

# **Estruturas de Dados e Algoritmos**

## **Projeto - Jogo de Palavras “Boggle”**

Prof. Jorge Alves da Silva

Mestrado Integrado em Bioengenharia  
Engenharia Biomédica

### **Grupo 02**

João Carlos Ramos Gonçalves de Matos

201704111

Maria Jorge Miranda Loureiro

201704188

Maria Manuel Domingos Carvalho

201706990

Data de entrega do projeto: 3 de Janeiro de 2020

## Introdução

No âmbito da unidade curricular de Estruturas de Dados e Algoritmos realizou-se este projeto, que teve como objetivo implementar um programa em linguagem C++ que permita jogar o jogo de palavras *Boggle*. Neste jogo, um ou mais jogadores escrevem palavras com as letras mostradas num tabuleiro e ganham pontos se esta for válida, mediante as regras do jogo, e conforme o seu tamanho.

## Classes implementadas

A elaboração deste programa foi feita segundo o paradigma “orientado a objetos”, isto porque se implementaram várias classes de modo a executar os vários passos deste jogo. Assim, as classes desenvolvidas foram as seguintes:

### Classe *Cube*

A classe *Cube* tem um construtor que gera um dado com seis faces, cada uma delas com uma letra; para isso é necessário fornecer vetor de *char* como argumento de entrada. Além da criação do cubo, esta classe contém ainda vários métodos que permitem rodar o dado (método *roll*) e obter a letra que está na face virada superiormente (*getTopLetter*); esta letra, pode ser observada através de um método de *display* (método *displayTop*). A declaração dos métodos e atributos da classe pode ser vista na figura 1.

```
class Cube
{
public:
    Cube(const std::vector<char>& letters); // build a cube with 'letters' on faces
    void roll(); // roll cube and change _topSide
    char getTopLetter() const; // get top letter
    void displayTop(std::ostream& os) const; // display top letter

private:
    std::vector<char> _letters; // the letters of the cube
    std::size_t _topSide; // the side up: from 0 to 5
};
```

Figura 1 - Declaração dos métodos e atributos da classe *Cube*

### Classe *Board*

A classe *Board* gera, através da classe *Cube*, um tabuleiro com um tamanho variável de cubos. Tal como na classe *Cube*, existe um método construtor que irá desenvolver um tabuleiro com um tamanho dependente das configurações do jogo, obtendo-se o número de linhas e colunas desejadas através do método privado *readSize* e a letras para construir cada cubo com o método privado *readChars*. Obtendo o tamanho, é possível construir um tabuleiro com o número de cubos desejados.

Nesta classe existem ainda métodos que possibilitam baralhar o tabuleiro (método *shuffle*), ver qual é a letra que está virada para cima numa determinada posição do tabuleiro (método *getTopLetter*) e a visualização do tabuleiro (método *display*). Além disso, os métodos *findWord* e *findWordAux* inferem se a palavra jogada pelo utilizador existe no tabuleiro.

Com o método *newRound* é possível perguntar aos jogadores se permitem continuar o jogo ou terminá-lo. A declaração dos métodos e atributos da classe pode ser vista na figura 2.

```

class Board
{
public:
    Board(const string& filename); // load and construct board from 'filename'
    char getTopLetter(const Position& pos) const; // get top letter at position 'pos'
    void shuffle(); // roll all cubes and shuffle them on the board
    void display(ostream& os) const; // display board
    bool findWord(string word, vector<Position>& path); // find 'word' on board
    // if 'word' is found, 'path' will contain the coordinates (lin,col) of the letters of the word
    bool newRound(ostream& os, ostream& osRep, unsigned int& round); //ask the players if they want to continue or quit the game

private:
    int _numRows, _numCols; // number of columns and number of rows of the board
    vector<vector<Cube>> _board; // board representation
    bool findWordAux(vector<vector<bool>>& visited, int i, int j, vector<Position>& path, string& str, const string& word, bool& found);
    //return true if path of word is found
    void readSize(const string& filename, int& _numRows, int& _numCols); //read size of board from file
    vector<string> readChars(const string& filename, int& _numRows, int& _numCols); //read letters from board
};

```

Figura 2 - Declaração dos métodos e atributos da classe *Board*

## Classe *Player*

A classe *Player* tem como função criar os diferentes jogadores e manusear a sua informação. O construtor não tem nenhum parâmetro de entrada, criando um jogador sem nome, apostas ou pontuação. O método *readInfo* é responsável pela recolha da informação do jogador, isto é, o seu nome. O método estático *collectPlayers* tem o intuito de criar e ler a informação de diferentes jogadores, utilizando o método *readInfo* e o construtor da classe para o fazer. O jogador pode começar a jogar pressionando ENTER, assim que queria e o tabuleira esteja a ser mostrado, com o método *startPlay*. Além disso, o método *readWordsTimed* permite ler as apostas dos jogadores e guardá-las na variável privada *bets*, durante o tempo determinado pelas configurações do jogo. Para isto, este recorre a dois métodos privados: *readStrTimed* (lê uma palavra antes do tempo terminar) e *findRepetition* (verifica que a palavra não está já repetida nas apostas feitas anteriormente). Após a recolha das apostas de cada jogador é necessário analisar cada aposta e pontuá-la de acordo com as regras do jogo, para isso é usado o método *verifyBets*, que recorre a dois métodos privados: *findRepetition* (verifica se a palavra foi repetida por algum jogador) e *displayWordsTimed* (mostra na consola a pontuação de cada palavra e a razão pela qual a obteve). Quando já temos todas as pontuações calculadas, é necessário verificar se já existe um vencedor do jogo, verificação feita pelo método estático *findWinner*. Para mostrar as pontuações de cada jogador utilizamos o método *displayPlayersScores* e, para mostrar o vencedor, o método *displayWinner*. A declaração dos métodos e atributos da classe pode ser vista na figura 3.

```

class Player
{
public:
    Player(); // build a player with name = "NO NAME"

    void readInfo(); // read player's name
    void readWordsTimed(const unsigned int& duration, ostream& os); // read words of the player's bet, in one round
    void displayPlayersScores(ostream& os) const; // show table of Scores
    void verifyBets(Dictionary dic, Board bo, ostream& os_console, ostream& os_report, Configurations config); //verify and display the score of each bet
    void displayWinner(ostream& os) const; //display winners info
    void startPlay(); //start to play the game
    int playerNumber;

    static vector<Player> collectPlayers(); // loop to read several players' info
    static bool findWinner(const int& minScore, ostream& os, int& winnerIndex, vector<Player>& players);
    //return true if a player already reached the minimum score to win

private:
    string name;
    vector<string> bets;
    int score;

    static unsigned int numPlayers;
    static vector<string> roundBets;
    static vector<int> allScores;

    void readStrTimed(string& str, time_t duration); //Read a word to string 'str', before 'duration' time is elapsed, otherwise 'str' will be an empty string

    bool findRepetition(const string& str, vector<string> _bets); // check if word 'str' exists in vector _bets

    void displayWordScore(int& word_score, ostream& os, const string& cause); //display score and cause for a word
};

```

Figura 3 - Declaração dos métodos e atributos da classe *Player*

## Classe *Dictionary*

A classe dicionário tem como objetivo criar um dicionário, lendo um ficheiro com todas as palavras que deverão ser adicionadas a este. Assim, o construtor lê o ficheiro cujo nome é introduzido como parâmetro de entrada e guarda cada palavra no dicionário, utilizando o método privado *wordToUpper* para garantir que todas as palavras são guardadas com as letras em maiúscula. Além disso, esta classe possui ainda o método *find*, que permite encontrar uma determinada palavra do dicionário. A declaração dos métodos e atributos da classe pode ser vista na figura 4.

```

class Dictionary
{
public:
    Dictionary(const string& filename); // load "dictionary" from 'filename'
    bool find(const string& word); // find 'word' in "dictionary"

private:
    set<string> dictionary; // set with the dictionary
    void wordToUpper(string& word); // transform word in WORD
};

```

Figura 4 - Declaração dos métodos e atributos da classe *Dictionary*

## Ficheiros extra necessários ao jogo

Para o bom funcionamento do jogo, além das classes, surgiu a necessidade de criar determinados ficheiros para implementar diferentes funções. Assim, construíram-se os ficheiros *GameReport*, *Configurations* e *Position*. Além disso, utilizou-se o ficheiro *Console*, fornecido pelo professor.

### Console

As funções relativas à consola permitem uma personalização da interface; a primeira função, *gotoxy*, permite personalizar coordenadas específicas da consola; *setcolor* permite alterar as cores do texto e de fundo e, finalmente, *clrscr* possibilita a limpeza do ecrã. Este conjunto de funções foi disponibilizado previamente pelo docente. A declaração das funções pode ser vista na figura 5.

```
// Move cursor to column 'x', line 'y'
void gotoxy(int x, int y);

//-----
// Set text color & background
void setcolor(unsigned int color, unsigned int background_color);

//-----
// Clear screen
void clrscr(void);
```

Figura 5 - Funções da Consola

### Game Report

As funções presentes no *Game Report* permitem criar ou abrir, se este já existir, um ficheiro que guarde o número total de jogos já realizados (função *gameCounter*) e criar, em cada jogo, um ficheiro que guarda as informações relativas a esse jogo, tais como, configurações iniciais, apostas de cada jogador, pontuações, etc. A declaração das funções pode ser vista na figura 6.

```
ofstream gameReportCreate(unsigned int& gameNumber); // create game Report file
void gameReportClose(ofstream& ofs); // close game Report file
unsigned int gameCounter(); //open or create file to increment number of games
bool fileExists(const string& filename); //check if file for game Counter exists
```

Figura 6 - Funções para a elaboração do relatório de jogo

### Configurations

As funções do ficheiro *Configurations* permitem a leitura do ficheiro onde estão definidas as configurações que podem ser variáveis no jogo, como o tempo de jogada (*gameDuration*), o número mínimo de letras numa palavra para esta ser considerada válida (*minLetters*) e o número mínimo de pontos que um jogador tem de ter para ganhar o jogo (*minPointsToWin*). Para guardar estas variáveis é criada uma struct de *configurations*. A função *readConfigAux* devolve o texto que está à frente de uma certa frase modelo e a *readConfigurations* abre o ficheiro e define estas frases modelo, associando cada string de saída da primeira função a um parâmetro das configurações. Por último, desenvolveu-se uma função de display, que permite a visualização na consola destas informações, no início de cada jogo. A declaração das funções e criação da struct *configurations* pode ser vista na figura 7.

```

struct Configurations
{
    string boardFile;
    string dictionaryFile;
    unsigned int gameDuration = 0;
    unsigned int minLetters = 0;
    unsigned int minPointsToWin = 0;
};

Configurations readConfigurations(const string& configFile); //read all configurations from file

string readConfigAux(ifstream& configFile, const string& configToRead); //read values from configurations file

void displayConfigurations(Configurations& config, ostream& os); //display game configurations

```

Figura 7 - Funções para a leitura das configurações e criação da struct *configurations*

### Position

O ficheiro *Position* foi criado para construir a struct *Position*, que contém as coordenadas de uma posição do tabuleiro, e declarar a função *showPath* que, tal como o nome indica, mostra o vetor *path* que é criado num dos métodos da classe *Board*. A declaração das funções e criação da struct *Position* pode ser vista na figura 8.

```

struct Position
{
    int row, col; // position of a cube (top letter) on the board
};

void showPath(const vector<Position>& path, ostream& os); //display vector path

```

Figura 8 - Função para mostrar o Path e criação da struct *Position*

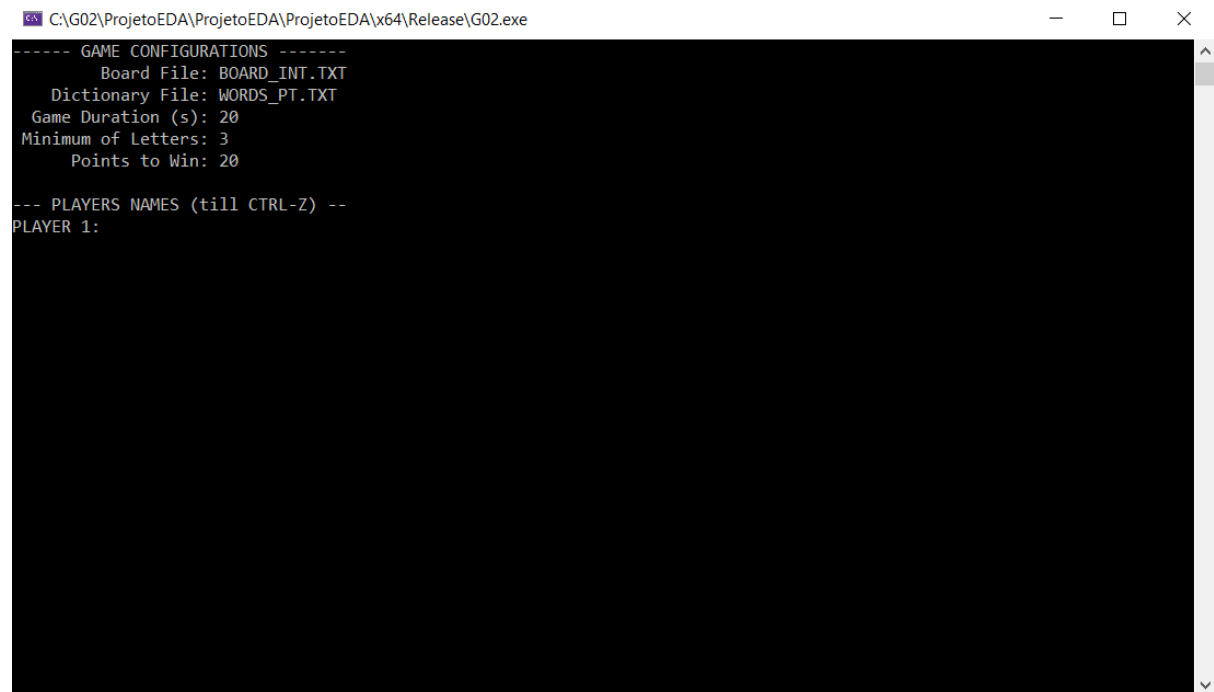
## Estado de desenvolvimento

Considera-se que todos os objetivos foram cumpridos, uma vez que o programa final permite que um, ou mais jogadores, joguem *Boggle*: o programa gera um tabuleiro com o tamanho definido no file disponibilizado; a partir deste, os jogadores (1 no mínimo) têm um intervalo de tempo para apostar em palavras e, se estas forem consideradas válidas (estão presentes no dicionário e é possível construir a palavras com letras do tabuleiro contíguas sem serem repetidas) são distribuídas pontuações com base no tamanho da palavra e é possível visualizar o path - coordenadas, no tabuleiro, das letras usadas na palavra. Para além disso, são consideradas as ocasiões em que dois jogadores jogam uma palavra igual na mesma ronda - esta palavra torna-se inválida e não lhe é atribuída uma pontuação. O jogo acaba quando, no final de uma ronda, um jogador obtém pontuação mínima ou a meio do jogo quando pressiona 'Q' em vez de enter entre uma ronda. Em caso de empate, jogam mais uma ronda para desempatar.

É de notar que, com o intuito de tornar o programa mais robusto e preparado para erros, foram utilizados dois blocos de *try-throw-catch* para lidar com exceções. Assim, quando um ficheiro não é aberto ou lido corretamente, o programa termina e apresenta uma mensagem de erro que explica aos jogadores o que aconteceu.

## Exemplo de execução

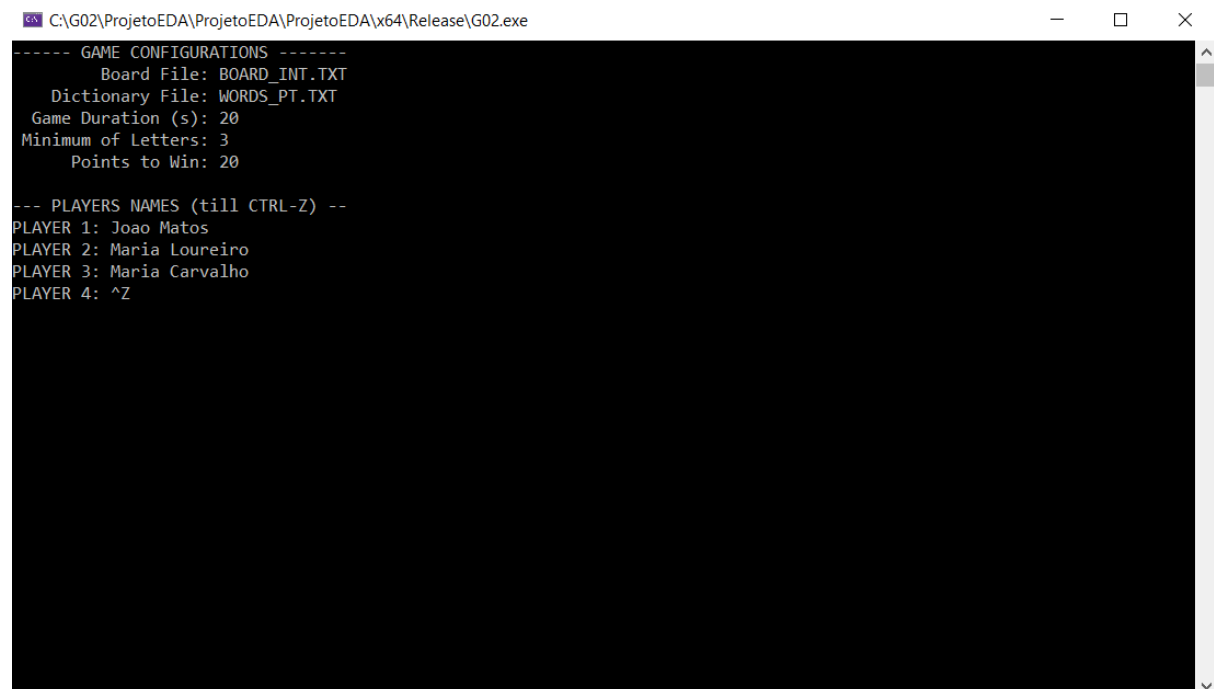
Inicialização do Jogo, mostram-se as configurações e os jogadores devem ser inseridos



```
C:\G02\ProjetoEDA\ProjetoEDA\x64\Release\G02.exe
----- GAME CONFIGURATIONS -----
      Board File: BOARD_INT.TXT
      Dictionary File: WORDS_PT.TXT
      Game Duration (s): 20
      Minimum of Letters: 3
      Points to Win: 20

--- PLAYERS NAMES (till CTRL-Z) --
PLAYER 1:
```

Os jogadores podem ser tantos quantos se queira, terminando com CTRL-Z



```
C:\G02\ProjetoEDA\ProjetoEDA\x64\Release\G02.exe
----- GAME CONFIGURATIONS -----
      Board File: BOARD_INT.TXT
      Dictionary File: WORDS_PT.TXT
      Game Duration (s): 20
      Minimum of Letters: 3
      Points to Win: 20

--- PLAYERS NAMES (till CTRL-Z) --
PLAYER 1: Joao Matos
PLAYER 2: Maria Loureiro
PLAYER 3: Maria Carvalho
PLAYER 4: ^Z
```

Jogadores recolhidos, mostra-se a tabela de pontuações inicial e pergunta se queremos começar uma nova ronda

```
C:\G02\ProjetoEDA\ProjetoEDA\ProjetoEDA\x64\Release\G02.exe

----- GAME CONFIGURATIONS -----
      Board File: BOARD_INT.TXT
      Dictionary File: WORDS_PT.TXT
      Game Duration (s): 20
      Minimum of Letters: 3
      Points to Win: 20

--- PLAYERS NAMES (till CTRL-Z) --
PLAYER 1: Joao Matos
PLAYER 2: Maria Loureiro
PLAYER 3: Maria Carvalho
PLAYER 4: ^Z

----- TABLE OF SCORES -----
PLAYER  NAME      SCORE
  1  Joao Matos      0
  2  Maria Loureiro  0
  3  Maria Carvalho  0

Board 4x4 loaded with success.

Dictionary is loading ...
Dictionary loaded with success.

Press ENTER to roll the dices and start a new round, or 'Q' to quit the game
```

Pressionando ENTER, o tabuleiro baralhado é mostrado e o 1º jogador pode começar a jogar, assim que pressionar ENTER novamente

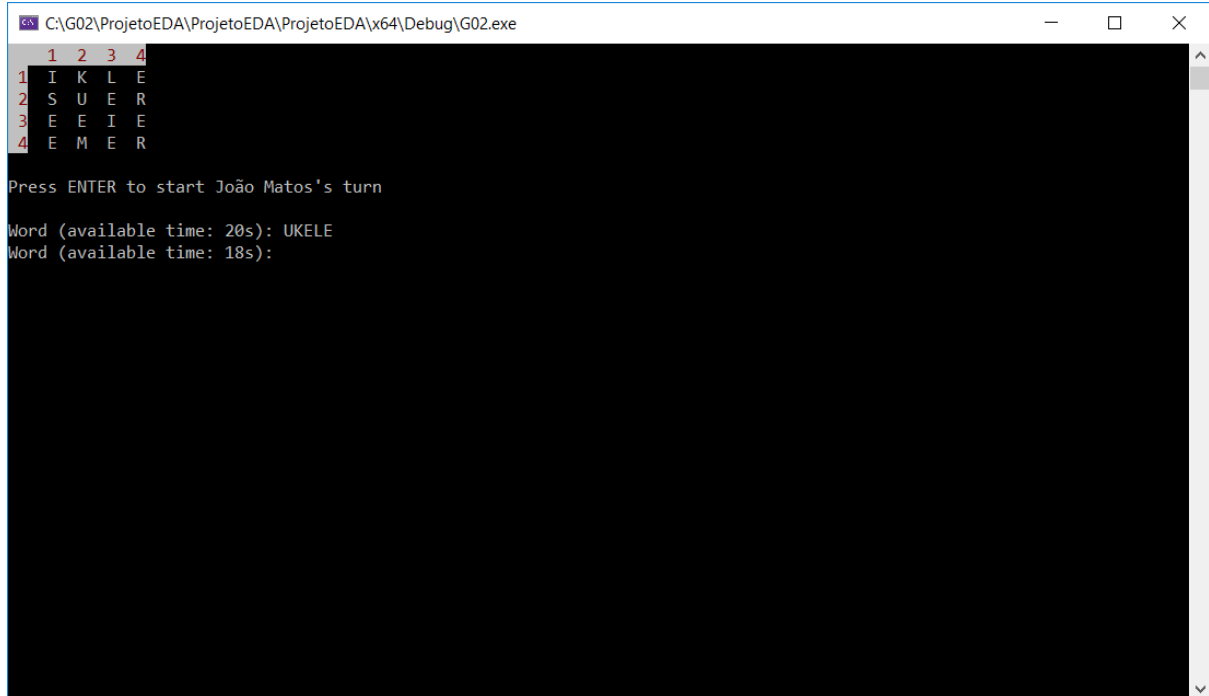
```
C:\G02\ProjetoEDA\ProjetoEDA\ProjetoEDA\x64\Debug\G02.exe

  1  2  3  4
1  I  K  L  E
2  S  U  E  R
3  E  E  I  E
4  E  M  E  R

Press ENTER to start João Matos's turn
```



Ao longo de 20s (tempo das configurações), o 1º jogador pode agora escrever palavras



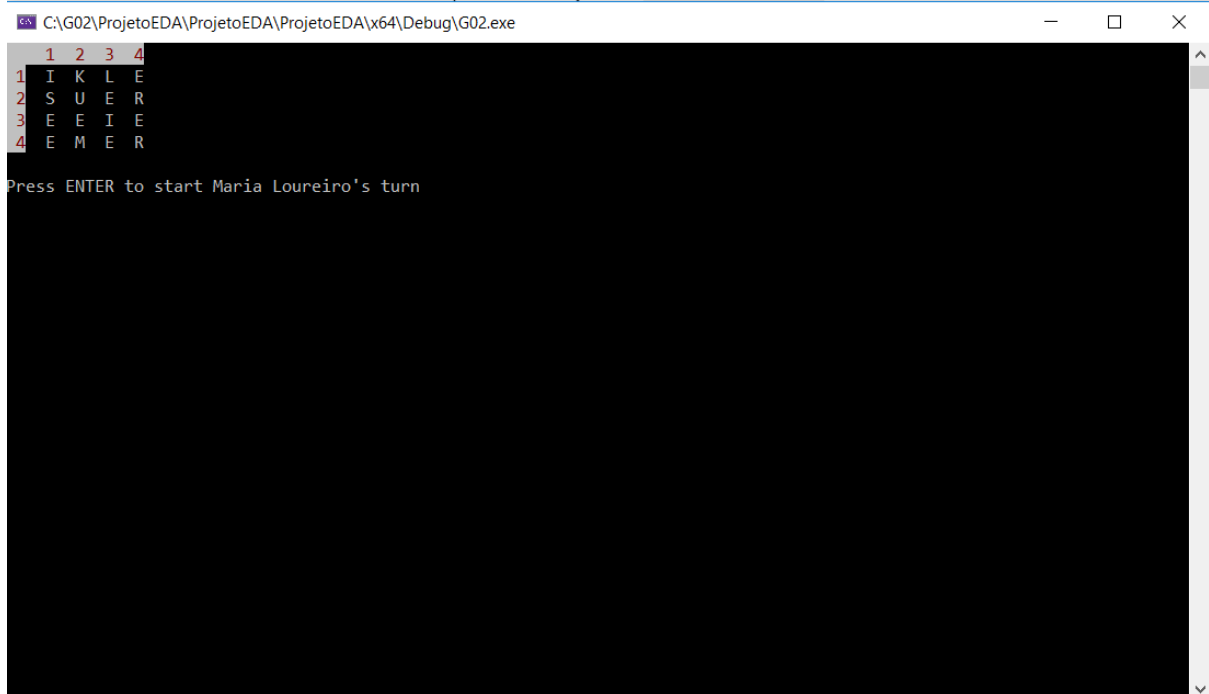
A screenshot of a game window titled "C:\G02\ProjetoEDA\ProjetoEDA\x64\Debug\G02.exe". The window displays a 4x4 grid of letters. The first row is "I K L E", the second is "S U E R", the third is "E E I E", and the fourth is "E M E R". Below the grid, the text "Press ENTER to start João Matos's turn" is visible. Further down, there are two lines of text: "Word (available time: 20s): UKELE" and "Word (available time: 18s):". The grid is highlighted with a red border, and the letters are in a light blue color.

	1	2	3	4
1	I	K	L	E
2	S	U	E	R
3	E	E	I	E
4	E	M	E	R

Press ENTER to start João Matos's turn

Word (available time: 20s): UKELE  
Word (available time: 18s):

Findos os 20s, o ecrã é limpo para que o jogador seguinte não veja as palavras do anterior

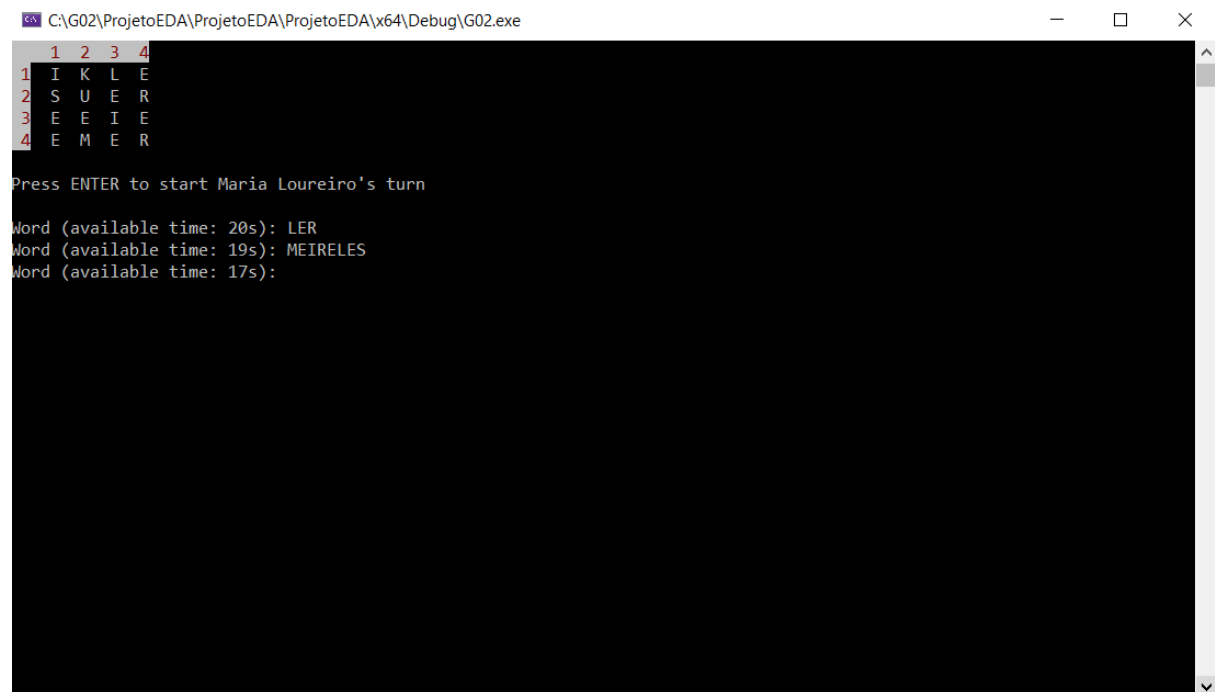


A screenshot of the same game window after the word input prompt has been cleared. The 4x4 grid of letters remains the same, but the text "Press ENTER to start João Matos's turn" is now "Press ENTER to start Maria Loureiro's turn". The word input prompt is no longer visible.

	1	2	3	4
1	I	K	L	E
2	S	U	E	R
3	E	E	I	E
4	E	M	E	R

Press ENTER to start Maria Loureiro's turn

Pressionando ENTER, o 2º jogador pode agora apostar as suas palavras



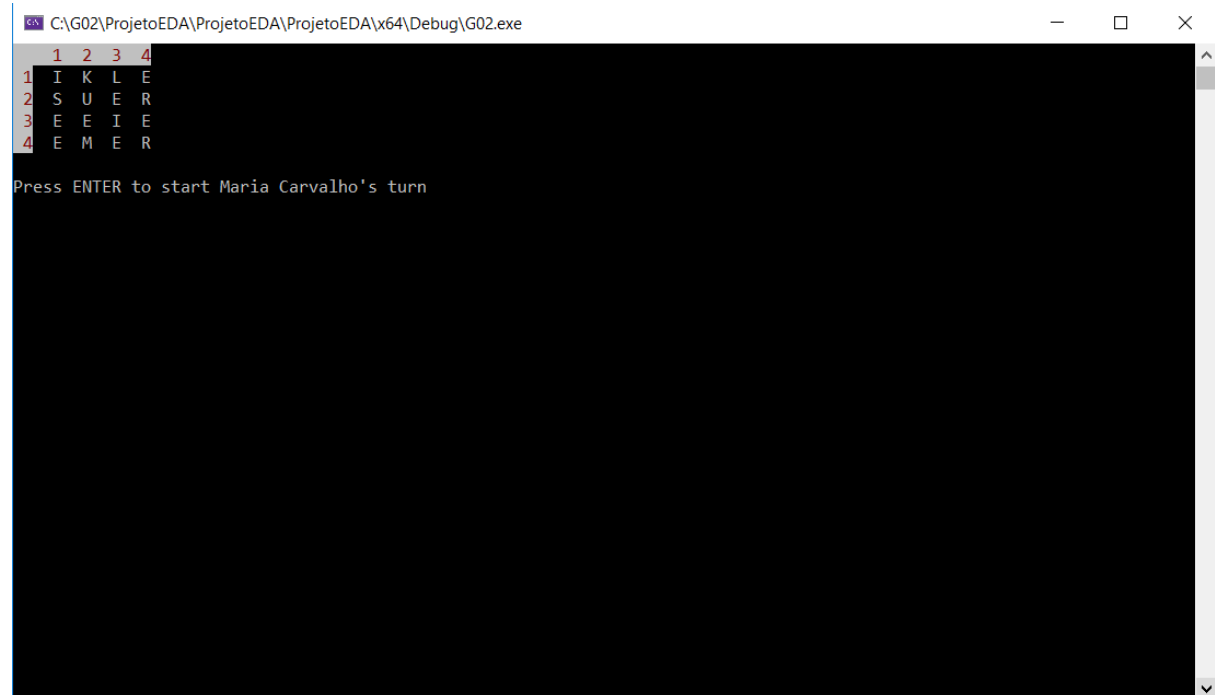
A screenshot of a game window titled "C:\G02\ProjetoEDA\ProjetoEDA\x64\Debug\G02.exe". The window displays a 4x4 grid of letters. The first row contains 'I', 'K', 'L', 'E'. The second row contains 'S', 'U', 'E', 'R'. The third row contains 'E', 'E', 'I', 'E'. The fourth row contains 'E', 'M', 'E', 'R'. Below the grid, the text "Press ENTER to start Maria Loureiro's turn" is displayed. Below that, a list of words is shown: "Word (available time: 20s): LER", "Word (available time: 19s): MEIRELES", and "Word (available time: 17s):".

	1	2	3	4
1	I	K	L	E
2	S	U	E	R
3	E	E	I	E
4	E	M	E	R

Press ENTER to start Maria Loureiro's turn

Word (available time: 20s): LER  
Word (available time: 19s): MEIRELES  
Word (available time: 17s):

Ao fim de 20s, o ecrã é limpo e começa o turno do 3º jogador



A screenshot of a game window titled "C:\G02\ProjetoEDA\ProjetoEDA\x64\Debug\G02.exe". The window displays a 4x4 grid of letters. The first row contains 'I', 'K', 'L', 'E'. The second row contains 'S', 'U', 'E', 'R'. The third row contains 'E', 'E', 'I', 'E'. The fourth row contains 'E', 'M', 'E', 'R'. Below the grid, the text "Press ENTER to start Maria Carvalho's turn" is displayed. Below that, a list of words is shown: "Word (available time: 20s): LER", "Word (available time: 19s): MEIRELES", and "Word (available time: 17s):".

	1	2	3	4
1	I	K	L	E
2	S	U	E	R
3	E	E	I	E
4	E	M	E	R

Press ENTER to start Maria Carvalho's turn

Word (available time: 20s): LER  
Word (available time: 19s): MEIRELES  
Word (available time: 17s):

Pressionando ENTER, o 3º jogador tem também 20s para jogar

```
C:\G02\ProjetoEDA\ProjetoEDA\ProjetoEDA\x64\Debug\G02.exe
 1 2 3 4
1 I K L E
2 S U E R
3 E E I E
4 E M E R

Press ENTER to start Maria Carvalho's turn

Word (available time: 20s): MEU
Word (available time: 19s): SEU
Word (available time: 17s): LER
Word (available time: 14s):
```

Terminadas os turnos de cada jogador, é feita limpeza do ecrã e são mostradas todas as palavras apostadas, por jogador, acompanhadas pelos pontos, justificação e caminho, caso a palavra seja válida. No final, a tabela de pontuação ao fim da 1ª ronda é atualizada.

```
C:\G02\ProjetoEDA\ProjetoEDA\ProjetoEDA\x64\Debug\G02.exe
 1 2 3 4
1 I K L E
2 S U E R
3 E E I E
4 E M E R

Round 1 RESULTS:

João Matos's round results:
      UKELE | SCORE: 0 | Word not found in dictionary
      ÔK  | SCORE: 0 | Word has not minimum letters

Maria Loureiro's round results:
      LER | SCORE: 0 | Word already written by other players
      MEIRELES | SCORE: 0 | Word cannot be written on the board

Maria Carvalho's round results:
      MEU | SCORE: 1 | Word is valid | PATH: (4,2) -> (3,1) -> (2,2)
      SEU | SCORE: 1 | Word is valid | PATH: (2,1) -> (3,1) -> (2,2)
      LER | SCORE: 0 | Word already written by other players

----- TABLE OF SCORES -----
PLAYER  NAME      SCORE
 1 João Matos      0
 2 Maria Loureiro  0
 3 Maria Carvalho  2

Press ENTER to roll the dices and start a new round, or 'Q' to quit the game
```

Repetindo as rondas, da forma mostrada anteriormente, vão se obtendo os resultados e a tabela de pontuação vai sendo atualizada. Para a 2ª ronda, apenas a 3ª jogadora apostou palavras, com o seguinte resultado:

```
C:\G02\ProjetoEDA\ProjetoEDA\ProjetoEDA\Debug\G02.exe
  1  2  3  4
1 O K D T
2 T N U O
3 L S E E
4 M E N Z

Round 2 RESULTS:

João Matos's round results:

No words to process

Maria Loureiro's round results:

No words to process

Maria Carvalho's round results:

TUDO | SCORE: 1 | Word is valid | PATH: (1,4) -> (2,3) -> (1,3) -> (2,4)
MEL | SCORE: 1 | Word is valid | PATH: (4,1) -> (4,2) -> (3,1)
ELISE | SCORE: 0 | Word not found in dictionary
ZES | SCORE: 1 | Word is valid | PATH: (4,4) -> (3,3) -> (3,2)
LION | SCORE: 0 | Word not found in dictionary
ZEUS | SCORE: 1 | Word is valid | PATH: (4,4) -> (3,3) -> (2,3) -> (3,2)
TUS | SCORE: 0 | Word not found in dictionary
DOE | SCORE: 1 | Word is valid | PATH: (1,3) -> (2,4) -> (3,3)
NUO | SCORE: 0 | Word not found in dictionary
SEM | SCORE: 1 | Word is valid | PATH: (3,2) -> (4,2) -> (4,1)
SUE | SCORE: 1 | Word is valid | PATH: (3,2) -> (2,3) -> (3,3)
NUDO | SCORE: 0 | Word not found in dictionary

----- TABLE OF SCORES -----
PLAYER NAME SCORE
1 João Matos 0
2 Maria Loureiro 0
3 Maria Carvalho 9

Press ENTER to roll the dices and start a new round, or 'Q' to quit the game
```

O procedimento foi repetido para a 3ª ronda, obtendo-se os seguintes resultados:

```
C:\G02\ProjetoEDA\ProjetoEDA\ProjetoEDA\Debug\G02.exe
  1  2  3  4
1 E S G X
2 P O T I
3 I T B L
4 S T R K

Round 3 RESULTS:

João Matos's round results:

No words to process

Maria Loureiro's round results:

No words to process

Maria Carvalho's round results:

SITO | SCORE: 0 | Word not found in dictionary
PISTO | SCORE: 0 | Word not found in dictionary
LITO | SCORE: 0 | Word not found in dictionary
LITOS | SCORE: 0 | Word not found in dictionary
GIL | SCORE: 1 | Word is valid | PATH: (1,3) -> (2,4) -> (3,4)
SOPE | SCORE: 1 | Word is valid | PATH: (1,2) -> (2,2) -> (2,1) -> (1,1)
GOI | SCORE: 0 | Word not found in dictionary
GOTS | SCORE: 1 | Word is valid | PATH: (1,3) -> (2,2) -> (3,1) -> (4,1)

----- TABLE OF SCORES -----
PLAYER NAME SCORE
1 João Matos 0
2 Maria Loureiro 0
3 Maria Carvalho 12

Press ENTER to roll the dices and start a new round, or 'Q' to quit the game
```

Finalmente, na 4ª ronda, a 3ª jogadora conseguiu atingir os 20 pontos, tornando-se vencedora. Caso obtivesse mais pontos, seria vencedora na mesma; caso não atingisse os 20 pontos ou houvesse um empate, uma nova ronda seria jogada.

```
Microsoft Visual Studio Debug Console

1 2 3 4
1 I N E S
2 I G C N
3 A T E N
4 B E O E

Round 4 RESULTS:

João Matos's round results:

No words to process

Maria Loureiro's round results:

No words to process

Maria Carvalho's round results:

IATE | SCORE: 1 | Word is valid | PATH: (2,1) -> (3,1) -> (3,2) -> (3,3)
META | SCORE: 1 | Word is valid | PATH: (2,4) -> (3,3) -> (3,2) -> (3,1)
NETO | SCORE: 1 | Word is valid | PATH: (2,4) -> (3,3) -> (3,2) -> (4,3)
MOTE | SCORE: 1 | Word is valid | PATH: (3,4) -> (4,3) -> (3,2) -> (3,3)
MOE | SCORE: 0 | Word not found in dictionary
BATE | SCORE: 1 | Word is valid | PATH: (4,1) -> (3,1) -> (3,2) -> (3,3)
GATO | SCORE: 1 | Word is valid | PATH: (2,2) -> (3,1) -> (3,2) -> (4,3)
INES | SCORE: 1 | Word is valid | PATH: (1,1) -> (1,2) -> (1,3) -> (1,4)
GIN | SCORE: 0 | Word not found in dictionary
NEO | SCORE: 0 | Word not found in dictionary
META | SCORE: 1 | Word is valid | PATH: (3,4) -> (3,3) -> (3,2) -> (3,1)

----- TABLE OF SCORES -----
PLAYER NAME SCORE
1 João Matos 0
2 Maria Loureiro 0
3 Maria Carvalho 20

----- THE WINNER IS -----
Maria Carvalho, WITH A SCORE OF 20 POINTS.

C:\G02\ProjetoEDA\ProjetoEDA\ProjetoEDA\x64\Debug\G02.exe (process 32212) exited with code 0.
Press any key to close this window . . .
```

Encontrado o vencedor, o programa termina. Também se poderia terminar o programa entre rondas, escrevendo 'Q' em vez de ENTER para continuar. Nesses casos não haveria vencedor e seria apenas mostrada a tabela de pontos final.