Projeto de EDA 2019/2020

Jogo de palavras

BOGGLE

Projeto executado por:
Duarte Rodrigues
Mariana Xavier
Ricardo Brioso

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

1. Classes

O desenvolvimento do projeto baseou-se no desenvolvimento de 6 classes distintas de forma a organizar e dividir os elementos fundamentais do jogo.

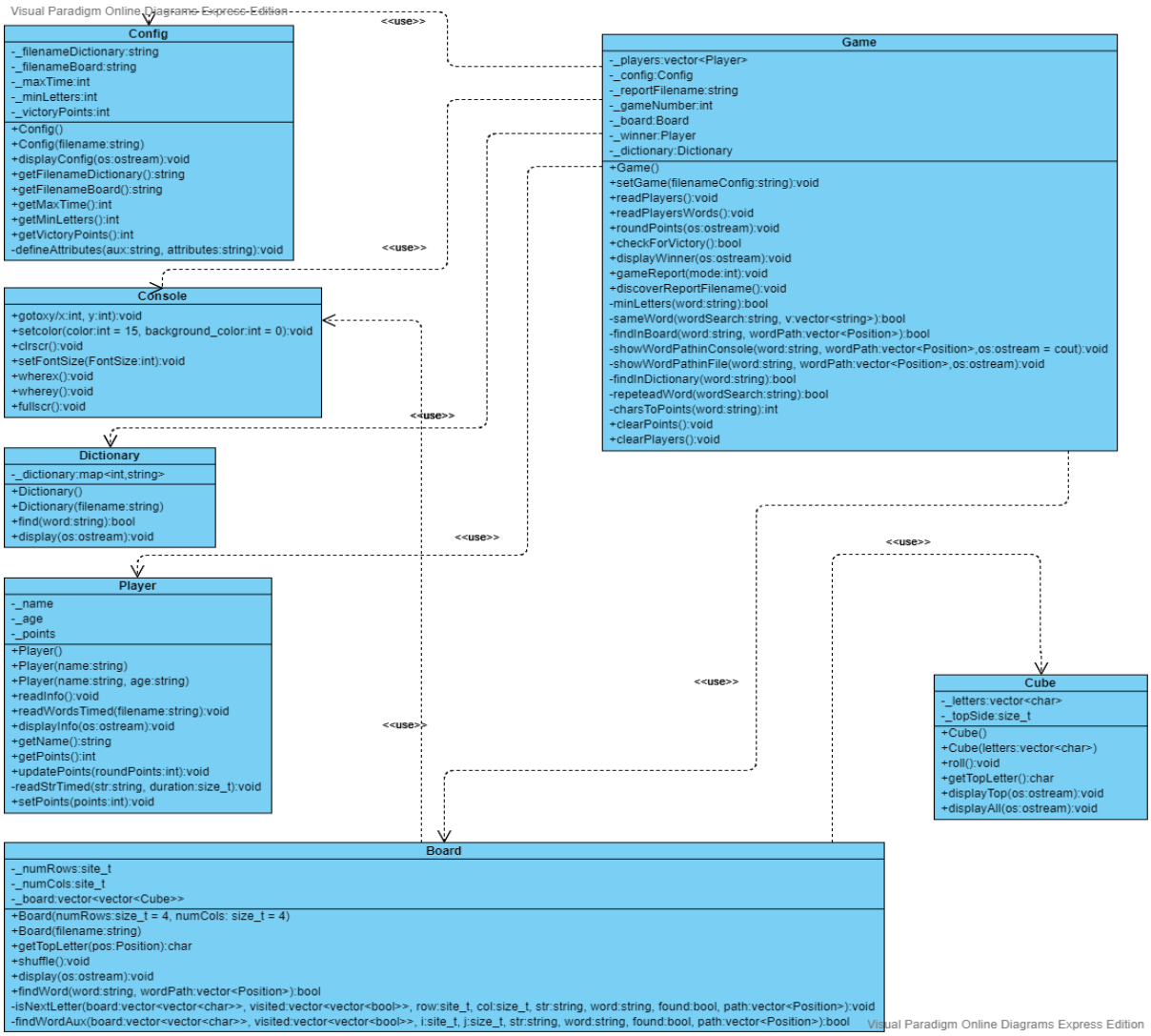


Figura 1. Diagrama de classes relativo ao funcionamento geral do jogo.

★ Cube

A primeira classe a ser implementada foi a classe ***Cube***, a qual tem como objetivo representar os dados usados num tabuleiro de jogo. Assim, a esta classe foi definida com dois atributos - vetor de *char* chamado *_letters,* que representa as 6 letras que se encontram nas faces dos cubos, e um valor numérico que se refere à posição do cubo que traduz a letra que se encontra em cima, ou seja, *_topSide*.

No que toca aos métodos esta classe apresenta 2 construtores, um sem argumentos, *Cube()*, em que o dado é iniciado com asteriscos nas suas faces e a face de cima é a primeira posição do vetor. O segundo construtor tem um argumento para atribuir letras às faces do dado, *Cube(const vector<char>& letters)*, contudo a face de cima continua a ser a primeira posição do vetor.

De seguida, o método para rolar o dado, *void roll()*, em que este não retorna nenhum valor, simplesmente muda o atributo *_topSide* para um valor aleatório. Depois aparece o método para obter a letra que está visível no topo do dado, *char getTopLetter() const*. Por fim, temos 2 funções que vão apresentar numa *output stream* (argumento das funções) a letra da face que está virada para cima, *void displayTop(ostream& os) const*, ou as letras das faces todas, *void displayAll(ostream& os) const*. Como a informação obtida por estas 3 últimas funções não muda esta pode ser definida como uma constante, o que também previne que esta seja mudada, evitando erros de programação.

★ Board

A partir do momento que conseguimos definir um cubo/dado, podemos começar a distribuí-los num tabuleiro. Na implementação do jogo, o tabuleiro é definido pela classe ***Board***. Os dados nele presentes organizam-se em linhas e colunas, sendo assim o atributo principal da classe um vetor 2D de cubos chamado *_board*. Outros 2 atributos muito importantes, que vão definir a dimensão do tabuleiro, é o número de linhas *_numRows* e colunas *_numCols*.

A implementação desta classe é um pouco mais complexa do que a *Cube*, o que levou à criação de vários métodos público e privados.

Os métodos públicos iniciam-se com 2 construtores, em que o primeiro tem como argumentos 2 inteiros ,em que por omissão têm o valor 4, referentes aos atributos de número de linhas e colunas, *Board(size_t numRows = 4, size_t numCols = 4)*, construindo um tabuleiro com a dimensões especificadas, com cubos que só têm asteriscos nas suas faces. O segundo construtor cria um tabuleiro a partir das especificações, o número de linhas, colunas e a composição de cada um dos cubos, declaradas num ficheiro de texto, *Board(const string& filename)*. Caso o ficheiro não exista ou caso o ficheiro não esteja escrito de acordo com o *template* adequado, este manda uma mensagem de erro.

Para sabermos a letra que está no topo do dado numa posição específica do tabuleiro foi criada a função *char getTopLetter(const Position& pos) const*. No decorrer do jogo será necessário alterar constantemente, ronda após ronda, a instância da *board*, sendo por isso, necessário implementar um método que baralhe os dados, *void shuffle()*, isto é, que altera aleatoriamente a posição dos dados dentro do vetor 2D, e, após isso, rola todos os dados, com a função da classe *Cube*. Como não podia faltar será também necessário mostrar o tabuleiro na *outstream* desejada, sendo para isso criada a função *void display(ostream& os) const*, na qual a *board* é apresentada com as respetivas linhas e colunas, indexadas de 1 a x, a azul, o que facilitará posteriormente a identificação do caminho percorrido para encontrar as palavras.

O segundo método mais exigente apresentado nesta classe é o responsável por encontrar as palavras no tabuleiro, *bool findWord(string word, vector<Position>& wordPath)*, no qual o objetivo é pegar na palavra que o jogador apostou e tentar reconstruí-la somando letras adjacentes no tabuleiro. Nesta função em específico corremos o tabuleiro à procura da primeira letra que inicia a palavra que queremos testar. Quando encontrada essa letra fica registada como visitada/lida, o que impede que possa ser usada novamente e a sua posição é também guardada num vetor, que irá ser responsável por guardar as posições das consecutivas letras até a palavra estar formada. De seguida é chamada uma função auxiliar para encontrar o resto das letras, até a palavra estar completa, *bool findWordAux(const vector<vector<char>>& board, vector<vector<bool>>& visited, size_t i, size_t j, string& str, const string& word, bool& found, vector<Position>& path)*. Este método e os a ele associados

usam uma técnica de *recursive backtracking*, no qual, a iteração seguinte só é chamada caso todos os critérios de passagem sejam cumpridos. Esta função auxiliar avalia se as 8 posições adjacentes à primeira letra encontrada, começando no topo esquerdo e acabando na base direita, têm alguma letra que permite construir a palavra. Se sim , então adiciona-a a uma string de comparação e é, também, chamada uma função *void isNextLetter(const vector<vector<char>>& board, vector<vector<bool>>& visited, size_t row, size_t col, string& str, const string& word, bool& found, vector<Position>& path)* que permite verificar se essa letra é concordante com a palavra molde do jogador, se a reconstrução da palavra está a ser correta e se a posição da nova letra pode ser guardada. Na eventualidade de a palavra não conseguir ser formada pelo caminho que se estava a seguir, então ela volta à iteração anterior e continua à procura, assim sucessivamente, até ao fim do tabuleiro, dando o veredito final se encontrou ou não a palavra.

★ Player

A classe ***Player*** possui três atributos: *_name*, *_age* e *_points*. Estes representam o nome, a idade e o número de pontos que o jogador fez durante o jogo.
Possui três construtores, um vazio, e outros dois que dependem dos parâmetros disponíveis, inicializando com o nome e/ou idade definidos pelos parâmetros: *Player(); Player(string name); Player(string name, string age).*

Descrevendo os métodos mais simples primeiro, existem dois métodos: um permite ler a informação de um jogador a partir da consola, *void readInfo()* (nome e idade) e o outro permite imprimir essa informação na consola, *void displayInfo()*. Para visualizar esta informação numa *output stream* é possível recorrer à função *void displayInfo(ostream& os) const*.

Para além desses, existem os métodos *string getName()* e *unsigned int getPoints()*, que retornam o nome e os pontos do jogador, respetivamente. Existe também, um método que é usado para somar os pontos obtidos numa nova ronda aos pontos previamente existentes do jogador, que é o método *void updatePoints(unsigned int roundPoints)*, e um método que permite definir os pontos através do seu parâmetro, chamado *void setPoints(unsigned int newPoints)*.

Os métodos mais complexos desta classe, *void readStrTimed(string& str, time_t duration)* e *void readWordsTimed(const string& fileName, time_t duration)* foram implementados pelo professor, sendo posteriormente adpatados. O primeiro é um método privado e é usado para ler palavras individuais introduzidas pelo utilizador e parar de o fazer quando a duração da jogada terminar. A contabilização do tempo começa quando a função é chamada e é cronometrado num ciclo *while* que também trata de analisar a variável do tipo *string* que está a ser introduzida. A escrita da *string* é finalizada quando o utilizador pressiona a tecla espaço, *tab* ou *enter*. O segundo utiliza o anterior para ler uma sequência de *strings* dentro do tempo disponível e para comunicar ao utilizador através da consola que foi introduzida uma palavra e qual o tempo restante que ele possui para introduzir mais palavras.

★ Dictionary

De seguida tornou-se necessário implementar a classe ***Dictionary*** que, tal como o nome indica, contém a lista das palavras válidas, apelidada de dicionário, que servirá para futura comparação com as palavras introduzidas pelos jogadores. Para tal recorre a um único atributo, que é um mapa cujos pares são constituídos na primeira posição por um índice e na segunda posição pela palavra a ele correspondente, *map<int, string> _dictionary*.

Quanto aos seus métodos, esta classe possui dois construtores. O primeiro é um construtor vazio, *Dictionary()*, que insere no mapa um par meramente indicativo, cujos valores são 0 e *"NO WORDS"* para a primeira e segunda posição deste, respetivamente. O segundo construtor recebe como parâmetro o nome de um ficheiro e, após testar se esse ficheiro existe, executa um ciclo que vai lendo dele para a segunda posição dos diferentes pares, incrementando sempre o valor da primeira posição, *Dictionary(const string& filename)*. Nesta leitura é de notar que as palavras são passadas para letras maiúsculas, devido ao facto de as palavras introduzidas pelos jogadores também se encontrarem em maiúsculas.

Para além disso, possui uma função que permite mostrar o dicionário guardado numa *output stream*, sendo esta passada como parâmetro, *void display(ostream& os) const*. Por fim, possui a função que permite descobrir se uma determinada palavra, que lhe é passada como parâmetro, está ou não presente no dicionário, retornando um booleano, *bool find(const string& word)*. Para tal é criado um iterador constante, que aponta para e percorre os diferentes pares do mapa, verificando a igualdade dessa palavra com o segundo elemento do par.

★ Config

A partir do momento em que todas as classes anteriores estão implementadas, é possível conjugá-las e criar a classe ***Config***, que fornece a configuração geral do jogo. Para tal, tem como atributos o nome do ficheiro em que se encontra o dicionário, *_filenameDictionary*, o nome do ficheiro em que se encontra a *board, _filenameBoard*, o número máximo de tempo por cada jogada, *_maxTime*, o número mínimo de letras por palavra, *_minLetters*, e o número de pontos necessários para um jogador vencer o jogo, *_victoryPoints*.

O primeiro construtor é um construtor vazio *Config()*, que atribui ao nome dos ficheiros *"NO FILE"* e o valor de 0 aos restantes atributos. Quanto ao construtor que recebe como parâmetro o ficheiro com a configuração, *Config(const string& filename)*, este testa a abertura do ficheiro e retira em primeiro lugar o título e um separador. De seguida, retira linha a linha do ficheiro aberto, colocando num parâmetro auxiliar e recorre a função privada, *void defineAttributes(string aux, string& atribute)*, que retira a informação essencial de cada linha e a relaciona com cada um dos atributos.

Depois, possui uma função que permite visualizar a configuração numa *output stream*, *void displayConfig(ostream& os) const* e diversas funções *get* que permitem retornar os diferentes 5 atributos da classe: *string getFilenameDictionary() const; string getFilenameBoard() const; unsigned int getMaxTime() const; unsigned int getMinLetters() const; unsigned int getVictoryPoints() const*.

★ Console

Como o *Boggle* foi planeado para ser jogado na consola, foram utilizadas algumas funções que permitem alterar e manipular certos elementos gráficos, dando uma melhor jogabilidade e experiência. Estas funções atuam diretamente nas definições da consola, não sendo necessário formar uma classe, contudo estão declaradas na *header* ***Console***. Esta começa com uma lista de cores que vão ser utilizadas pela função *void setcolor(unsigned int color=15, unsigned int background_color=0)*, que permite alterar a cor das letras da consola e/ou do fundo. Por omissão, é exibido letras brancas com fundo preto. A função *void gotoxy(int x, int y)* permite mover o cursor da consola para qualquer ponto indicado, fazendo

aí o *display* do que foi enviado (mensagens de texto, alteração de cor, etc). A função *void clrscr(void)* limpa o ecrã da consola, deixando uma tela vazia.

De seguida foram adicionadas também algumas funções que não foram fornecidas pelo docente, entre as quais, a função *void setFontSize(int FontSize)* que permite alterar o tamanho das letras da consola e as funções *int wherex()* e *int wherey()* que permitem descobrir a posição atual em que o cursor se encontra na consola.

★ Game

Por fim, pode-se finalmente implementar a classe que relaciona tudo, a classe ***Game***. Esta possui como atributos um vetor de jogadores *vector<Player> _players*, uma configuração *Config _config*, uma lista de palavras *Dictionary _dictionary*, um tabuleiro *Board _board* e o jogador vencedor do jogo *Player _winner*.

Possui um construtor vazio *Game()* que chama os construtores também vazios da configuração, dicionário e *board*. Para definir estes atributos existe uma função *set* que lê a configuração, e através dos seus atributos lê o dicionário e a *board, void setGame(const string& filenameConfig)*.

De seguida, apresenta a função que permite ler os diferentes jogadores, *void readPlayers(),* recorrendo a uma função da classe e fazendo um *push_back* ao vetor que os contém. Esta leitura termina quando for pressionado CTRL+Z no nome de um dos jogadores. Tendo isto, o jogo propriamente dito pode começar recorrendo a uma função que lê as palavras das apostas dos jogadores de uma ronda, *void readPlayersWords()*. Esta começa por dar *shuffle* na *board* e dispô-la no ecrã, por ler a duração máxima da ronda e criar um ficheiro com o nome do jogador. Por fim, recorre à função previamente explicada da classe *Player* para ler as palavras do jogador, guardando-as no seu ficheiro e após o tempo definido passa para o jogador seguinte, limpando o ecrã primeiro. Com as palavras lidas, torna-se necessário testá-las e atribuir pontos caso estas sejam válidas, *void roundPoints(ostream& os)*. Para tal abre o ficheiro de cada jogador à vez, mostra a *board* e as palavras obtidas com a respetiva pontuação, caso esta seja válida, ou o motivo pela qual a torna inválida. Para tal recorre a cinco funções privadas que permitem realizar cinco testes diferentes:

1. *bool minLetters(const string word)*: testa se a palavra tem o número mínimo de letras necessárias;
2. *bool findInBoard(const string word, vector<Position>& wordPath)*: testa se é possível formar aquela palavra com a *board* disponibilizada;
3. *bool findInDictionary(const string word)*: testa se a palavra se encontra no dicionário;
4. *bool repeatedWord(const string wordSearch):* Testa se a palavra foi também escolhida por outro jogador;
5. *bool sameWord(const string wordSearch, vector<string> v)*: Testa se a palavra foi escolhida várias vezes na mesma ronda pelo mesmo jogador, impedindo a falsa acumulação de pontos.

Caso todos estes testes admitam que a palavra seja aceite, esta é convertida em pontos tendo como critério o seu tamanho, segundo as regras oficiais, *int charsToPoints(const string word)*. Como anteriormente foi testado com a função *findInBoard*, então temos também registada a informação do caminho necessário a percorrer para formar a palavra. Este é demonstrado mudando a cor da palavra à medida que as letras são associadas e a sua posição indexada.

Quanto à vitória de um dos jogadores existem duas funções: uma que percorre os jogadores e testa se algum destes já atingiu o valor estipulado na configuração de pontos que

lhe permitem alcançar a vitória, *bool checkForVictory()*, e a função que mostra o vencedor, *void displayWinner(ostream& os)*.

Há duas funções que são usadas para criar num ficheiro de texto o resumo do jogo ocorrido. A função *void discoverReportFilename()* que descobre qual é o número do jogo a ser realizado através do ficheiro de texto "BOGGLE_GAME_NUM.TXT". A função *void gameReport(const unsigned int mode)* serve para criar o relatório do jogo realizado. O relatório contém a seguinte informação: o número do jogo no nome de ficheiro, o nome e idade dos jogadores, a instância de cada tabuleiro usado seguido das apostas de cada jogador nesse tabuleiro e o vencedor do jogo. Esta função apenas escreve no relatório do jogo a informação sobre os jogadores e o vencedor sendo que as instâncias dos tabuleiros usados durante o jogo e as apostas dos jogadores são escritos no ficheiro pela função *roundPoints*.

Por fim, possui duas funções que permitem limpar tanto os pontos dos vários jogadores, *void clearPoints()*, como o vetor com as suas informações, *void clearPlayers()*.

★ Main

O **Main** do projeto é a fase final da implementação onde se concretiza o uso das classes declaradas anteriormente, sendo estas incluídas no início deste ficheiro. O main possui 4 funções que vão relacionar as funções das classes. Começa por explicar as regras do jogo que são obtidas pelo ficheiro de configuração, numa função *void rules(Config c)*. No que toca à jogabilidade e aos dados relativos ao jogo foram criadas 3 funções, todas baseadas na classe *Game*. De forma a inicializar o jogo, os seus dados e proceder à leitura dos diversos jogadores foi criada uma função *void set(Game& game)*. De seguida, como o *Boggle* consiste na repetição de rondas, onde aparece sempre uma *board* e as apostas de palavras dos jogadores, até se encontrar o vencedor, então foi criada uma função que faz um ciclo que traduz isto mesmo, *void loop(Game& game)*. Por fim como se trata de um jogo é importante dar opções básicas ao jogador, sendo para isso criado um menu simples, evidenciado na função *void menu(Game& game)*, que permite saber se o jogador quer começar um jogo novo ou terminar/fechar o jogo. Após se descobrir o vencedor, também há a opção de voltar a jogar, mantendo-se os jogadores, ou de começar um jogo novo, com jogadores diferentes ou de sair. Em ambas os pontos são reiniciados.

2. Estado de desenvolvimento

- O programa corre em modo de consola e foram usadas cores para mostrar o caminho da palavra no tabuleiro.
- A informação sobre a configuração do jogo é lida do ficheiro "BOGGLE_CONFIG.TXT", essa informação é: o nome do ficheiro da *board*, o nome do ficheiro da lista de palavras, o tempo máximo de duração de cada ronda, o número mínimo de letras que deve ter uma palavra, e o número de pontos necessário para ganhar o jogo.
- O nome dos jogadores é lido até ser pressionado o CTRL+Z.
- O jogo é preparado ao ler a *board* disponível e o dicionário (lista de palavras válidas).
- O tabuleiro é mostrado no ecrã e os jogadores escrevem as suas apostas dentro do tempo limite.

- As palavras são verificadas (se existem na lista de palavras e se são válidas de acordo com as regras do jogo) e a pontuação das palavras/pontuação total é mostrada para cada jogador. Para as palavras cuja pontuação é 0, é dada uma explicação.

- O jogador que atingir a pontuação máxima primeiro é o vencedor.

- Um relatório do jogo,"BOGGLE_GAME_X.TXT" é criado ao longo do jogo contendo a informação de todos os jogadores, e as instâncias do tabuleiro usadas juntamente com as apostas de cada jogador para essa instância e a sua respetiva validade e pontuação.

3. Exemplo de execução

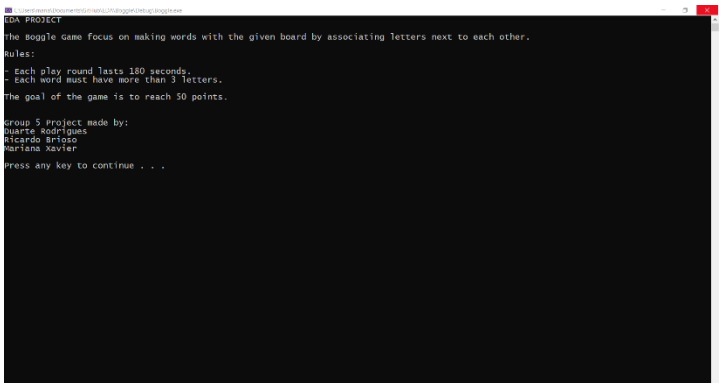


Figura 2. Apresentação regras do jogo.

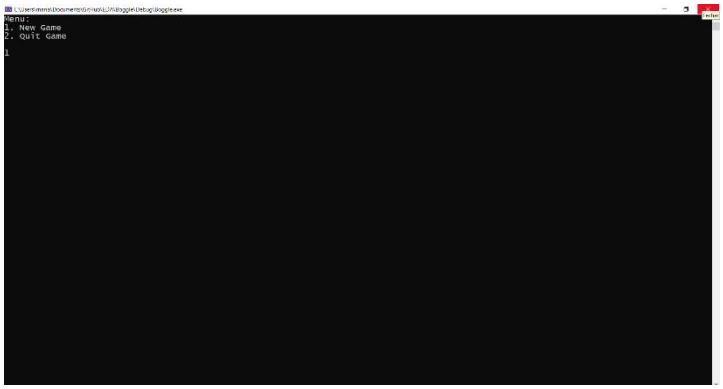


Figura 3. Primeira possibilidade de menu, escolhendo opção 1.

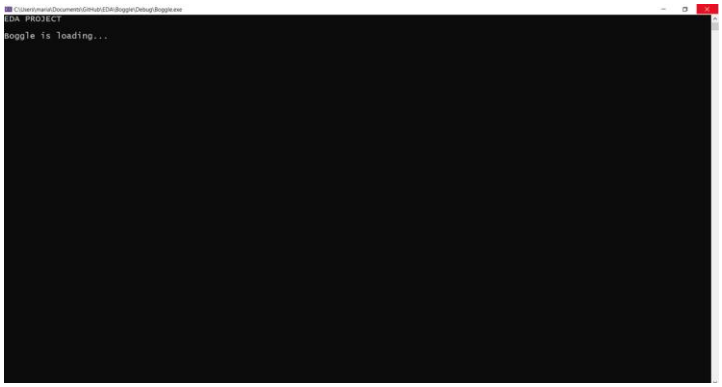


Figura 4. Janela enquanto o jogo é carregado.

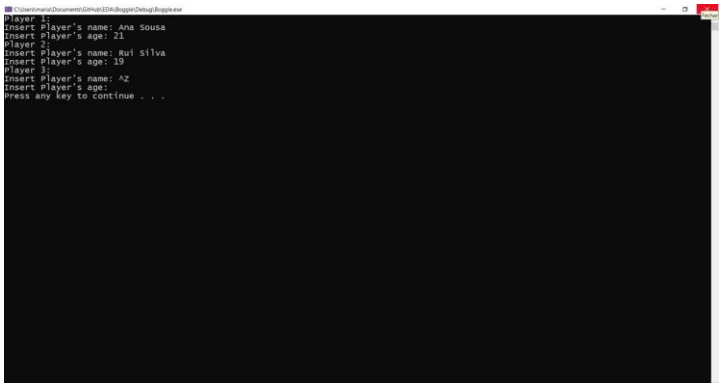
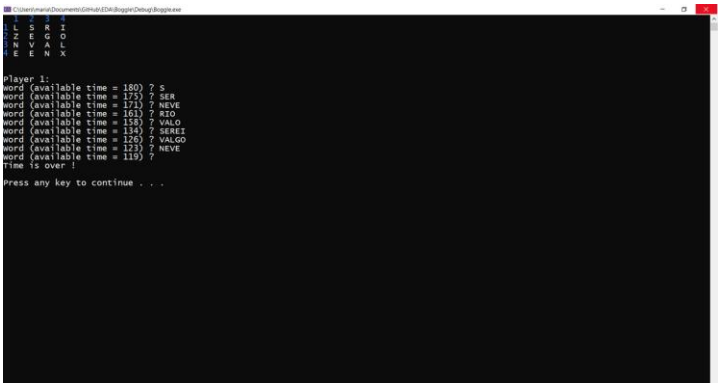
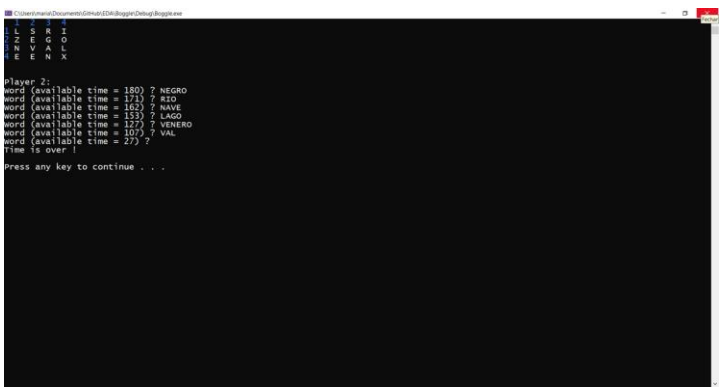
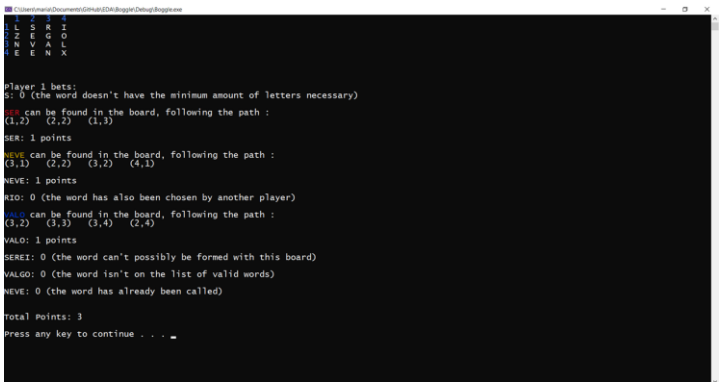


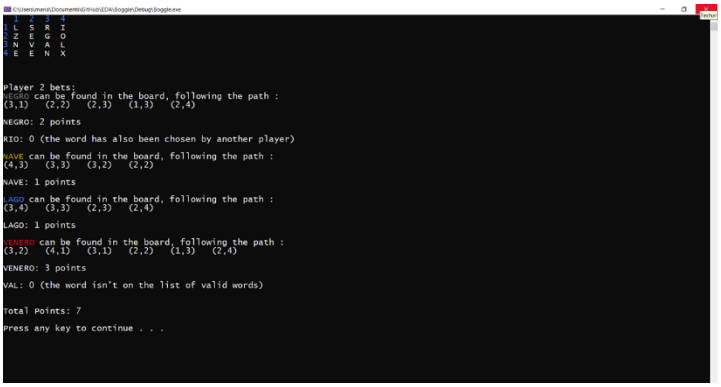
Figura 5. Escolha dos jogadores.





Figuras 6 e 7. Escolha das palavras relativas à primeira instância da *board*, tanto do primeiro como do segundo jogador.





Figuras 8 e 9. Pontuação, se obtida, relativamente a cada uma das palavras, tanto do primeiro como do segundo jogador. Podemos ver através destas duas imagens os diversos erros possíveis: “s” não tem o número mínimo de letras necessárias; “rio” foi escolhida por ambos os jogadores; não é possível formar a palavra “serei” com esta board; “valgo” não está na lista de palavras válidas, ou seja, no dicionário; “neve” foi escolhida duas vezes, por lapso, pelo primeiro jogador, pelo que apenas foi dada pontuação à primeira inserção.

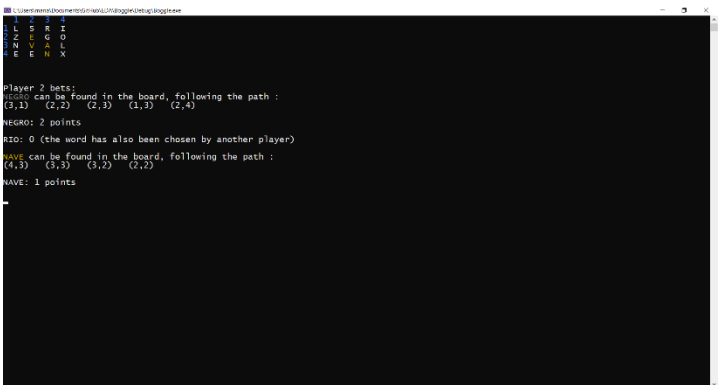


Figura 10. Pormenor de como as palavras são apresentadas quando válidas.

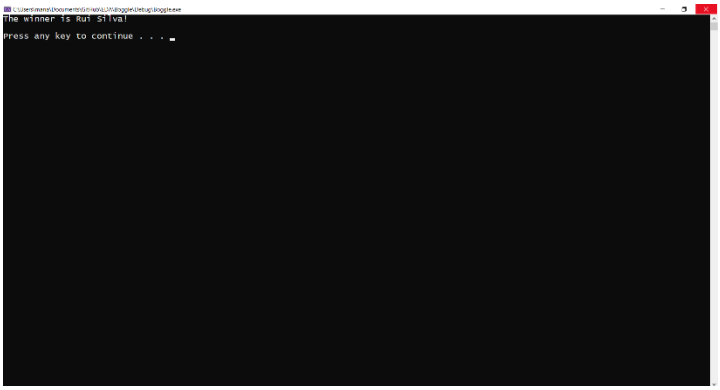


Figura 11. Apresentação do vencedor. O número de pontos tinha sido estabelecido como 5 de forma a tornar a execução mais rápida.

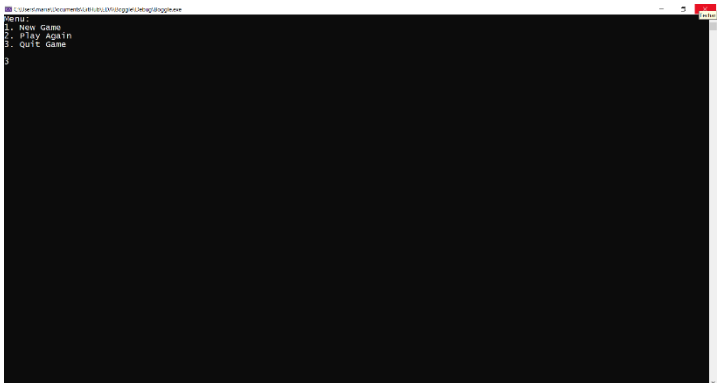
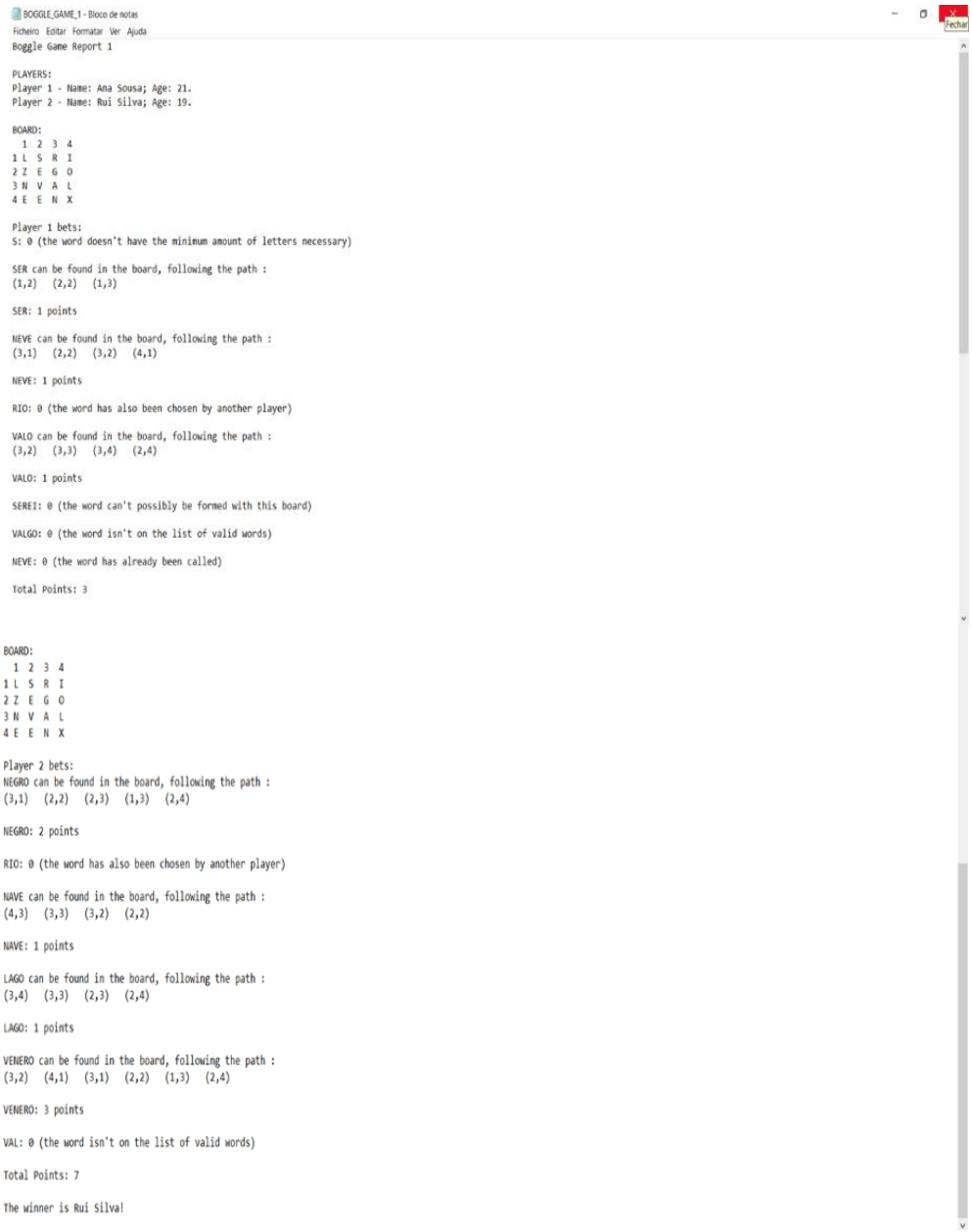


Figura 12. Segunda possibilidade de menu, escolhendo opção 3.

Figura 13. Janela de despedida do jogo.



BOGGLE_GAME_1 - Bloco de notas

Ficheiro Editar Formatar Ver Ajuda

Fechar

```
Boggle Game Report 1

PLAYERS:
Player 1 - Name: Ana Sousa; Age: 21.
Player 2 - Name: Rui Silva; Age: 19.

BOARD:
  1  2  3  4
1 L  S  R  I
2 Z  E  G  O
3 N  V  A  L
4 E  E  N  X

Player 1 bets:
S: 0 (the word doesn't have the minimum amount of letters necessary)

SER can be found in the board, following the path :
(1,2)   (2,2)   (1,3)

SER: 1 points

NEVE can be found in the board, following the path :
(3,1)   (2,2)   (3,2)   (4,1)

NEVE: 1 points

RIO: 0 (the word has also been chosen by another player)

VALO can be found in the board, following the path :
(3,2)   (3,3)   (3,4)   (2,4)

VALO: 1 points

SEREI: 0 (the word can't possibly be formed with this board)

VALGO: 0 (the word isn't on the list of valid words)

NEVE: 0 (the word has already been called)

Total Points: 3


BOARD:
  1  2  3  4
1 L  S  R  I
2 Z  E  G  O
3 N  V  A  L
4 E  E  N  X

Player 2 bets:
NEGRO can be found in the board, following the path :
(3,1)   (2,2)   (2,3)   (1,3)   (2,4)

NEGRO: 2 points

RIO: 0 (the word has also been chosen by another player)

NAVE can be found in the board, following the path :
(4,3)   (3,3)   (3,2)   (2,2)

NAVE: 1 points

LAGO can be found in the board, following the path :
(3,4)   (3,3)   (2,3)   (2,4)

LAGO: 1 points

VENERO can be found in the board, following the path :
(3,2)   (4,1)   (3,1)   (2,2)   (1,3)   (2,4)

VENERO: 3 points

VAL: 0 (the word isn't on the list of valid words)

Total Points: 7

The winner is Rui Silva!
```

Figuras 14 e 15. Game Report.