemotePortField() ? TextInputAction.next : TextInputAction.go,
100       onSubmitted: (_) => hasRemotePortField() ? null : _initConnection(),
101       onChanged: (String port) {
102         setState(() {});
103       },
104       keyboardType: TextInputType.number,
105       inputFormatters: [
106         FilteringTextInputFormatter.digitsOnly,
107       ],
108       decoration: InputDecoration(
109         labelText: 'Porta',
110         errorText:
111             isValidPort() ? null : 'Porta inválida. Precisa ser maior que 1024',
112       ),
113     );
114   }
115
116   Widget _remotePortField() {
117     return TextField(
118       controller: _remotePortFieldController,
119       textInputAction: TextInputAction.go,
120       onSubmitted: (_) => _initConnection(),
121       onChanged: (String port) {
122         setState(() {});
123       },
124       keyboardType: TextInputType.number,
125       inputFormatters: [
126         FilteringTextInputFormatter.digitsOnly,
127       ],
128       decoration: InputDecoration(
129         labelText: 'Porta remota',
130         errorText: isValidRemotePort()
131             ? null
132             : 'Porta inválida. Precisa ser maior que 1024',
133       ),
134     );
135   }
136
137   Widget _submitButton() {
138     return SizedBox(
139       width: double.infinity,
140       child: RaisedButton(
141         color: Theme.of(context).primaryColor,
142         textColor: Colors.white,
143         child: Padding(
144           padding: const EdgeInsets.all(12),
145           child: Text(
146             _submitButtonText(),
147             textAlign: TextAlign.center,
148             style: TextStyle(
149               fontSize: 18,
150               fontWeight: FontWeight.w400,
151             ),
152           ),
153         ),
154         onPressed:
155             isValidPort() && (hasRemotePortField() ? isValidRemotePort() : true)
156                 ? _initConnection
```

```
157                 : null,
158       ),
159     );
160   }
161
162   String _submitButtonText() {
163     if (!this._isTcp) {
164       return 'Abrir porta UDP';
165     }
166
167     if (this._isServer) {
168       return 'Iniciar servidor TCP';
169     }
170
171     return 'Estabelecer conexão TCP';
172   }
173
174   _initConnection() {
175     ConnectionParams params = this.getValues();
176     Navigator.of(this.context).pop(params);
177   }
178
179   ConnectionParams getValues() {
180     return ConnectionParams(
181       isTcp: _isTcp,
182       isServer: hasIsServerField() ? _isServer : null,
183       ipAddress:
184           hasIpAddressField() ? this._ipAddressFieldController.text : null,
185       port: int.parse(this._portFieldController.text),
186       remotePort: int.parse(this._remotePortFieldController.text),
187     );
188   }
189
190   bool isValidPort() {
191     String port = _portFieldController.text;
192
193     if (port.isNotEmpty && int.parse(port) <= 1024) {
194       return false;
195     }
196
197     return true;
198   }
199
200   bool isValidRemotePort() {
201     String remotePort = _remotePortFieldController.text;
202
203     if (remotePort.isNotEmpty && int.parse(remotePort) <= 1024) {
204       return false;
205     }
206
207     return true;
208   }
209
210   bool hasIsServerField() {
211     return _isTcp;
212   }
213
214   bool hasRemotePortField() {
215     return !_isTcp;
216   }
217
```

```
218   bool hasIpAddressField() {
219     return !_isTcp || _isTcp && !_isServer;
220   }
221 }
```

4.1.7 features/multiplayer/components/connection_dialog/connection_dialog_service.dart

```
1  import 'package:drench/features/multiplayer/components/connection_dialog/connection_dialog_form.dart';
2  import 'package:flutter/material.dart';
3
4  class ConnectionDialogService {
5    Future show(BuildContext context) {
6      return showDialog(
7        context: context,
8        builder: (BuildContext context) => _alertDialog(context),
9      );
10   }
11
12   AlertDialog _alertDialog(BuildContext context) {
13     return new AlertDialog(
14       title: Text('Conectar-se a outro dispositivo'),
15       contentPadding: EdgeInsets.fromLTRB(16, 20, 16, 16),
16       content: _content(),
17     );
18   }
19
20   Widget _content() {
21     return ConnectionDialogForm();
22   }
23 }
```

4.1.8 features/multiplayer/socket/tcp/tcp_connection.dart

```
1  import 'dart:io';
2
3  import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
4  import 'package:drench/features/multiplayer/socket/socket_connection_service.dart';
5
6  class TcpConnection {
7    SocketConnectionService socketConnectionService;
8
9    ServerSocket tcpServer;
10   Socket tcpClient;
11   Socket tcpRemoteClient;
12
13   TcpConnection({this.socketConnectionService});
14
15   void openConnection(ConnectionParams connectionParams) async {
16     if (connectionParams.isServer) {
17       openTcpServer(connectionParams);
18       return;
19     }
```

```
20
21      connectWithTcpClient(connectionParams);
22    }
23
24    void openTcpServer(ConnectionParams connectionParams) async {
25      this.tcpServer =
26          await ServerSocket.bind(InternetAddress.anyIPv4, connectionParams.port);
27
28      this.socketConnectionService.updateConnectionParams(connectionParams);
29      this.tcpServer.listen(handleClientConnectionInTcpServer);
30    }
31
32    handleClientConnectionInTcpServer(Socket client) {
33      print(
34        'Connection from '
35        '${client.remoteAddress.address}:${client.remotePort}',
36      );
37
38      if (tcpRemoteClient != null) {
39        rejectClientConnection(client);
40        return;
41      }
42
43      ConnectionParams connectionParams =
44          this.socketConnectionService.getConnectionParams();
45
46      client.write(
47        this.socketConnectionService.getInformationMessage('welcome-to-drench'),
48      );
49
50      this.tcpRemoteClient = client;
51      this.listenDataReceiving(client);
52
53      connectionParams.remoteIpAddress = client.remoteAddress.address;
54      connectionParams.remotePort = client.remotePort;
55
56      this.socketConnectionService.updateConnectionParams(connectionParams);
57    }
58
59    rejectClientConnection(Socket client) {
60      print(
61        'Another client connected. Closing connection with '
62        '${client.remoteAddress.address}:${client.remotePort}',
63      );
64
65      client.write(this
66          .socketConnectionService
67          .getInformationMessage('another-client-connected'));
68      client.close();
69    }
70
71    void connectWithTcpClient(ConnectionParams connectionParams) async {
72      this.tcpClient = await Socket.connect(
73        connectionParams.ipAddress,
74        connectionParams.port,
75      );
76      this.listenDataReceiving(this.tcpClient);
77
78      this.socketConnectionService.updateConnectionParams(connectionParams);
79    }
80
```

```
81    void listenDataReceiving(Socket client) {
82      client.listen((event) {
83        print(
84          '---- Message from ${client.remoteAddress.address}:${client.remotePort}',
85        );
86
87        var data = new String.fromCharCodes(event).trim();
88
89        this.socketConnectionService.broadcastMessageReceived(data);
90      });
91    }
92
93    void sendMessage(String data) {
94      Socket client = getActiveTcpClient();
95
96      if (client == null) {
97        print('Inactive TCP client');
98        print(this.socketConnectionService.getConnectionParams().toJson());
99
100       this.socketConnectionService.updateConnectionParams(null);
101       return;
102     }
103
104     client.write(data);
105   }
106
107   Socket getActiveTcpClient() {
108     ConnectionParams connectionParams =
109         this.socketConnectionService.getConnectionParams();
110
111     if (connectionParams.isServer) {
112       return this.tcpRemoteClient;
113     }
114
115     return this.tcpClient;
116   }
117
118   void closeActiveConnections() {
119     if (this.tcpClient != null) {
120       this.tcpClient.destroy();
121       print('destroy tcpClient');
122       this.tcpClient = null;
123     }
124
125     if (this.tcpRemoteClient != null) {
126       this.tcpRemoteClient.destroy();
127       print('destroy tcpRemoteClient');
128       this.tcpRemoteClient = null;
129     }
130
131     if (this.tcpServer != null) {
132       this.tcpServer.close();
133       print('close tcpServer');
134       this.tcpServer = null;
135     }
136   }
137 }
```

4.1.9 features/multiplayer/socket/udp/udp_connection.dart

```dart
import 'dart:io';

import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
import 'package:drench/features/multiplayer/socket/socket_connection_service.dart';

class UdpConnection {
  SocketConnectionService socketConnectionService;

  RawDatagramSocket udpServer;

  UdpConnection({this.socketConnectionService});

  void openConnection(ConnectionParams connectionParams) async {
    openUdpServer(connectionParams);
  }

  void openUdpServer(ConnectionParams connectionParams) async {
    this.udpServer = await RawDatagramSocket.bind(
        InternetAddress.anyIPv4, connectionParams.port);

    this.socketConnectionService.updateConnectionParams(connectionParams);

    this.listenDataReceiving();
  }

  void listenDataReceiving() {
    print('listen');
    print(this.udpServer);

    print(
        {'addr': this.udpServer.address.address, 'port': this.udpServer.port});

    this.udpServer.listen((RawSocketEvent event) {
      Datagram datagram = this.udpServer.receive();

      if (datagram == null) {
        return;
      }

      print(
        '---- Message from ${datagram.address.address}:${datagram.port}',
      );

      String message = new String.fromCharCodes(datagram.data).trim();

      this.socketConnectionService.broadcastMessageReceived(message);
    });
  }

  void sendMessage(String data) async {
    ConnectionParams connectionParams =
        this.socketConnectionService.getConnectionParams();

    List<InternetAddress> addresses = await InternetAddress.lookup(
        connectionParams.ipAddress,
        type: InternetAddressType.IPv4);

```

```
58      print(addresses);
59
60      this
61          .udpServer
62          .send(data.codeUnits, addresses[0], connectionParams.remotePort);
63    }
64
65    void closeActiveConnections() {
66      if (this.udpServer != null) {
67        this.udpServer.close();
68        print('close udpServer');
69        this.udpServer = null;
70      }
71    }
72  }
```

4.1.10 features/multiplayer/socket/connection_params.model.dart

```
1   class ConnectionParams {
2     bool isTcp;
3     bool isServer;
4     String ipAddress;
5     int port;
6     String remoteIpAddress;
7     int remotePort;
8
9     ConnectionParams(
10        {this.isTcp,
11        this.isServer,
12        this.ipAddress,
13        this.port,
14        this.remoteIpAddress,
15        this.remotePort});
16
17    Map<String, dynamic> toJson() => {
18          'isTcp': isTcp,
19          'isServer': isServer,
20          'ipAddress': ipAddress,
21          'port': port,
22          'remoteIpAddress': remoteIpAddress,
23          'remotePort': remotePort,
24        };
25  }
```

4.1.11 features/multiplayer/socket_connection_service.dart

```
1   import 'dart:convert';
2   import 'dart:io';
3
4   import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
5   import 'package:drench/features/multiplayer/socket/tcp/tcp_connection.dart';
6   import 'package:drench/features/multiplayer/socket/udp/udp_connection.dart';
```

```dart
import 'package:rxdart/subjects.dart';

class SocketConnectionService {
  BehaviorSubject<ConnectionParams> currentConnectionParams$ =
      BehaviorSubject<ConnectionParams>();

  ReplaySubject<Map<String, dynamic>> dataReceiving$ =
      ReplaySubject<Map<String, dynamic>>();

  TcpConnection _tcp;
  UdpConnection _udp;

  SocketConnectionService() {
    _tcp = TcpConnection(socketConnectionService: this);
    _udp = UdpConnection(socketConnectionService: this);
  }

  void connect(ConnectionParams connectionParams) {
    closeActiveConnections();

    if (connectionParams.isTcp) {
      this._tcp.openConnection(connectionParams);
      return;
    }

    this._udp.openConnection(connectionParams);
  }

  void sendData(Map<String, dynamic> data) {
    ConnectionParams connectionParams = getConnectionParams();

    if (connectionParams == null) {
      print("There's no active connection");
      return;
    }

    if (connectionParams.isTcp) {
      this._tcp.sendMessage(json.encode(data));
      return;
    }

    this._udp.sendMessage(json.encode(data));
  }

  void updateConnectionParams(ConnectionParams connectionParams) {
    if (connectionParams == null) {
      this.currentConnectionParams$.add(null);
      return;
    }

    ConnectionParams newObject = ConnectionParams(
        isTcp: connectionParams.isTcp,
        isServer: connectionParams.isServer,
        ipAddress: connectionParams.ipAddress,
        port: connectionParams.port,
        remoteIpAddress: connectionParams.remoteIpAddress,
        remotePort: connectionParams.remotePort);

    this.currentConnectionParams$.add(newObject);
  }

```

```
68    void broadcastMessageReceived(dynamic data) {
69      try {
70        dataReceiving$.add(json.decode(data));
71      } catch (e) {
72        print('decode error:');
73        print(e);
74      }
75    }
76
77    void closeActiveConnections() {
78      this._tcp.closeActiveConnections();
79      this._udp.closeActiveConnections();
80    }
81
82    getInformationMessage(String message) {
83      return json.encode({'type': 'information', 'message': message});
84    }
85
86    ConnectionParams getConnectionParams() {
87      return this.currentConnectionParams$.value;
88    }
89  }
```

4.1.12 pages/home_page/components/drench/drench_controller.dart

```
1   import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
2   import 'package:drench/features/multiplayer/socket/socket_connection_service.dart';
3
4   class DrenchController {
5     void Function(bool newGame) newGame;
6     void Function(int colorIndex) updateBoard;
7     void Function(List<List<int>> colorIndex) syncBoard;
8     void Function(ConnectionParams connectionParams) setConnectionParams;
9
10    SocketConnectionService _socketConnectionService;
11
12    setSocketConnectionService(SocketConnectionService socketConnectionService) {
13      this._socketConnectionService = socketConnectionService;
14
15      socketConnectionService.currentConnectionParams$
16          .listen(handleChangeConnectionParams);
17
18      socketConnectionService.dataReceiving$.listen(handleSocketData);
19    }
20
21    handleChangeConnectionParams(ConnectionParams connectionParams) {
22      this.setConnectionParams(connectionParams);
23    }
24
25    void handleSocketData(value) {
26      print('===== data received');
27      print(value);
28
29      if (value['type'] == 'updateBoard') {
30        this.updateBoard(value['colorIndex']);
31        return;
```

```
32      }
33
34      if (value['type'] == 'syncBoard') {
35        List<List<int>> board = new List<List<int>>();
36
37        value['board'].forEach((vector) {
38          List<int> list = new List<int>();
39
40          vector.forEach((value) {
41            list.add(value as int);
42          });
43
44          board.add(list);
45        });
46
47        if (value['reset'] == true) {
48          this.newGame(false);
49        }
50
51        this.syncBoard(board);
52        return;
53      }
54    }
55
56    sendBoardSync(List<List<int>> board, bool reset) {
57      this
58          ._socketConnectionService
59          .sendData({'type': 'syncBoard', 'board': board, 'reset': reset});
60    }
61
62    sendBoardUpdate(int colorIndex) {
63      this._socketConnectionService.sendData({
64        'type': 'updateBoard',
65        'colorIndex': colorIndex,
66      });
67    }
68  }
```

4.1.13 pages/home_page/home_page.dart

```
1   import 'package:drench/features/multiplayer/components/connection_dialog/connection_dialog_service.dart';
2   import 'package:drench/features/multiplayer/socket/connection_params.model.dart';
3   import 'package:drench/features/multiplayer/socket/socket_connection_service.dart';
4   import 'package:drench/features/drench_game/widgets/drench.dart';
5   import 'package:drench/pages/home_page/components/drench/drench_controller.dart';
6   import 'package:flutter/material.dart';
7   import 'package:flutter/widgets.dart';
8
9   class HomePage extends StatefulWidget {
10    @override
11    _HomePageState createState() => _HomePageState();
12  }
13
14  class _HomePageState extends State<HomePage> {
15    final ConnectionDialogService _connectionDialogService =
16        ConnectionDialogService();
```

```
17
18    final SocketConnectionService _socketConnectionService =
19        SocketConnectionService();
20
21    final DrenchController drenchController = DrenchController();
22
23    _HomePageState() {
24      this.drenchController.setSocketConnectionService(_socketConnectionService);
25    }
26
27    @override
28    Widget build(BuildContext context) {
29      return Container(
30        child: Scaffold(
31          appBar: _appBar(),
32          body: _body(),
33        ),
34      );
35    }
36
37    Widget _appBar() {
38      return AppBar(
39        title: Text("Drench"),
40        actions: _appBarActions(),
41      );
42    }
43
44    List<Widget> _appBarActions() {
45      return [
46        _connectToDeviceActionButton(),
47        _newGameActionButton(),
48      ];
49    }
50
51    IconButton _connectToDeviceActionButton() {
52      return IconButton(
53        icon: Icon(
54          Icons.offline_share,
55          color: Colors.white,
56        ),
57        onPressed: this.showConnectionDialog,
58      );
59    }
60
61    IconButton _newGameActionButton() {
62      return IconButton(
63        icon: Icon(
64          Icons.refresh,
65          color: Colors.white,
66        ),
67        onPressed: () {
68          this.drenchController.newGame(true);
69        },
70      );
71    }
72
73    Widget _body() {
74      return Drench(controller: drenchController);
75    }
76
77    void showConnectionDialog() async {
```

```
78      ConnectionParams connectionParams =
79          await _connectionDialogService.show(this.context);
80
81      if (connectionParams == null) {
82        return;
83      }
84
85      this._socketConnectionService.connect(connectionParams);
86    }
87  }
```

4.1.14 main.dart

```
1   import 'package:drench/pages/home_page/home_page.dart';
2   import 'package:flutter/material.dart';
3
4   void main() {
5     runApp(MyApp());
6   }
7
8   class MyApp extends StatelessWidget {
9     @override
10    Widget build(BuildContext context) {
11      return MaterialApp(
12        title: 'Drench ',
13        debugShowCheckedModeBanner: false,
14        theme: ThemeData(
15          primarySwatch: Colors.blue,
16          visualDensity: VisualDensity.adaptivePlatformDensity,
17          // platform: TargetPlatform.iOS,
18        ),
19        home: HomePage(),
20      );
21    }
22  }
```

Referências

[1] James Slocum - network socket programming with dart 1.0, . URL https://jamesslocum.com/blog/post/67566023889.

[2] James Slocum - udp socket programming with dart (unicast and multicast), . URL https://jamesslocum.com/blog/post/77759061182.

[3] RawDatagramSocket class - dart:io library - Dart API, . URL https://api.flutter.dev/flutter/dart-io/RawDatagramSocket-class.html.

[4] Socket class - dart:io library - Dart API, . URL https://api.dart.dev/stable/2.10.5/dart-io/Socket-class.html.