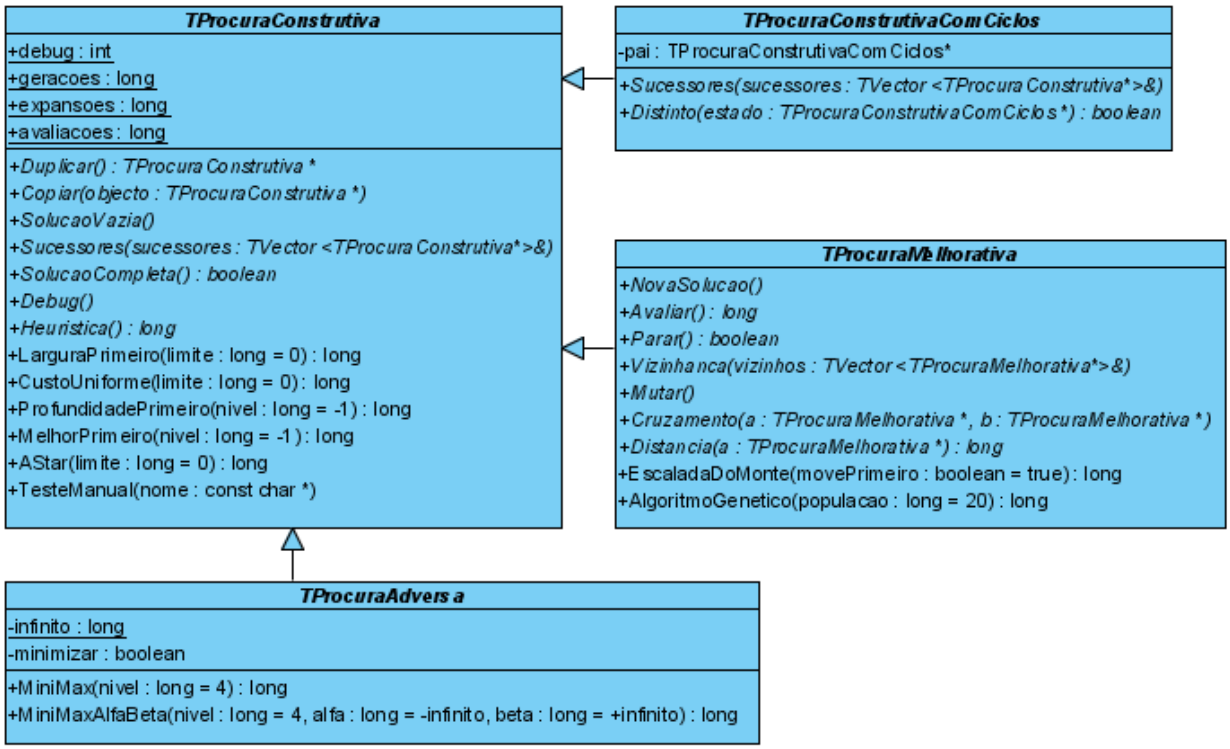
Algoritmos de Introdução à Inteligência Artificial

Neste texto está uma breve introdução ao código disponível e aos algoritmos e problemas implementados. Os algoritmos são genéricos, aplicáveis a um conjunto alargado de problemas de procura, sendo indicados para os problemas *NP-hard* e *P-Space*.

Este código tem dois objectivos distintos. Primeiro destina-se a permitir-lhe ver, com o detalhe que entender, como funcionam os algoritmos base de Procura, para resolver problemas concretos, implementados e sem caixas pretas, bem como os limites de certas procuras, no seu próprio computador. Como segundo objectivo destina-se a servir de base à implementação de novos algoritmos e/ou problemas concretos, que de outra forma seria testado apenas um ou dois algoritmos para um só problema durante toda a unidade curricular, e isso já com grande esforço, enquanto que desta forma os algoritmos base não necessitam de ser implementados, permitindo assim utilizar todo o tempo no problema específico em causa, sem impedir naturalmente que se implemente um novo algoritmo dedicado. Desta forma pretende-se acomular a experiência necessária para que saiba realmente dar respostas a problemas concretos que lhe venham a surgir pela frente.

Para suportar os algoritmos de procura, são construidas as 4 superclasses no diagrama seguinte, sendo indicados os atributos e métodos mais importantes. Para implementar de um problema concreto, tem que se redefinir uma destas superclasses, e implementar os métodos virtuais respectivos, podendo nessa altura chamar as procuras implementadas na superclasse.



A classe *TProcuraConstrutiva* é a superclasse de todas as outras, e tem 3 variáveis globais com contagens da procura (geracoes, expansoes, avaliacoes) e uma variável com o nível de informação de *debug*, ou seja, o nível de detalhe que o utilizador pretende ser informado de como a procura está a decorrer. Estas variáveis são globais, pelo que não são dublicadas em cada instância, mas também não é possível ter duas procuras a correr em simultâneo.

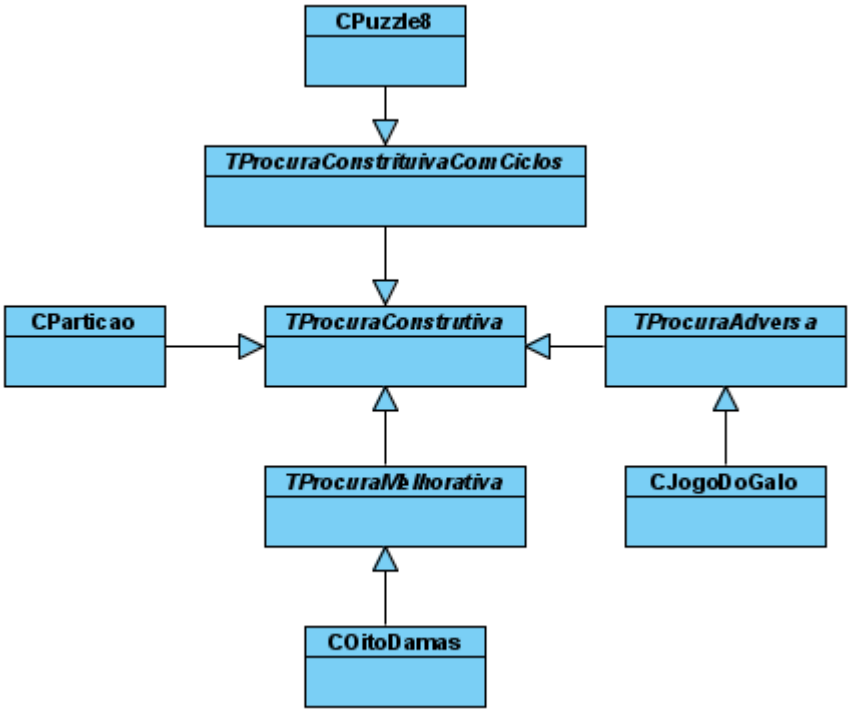
Esta classe tem implementadas as procuras cegas, e das informadas o *A** e o *MelhorPrimeiro*, sendo cada estado uma instância da classe, representando uma solução parcial, que vai sendo construída até que se obtenha um estado final, altura em que a solução ou é o estado final, ou o caminho desde o estado inicial ao estado final. Esta classe não tem variáveis de instância, pelo que não ocupa espaço nenhum, devendo ser a classe redefinida a colocar as variáveis de estado. Deve redefinir os métodos virtuais, mas pode não redefinir o método *Heurística* se apenas chamar as procuras cegas.

A classe *TProcuraConstrutivaComCiclos* destina-se a adicionar o código necessário quando temos um problema em que o mesmo estado pode voltar a ocorrer. Para evitar que a procura explore estados desnecessários, um teste extra é feito no método da geração de sucessores, de forma a remover os estados já explorados. A classe concreta não necessita de ter código específico para remover os sucessores.

A classe *TProcuraMelhorativa* permite correr procuras no espaço das soluções completas, ou seja, um estado representa agora uma solução completa para o problema em questão, pretendendo-se optimizar o valor da solução. Dois algoritmos estão implementados, a *EscaladaDoMonte* e o *AlgoritmoGenético* (uma versão). Para tal há que redefinir os métodos *Duplicar* e *Copiar* da superclasse *TProcuraConstrutiva*, mas não é necessário definir qualquer outro método se se pretender executar apenas os algoritmos melhorativos. Da classe *TProcuraMelhorativa* tem que se redefinir os métodos *NovaSolucao*, *Avaliar*, *Parar*, que embora tenha parcenças com os métodos *SolucaoVazia*, *Heuristica* e *SolucaoCompleta*, são completamente distintos, dado que uma solução completa é uma hipótese de resposta do algoritmo, que tem um determinado valor (método *Avaliar*), conhecido, enquanto que uma solução vazia não é em princípio uma hipótese de resposta do algoritmo, nem o seu real valor é conhecido a não ser que seja uma solução completa, daí a utilização do nome da função *Heuristica*. O método *SolucaoCompleta* testa se o estado currente é uma solução completa, enquanto que o método *Parar* verifica se os critérios de paragem foram ou não atingidos. Mantendo estes métodos distintos, pode-se ter para o mesmo problema, na mesma classe uma implementação construtiva e outra melhorativa, permitindo assim ter todos os algoritmos disponíveis.

A classe *TProcuraAdversa* tem uma variável global *infinito*, que representa a vitória das brancas, sendo que o -infinito representa a vitória das pretas. Tem ainda uma variável de instância para indicar se se deve minimizar ou maximizar o resultado, o que depende naturalmente do jogador actual. Dois algoritmos estão disponíveis, o *MiniMax* e o *MiniMaxAlfaBeta*. Para esta classe não tem sentido correr os algoritmos da superclasse, mas se se correr teria-se uma situação em que um adversário jogava a favor do outro.

No diagrama de classes seguinte, indicam-se as classes dos problemas implementados.



O problema Puzzle8 pode ter ciclos, pelo que deve herdar de *TProcuraConstrutivaComCiclos*. É no entanto interesante colocar a herança directamente da classe *TProcuraConstrutiva*, para analisar as diferenças. O problema Particao é um problema clássico *NP-hard*, e o problema das OitoDamas tem implementados os métodos que permitem executar tanto procuras construtivas como melhorativas. Finalmente o jogo do galo é o exemplo de um problema para a procura adversa.

Estes problemas concretos são propositadamente em número reduzido, apenas um problema para cada superclasse, de forma a que possa aprender o máximo das suas diferenças, e consiga para um novo problema saber que abordagens pode implementar, tendo para tal um percurso sugerido na primeira actividade formativa de cada módulo. A sua análise deve natualmente ser complementada com a implementação de problemas concretos, que é proposto na segunda actividade formativa de cada módulo.