

**Instituto Superior de Engenharia de Coimbra**

Departamento de Engenharia Informática e de Sistemas

Licenciatura em Engenharia Informática

**Ultimate Tic-Tac-Toe**

A picture containing text

Description automatically generated

**Docente**

Francisco José Baptista Pereira

**Discente**

David Leonel Melo nº 2021129559

**Coimbra, 2022**

# **Introdução**

O projeto Ultimate Tic-Tac-Toe surge no âmbito da unidade curricular de Programação e tem o objetivo da criação do jogo com funcionalidades extra utilizando ferramentas em C que foram previamente instruídas nas aulas.

Para a realização e testagem deste trabalho servi-me do IDE CLion, cuja licença é disponibilizada gratuitamente para os alunos do ISEC em [***https://my.isec.pt***](https://my.isec.pt)

A minha abordagem ao projeto tem como objetivo a clareza e simplicidade de cada função e ficheiro, estando o trabalho dividido por vários ficheiros, cada um com uma designação indicativa do seu conteúdo, de modo a ajudar o programador na modificação de algum aspeto do jogo.

Para tornar a interface mais agradável e o jogo mais carismático, decidi criar um assistente pessoal, ***Levi***, que auxilia o utilizador ao longo da sua experiência.

Esta implementação, apesar de relativamente simples, permite distinguir o meu projeto dos demais.

Por fim, de realçar que usufruí de plataformas como o *YouTube* e *Stack Overflow* de modo a aperfeiçoar o meu conhecimento da matéria lecionada nas aulas e melhorar o planeamento do meu projeto.

1. **Ultimate Tic-Tac-Toe**

Existe uma área de jogo no qual estão organizados 9 mini-tabuleiros do jogo do galo, dispostos numa grelha 3´3.

O jogo desenrola-se entre 2 jogadores, de acordo com as seguintes regras:

1. Na sua vez, cada jogador coloca a sua peça num dos 9 mini-tabuleiros
2. Um jogador que faça 3 em linha num dos mini-tabuleiros ganha essa secção
3. Um jogador que ganhe 3 mini-tabuleiros em linha (no tabuleiro grande) ganha o jogo
4. A escolha do mini-tabuleiro em que se joga não é livre, sendo determinada pela jogada anterior do adversário. A posição escolhida pelo adversário identifica o minitabuleiro onde deve ser continuado o jogo.
5. Verifica-se um empate se o jogo chegar ao final sem que nenhum dos jogadores consiga fazer 3 mini-tabuleiros em linha.

A picture containing application

Description automatically generated

1. **Ficheiros**

***main.c***

Ficheiro principal do programa que contém as condições relativas aos vários modos de jogo, servindo como base de controlo para a inicialização do mesmo.

***vshuman.c***

Ficheiro que contém todas as instruções para a realização do jogo caso seja entre dois jogadores humanos.

***vsAI.c***

Ficheiro que contém todas as instruções para a realização do jogo caso seja entre um jogador humano e o computador (AI).

***functions.c***

Ficheiro que contém funções miscellaneous essenciais ao funcionamento do programa.

***levi.c***

Ficheiro que contém as funções do assistente pessoal Levi.

***boardFunctions.c***

Ficheiro que contém as funções relativas à dinâmica do tabuleiro.

***winConditionFunctions.c***

Ficheiro que contém as funções de verificação de vitória, tanto nos tabuleiros inferiores como no principal.

***linkedlist.c***

Ficheiro que contém as funções relativas à dinâmica da lista ligada do jogo.

***functions.h***

Ficheiro *header* que contém todas as funções e estruturas do programa.

1. **Funcionamento geral do programa**

Ao iniciar o jogo deparamo-nos com uma interface simples e agradável na qual temos a possibilidade de escolher o modo de jogo (1 vs 1 / 1 vs AI) ou de concluir um jogo anterior. Inicia-se um ciclo de modo a garantir que o utilizador seleciona uma opção exequível para prosseguir no modo desejado.

4.1. ***Opção ‘1’ – vshuman()***

A variável ***load*** é colocada a ‘0’, indicando que não é para continuar um jogo previamente interrompido.

É inicializado um novo jogo entre dois humanos, será neste momento que o nosso assistente pessoal ***Levi*** faz a sua primeira aparição com a função de decidir quem fará a primeira jogada. É escolhido um jogador aleatório para adivinhar o número (1 ou 2) que ***Levi*** estará a pensar, sendo o número também escolhido aleatoriamente em cada jogo. Caso o jogador escolhido acerte será ele o primeiro a jogar, se errar será o outro.

É então criada uma matriz bidimensional dinâmica recorrendo a ***malloc***, onde, inicialmente, todas as indexações estão preenchidas por um espaço em branco ‘ ‘.

O tabuleiro onde será feita a primeira jogada é também escolhido aleatoriamente com a função ***RandomNumber()***. É impressa uma representação atual da matriz e a indicação do tabuleiro onde deverá jogar. Inicia-se outro ciclo até que seja escolhida uma jogada válida, prosseguindo depois desta condição se verificar.

A jogada, após a sua validação, é adicionada no fim da lista ligada e o jogador atual é trocado com a função ***SwitchPlayer()***.

Na jogada seguinte ***Levi*** questiona o jogador sobre a sua vontade de continuar o jogo. Caso queira interromper, todas as jogadas até então registadas são exportadas para o ficheiro binário ‘***jogo.bin***’.

Se pretender continuar, o jogador é novamente questionado por ***Levi***, desta vez com o objetivo de saber se o jogador pretende visualizar alguma jogada anterior (*máx*: 10).

Se a resposta for ‘Y’ ou ‘y’ e o número das ‘k’ jogadas for válido estas são impressas, da mais antiga para a mais recente, contendo as informações do jogador, tabuleiro e posição, prosseguindo normalmente o jogo após a impressão.

O ciclo continua até que seja atingida a vitória ‘***Ultimate***’ ou alcançadas 81 jogadas, sendo que, após uma destas condições se verificar é exibida uma mensagem que contém o resultado final do jogo e um novo inquérito de ***Levi***, necessário para a escolha do nome do ficheiro de texto onde serão guardadas todas as jogadas, da mais antiga para a mais recente, referentes ao jogo finalizado. É verificada a existência de caracteres inválidos na *string*, de modo a assegurar que os dados não sejam perdidos.

4.2. ***Opção ‘2’ – vsAI()***

A variável ***load*** é colocada a ‘0’, indicando que não é para continuar um jogo previamente interrompido.

É inicializado um novo jogo entre um humano e o computador, será neste momento que o nosso assistente pessoal ***Levi*** faz a sua primeira aparição com a função de decidir quem fará a primeira jogada. O jogador humano tentará adivinhar o número (1 ou 2) que ***Levi*** está a pensar, sendo o número também escolhido aleatoriamente em cada jogo. Caso acerte será o primeiro a jogar, se errar será o computador.

É então criada uma matriz bidimensional dinâmica recorrendo a ***malloc***, onde, inicialmente, todas as indexações estão preenchidas por um espaço em branco ‘ ‘.

O tabuleiro onde será feita a primeira jogada é também escolhido aleatoriamente com a função ***RandomNumber()***. É impressa uma representação atual do tabuleiro ao jogador e a indicação do tabuleiro onde deverá jogar.

Caso seja o turno do computador, este escolherá uma posição aleatória válida no tabuleiro correto, podendo esta não constituir uma opção lógica, apenas válida.

Se o turno pertencer ao jogador humano, inicia-se um novo ciclo até que seja escolhida uma jogada válida, prosseguindo depois desta condição se verificar.

A jogada, após a sua validação, é adicionada no fim da lista ligada e o jogador atual é trocado com a função ***SwitchPlayer()***.

Na jogada seguinte ***Levi*** questiona o jogador sobre a sua vontade de continuar ou interromper o jogo. Caso queira interromper todas as jogadas até então registadas são exportadas para o ficheiro binário ‘***jogo.bin***’.

Se pretender continuar, o jogador é novamente questionado por ***Levi***, desta vez com o objetivo de saber se o jogador pretende visualizar alguma jogada anterior (*máx*: 10).

Se a resposta for ‘Y’ ou ‘y’ e o número das ‘k’ jogadas for válido estas são impressas, da mais antiga para a mais recente, contendo as informações do jogador, tabuleiro e posição, prosseguindo normalmente o jogo após a impressão.

O ciclo continua até que seja atingida a vitória ‘***Ultimate***’ ou alcançadas 81 jogadas, sendo que, após uma destas condições se verificar é exibida uma mensagem que contém o resultado final do jogo e um novo inquérito de ***Levi***, necessário para a escolha do nome do ficheiro de texto onde serão guardadas todas as jogadas, da mais antiga para a mais recente, referentes ao jogo finalizado. É verificada a existência de caracteres inválidos na *string*, de modo a assegurar que os dados não são perdidos.

4.3. ***Opção ‘3’ – Load Game***

A variável ***load*** é colocada a ‘1’, indicando às funções ***vshuman()*** e ***vsAI()*** que é para continuar um jogo previamente interrompido. Desta forma e recorrendo à função ***readList()***, as jogadas da partida anterior registadas no ficheiro binário são recuperadas e adicionadas a uma lista ligada.

Dependendo do modo de jogo guardado no ficheiro binário, é chamada a função ***vshuman()*** ou ***vsAI()***, inicia-se um ciclo de preenchimento da matriz com base nas jogadas presentes na lista ligada e, no final, é impresso o tabuleiro completo.

A partir deste momento a partida seguirá normalmente, de acordo com as funções esclarecidas nas duas opções anteriores.

1. **Conclusão**

A realização deste projeto contribuiu imenso para a consolidação dos conhecimentos adquiridos nas aulas, sendo que conceitos mais abstratos como as listas ligadas e os ponteiros foram tornados mais “palpáveis” quando aplicados na criação das várias funcionalidades do jogo.

Em termos de dificuldade global do trabalho, apesar de não ter sido fácil, teve um grau adequado para um aluno de engenharia informática.

A minha autoavaliação é de 7 valores dado que, além de ter o jogo completamente funcional com todos os requisitos do enunciado, adicionei alguns extras criativos e garanti a segurança do jogo com vários ciclos de verificação de *inputs*, uns mais simples e outros mais complexos, como a verificação do nome do ficheiro de texto e ainda a prevenção de *memory leaks* com as funções que recorrem de *malloc*.