Pesquisa Aplicada ao Problema de Alocação de Lotes de Terreno

Relatório Final

3º Ano do Mestrado Integrado em Engenharia Informática e Computação

Inteligência Artificial

Elementos do Grupo:

Duarte Nuno Pereira Duarte – 201109179 – ei11101@fe.up.pt

Luís Filipe Correia Cleto – 201104279 – ei11077@fe.up.pt

26 de Maio de 2014

Conteúdo

1. Objetivo

O trabalho a ser realizado tem como propósito a implementação de uma ferramenta que permita resolver problemas de alocação de lotes de terreno usando métodos de pesquisa. O problema consiste em atribuir usos a vários lotes de terreno com características e custos próprios. Para atribuir um uso a um lote, este deve satisfazer os requisitos de que esse uso necessita. As restrições impostas pelos usos podem ser de dois tipos: restrições fortes, que têm de ser cumpridas obrigatoriamente, e restrições fracas, que devem ser satisfeitas tanto quanto possível.

A ferramenta a desenvolver deve permitir a especificação do número de lotes disponíveis e as suas características, as utilizações a atribuir aos lotes e as restrições existentes. Uma vez fornecidos os dados do problema, deve encontrar uma solução usando métodos de pesquisa. Uma solução é tanto melhor quanto mais baixo for o custo associado e quanto mais se adequar às restrições estabelecidas.

Os métodos de pesquisa utilizados podem ser informados, como A* e algoritmos gananciosos, ou fracos, como pesquisa em largura ou pesquisa em profundidade.

2. Especificação

2.1. Análise

Com o projeto desenvolvido, foi criada uma ferramenta que permite encontrar soluções para problemas de alocação de lotes de terreno usando técnicas de pesquisa. Este tipo de problema consiste em atribuir utilizações a lotes de terreno, respeitando as restrições impostas pelas utilizações em questão. Cada lote de terreno tem um custo associado bem como um conjunto de características, como a inclinação, a qualidade do solo, a proximidade a certos elementos como um lago ou uma autoestrada, a dimensão do lote, entre outras. As utilizações podem ser de vários tipos como habitação, escola, aeroporto e vários outros. Terão associadas restrições, como ter inclinação baixa ou estar próxima de um lago, que irão condicionar os lotes que serão adequados para aquele tipo de uso. Estas restrições podem ser obrigatórias de satisfazer (restrições fortes) ou apenas características desejáveis para obter uma melhor solução (restrições fracas). Cada utilização deve ser atribuída a um lote que satisfaça todas as suas restrições fortes e tantas restrições fracas quanto possível.

A nível de complexidade, o número de soluções possíveis para um problema deste tipo é dado pela equação $\frac{L!}{(L-U)!}$ (L arranjos a U), onde L representa o número de lotes disponíveis e U representa o número de utilizações a alocar. Contudo, a existência de restrições fortes pode eliminar parte das soluções, permitindo podar a árvore de pesquisa.

Adicionalmente, a ferramenta desenvolvida possui uma interface gráfica que permite a um utilizador especificar o número de lotes do problema e as suas características, bem como as utilizações que pretende atribuir e as restrições associadas. Posteriormente, permite que o utilizador escolha o algoritmo de pesquisa a usar para encontrar uma solução para o problema. A qualidade da solução é tanto maior quanto menor for o custo associado e maior for a adequação dos lotes de terrenos às utilizações a eles atribuídas. A adequação de um lote a uma utilização irá depender da satisfação das restrições dessa utilização por parte do lote.

2.2. Abordagem

De modo a resolver o problema proposto, foram primeiro criadas estruturas para representar os diferentes elementos do problema: *Lot*, *Landuse*, *HardConstraint*, *SoftConstraint, LanduseAllocations* (representa um elemento da árvore de pesquisa) e *Problem* (contém todos os dados relativos a um problema). Adicionalmente, para cada problema é computada uma tabela de restrições com entradas para cada par de lotes e utilizações, tanto para as restrições fortes, como para as fracas. No caso das fortes, indica apenas se um lote satisfaz todas as restrições de uma utilização, no caso das fracas, indica o acréscimo de custo associado a atribuir aquela utilização a esse lote (será 0 se o lote obedecer a todas as restrições fracas da utilização, caso contrário irá variar conforme a restrição). É ainda relevante mencionar que para as restrições fracas de tamanho dos lotes e de distância, o custo acrescido por um lote não cumprir com a restrição é calculado com base num grau de cumprimento da restrição, ou seja, se cumprir a restrição, tem custo acrescido 0, caso contrário, o custo acrescido será tanto maior quanto mais distante o lote estiver de cumprir com a restrição estabelecida. Por este motivo, algoritmos que tenham em conta a qualidade da solução obtida, caso não seja possível satisfazer a restrição fraca, irão procurar uma solução que esteja o mais perto possível de o fazer. É ainda relevante mencionar que se existirem mais utilizações que lotes, o problema não tem solução. Esta verificação é realizada antes de iniciar uma pesquisa por uma solução.

Posteriormente, foram implementados diversos métodos de pesquisa, tanto fracos (pesquisa em largura, pesquisa em profundidade, custo uniforme) como informados (A* e

algoritmo ganancioso). Adicionalmente, foi implementado um algoritmo *BruteForce*, baseado em pesquisa em largura, que encontra todas as soluções e escolhe a melhor, de modo a validar os resultados obtidos pelos restantes algoritmos. Visto que as soluções do problema se encontram todas no mesmo nível de profundidade da árvore (igual ao número de utilizações a atribuir), a complexidade deste algoritmo é semelhante à pesquisa em largura. Para realizar a pesquisa, é usado como estado inicial um objeto do tipo Alocações, em que não foram ainda atribuídas utilizações a lotes. Para obter um sucessor de um estado, é realizada uma atribuição de uma utilização a um lote que satisfaça as suas restrições fortes. Por este motivo, cada estado tem no máximo LL*UNA sucessores, onde LL representa o número de lotes livres e UNA o número utilizações não atribuídas nesse estado. Para todos os algoritmos implementados é evitado o processamento de nós repetidos.

Como alguns dos algoritmos usados exigem o uso de uma função heurística, foi criada uma função para estimar o custo heurístico de cada estado. Este custo é calculado como o custo dos N lotes mais baratos ainda livres, sendo N o número de utilizações por alocar. No entanto, os lotes que não satisfaçam as restrições fortes para nenhuma das utilizações por alocar, são considerados como tendo custo infinito. Após este cálculo, para cada utilização por alocar, é identificado o menor custo acrescido (da tabela de restrições fracas), para os lotes que ainda estão livres, e adicionado ao custo heurístico. Esta função heurística é admissível pois o custo estimado será sempre igual ou inferior ao custo real, garantindo assim que o A* irá encontrar sempre a melhor solução, facto apoiado pelos resultados obtidos com os testes realizados ao programa.

Numa fase final de desenvolvimento foram realizadas várias otimizações ao código, nomeadamente a nível de estruturas internas que permitissem um processamento mais rápido para certas operações como obter o melhor estado seguinte da lista de estados nos algoritmos do tipo Melhor Primeiro.

3. Desenvolvimento

3.1. Ferramentas e Ambiente de Desenvolvimento

O projeto foi desenvolvido no sistema operativo *Microsoft Windows*, versões 7 e 8. Foi implementado na linguagem de programação C#, usando a IDE *Microsoft Visual Studio Ultimate 2013*.

3.2. Estrutura

O programa desenvolvido foi dividido em dois módulos principais, um para representar e resolver os problemas, e outro para a interface gráfica. O módulo dos problemas foi subdividido em componentes de representação do problema e de pesquisa por solução. Cada uma destas componentes contém, respetivamente, classes para cada elemento de representação e classes para os métodos de pesquisa. As representações exatas das estruturas criadas podem ser observadas nos seguintes diagramas de classes.

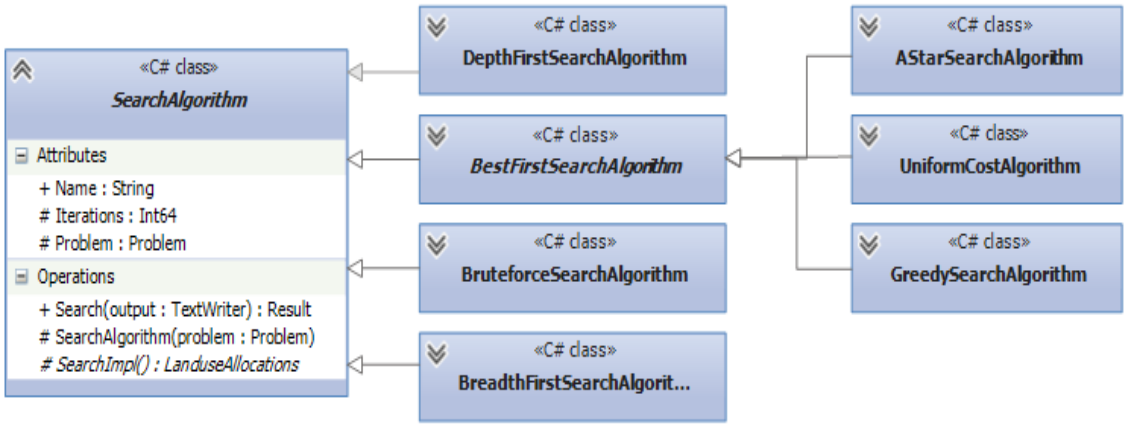

Figura 1. – Diagrama de classes para os algoritmos de pesquisa.

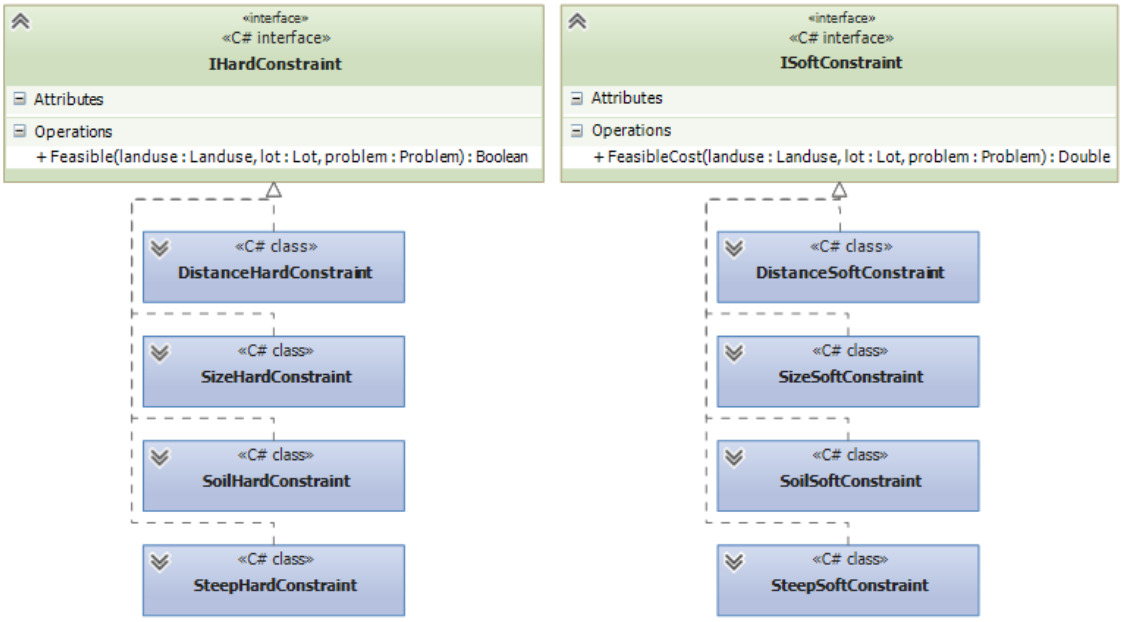

Figura 2. – Diagrama de classes para as restrições.

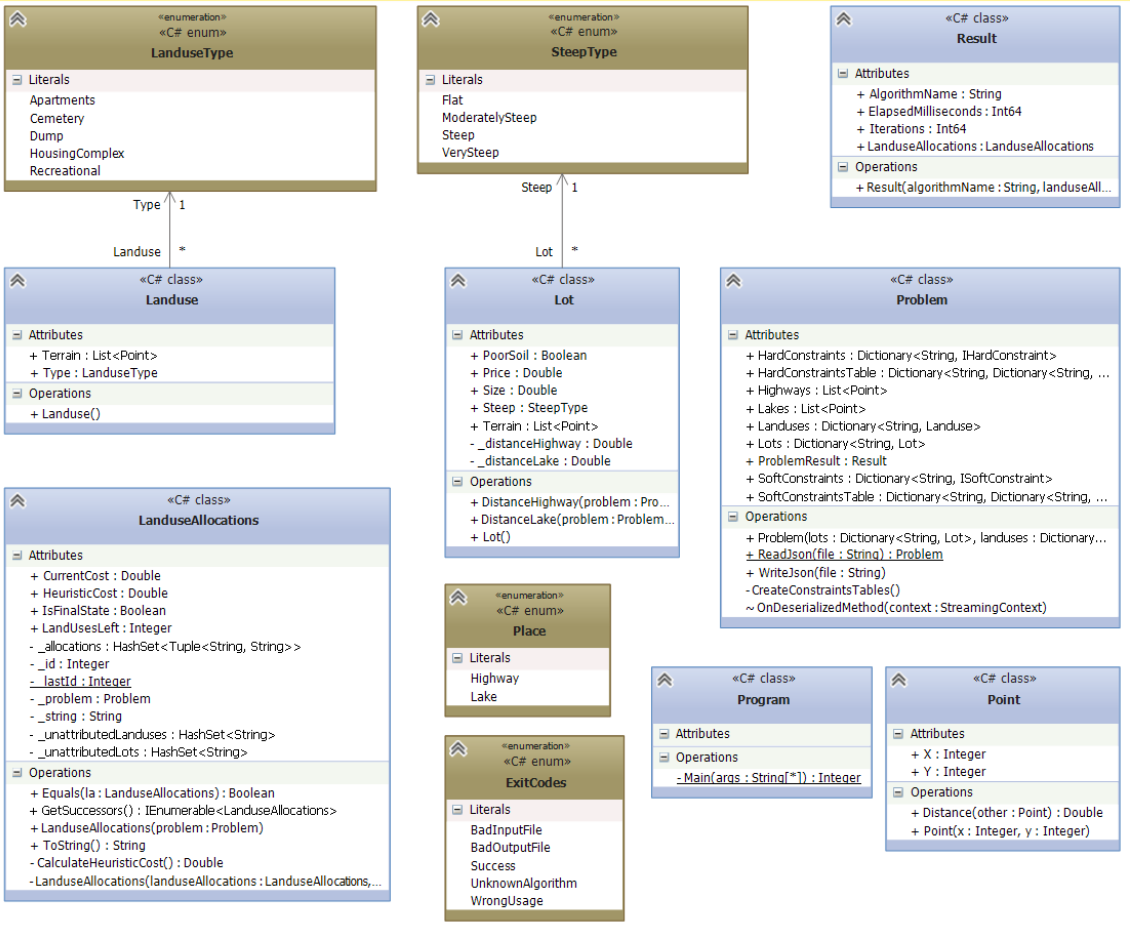

Figura 3. – Diagrama de classes para a representação de um problema.

3.3. Detalhes Relevantes

Outros detalhes importantes sobre a implementação do projeto são o facto de permitir gravar os problemas criados em ficheiros JSON, bem como ler estes ficheiros para carregar a informação dos problemas. Permite também gravar as suas soluções. Adicionalmente, vale a pena mencionar que o módulo de resolução dos problemas é completamente independente da interface gráfica e pode ser corrido isoladamente para obter soluções para problemas e até estatísticas (tempo de execução e número de iterações dos algoritmos). Por fim, pode-se mencionar ainda que foram feitas várias otimizações ao nível de estruturas internas, como usar estruturas ordenadas para as listas de estados por expandir dos algoritmos do tipo Melhor Primeiro, e ainda ao nível de algoritmos recursivos que foram convertidos em iterativos de modo a diminuir a sua complexidade espacial.

4. Experiências

De modo a validar o desempenho do programa foram realizadas várias experiências, com problemas de dimensões diferentes, e medidos os resultados com o objetivo de comparar tanto a qualidade das soluções obtidas para cada algoritmo como os recursos necessários para calcular uma solução.

Apresentam-se abaixo as comparações dos recursos consumidos por cada algoritmo para os problemas analisados sob a forma de gráficos, feitos em função do número de estados na árvore de pesquisa, incluindo soluções.

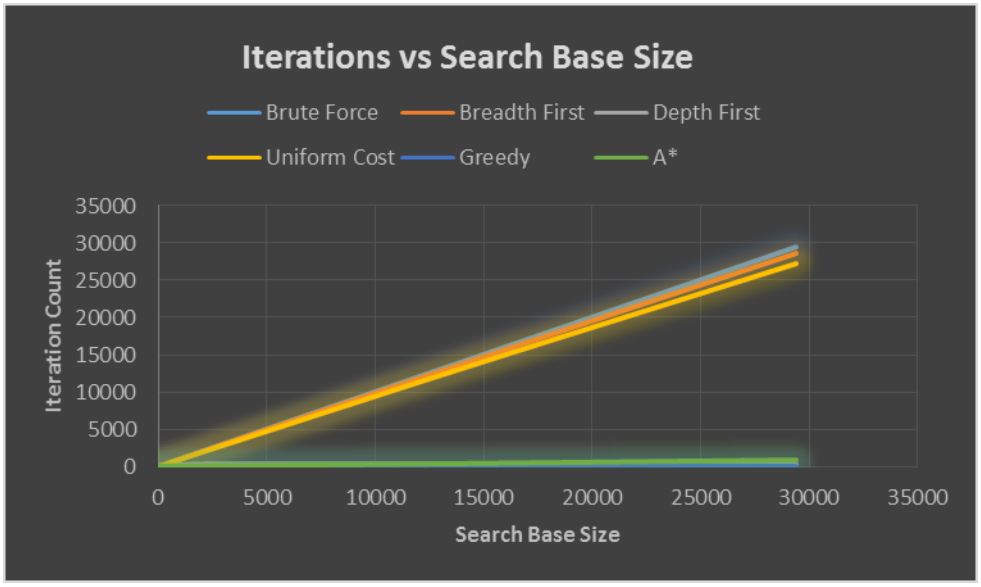

Figura 4. – Número de iterações em função do número de elementos da árvore de pesquisa.

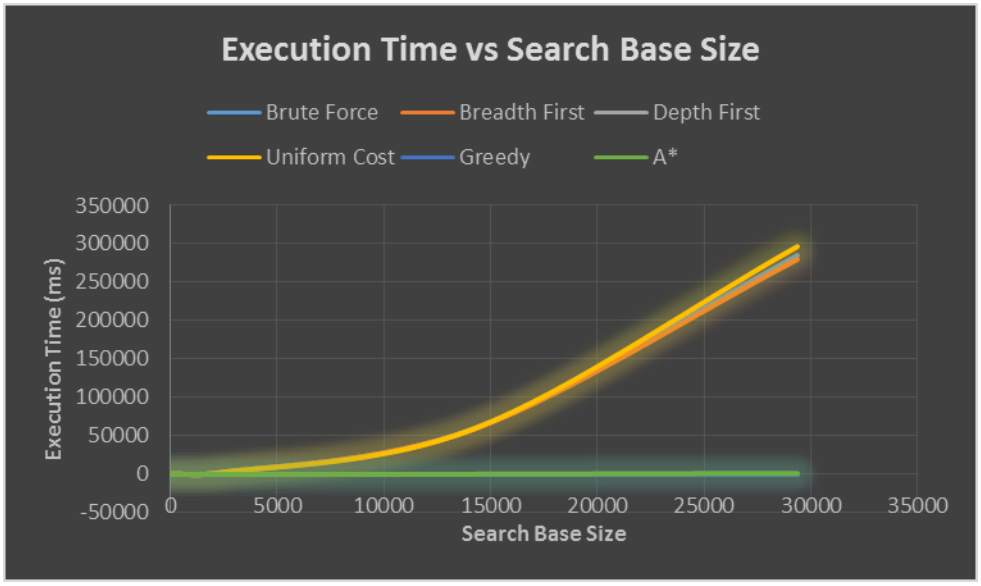

Figura 5. – Tempo de execução em função do número de elementos da árvore de pesquisa. *(Processador Intel® Core™ i7-2630QM CPU @2.00GHz 2.00GHz)*

5. Conclusões

Analisando os resultados obtidos, verificou-se que os algoritmos A* e de Custo Uniforme obtêm sempre soluções ótimas, com custo idêntico às obtidas pelo algoritmo *BruteForce* que calcula todas as soluções e retorna a melhor. Por outro lado, os algoritmos de Pesquisa em Largura, Pesquisa em Profundidade e Ganancioso podem não encontrar a melhor solução. No entanto, todos os algoritmos implementados são completos para este problema, encontrando sempre uma solução caso exista ou indicando que falhou caso não haja solução. Isto deve-se, em parte, ao facto de que o espaço de pesquisa para um problema de alocações de terreno pode ser descrito por uma árvore, sem ciclos e com um fator de ramificação finito. Adicionalmente, confirmou-se que a dimensão do espaço de pesquisa do problema aumenta de forma combinatória com o número de lotes e utilizações. No entanto, adicionando um número elevado de restrições fortes, o espaço de pesquisa pode diminuir significativamente. Pode-se ainda verificar através da comparação de dados estatísticos de cada algoritmo, como o tempo de execução e o número de iterações necessárias, que o algoritmo mais veloz e que realiza menos iterações é o Ganancioso, seguido da Pesquisa em Profundidade e do A*. Imediatamente a seguir estão os algoritmos de Custo Uniforme, Pesquisa em Largura e *BruteForce*, por esta ordem.

Os resultados obtidos estão de acordo com o esperado, sendo a diferença do tempo necessário para obter uma solução ótima através do algoritmo A* e através dos restantes algoritmos ótimos muito significativa, uma vez que o custo do A* aumenta quase linearmente enquanto que o custo para os restantes algoritmos varia exponencialmente. Isto indica que a heurística usada é poderosa. Adicionalmente, verifica-se que o número de iterações para os algoritmos *BruteForce*, Pesquisa em Largura e Custo Uniforme aumenta linearmente com o tamanho do espaço de pesquisa, enquanto que os restantes aumentam de forma aproximadamente linear apenas com o número de utilizações a atribuir aos lotes. Isto deve-se ao facto de que as soluções se encontram todas no mesmo nível de profundidade da árvore de pesquisa, correspondendo ao último nível da mesma e ao número de utilizações a atribuir. Pode-se também verificar que o algoritmo Ganancioso encontra soluções razoáveis muito rapidamente, podendo ser usado em vez do A* para problemas de dimensão muito elevada. Para soluções rápidas, pode-se também usar o algoritmo de Pesquisa em Profundidade. No entanto, este último não é tão rápido como o Ganancioso e a qualidade das soluções que encontra varia drasticamente de problema para problema, já que este não tem em conta o seu custo

6. Melhoramentos

Dispondo de mais tempo para melhorar a qualidade do trabalho realizado, seria despendido principalmente realizando mais otimizações que permitam melhorar o desempenho de operações pesadas que sejam realizadas com frequência como a comparação de estados (*LanduseAllocations*), tanto para os ordenar como para verificar igualdade. Adicionalmente, poderiam ser pensadas algumas formas de determinar mais precocemente quando é que um estado já não pode levar a uma solução válida, permitindo podar a árvore de pesquisa mais cedo. Por fim, tentar-se-ia melhorar a função heurística de modo a que se aproxime ainda mais do valor real, mas sem comprometer a sua admissibilidade ou eficiência, já que se se demorar demasiado tempo a computar a heurística, esta deixa de ser útil.

7. Recursos

7.1. Bibliografia

1. Russel, Stuart; Norvig, Peter: *Artificial Intelligence: A Modern Approach*. Prentice-Hall, New Jersey (2009)
2. Sriram, Ram D.*: Intelligent Systems for Engineering*, Springer (1997), pgs. 64-71
3. Nilsson, Nills: *Artificial Intelligence: A new synthesis*. Morgan Kaufmann (1998)
4. Oliveira, Eugénio: Apontamentos das aulas da cadeira de Inteligência Artificial, http://paginas.fe.up.pt/~eol/IA/1314/apontamentos.html. Disponível a 28 de Maio de 2014.
5. Gero, J.; Sudweeks, F. (Editors): *AI in Design '96*. Kluwer Academic Publishers (1996)
6. Stewart, Theodor J.; Janssenb, Ron; Herwijnenb, Marjan: *A genetic algorithm approach to multiobjective land use planning*. College of Information Sciences and Technologies, Penn State University (2004)

7.2. *Software*

1. *Microsoft Visual Studio 2013 Ultimate*
2. *Microsoft Windows*

7.3. Distribuição de Trabalho

Duarte Duarte – 50%

Luís Cleto – 50%

8. Apêndice - Manual do Utilizador