Pontifícia Universidade Católica de Minas Gerais

Amanda Canizela Guimarães
Antonella de Paula Menegaz
Felipe de Faria Rios Coelho
Lucas Alvarenga Fernandes
Lucca Mendes Alves Pellegrini

Jogo da Velha

Engenharia de Computação

Belo Horizonte
2022

RESUMO

O projeto consiste em um jogo da velha com 3 modos de jogo (jogador vs jogador, jogador vs computador e computador vs computador), em que é possível decidir quem irá fazer o primeiro movimento e qual será a dificuldade da rodada (fácil, médio ou difícil). Após as escolhas feitas por meio do aplicativo, o jogador irá utilizar o botão amarelo da direita, presente no próprio tabuleiro, para escolher em qual das 9 casas posicionar sua primeira jogada (a cor dos leds ficará verde para o primeiro jogador) e o botão amarelo da esquerda para confirmar sua jogada. Com isso, a jogada estará concluída e será a vez do segundo jogador. Caso o jogador seja o computador, ele irá posicionar sua jogada automaticamente. Quando algum dos jogadores conseguir alinhar 3 das jogadas em uma linha, em uma coluna ou na diagonal e, assim, vencer a rodada, os leds da cor do campeão irão piscar para anunciar quem venceu. Assim que uma partida acaba, outra se inicia automaticamente.

Palavras-chave: Jogador; Computador; Jogo da velha;

1 INTRODUÇÃO

O projeto apresenta o objetivo de desenvolver uma aplicação que integra o APP Inventor, o ESP-32 e um Smartphone para criar um jogo da velha inteligente. Esse trabalho se encaixa no contexto de jogos eletrônicos iterativos em que há a implementação de um jogo muito conhecido mundialmente a uma tecnologia capaz de complementar o jogo com outras funcionalidades, como o nível de dificuldade e a opção de competir com um computador.

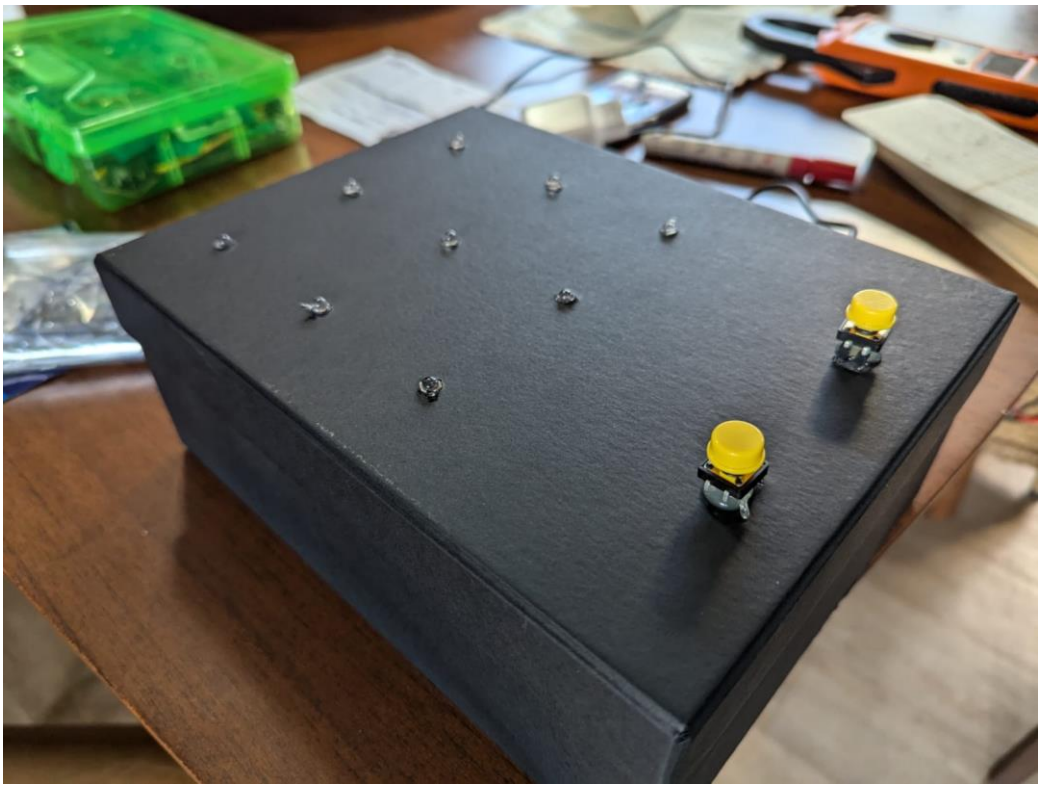
O aplicativo criado por meio do APP inventor apresenta opções para a escolha dos oponentes (jogador vs jogador, jogador vs computador e computador vs computador), das dificuldades (fácil, médio ou difícil) e do primeiro jogador. Após as escolhas, um placar irá aparecer na tela, assim como, a opção de desistir e reiniciar.

O tabuleiro, composto por uma caixa com 9 leds e dois botões (o primeiro para movimentar o led e o segundo para concluir a jogada), é onde o jogo realmente acontece, sendo possível fazer e analisar as jogadas.

2 DESENVOLVIMENTO

O tabuleiro do jogo, na parte externa, é composto por 9 leds e por dois botões. Cada botão apresenta uma função: o da direita é utilizado para que o jogador escolha qual casa posicionar a jogada e o da esquerda é utilizado para que o jogador aperte assim que o led acender na casa desejada. Já os leds acenderão com a cor verde para o primeiro jogador e com a cor vermelha para o segundo

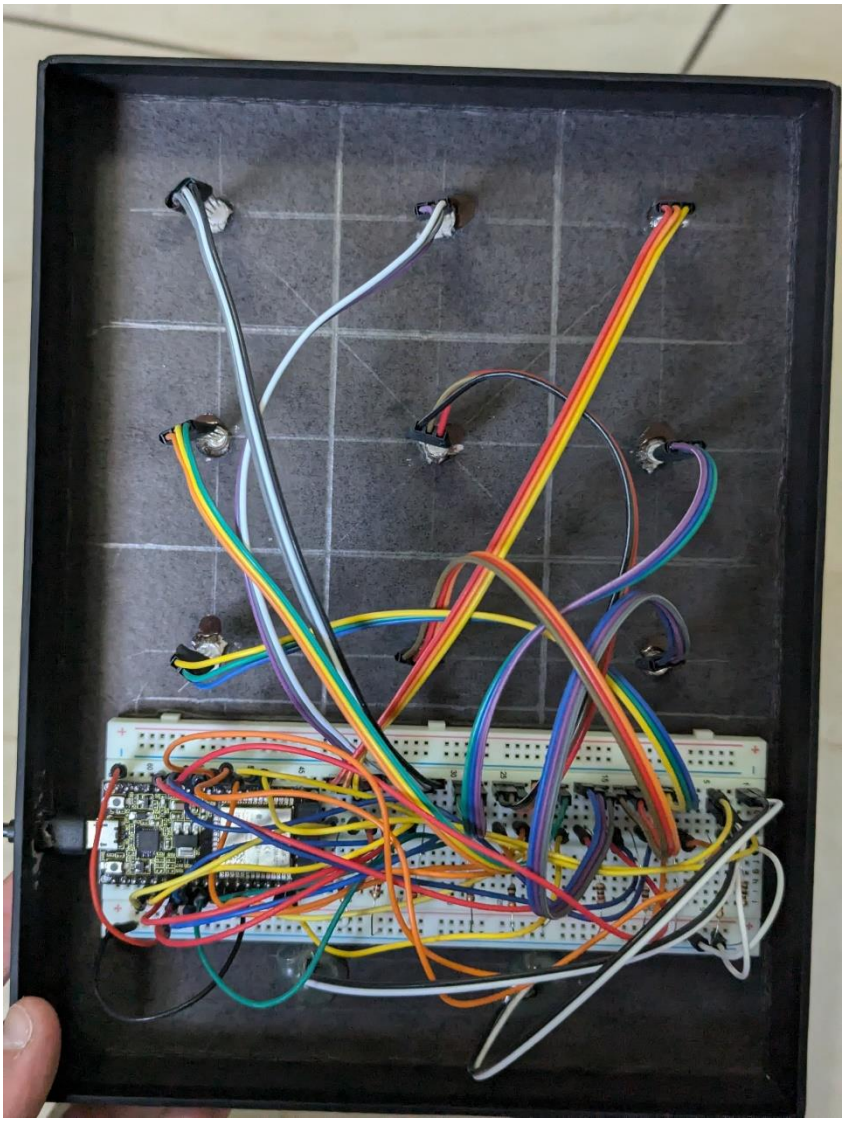
Imagem 1 – Parte externa do tabuleiro do Jogo da Velha montado



Fonte: (Foto tirada por Lucca Mendes Alves Pellegrini)

Na parte interna do Tabuleiro, a placa de ensaio, junto ao o ESP-32, foi fixada a tampa e os fios presentes estão ligados aos leds e aos botões para que o jogo funcione.

Imagem 2 – Parte interna do tabuleiro do Jogo da Velha montado



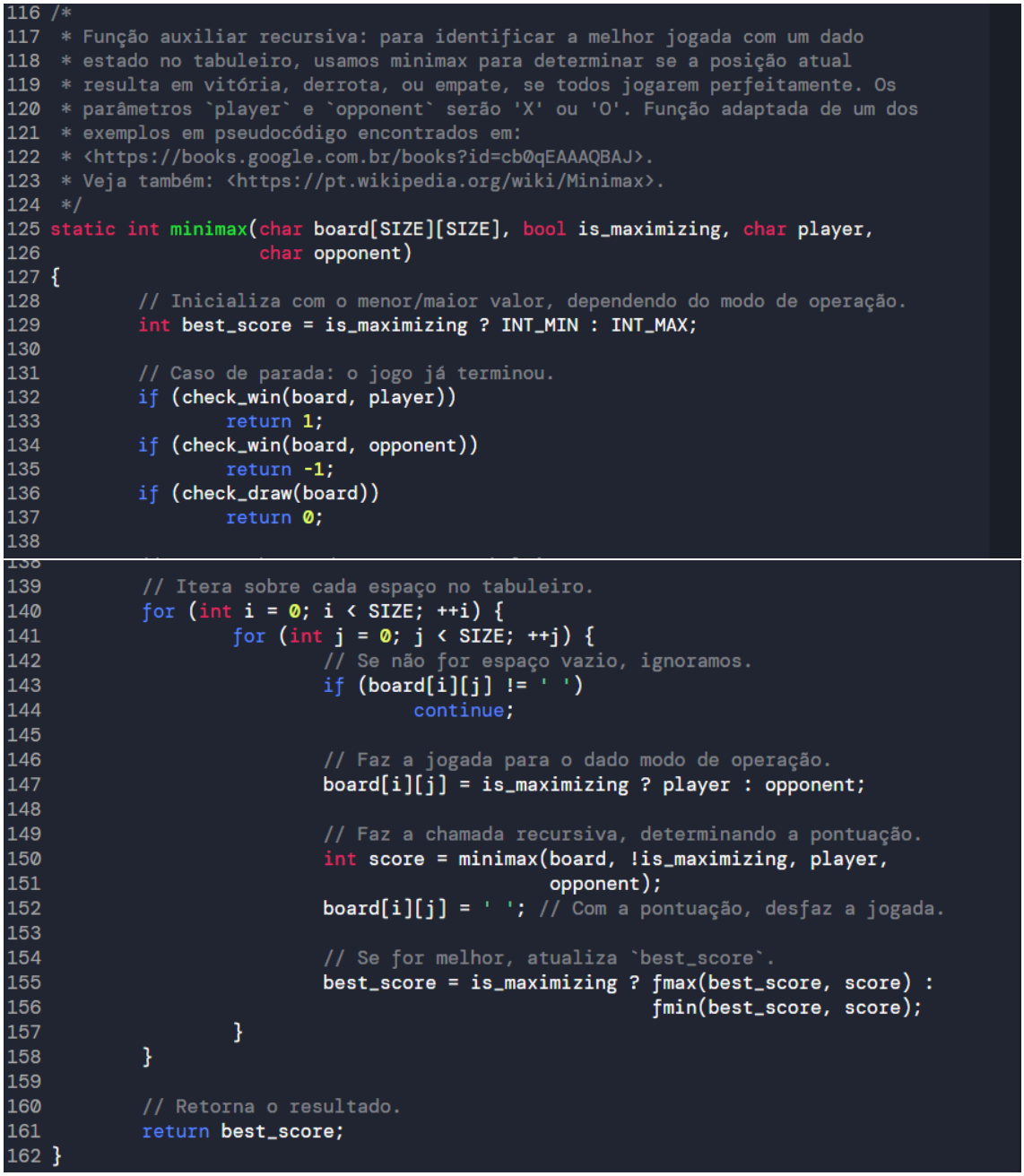
Fonte: (Foto tirada por Lucca Mendes Alves Pellegrini)

O código do Jogo da Velha faz com que o computador possa reconhecer as jogadas e indicar qual foi o jogador vencedor, por meio do reconhecimento de qual deles conseguiu completar três casas em sequência (na coluna, na linha ou na diagonal). Além disso, o código torna possível que o oponente do jogador seja o computador.

Imagem 3 – Código do jogo da velha

```c
#include <assert.h>
#include <limits.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "board.h"
#include "cpu.h"

// Protótipos das funções internas deste arquivo.
static void easy_move(char board[SIZE][SIZE], char player);
static void medium_move(char board[SIZE][SIZE], char player);
static void hard_move(char board[SIZE][SIZE], char player);
static int minimax(char board[SIZE][SIZE], bool is_maximizing, char player,
                   char opponent);

// Função pública. Recebe a dificuldade e invoca o procedimento adequado.
void cpu_move(char board[SIZE][SIZE], char player, Difficulty difficulty)
{
        switch (difficulty) {
        case EASY:
                easy_move(board, player);
                break;
        case NORMAL:
                medium_move(board, player);
                break;
        case HARD:
                hard_move(board, player);
                break;
        default: // Dificuldade inválida.
                fprintf(stderr, "Erro: %s recebeu dificuldade inválida\n",
                        __func__); // Mensagem de erro.
                assert(false); // Crasha o programa.
        }
}

// Modo fácil: jogada aleatória.
static void easy_move(char board[SIZE][SIZE], char player)
{
        int row, col;
        do { // Loop garante jogada válida.
                row = rand() % SIZE;
                col = rand() % SIZE;
        } while (!make_move(board, row, col, player));
}

// Modo normal: evita erros graves e toma oportunidade de ganhar.
static void medium_move(char board[SIZE][SIZE], char player)
{
        char opponent = (player == 'X') ? 'O' : 'X'; // Quem é o oponente.

        // Verifica se alguma jogada nos faz ganhar imediatamente.
        for (int i = 0; i < SIZE; ++i) {
                for (int j = 0; j < SIZE; ++j) {
                        if (board[i][j] != ' ')
                                continue;

                        board[i][j] = player;
                        if (check_win(board, player))
                                return;
                        board[i][j] = ' ';
                }
        }
```

```
65          // Bloqueia jogada do oponente que o faria ganhar, se existir.
66          for (int i = 0; i < SIZE; ++i) {
67                  for (int j = 0; j < SIZE; ++j) {
68                          if (board[i][j] != ' ')
69                                  continue;
70
71                          board[i][j] = opponent;
72                          if (check_win(board, opponent)) {
73                                  board[i][j] = player;
74                                  return;
75                          }
76                          board[i][j] = ' ';
77                  }
78          }
79
80          // Caso contrário, faz uma jogada aleatória.
81          easy_move(board, player);
82  }
83
84  // Modo difícil: Usa minimax para determinar a melhor jogada possível.
85  static void hard_move(char board[SIZE][SIZE], char player)
86  {
87          char opponent = (player == 'X') ? 'O' : 'X'; // Quem é o oponente.
88          int best_score = INT_MIN; // Melhor resultado possível. (1, 0, ou -1.)
89          int best_row = -1, best_col = -1; // Coordenadas da melhor jogada.
90
91          // Itera sobre cada espaço no tabuleiro.
92          for (int i = 0; i < SIZE; ++i) {
93                  for (int j = 0; j < SIZE; ++j) {
94                          // Se não for espaço vazio, ignoramos.
95                          if (board[i][j] != ' ')
96                                  continue;
97
98                          // Para cada jogada, determina a pontuação com minimax.
99                          board[i][j] = player;
100                         int score = minimax(board, false, player, opponent);
101                         board[i][j] = ' '; // Com a pontuação, desfaz a jogada.
102
103                         // Se for melhor, atualiza as coordenadas.
104                         if (score > best_score) {
105                                 best_score = score;
106                                 best_row = i;
107                                 best_col = j;
108                         }
109                 }
110         }
111
112         // Executa a melhor jogada, que determinamos com o minimax.
113         make_move(board, best_row, best_col, player);
114 }
115
```



```c
/*
 * Função auxiliar recursiva: para identificar a melhor jogada com um dado
 * estado no tabuleiro, usamos minimax para determinar se a posição atual
 * resulta em vitória, derrota, ou empate, se todos jogarem perfeitamente. Os
 * parâmetros `player` e `opponent` serão 'X' ou 'O'. Função adaptada de um dos
 * exemplos em pseudocódigo encontrados em:
 * <https://books.google.com.br/books?id=cb0qEAAAQBAJ>.
 * Veja também: <https://pt.wikipedia.org/wiki/Minimax>.
 */
static int minimax(char board[SIZE][SIZE], bool is_maximizing, char player,
                   char opponent)
{
        // Inicializa com o menor/maior valor, dependendo do modo de operação.
        int best_score = is_maximizing ? INT_MIN : INT_MAX;

        // Caso de parada: o jogo já terminou.
        if (check_win(board, player))
                return 1;
        if (check_win(board, opponent))
                return -1;
        if (check_draw(board))
                return 0;

        // Itera sobre cada espaço no tabuleiro.
        for (int i = 0; i < SIZE; ++i) {
                for (int j = 0; j < SIZE; ++j) {
                        // Se não for espaço vazio, ignoramos.
                        if (board[i][j] != ' ')
                                continue;

                        // Faz a jogada para o dado modo de operação.
                        board[i][j] = is_maximizing ? player : opponent;

                        // Faz a chamada recursiva, determinando a pontuação.
                        int score = minimax(board, !is_maximizing, player,
                                            opponent);
                        board[i][j] = ' '; // Com a pontuação, desfaz a jogada.

                        // Se for melhor, atualiza `best_score`.
                        best_score = is_maximizing ? fmax(best_score, score) :
                                                     fmin(best_score, score);
                }
        }

        // Retorna o resultado.
        return best_score;
}
```

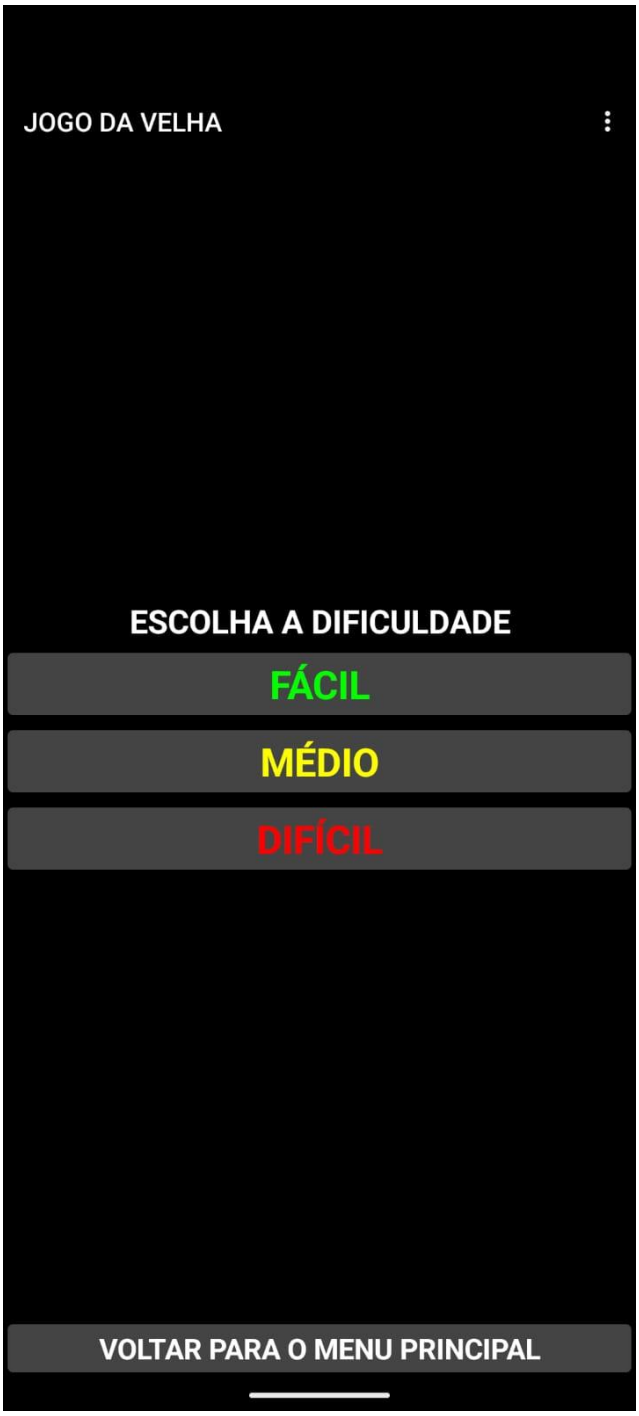
Fonte: (Código criado por Lucca Mendes Alves Pellegrini)

O aplicativo apresenta as opções de: escolher o modo de jogo (jogador vs jogador, jogador vs computador e computador vs computador), a dificuldade (fácil, médio ou difícil) e qual jogador fará o primeiro movimento. Após as escolhas serem feitas, o aplicativo mostrará o placar do jogo na tela e a opção de reiniciar o jogo e, com isso, ter o placar zerado.

Imagem 4 – Primeira tela do aplicativo (escolher o modo de jogo)

Fonte: (Aplicativo criado por Felipe de Faria Rios Coelho)

Imagem 5 – Segunda tela do aplicativo (escolher a dificuldade)

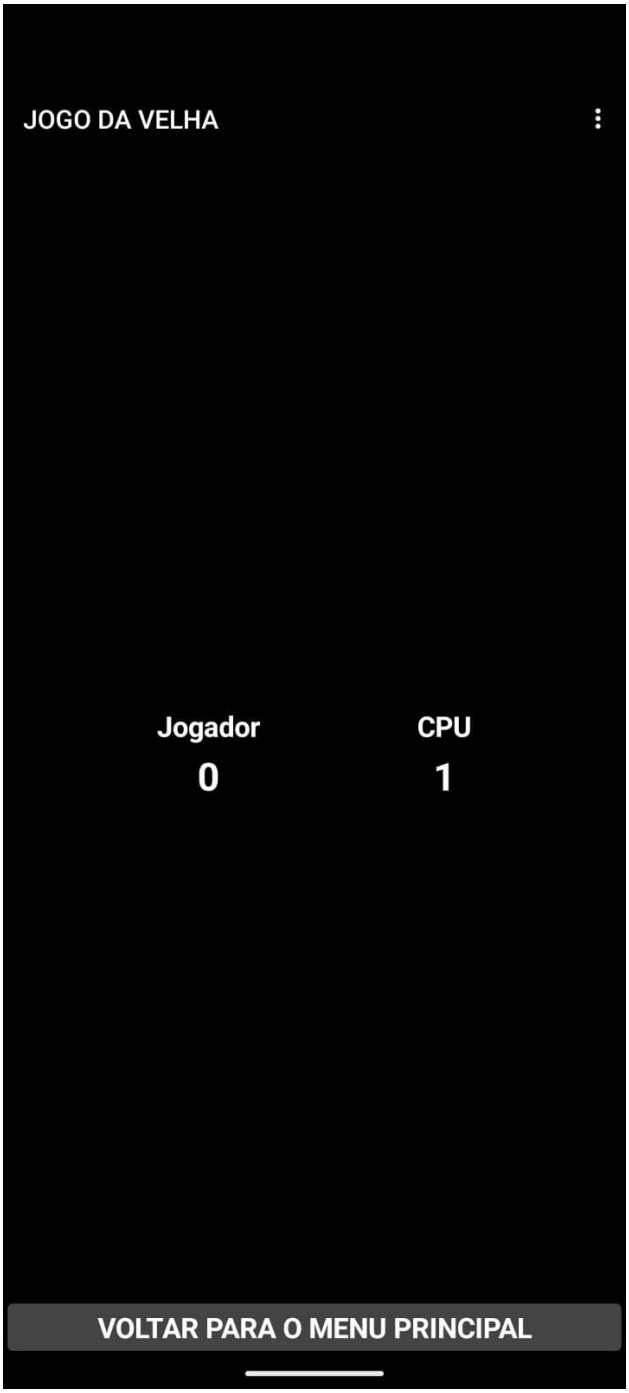


Fonte: (Aplicativo criado por Felipe de Faria Rios Coelho)

Imagem 6 – Terceira tela do aplicativo (escolher o primeiro jogador)

Fonte: (Aplicativo criado por Felipe de Faria Rios Coelho)

Imagem 7 – Quarta tela do aplicativo (placar e opção de reiniciar)

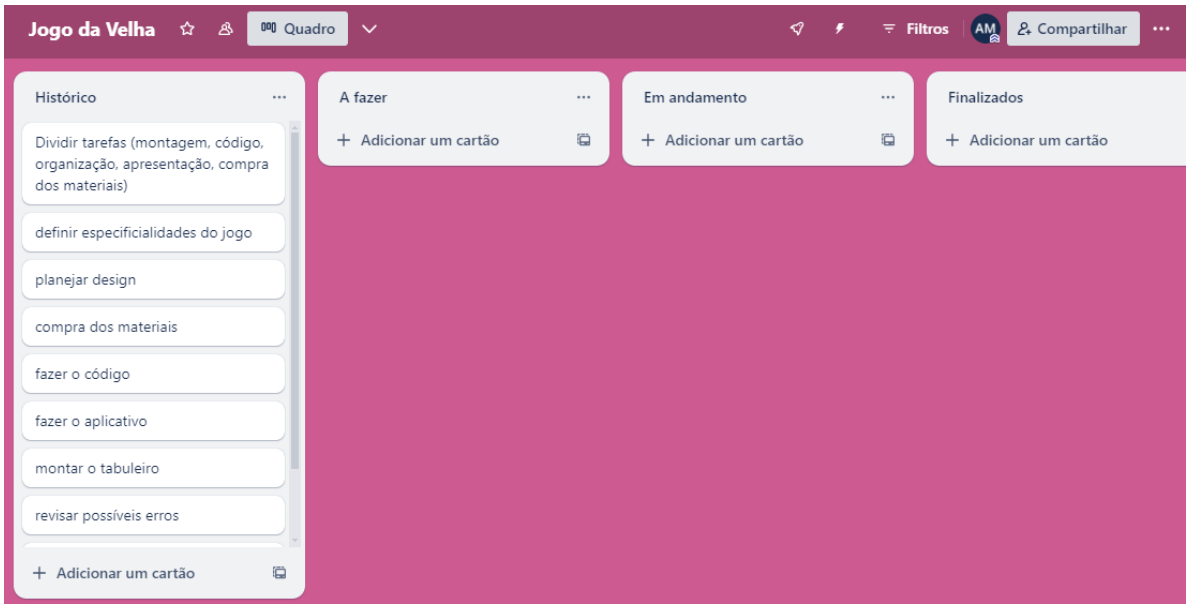


Fonte: (Aplicativo criado por Felipe de Faria Rios Coelho)

3 METODOLOGIAS ÁGEIS

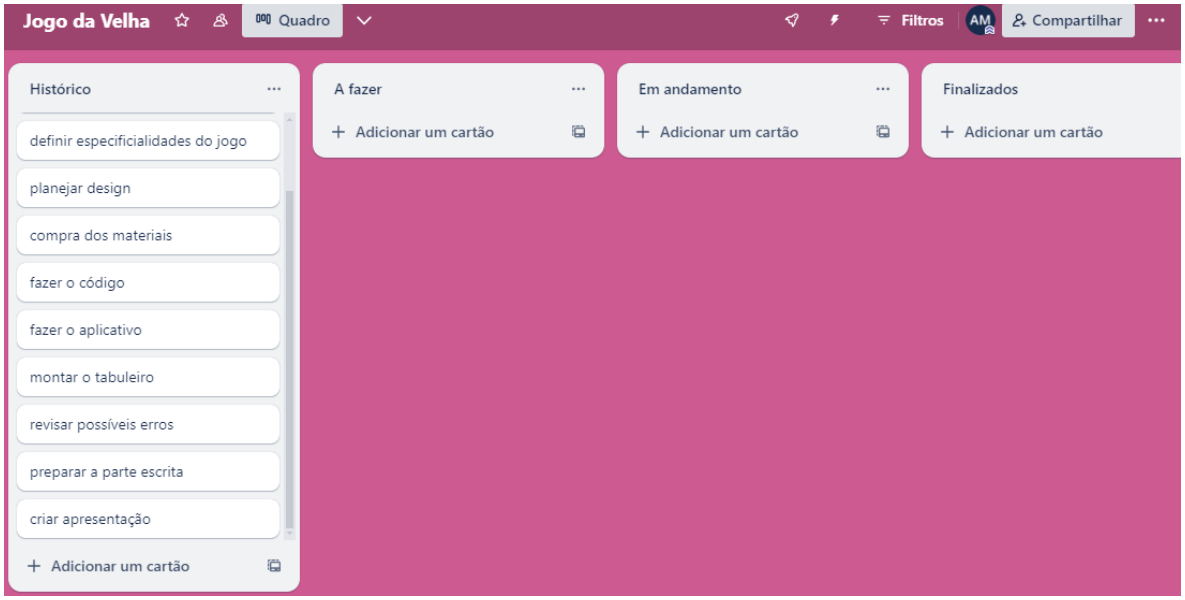
No desenvolvimento do projeto, a metodologia ágil, foi aplicada por meio da definição do Product Owner (Amanda Canizela Guimarães), do Scrum Master (Lucca Mendes Alves Pellegrini) e da equipe (Antonella de Paula Menegaz, Felipe de Faria Rios Coelho, Lucas Alvarenga Fernandes) para que o projeto transcorresse com organização.

Imagem 4 – Organização do histórico das tarefas



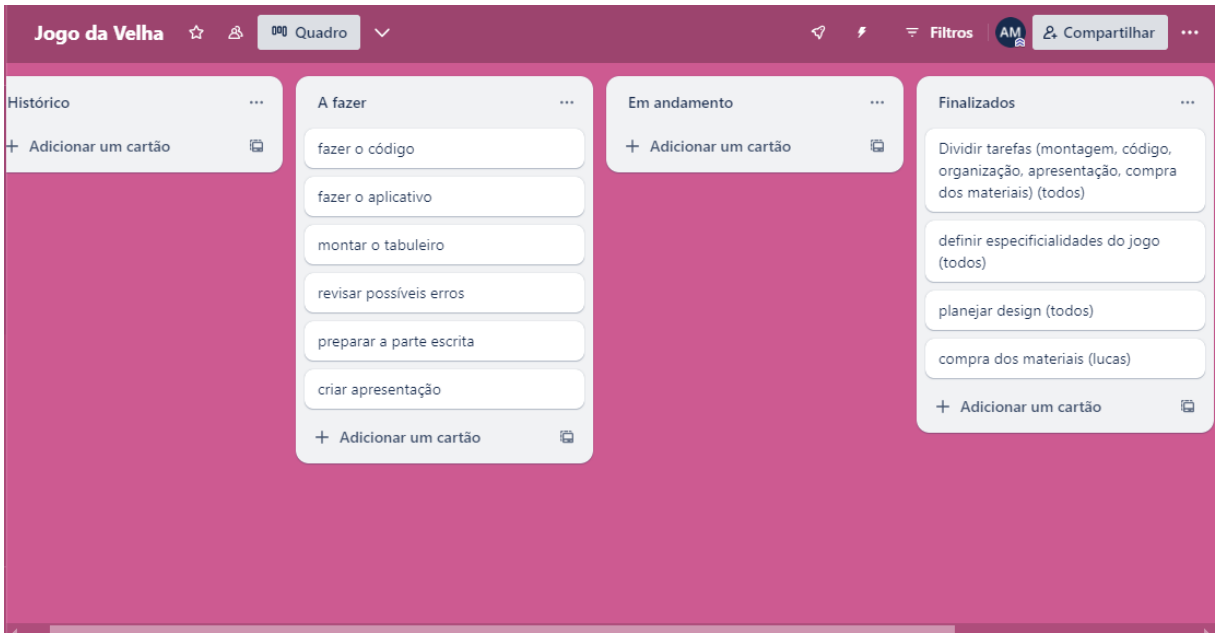
Fonte: (Feito pelo grupo por meio do site Trello)

Imagem 5 – Organização do histórico das tarefas



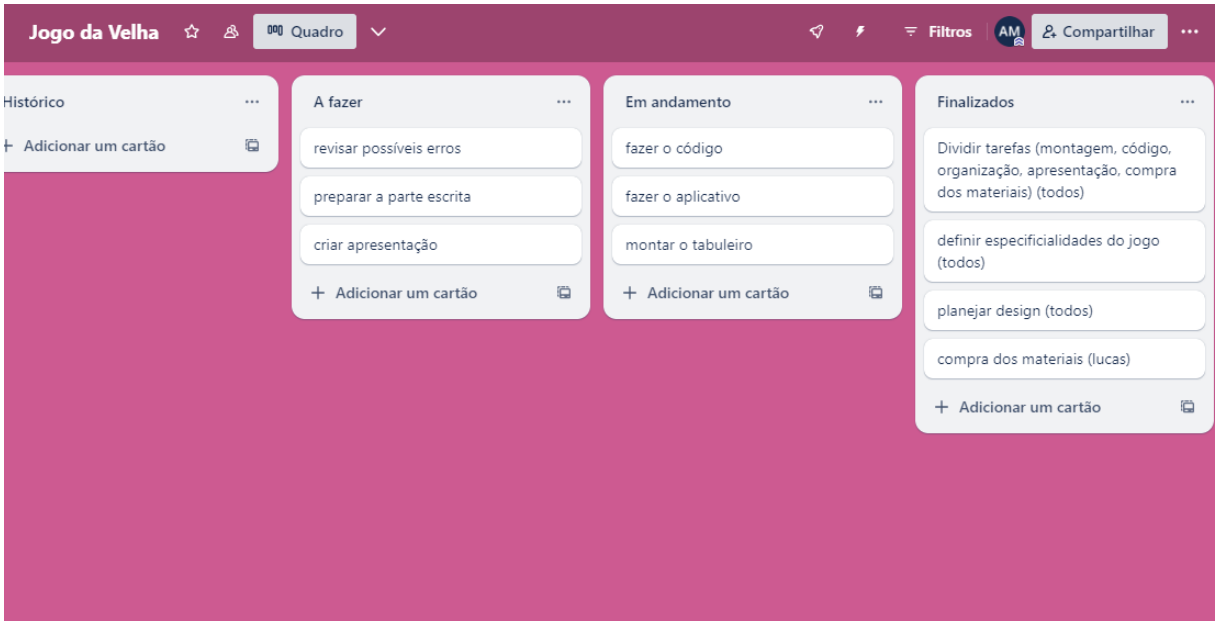
Fonte: (Feito pelo grupo por meio do site Trello)

Imagem 6 – Organização finalizada e a criação do projeto a fazer



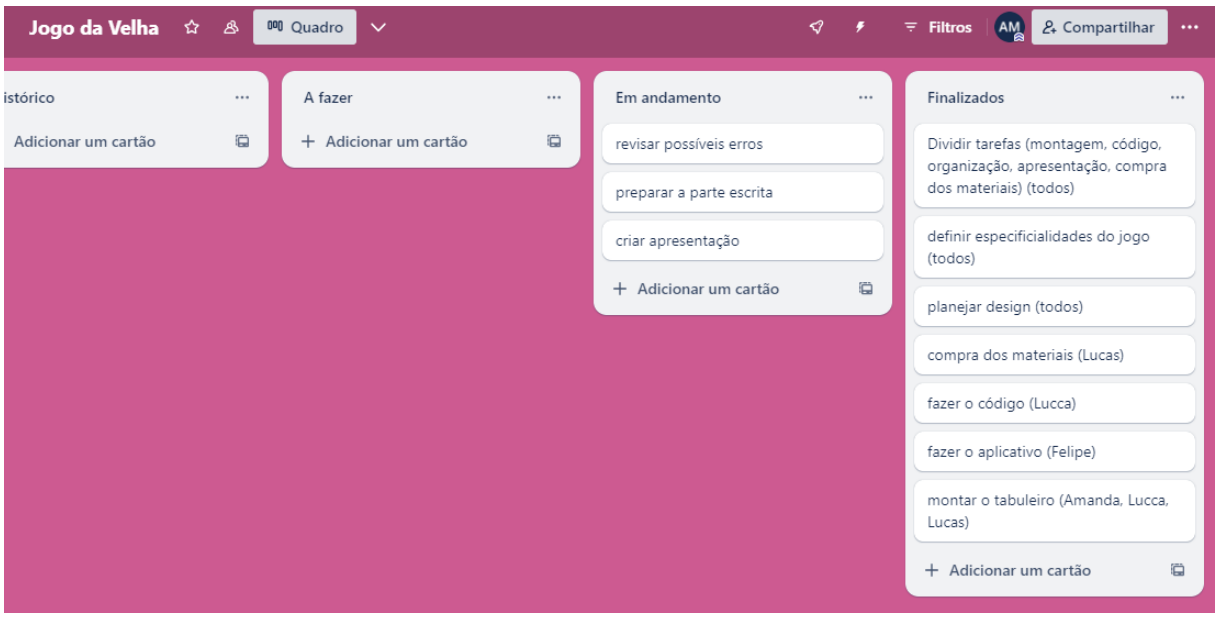
Fonte: (Feito pelo grupo por meio do site Trello)

Imagem 7 – A confecção do código, do aplicativo e do tabuleiro em andamento



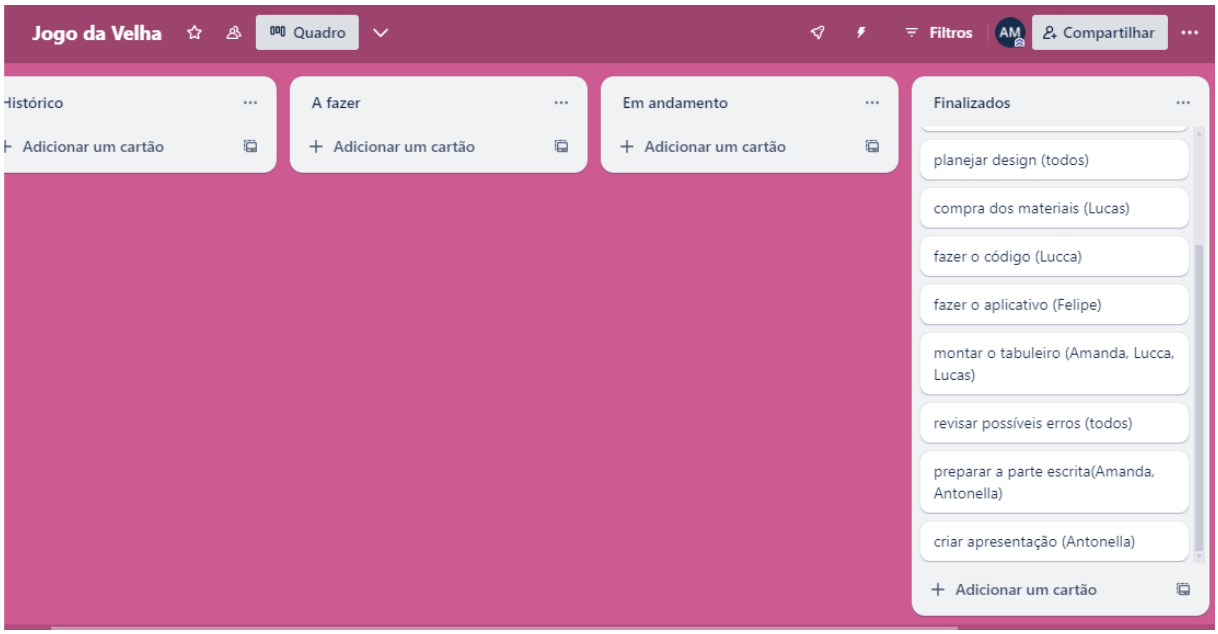
Fonte: (Feito pelo grupo por meio do site Trello)

Imagem 8 – Finalização e reparação de erros em andamento



Fonte: (Feito pelo grupo por meio do site Trello)

Imagem 9 – Tarefas finalizadas



Fonte: (Feito pelo grupo por meio do site Trello)

Durante o trabalho, cada etapa foi designada a um dos integrantes do grupo: o código foi feito e implementado por Lucca Mendes Alves Pellegrini; a montagem do tabuleiro do jogo da velha foi feita por Amanda Canizela Guimarães, Lucas Alvarenga Fernandes e Lucca Mendes Alves Pellegrini; a confecção do aplicativo foi feita por Felipe de Faria Rios Coelho; a organização do Trello, a distribuição das tarefas e a

confecção do documento foram feitas por Amanda Canizela Guimarães e Antonella de Paula Menegaz.

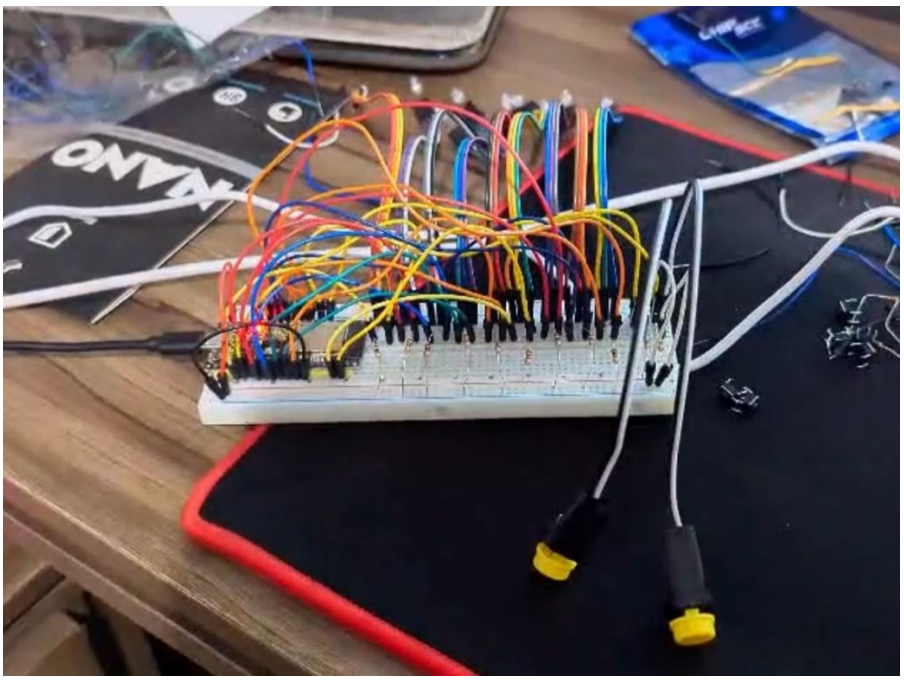
4 TESTES

No primeiro teste, logo no começo da montagem, o grupo já tinha o programa e o aplicativo prontos e rodando, então, para ter certeza de que não haveria maiores complicações, os integrantes decidiram montar apenas um led para testar. Nesse momento, o computador que continha os comandos afirmava não ser compatível com o sistema necessário e atrasou o processo. Com a demora, foi-se decidido montar todos os leds de uma vez para agilizar o processo. No fim, com todos os fios já conectados de acordo com as especificações do esp32 e com outro computador conectado ao programa, o teste foi efetuado. Porém, não estava sendo possível controlar quais luzes seriam acessas e estava muito fraco, sendo o primeiro teste falho do jogo.

No segundo, para tentar obter êxito na montagem dos leds, analisou-se na compra do microcontrolador mais detalhes sobre ele, e foi-se verificado que o esp32 em questão era chinês, então os pinos de positivo e negativo eram os lugares diferentes do original, o que explicava as luzes fracas e a falta de controle. Ao resolver tal problema, o grupo ligou o programa e obteve sucesso, indo para o próximo passo: a montagem dos botões.

Nesse momento, o grupo tinha separado outra protoboard para colocar os botões e eles foram organizados como tal. Porém, quando juntou os fios deles no esp32, notou-se que não havia espaço para os seis botões, havendo apenas quatro pinos. Sendo assim, os integrantes precisaram desmontar a segunda parte do projeto e refazer o programa inteiro, fazendo com que se diminui se seis botões para dois, ajustando para um deles mudar a posição para o jogador escolher onde colocar sua ‘peça’ e o outro para confirmar a posição.

Imagem 10 – Confecção da parte interior do tabuleiro



Fonte: (Foto tirada por Lucca Mendes Alves Pellegrini)

Por fim, ligou-se o jogo para verificar qualquer erro, jogando com todas as possibilidades (jogador versus jogador, jogador versus computador e todos os níveis de dificuldade) para garantir que nenhum problema fosse deixado passar. Notando estar tudo correto, o jogo da velha estava perfeito e completamente funcional.

No geral, as maiores complicações nos testes foram durante a montagem na protoboard, o tipo do microcontrolador, a quantidade de pinos nele e os a potência dos leds.

5 CONCLUSÃO

Em suma, o jogo da velha foi desenvolvido com êxito apesar das complicações listadas ao longo do estudo. O programa e o aplicativo foram as partes mais complexas a serem realizadas, levando mais tempo que o planejado inicialmente. Além disso, a montagem teve alguns imprevistos, estes que foram resolvidos após algumas tentativas.

Sendo assim, o jogo é recomendado para todas as idades, graças ao desenvolvimento de diferentes níveis, como forma de lazer e para a melhoria no pensamento lógico, tornando-se bastante recomendado para aqueles que querem se divertir de maneira simples e fácil.