Escola de Engenharia – Departamento de Informática da Universidade do Minho

Mestrado em Bioinformática

Introdução aos Algoritmos e à Programação 2021/2022

Desenvolvimento de uma aplicação em Python para a resolução do puzzle *Illuminati*

Grupo 11

Alexandre Miguel Magalhães Esperança (pg45963)

António Nuno Carrilho Canatário Duarte (pg45464)

Cristiana Martins Albuquerque (pg45468)

Mónica Rafaela Machado Leiras (pg45473)

Conteúdos

1. Introdução e objetivos 1

2. Descrição e estrutura geral 1

3. Módulo *IlluminatiWindow* 1

4. Módulo *IlluminatiEngine* 1

4.1. Métodos *ler_tabuleiro_ficheiro* e *gravar_tabuleiro_ficheiro* 1

4.2. Métodos *add_historico* 2

4.3. Métodos *undo* e *returnStart* 2

4.4. Métodos *jg* e *mc* 2

4.5. Métodos para Estratégia 1 2

4.6. Métodos para Estratégia 2 2

4.7. Métodos para Estratégia 3 2

4.8. Métodos para Estratégia 4 3

4.9. Métodos para Estratégia 5 3

4.10. Métodos para Estratégia 6 3

4.11. Métodos para Estratégia 7 3

4.12. Método *resolve* 3

4.13. Método *verify_lights* e *check_numbers* 4

5. Módulo *IlluminatiShell* 4

5.1. Método *do_mr* 4

5.2. Método *do_cr* e *do_gr* 4

5.3. Método *do_jg* 4

5.4. Método *do_mc* 4

5.5. Método *do_est1*, *do_est2*, *do_est3*, *do_est4*, *do_est5*, *do_est6* e do_est7 4

5.6. Método *do_undo* 5

5.7. Método *do_rtstart* 5

5.8. Método *do_resolve* 5

5.9. Método *verify_victory* 5

5.10. Método *do_ver* 5

5.11. Método *do_sair* 5

1. Introdução e objetivos

Illuminati é um puzzle que é jogado numa área quadriculada, sendo que cada casa desta área pode ser de diferentes tipos: casa vazia, casa com lâmpada (que ilumina a linha e a coluna em que está colocada) e casa bloqueada (que indica um obstáculo que impede a passagem da luz). De salientar que, quando uma casa está bloqueada, esta pode conter um número, que obriga a que existam esse mesmo número de lâmpadas nas casas livres adjacentes. Diz-se que uma casa está iluminada quando tiver uma lâmpada ou se existir uma lâmpada na mesma linha ou coluna, sem que haja uma casa bloqueada entre a casa e a lâmpada. Para além disso, não podem existir duas lâmpadas na mesma linha ou coluna, a menos que exista uma casa bloqueada entre elas. Deste modo, o objetivo do puzzle passa por iluminar todas as casas desbloqueadas.

Este projeto tem como objetivo o desenvolvimento de uma aplicação, em *Python*, para a resolução do puzzle *"Illuminati".*

2. Descrição e estrutura geral

O desenvolvimento da aplicação para este trabalho utiliza a arquitetura MVC (*Model-View-Controller*) e apresenta a seguinte estrutura: Módulo ***IlluminatiEngine***, Módulo ***IlluminatWindow*** e Módulo ***IlluminatiShell***.

A descrição detalhada de cada módulo é referenciada nas secções seguintes. Os módulos ***IlluminatiWindow*** (ficheiro *IlluminatiWindow.py*) e ***graphics*** (ficheiro *graphics.py*) foram fornecidos pelo docente.

3. Módulo *IlluminatiWindow*

O módulo ***IlluminatiWindow*** constitui uma classe que cria uma janela para visualização gráfica de um tabuleiro a ser usado para o puzzle. Este tabuleiro é constituído por casas, previamente descritas. Esta classe tem como parâmetros o tamanho da casa no ecrã, em pixéis (*cell_size*) e o nome ficheiro a ler (*filename*). Neste módulo são implementadas funções como a criação dos vários tipos de casas (vazias, iluminadas e/ou bloqueadas) e lâmpadas.

4. Módulo *IlluminatiEngine*

O módulo ***IlluminatiEngine*** (ficheiro *IlluminatiEngine.py*) constitui uma classe que permite a implementação das regras do puzzle, posteriormente executadas pela classe *IlluminatiShell*.

4.1. Métodos *ler_tabuleiro_ficheiro* e *gravar_tabuleiro_ficheiro*

Os métodos *ler_tabuleiro_ficheiro* e *gravar_tabuleiro_ficheiro* recebem como parâmetro o nome de um ficheiro com o puzzle e permitem o carregamento e gravação de um ficheiro com o puzzle resolvido, respetivamente.

4.2. Métodos *add_historico*

O método *add_historico* permite adicionar à *stack* (ficheiro *stack.py*) do histórico as alterações feitas ao ficheiro.

4.3. Métodos *undo* e *returnStart*

O método *undo* permite desfazer o efeito do comando anterior, enquanto que o método *returnStart* permite desfazer o efeito de todos os comandos utilizados até ao momento. Ambos recorrem ao histórico de jogadas.

4.4. Métodos *jg* e *mc*

Os métodos *jg* e *mc* recebem como parâmetros os números correspondentes à linha e coluna de uma casa. Ambos os métodos permitem determinar, inicialmente, se a casa está ou não bloqueada. Se esta condição não se verificar, pode ser adicionada/retirada uma lâmpada (método *jg*) ou marcar uma casa como bloqueada (método *mc*).

Adicionalmente, o método *jg* invoca os métodos *IlluminaVert* e *IlluminaHoriz*, que recebem como parâmetros o número da linha e coluna correspondente à casa que foi adicionada a lâmpada. Estes métodos percorrem toda a linha e coluna, respetivamente, em ambos os sentidos, e substituem todas as casas vazias por iluminadas. Esta é uma implementação automática da estratégia 2 quando se adiciona uma lâmpada.

4.5. Métodos para Estratégia 1

Para a estratégia 1, se uma casa com um determinado número tem exatamente o mesmo número de casas vizinhas onde possam existir lâmpadas, então todas essas casas têm que conter lâmpadas.

Os métodos *casas_vizinhas_disp* e *num_lampadas_vizinhas* permitem verificar se as casas adjacentes a uma determinada casa estão disponíveis e, se existirem, contar o número de lâmpadas. Os métodos *estrategia1_cada_casa* aplica os dois métodos anteriores numa determinada casa do tabuleiro, funcionalidade que a *estrategia1* repete para todas as casas do tabuleiro.

4.6. Métodos para Estratégia 2

Para a estratégia 2, se uma casa contém uma lâmpada pode-se marcar como iluminadas todas as casas na mesma linha e coluna, desde que não sejam casas bloqueadas.

Esta estratégia é implementada sempre que existe uma jogada pelo método *jg* e adicionada uma lâmpada.

4.7. Métodos para Estratégia 3

Para a estratégia 3, podem-se colocar marcas numa casa quando se tem a certeza que esta casa não pode conter uma lâmpada, sendo que isto pode acontecer nas casas vizinhas de um 0 ou em casas onde colocar uma lâmpada criaria uma contradição.

O método *estrategia3* percorre todas as casas do tabuleiro e invoca métodos auxiliares de acordo com o número da casa que encontra. Caso essa casa já tenha o número de lâmpadas igual ao seu prórpio número, marca todas as outras casas vizinhas adjacentes.

Em alternativa, através dos métodos *casas_diagonais_imp* e *casas_vizinhas_bloq*, pode-se bloquear as diagonais adjacentes da casa com número 3 uma vez que, independentemente da colocação das lâmpadas, todas as diagonais ficarão bloqueadas pela luz. Para casas com número 2, caso tenham três casas vizinhas disponíveis, pode-se também bloquear as casas diagonais adjacentes pela mesma razão.

4.8. Métodos para Estratégia 4

Para a estratégia 4, nos casos em que haja dois números numa diagonal, as duas casas partilhadas não podem ser as duas iluminadas ao mesmo tempo. Logo, as outras duas casas têm que conter lâmpadas.

O método *_est4* recebe como parâmetro o número da linha e coluna correspondente a uma casa e serve de função auxiliar para o método *estrategia4* que, juntamente com a invocação dos métodos *IlluminaVert* e *IlluminaHoriz,* permite a aplicação da estratégia 4.

4.9. Métodos para Estratégia 5

Para a estratégia 5, verificou-se se uma casa não iluminada só pode ser iluminada se uma dada casa tiver uma lâmpada (devido a outras restrições). Os métodos *casas_vazias_coluna* e *casas_vazias_linha* permitem determinar o número de casas vazias na mesma coluna ou linha, respetivamente, de uma casa até encontrar uma casa bloqueada. Estes métodos também servem de suporte às estratégias 6 e 7. O método *estrategia5* permite a aplicação da estratégia a todas as casas do tabuleiro.

4.10. Métodos para Estratégia 6

Caso as estratégias 1 a 5 não resolvam totalmente o puzzle, é invocado o método *estrategia6*. Aqui, aplica-se uma estratégia de tentativa-erro para casas que tenham apenas duas opções disponíveis para iluminação.

4.11. Métodos para Estratégia 7

Caso as estratégias 1 a 6 não resolvam totalmente o puzzle, é invocado o método *estrategia7*. Aqui, aplica-se uma estratégia de tentativa-erro para casas que tenham 3 ou mais opções disponíveis.

4.12. Método *resolve*

O método *resolve* permite a implementação da resolução automática do puzzle. Para tal, inicialmente, é criada uma cópia do tabuleiro, seguindo-se a verificação da resolução do puzzle. São invocadas e aplicadas todas as estratégias de resolução descritas, em loop, até o puzzle ficar totalmente resolvido ou não apresentar diferenças em relação ao ciclo anterior.

4.13. Método *verify_lights* e *check_numbers*

O método *verify_lights* permite verificar se todas as casas desbloqueadas estão iluminadas, sendo o objetivo fulcral do puzzle. Sabendo que quando uma casa está bloqueada, esta pode conter um número, que obriga a que existam esse mesmo número de lâmpadas nas casas livres adjacentes, o método *check_numbers* permite verificar se as casas com um determinado número associado têm o mesmo número de lâmpadas adjacentes.

5. Módulo *IlluminatiShell*

O módulo ***IlluminatiShell*** (ficheiro *IlluminatiShell.py*) constitui uma classe para um interpretador de comandos (shell). Neste módulo são implementadas funcionalidades como a leitura de um ficheiro com o puzzle a ser resolvido, 7 estratégias, uma solução automática do puzzle e a gravação do seu estado atual. Todos os métodos e respetivos parâmetros são descritos nas secções seguintes.

5.1. Método *do_mr*

O método *do_mr* permite mostrar o puzzle existente no ficheiro. Este recebe como parâmetro o nome de um ficheiro com o puzzle e invoca o método *ler_tabuleiro_ficheiro* da classe *IlluminatiEngine*.

5.2. Método *do_cr* e *do_gr*

Os métodos *do_cr* e do_gr recebem como parâmetro o nome de um ficheiro e permitem carregar e gravar, respetivamente, o puzzle e invocam o método *gravar_tabuleiro_ficheiro* da classe *IlluminatiEngine*.

5.3. Método *do_jg*

O método *do_jg* permite colocar ou retirar uma lâmpada, caso a casa já esteja preenchida com uma. Este método recebe como parâmetros dois números inteiros que correspondem à linha e coluna de uma casa e invoca o método *jg* da classe *IlluminatiEngine*.

5.4. Método *do_mc*

O método *do_mc* permite colocar ou retirar uma marca, caso a casa já esteja preenchida com uma. Tal como o método *do_jg* recebe como parâmetros dois números inteiros correspondentes à linha e coluna de uma casa. Invoca o método *mc* da classe *IlluminatiEngine*.

5.5. Método *do_est1*, *do_est2*, *do_est3*, *do_est4*, *do_est5*, *do_est6* e *do_est7*

Os métodos *do_est1*, *do_est2*, *do_est3*, *do_est4*, *do_est5*, *do_est6* e *do_est7* invocam os métodos *estrategia1*, *estrategia2*, *estrategia3*, *estrategia4*, *estrategia5*, *estrategia6* e *estrategia7*, respetivamente, da classe *IlluminatiEngine*. Estes têm como objetivo implementar as estratégias 1 a 7 para a resolução do puzzle, previamente descritas no módulo *IlluminatiEngine*.

5.6. Método *do_undo*

O método *do_undo* permite desfazer os efeitos dos comandos anteriores, invocando o método *undo* da classe *IlluminatiEngine*.

5.7. Método *do_rtstart*

O método *do_rtstart* permite desfazer os efeitos de todos os comandos anteriores (o puzzle volta ao estado inicial). Invoca o método *returnStart* da classe *IlluminatiEngine*.

5.8. Método *do_resolve*

O método *do_resolve* tem como objetivo a resolução automática do puzzle, invocando os métodos *resolve* e *add_historico* da classe *IlluminatiEngine*.

5.9. Método *verify_victory*

O método *verify_victory* permite verificar se o puzzle foi completado corretamente, isto é se todas as casas desbloqueadas estão iluminadas, seguindo as regras do puzzle. Invoca os métodos *verify_lights* e *check_numbers* da classe *IlluminatiEngine*.

5.10. Método *do_ver*

O método *do_ver* permite visualizar o estado atual do puzzle em ambiente gráfico. Para tal, invoca a classe *IlluminatiWindow*.

5.11. Método *do_sair*

O método *do_sair* permite o encerramento do programa e respetiva saída do puzzle.

6. Conclusão

Um dos objetivos para o desenvolvimento deste trabalho foi a criação de uma estrutura em módulos de forma a organizar de forma mais eficiente a informação de cada aplicação que pudesse responder às regras do puzzle. Para tal, foram criadas 3 estruturas: Módulo ***IlluminatiWindow*** (fornecido pelo docente), Módulo ***IlluminatiEngine*** e Módulo ***IlluminatiShell***.

Para além disso, para este puzzle foram desenvolvidas 7 estratégias de resolução, que vão de encontro às regras deste, e todas as funcionalidades mínimas requeridas foram cumpridas. De forma a verificar estes critérios, se o código executa e resolve corretamente os puzzles, foi criado um ficheiro *unit testing* (ficheiro *IlluminatiVerify.py*).

Para concluir, e tendo em conta o descrito anteriormente, foi possível atingir os objetivos propostos deste projeto, permitindo um enriquecimento de conhecimentos.