SUDOKU EM FLUTTER: UMA ANÁLISE DO PROCESSO DE DESENVOLVIMENTO DE UM APLICATIVO PARA DISPOSITIVOS MÓVEIS

Bruno Garcia, Pr.Dr, Henrique Dezani

e-mail:
garciabrunodev@gmail.com
henrique.dezani@fatec.sp.gov.br

Resumo: Este trabalho descreve o processo de desenvolvimento de um aplicativo de Sudoku em Flutter, uma plataforma de desenvolvimento de aplicativos móveis criada pelo Google. O objetivo do projeto é criar uma aplicação capaz de gerar puzzles de Sudoku aleatórios e oferecer uma interface fácil de usar para os jogadores. O trabalho apresenta uma análise das ferramentas utilizadas no processo de desenvolvimento, incluindo a linguagem de programação Dart, a estrutura do Flutter e as bibliotecas utilizadas para construir a interface do usuário e gerar os puzzles. Também é realizada uma avaliação do aplicativo final, incluindo testes de usabilidade e desempenho. Os resultados indicam que o aplicativo é uma solução viável e eficiente para jogar Sudoku em dispositivos móveis, oferecendo uma experiência de usuário agradável e desafiadora. Conclui-se que o Flutter é uma plataforma promissora para o desenvolvimento de aplicativos móveis e pode ser uma opção interessante para desenvolvedores que desejam criar aplicativos para várias plataformas com uma única base de código

Palavras-chave: Flutter, Sudoku. Aplicativo móvel. Desenvolvimento de software. Dart. Geração de puzzles.

Abstract: *This paper describes the process of developing a Sudoku application in Flutter, a mobile app development platform created by Google. The project aims to create an application capable of generating random Sudoku puzzles and providing an easy-to-use interface for players. The paper presents an analysis of the tools used in the development process, including the Dart programming language, the Flutter framework, and the libraries used to build the user interface and generate the puzzles. An evaluation of the final application is also performed, including usability and performance testing. The results indicate that the application is a viable and efficient solution for playing Sudoku on mobile devices, offering a pleasant and challenging user experience. It is concluded that Flutter is a promising platform for mobile app development and may be an interesting option for developers who want to create apps for multiple platforms with a single code base.*

Keywords: *Flutter, Sudoku. Mobile application. Software development. Dart. Puzzle generation.*

1 Introdução

A criação de aplicativos móveis tem se tornado cada vez mais comum nos dias de hoje, sendo que muitos desenvolvedores buscam soluções que permitam criar aplicativos multiplataforma com facilidade e eficiência. Nesse contexto, o Flutter se destaca como uma plataforma de desenvolvimento de aplicativos móveis que oferece uma série de recursos e vantagens para os desenvolvedores, incluindo a possibilidade de criar aplicativos para iOS e Android com uma única base de código.

Este trabalho tem como objetivo descrever o processo de desenvolvimento de um aplicativo de Sudoku em Flutter, utilizando a linguagem de programação Dart e as ferramentas disponibilizadas pela plataforma. O Sudoku é um jogo de lógica que tem se tornado bastante popular em todo o mundo, sendo que a criação de um aplicativo que permita jogá-lo em

dispositivos móveis pode ser uma solução interessante para aqueles que buscam uma forma de passar o tempo e exercitar o cérebro.

O aplicativo desenvolvido neste trabalho é capaz de gerar jogos de Sudoku aleatórios e oferece uma interface de usuário intuitiva e fácil de usar. Para tanto, foram utilizadas bibliotecas específicas do Flutter para criar a interface gráfica e gerar os puzzles. Além disso, foram realizados testes de usabilidade e desempenho para avaliar o aplicativo final.

Espera-se que este trabalho possa contribuir para o desenvolvimento de aplicativos móveis em Flutter e para o uso dessa plataforma como uma opção viável e eficiente para a criação de aplicativos multiplataforma. Além disso, espera-se que o aplicativo de Sudoku desenvolvido possa ser uma opção interessante e desafiadora para os jogadores que buscam uma forma de passar o tempo e exercitar a mente.

2 Justificativa

A criação de aplicativos móveis é uma área em constante crescimento, com milhões de usuários em todo o mundo que buscam aplicativos para os mais diversos fins, como entretenimento, produtividade, educação e muitos outros. Nesse sentido, o desenvolvimento de aplicativos multiplataforma tem se tornado cada vez mais importante, pois permite que um único código seja utilizado para criar aplicativos para diversas plataformas, como iOS e Android, reduzindo custos e tempo de desenvolvimento.

O Flutter é uma plataforma de desenvolvimento de aplicativos móveis que tem ganhado popularidade por permitir o desenvolvimento de aplicativos multiplataforma com uma única base de código. Além disso, o Flutter oferece uma série de recursos e vantagens para os desenvolvedores, como uma linguagem de programação moderna e eficiente, uma vasta biblioteca de widgets personalizáveis e uma ferramenta de compilação rápida que permite visualizar as mudanças no aplicativo em tempo real.

O Sudoku, por sua vez, é um jogo de lógica que tem se tornado bastante popular em todo o mundo, sendo que muitos jogadores buscam uma forma de jogá-lo em seus dispositivos móveis. Dessa forma, o desenvolvimento de um aplicativo de Sudoku em Flutter pode ser uma solução interessante para aqueles que buscam uma forma de passar o tempo e exercitar o cérebro.

Nesse contexto, este trabalho tem como objetivo descrever o processo de desenvolvimento de um aplicativo de Sudoku em Flutter, utilizando as ferramentas disponibilizadas pela plataforma. A escolha desse tema se justifica pela importância do Flutter como uma plataforma de desenvolvimento de aplicativos móveis e pela popularidade do Sudoku como um jogo desafiador e divertido para jogar em dispositivos móveis.

Espera-se que este trabalho possa contribuir para o desenvolvimento de aplicativos móveis em Flutter e para a utilização dessa plataforma como uma opção viável e eficiente para a criação de aplicativos multiplataforma. Além disso, espera-se que o aplicativo de Sudoku desenvolvido possa ser uma opção interessante e desafiadora para os jogadores que buscam uma forma de passar o tempo e exercitar a mente

3 Objetivo(s)

1. Desenvolver um aplicativo de Sudoku em Flutter capaz de criar e resolver Sudokus customizados.
2. Oferecer uma interface fácil de usar para os jogadores.
3. Utilizar as ferramentas disponibilizadas pela plataforma Flutter, incluindo a linguagem de programação Dart e as bibliotecas de *widgets* personalizáveis.
4. Contribuir para o desenvolvimento de aplicativos móveis em Flutter e para a utilização dessa plataforma como uma opção viável e eficiente para a criação de aplicativos multiplataforma.
5. Apresentar o desenvolvimento do aplicativo de Sudoku como um estudo de caso para demonstrar a aplicabilidade do Flutter para a criação de outros tipos de aplicativos móveis.

4 Fundamentação Teórica

4.1 Sudoku

O Sudoku é um jogo de lógica matemática que se tornou muito popular em todo o mundo nas últimas décadas. É um jogo de tabuleiro que consiste em uma grade de 9x9 células, dividida em nove regiões menores de 3x3 células cada. O objetivo do jogo é preencher cada célula com um número de 1 a 9, sem repetir nenhum número em cada linha, coluna ou região menor.

O Sudoku tem sido alvo de várias pesquisas acadêmicas, que têm explorado diferentes aspectos do jogo, como suas propriedades matemáticas e estratégias de resolução. Além disso, o jogo tem sido utilizado como um exemplo para testar algoritmos de resolução de quebra-cabeças e jogos.

A criação de um aplicativo de Sudoku em Flutter pode contribuir para a melhoria da experiência do usuário ao jogar o jogo em um dispositivo móvel, oferecendo uma interface amigável e fácil de usar, além de recursos adicionais, como a capacidade de gerar aleatoriamente novos puzzles para manter o desafio constante.

4.2 Flutter

Flutter é uma plataforma de desenvolvimento de aplicativos móveis criada pelo Google, lançada em 2017. É uma estrutura de código aberto que utiliza a linguagem de programação Dart, desenvolvida pela própria empresa. A plataforma é projetada para criar aplicativos de alta qualidade para iOS, Android, web e desktop a partir de uma única base de código.

A arquitetura do Flutter é baseada em widgets, que são os blocos de construção da interface do usuário (UI). Cada widget representa uma parte da interface do usuário, como um botão, uma caixa de texto ou um menu suspenso. Os widgets podem ser combinados para criar layouts complexos e personalizados. A plataforma também possui uma grande variedade de widgets prontos para uso, permitindo que os desenvolvedores construam interfaces de usuário ricas e animadas.

Além dos widgets, o Flutter também inclui recursos de hot-reload, que permitem que os desenvolvedores vejam as alterações no código em tempo real, sem precisar reiniciar o

aplicativo. Isso acelera o processo de desenvolvimento e ajuda os desenvolvedores a identificar e corrigir erros com mais eficiência.

O Flutter também oferece suporte a recursos avançados, como animações e efeitos visuais personalizados, tornando possível criar interfaces de usuário altamente interativas e imersivas. A plataforma também oferece ferramentas de depuração e perfil para ajudar os desenvolvedores a identificar e solucionar problemas de desempenho em seus aplicativos.

Em geral, o Flutter é uma plataforma promissora para o desenvolvimento de aplicativos móveis, permitindo que os desenvolvedores criem aplicativos de alta qualidade e de desempenho eficiente para várias plataformas a partir de uma única base de código. A plataforma tem uma comunidade de desenvolvedores em constante crescimento e é apoiada por grandes empresas.

4.3 FireBase

Firebase é uma plataforma de desenvolvimento de aplicativos móveis e *web* fornecida pelo Google. Ele oferece uma variedade de recursos e serviços que permitem que os desenvolvedores criem aplicativos de alta qualidade, escaláveis e seguros, com uma infraestrutura de *back-end* pronta para uso.

O Firebase oferece serviços para autenticação de usuários, armazenamento de dados em nuvem, análise de dados, mensagens em tempo real, notificações *push*, testes de qualidade, entre outros. Esses recursos ajudam os desenvolvedores a criar aplicativos robustos sem precisar construir toda a infraestrutura do zero.

O Firebase é baseado em uma arquitetura de nuvem escalável e segura. Ele utiliza a tecnologia de computação em nuvem do Google para garantir a segurança e a confiabilidade dos dados do aplicativo. Além disso, o Firebase é altamente escalável, permitindo que os aplicativos cresçam facilmente à medida que mais usuários são adicionados.

Outro benefício do Firebase é a integração com outras ferramentas e serviços do Google, como o Google Cloud Platform e o Google Analytics. Isso permite que os desenvolvedores usem uma variedade de ferramentas e serviços para criar aplicativos de alta qualidade e escaláveis.

Em resumo, o Firebase é uma plataforma de desenvolvimento de aplicativos móveis e *web* poderosa e altamente escalável que oferece uma variedade de recursos e serviços para ajudar os desenvolvedores a criar aplicativos de alta qualidade. Com a infraestrutura de *back-end* pronta para uso e a integração com outras ferramentas do Google, o Firebase é uma solução para muitos desenvolvedores que buscam uma plataforma de desenvolvimento de aplicativos completa e segura.

5 Trabalhos Similares

5.1 Sudoku — A Little Lesson in OOP

Neste trabalho, SCHREINER e HELIOTIS apresentam uma série de projetos para um aplicativo de Sudoku que têm como objetivo guiar os alunos iniciantes no importante processo de tentativa e erro. O trabalho também fornece alguns conselhos gerais sobre como evitar erros iniciais no processo de design. Em geral, este trabalho enfatiza a importância

da aprendizagem ativa e do design iterativo para alcançar o sucesso na programação orientada a objetos.

5.2 Sudoku Access: A Sudoku Game for People with Motor Disabilities

O artigo aborda a criação de um jogo de Sudoku acessível para pessoas com deficiência motora, chamado Sudoku Access. NORTE e LOBO desenvolvem uma interface especial que permite o controle do jogo por voz ou por meio de um único botão. É realizada uma pesquisa com usuários para avaliar a eficácia do Sudoku Access, que mostrou que as pessoas podem jogar o jogo com rapidez e precisão. O estudo sugere que esse tipo de jogo pode ajudar mais pessoas com deficiência a se envolverem em jogos de computador e desenvolver habilidades de raciocínio lógico e concentração. O objetivo do trabalho é desenvolver tecnologias que aumentem a independência funcional das pessoas em um ambiente de jogo.

5.4 Genetic Operations to Solve Sudoku Puzzles

SATO e INOUE apresentam, neste artigo, métodos para aplicar algoritmos genéticos na resolução de quebra-cabeças Sudoku. Os autores determinaram que soluções podem ser encontradas com alta probabilidade, mesmo para quebra-cabeças classificados como muito difíceis de resolver pelo método convencional, utilizando algoritmos genéticos.

6 Metodologia

6.1 Definição dos Requisitos

Na definição dos requisitos do aplicativo de Sudoku, a escolha da tecnologia a ser utilizada para o desenvolvimento foi um fator importante a ser considerado. Após uma análise cuidadosa, foi escolhido utilizar o Flutter como plataforma de desenvolvimento, devido à sua flexibilidade e eficiência na criação de aplicativos móveis.

Além disso, o Firebase foi definido como serviço de banco de dados e autenticação do aplicativo. A escolha do Firebase foi baseada na sua facilidade de uso e eficiência na integração com o Flutter.

6.2 Projeto do Aplicativo

O projeto da aplicação foi desenvolvido com a definição da arquitetura do aplicativo, seleção e personalização das bibliotecas e componentes para a interface de usuário, integração com o Firebase para gestão do banco de dados e autenticação, desenvolvimento de testes de usabilidade e desempenho, adoção de um processo iterativo de design e desenvolvimento e manutenção de um cronograma e lista de tarefas para entrega dentro do prazo estabelecido. As melhores práticas de desenvolvimento foram adotadas para garantir que a aplicação atendesse aos requisitos definidos e proporcionasse uma experiência agradável e desafiadora aos usuários.

6.3 Implementação do Aplicativo

A implementação do aplicativo foi realizada utilizando a linguagem de programação Dart e o framework Flutter. Foi utilizado o modelo de arquitetura MVC (***Model-View-Controller***), que separa as camadas de lógica de negócio, apresentação e interação com o usuário. A interface de usuário foi construída utilizando widgets personalizados do Flutter e bibliotecas de design como o Material Design.

Para a geração de puzzles de Sudoku aleatórios, foi implementado um algoritmo que segue as regras do jogo e garante que a solução do puzzle seja única. Os dados dos usuários foram armazenados no *Firebase Realtime Database*, e o *Firebase Authentication* foi utilizado para a autenticação dos usuários.

Durante a implementação, foram realizados testes de usabilidade e desempenho para garantir que a aplicação atendesse aos requisitos definidos. Foram realizadas iterações de desenvolvimento para corrigir eventuais problemas identificados durante os testes.

No geral, a implementação do aplicativo foi bem-sucedida e atendeu aos requisitos definidos, proporcionando uma experiência de usuário agradável e desafiadora.

6.4 Análise dos Resultados

A análise dos resultados consistiu em avaliar a usabilidade e o desempenho do aplicativo de Sudoku desenvolvido em Flutter. Foram realizados testes de usabilidade com um grupo de usuários para identificar eventuais problemas na interface e nas funcionalidades do aplicativo.

Os resultados dos testes de usabilidade foram positivos, com os usuários relatando uma interface intuitiva e fácil de usar. Os usuários também destacaram a qualidade dos puzzles gerados e a diversão proporcionada pela jogabilidade desafiadora do jogo.

Além disso, foram realizados testes de desempenho para avaliar o tempo de carregamento do aplicativo e a eficiência na geração de puzzles. Os resultados indicaram que o aplicativo é capaz de gerar puzzles de forma rápida e eficiente, com um tempo de carregamento satisfatório para os usuários.

7 Desenvolvimento

7.1 Planejamento

O início do processo de desenvolvimento do aplicativo envolve a definição das funcionalidades que o programa deve ter. A partir dessas funcionalidades, é possível estabelecer a organização das entidades do sistema. Uma vez definidas as entidades, é possível avançar para a definição da estrutura geral do aplicativo.

7.2 Funcionalidades

O objetivo principal do aplicativo é desenvolver um sistema que seja capaz de gerar jogos de sudoku, oferecendo ao usuário as funcionalidades de edição, salvamento, jogo e resolução do quebra-cabeças com o auxílio de algoritmos.

Além disso, considerando a necessidade de salvar o progresso do jogo, é essencial implementar um método de autenticação de usuários.

7.2.1 Estrutura dos dados

No desenvolvimento do aplicativo, foi adotada a abordagem de orientação a objetos, visando a criação de entidades eficientes e bem estruturadas. Através desse paradigma, as entidades foram modeladas de forma a encapsular dados e comportamentos relacionados, garantindo uma organização coesa e facilitando a manutenção e reutilização de código. O uso da orientação a objetos permitiu uma implementação mais eficiente das funcionalidades do aplicativo, promovendo a coesão e a modularidade em todo o sistema.

Foram identificadas três entidades principais que desempenham um papel fundamental no funcionamento do sistema.

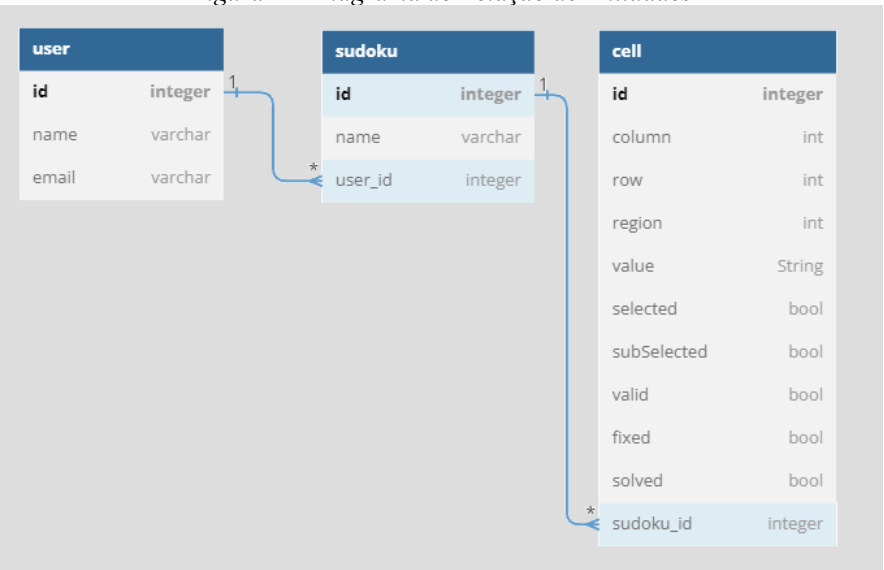
A primeira entidade é a entidade de **usuário**, responsável pela autenticação dos usuários. Essa entidade permite que os usuários se registrem, façam login e tenham acesso às funcionalidades do aplicativo de forma segura. Ela armazena informações como nome de usuário, senha e outras credenciais relevantes.

A segunda entidade é a entidade do **sudoku**, que tem a função de armazenar e gerenciar as informações relacionadas ao próprio quebra-cabeça do Sudoku. Essa entidade registra os valores das células do jogo, o estado do tabuleiro, as configurações específicas do jogo (como o nível de dificuldade) e outras propriedades relevantes. Ela permite a geração de puzzles aleatórios, a validação das soluções propostas pelos usuários e o controle do fluxo de jogo.

A terceira entidade é a entidade **células**, responsável por registrar informações sobre cada célula individual do Sudoku. Essa entidade armazena o valor atual da célula, se ela é editável ou fixa, além de outras propriedades como destaque ou erros de preenchimento. Ela é usada para rastrear e validar as entradas do usuário, bem como para fornecer pistas visuais durante o jogo.

A organização das entidades é demonstrada na Figura 1 através de um diagrama de relação de entidades.

Figura 1 – Diagrama de Relação de Entidades



Fonte: Criado pelo Autor

7.2.2 Padrão de Arquitetura

No desenvolvimento do aplicativo, foi adotado o padrão de arquitetura ***Model-View-Controller*** **(MVC)**. O modelo MVC é amplamente utilizado na construção de aplicativos para separar as preocupações e organizar o código de maneira mais eficiente.

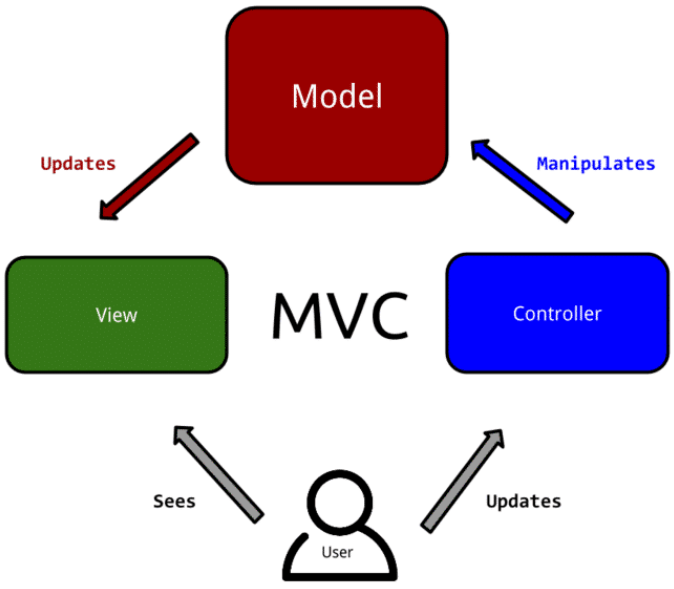
Nesse padrão, conforme ilustrado na Figura 2, o modelo representa a camada de dados e lógica de negócios. Ele é responsável por armazenar e gerenciar as informações relacionadas ao jogo de Sudoku, como o estado do tabuleiro, os valores das células e as regras do jogo. Além disso, o modelo também implementa a lógica para gerar puzzles aleatórios, validar soluções propostas pelos usuários e controlar o fluxo de jogo.

A camada de visualização, ou ***view***, é responsável por exibir a interface do usuário e receber as interações dos usuários. No contexto do aplicativo de Sudoku, a *view* apresenta o tabuleiro do jogo, as células, os botões de interação e outras informações relevantes. Ela também é responsável por notificar o controlador sobre as ações do usuário, como o preenchimento de uma célula ou a solicitação de dicas.

O controlador, ou ***controller***, por sua vez, atua como intermediário entre o modelo e a *view*. Ele recebe as ações do usuário da *view*, realiza as alterações necessárias no modelo e atualiza a *view* de acordo. No caso do aplicativo de Sudoku, o controlador é responsável por receber as entradas do usuário, validar sua correção, atualizar o estado do modelo e notificar a *view* sobre as mudanças a serem exibidas.

A adoção do padrão MVC traz alguns benefícios para o desenvolvimento do aplicativo de Sudoku. Primeiro, ele promove uma separação clara de responsabilidades, facilitando a manutenção e a evolução do código. Segundo, permite uma melhor reutilização de código, já que as diferentes camadas são independentes entre si. Terceiro, facilita a implementação de testes unitários, pois cada camada pode ser testada separadamente.

Figura 2 – Arquitetura MVC



Fonte: MIESSLER, DANIEL.
Disponível em: https://danielmiessler.com/images/MVC1.png.
Acesso em: 25/05/2023

7.3 Models

Os *models* foram criados de acordo com as três entidades principais do aplicativo: **usuário**, **sudoku** e **célula**. Cada entidade possui suas próprias propriedades, que foram definidas seguindo o mesmo modelo utilizado para o banco de dados.

Para a entidade de usuário, foram criadas propriedades como nome, sobrenome, email, senha, entre outras informações relevantes para a autenticação e identificação do usuário. Essas propriedades refletem as colunas correspondentes na tabela de usuários do banco de dados.

Já para a entidade de sudoku, as propriedades foram definidas para armazenar informações sobre o jogo, como o estado do tabuleiro, os números preenchidos e os espaços vazios. Além disso, outras propriedades podem ter sido incluídas para controle de tempo, níveis de dificuldade ou qualquer outra informação relacionada ao jogo de sudoku.

A entidade célula, por sua vez, possui propriedades específicas para registrar informações sobre cada célula do tabuleiro de sudoku. Isso pode incluir o valor da célula, se está preenchida ou vazia, se é uma célula fixa ou pode ser alterada pelo jogador, entre outras características necessárias para o funcionamento do jogo.

Ao desenvolver as propriedades das entidades, foi levado em consideração o mesmo modelo de banco de dados definido para o aplicativo. Isso garante consistência entre a estrutura de dados do aplicativo e o armazenamento das informações no banco, facilitando a persistência dos dados e a integração com o banco de dados.

Ao seguir esse modelo consistente, as propriedades dos models refletem diretamente as colunas e campos do banco de dados, facilitando a manipulação e o acesso aos dados durante a execução do aplicativo.

7.4 Views

Durante o processo de desenvolvimento do aplicativo, foram criadas *views* para as três entidades principais (usuário, sudoku e célula), além das telas de *login* e menu de opções. Essas *views* são responsáveis por apresentar a interface gráfica do aplicativo aos usuários e permitir a interação com as funcionalidades oferecidas.

A *view* de usuário, apresentada na Figura 3, foi projetada para exibir e coletar informações relacionadas ao cadastro e autenticação do usuário. Nessa tela, os usuários podem inserir seus dados, como nome, e-mail e senha, para criar uma conta no aplicativo. Além disso, a *view* de usuário também permite a realização do login, fornecendo acesso seguro às funcionalidades do aplicativo.

A *view* de sudokus, apresentada na Figura 4, foi desenvolvida com o objetivo de permitir que os usuários selecionem entre os sudokus salvos, apresentando informações específicas de cada um. Nessa tela, os usuários têm a capacidade de visualizar detalhes como o nível de dificuldade, data de criação e nome atribuído a cada sudoku.

Essa funcionalidade é especialmente útil para os usuários que desejam retomar jogos anteriores ou escolher sudokus com diferentes níveis de desafio. Ao exibir essas informações, a *view* do sudoku ajuda os usuários a tomarem decisões informadas sobre qual

sudoku desejam jogar, permitindo que eles selecionem um que esteja de acordo com suas preferências e habilidades.

A *view* de célula, apresentanda na Figura 5, foi desenvolvida para mostrar o tabuleiro do jogo de sudoku e permitir que os usuários interajam com as células. Nessa tela, os jogadores podem preencher os números nas células e resolver o quebra-cabeça.

Além das *views* específicas das entidades, foram desenvolvidas também as telas de login e menu seleção. A tela de login, apresentanda na Figura 6, oferece aos usuários a possibilidade de inserir suas credenciais para acessar o aplicativo. Já o menu de seleção, apresentado na Figura 7, permite aos usuários navegar pelas diferentes funcionalidades oferecidas.

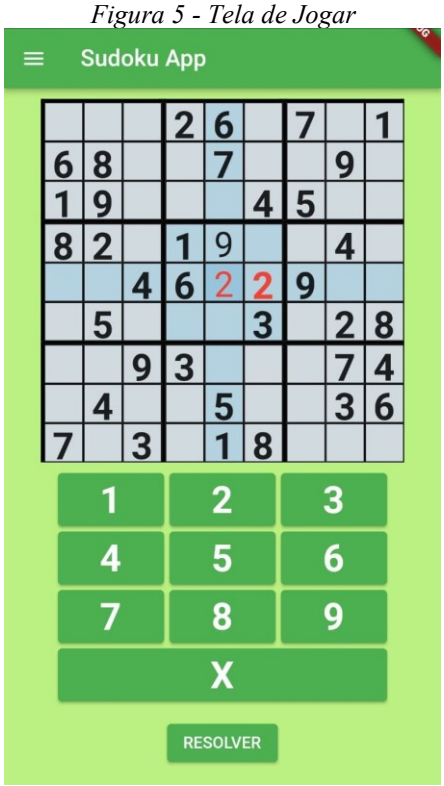
Figura 3 - Tela Usuário

Fonte: Criado pelo Autor

Figura 4 – Tela Escolha de Sudoku

Fonte: Criado pelo Autor

Figura 5 - Tela de Jogar

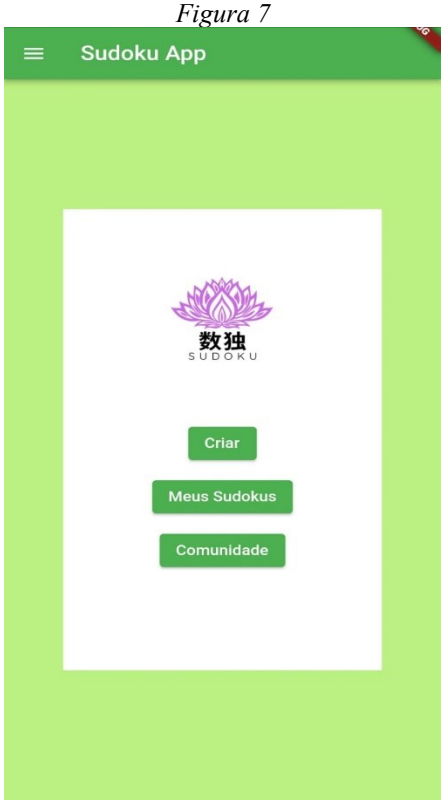


Fonte: Criado pelo Autor

Figura 6 - Tela de Login

Fonte: Criado pelo Autor

Figura 7



Fonte: Criado pelo Autor

7.5 Controllers

Foram desenvolvidos *controllers* para as três entidades principais do aplicativo: usuário, sudoku e célula. Os *controllers* desempenham um papel fundamental na lógica de funcionamento do aplicativo, sendo responsáveis por receber e processar as interações dos usuários, bem como gerenciar os dados e a lógica de negócio relacionados a cada entidade.

O *controller* de usuário tem a responsabilidade de lidar com as funcionalidades relacionadas à autenticação e gerenciamento de contas dos usuários. Ele recebe os dados inseridos pelos usuários nas telas de cadastro e login, verifica sua validade e autentica os usuários, fornecendo acesso seguro às funcionalidades do aplicativo. Além disso, o *controller* de usuário também gerencia o armazenamento e a recuperação das informações dos usuários, garantindo a integridade e a segurança dos dados.

O *controller* do sudoku é responsável por uma tarefa fundamental: carregar os valores das células. Ele é responsável por recuperar as informações armazenadas nas células do sudoku, como os números preenchidos e as células vazias. Essas informações são essenciais para apresentar o estado atual do jogo aos usuários.

O *controller* de célula é responsável por controlar as interações específicas das células do sudoku. Ele recebe as ações dos usuários nas células, como seleção, edição e visualização de informações detalhadas. O *controller* de célula também lida com a lógica de validação dos números inseridos pelo usuário em cada célula, garantindo a consistência e a conformidade com as regras do sudoku. Ele também coordena o armazenamento e a recuperação das informações das células, garantindo que os dados sejam mantidos corretamente.

Ao criar os *controllers* para as entidades principais, buscou-se uma estrutura organizada e coesa, permitindo um controle eficiente das funcionalidades do aplicativo. Esses *controllers* desempenham um papel fundamental no funcionamento adequado do aplicativo de sudoku em Flutter, garantindo uma experiência de jogo fluida e consistente para os usuários.

7.6 Autenticação

O sistema de autenticação do aplicativo foi implementado utilizando o Firebase. O Firebase oferece a biblioteca ***firebase_auth.dart*** para Flutter, que simplifica o processo de autenticação. Essa biblioteca permite organizar todas as operações relacionadas à autenticação em um único *controller*, responsável por lidar com o registro de usuários, o login e o gerenciamento de sessões. A integração com esta biblioteca proporciona uma solução eficiente e confiável para o sistema de autenticação do aplicativo, garantindo a segurança e a privacidade dos usuários. Além disso, o Firebase também oferece recursos avançados, como autenticação por meio de redes sociais, ampliando as opções de login disponíveis para os usuários. Com essa integração, o aplicativo oferece uma experiência de autenticação simplificada e segura, permitindo que apenas usuários autenticados acessem os recursos e as funcionalidades do aplicativo.

O trecho do código que utiliza a biblioteca ***firebase_auth.dart*** é apresentado na Figura 8.

Figura 8 - Trecho do Código de Autenticação

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:sudoku_app/main.dart';
import '../models/userModel.dart';

class AuthService {
  final FirebaseAuth _auth = FirebaseAuth.instance;

  MyUser? _userfromFirebase(User user) {
    return MyUser(uid: user.uid);
  }

  Stream<MyUser?> get user {
    return _auth
        .authStateChanges()
        .map((User? user) => _userfromFirebase(user!));
  }
```

Fonte: Criado pelo Autor

7.7 Banco de Dados

O banco de dados do aplicativo foi implementado utilizando o serviço FireStore do Firebase. O FireStore é um banco de dados não relacional, o que oferece diversas vantagens para a aplicação. Diferente de um banco de dados relacional, onde os dados são organizados em tabelas com relações complexas, o FireStore adota um modelo de documentos, onde os dados são armazenados em coleções de documentos individuais. Isso proporciona maior flexibilidade na estrutura dos dados, permitindo a adição e a modificação de campos de forma mais dinâmica. Além disso, o FireStore oferece escalabilidade automática, ou seja, a capacidade de armazenamento e o desempenho do banco de dados são dimensionados automaticamente conforme a demanda, garantindo uma experiência consistente mesmo com muitos usuários. Outra vantagem é a sincronização em tempo real, onde as atualizações feitas por um usuário são imediatamente refletidas para os demais usuários, possibilitando uma experiência colaborativa e em tempo real. Essas características do FireStore tornam-no uma escolha ideal para o aplicativo, garantindo um armazenamento eficiente, escalável e ágil dos dados.

Um trecho do código utilizado para importar os Sudokus de um determinado usuário no FireStore é apresentado na Figura 9.

Figura 9 - Trecho do código de acesso ao banco de dados

```
import 'package:cloud_firestore/cloud_firestore.dart';
import '../models/sudoku.model.dart';

class SudokuController {
  final sudokus = [];

  void addSudoku(Sudoku sudoku) {
    sudokus.add(sudoku);
  }

  void importSudokus() async {
    CollectionReference sudokuColl =
        FirebaseFirestore.instance.collection('sudokus');
    sudokuColl.get().then((QuerySnapshot querySnapshot) {
      for (var doc in querySnapshot.docs) {
        var sudoku = Sudoku(doc['owner'], doc['cells'], doc['name']);
        addSudoku(sudoku);
      }
    });
  }
}
```

Fonte: Criado pelo Autor

7.8 Sudoku

O layout de um jogo de Sudoku é composto por uma grade 9x9, dividida em nove blocos menores de 3x3. Cada célula da grade pode conter um número de 1 a 9. O objetivo do jogo é preencher todas as células da grade de forma que cada número de 1 a 9 apareça apenas uma vez em cada linha, coluna e bloco de 3x3.

Existem duas categorias de números em um jogo de Sudoku: os números fixos e os números preenchidos pelo usuário. Os números fixos são aqueles que já estão inicialmente preenchidos no tabuleiro e não podem ser alterados. Eles fornecem as pistas iniciais para resolver o quebra-cabeça. Geralmente, os números fixos são definidos estrategicamente para garantir que haja apenas uma solução possível para o Sudoku.

Por outro lado, os números preenchidos pelo usuário são aqueles que o jogador insere durante o jogo. Eles representam as tentativas do jogador para resolver o quebra-cabeça. Os números preenchidos pelo usuário podem ser modificados ou removidos a qualquer momento, permitindo ao jogador experimentar diferentes possibilidades até encontrar a combinação correta.

7.8.1 Grid

Para o desenvolvimento da criação do grid do Sudoku foi utilizado a classe *InkWell* para gerar as células. A escolha dessa classe se deu por alguns motivos importantes. Primeiramente, a classe *InkWell* permite adicionar interatividade às células, permitindo que o jogador clique em uma célula específica para inserir um número. Além disso, ela oferece recursos de detecção de toque e feedback visual, tornando a experiência do usuário mais intuitiva e agradável.

Em vez de criar cada célula do grid individualmente, optou-se por desenvolver uma função recursiva para replicar múltiplas colunas, linhas e áreas do grid do Sudoku. Essa abordagem proporcionou um desenvolvimento mais eficiente e escalável, reduzindo a necessidade de escrever código repetitivo. A função recursiva permite a criação dinâmica das células do Sudoku, facilitando a manutenção e a expansão do código.

Ao implementar a função recursiva, foram definidos os parâmetros necessários para controlar o número de colunas, linhas e áreas a serem criadas. A função utiliza loops e condições para gerar as células de forma estruturada, garantindo que o grid do Sudoku seja criado corretamente e respeitando as regras do jogo.

O trecho do código responsável pela criação individual de cada célula é apresentado na Figura 10.

Figura 10 - Criação das Células em componente InkWell

```
Container createCell(int region, int row, int column) {
  cellId += 1;
  var id = cellId;

  return Container(
    decoration: cellDecoration(id, cellController),
    child: SizedBox(
      width: cellWidth,
      height: cellHeight,
      child: InkWell(
          onTap: () {
            var cell = cellController.getCell(id);
            if (!cell.fixed) {
              setState(
                  () => {cellController.selectCell(id), updateBody(context)});
            } else {
              setState(() {
                cellController.clearSelected();
                updateBody(context);
              });
            }
          },
          child: Center(
              child: Text(
            cellController.getValueById(id),
            style: cellTextStyle(id, cellController),
          ))), // Text // Center // InkWell
    ), // SizedBox
  ); // Container
}
```

Fonte: Criado pelo Autor

7.8.2 Criar e Editar

No desenvolvimento do aplicativo de Sudoku, foi criada uma tela específica para a criação e edição do jogo. Essa tela permite que o usuário posicione os números no grid do Sudoku de acordo com sua preferência. Ao posicionar um número, ele é salvo no banco de dados com a propriedade booleana "*fixed*", indicando que aquele número foi definido pelo próprio usuário.

A utilização da propriedade booleana "*fixed*" é importante para distinguir os números fixos dos números de preenchimento. Isso permite que o jogo identifique corretamente quais números podem ser alterados pelo jogador durante a resolução do Sudoku e quais são fixos e devem ser mantidos inalterados.

Dessa forma, ao salvar os números com a propriedade booleana "*fixed*", o aplicativo garante que os números posicionados pelo usuário sejam preservados durante a jogabilidade, evitando que o jogador acidentalmente altere números já posicionados. Além disso, a propriedade booleana "*fixed*" é utilizada para realizar a validação do puzzle e verificar se a solução proposta pelo usuário está correta.

7.8.3 Salvar

No processo de salvar o Sudoku no aplicativo, são utilizadas duas entidades distintas: a entidade "sudoku" e a entidade "*cells*". A entidade "sudoku" armazena informações gerais sobre o jogo, como nível de dificuldade e data de criação, enquanto a entidade "*cells*" é responsável por armazenar os valores das células do Sudoku.

A entidade "*cells*" é salva como um objeto no banco de dados, utilizando um modelo não relacional. Essa abordagem permite que as células do Sudoku sejam representadas de forma mais flexível, sem a necessidade de seguir uma estrutura rígida de tabelas e colunas. Cada célula é tratada como um objeto independente, facilitando a manipulação e atualização dos valores.

Além disso, a entidade "sudoku" possui uma propriedade chamada "*userId*", que é utilizada para identificar o dono do Sudoku. Essa propriedade permite que o aplicativo associe corretamente cada Sudoku a seu respectivo proprietário, possibilitando o controle e gerenciamento dos jogos criados por cada usuário.

Ao salvar as informações nas entidades "sudoku" e "*cells*", o aplicativo garante a integridade dos dados do Sudoku, mantendo uma estrutura organizada e consistente. Isso permite que os jogadores possam salvar e recuperar seus jogos de forma confiável, possibilitando a continuidade e a retomada do jogo a qualquer momento.

7.8.4 Jogar

No modo "Jogar" do aplicativo, os jogadores têm a capacidade de interagir com o Sudoku e selecionar individualmente cada célula do tabuleiro. Essa interação é possibilitada pela função “onTap()” da classe *InkWell*, que é ativada quando o usuário toca em uma célula específica.

Ao utilizar a função “onTap()”, é acionado um método que tem a responsabilidade de modificar a propriedade “selected” da célula selecionada. Essa propriedade permite destacar visualmente a célula selecionada, facilitando a visualização por parte do jogador.

Além disso, o método também remove a propriedade "selected" das células que foram previamente selecionadas. Isso garante que apenas uma célula esteja selecionada de cada vez, evitando ambiguidades ou erros na manipulação do Sudoku.

Para o preenchimento das células do Sudoku foi criado um teclado virtual utilizando a classe *ElevatedButton*. A escolha por criar um teclado virtual próprio, em vez de utilizar o teclado nativo do celular, deve-se à utilização da classe *InkWell* para as células do Sudoku.

A classe *InkWell* é uma classe de interação que oferece recursos específicos para detecção de toques em elementos gráficos, porém, não se trata de uma classe de preenchimento de texto. Portanto, para permitir que o usuário possa inserir os números nas células do Sudoku, foi necessário criar um teclado virtual personalizado.

O teclado virtual desenvolvido utiliza a classe *ElevatedButton* para representar os botões numéricos, permitindo ao jogador escolher o número desejado e preencher a célula selecionada.

7.8.5 Algoritmo de Validação

O algoritmo de validação do valor da célula utiliza a função de mapeamento “*Where”* do Flutter para procurar se existe outra célula na mesma área, linha ou coluna com o mesmo valor da célula selecionada. Esse algoritmo é implementado de forma a verificar a validade do valor escolhido pelo jogador de acordo com as regras do Sudoku.

No início, cada célula do tabuleiro do Sudoku é armazenada com as propriedades '*value*', “region”, “*row”* e “*column*”. Essas propriedades são importantes para facilitar a validação de cada célula individualmente. Apesar de parecer redundante armazenar essas propriedades para cada célula, isso torna o algoritmo de validação mais rápido e eficiente.

Quando o jogador insere um valor em uma célula específica, o algoritmo de validação é acionado. Ele utiliza a função *“Where”* para procurar em todas as outras células se existe alguma que pertença à mesma região, linha ou coluna e possua o mesmo valor da célula selecionada.

Ao encontrar uma célula com as mesmas características, o algoritmo identifica uma duplicação e marca a célula como inválida. Isso permite que o jogador corrija o valor inserido na célula antes de prosseguir com o jogo.

Esse algoritmo de validação baseado na função *“Where”* do Flutter é eficiente porque utiliza as informações de localização das células no tabuleiro para evitar a necessidade de percorrer todas as células a cada validação. Dessa forma, ele reduz a complexidade computacional e melhora o desempenho geral do jogo.

A implementação do algoritmo de validação é apresentado na Figura 11.

Figura 11 – Código de validação das células

```
void validateCell(Cell cell) {
  var subCells = cells.where((c) =>
      (c.value == cell.value) &&
      (c.region == cell.region ||
          c.row == cell.row ||
          c.column == cell.column) &&
      (c.id != cell.id));
  if (subCells.isEmpty) {
    cell.valid = true;
  } else {
    cell.valid = false;
  }
}
```

Fonte: Criado pelo Autor

7.8.6 Algoritmos de Resolução

Resolver Sudokus é uma área de pesquisa tanto na ciência da computação quanto na matemática, envolvendo aspectos como a resolução de problemas, a classificação da dificuldade dos quebra-cabeças e a geração desses quebra-cabeças. O problema de resolver Sudokus de tamanho $n^2 * n^2$ é considerado NP-completo (BERGGREN e NILSSON, 2012).

Além de ser teoricamente interessante, isso também tem motivado pesquisas em heurísticas, resultando em uma ampla variedade de métodos de resolução disponíveis. Alguns dos algoritmos de resolução existentes incluem:

- ***Backtracking***: Também conhecido como "tentativa e erro", o algoritmo de *backtracking* começa preenchendo a primeira célula vazia com um número válido. Em seguida, ele avança para a próxima célula e repete o processo. Se o algoritmo encontrar uma situação em que nenhum número seja válido, ele retorna à célula anterior e tenta um número diferente. Esse processo é repetido até que uma solução seja encontrada.
- **Regras Fixas**: Esse método aplica regras lógicas condicionais para excluir combinações numéricas inválidas para um determinado conjunto de células. Este algoritmo é semelhante ao método que seres humanos geralmente utilizam para resolver Sudokus.
- **Máquina de Boltzmann**: O algoritmo de máquina de Boltzmann modela o Sudoku usando uma rede neural artificial de resolução de restrições. Os quebra-cabeças são vistos como restrições que descrevem quais nós não podem ser conectados entre si. Essas restrições são codificadas nos pesos de uma rede neural artificial e, em seguida, resolvidas até que uma solução válida apareça, com nós ativos indicando os dígitos escolhidos.
- **Bogosort**: O *Bogosort* é um algoritmo de ordenação aleatória que envolve embaralhar repetidamente uma lista até que ela esteja ordenada. No contexto do

> Sudoku, isso significa embaralhar repetidamente as células até que todas as células estejam preenchidas corretamente. Embora improvável, existe uma possibilidade não nula de que o algoritmo *Bogosort* seja o mais rápido para resolver um determinado quebra-cabeça de Sudoku (GRUBER, HOLZER e RUEPP, 2017).

Neste projeto, optou-se por utilizar o algoritmo de backtracking para resolver Sudokus devido a suas vantagens significativas. O backtracking é conhecido por sua eficiência, permitindo a exploração de maneira inteligente das diferentes possibilidades, especialmente quando o número de soluções é limitado (BERGGREN e NILSSON, 2012).

Além disso, sua implementação é relativamente simples, facilitando o desenvolvimento e a manutenção do código. O algoritmo de backtracking também garante uma solução correta para o Sudoku, desde que uma exista, e pode ser adaptado para lidar com diferentes variantes e tamanhos de Sudokus. Sua versatilidade e confiabilidade tornaram-no a escolha ideal para abordar os desafios de resolução de Sudokus neste projeto.

7.8.7 Implementação do Algoritmo De Resolução

O algoritmo começa examinando cada célula do Sudoku. Para cada célula, verifica-se se ela já possui um valor fixo ou se está vazia. Se a célula tiver um valor fixo, o algoritmo avança para a próxima célula.

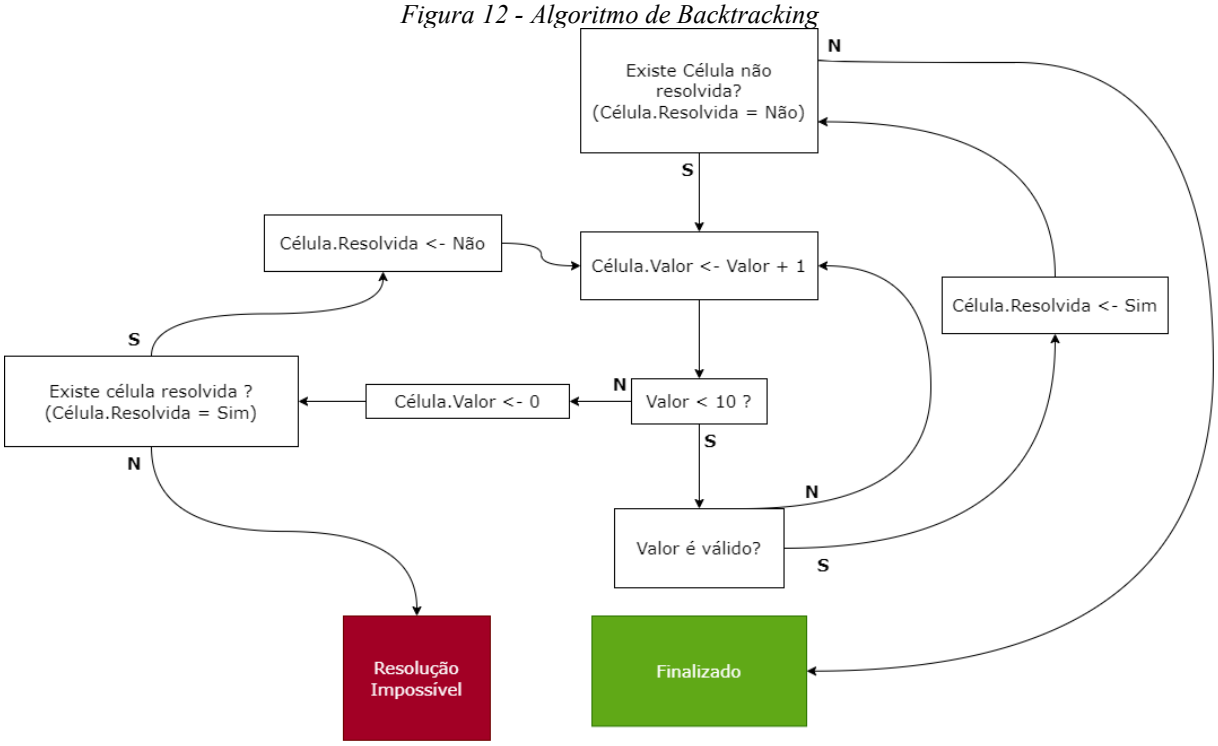
No entanto, se a célula estiver vazia, o algoritmo tentará atribuir um valor válido a ela. Ele percorrerá os possíveis valores, de acordo com as restrições do jogo, como a exclusão de números repetidos em cada linha, coluna e região.

Para atribuir um valor a uma célula, o algoritmo verifica se o número selecionado é válido em relação às outras células já preenchidas. Ele leva em consideração as propriedades *"value"*, que armazena o valor da célula, *"fixed"*, que indica se o valor é fixo ou não, e *"solved"*, que indica se a célula faz parte da solução final.

Se em algum momento o algoritmo encontrar uma contradição, ou seja, não conseguir atribuir um valor válido a uma célula, ele fará um retrocesso para a célula anterior e tentará um valor diferente. Esse processo de retrocesso e tentativa continua até que todas as células sejam preenchidas corretamente e uma solução válida seja encontrada.

O algoritmo de resolução é apresentado na Figura 12.

Figura 12 - Algoritmo de Backtracking



Fonte: Criado pelo Autor

8 Conclusão

Em conclusão, este trabalho foi bem-sucedido em alcançar os objetivos propostos e oferecer um aplicativo de Sudoku abrangente e personalizável. A organização do código seguindo o padrão de arquitetura MVC (Model-View-Controller) proporcionou uma estrutura clara e modular, facilitando a manutenção e a expansão do aplicativo.

A aplicação dos princípios de orientação a objetos permitiu uma abordagem mais eficiente e flexível no desenvolvimento do aplicativo. Através da encapsulação, herança e polimorfismo, conseguimos criar uma base sólida para a manipulação das células, tabuleiros e lógica do jogo, tornando o código mais legível, reutilizável e escalável.

A capacidade do aplicativo de criar e resolver Sudokus customizáveis é um diferencial importante. Os usuários têm a liberdade de criar seus próprios desafios e personalizar os níveis de dificuldade, proporcionando uma experiência de jogo mais envolvente e adaptada às suas preferências.

A resolução automática de Sudokus pelo aplicativo, utilizando o algoritmo de *backtracking*, oferece aos usuários uma solução rápida e confiável para os quebra-cabeças mais difíceis. Isso amplia o alcance do aplicativo, atendendo tanto a jogadores casuais que desejam uma solução rápida quanto a entusiastas que desejam aprimorar suas habilidades no jogo.

Em resumo, a combinação da organização MVC, a abordagem orientada a objetos e a capacidade do aplicativo de criar e resolver Sudokus customizáveis resultou em um aplicativo completo, intuitivo e flexível.

Como sugestões para trabalhos futuros, destacam-se a implementação de modos multijogadores, o suporte para diferentes tamanhos de grades e o aprimoramento do algoritmo de resolução.

A adição de modos multijogadores permitiria que os usuários desafiassem seus amigos ou jogadores de todo o mundo em partidas de Sudoku competitivas. Isso poderia incluir recursos como partidas em tempo real, chat integrado e tabelas de classificação online, proporcionando uma experiência social e interativa.

Além disso, a expansão do suporte para diferentes tamanhos de grades ofereceria mais variedade e desafios aos jogadores. A inclusão de Sudokus com tamanhos de grade não convencionais, como 6x6, 12x12 ou até mesmo Sudokus irregulares, abriria novas possibilidades e estimularia a criatividade dos jogadores.

Por fim, o aprimoramento do algoritmo de resolução é uma área de pesquisa contínua. Explorar técnicas avançadas, como algoritmos baseados em inteligência artificial, heurísticas ou otimizações, poderia resultar em tempos de resolução mais rápidos e eficientes, aprimorando a experiência do usuário.

9 Referências

NORTE, S., & LOBO, F. G. (2008). SUDOKU ACCESS: A SUDOKU GAME FOR PEOPLE WITH MOTOR DISABILITIES.**PROCEEDINGS OF THE 10TH INTERNATIONAL ACM SIGACCESS CONFERENCE ON COMPUTERS AND ACCESSIBILITY** (PP. 251-252). DOI: 10.1145/1414471.1414502.

SATO, Y.; INOUE, H. GENETIC OPERATIONS TO SOLVE SUDOKU PUZZLES. **GENETIC AND EVOLUTIONARY COMPUTATION CONFERENCE**, 12., 2010, PORTLAND. PROCEEDINGS […]. NEW YORK: ACM, 2010. P. 1265-1272. DOI: 10.1145/1830761.1830887.

SCHREINER, A. T., & HELIOTIS, J. E. (2008). SUDOKU — A LITTLE LESSON IN OOP. **ACM SIGCSE BULLETIN,** 40(1), 249-253. DOI: 10.1145/1383602.1383632.

BERGGREN, P., & NILSSON, D. (2012). A STUDY OF SUDOKU SOLVING ALGORITHMS. **KTH ROYAL INSTITUTE OF TECHNOLOGY**. DISPONÍVEL EM: <WWW.CSC.KTH.SE/UTBILDNING/KANDIDATEXJOBB/DATATEKNIK/2012/BERGGREN_PATRIK_OCH_NILSSON_DAVID_K12011.PDF>. ACESSO EM: 20 DE MAI. DE 2023.

GRUBER, H., HOLZER, M., & RUEPP, O. (2007). SORTING THE SLOW WAY: AN ANALYSIS OF PERVERSELY AWFUL RANDOMIZED SORTING ALGORITHMS. IN LECTURE NOTES IN COMPUTER SCIENCE (PP. 183–197**). SPRINGER BERLIN HEIDELBERG**. DOI: /10.1007/978-3-540-72914-3_17