Programação II

Licenciatura em Engenharia Informática

Relatório do Trabalho Prático

Palavra Guru© Guru2P2

UNIVERSIDADE DE ÉVORA

Miguel Pombeiro, 57829 | Miguel Rocha, 58501

Departamento de Informática
Universidade de Évora
Janeiro 2025

Índice

1 Introdução

Este relatório descreve o desenvolvimento e implementação em Java do jogo ”Palavra Guru” tanto em ambiente de terminal de computador como numa interface gráfica.
O jogo ”Palavra Guru” é um jogo de palavras interativo individual em que o objetivo é formar palavras a partir de um conjunto de letras fornecidas, permitindo ao jogador desenvolver o seu vocabulário.
Este jogo é constituído por vários níveis sequenciais que o jogador deve vencer para progredir. Cada nível começa por dar ao jogador um número limitado de letras embaralhadas. Utilizando estas letras, o jogador deve formar palavras que preencham os espaços pré-definidos. Estes espaços representam os vários tamanhos das palavras a adivinhar que devem ser construídas apenas com as letras fornecidas, sem repetir. Ao encontrar uma palavra correta, esta será preenchida no tabuleiro de jogo, sendo que o nível é terminado quando o jogador completar todas as palavras.
O jogo inclui ainda um sistema de moedas, em que o jogador é recompensado com um determinado número de moedas por cada palavra acertada. Estas moedas podem depois ser utilizadas pelo jogador para pedir dicas, que revelam determinadas letras das palavras a adivinhar.
De forma a tornar o jogo mais dinâmico, todas as palavras inseridas pelo jogador que respeitem as regras, serão também verificadas num dicionário e, caso existam, irão também recompensar o jogador com moedas.

2 O Programa

Após uma cuidadosa análise das regras do jogo e de forma a melhor replicar o ”Palavra Guru” original, foram definidas algumas funcionalidades principais que a implementação teria de incluir:

- **Escolha de letras para gerar os níveis** em Português, que, tal como sugerido no enunciado, foi feita através de três métodos distintos - a partir dos níveis originais dos jogo ”Palavra Guru”, uma escolha aleatória de letras do alfabeto e através de uma escolha aleatória de uma palavra presente no dicionário.
- **Gestão dos níveis**, que acabam, também, por representar a pontuação e a progressão do jogador. No caso desta implementação, cada jogo terá, no máximo, quinze níveis.
- **Validação das palavras** inseridas pelo jogador - se são respeitadas as regras do jogo, mas também se as palavras são resposta aos níveis ou se existem no dicionário.
- **Interação com o jogador** onde é apresentado o estado do jogo e onde o jogador deve dar o seu *input*. No caso desta implementação, foram feitas duas formas de interação, uma a partir do terminal de computador e uma a partir de uma GUI (*Graphical User Interface*).
- **Manipulação de ficheiros externos**, que permitiria carregar os níveis predefinidos e o dicionário, bem como guardar e continuar um jogo anterior.

3 Classes Implementadas

Tirando partido dos princípios de encapsulamento e abstração da Programação Orientada a Objetos em Java, a implementação do jogo foi feita com recurso a várias classes. Estas classes representam os vários componentes do jogo.

De forma a melhor lidar com erros e situações inesperadas que possam ocorrer no normal decurso do jogo, são geradas exceções em todos métodos onde foi possível prever a ocorrência destes erros. Na sua maioria, são geradas exceções de *runtime* da classe *IllegalArgumentException* para lidar com situações em que o input do utilizador não corresponde ao esperado.

3.1 Word

A classe *Word* representa uma palavra no jogo. Isto é, cada palavra que se tem de adivinhar será uma instância desta classe.

Variáveis

Todas variáveis descritas são privadas e de instância.

- **word:** A palavra a adivinhar, do tipo *String*.
- **hiddenWord:** Representa a palavra a adivinhar oculta com *underscores* por cada letra, do tipo *StringBuilder*.
- **isGuessed:** Um identificador do tipo booleano que passará a ter o valor *true* se a palavra for adivinhada.

Principais Métodos

```
public String applyHint (int index)
```

Este método é responsável por aplicar uma dica à palavra desta classe, isto é, trocar o *underline* que representa a letra escondida na palavra oculta, para a letra correta da palavra a adivinhar.

```
public String getWord ()
```

Este método retorna a palavra se já foi adivinhada, e caso contrário, a palavra escondida.

3.2 Level

A classe ***Level*** representa um nível do jogo.

Variáveis

Todas estas variáveis são privadas e de instância.

- **levelNumber:** O número do nível.
- **letters:** Uma *String* que contém as letras deste nível sem espaços entre elas.
- **wordsToGuess:** Uma *ArrayList* de *Words*, que contém as palavras a ser adivinhadas neste nível.
- **dictGuessed:** Um *HashSet* de *Strings*, que contém todas as palavras que estão corretas, mas que não estão na lista de palavras a adivinhar. É de notar que também irá conter as palavras a adivinhar, caso o utilizador acerte essa palavra.
- **isCompleted:** Um identificador do tipo booleano que passará a ter o valor *true* se o nível estiver completo.

Principais Métodos

```
public boolean isValidInput (String input)
```

Este método recebe uma *String* que contém uma tentativa do utilizador. É responsável por verificar se essa tentativa cumpre todas as regras do jogo, e caso isso se verifique, retorna *true*. De forma a verificar se o jogador não utilizou mais letras do que aquelas que estão à sua disposição no nível, utilizou-se um *HashMap*, que contém o número de cada uma das letras do nível. Este valor será decrementado por cada letra utilizada pelo jogador sendo que nunca poderá ser inferior a zero.

```
public String getWordsToGuess ()
```

Este método retorna as palavras a adivinhar em formato de *String*, já formatadas para serem utilizadas no *output* para o jogador. As palavras a adivinhar são numeradas para ser mais fácil a escolha da palavra, ao pedir uma dica.

```
public String getHint (int wordNumber, int letterNumber)
```

É responsável por aplicar uma dica a um certo índice de uma palavra a adivinhar. É de notar que recebe o número da letra e da palavra como argumento, e converte em índices.

```
public boolean checkDictGuessed (String word)
```

Este método irá verificar se uma determinada palavra pertence ao *HashSet* das palavras do dicionário já adivinhadas. Caso não pertença, irá adicionar a palavra a esse *HashSet* e retornar o valor lógico *true*.

```
public boolean checkWord (String word)
```

Este método recebe uma *String* com a tentativa do utilizador, e verifica se essa palavra está por adivinhar. Para tal, percorre a *ArrayList* das palavras a adivinhar e caso se verifique que essa palavra não foi ainda adivinhada e pertence à *ArrayList*, passa a estar marcada como adivinhada. É, ainda, adicionada ao *HashSet* das palavras já adivinhadas.

```
public boolean isCompleted ()
```

Este método é responsável por verificar se um nível está completo. Para tal, percorre a *ArrayList* de palavras a adivinhar devolvendo *true* apenas se todas as palavras dessa lista já tiverem sido adivinhadas.

```
public String getWordsString()
```

Este método é responsável por transformar a *ArrayList* de palavras a adivinhar em formato de String. Se alguma palavra foi adivinhada, irá colocar um caracter extra "#" para mostrar que já foi adivinhada.

```
public String getDictGuessedString()
```

Para todas as palavras na HashSet de palavras adivinhadas do dicionário, este método coloca o caracter "%" numa *String* para ser utilizada no *output*.

3.3 Player

A classe ***Player*** representa um jogador no jogo. Este jogador apenas terá um modo de jogo associado, que é definido quando o jogo começa pela primeira vez, não podendo ser alterado uma vez escolhido.

Variáveis

Todas estas variáveis são privadas e de instância.

- **coins**: O número de moedas do jogador.
- **name**: O nome do jogador.
- **isLevel**: O modo de jogo associado ao jogador, com o valor *true* para o modo dos níveis já definidos e *false* para o modo dos níveis gerados pelo dicionário.

Principais Métodos

```
public void addCoins(int coins)
```

Este método adiciona ou retira moedas ao jogador.

3.4 Dictionary

A classe ***Dictionary*** representa um dicionário. Esta classe estende a classe *HashSet* parametrizada com *String*. Desta forma, contém todos os métodos implementados na classe *HashSet*

Variáveis

Todas estas variáveis são de classe e constantes.

- **MIN_WORD_LENGTH:** Representa o tamanho mínimo de uma palavra a adivinhar.
- **MAX_WORD_LENGTH:** Representa o tamanho máximo de uma palavra a adivinhar.
- **MIN_WORDS_LEVEL:** Representa o tamanho mínimo de palavras por nível.
- **MAX_WORDS_LEVEL:** Representa o tamanho máximo de palavras por nível.
- **PORTUGUESE_CONSONANTS:** É um *array* de caracteres que contém todas as consoantes portuguesas.
- **PORTUGUESE_VOWELS:** É um *array* de caracteres que contém todas as vogais portuguesas.
- **PORTUGUESE_ACCENTED_VOWELS:** É um *array* de caracteres que contém todas as vogais portuguesas acentuadas.

Principais Métodos

```
private String[] generateRandomLetters ()
```

Este método gera um *array* de tamanho aleatório (entre o comprimento mínimo e máximo da palavra) de letras que serão utilizadas para gerar palavras de forma a construir um novo nível. São utilizadas probabilidades diferentes de escolher uma consoante, vogal ou vogal acentuada.

```
private String[] generateRandomLetters_Dict ()
```

Este método tem como função escolher aleatoriamente uma palavra do dicionário entre o comprimento mínimo e máximo definido na classe. Para tal, é escolhido um índice aleatório entre zero e o tamanho do dicionário, e percorre-o até achar esse índice. Esta palavra é partida em letras para um *array* de *Strings*. De modo a garantir que as letras não fiquem pela mesma ordem da palavra original, estas são baralhadas.

```
private Level generateLevel (int currentLevel)
```

Gera um novo nível para o jogo com base em palavras verificadas no dicionário. O número de palavras a adivinhar no nível é aleatório e está entre os valores já definidos para mínimo e máximo de palavras por nível. Já o método utilizado para gerar as letras é aleatoriamente (com igual probabilidade) escolhido entre os dois já descritos acima. Depois de encontradas todas as palavras possíveis de gerar com aquelas letras, que estão no dicionário, um número aleatório destas será escolhido como palavras a adivinhar no nível.

```
private ArrayList<String> generateWordsToGuess (String[] letters,
    int wordLength)
```

Este método gera uma *ArrayList* de todas as palavras encontradas a partir das letras passadas como argumento. Para tal é chamado o método *generatePermutations*, descrito a seguir, que necessita de um *array* do tipo booleano para não repetir letras ao escolher as palavras.

```
private void generatePermutations (String[] letters, String current,
    int wordLength, boolean[] used, ArrayList<String> validWords)
```

Este é um método recursivo e responsável por gerar todas as permutações possíveis com as letras já encontradas para gerar um nível. Como são necessárias palavras de vários tamanhos para o jogo, este método terá de ser chamado várias vezes com os vários tamanhos desejados, uma vez que se trata de um método recursivo. Após gerar cada permutação, esta só será adicionada à *ArrayList* de resultados se respeitar o tamanho dado e se estiver contida no dicionário. De forma a garantir que não são utilizadas letras que já foram incluídas na palavra numa chamada recursiva anterior, é marcado o índice da letra utilizada no *array* de booleanos.

```
public ArrayList<Level> generateLevels (int nLevels,
    int currentLevel)
```

É responsável por gerar todos os níveis necessários para o bom funcionamento do jogo, chamando, para tal, o método *generateLevel*, já explicado. Como foi definido um limite de 15 níveis, este irá gerar tantos níveis quantos forem necessários para não ultrapassar este limite.

3.5 FileHandler

A classe ***FileHandler*** trata de todos os ficheiros externos necessários para o bom funcionamento do jogo. É de notar que os ficheiros associados a um determinado jogador serão guardados numa pasta com o seu nome, caso o utilizador decida guardar o jogo.

Variáveis

Todas estas variáveis são de classe, públicas e constantes.

- **DICTIONARY_NAME:** Contém o nome do ficheiro do dicionário.
- **LEVELS:** Contém o nome do ficheiro que contém os níveis já definidos.
- **GEN_LEVELS:** Contém o nome do ficheiro onde serão guardados os jogos gerados pelo dicionário.
- **LAST_LEVEL:** Contém o nome do ficheiro onde será guardado o último nível jogado.
- **SAVE_FILE:** Contém o nome do ficheiro onde serão guardadas todas as informações do jogador.

Principais Métodos

```
public static Dictionary readDictionary()
```

Este método é responsável por ler todas as palavras no ficheiro do dicionário e guardar no *HashSet* do dicionário. Para tal, irá ler o ficheiro linha a linha até chegar ao fim do ficheiro. Foram desconsideradas as palavras com hífen porque estas palavras não pertencem à versão original deste jogo.

```
public static ArrayList<Level> loadLevels(int startLevel)
```

É responsável por carregar níveis pré-definidos que se encontram no formato proposto dentro de um ficheiro. Foram ignorados todos os níveis anteriores que já foram jogados, de modo a carregar apenas os níveis necessários para continuar o jogo. Foi interpretado a formatação do ficheiro para garantir que primeiro são lido as letras e depois são lidas as palavras. Por fim, estes níveis serão guardados numa *ArrayList*.

`private static void savePlayerData (String dirName, Player player)`

Utilizado para guardar todas a informações do jogador num ficheiro de texto para que este possa retomar o jogo posteriormente. As informações do jogador guardadas são o tipo de jogo (Dicionário ou Pré-definido), o nome do jogador e o número de moedas que possui.

`private static void saveLastLevel (Level level, String dirPlayer)`

Este método irá criar um ficheiro que contém o número do nível e o estado desse nível, ou seja as letras, as palavras por adivinhar, palavras já adivinhadas e palavras adivinhadas que se encontravam no dicionário, com formatações distinguíveis.

`public static String saveGame (Player player, Level level)`

Este método contém toda a lógica necessária para guardar o estado do jogo, quando o jogador quer sair. O estado do jogo é guardado num novo diretório que terá o nome do jogador e que irá conter os dois ficheiros gerados com a chamada dos métodos *savePlayerData* e *saveLastLevel*.

`public static Player loadPlayerData(String dirName)`

Este método carrega o ficheiro com os dados do jogador, que se encontram num ficheiro de texto dentro do diretório com o nome deste jogador. Para tal, é lido e interpretado o ficheiro de acordo com a formatação, e retorna um objeto do tipo *Player* com os seus respetivos dados.

`public static Level loadLastLevel(String dirName)`

Este método lê um ficheiro de texto que contém toda a informação relativa ao último nível jogado pelo utilizador (que ficou por completar). O ficheiro será interpretado de acordo com a formatação que foi definida para guardar o jogo. Neste caso a formatação do texto será semelhante à dada no enunciado para os níveis pré-definidos. As palavras já adivinhadas foram marcadas com um "#". Já as palavras que não estão na lista de palavras a adivinhar mas foram inseridas pelo jogador e estão no dicionário, foram marcadas com um "%".

`public static void saveLevels(ArrayList<Level> level)`

É aqui onde são guardados para um ficheiro de texto todos os níveis que foram gerados pelo dicionário, com a formatação proposta no enunciado.

3.6 GameController

A classe ***GameController*** é responsável por controlar toda a dinâmica do jogo.

Variáveis de classe

Todas estas variáveis são públicas e constantes.

- **HINT_COST:** Custo que o jogador deve pagar ao pedir uma dica.
- **WORD_REWARD:** Recompensa em moedas que o jogador recebe ao adivinhar uma palavra.
- **MAX_LEVELS:** Número máximo de níveis por jogo.
- **START_COINS:** Número inicial de moedas com que o jogador começa o jogo.

Variáveis de instância

Todas estas variáveis são privadas.

- **player:** Um objeto do tipo *Player*.
- **dictionary:** Um objeto do tipo *Dictionary*.
- **levels:** Uma *ArrayList* de objetos do tipo *Level*.
- **levelIndex:** Um inteiro que contém o índice do nível atual.
- **view:** Uma instância da interface *View* que pode representar tanto a interface gráfica tanto a interface da consola.

Principais Métodos

```
public void startGame(int viewType)
```

Este é o método que dá inicio ao jogo em si, inicializando a *View* escolhida pelo jogador. Depois é chamado o método *play*, que será explanado mais adiante.

```
private void levelUp()
```

Deve ser chamada sempre que o jogador completar um nível. Contém a lógica necessária para que o jogador seja informado dessa situação e do incremento do índice para aceder à *ArrayList* que contém os níveis do jogo.

```
private void play()
```

Contém toda a lógica necessária à normal execução do ciclo de jogo. É neste método que são geradas e interpretadas todas as interações com o jogador bem como toda a atualização do estado do jogo. Contém um ciclo que irá correr enquanto o nível atual em que o jogador se encontra for inferior ao nível máximo ou até que o jogador queira sair e guardar o jogo. Caso o jogador queria sair e guardar o jogo, é necessário que o *input* corresponda à *String* "0" caso o jogador queira pedir uma dica, deverá ser "1". Se o *input* não for nenhuma desta opções é verificado se corresponde a uma *String* válida e caso o seja são efetuadas verificações se a palavra inserida pertence à lista de palavras a adivinhar ou ao dicionário.

```
private boolean saveGame()
```

É o método que pede uma confirmação de sair do jogo na interface atual. Caso confirme para sair, irá guardar o estado atual do jogo e do jogador no seu respetivo diretório.

```
private void askHint()
```

Este método pede uma confirmação de uma dica na interface atual. Caso o utilizador confirme, irá tentar deduzir o preço de uma dica ao seu saldo. Se o jogador tiver as moedas necessárias, será então tentado aplicar a dica ao índice da palavra escolhida.

3.7 *Interface*: View

De forma a tirar partido do polimorfismo em Java e simplificar a implementação das duas formas de interação com o jogador (Consola e GUI), foi feita a implementação de uma Interface.
A interface ***View*** representa ambas as "vistas" que o jogador pode ter do jogo e contém apenas a declaração dos métodos necessários à interação entre a classe que trata a lógica principal do jogo (*GameController*), e a interface escolhida pelo jogador para jogar.

3.7.1 ConsoleView

A classe ***ConsoleView*** é responsável por toda a interação com o utilizador a partir do terminal.

Variáveis

- **sc:** Variável de instância do tipo *Scanner*, privada.

Principais Métodos

```
public void printMessage (String message)
```

Este método é responsável por imprimir uma mensagem na consola.

```
public String askGuess ()
```

É responsável por interagir com o utilizador na consola para pedir um *input*, quer seja uma palavra, uma dica, ou a opção para sair.

```
public boolean askConfirmation (String action)
```

Este método pede uma confirmação ao utilizador, quer seja para sair ou para pedir uma dica. Caso confirme, retorna o valor lógico *true*.

```
public int[] askHintInfo ()
```

Este método pede dois valores ao utilizador, o número da palavra e o número da letra nessa palavra. Esses valores são depois retornados num *array* de inteiros, para ser tratado no seu respetivo método.

3.7.2 GraphicalView

A classe ***GraphicalView*** é responsável por toda a interação com o utilizador a partir da GUI.

Variáveis

- **frame:** Instância de *JFrame* onde estão inseridos todos os componentes que fazem parte do normal funcionamento do ciclo do jogo.
- **Componentes de *output*:** Várias instâncias de *JLabel* e de *JTextArea* onde são mostradas várias informações ao jogador que são necessárias para o normal decorrer do jogo.
- **Componentes de *input*:** Instâncias de *JButton* e de *JTextField* que irão ser utilizadas pelo utilizador para dar *input* ao programa.
- **input:** Variável do tipo *String* onde serão guardados os vários *inputs* dados quando utilizador pressionar botões ou inserir texto. É atualizado através dos *action listeners* definidos para cada componente.

Principais Métodos

```
private void initGameFields(String playerName, int levelNumber,
    int coins)
```

Inicializa todos os componentes da GUI necessários para o funcionamento do jogo. Todos os componentes são inseridos num *JPanel*, com um *layout Grid Bag*, que permite um melhor posicionamento de todos os componentes. Todos os restantes métodos que inicializam componentes da interface gráfica do jogo são chamados a partir deste, para serem posteriormente inseridos no painel de jogo.

```
private void initInputComponents ()
```

Este método inicializa todos os componentes de *input* da GUI. Todos os botões aqui inicializados são configurados com *action listeners* apropriados de forma a terem a resposta apropriada pela dinâmica de jogo, sendo que todos eles guardam valores na variável *input*. O botão de "Submeter" guarda nessa variável o valor lido do campo de *input* onde o utilizador escreve a nova tentativa, e os restantes botões guardam os valores necessários para que as ações que descrevem sejam executadas. De realçar que todos os *action listeners* têm de ser sincronizados com o método *askGuess*.

`private JPanel createCoinsPanel (int coins)`

Inicializa um *JPanel* que tem a representação do número de modas que o jogador tem. O painel contém uma ícone representativo do "mealheiro" bem como uma representação numérica do número de moedas que o jogador tem. Este número será atualizado a cada iteração do ciclo de jogo.

`private void initOutputFields (String playerName, int levelNumber)`

Este método inicializa os campos de *output* principais, nomeadamente a informação do número de nível atual, uma área de texto (*JTextArea*) que é composta pelas letras e as palavras a adivinhar dos níveis, e outra área de texto que terá informações de erros e avisos ao utilizador.

`private JPanel createNamePanel (String playerName)`

É aqui onde é criado o painel (*JPanel*) superior da GUI, que contém o nome do jogador e um ícone de perfil.

`private void initFrame ()`

É aqui onde é criada a janela principal do jogo, do tipo (*JFrame*), que irá conter todos os outros elementos necessários para o bom funcionamento do jogo.

`private void initWelcomePanel (String playerName)`

Este método inicia uma janela de diálogo, *JDialog*, para receber o utilizador antes de começar o jogo. Quando o utilizador quiser começar, irá ativar o *ActionListener* desse botão, que irá desencadear o encerramento desta janela, começando assim o jogo.

`public String askGuess ()`

Este método espera um *input* do utilizador tanto no campo de texto como através de outro botão. De notar que este método tem de ser sincronizado com os *action listeners* usados em cada botão, uma vez que apenas pode retornar quando o utilizador der o seu *input*.

`public boolean askConfirmation (String text)`

É responsável por pedir uma confirmação ao utilizador, quer seja ao confirmar a saída, quer seja pedir uma dica. Para tal, cria uma janela de diálogo com dois botões ("SIM" e "NÃO"). Retorna o valor lógico da confirmação.

```
public int[] askHintInfo ()
```

Este método é responsável por pedir a informação necessária para uma dica. Cria uma janela de diálogo, com dois campos de *input* (*JTextField*), um para pedir um número da palavra e outro o número da letra nessa palavra, até que o jogador submeta o pedido. Quando o utilizador submete o pedido, o *actionListener* do botão atribui os respetivos valores a um *array* de inteiros, que irá ser retornado.

```
public void close ()
```

Este é o método que fecha a janela principal que contém a GUI do jogo, depois de um tempo definido.

3.8 Main

Esta classe é o ponto de entrada para o jogo, lidando com todo o *input* inicial do utilizador que permite fazer a configuração inicial da sessão de jogo. Permite ao utilizador fazer várias escolhas:

- Começar um jogo novo ou carregar um jogo anterior.
- Escolher o nome do jogador.
- Jogar a partir dos níveis pré-definidos ou de níveis gerados a partir do dicionário.
- Tipo de interface onde jogar: consola ou GUI.

Caso o utilizador insira alguma opção errada nas configurações iniciais o programa é terminado.

É, também aqui que é inicializado o *GameController* com as definições escolhidas.

4 Discussão/Conclusão

A implementação de um jogo semelhante ao jogo "Palavra Guru" em Java permitiu consolidar os vários conceitos estudados ao longo do semestre em programação orientada a objetos. Foi ainda possível alargar e aplicar conhecimentos em diferentes estruturas de dados e na interação com o utilizador. Contudo, apesar da implementação das principais funcionalidades ter sido bem sucedida, há bastantes aspetos que podem ser melhorados de forma a tornar o programa mais eficiente e apelativo ao utilizador.
A lógica utilizada para a criação de novos níveis a partir do dicionário pode ser refinada. A implementação atual não garante que todas as letras que pertencem à lista de letras do nível são utilizadas. Uma forma de resolver este problema seria remover letras que depois não seriam utilizadas nas palavras a adivinhar. Da mesma forma, também não é garantida uma curva de dificuldade progressiva na geração dos níveis. Uma solução seria selecionar palavras com base no seu comprimento ou mesmo na sua frequência de utilização.
O tratamento de erros no programa poderia ser feito de uma forma mais elegante com a criação de exceções dedicadas ao jogo. Estas exceções personalizadas permitiriam capturar problemas específicos e apresentar mensagens mais claras ao utilizador.
Ao nível da interface gráfica (GUI), é notar que, embora esta cumpra os requisitos mínimos, há espaço para bastantes melhorias. Uma interface que mais se assemelhasse à do jogo original, em que o jogador não tem de fazer *inputs* de texto, iria permitir reduzir bastante os erros que podem vir a surgir proporcionando uma experiência mais fluída. Ao nível do aspeto, há também bastantes melhorias que podem ser feitas de forma a tornar a GUI mais apelativa e intuitiva. Seria ainda interessante, adicionar ao painel inicial do jogo um botão que abrisse uma janela com as regras de forma a ajudar os utilizadores a compreender melhor a dinâmica do jogo.

Com a realização deste projeto, foi possível aprimorar as competências ao nível de POO, reforçando a ideia de que um design modular e bem estruturado é crucial no desenvolvimento de programas.
Por fim, é de realçar que todas as partes do jogo aparentam estar funcionais, tendo sido alcançados todos os objetivos principais do trabalho prático que incluíam reproduzir o jogo "Palavra Guru".