



UNIVERSIDADE DE SÃO PAULO

ENGENHARIA DE COMPUTAÇÃO

SSC0600 - INTRODUÇÃO À CIÊNCIA DA COMPUTAÇÃO I

QUARTO

Autor:

Ivan Mateus DE LIMA
AZEVEDO

Professor:

Dr. Adenilso DA SILVA
SIMÃO

Nº USP:

10525602

28 de abril de 2018

1 Introdução

Quarto é um jogo de tabuleiro para dois jogadores inventado pelo *board game designer* Blaise Müller. É jogado em um tabuleiro 4x4 e possui 16 peças diferentes, cada uma com 4 qualidades que as distinguem entre si. São elas:

1. quadrada ou redonda;
2. preta ou branca (ou quaisquer outras duas cores distintas);
3. pequena ou grande;
4. furada ou maciça.

O jogo consiste em escolher as peças que o adversário jogará e deixar que ele as posicione de maneira a completar uma linha, coluna ou diagonal de peças com pelo menos uma qualidade igual (todas pretas, todas furadas, etc). Assim, ganha quem conseguir fazer isto. O jogo empata se todas as peças tiverem sido jogadas e não houver este cenário de vitória.

2 Descrição do projeto

2.1 Ambiente de desenvolvimento

O código foi escrito na plataforma Linux através do editor de texto Atom e o projeto foi disponibilizado na plataforma de hospedagem de código-fonte GitHub.

2.2 Compilador utilizado

Foi utilizada a versão 5.4.0 do compilador GCC. E a plataforma de compilação da versão binária, i686.

2.3 Códigos-fonte

Para executar o programa, os seguintes arquivos são necessários:

1. main.c

2. board.c
3. board.h
4. player.c
5. player.h
6. ai.c
7. ai.h
8. game.c
9. game.h

3 Tutorial

3.1 Como compilar

Para compilar o programa é necessário ter o GCC instalado no micro utilizado. Após isso, abra o terminal, vá para o diretório em que os arquivos acima estão salvos e digite o seguinte comando:

```
$ gcc main.c board.c game.c player.c ai.c -o main
```

Para compilar usando o Code::Blocks, abra um projeto vazio (*empty project*), então vá em *Project* → *Add files...*, selecione todos os arquivos da seção 2.3, clique em *Open* e em *Ok* na janela seguinte. Por fim, tecle Ctrl+F9 para compilar o programa.

3.2 Como executar

Caso queira executar o programa normalmente, basta digitar no terminal o seguinte comando:

```
$ ./main
```

Esse comando abre um menu que exhibe duas opções ao usuário (jogar no modo multiplayer ou contra a IA). Entretanto, caso queira executar o programa teste com valores previamente estabelecidos, basta digitar:

```
$ ./main -test
```

Caso esteja utilizando o compilador Code::Blocks, basta clicar no botão *Run*, na barra de tarefas, ou teclar Ctrl+F10. Caso queira executar o programa-teste, vá em *Project* → *Set programs' arguments....* No campo *Program arguments*, digite `-test` em seguida, compile e execute o programa como nas instruções dadas acima.

3.3 Entradas-exemplo

Ao executar o programa normalmente, o menu abaixo será exibido:

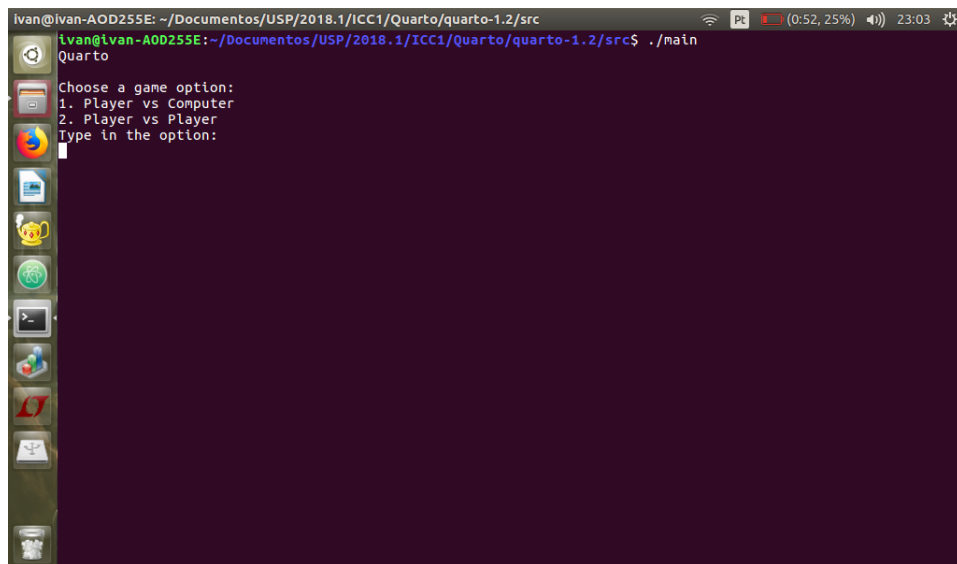


Figura 1: Figura que mostra o primeiro menu de opções.

Se o usuário escolher a opção 1, ele irá ser redirecionado a um segundo menu para que decida se quer começar primeiro ou se quer que a IA comece:

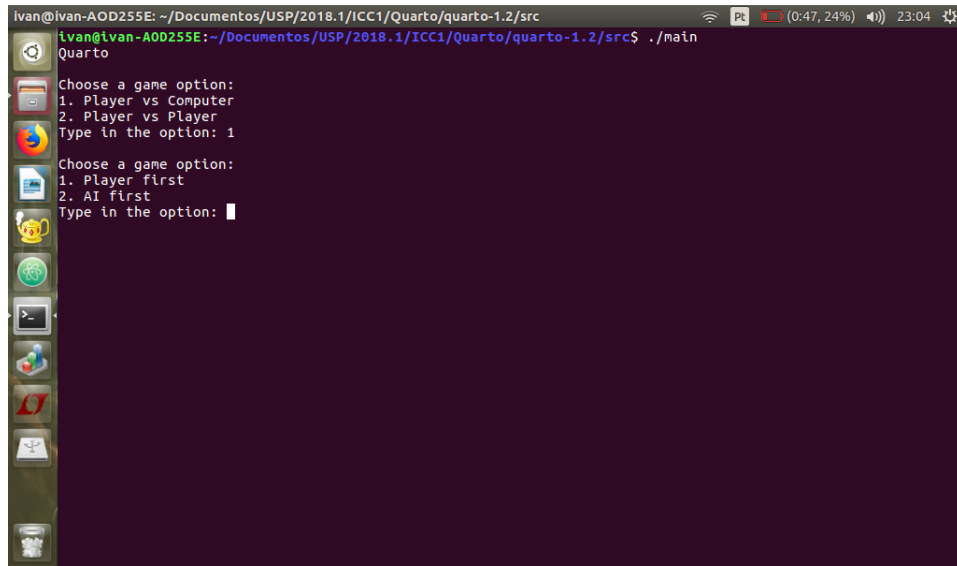


Figura 2: Figura que mostra o menu da opção de jogo contra a IA.

Se escolher a opção 2 do primeiro menu, o usuário irá automaticamente para o início do jogo e, obviamente, o *player 1* será, por *default*, o primeiro a jogar:

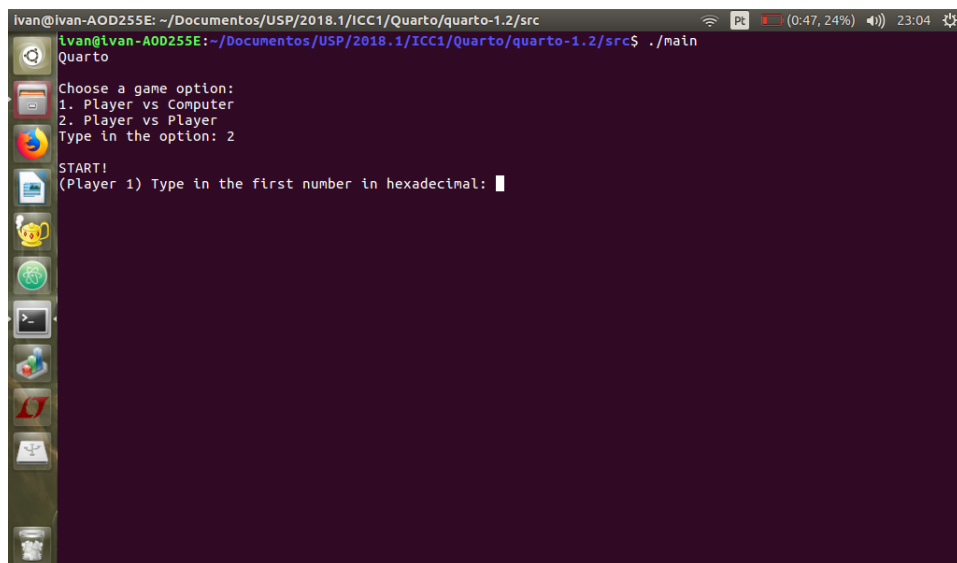
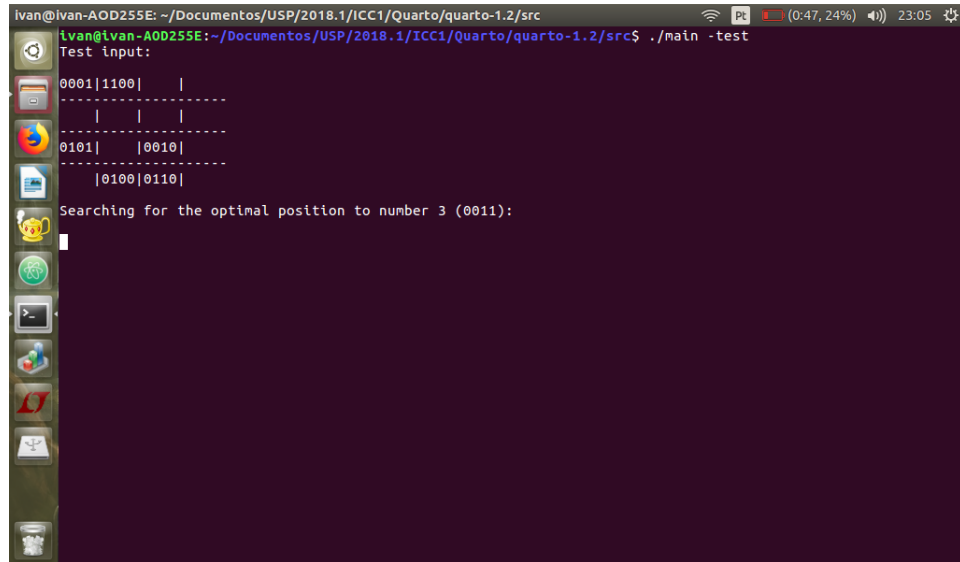


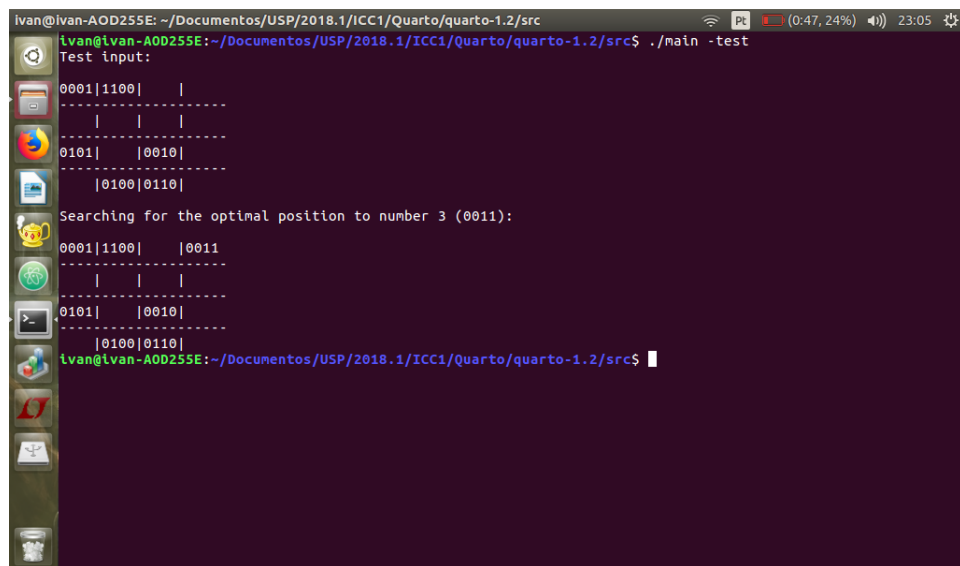
Figura 3: Figura que mostra o início do jogo no modo *multiplayer*.

Porém, no caso em que o usuário escolher executar a entrada-teste, será exibido um tabuleiro com valores previamente definidos e o programa irá procurar a posição ótima para o número 3 através do algoritmo *minimax* (que será explicado na seção 4).



```
ivan@ivan-AOD255E: ~/Documentos/USP/2018.1/ICC1/Quarto/quarto-1.2/src
ivan@ivan-AOD255E:~/Documentos/USP/2018.1/ICC1/Quarto/quarto-1.2/src$ ./main -test
Test input:
0001|1100|  |
-----
|  |  |
-----
0101|  |0010|
-----
|0100|0110|
-----
Searching for the optimal position to number 3 (0011):
█
```

Figura 4: Figura que mostra um tabuleiro com algumas peças já posicionadas.



```
ivan@ivan-AOD255E: ~/Documentos/USP/2018.1/ICC1/Quarto/quarto-1.2/src
ivan@ivan-AOD255E:~/Documentos/USP/2018.1/ICC1/Quarto/quarto-1.2/src$ ./main -test
Test input:
0001|1100|  |
-----
|  |  |
-----
0101|  |0010|
-----
|0100|0110|
-----
Searching for the optimal position to number 3 (0011):
0001|1100|  |0011
-----
|  |  |
-----
0101|  |0010|
-----
|0100|0110|
-----
ivan@ivan-AOD255E:~/Documentos/USP/2018.1/ICC1/Quarto/quarto-1.2/src$ █
```

Figura 5: Figura que mostra o número 3 na melhor posição dada pelo algoritmo *minimax*.

4 Sobre a IA

O posicionamento de valores da IA funciona basicamente através de dois algoritmos. O mais sofisticado deles, o *minimax*, funciona da seguinte forma: ele representa todas as possibilidades de jogada em uma árvore (chamada *game tree*), onde cada nó é uma jogada em uma posição diferente e cada nível da árvore é a vez de um jogador. O algoritmo atribui a cada nó uma pontuação (*score*; a pontuação máxima (situação em que o jogador vence a partida) é 1 e a pontuação mínima (quando o oponente vence) é -1. Quando há empate, a pontuação é zero. Assim, ao atribuir essas pontuações, o algoritmo escolhe o caminho mais provável para que aconteça a vitória do jogador e, quando esse cenário não é possível de ser atingido, ele busca o empate.

Por ser um algoritmo que cria todas as possibilidades de jogada, ele leva muito tempo para dar um retorno nas primeiras jogadas. Assim, ele só é usado no programa quando já houver 6 posições ocupadas, pois a partir deste número, o programa já retorna valores mais rapidamente. Antes disso, a IA funciona basicamente se defendendo. Para escolher a posição na qual vai jogar, o programa verifica qual dos números que ainda não foram jogados tem a maior pontuação (maior número de sequências de 0 ou de 1). E para dar um valor para o jogador jogar, a IA verifica, utilizando este mesmo algoritmo, qual dos números não jogados possui a menor pontuação e entrega esse número.