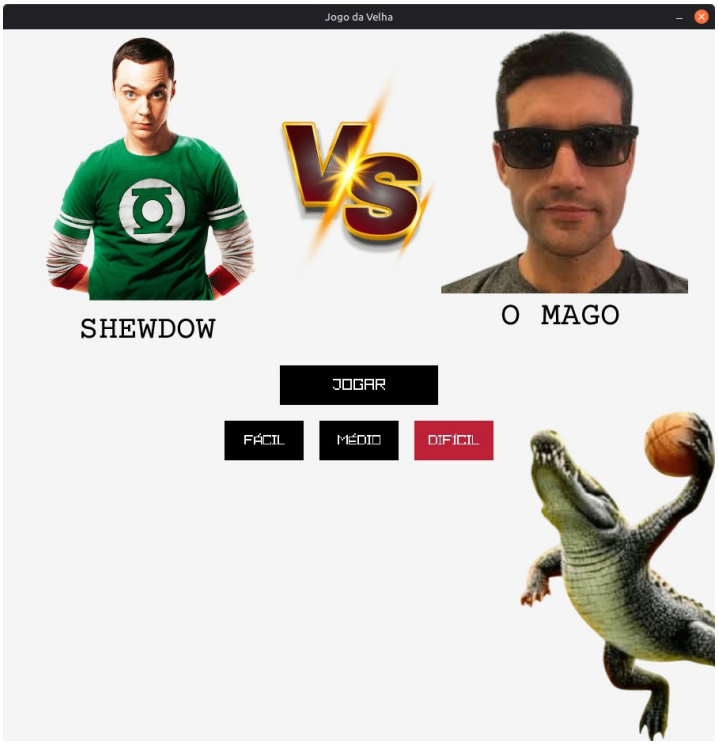
Luiz Frederico Alves Jucá 845659
Iago Moreira Lopes 845012
Lucas de Sousa Albuquerque 831104
Pedro Zanchetta 823107

2)
Nessa fase do trabalho a aplicação realizada foi uma aplicação de uma árvore em um bot do Jogo da Velha. Se trata basicamente de um jogo simples em que você disputa contra um robô em um jogo da velha, possui um menu básico em que você seleciona a dificuldade que quer jogar e inicia o jogo. Essa é apenas a descrição da lógica e do funcionamento do jogo já que ele possui uma "skin" própria que deixa o jogo mais divertido e engraçado.

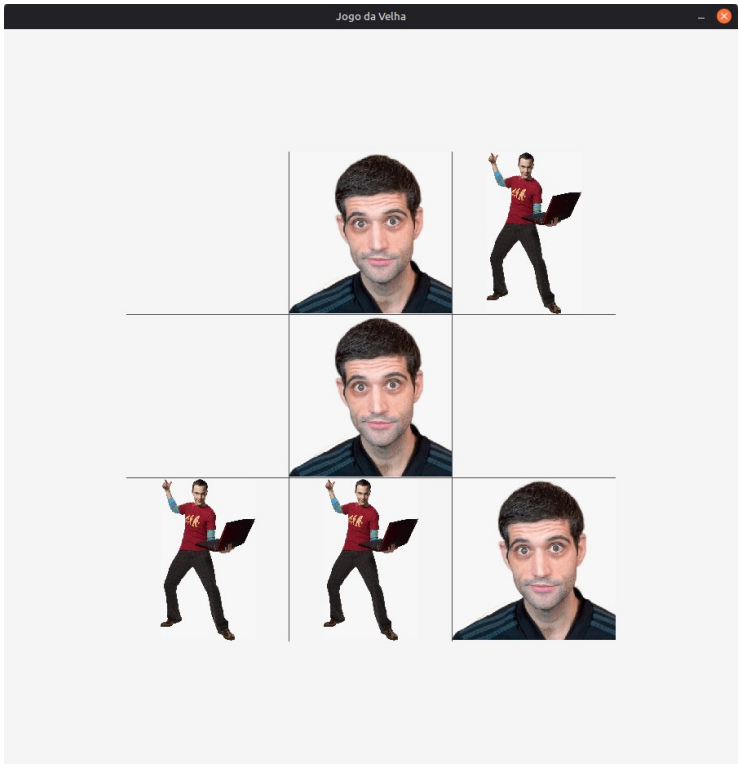
3)
A estrutura sendo utilizada é uma Árvore de Decisão em que nesse contexto é utilizada pelo robô que joga contra o jogador para definir a próxima jogada do robô. Para cada dificuldade selecionada o robô funciona de uma certa forma e possui uma taxa de erro específica na jogada.

4)
Essa aplicação foi toda realizada em linguagem C com suporte da biblioteca gráfica Raylib que permitiu desenhar e projetar tanto o menu quanto o desenho do próprio jogo e a implementação da "skin" do jogo. O jogo esta completo e funcional e é composto por 4 arquivos header sendo eles ai.h, input.h, logic.h e render.h e mais 5 arquivos .c sendo eles ai.c, input.c, logic.c, render.c e main.c além de uma pasta assets que contém todas as imagens e sons utilizados para a "skin" do jogo.

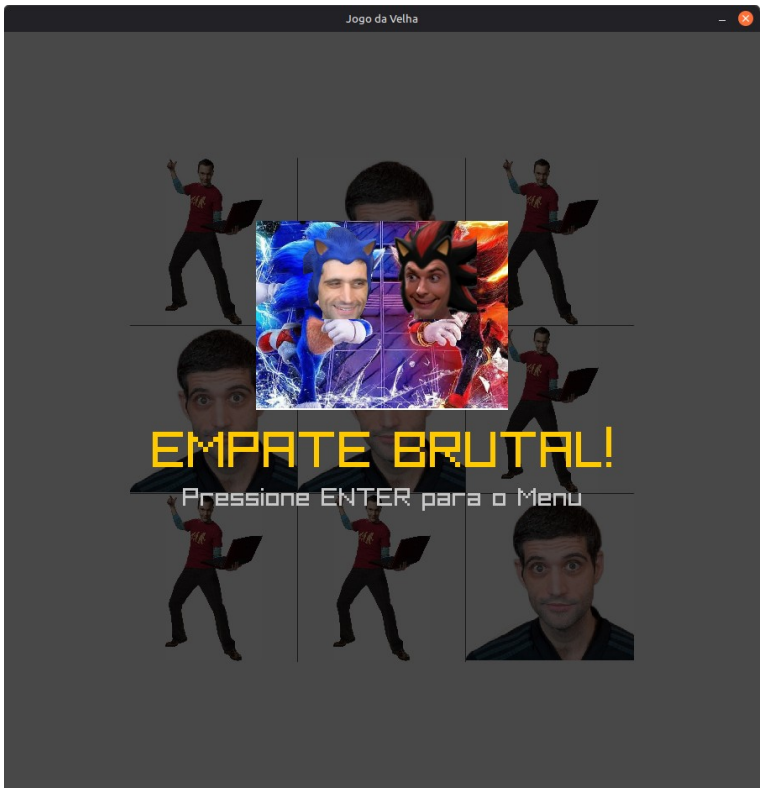
5)



Menu do Jogo



Durante o Jogo



Tela de Empate

6) O jogo está funcional como esperávamos, entretanto, pretendemos melhorar a parte do design: melhorar o menu, as linhas do tabuleiro, adicionar um contador de vitórias/empates/derrotas e talvez mais skins para jogar.

7) Trecho do Código e link do repositório no github

ai.c

```
#include "ai.h"
#include "logic.h"
#include "raylib.h"

#include <stdlib.h>
#include <stddef.h>

struct Node
{
  char BoardState;
  char Player;
  struct Node *Sons[MAX_SONS];
};

GameTree *CreateTree(char BoardState, char Player)
{
  GameTree *NewTree = malloc(sizeof(GameTree));
  NewTree->BoardState = BoardState;
  NewTree->Player = Player;

  for (int i = 0; i < MAX_SONS; i++)
  {
    NewTree->Sons[i] = NULL;
  }

  return NewTree;
}

void FreeTree(GameTree *root)
{
  if (root == NULL)
    return;

  for (int i = 0; i < MAX_SONS; i++)
  {
    FreeTree(root->Sons[i]);
  }

  free(root);
}

#define EASY_ERROR_CHANCE 60  // %
#define MEDIUM_ERROR_CHANCE 25 // %
```

```
#define HARD_ERROR_CHANCE 0    // %

#define MIN(N1, N2) ((N1 < N2) ? N1 : N2)
#define MAX(N1, N2) ((N1 > N2) ? N1 : N2)

GameTree *FindGameTree(Board b, char Player);
int MinMax(GameTree *root, char Player);

int FindRandomMove(Board b, int BestMove)
{
  int valid[9];
  int len = 0;
  for (int i = 0; i < 9; i++)
  {
    if (IsValidMove(b, i % 3, i / 3) && i != BestMove)
    {
      valid[len] = i;
      len++;
    }
  }

  if (len == 0)
    return BestMove;

  return valid[GetRandomValue(0, len-1)];
}

int GetMove(Board b, char CurrPlayer, Difficulty d)
{
  GameTree *root = FindGameTree(b, CurrPlayer);

  if (root->BoardState != 'N')
  {
    return -1;
  }

  int BestScore = -2;
  int BestMove = -1;

  for (int i = 0; i < 9; i++)
  {
    if (root->Sons[i] != NULL)
    {
      int score = MinMax(root->Sons[i], CurrPlayer);
      if (score > BestScore)
      {
        BestScore = score;
        BestMove = i;
```

```
    }
   }
  }

  FreeTree(root);

  int chance = GetRandomValue(0, 100);
  switch (d)
  {
  case Easy:
    if (chance < EASY_ERROR_CHANCE)
      BestMove = FindRandomMove(b, BestMove);
    break;

  case Medium:
    if (chance < MEDIUM_ERROR_CHANCE)
      BestMove = FindRandomMove(b, BestMove);
    break;

  case Hard:
    if (chance < HARD_ERROR_CHANCE)
      BestMove = FindRandomMove(b, BestMove);
    break;
  }
  return BestMove;
}

int MinMax(GameTree *root, char Player)
{
  if (root->BoardState == 'E') // Empate
  {
    return 0;
  }
  else if (root->BoardState == Player) // Vitória do ‘Player’, no caso o bot
  {
    return 1;
  }
  else if (root->BoardState == ((Player == 'X') ? 'O' : 'X')) // Vitória do adversário
  {
    return -1;
  }

  int BestScore;
  if (root->Player == Player) // Agente maximizante
  {
    BestScore = -2;

    for (int i = 0; i < 9; i++)
```

```
    {
      if (root->Sons[i] != NULL)
      {
        int score = MinMax(root->Sons[i], Player);
        BestScore = MAX(score, BestScore);
      }
    }
  }
  else // Agente minimizante
  {
    BestScore = 2;

    for (int i = 0; i < 9; i++)
    {
      if (root->Sons[i] != NULL)
      {
        int score = MinMax(root->Sons[i], Player);
        BestScore = MIN(score, BestScore);
      }
    }
  }

  return BestScore;
}

GameTree *FindGameTree(Board b, char Player)
{
  GameTree *current = CreateTree(BoardState(b), Player);

  if (current->BoardState == 'N')
  {
    for (int i = 0; i < 9; i++)
    {
      if (IsValidMove(b, i % 3, i / 3))
      {
        Board b_copy = b;
        MakeMove(&b_copy, i % 3, i / 3, Player);

        char NextPlayer = (Player == 'O') ? 'X' : 'O';

        current->Sons[i] = FindGameTree(b_copy, NextPlayer);
      }
    }
  }

  return current;
}
```

Link do Projeto (Essa parte está na pasta “JogoDaVelha”)