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 02:
Invocação remota

1 Introdução

Nesse relatório, descreveremos como implementamos e como utilizamos o software que desenvolvemos para o trabalho de *invocação remota* 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.

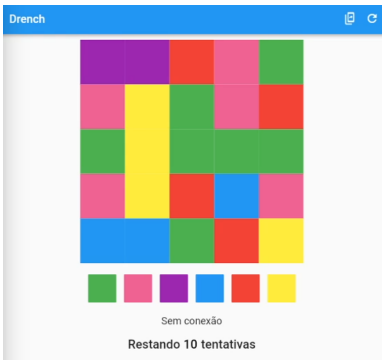

Figura 1: Print do jogo. Fonte: autores.

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 invocação remota

A comunicação entre os dois jogadores é feita utilizando *gRPC*. O *gRPC* é um framework open source de invocação remota de alto desempenho. Ele possui suporte para várias linguagens, incluindo Dart, a linguagem utilizada no trabalho.

No *gRPC*, existe um processo executando como servidor e um como stub (em outros modelos também conhecido como cliente). O servidor implementa um serviço, com todos os métodos que o cliente pode invocar. O cliente por sua vez, pode invocar os métodos oferecidos pelo servidor. Para a comunicação entre os dois processos, o *gRPC* utiliza um protocolo chamado *Protocol Buffers*[2], desenvolvido pela Google. É um protocolo neutro a linguagem e é extensível à serialização de dados estruturados [1].

2.4 Funcionamento

Abaixo, descreveremos o funcionamento geral do aplicativo. Para ver o código inteiro, vá na seção de Anexos (4) ou acesse o repositório públic no **github**

2.4.1 Funcionamento geral

Para ter o máximo de aproveitamento do código desenvolvido no trabalho anterior, focamos em separar os módulos do jogo e da comunicação, de forma que foi possível adaptar as trocas de mensagens para serem feitas via gRPC. A seguir estão resumidas as explicações sobre os componentes responsáveis pelo funcionamento do modo *multiplayer*.

O arquivo `drench_multiplayer_connection_service.dart` gerencia a interação entre o controlador do jogo e o outro *host*. Esse arquivo possui conhecimento de ambas as comunicações implementadas no projeto: A comunicação via *sockets*, desenvolvida no trabalho a anterior, e a comunicação via *gRPC*.

```
...
class DrenchMultiplayerConnectionService {
  BehaviorSubject<ConnectionParams> currentConnectionParams$ = BehaviorSubject<ConnectionParams>();
  ReplaySubject<int> updateBoard$ = ReplaySubject<int>();
  ReplaySubject<Map<String, dynamic>> syncBoard$ = ReplaySubject<Map<String, dynamic>>();
  SocketConnectionService _socketConnectionService;
  GRPCConnectionService _gRPCConnectionService;

  DrenchMultiplayerConnectionService() {
    this.createSocketService();
    this.createGRPCService();
  }

  createSocketService() {
    final socketService = SocketConnectionService();
    this._socketConnectionService = socketService;

    socketService.currentConnectionParams$.listen((params) => this.currentConnectionParams$.add(params));

    socketService.dataReceiving$.listen(handleSocketData);
  }

  createGRPCService() {
    final gRPCService = GRPCConnectionService();
    this._gRPCConnectionService = gRPCService;
  }

  void connect(ConnectionParams connectionParams) {
    closeActiveConnections();

    if (connectionParams.isSocket) {
```

```
32        this._socketConnectionService.connect(connectionParams);
33        return;
34      }
35
36      this.currentConnectionParams$.add(connectionParams);
37      this._gRPCConnectionService.connect(connectionParams, this);
38    }
39
40    closeActiveConnections() {
41      this._socketConnectionService.closeActiveConnections();
42      this._gRPCConnectionService.closeActiveConnections();
43    }
44
45    void handleSocketData(value) {
46      if (value['type'] == 'updateBoard') {
47        updateBoard(value['colorIndex']);
48        return;
49      }
50
51      if (value['type'] == 'syncBoard') {
52        syncBoard(value);
53        return;
54      }
55    }
56
57    updateBoard(int colorIndex) {
58      this.updateBoard$.add(colorIndex);
59    }
60
61    syncBoard(value) {
62      this.syncBoard$.add(value);
63    }
64
65    sendBoardSync(List<List<int>> board, bool reset) {
66      final connectionParams = this.currentConnectionParams$.value;
67
68      if (!connectionParams.isSocket) {
69        this._gRPCConnectionService.sendBoardSync(board, reset);
70        return;
71      }
72
73      this._socketConnectionService.sendData({'type': 'syncBoard', 'board': board, 'reset': reset});
74    }
75
76    sendBoardUpdate(int colorIndex) {
77      final connectionParams = this.currentConnectionParams$.value;
78
79      if (!connectionParams.isSocket) {
80        this._gRPCConnectionService.sendBoardUpdate(colorIndex);
81        return;
82      }
83
84      this._socketConnectionService.sendData({
85        'type': 'updateBoard',
86        'colorIndex': colorIndex,
87      });
88    }
89  }
```

Enquanto isso, o arquivo `drench_controller.dart` interage diretamente com o arquivo citado anteriormente, além de controlar a exibição da interface e receber comandos do jogo que devem ser enviados ao outro *host*.

```
...
class DrenchController {
  void Function(bool newGame) newGame;
  void Function(int colorIndex) updateBoard;
  void Function(List<List<int>> colorIndex) syncBoard;
  void Function(ConnectionParams connectionParams) setConnectionParams;

  DrenchMultiplayerConnectionService _drenchMultiplayerConnectionService;

  setMultiplayerConnectionService(
      DrenchMultiplayerConnectionService multiplayerConnectionService) {
    this._drenchMultiplayerConnectionService = multiplayerConnectionService;

    _drenchMultiplayerConnectionService.currentConnectionParams$
        .listen(handleChangeConnectionParams);

    _drenchMultiplayerConnectionService.updateBoard$.listen(handleUpdateBoard);
    _drenchMultiplayerConnectionService.syncBoard$.listen(handleSyncBoard);
  }

  handleChangeConnectionParams(ConnectionParams connectionParams) {
    this.setConnectionParams(connectionParams);
  }

  handleUpdateBoard(int colorIndex) {
    this.updateBoard(colorIndex);
  }

  handleSyncBoard(Map<String, dynamic> args) {
    List<List<int>> board = new List<List<int>>();

    args['board'].forEach((vector) {
      List<int> list = new List<int>();

      vector.forEach((value) {
        list.add(value as int);
      });

      board.add(list);
    });

    if (args['reset'] == true) {
      this.newGame(false);
    }

    this.syncBoard(board);
  }

  sendBoardSync(List<List<int>> board, bool reset) {
    this._drenchMultiplayerConnectionService.sendBoardSync(board, reset);
  }

  sendBoardUpdate(int colorIndex) {
    this._drenchMultiplayerConnectionService.sendBoardUpdate(colorIndex);
```

```
55    }
56  }
```

Para definir os serviços utilizados pelo *gRPC*, utilizamos um arquivo **.proto**. Nesse arquivo, definimos os métodos que serão invocados, assim como as estruturas que serão passadas e retornadas por estes. No nosso caso, definimos o seguinte arquivo:

```
syntax = "proto3";
package drenchPackage;

service Drench {
    rpc syncBoard(SyncBoardData) returns (voidNoParam);
    rpc updateBoard(UpdateBoardData) returns (voidNoParam);

    rpc subscribeSyncBoard(voidNoParam) returns (stream SyncBoardData);
    rpc subscribeUpdateBoard(voidNoParam) returns (stream UpdateBoardData);
}

message voidNoParam {}

message SyncBoardData {
    string board = 1;
    bool reset = 2;
}

message UpdateBoardData {
    int32 colorIndex = 1;
}
```

Esse arquivo é utilizado na execução do seguinte comando:

```
protoc -I protos/ protos/drench.proto --dart_out=grpc:lib/generated
```

Após sua execução, serão gerados quatro arquivos *boilerplate* dentro da pasta **lib/generated**, que implementam imperativamente a invocação dos métodos definidos no arquivo **drench.proto**

A conexão entre dois dispositivos é feita no modelo cliente-servidor, onde o primeiro *host* precisa abrir a conexão, e o segundo conectar-se ao primeiro na porta especificada.

O arquivo **.proto** mostrado acima define o serviço de comunicação da nossa aplicação com quatro métodos a serem invocados:

- **syncBoard**: Método invocado pelo cliente para sincronizar o tabuleiro com o servidor – quando um dispositivo inicia ou reinicia o jogo, ele deve transmitir seu tabuleiro para o segundo dispositivo, para que ambos tenham o mesmo tabuleiro.

- **updateBoard**: Método utilizado pelo cliente para atualizar o tabuleiro do servidor com a cor selecionada.
- **subscribeSyncBoard** e **subscribeUpdateBoard**: Métodos também invocados pelo cliente, mas com o objetivo de escutar atualizações vindas do servidor, no formato de *stream*. Os métodos, respectivamente, notificam o cliente quando o servidor envia uma cópia do seu tabuleiro para sincronização e quando o servidor realiza uma jogada.

Como visto anteriormente, a invocação de métodos é feita do cliente para o servidor. Isso torna trivial a implementação dos métodos **syncBoard** e **updateBoard**, que consiste em apenas realizar as devidas ações no controlador do jogo que executa no servidor.

No entanto, para realizar a comunicação reversa, ou seja, enviar dados do servidor para o cliente, é necessário utilizar o recurso de *streams*, oferecido pelo *gRPC*. Para isso, ao estabelecer a conexão, o cliente deve invocar os métodos **subscribeSyncBoard** e **subscribeUpdateBoard**, que retornam um objeto *Stream*. Esse objeto receberá um novo dado sempre que o servidor enviar algum comando do jogo para o cliente.

Para isso, o arquivo `grpc_connection_service.dart` tem a função de realizar o gerenciamento das comandos invocados pelo cliente ou pelo servidor. Abaixo está o trecho do arquivo que implementa os métodos que abrem conexão a partir de um servidor ou cliente, respectivamente, além dos métodos para envio dos comandos **syncBoard** e **updateBoard**:

```
...

  openGRPCServer(
      ConnectionParams connectionParams,
      DrenchMultiplayerConnectionService
          drenchMultiplayerConnectionService) async {
    print('/// open grpc server');

    this.server =
        Server([DrenchGRPCService(drenchMultiplayerConnectionService, this)]);

    await server.serve(port: connectionParams.port);
  }

  openGRPCClient(ConnectionParams connectionParams,
      DrenchMultiplayerConnectionService drenchMultiplayerConnectionService) {
    clientChannel = ClientChannel(
      connectionParams.ipAddress,
      port: connectionParams.port,
      options: const ChannelOptions(credentials: ChannelCredentials.insecure()),
    );

    stub = DrenchClient(
      clientChannel,
      options: CallOptions(),
```

```
26      );
27
28      this.stub.subscribeSyncBoard(voidNoParam()).listen((value) {
29        drenchMultiplayerConnectionService.syncBoard({
30          'board': json.decode(value.board),
31          'reset': value.reset,
32        });
33      });
34
35      this.stub.subscribeUpdateBoard(voidNoParam()).listen((value) {
36        drenchMultiplayerConnectionService.updateBoard(value.colorIndex);
37      });
38    }
39
40    sendBoardSync(List<List<int>> board, bool reset) {
41      final data = SyncBoardData(board: json.encode(board), reset: reset);
42
43      if (this.server == null) {
44        this.stub.syncBoard(data);
45        return;
46      }
47
48      this.syncBoard$.add(data);
49    }
50
51    sendBoardUpdate(int colorIndex) {
52      final data = UpdateBoardData(colorIndex: colorIndex);
53
54      if (this.server == null) {
55        this.stub.updateBoard(data);
56        return;
57      }
58
59      this.updateBoard$.add(data);
60    }
61
62  ...
```

É possível perceber nos métodos **sendBoardSync** e **sendBoardUpdate** que, quando o comando é enviado do servidor para o cliente, os objetos de *stream* **syncBoard$** e **updateBoard$** são utilizados para armazenarem temporariamente os dados a serem enviados.

Para completar a comunicação, o arquivo `drench_grpc_service.dart` implementa as funções que serão utilizadas pelo servidor para manipular as invocações dos métodos pelo cliente. É possível notar o uso dos objetos de *stream* citados anteriormente no retorno das funções **subscribeSyncBoard** e **subscribeUpdateBoard**:

```
1   ...
2   class DrenchGRPCService extends DrenchServiceBase {
3     DrenchMultiplayerConnectionService drenchMultiplayerConnectionService;
4     GRPCConnectionService gRPCConnectionService;
5
6     DrenchGRPCService(
```

```
7         this.drenchMultiplayerConnectionService, this.gRPCConnectionService);
8
9     @override
10    Stream<SyncBoardData> subscribeSyncBoard(
11        ServiceCall call, voidNoParam request) {
12      return this.gRPCConnectionService.syncBoard$;
13    }
14
15    @override
16    Stream<UpdateBoardData> subscribeUpdateBoard(
17        ServiceCall call, voidNoParam request) {
18      return this.gRPCConnectionService.updateBoard$;
19    }
20
21    @override
22    Future<voidNoParam> syncBoard(ServiceCall call, SyncBoardData request) async {
23      this
24          .drenchMultiplayerConnectionService
25          .syncBoard$
26          .add({'board': json.decode(request.board), 'reset': request.reset});
27
28      return voidNoParam();
29    }
30
31    @override
32    Future<voidNoParam> updateBoard(
33        ServiceCall call, UpdateBoardData request) async {
34      print(this.drenchMultiplayerConnectionService);
35
36      this
37          .drenchMultiplayerConnectionService
38          .updateBoard$
39          .add(request.colorIndex);
40
41      return voidNoParam();
42    }
43  }
```

2.5 Resultados e análise

A implementação da comunicação via **gRPC** funcionou como esperado. A comunicação de cliente para servidor, por mais direta, foi mais simples para ser implementada.

A comunicação no sentido inverso, do servidor para o cliente, também foi possibilitada com facilidade devido à possibilidade de usar retornos de objetos *stream* – um recurso implementado pelo *gRPC*–, o que permite notificações diretas e dinâmicas vindas do servidor.

As figuras 2 e 3 exibem a utilização do jogo no momento da conexão entre cliente e servidor, e após o jogo iniciado, respectivamente.

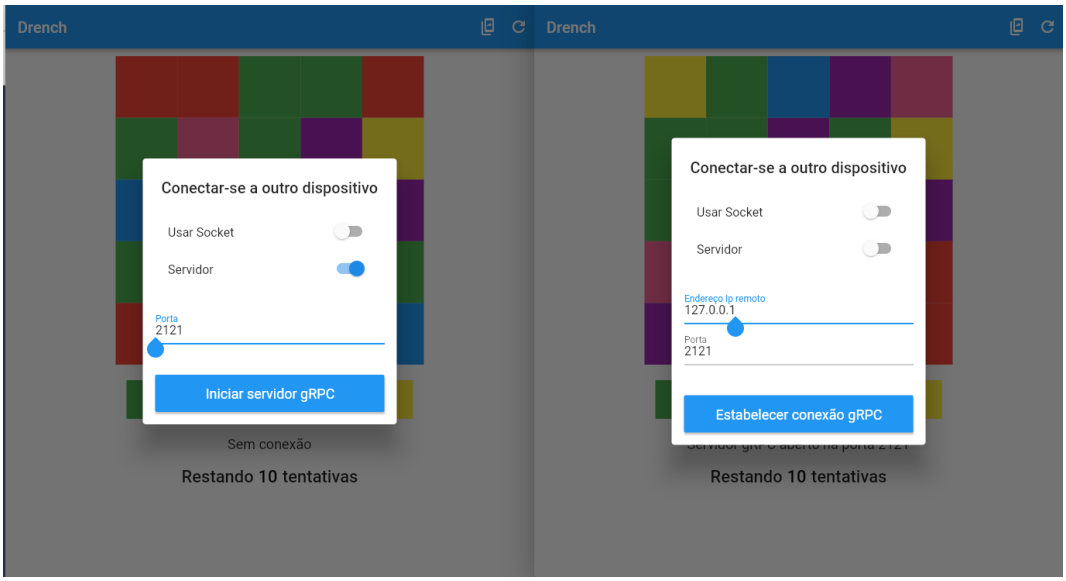

Figura 2: Print de dois simuladores na tela de conexão via gRPC. Fonte: autores.

Figura 3: Print de dois simuladores se comunicando via gRPC. Fonte: autores.

3 Conclusão

Utilizar Chamada Remota de Procedimento para comunicação entre dois processos, nos permite obter uma forma eficiente e abstraída para que possamos trocar informações entre softwares. Anteriormente utilizamos sockets para tal, e diferentemente do trabalho atual, a interface entre o processo **A** e o processo **B** era muito mais baixo nível; estávamos trocando mensagens simples e de acordo com o conteúdo dessas mensagens, realizávamos nossas ações. Essa forma de comunicação no entanto, é falha e pode facilmente comportar de maneira inesperada: basta trocarmos uma informação não padronizada, uma string com um caracter trocado e temos um sistema não-funcional.

Na abordagem RPC, estamos muito mais atados ao que os desenvolvedores definem: os métodos são estabelecidos e chamados, obedecendo sempre a estrutura utilizada. Assim, podemos reduzir a chance de falhas através da limitação das ações tomadas pelos usuários.

Além disso, o padrão gRPC permite a definição de métodos que retornam uma *stream*, o que permite implementar um sistema de notificações do servidor para o cliente, e ampliar as possibilidade de uso dessa tecnologia.

Referências

[1] gRPC, . URL https://grpc.io/.

[2] Protocol Buffers, . URL https://developers.google.com/protocol-buffers.

4 Anexos

4.1 Código utilizado

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

4.1.1 features/drench_game/drench_multiplayer/drench_multiplayer_connection_service.dart

```dart
import 'package:drench/features/multiplayer/connection_params.model.dart';
import 'package:drench/features/multiplayer/grpc/grpc_connection_service.dart';
import 'package:drench/features/multiplayer/socket/socket_connection_service.dart';
import 'package:rxdart/subjects.dart';

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

  ReplaySubject<int> updateBoard$ = ReplaySubject<int>();

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

  SocketConnectionService _socketConnectionService;
  GRPCConnectionService _gRPCConnectionService;

  DrenchMultiplayerConnectionService() {
    this.createSocketService();
    this.createGRPCService();
  }

  createSocketService() {
    final socketService = SocketConnectionService();
    this._socketConnectionService = socketService;

    socketService.currentConnectionParams$
        .listen((params) => this.currentConnectionParams$.add(params));

    socketService.dataReceiving$.listen(handleSocketData);
  }

  createGRPCService() {
    final gRPCService = GRPCConnectionService();
    this._gRPCConnectionService = gRPCService;
  }

  void connect(ConnectionParams connectionParams) {
    closeActiveConnections();

    if (connectionParams.isSocket) {
      this._socketConnectionService.connect(connectionParams);
      return;
    }
```

```
45
46      this.currentConnectionParams$.add(connectionParams);
47      this._gRPCConnectionService.connect(connectionParams, this);
48    }
49
50    closeActiveConnections() {
51      this._socketConnectionService.closeActiveConnections();
52      this._gRPCConnectionService.closeActiveConnections();
53    }
54
55    void handleSocketData(value) {
56      if (value['type'] == 'updateBoard') {
57        updateBoard(value['colorIndex']);
58        return;
59      }
60
61      if (value['type'] == 'syncBoard') {
62        syncBoard(value);
63        return;
64      }
65    }
66
67    updateBoard(int colorIndex) {
68      this.updateBoard$.add(colorIndex);
69    }
70
71    syncBoard(value) {
72      this.syncBoard$.add(value);
73    }
74
75    sendBoardSync(List<List<int>> board, bool reset) {
76      final connectionParams = this.currentConnectionParams$.value;
77
78      if (!connectionParams.isSocket) {
79        this._gRPCConnectionService.sendBoardSync(board, reset);
80        return;
81      }
82
83      this
84          ._socketConnectionService
85          .sendData({'type': 'syncBoard', 'board': board, 'reset': reset});
86    }
87
88    sendBoardUpdate(int colorIndex) {
89      final connectionParams = this.currentConnectionParams$.value;
90
91      if (!connectionParams.isSocket) {
92        this._gRPCConnectionService.sendBoardUpdate(colorIndex);
93        return;
94      }
95
96      this._socketConnectionService.sendData({
97        'type': 'updateBoard',
98        'colorIndex': colorIndex,
99      });
100   }
101 }
```

4.1.2 features/drench_game/widgets/drench_connection_status.dart

```dart
import 'package:drench/features/multiplayer/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.isSocket) {
      return _grpc();
    }

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

    return _tcpClientOrUpd();
  }

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

  _tcpServer() {
    return Column(
      children: <Widget>[
        Text(
          'Servidor TCP aberto na porta ${this.connectionParams.port}',
          style: textStyle,
        ),
        SizedBox(
          height: 5,
        ),
        _clienteInfoInTcpServer(),
      ].where((element) => element != null).toList(),
    );
```

```
58    }
59
60    Widget _clienteInfoInTcpServer() {
61      if (this.connectionParams.remoteIpAddress == null) {
62        return null;
63      }
64
65      return Text(
66        'Cliente connectado: ${this.connectionParams.remoteIpAddress}:${this.connectionParams.remotePort}',
67        style: textStyle,
68      );
69    }
70
71    Widget _tcpClientOrUpd() {
72      return Column(
73        children: <Widget>[
74          Text(
75            getTcpClientOrUpdText(),
76            style: textStyle,
77          ),
78        ],
79      );
80    }
81
82    String getTcpClientOrUpdText() {
83      if (connectionParams.isTcp) {
84        return 'Cliente TCP conectado ao servidor
      ↪  ${this.connectionParams.ipAddress}:${this.connectionParams.port}';
85      }
86
87      return 'Host UDP aberto na porta ${this.connectionParams.port}';
88    }
89
90    Widget _grpc() {
91      return Column(
92        children: <Widget>[
93          Text(
94            getGRPCText(),
95            style: textStyle,
96          ),
97        ],
98      );
99    }
100
101   String getGRPCText() {
102     if (this.connectionParams.isServer) {
103       return 'Servidor gRPC aberto na porta ${this.connectionParams.port}';
104     }
105
106     return 'Cliente gRPC connectado ao servidor
      ↪  ${this.connectionParams.ipAddress}:${this.connectionParams.port}';
107   }
108 }
```

4.1.3 features/drench_game/widgets/drench_control_menu.dart

```dart
import 'package:drench/features/drench_game/drench_game.model.dart';
import 'package:drench/features/drench_game/widgets/drench_connection_status.dart';
import 'package:drench/features/multiplayer/connection_params.model.dart';
import 'package:drench/features/drench_game/drench_controller.dart';
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';

class DrenchControlMenu extends StatelessWidget {
  final bool gameOver;
  final DrenchController controller;
  final double controlMenuSize;
  final DrenchGame drenchGame;
  final ConnectionParams connectionParams;

  DrenchControlMenu({
    this.gameOver,
    this.controller,
    this.controlMenuSize,
    this.drenchGame,
    this.connectionParams,
  });

  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Column(
        children: [
          buildBottomMenu(),
          buildBottomConnectionStatus(),
          buildBottomStatus(),
          buildBottomOption(),
        ],
      ),
    );
  }

  Container buildBottomMenu() {
    List<Container> buttons = [];

    for (int i = 0; i < 6; i++) {
      buttons.add(Container(
        height: controlMenuSize / 8,
        width: controlMenuSize / 8,
        color: DrenchGame.getColor(i),
        child: FlatButton(
          color: DrenchGame.getColor(i),
          onPressed: () {
            this.controller.updateBoard(i);
            this.controller.sendBoardUpdate(i);
          },
          child: SizedBox.shrink(),
        ),
      ));
    }

    return Container(
      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,
```

```
        ),
      ),
    );
  }

  Container buildBottomOption() {
    if (!gameOver) {
      return Container(
        child: SizedBox.shrink(),
      );
    }

    return Container(
      padding: EdgeInsets.fromLTRB(0, 10, 0, 15),
      width: controlMenuSize,
      child: FlatButton(
        color: Colors.green,
        onPressed: () {
          this.controller.newGame(true);
        },
        child: Padding(
          padding: const EdgeInsets.symmetric(vertical: 4),
          child: Text(
            'Novo Jogo',
            style: TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.w500,
              color: Colors.white,
            ),
          ),
        ),
      ),
    );
  }
}
```

4.1.4 features/drench_game/widgets/drench_matrix.dart

```
import 'package:drench/features/drench_game/drench_game.model.dart';
import 'package:flutter/widgets.dart';

class DrenchMatrix extends StatelessWidget {
  final DrenchGame drenchGame;
  final double widgetSize;

  DrenchMatrix({this.drenchGame, this.widgetSize});

  @override
  Widget build(BuildContext context) {
    List<Widget> result = [];

    for (int i = 0; i < this.drenchGame.size; i++) {
      result.add(
        _row(i),
      );
    }
```

```
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,
34      );
35    }
36
37    _square(i, j) {
38      return Container(
39        height: this.widgetSize / this.drenchGame.size,
40        width: this.widgetSize / this.drenchGame.size,
41        color: DrenchGame.getColor(this.drenchGame.matrix[i][j]),
42      );
43    }
44  }
```

4.1.5 features/drench_game/drench_component.dart

```
1   import 'dart:math';
2   import 'package:drench/features/drench_game/drench_game.model.dart';
3   import 'package:drench/features/drench_game/widgets/drench_control_menu.dart';
4   import 'package:drench/features/drench_game/widgets/drench_matrix.dart';
5   import 'package:drench/features/multiplayer/connection_params.model.dart';
6   import 'package:drench/features/drench_game/drench_controller.dart';
7   import 'package:flutter/material.dart';
8
9   class DrenchComponent extends StatefulWidget {
10    final DrenchController controller;
11
12    DrenchComponent({Key key, this.controller});
13
14    @override
15    _DrenchComponentState createState() =>
16        _DrenchComponentState(controller: controller);
17  }
18
19  class _DrenchComponentState extends State<DrenchComponent> {
20    final DrenchController controller;
21
22    final double topWidgetHeight = 10;
23    double get bottomWidgetHeight => min(
24          0.5 * MediaQuery.of(context).size.height,
25          275,
26        );
27
```

```
28    double get maxDrenchBoardHeight => max(
29          MediaQuery.of(context).size.height -
30              topWidgetHeight -
31              bottomWidgetHeight -
32              56,
33          0,
34        );
35
36    double get drenchBoardSize => min(
37          MediaQuery.of(context).size.width - 10,
38          maxDrenchBoardHeight,
39        );
40
41    double get controlMenuSize => min(
42          MediaQuery.of(context).size.width,
43          max(
44            drenchBoardSize,
45            300,
46          ),
47        );
48
49    DrenchGame drenchGame;
50    List<Color> colors = DrenchGame.colors;
51
52    ConnectionParams connectionParams;
53
54    bool gameOver = false;
55
56    _DrenchComponentState({this.controller}) {
57      controller.newGame = newGame;
58      controller.updateBoard = updateBoard;
59      controller.syncBoard = syncBoard;
60      controller.setConnectionParams = setConnectionParams;
61
62      this.setDrenchGame();
63    }
64
65    setDrenchGame() {
66      this.drenchGame = DrenchGame(maxClicks: 10, size: 5);
67    }
68
69    void newGame(bool syncBoard) {
70      setState(() {
71        this.setDrenchGame();
72        gameOver = false;
73      });
74
75      if (syncBoard && this.connectionParams != null) {
76        this.controller.sendBoardSync(this.drenchGame.matrix, true);
77      }
78    }
79
80    void updateBoard(int value) {
81      if (gameOver == true) {
82        return;
83      }
84
85      this.drenchGame.paintFirstSquare(value);
86
87      if (this.drenchGame.isGameOver()) {
88        gameOver = true;
```

```
89        }
90
91        setState(() {});
92      }
93
94      void syncBoard(List<List<int>> board) {
95        setState(() {
96          this.drenchGame.matrix = board;
97        });
98      }
99
100     void setConnectionParams(ConnectionParams connectionParams) {
101       setState(() {
102         this.connectionParams = connectionParams;
103       });
104     }
105
106     @override
107     Widget build(BuildContext context) {
108       return SingleChildScrollView(
109         child: Column(
110           children: <Widget>[
111             _topWidget(),
112             DrenchMatrix(
113               drenchGame: this.drenchGame,
114               widgetSize: drenchBoardSize,
115             ),
116             _bottomWidget(),
117           ],
118         ),
119       );
120     }
121
122     Widget _topWidget() {
123       return ConstrainedBox(
124         constraints: BoxConstraints(
125           maxHeight: max(topWidgetHeight, 0),
126         ),
127         child: SizedBox(
128           height: max(topWidgetHeight, 0),
129         ),
130       );
131     }
132
133     Widget _bottomWidget() {
134       return ConstrainedBox(
135         constraints: BoxConstraints(
136           maxHeight: bottomWidgetHeight,
137         ),
138         child: DrenchControlMenu(
139           gameOver: this.gameOver,
140           controller: this.controller,
141           controlMenuSize: this.controlMenuSize,
142           drenchGame: this.drenchGame,
143           connectionParams: this.connectionParams,
144         ),
145       );
146     }
147   }
```

4.1.6 features/drench_game/drench_controller.dart

```dart
import 'package:drench/features/drench_game/drench_multiplayer/drench_multiplayer_connection_service.dart';
import 'package:drench/features/multiplayer/connection_params.model.dart';

class DrenchController {
  void Function(bool newGame) newGame;
  void Function(int colorIndex) updateBoard;
  void Function(List<List<int>> colorIndex) syncBoard;
  void Function(ConnectionParams connectionParams) setConnectionParams;

  DrenchMultiplayerConnectionService _drenchMultiplayerConnectionService;

  setMultiplayerConnectionService(
      DrenchMultiplayerConnectionService multiplayerConnectionService) {
    this._drenchMultiplayerConnectionService = multiplayerConnectionService;

    _drenchMultiplayerConnectionService.currentConnectionParams$
        .listen(handleChangeConnectionParams);

    _drenchMultiplayerConnectionService.updateBoard$.listen(handleUpdateBoard);
    _drenchMultiplayerConnectionService.syncBoard$.listen(handleSyncBoard);
  }

  handleChangeConnectionParams(ConnectionParams connectionParams) {
    this.setConnectionParams(connectionParams);
  }

  handleUpdateBoard(int colorIndex) {
    this.updateBoard(colorIndex);
  }

  handleSyncBoard(Map<String, dynamic> args) {
    List<List<int>> board = new List<List<int>>();

    args['board'].forEach((vector) {
      List<int> list = new List<int>();

      vector.forEach((value) {
        list.add(value as int);
      });

      board.add(list);
    });

    if (args['reset'] == true) {
      this.newGame(false);
    }

    this.syncBoard(board);
  }

  sendBoardSync(List<List<int>> board, bool reset) {
    this._drenchMultiplayerConnectionService.sendBoardSync(board, reset);
  }

  sendBoardUpdate(int colorIndex) {
    this._drenchMultiplayerConnectionService.sendBoardUpdate(colorIndex);
```

```
57    }
58  }
```

4.1.7 features/drench_game/drench_game_model.dart

```dart
import 'dart:math';

import 'package:flutter/material.dart';

class DrenchGame {
  static final List<Color> colors = [
    Colors.green,
    Colors.pink[300],
    Colors.purple,
    Colors.blue,
    Colors.red,
    Colors.yellow,
  ];

  final int maxClicks;
  final int size;

  int _paintsCount;
  List<List<int>> _matrix;

  DrenchGame({@required this.maxClicks, @required this.size}) {
    _paintsCount = 0;

    _matrix = List.generate(size, (i) => List(size), growable: false);

    var rng = new Random();
    for (int i = 0; i < size; i++) {
      for (int j = 0; j < size; j++) {
        _matrix[i][j] = rng.nextInt(100) % 6;
      }
    }
  }

  List<List<int>> get matrix => this._matrix;
  set matrix (List<List<int>> matrix) => this._matrix = matrix;

  int get remainingPaints => this.maxClicks - this._paintsCount;

  bool isGameOver() {
    if (_paintsCount >= maxClicks) {
      return true;
    }

    int val = _matrix[0][0];
    int cont = 0;

    for (int i = 0; i < size; i++) {
      for (int j = 0; j < size; j++) {
        if (_matrix[i][j] != val) {
          return false;
        }
```

```
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  }
```

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

```
1   import 'package:drench/features/multiplayer/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 _isSocket = false;
```

```
15    bool _isTcp = true;
16    bool _isServer = false;
17    TextEditingController _ipAddressFieldController;
18    TextEditingController _portFieldController;
19    TextEditingController _remotePortFieldController;
20
21    _ConnectionDialogFormState() {
22      this._ipAddressFieldController = TextEditingController(text: '127.0.0.1');
23      this._portFieldController = TextEditingController(text: '2121');
24      this._remotePortFieldController = TextEditingController(text: '2122');
25    }
26
27    @override
28    Widget build(BuildContext context) {
29      return SingleChildScrollView(
30        child: Column(
31          mainAxisSize: MainAxisSize.min,
32          crossAxisAlignment: CrossAxisAlignment.start,
33          children: _getFields(),
34        ),
35      );
36    }
37
38    List<Widget> _getFields() {
39      List<Widget> list = [
40        _isSocketField(),
41        hasIsTcpField() ? _isTcpField() : null,
42        hasIsServerField() ? _isServerField() : null,
43        SizedBox(height: 20),
44        hasIpAddressField() ? _ipAddressField() : null,
45        _portField(),
46        hasRemotePortField() ? _remotePortField() : null,
47        SizedBox(height: 40),
48        _submitButton()
49      ];
50
51      return list.where((field) => field != null).toList();
52    }
53
54    Widget _isSocketField() {
55      return SwitchListTile(
56        title: const Text('Usar Socket'),
57        value: _isSocket,
58        onChanged: (bool value) {
59          setState(() {
60            _isSocket = value;
61          });
62        },
63      );
64    }
65
66    Widget _isServerField() {
67      return SwitchListTile(
68        title: const Text('Servidor'),
69        value: _isServer,
70        onChanged: (bool value) {
71          setState(() {
72            _isServer = value;
73          });
74        },
75      );
```

```
76    }
77
78    Widget _isTcpField() {
79      return SwitchListTile(
80        title: const Text('Usar TCP'),
81        value: _isTcp,
82        onChanged: (bool value) {
83          setState(() {
84            _isTcp = value;
85          });
86        },
87      );
88    }
89
90    Widget _ipAddressField() {
91      return TextField(
92        controller: _ipAddressFieldController,
93        autofocus: true,
94        textInputAction: TextInputAction.next,
95        decoration: InputDecoration(
96          labelText: getIpAddressFieldLabel(),
97        ),
98      );
99    }
100
101   getIpAddressFieldLabel() {
102     if (!_isTcp || !_isServer) {
103       return 'Endereço Ip remoto';
104     }
105
106     return 'Endereço Ip';
107   }
108
109   Widget _portField() {
110     return TextField(
111       controller: _portFieldController,
112       textInputAction:
113           hasRemotePortField() ? TextInputAction.next : TextInputAction.go,
114       onSubmitted: (_) => hasRemotePortField() ? null : _initConnection(),
115       onChanged: (String port) {
116         setState(() {});
117       },
118       keyboardType: TextInputType.number,
119       inputFormatters: [
120         FilteringTextInputFormatter.digitsOnly,
121       ],
122       decoration: InputDecoration(
123         labelText: 'Porta',
124         errorText:
125             isValidPort() ? null : 'Porta inválida. Precisa ser maior que 1024',
126       ),
127     );
128   }
129
130   Widget _remotePortField() {
131     return TextField(
132       controller: _remotePortFieldController,
133       textInputAction: TextInputAction.go,
134       onSubmitted: (_) => _initConnection(),
135       onChanged: (String port) {
136         setState(() {});
```

```
137         },
138         keyboardType: TextInputType.number,
139         inputFormatters: [
140           FilteringTextInputFormatter.digitsOnly,
141         ],
142         decoration: InputDecoration(
143           labelText: 'Porta remota',
144           errorText: isValidRemotePort()
145               ? null
146               : 'Porta inválida. Precisa ser maior que 1024',
147         ),
148       );
149     }
150
151     Widget _submitButton() {
152       return SizedBox(
153         width: double.infinity,
154         child: RaisedButton(
155           color: Theme.of(context).primaryColor,
156           textColor: Colors.white,
157           child: Padding(
158             padding: const EdgeInsets.all(12),
159             child: Text(
160               _submitButtonText(),
161               textAlign: TextAlign.center,
162               style: TextStyle(
163                 fontSize: 18,
164                 fontWeight: FontWeight.w400,
165               ),
166             ),
167           ),
168           onPressed:
169               isValidPort() && (hasRemotePortField() ? isValidRemotePort() : true)
170                   ? _initConnection
171                   : null,
172         ),
173       );
174     }
175
176     String _submitButtonText() {
177       if (!_isSocket) {
178         if (this._isServer) {
179           return 'Iniciar servidor gRPC';
180         }
181
182         return 'Estabelecer conexão gRPC';
183       }
184
185       if (!this._isTcp) {
186         return 'Abrir porta UDP';
187       }
188
189       if (this._isServer) {
190         return 'Iniciar servidor TCP';
191       }
192
193       return 'Estabelecer conexão TCP';
194     }
195
196     _initConnection() {
197       ConnectionParams params = this.getValues();
```

```
198       Navigator.of(this.context).pop(params);
199     }
200
201     ConnectionParams getValues() {
202       return ConnectionParams(
203         isSocket: _isSocket,
204         isTcp: _isTcp,
205         isServer: hasIsServerField() ? _isServer : null,
206         ipAddress:
207             hasIpAddressField() ? this._ipAddressFieldController.text : null,
208         port: int.parse(this._portFieldController.text),
209         remotePort: int.parse(this._remotePortFieldController.text),
210       );
211     }
212
213     bool isValidPort() {
214       String port = _portFieldController.text;
215
216       if (port.isNotEmpty && int.parse(port) <= 1024) {
217         return false;
218       }
219
220       return true;
221     }
222
223     bool isValidRemotePort() {
224       String remotePort = _remotePortFieldController.text;
225
226       if (remotePort.isNotEmpty && int.parse(remotePort) <= 1024) {
227         return false;
228       }
229
230       return true;
231     }
232
233     bool hasIsTcpField() {
234       return _isSocket;
235     }
236
237     bool hasIsServerField() {
238       return !_isSocket || _isTcp;
239     }
240
241     bool hasRemotePortField() {
242       return _isSocket && !_isTcp;
243     }
244
245     bool hasIpAddressField() {
246       if (!_isSocket) {
247         return !_isServer;
248       }
249
250       return !_isTcp || _isTcp && !_isServer;
251     }
252   }
```

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

```dart
import 'package:drench/features/multiplayer/components/connection_dialog/connection_dialog_form.dart';
import 'package:flutter/material.dart';

class ConnectionDialogService {
  Future show(BuildContext context) {
    return showDialog(
      context: context,
      builder: (BuildContext context) => _alertDialog(context),
    );
  }

  AlertDialog _alertDialog(BuildContext context) {
    return new AlertDialog(
      title: Text('Conectar-se a outro dispositivo'),
      contentPadding: EdgeInsets.fromLTRB(16, 20, 16, 16),
      content: _content(),
    );
  }

  Widget _content() {
    return ConnectionDialogForm();
  }
}
```

4.1.10 features/multiplayer/grpc/drench_grpc_service.dart

```dart
import 'dart:convert';

import
↪  'package:drench/features/drench_game/drench_multiplayer/drench_multiplayer_connection_service.dart';
import 'package:drench/features/multiplayer/grpc/grpc_connection_service.dart';
import 'package:drench/generated/drench.pbgrpc.dart';
import 'package:grpc/src/server/call.dart';

class DrenchGRPCService extends DrenchServiceBase {
  DrenchMultiplayerConnectionService drenchMultiplayerConnectionService;
  GRPCConnectionService gRPCConnectionService;

  DrenchGRPCService(
      this.drenchMultiplayerConnectionService, this.gRPCConnectionService);

  @override
  Stream<SyncBoardData> subscribeSyncBoard(
      ServiceCall call, voidNoParam request) {
    return this.gRPCConnectionService.syncBoard$;
  }

  @override
  Stream<UpdateBoardData> subscribeUpdateBoard(
      ServiceCall call, voidNoParam request) {
    return this.gRPCConnectionService.updateBoard$;
  }

```

```
27    @override
28    Future<voidNoParam> syncBoard(ServiceCall call, SyncBoardData request) async {
29      this
30          .drenchMultiplayerConnectionService
31          .syncBoard$
32          .add({'board': json.decode(request.board), 'reset': request.reset});
33
34      return voidNoParam();
35    }
36
37    @override
38    Future<voidNoParam> updateBoard(
39        ServiceCall call, UpdateBoardData request) async {
40      print(this.drenchMultiplayerConnectionService);
41
42      this
43          .drenchMultiplayerConnectionService
44          .updateBoard$
45          .add(request.colorIndex);
46
47      return voidNoParam();
48    }
49  }
```

4.1.11 features/multiplayer/grpc/grpc_connection_service.dart

```
1   import 'dart:convert';
2
3   import 'package:drench/features/multiplayer/grpc/drench_grpc_service.dart';
4   import
    ↪  'package:drench/features/drench_game/drench_multiplayer/drench_multiplayer_connection_service.dart';
5   import 'package:drench/features/multiplayer/connection_params.model.dart';
6   import 'package:drench/generated/drench.pbgrpc.dart';
7   import 'package:grpc/grpc.dart';
8   import 'package:rxdart/subjects.dart';
9
10  class GRPCConnectionService {
11    Server server;
12
13    ClientChannel clientChannel;
14    DrenchClient stub;
15
16    ReplaySubject<UpdateBoardData> updateBoard$ =
17        ReplaySubject<UpdateBoardData>();
18
19    ReplaySubject<SyncBoardData> syncBoard$ = ReplaySubject<SyncBoardData>();
20
21    void connect(ConnectionParams connectionParams,
22        DrenchMultiplayerConnectionService drenchMultiplayerConnectionService) {
23      if (connectionParams.isServer) {
24        openGRPCServer(connectionParams, drenchMultiplayerConnectionService);
25        return;
26      }
27
28      openGRPCClient(connectionParams, drenchMultiplayerConnectionService);
29    }
```

```
30
31    openGRPCServer(
32        ConnectionParams connectionParams,
33        DrenchMultiplayerConnectionService
34            drenchMultiplayerConnectionService) async {
35      print('/// open grpc server');
36
37      this.server =
38          Server([DrenchGRPCService(drenchMultiplayerConnectionService, this)]);
39
40      await server.serve(port: connectionParams.port);
41    }
42
43    openGRPCClient(ConnectionParams connectionParams,
44        DrenchMultiplayerConnectionService drenchMultiplayerConnectionService) {
45      clientChannel = ClientChannel(
46        connectionParams.ipAddress,
47        port: connectionParams.port,
48        options: const ChannelOptions(credentials: ChannelCredentials.insecure()),
49      );
50
51      stub = DrenchClient(
52        clientChannel,
53        options: CallOptions(),
54      );
55
56      this.stub.subscribeSyncBoard(voidNoParam()).listen((value) {
57        drenchMultiplayerConnectionService.syncBoard({
58          'board': json.decode(value.board),
59          'reset': value.reset,
60        });
61      });
62
63      this.stub.subscribeUpdateBoard(voidNoParam()).listen((value) {
64        drenchMultiplayerConnectionService.updateBoard(value.colorIndex);
65      });
66    }
67
68    sendBoardSync(List<List<int>> board, bool reset) {
69      final data = SyncBoardData(board: json.encode(board), reset: reset);
70
71      if (this.server == null) {
72        this.stub.syncBoard(data);
73        return;
74      }
75
76      this.syncBoard$.add(data);
77    }
78
79    sendBoardUpdate(int colorIndex) {
80      final data = UpdateBoardData(colorIndex: colorIndex);
81
82      if (this.server == null) {
83        this.stub.updateBoard(data);
84        return;
85      }
86
87      this.updateBoard$.add(data);
88    }
89
90    closeActiveConnections() {
```

```
91      if (this.server != null) {
92        this.server.shutdown();
93      }
94
95      if (this.clientChannel != null) {
96        this.clientChannel.shutdown();
97      }
98    }
99  }
```

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

```
1   import 'dart:io';
2
3   import 'package:drench/features/multiplayer/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 }
```

4.1.13 multiplayer/socket/udp/udp_connection.dart

```
1  import 'dart:io';
2
3  import 'package:drench/features/multiplayer/connection_params.model.dart';
4  import 'package:drench/features/multiplayer/socket/socket_connection_service.dart';
5
6  class UdpConnection {
7    SocketConnectionService socketConnectionService;
8
9    RawDatagramSocket udpServer;
10
11   UdpConnection({this.socketConnectionService});
12
13   void openConnection(ConnectionParams connectionParams) async {
14     openUdpServer(connectionParams);
15   }
16
17   void openUdpServer(ConnectionParams connectionParams) async {
18     this.udpServer = await RawDatagramSocket.bind(
19         InternetAddress.anyIPv4, connectionParams.port);
20
21     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);

    print(addresses);

    this
        .udpServer
        .send(data.codeUnits, addresses[0], connectionParams.remotePort);
  }

  void closeActiveConnections() {
    if (this.udpServer != null) {
      this.udpServer.close();
      print('close udpServer');
      this.udpServer = null;
    }
  }
}
```

4.1.14 features/mulçtiplayer/socket/socket_connection_service.dart

```
import 'dart:convert';

```

```dart
import 'package:drench/features/multiplayer/connection_params.model.dart';
import 'package:drench/features/multiplayer/socket/tcp/tcp_connection.dart';
import 'package:drench/features/multiplayer/socket/udp/udp_connection.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) {
    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);
  }
```

```
64
65    void broadcastMessageReceived(dynamic data) {
66      try {
67        dataReceiving$.add(json.decode(data));
68      } catch (e) {
69        print('decode error:');
70        print(e);
71      }
72    }
73
74    void closeActiveConnections() {
75      this._tcp.closeActiveConnections();
76      this._udp.closeActiveConnections();
77    }
78
79    getInformationMessage(String message) {
80      return json.encode({'type': 'information', 'message': message});
81    }
82
83    ConnectionParams getConnectionParams() {
84      return this.currentConnectionParams$.value;
85    }
86  }
```

4.1.15 features/multiplayer/connection_params.model.dart

```
1   class ConnectionParams {
2     bool isSocket;
3     bool isTcp;
4     bool isServer;
5     String ipAddress;
6     int port;
7     String remoteIpAddress;
8     int remotePort;
9
10    ConnectionParams(
11        {this.isSocket,
12        this.isTcp,
13        this.isServer,
14        this.ipAddress,
15        this.port,
16        this.remoteIpAddress,
17        this.remotePort});
18
19    Map<String, dynamic> toJson() => {
20          'isSocket': isSocket,
21          'isTcp': isTcp,
22          'isServer': isServer,
23          'ipAddress': ipAddress,
24          'port': port,
25          'remoteIpAddress': remoteIpAddress,
26          'remotePort': remotePort,
27        };
28  }
```

4.1.16 pages/home_page/home_page.dart

```dart
import
↪  'package:drench/features/drench_game/drench_multiplayer/drench_multiplayer_connection_service.dart';
import 'package:drench/features/multiplayer/components/connection_dialog/connection_dialog_service.dart';
import 'package:drench/features/multiplayer/connection_params.model.dart';
import 'package:drench/features/drench_game/drench_component.dart';
import 'package:drench/features/drench_game/drench_controller.dart';
import 'package:flutter/material.dart';
import 'package:flutter/widgets.dart';

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  final ConnectionDialogService _connectionDialogService =
      ConnectionDialogService();

  final DrenchMultiplayerConnectionService _drenchMultiplayerConnectionService =
      DrenchMultiplayerConnectionService();

  final DrenchController drenchController = DrenchController();

  _HomePageState() {
    this
        .drenchController
        .setMultiplayerConnectionService(_drenchMultiplayerConnectionService);
  }

  @override
  Widget build(BuildContext context) {
    return Container(
      child: Scaffold(
        appBar: _appBar(),
        body: _body(),
      ),
    );
  }

  Widget _appBar() {
    return AppBar(
      title: Text("Drench"),
      actions: _appBarActions(),
    );
  }

  List<Widget> _appBarActions() {
    return [
      _connectToDeviceActionButton(),
      _newGameActionButton(),
    ];
  }

  IconButton _connectToDeviceActionButton() {
    return IconButton(
      icon: Icon(
        Icons.offline_share,
```

```
57          color: Colors.white,
58        ),
59        onPressed: this.showConnectionDialog,
60      );
61    }
62
63    IconButton _newGameActionButton() {
64      return IconButton(
65        icon: Icon(
66          Icons.refresh,
67          color: Colors.white,
68        ),
69        onPressed: () {
70          this.drenchController.newGame(true);
71        },
72      );
73    }
74
75    Widget _body() {
76      return DrenchComponent(controller: drenchController);
77    }
78
79    void showConnectionDialog() async {
80      ConnectionParams connectionParams =
81          await _connectionDialogService.show(this.context);
82
83      if (connectionParams == null) {
84        return;
85      }
86
87      this._drenchMultiplayerConnectionService.connect(connectionParams);
88    }
89  }
```