Pentago

Relatório Final

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Inteligência Artificial

Grupo
Ângela Filipa Pereira Cardoso - 200204375 - angela.cardoso@fe.up.pt
Bruno Miguel Dias Madeira - 201306619 - up201306619@fe.up.pt
Franscisco José Lopes Veiga- 201201604 - ei12170@fe.up.pt

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

29 de Maio de 2016

Conteúdo

1 Objetivo

1.1 O Jogo Pentago

O Pentago é um jogo de estratégia para dois jogadores, jogado num tabuleiro 6×6 dividido em 4 quadrantes 3×3, que podem ser rodados. Além do tabuleiro, o jogo é composto por 18 peças brancas, que serão usadas por um dos jogadores, e 18 peças pretas, que serão usadas pelo outro.

Alternadamente, os jogadores colocam uma peça da sua cor numa casa vazia, e depois rodam um dos quadrantes 90 graus num sentido à escolha. Um jogador ganha o jogo se conseguir colocar 5 peças em linha contínua na horizontal, diagonal ou vertical, antes ou depois da rotação. Há ainda duas situações em que o jogo pode terminar em empate: se após uma rotação ambos os jogadores fizerem 5 em linha simultaneamente; ou se todas as 36 peças forem colocadas no tabuleiro e a última rotação for feita, sem que nenhum jogador obtenha 5 em linha.

Por se tratar de um jogo com regras muito simples, rapidamente se aprende a jogar. No entanto, o facto de se poderem rodar os quadrantes, faz com que por vezes seja difícil ver o objetivo do adversário, e até jogadores mais experientes podem ser derrotados por distrações. Neste sentido, é um jogo interessante, porque não é fácil de dominar.

1.2 Uma Implementação do Pentago com Inteligência Artificial

O nosso objetivo neste trabalho foi realizar uma aplicação para jogar Pentago no computador, com especial enfoque na criação de adversários com inteligência artificial. A aplicação permite realizar jogos em três modos distintos: humano vs humano, humano vs computador e computador vs computador. Este último modo, apesar de não ser muito comum em aplicações do mesmo tipo, é bastante útil, porque permite determinar qual a melhor entre duas implementações diferentes do jogador computador.

Dado que o ganho de um jogador corresponde exatamente à perda do outro, o Pentago é um jogo de soma zero. Neste tipo de jogos, o algoritmo minmax e as suas variantes são particularmente adequados, dado que procuram minimizar as perdas e maximizar os ganhos. Para implementar estes algoritmos, é necessário criar heurísticas que dado um tabuleiro avaliem o seu valor do ponto de vista de um dos jogadores. Neste relatório descrevemos as nossas implementações dos algoritmos minmax e as heurísticas que criamos para os jogadores computador.

2 Especificação

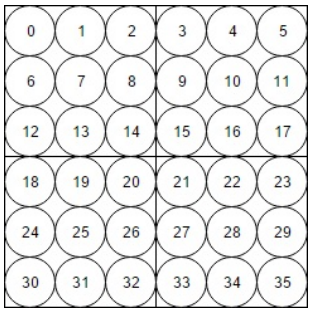
2.1 Representação do tabuleiro

2.1.1 Array Unidimensional (usada no projeto principal)

Na representação que decidimos usar um tabuleiro de Pentago é um array unidimensional indexado dado por:

$$3Qx + 18Qy + Px + 6Py \qquad Qx, Qy \in 0, 1 \wedge x, y \in 0, 2.$$

Onde Qx e Qy indicam qual o quadrante e x e y a posição dentro do quadrante. A figura seguinte ilustra melhor como são mapeados os indexes do array no tabuleiro:



Na prática, no código desenvolvido, não usamos Qx e Qy mas apenas uma única variável para escolher um quadrante quando necessitamos de realizar operações no tabuleiro. A lógica subjacente continua a mesma, apenas mais ofuscada, fazendo a indexação de quadrantes analogamente à leitura de texto (da esquerda para direita e de cima para baixo).

Além do array, na implementação, incluímos também o turno, isto é o próximo a jogar, e o estado da jogada, isto é, colocar peça ou rodar quadrante.

2.1.2 72 bits - versão "Pandora"

Uma das formas de representação do tabuleiro considerada foi como uma estrutura de 72 bits (9 bytes) que podem ser usados, na linguagem de programação usada, com apenas duas variáveis: uma do tipo `long` e outra do tipo `char`.

Para cada posição do tabuleiro são utilizados 2 bits, sendo que apenas 3 das 4 possíveis configurações de bits seriam usadas. Como o tabuleiro tem 36 posições, ao todo são necessários $2 * 36 = 72$ bits.

Apesar de não ser usada no projeto principal, esta representação foi implementada em *PentagoPandora.cs* (3.3.2).

2.1.3 55 bits - apenas teorizado

Removendo simetrias, o número de estados que um jogo de Pentago pode ter é 3.009.081.623.421.558. Isto significa que o número de estados pode ser representado em 52bits, mais alguns extra para representar as simetrias. Se houver um algoritmo que permita inferir os tabuleiros fazendo uso da gama de números de 0 ao valor acima, é possível representar um tabuleiro numa única variável que poderia ser útil para poupar recursos de memória. Tal hipótese levou a algum investimento na consideração das diferentes representações possíveis do tabuleiro.

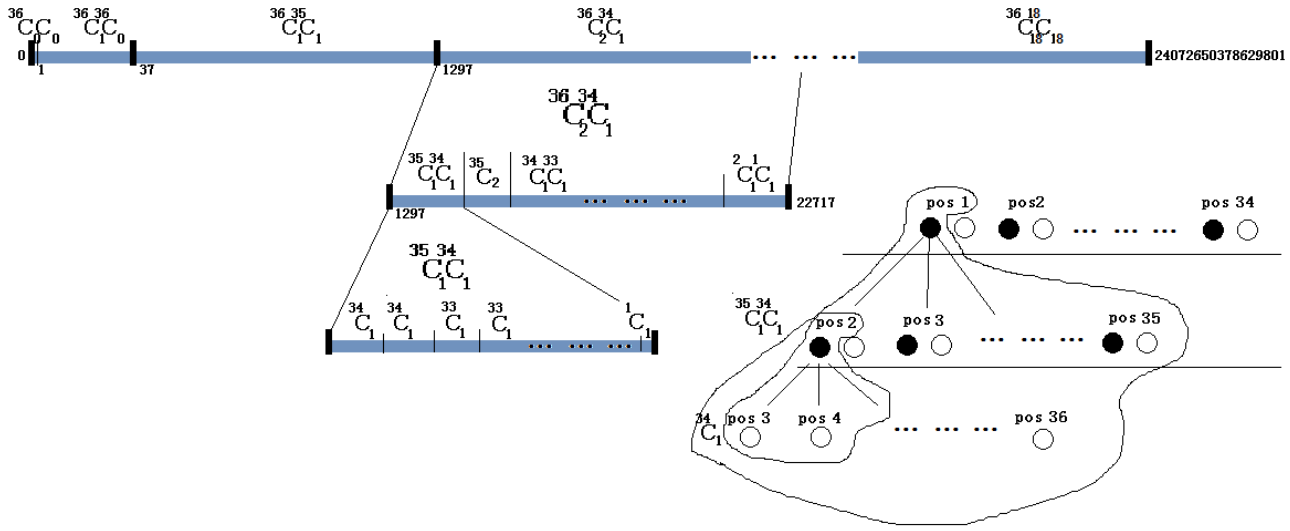
Uma das forma de representação que conseguimos teorizar necessitaria de apenas 55 bits para representar o tabuleiro, sendo que qualquer estado possível seria identificado por um número único. Ela consiste em considerar todas as possíveis combinações de peças do tabuleiro, sendo que a busca do estado seria feita por "camadas", inferindo em cada camada informação sobre o tabuleiro que seria usada também nas camadas seguintes. Esta forma teria um número reduzido de estados impossíveis, que representam estados de tabuleiro aparentemente corretos mas

que nunca poderiam ser atingidos devido às condições de fim de jogo, sendo por isso bastante compacta. Apesar de poupar memória e ser possível descobrir se 2 estados de tabuleiro são iguais rapidamente, saber qual o tabuleiro que um número representa seria pouco eficiente pelo que a ideia foi descartada do ponto de vista de implementação. Apresentamos contudo uma representação de como se poderia inferir o estado do tabuleiro a partir de um número na gama de 55 bits e qual a fórmula usada no cálculo do número de estados.

$$Nº\ de\ Estados = \sum_{i=1}^{18}\left\{\binom{36}{i} * \left(\binom{36-i}{i} + \binom{36-i}{i-1}\right)\right\} + \binom{36}{0} = 24072650378629801$$

$$Nº\ de\ bits\ necessário\ para\ representar = \lceil \log_2 24072650378629801 \rceil = 55$$

$$\left\{ {}^{36\ 36-i}_{i\ \ i}C : i \in N\ \ 0<=i<=18 \right\} \cup \left\{ {}^{36\ 36-i}_{i\ \ i-1}C : i \in N\ \ 0<i<=18 \right\}$$



2.2 Algoritmos

2.2.1 Minmax

Quando o jogador computador tem que efetuar uma jogada, o algoritmo Minmax determina quais as jogadas possíveis. Cada uma destas jogadas é analisada da seguinte forma:

- se tiver sido atingido o nível de profundidade especificado para o algoritmo, é calculado um valor para o tabuleiro, usando uma heurística do ponto de vista do computador;
- se a jogada conduzir ao final do jogo, é atribuído um valor ao tabuleiro tendo em conta quem ganhou, isto é, um valor positivo muito alto se ganha o computador ou um valor negativo muito baixo se ganha o adversário. Estes valores, assim como o valor atribuído em caso de empate, podem ser definidos por quem utilize a nossa implementação do Minimax e podem não ser constantes. No caso do Pentado implementado, fazemos uso do valor da profundidade para distinguir entre derrotas e vitórias que ocorrem em diferentes profundidades;
- se nenhuma das condições anteriores se verificar, são calculadas todas as jogadas seguintes possíveis e o processo repete-se.

Para determinar o valor da jogada original (a decisão que o computador tem que tomar no momento), entra a parte do Minmax propriamente dita:

- se for a vez do adversário, o valor do tabuleiro é o mínimo dos valores das possíveis jogadas seguintes;
- se for a vez do computador, o valor do tabuleiro é o máximo dos valores das possíveis jogadas seguintes.

A ideia é bastante simples, dado que assumimos que o adversário fará o melhor para ele, ou seja o pior para o computador e, portanto, escolherá a jogada com um valor mínimo do ponto de vista do computador. Por outro lado, o computador a cada passo escolhe a jogada com um valor máximo para si, o que também faz todo o sentido.

2.2.2 Minmax Alfa-Beta

Para melhorar a eficiência do algoritmo Minmax, em termos temporais, utilizam-se cortes Alfa-Beta, que cortam ramos da árvore de decisão cuja análise é desnecessária. Em cada nó da árvore, calculam-se valores α e β:

- α é o melhor valor já encontrado para o maximizador desde esse nó até à raiz;
- β é o melhor valor já encontrado para o minimizador desde esse nó até à raiz.

Nos nós maximizadores atualiza-se α se um dos seus nós filhos apresenta maior valor que o α atual. Nos nós minimizadores atualiza-se β se um dos seus nós filhos apresenta menor valor que o β atual. Sempre que $\alpha \geq \beta$ efetuamos uma poda, isto é, não continuamos com a análise desses ramos da árvore, porque já sabemos que não apresentarão resultados que nos interessem.

2.2.3 Minmax Alfa-Beta Delimitado

Este é um algoritmo consiste a adição de uma modificação ao algoritmo original de Alfa-Beta. Nele é possível definir um limite máximo e mínimo. Quando um nodo do tipo Max encontra um valor superior ao limite máximo ou um nodo do tipo encontra um valor inferior ao limite mínimo considera-se que se encontrou uma solução suficientemente boa, não se visitando possíveis jogadas ainda visitadas num determinado nodo.

Com esta abordagem podemos facilmente cortar nós após um nodo encontrar uma jogada vencedora pois o valor de utilidade atribuído a estes tabuleiros é muito superior ao obtido, geralmente, pelas heurísticas. Além disso também é possível usar um Minmax mais flexível, que tenha mais consideração pela performance temporal em detrimento da precisão do resultado obtido.

2.3 Funcionalidades e Otimizações

2.3.1 Generalização para múltiplas jogadas

Os algoritmos implementados permitem a realização de vários movimentos (sub-jogadas) por um jogador, o que é um detalhe crítico no Pentago cuja vitória pode ser alcançada após o colocar de uma peça sem se realizar a rotação dos quadrantes. Detalhes da implementação são mencionados na secção 3.2 deste relatório.

2.3.2 Remoção de duplicados

A fim de tentar melhorar a performance temporal são identificados e descartados tabuleiros iguais quando são efetuadas rotações de quadrantes dentro do Minmax.

Esta verificação compara todas as posições dos tabuleiros. Provavelmente não constitui a solução mais eficaz. Nesse sentido a representação de 72 bits parece mais tentadora. Claro que também teria algum processamento adicional devido na realização de rotações. Uma vez que não foi usada não podemos tirar conclusões sobre esta alternativa.

Dos testes de eficiência temporal usados esta modificação apenas é benéfica em tabuleiros com poucas peças, perdendo a sua eficiência quando usada com tabuleiros com mais de 7 peças aproximadamente. A partir de 7 peças, para a maioria dos tabuleiros, a eficiência temporal com remoção de duplicados é pior do que aquela sem remoção

de duplicados. No entanto, mesmo nestes casos, a perda de eficiência provocada não é substancial, pelo menos para as profundidades usadas no projeto.

2.3.3 Simetrias

O Pentago é um jogo particularmente interessante do ponto de vista das simetrias. O grupo de simetrias do Pentago é o grupo diedral D_4, que tem 8 elementos:

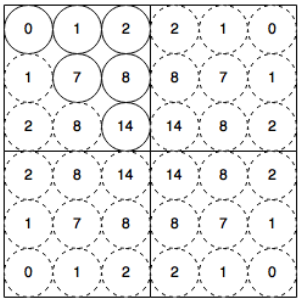
- o elemento neutro ou identidade;
- 3 rotações, de 90, 180 e 270 graus centígrados;
- 4 reflexões, horizontal, vertical e duas diagonais.

Além destas simetrias, por vezes é ainda útil considerar simetrias de cada quadrante. De facto, o grupo de simetrias de cada quadrante é também D_4, mas nós apenas consideramos as reflexões na diagonal principal do primeiro quadrante.

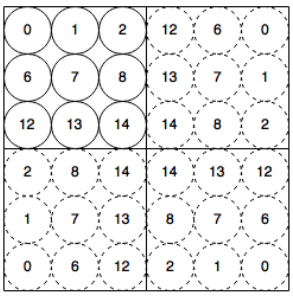
É de notar que quando o tabuleiro é simétrico relativamente à rotação de 90° também é simétrico relativamente às rotações de 180° e de 270°. Além disso, há várias composições de transformações cujo resultado implica outras transformações. Por exemplo, se tiver simetria de reflexão vertical e simetria de reflexão horizontal, então também tem simetria de rotação de 180°. Isto permite evitar verificações desnecessárias de simetrias.

O algoritmo utilizado para validar simetrias é o que se apresenta a seguir.

1. O tabuleiro é simétrico relativamente à rotação de 180°.
   (a) O tabuleiro é simétrico relativamente à rotação de 90°.
      i. O primeiro quadrante é simétrico relativamente à sua diagonal principal. Então basta explorar as próximas jogadas nos nós acima da diagonal principal do primeiro quadrante.

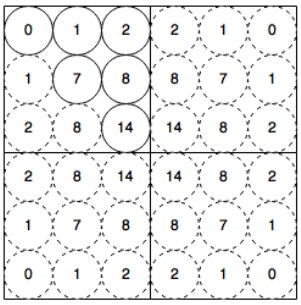


      ii. O primeiro quadrante não é simétrico relativamente à sua diagonal principal. Então basta explorar as próximas jogadas nos nós do primeiro quadrante.

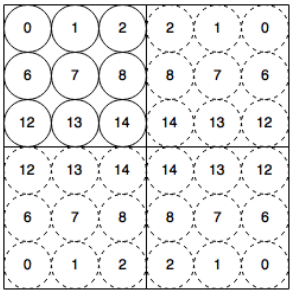


(b) O tabuleiro não é simétrico relativamente à rotação de 90°, mas é simétrico relativamente à reflexão vertical.

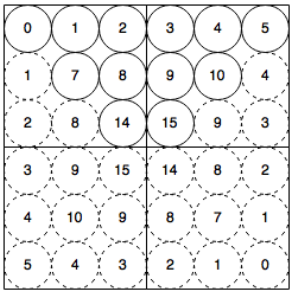
i. O primeiro quadrante é simétrico relativamente à sua diagonal principal. Então basta explorar as próximas jogadas nos nós acima da diagonal principal do primeiro quadrante.



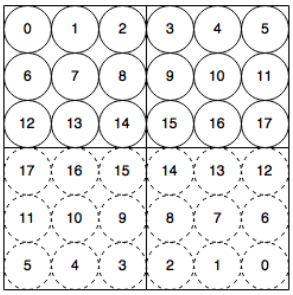
ii. O primeiro quadrante não é simétrico relativamente à sua diagonal principal. Então basta explorar as próximas jogadas nos nós do primeiro quadrante.



(c) O tabuleiro não é simétrico relativamente à rotação de 90°, nem é relativamente à reflexão vertical, mas é simétrico relativamente à reflexão na diagonal principal. Então basta explorar as próximas jogadas no *triangulo superior*.

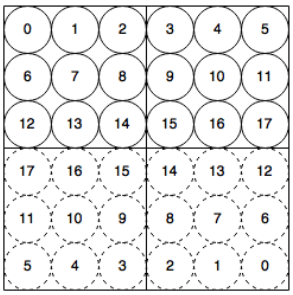


(d) O tabuleiro não é simétrico relativamente à rotação de 90°, nem é relativamente à reflexão vertical, nem relativamente à reflexão na diagonal principal. Então basta explorar as próximas jogadas nos nós dos dois primeiros quadrantes.

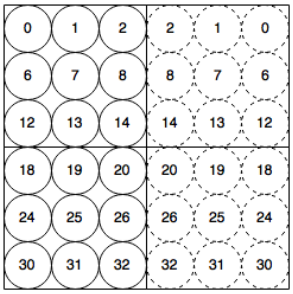


2. O tabuleiro não é simétrico relativamente à rotação de 180°.

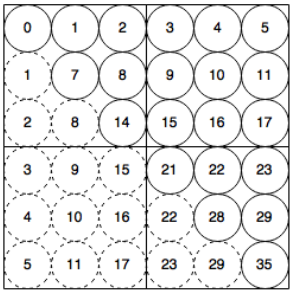
   (a) O tabuleiro é simétrico relativamente à reflexão vertical. Então basta explorar as próximas jogadas nos nós dos dois primeiros quadrantes.



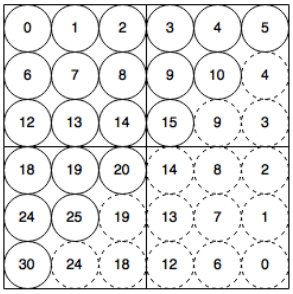
   (b) O tabuleiro não é simétrico relativamente à reflexão vertical, mas é simétrico relativamente à reflexão horizontal. Então basta explorar as próximas jogadas no primeiro e no terceiro quadrantes.



   (c) O tabuleiro não é simétrico relativamente à reflexão vertical, nem relativamente à reflexão horizontal, mas é simétrico relativamente à reflexão na diagonal principal. Então basta explorar as próximas jogadas acima da diagonal principal.



(d) O tabuleiro não é simétrico relativamente à reflexão vertical, nem relativamente à reflexão horizontal, nem relativamente à reflexão na diagonal principal, mas é simétrico relativamente à reflexão na anti diagonal. Então basta explorar as próximas jogadas acima da anti diagonal.



Uma vez que apenas podem existir simetrias acontecer quando o número de peças no tabuleiro é par, para agilizar o processamento, apenas verificamos as simetrias nesta situação. Quando o número de peças no tabuleiro é muito reduzido, a verificação de simetrias introduz melhorias significativas. Uma vez perante um número de peças maior os resultados são significativamente piores. Na versão final do projeto, foi colocada uma restrição para apenas ser efetuada validação de simetrias para tabuleiros com um número de peças reduzido. Esta restrição está em vigor nos testes de eficiência temporal apresentados.

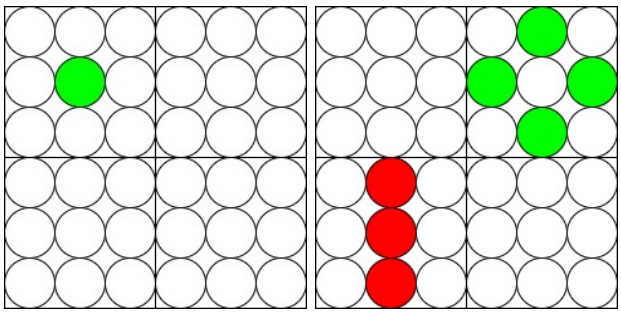
2.3.4 Profundidade Automática

Ao aumentar o número de peças de um tabuleiro o número médio de nodos filho a analisar por cada nodo no Minmax diminui. Assim é possível analisar a uma profundidade mais elevada, mantendo (aproximadamente) o mesmo tempo de execução. As profundidades usadas automaticamente face ao número de peças foram escolhidas de acordo com palpites educados, calculando o número aproximado de nós a serem visitados, sendo depois realizados ajustes manualmente após a realização de testes de eficiência temporal. Por prevenção foi dada alguma margem de erro, isto é, algumas profundidades podiam ser aplicadas com um maior número de peças no tabuleiro do que aquele que está estabelecido no código.

2.4 Heurísticas

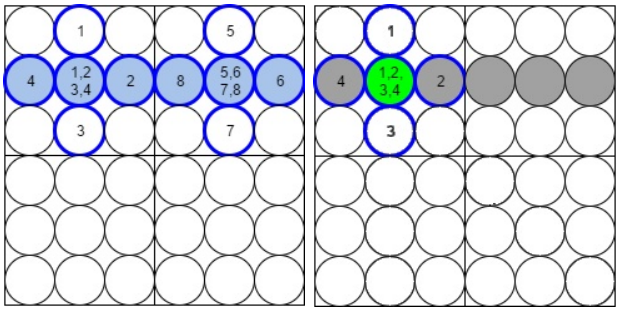
2.4.1 Heurística 1

A heurística 1, cujo o nome vem do facto de ter sido a primeira a ser implementada, tenta analisar as possibilidades de vitória para ambos os jogadores de uma forma relativamente ingénua. Para explicar como funciona observem-se os seguintes tabuleiros, onde os círculos a verde representam posições ocupadas pelo jogador 1, e a vermelhos as ocupadas pelo jogador 2.

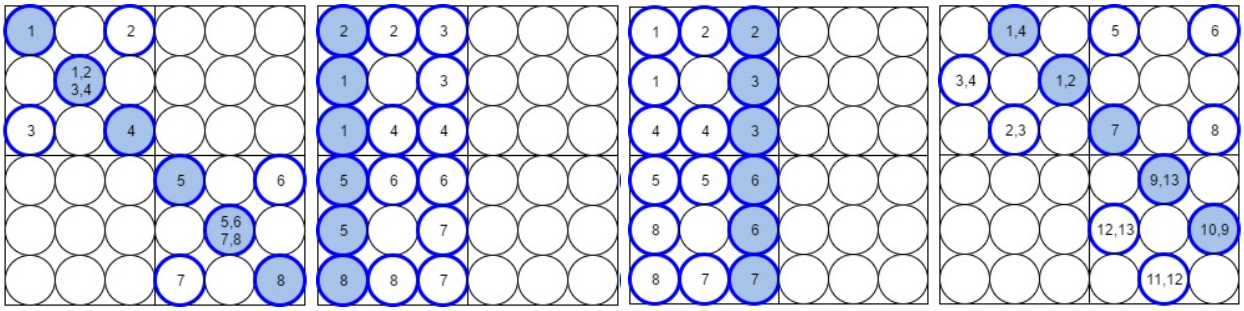


Se atentarmos à segunda linha horizontal, em ambos os tabuleiros, podemos concluir rapidamente que o jogador 2 nunca poderá efetuar quatro em linha na segunda linha horizontal uma vez que esta se encontra bloqueada pelas peças do jogador 1.

Podemos averiguar que temos 4 conjuntos de posições em cada um dos quadrantes que contêm a segunda linha horizontal. Quando um dos quadrantes fica com os 4 conjuntos ocupados por peças de um jogador deixa de ser possível ao jogador adversário usar essa linha para ganhar o jogo. As figuras seguintes ajudam a ilustrar esta situação no caso da segunda linha horizontal.



A heurística 1 considera que o *número de possibilidades* que um jogador tem para fazer 5 em linha na segunda linha horizontal é igual ao mínimo número de grupos livres (ainda não ocupados pelo adversário) entre os dois quadrantes que contêm a linha. Este raciocínio é estendido para as outras linhas que podem ser usadas, sendo no caso das diagonais menores, considerados o mínimo de 3 quadrantes. As figuras seguintes ajudam a perceber o quais os grupos usados para cada linha em cada um dos quadrantes.



A heurística inclui também uma variável `bias` que permite dar mais peso às possibilidades dos jogador ou às possibilidades do jogador adversário. Assim é possível parametrizar a heurística de modo a que tenha um comportamento mais otimista, pessimista, defensivo ou ofensivo. A seguinte fórmula apresenta como é obtido um valor pela heurística, onde Pm é o número de possibilidades para o próprio, Po o número de possibilidades para o oponente e alfa a variável de `bias`.

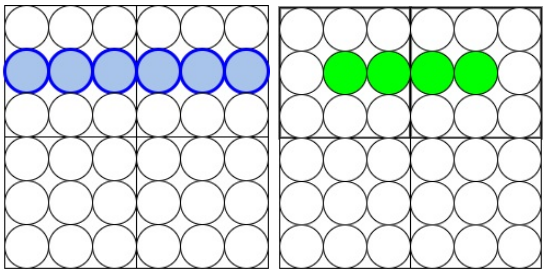
$$\alpha Pm - (1 - \alpha)Po \qquad \alpha \in [0, 1] \tag{1}$$

Esta consideração é algo ingénua pois é possível que o adversário, fazendo uso das rotações, não permitir o uso de determinadas linhas enganando assim a heurística. Além disso se não é possível usar o minmax para evitar jogadas fatalistas no inicio do jogo, dado existirem demasiadas possibilidades a analisar, a heurística 1, isoladamente, não permite criar um adversário muito forte pois é muito passiva em relação ao estado do tabuleiro, não averiguando derrotas eminentes.

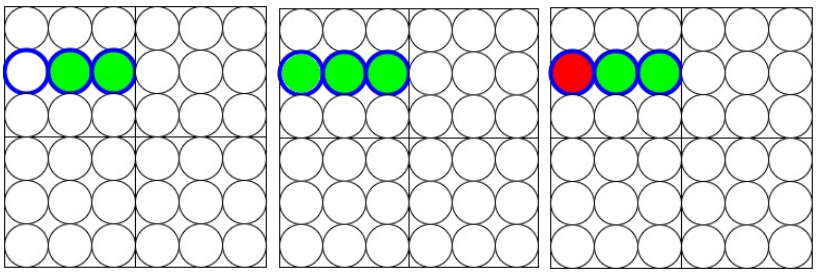
Foram também adicionados pesos aos diversos tipos de linhas. Assim, é possível ao algoritmo priorizar ou desprezar alguns tipos de linhas. Sem entrar em detalhe nos tipos de linhas considerados, intuitivamente estes pesos parecem ser importantes para desprezar as diagonais pequenas que não podem ser usadas em ataques de duas pontas e que ocupam 3 quadrantes. Foi realizada alguma experimentação com estes pesos mas não se chegou a uma solução ideal e não foram realizados testes similares aos apresentados nos anexos dado que já existiam demasiadas variáveis a testar. A adição destas nos testes ia aumentar o número de testes a realizar consideravelmente caso se testasse para todas as possíveis combinações dos `biases`(pesos) na heurística 1 e 1.2.

2.4.2 Heurística 1.2

A heurística 1.2 surgiu como na tentava de melhorar a heurística 1. Esta heurística faz o que a sua predecessora já fazia e adicionalmente mais algumas avaliações semelhantes que visão evitar jogadas fatalistas ou aproveitar oportunidades ofensivas. Para tal a heurística considera além do *número de possibilidades* o *número de possibilidades fortes*. Para explicar o que o algoritmo entende como possibilidades fortes vamos atentar a segunda linha horizontal do tabuleiro do Pentago.



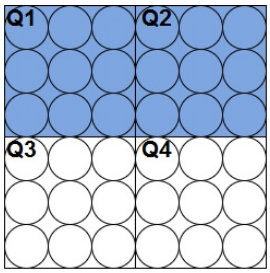
Podemos observar que o tipo de padrão usado na figura à direita é interessante do ponto de vista ofensivo, pois cria uma vitória garantida, e é contra este género de situações que o algoritmo tenta precaver-se ou tirar proveito ofensivo. Para tal, o algoritmo contabiliza 2 tipos de casos. Denominemos esses tipos de tipo *forte* e de tipo *muito forte* sendo que qualquer caso que seja do tipo muito forte é também um caso do tipo forte mas o inverso pode não ser verdade. Considera-se um caso forte quando um grupo, dos definidos na heurística 1, está ocupado por peças de um jogador. Um caso muito forte acontece quando um grupo está ocupado por peças de um jogador e a sua extensão no mesmo quadrante se encontra sem uma peça adversária (podendo estar vazia ou não). Assim as 3 imagens seguintes ilustram respetivamente 2 jogadas do tipo muito forte e uma do tipo forte.



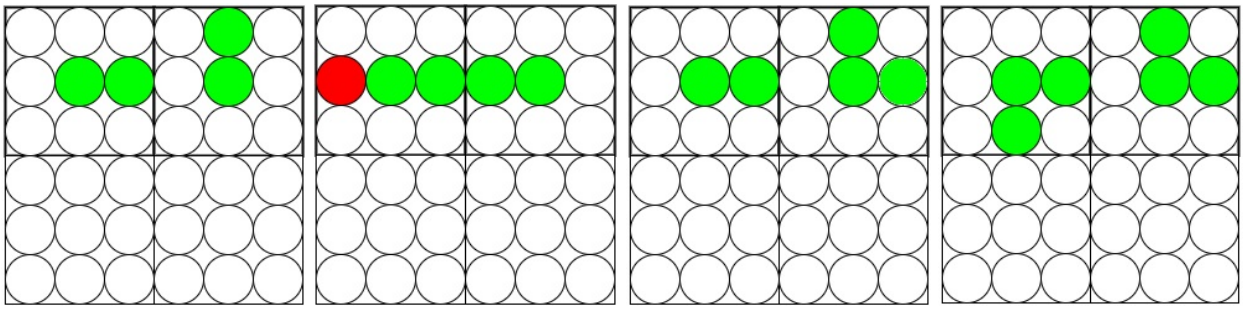
Depois de contabilizadas estes casos o número de jogadas fortes para a segunda linha horizontal é dada pela seguinte fórmula:

$$F_hor2 = max(min(F_1, M_1), min(M_2, F_2)) \quad (2)$$

Onde Fi e Mi representam respetivamente o número de casos fortes e o número de casos muito fortes no quadrante i. Os quadrantes usados, são os que contêm a linha como mostra a figura seguinte.

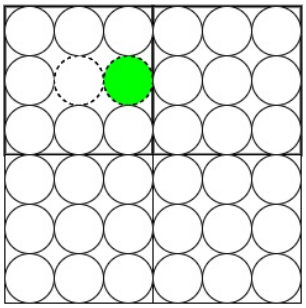


Assim, para melhor ilustrar a contabilização referida, são apresentadas abaixo exemplos onde o número de jogadas fortes para a segunda linha horizontal são respetivamente 1, 1, 1 e 2.



A mesmos cálculos são efetuados para as outras linhas (excluindo s diagonais menores que são tratadas de forma diferente) para ambos os jogadores. A heurística usa o número de jogadas fortes para o jogador (com valor positivo), para adversário (com valor negativo) e também as possibilidades da heurísticas 1. Todas estas componentes têm um peso associado que confere à heurística alguma flexibilidade podendo esta variar não só entre uma abordagem otimista ou pessimista mas também por abordagem mais passiva ou reativa.

Apesar destas considerações a heurística têm algumas dificuldades em lidar com limitações de profundidade no Minmax pelo que foi implantada uma versão relaxada. Na sua versão relaxada consideram-se como fortes os blocos ocupados com pelo menos uma peça do jogador e nenhuma do adversário, tal como apresentado na ilustração abaixo.



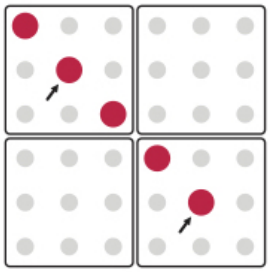
Apesar de melhorar a sua predecessora também herda as suas fraquezas não sabendo lidar com as rotações do jogador adversário.

2.4.3 Heurística A

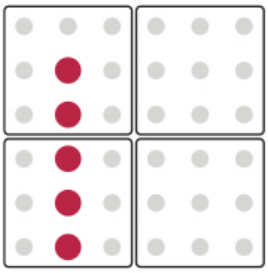
Esta heurística foi inspirada pelo guia estratégico [2].

Há 4 formas de obter 5 em linha e ganhar um jogo de Pentago:

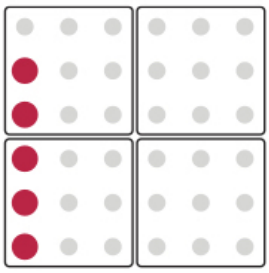
- Monica - numa das diagonais principais do tabuleiro - força relativa 3



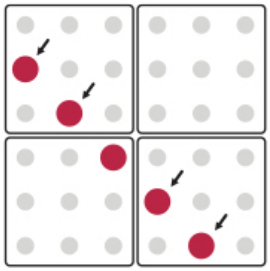
- Meio - na vertical ou na horizontal através do meio de dois quadrantes adjacentes - força relativa 5



- Direito - na vertical ou na horizontal usando os bordos de dois quadrantes adjacentes - força relativa 7



- Tripla - diagonalmente, abaixo ou acima das diagonais principais do tabuleiro - força relativa 9



Para cada uma destas estratégias, considera-se a sua versatilidade, isto é, o número de diferentes possibilidades de aproveitar parte das peças quando um dos caminhos é bloqueado pelo adversário. Além disso, também nos interessa a dificuldade que o adversário tem de se defender contra um jogador que use a estratégia. São esses dois fatores que determinam a força relativa.

Dada uma posição do jogo, a heurística A começa por percorrer cada uma das linhas em que é possível ganhar e calcular o número de peças de cada cor. No entanto, apenas nos interessa contabilizar as peças quando há possibilidade do jogador conseguir ganhar nessa linha. Para isso, vamos verificar quantas peças há no interior da linha e quantas peças há nos bordos, dado que peças do adversário no interior bloqueiam imediatamente e para bloquear nos bordos são necessárias duas peças. Depois de efetuada esta contagem, sabemos quão forte é a posição de cada jogador em cada linha. Por exemplo, se há 3 peças brancas no interior e nenhuma peça preta em toda a linha, a posição branca é muito forte, porque basta mais uma peça para assegurar a sua vitória.

No passo seguinte, usamos as forças relativas e as contagens para calcular o valor do tabuleiro. Sendo b o número de peças brancas, p o número de peças pretas e r a força relativa na linha em questão, adicionamos $r^b - r^p$ ao valor previamente calculado. Este valor está a ser calculado assumindo que a IA está a jogar com as peças brancas, mas caso esteja a jogar com as pretas, basta multiplicar por -1 no final. O motivo pelo qual usamos a potência, é para garantir que ter várias peças na mesma linha é mais valorizado do que ter as mesmas peças espalhadas por linhas diferentes.

O método acima é usado para calcular o valor de todos os 8 tabuleiros que se obtém do tabuleiro atual por rotação de um quadrante. Depois disso, é usado o máximo ou o mínimo destes tabuleiros rodados, consoante o próximo jogador é a IA ou o adversário, tal como no minmax.

Em relação às forças relativas, apesar de querermos favorecer determinadas estratégias em detrimento de outras, não usamos os valores 3, 5, 7, e 9, porque devido ao uso de potências, eles conduzem a discrepâncias enormes entre jogadas, que fariam com que algumas das estratégias fossem completamente desprezadas. Sendo assim, após alguns cálculos com várias posições de jogo, optamos pelos valores 1.13, 1.15, 1.17 e 1.19. Isto contribui também para que os valores finais do tabuleiro sejam pequenos, o que será usado noutras considerações da heurística como veremos adiante. Dada a arbitrariedade da escolha dos valores, decidimos efetuar experiências estatísticas para determinar os melhores valores, mas isso será visto mais adiante na secção das experiências.

Observando jogos em que a heurística A perdia, rapidamente conseguimos determinar algumas das suas falhas. A primeira destas observações é que se um jogador conseguir 4 em linha com as duas pontas abertas (em jogadas do tipo monica, meio ou direito), então irá ganhar na próxima jogada, a não ser que o seu adversário ganhe já nesta. Assim, ao detetar estas situações a heurística A ignora todos os outros cálculos e dá valor 100 ao tabuleiro se ganha e -100 se perde.

Outra maneira certa de ganhar é conseguir duas ou mais linhas em que apenas falta uma peça para ganhar. Estas situações também foram introduzidas como exceção na heurística A, embora possam levar a alguns falsos positivos, quando ambas as linhas de vitória necessitam de uma peça na mesma posição do tabuleiro.

Finalmente, fizemos ainda algumas pequenas alterações aos valores da heurística:

- se tivermos 4 peças em linha com as pontas abertas, mas for a vez do adversário, calculamos o valor como se fossem 10 peças, para que seja claramente mais valorizada que as restantes;

- se faltar apenas uma peça para obter 5 em linha e não estivermos na situação anterior, incrementamos a contagem de peças em 1;
- se houver 1, 2 ou 3 peças não bloqueadas na mesma linha (monica, meio ou direito), como ambas as pontas abertas, incrementamos a contagem em 1.

2.4.4 Heurística A hacked

A heurística A hacked não apresenta nenhuma construção nova face às anteriores. Na verdade apenas aparece na secção de especificação para ser referida na mesma secção das outras heurísticas neste relatório. A heurística A hacked resulta de experimentação, tendo sido descoberto, combinando pesos das diferentes heurísticas implementadas, que a heurística A tinha alguma facilidade em perder nas diagonais principais. A heurística A hacked apareceu como uma contra-heurística à A fazendo uso de funcionalidades da heurística 1.2 para evitar perder nas diagonais principais. Atualmente a heurística A star apresenta melhores resultados ao usar uma profundidade de 4 (2 jogadas).

2.4.5 Heurística A star

Esta heurística é uma melhoria da heurística A. A ideia básica é a mesma, mas introduzimos mais algumas verificações e alteramos outras. Além disso, nesta heurística também é possível determinar se se estudam todos os tabuleiros rodados quando a jogada é de rotação. No entanto, como veremos na secção das experiências, o desempenho da heurística é melhor quando se estudam todas as rotações.

Uma das mudanças essenciais nesta heurística é que há uma maior preocupação com quem é o próximo jogador a jogar. Por exemplo, imaginemos que ambos os jogadores têm 4 em linha com as duas pontas abertas. Então, quem irá ganhar é o próximo a jogar. Por outro lado, se apenas um dos jogadores estiver nessa posição, mesmo que não seja a vez dele, ele irá ganhar a seguir, a não ser que o outro ganhe antes por outra situação. Sendo assim, há que estudar em todos os tabuleiros rodados, quais as possibilidades de vitória dos dois jogadores. O próximo a jogar apenas precisa que um dos rodados lhe dê vitória. Enquanto que o outro precisa que um dos rodados lhe dê vitória e nenhum dê ao seu adversário.

Outra mudança significativa tem a ver com possibilidades de vitória, ainda que não seja certa. Isto é, com situações em que um dos jogadores ganha, a não ser que o outro o impeça. Nesses casos o tabuleiro é classificado com os valores 50 ou -50 consoante é o jogador computador que tem a possibilidade de ganhar ou o seu adversário. Mais uma vez, isto é estudado tendo em conta todos os rodados e quem é o próximo a jogar.

Em vários aspetos, esta heurística efetua correções à heurística A, isto é, calcula o valor da forma que realmente se pretendia fazer em A, mas que pela complexidade da heurística não foi inicialmente conseguido. No entanto, como a performance da A por vezes é melhor que a da A star, optamos por manter as duas heurísticas no jogo.

Usando profundidade 4, a profundidade máxima que se pode usar nas nossas máquinas que ainda obtém uma reposta imediata do ponto de vista do utilizador, esta é a única heurística que com os valores das forças relativas default consegue precaver-se contra uma tática ofensiva específica no Pentago. Ao controlar o centro dos quadrantes e atacar pelas diagonais as outras heurísticas perdem se não usarem profundidades mais elevadas.

3 Desenvolvimento

3.1 Ferramentas utilizadas

3.1.1 Geral

Para partilha de documentação, código e backups de segurança foi usado o Github. O repositório está disponível em https://github.com/bmad4ever/IART.git.

Foi usado o wxMáxima para realização de cálculos diversos.

Para a elaboração de documentação e do relatório foram usados o Draw.io e o Overleaf.

3.1.2 Protótipo Inicial em Consola

Foi usado o Eclipse Mars na implementação do protótipo inicial do jogo em Java. Este protótipo apenas funcionava em consola e ainda apresentava algumas inconsistência. Terá sido exportado para C# para que se pudesse numa fase posterior integrar o projeto com o Unity3d.

3.1.3 Código Core

O código relativo ao Pentago e á inteligência artificial foram desenvolvidos na linguagem de programação C#, no Visual Studio IDE para Windows OS. Como um dos elementos usou MAC OS X foi usado também o Virtual Box para trabalhar em Windows através de Máquina Virtual. Alternativamente poderia ter sido utilizado o Monodevlop, também compatível com MAC OS X. A possibilidade de utilizar *Partial Classes* em C# ofereceu alguma flexibilidade adicional aos elementos do grupo ao permitir trabalhar nos mesmos módulos sem que ocorram grandes conflitos nas diferentes versões do código.

3.1.4 Jogo com Interface Gráfica

Para a criar o jogo Pentago com uma interface gráfica apelativa foi utilizado o Unity3d. Todo o código, excluindo *enhancements* visuais foi criado de raiz, sendo integrado com os módulos do projeto em consola desenvolvido no Visual Studio. Foi também usado o Blender. na criação dos modelos 3d para interface gráfico do jogo.

3.1.5 Testes de Performance temporal e de desempenho das Heurísticas

Os dados que apresentamos neste relatório demoraram um tempo considerável a ser recolhidos, desde um dia no caso dos testes para avaliar o desempenho das heurísticas, a aproximadamente 3 no caso dos testes de performance temporal. Assim, para a realização destes, foram usados dois desktops controlados remotamente. Para a maioria dos testes foi utilizado um desktop com os seguintes características:

- Windows 10 SO, 64 bits
- Processador Pentium(R) Dual-Core CPU E5500 @ 2.8GHz 2.8GHz
- Ram instalada 4GB(3.65GB utilizável)

Apenas nos testes de performance temporal com 100 tabuleiros foi usado um desktop com as seguintes características:

- Windows 8 SO, 64 bits
- Processador Pentium(R) Dual-Core CPU E5700 @ 3GHz 3GHz
- Ram instalada 2GB

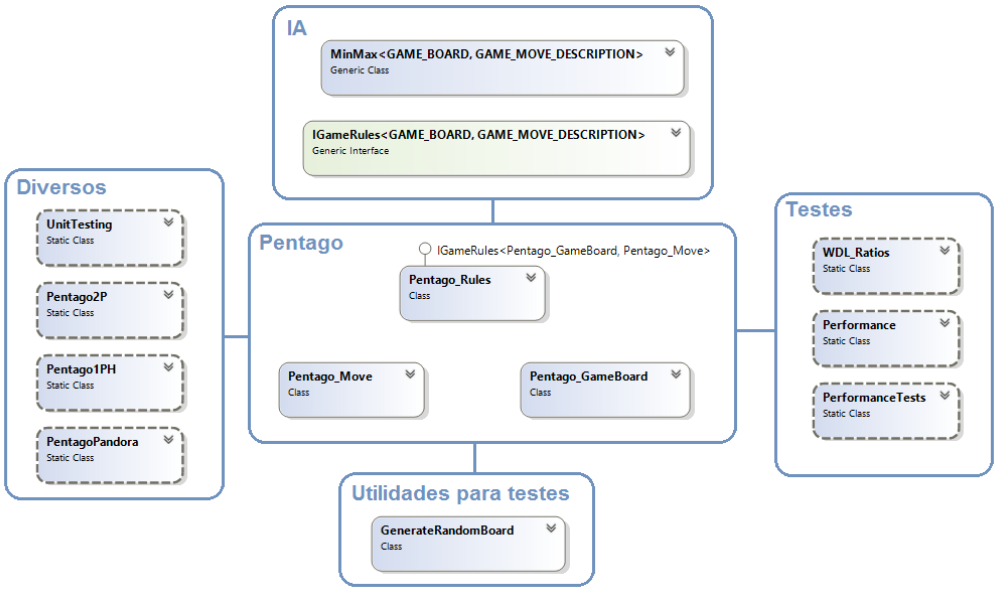
3.2 Minmax Genérico

Para que o Minmax pudesse ser reutilizado em outros jogos foi implementada a classe Genérica MinMax e o Interface IGameRules, O IGameRules estabelece um conjunto de métodos que devem ser implementados de modo a que se possa correr o Minmax sobre um jogo especifico.Estes fazem uso de duas estruturas de dados uma para representação do estado do jogo e outra para representação de jogadas que devem ser fornecidas por quem quiser fazer uso deste Minmax genérico. De modo a permitir as sub-jogadas presentes no Pentago e também qualquer conjunto de sub-jogadas possível em outros jogos, em vez de alternarmos entre nodos *Min* e *Max*. obrigamos, através da Interface IGameRules, a implementação de um método que diz qual o nodo a seguir consoante o estado do tabuleiro. Isto obrigado que as representação do estado de jogo permitam identificar o tipo nodo, seja de forma direta ou indireta. No caso da nossa implementação do Pentago esta identificação é direta, guardando de quem é o turno na estrutura que representa o tabuleiro. Alternativamente, poderia ser indireta contando o número de peças de cada jogador no tabuleiro, o que não foi usado devido ao óbvio decréscimo na eficiência temporal do Minmax. Em consequência desta decisão de implementação seria necessário, em muitos jogos, ter sempre o turno na estrutura que representa o estado de jogo pois o tabuleiro só por si pode não ser suficiente para identificar o tipo de nodo.

As primeiras iterações relativas à primeira jogada são efetuadas de forma relativamente diferente do resto do algoritmo, guardando o conjunto de sub-jogadas que o jogador faz até terminar a sua jogada. Esta modificação foi implementada a pensar na componente de Interdace Gráfica do jogo. Alternativamente ao caminho seguido, seria também possível usar o Minmax apenas usando estados do jogo. Nesta alternativa, de modo a representar uma jogada na Interface Gráfica seria necessário ter uma função que obtivesse uma jogada (ou várias) analisando o tabuleiro inicial e final.

3.3 Estrutura do código

3.3.1 Diagrama de Classes



3.3.2 Descrição das Classes e Interfaces

- MinMax
  Contem a implementação genérica do algoritmo de Minmax e variantes.

- IGameRules
  Interface necessária à implementação de um Minmax genérico, não dependente do jogo que analisa. Uma implementação da interface deve conter duas estruturas de dados diferentes, uma para representação do ambiente de jogo (tabuleiro no caso do Pentago) e outra para representação das jogadas a efetuar sobre este. A Interface pode ser consultada no anexo .2.1.

- Pentago_GameBoard
  Contem o array que representa o tabuleiro. Contem também uma variável que indica o qual o jogador a jogar e ainda qual o estado da jogada (colocar peça ou rodar quadrante). Também contem métodos auxiliares para identificar fim de jogo, clonar tabuleiros, transformar índices do tabuleiro e coordenadas e virse-versa. A classe é necessária para fazer uso da interface implementada.

- Pentago_Move
  Contem informação necessária para representar uma jogada. Havendo dois tipos de jogadas poderia alternativamente optar-se pela a criação de 2 subclasses, uma para cada tipo de jogada ao invés de guardar as duas numa mesma. A classe é necessária para fazer uso da interface implementada.

- Pentago_Rules
  Classe que implementa a Interface IGameRules para o Pentago. Contem também as funções de heurística, métodos para identificação de simetrias e de tabuleiros idênticos. A classe é extensa e foi divida em vários ficheiros. Em conjunto com Pentago_Move e Pentago_GameBoard formam a implementação do jogo concreto.

- GenerateRandomBoard
  Como o nome sugere a classe é responsável pela geração de tabuleiros aleatórios que são usados nos testes de `performance`.

- WDL_Ratios
  Responsável por testar a `performance` das heurísticas e do Minmax em termos de taxa de vitórias, derrotas e empates e fazer output dos resultados já formatados para serem inseridos em Latex.

- Performance
  Classe que testa a `performance` temporal de uma heurística para um dado conjunto de tabuleiros.

- PerformanceTests
  Classe responsável por testar a `performance` temporal das várias heurísticas.

- PentagoPandora
  Classe cuja única função é descobrir e exportar para memória não volátil uma árvore de Minmax apenas para o primeiro jogador, com raiz no tabuleiro inicial, que permita garantir a vitória. Não tendo sido totalmente otimizada para efetuar o seu objetivo, acabou por ser deixada de lado uma vez que saia do âmbito do projeto.

- Pentago2P
  Classe que apenas implementa uma versão do Pentago para 2 jogadores em consola. Usada inicialmente para testar o código.

- Pentago1PH
  Apenas usada no inicio do desenvolvimento, antes de codificado o algoritmo de Minmax. Permitia a um jogador humano jogar contra um minmax com profundidade de 1 jogada.

- UnitTesting
  Classe com testes unitários.

4 Experiências

Foram realizados 2 tipos de experiências distintas. As primeiras experiências foram desenhadas para testar a qualidade das heurísticas implementadas. Isto é, para determinar se os jogadores computador que usam estas heurísticas conseguem atingir bons desempenhos a jogar Pentago. Nomeadamente, tendo duas heurísticas diferentes a jogar Pentago, interessa-nos saber qual ganha. O segundo grupo de experiências pretende determinar qual a eficiência dos algoritmos implementados. Em particular, queremos comparar estes algoritmos entre si.

4.1 Qualidade das Heurísticas

As estatísticas apresentadas nesta secção referem-se ao jogador destacado a negrito. Por exemplo, o número de vitórias refere-se a vitórias desse jogador. As vitórias, derrotas, empates e número médio de turnos são representados respetivamente pelas cores azul, vermelha, verde e preta. As três primeiras são apresentadas em permilagem. Todos valores originais foram arredondados para baixo pelo que o somatório das componentes pode não igualar 1000‰.

O jogador artificial que jogar com as brancas inicia a partida (ao contrário do Pentago original).

As informações apresentadas abaixo do nome das heurísticas referem-se aos pesos/opções usadas. Nas heurísticas 1 e 1.2 são sempre usados (e ocultados nas tabelas) os pesos `default` usados para os diferentes tipos de linhas. Isto porque consideramos que a sua inclusão nos testes não se justifica, dada a reduzida relevância para projeto. Muitas das combinações usadas foram descobertas ao testar, de forma automatizada, as heurísticas com valores pseudo-aleatórios (dentro de determinados intervalos).

Para cada cada heurística os dados apresentados referem-se a:

- 1 - `bias`
- 1.2 Relaxada? (Y=sim, N=não); Usa Hack Diagonal? (Y=sim, N=não); possibilidades ; possibilidades para oponente; possibilidades fortes; possibilidades fortes para oponente
- A - monica ; meio ; direito ; tripla
- A hacked - Pesos da 1.2 ; Pesos da A
- A star - Pesos da A ; Analisa Rotações? (Y=sim,N=não)

A primeira ocorrência de uma heurística numa tabela apresenta sempre os pesos `default` da mesma.

Além de testes automáticos foram realizadas também alguns jogos de humano contra as heurísticas implementadas. Para uma profundidade de 4 a heurística A e variantes normalmente conseguem realizar um jogo interessante contra um humano. Ao aumentar a profundidade o jogador humano tem que esperar algum tempo pela resposta, no entanto torna-se muito mais difícil vencer estas heurísticas.

4.1.1 Testes de Heurísticas versus Controlo

Cada teste consiste em 400 partidas com tabuleiro inicial vazio e 400 com tabuleiros aleatórios. Dos 400 tabuleiros aleatórios uma metade são com um número de peças ímpares (jogador com pretas a iniciar a primeira jogada) e a outra metade com um número de peças par (jogador com brancas a iniciar a primeira jogada).

Joga com Brancas	Joga com Pretas	Tabuleio Inicial Vazio				Tabuleiro Inicial Aleatorio				Total			
		Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur
control	control	627	330	42	25	660	297	42	12	643	313	42	18

Tabela 1: Heurística de Controlo versus Controlo usando Alfa-Beta com profundidade 4 (2 jogadas)

Joga com Brancas	Joga com Pretas	Tabuleio Inicial Vazio				Tabuleiro Inicial Aleatorio				Total			
		Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur
one 0,5	control	775	170	55	23	742	175	82	12	758	172	68	17
control	**one** 0,5	482	432	85	24	377	555	67	12	430	493	76	18
one 0	control	762	105	132	24	712	185	102	12	737	145	117	18
control	**one** 0	475	355	170	24	352	530	117	13	413	442	143	18
one 1	control	695	255	50	23	770	217	12	11	732	236	31	17
control	**one** 1	427	542	30	23	365	620	15	11	396	581	22	17
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	control	787	157	55	23	767	175	57	12	777	166	56	17
oneDotTwo R:Y ; DH:N ; 4 ; 3 ; 2 ; 8	control	790	112	97	24	760	152	87	13	775	132	92	18
oneDotTwo R:Y ; DH:N ; 3 ; 4 ; 8 ; 3	control	685	250	65	26	750	190	60	12	717	220	62	19
oneDotTwo R:Y ; DH:N ; 1 ; 5 ; 3 ; 10,1	control	805	97	97	24	732	185	82	12	768	141	90	18
oneDotTwo R:Y ; DH:N ; 5 ; 1 ; 8,1 ; 3	control	730	205	65	25	755	207	37	12	742	206	51	18
oneDotTwo R:Y ; DH:N ; 10 ; 10 ; 6 ; 4	control	795	170	35	23	727	192	80	12	761	181	57	17
control	**oneDotTwo** R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	650	300	50	22	412	520	67	12	531	410	58	17
control	**oneDotTwo** R:Y ; DH:N ; 4 ; 3 ; 2 ; 8	475	405	120	26	322	585	92	13	398	495	106	19
control	**oneDotTwo** R:Y ; DH:N ; 3 ; 4 ; 8 ; 3	835	155	10	18	480	477	42	11	657	316	26	14
control	**oneDotTwo** R:Y ; DH:N ; 1 ; 5 ; 3 ; 10,1	510	382	107	26	365	552	82	13	437	467	95	19
control	**oneDotTwo** R:Y ; DH:N ; 5 ; 1 ; 8,1 ; 3	482	495	22	22	370	590	40	12	426	542	31	17
control	**oneDotTwo** R:Y ; DH:N ; 10 ; 10 ; 6 ; 4	522	442	35	23	395	535	70	12	458	488	52	17

Tabela 2: Heurística '1' e variantes versus Controlo usando Alfa-Beta com profundidade 4 (2 jogadas)

Joga com Brancas	Joga com Pretas	Tabuleio Inicial Vazio				Tabuleiro Inicial Aleatorio				Total			
		Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur
A 1,13 ; 1,15 ; 1,17 ; 1,19	control	977	20	2	17	905	67	27	8	941	43	15	12
A 1,23308 ; 1,26326 ; 1,11807 ; 0,99693	control	997	2	0	13	912	67	20	6	955	35	10	9
A 0,946981 ; 1,120938 ; 1,126149 ; 1,23813	control	940	40	20	20	902	80	17	9	921	60	18	14
A 0,917492 ; 1,138074 ; 1,259564 ; 1,168299	control	950	35	15	21	895	80	25	10	922	57	20	15
control	**A** 1,13 ; 1,15 ; 1,17 ; 1,19	912	72	15	20	697	265	37	9	805	168	26	14
control	**A** 1,23308 ; 1,26326 ; 1,11807 ; 0,99693	992	5	2	14	717	252	30	7	855	128	16	10
control	**A** 0,946981 ; 1,120938 ; 1,126149 ; 1,23813	847	122	30	22	687	280	32	10	767	201	31	16
control	**A** 0,917492 ; 1,138074 ; 1,259564 ; 1,168299	775	170	55	23	612	335	52	11	693	252	53	17
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	control	985	7	7	11	897	67	35	6	941	37	21	8
Astar 1,23308 ; 1,26326 ; 1,11807 ; 0,99693 ; Y	control	1000	0	0	9	907	65	27	6	953	32	13	7
Astar 0,946981 ; 1,120938 ; 1,126149 ; 1,23813 ; Y	control	980	10	10	15	897	70	32	7	938	40	21	11
Astar 0,917492 ; 1,138074 ; 1,259564 ; 1,168299 ; Y	control	985	7	7	14	902	67	30	7	943	37	18	10
control	**Astar** 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	982	12	5	12	712	245	42	7	847	128	23	9
control	**Astar** 1,23308 ; 1,26326 ; 1,11807 ; 0,99693 ; Y	1000	0	0	11	742	220	37	7	871	110	18	9
control	**Astar** 0,946981 ; 1,120938 ; 1,126149 ; 1,23813 ; Y	920	52	27	18	705	232	62	8	812	142	45	13
control	**Astar** 0,917492 ; 1,138074 ; 1,259564 ; 1,168299 ; Y	922	45	32	16	660	265	75	9	791	155	53	12
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	control	975	15	10	16	887	70	42	8	931	42	26	12
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,23308 ; 1,26326 ; 1,11807 ; 0,99693	control	965	22	12	14	882	87	30	7	923	55	21	10
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 0,946981 ; 1,120938 ; 1,126149 ; 1,23813	control	950	35	15	18	895	80	25	9	922	57	20	13
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 0,917492 ; 1,138074 ; 1,259564 ; 1,168299	control	935	37	27	19	877	95	27	9	906	66	27	14
control	**AplusDiagHack** 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	912	60	27	18	745	227	27	9	828	143	27	13
control	**AplusDiagHack** 0 ; 0 ; 1 ; 2 ; 1,23308 ; 1,26326 ; 1,11807 ; 0,99693	975	20	5	13	720	260	20	7	847	140	12	10
control	**AplusDiagHack** 0 ; 0 ; 1 ; 2 ; 0,946981 ; 1,120938 ; 1,126149 ; 1,23813	880	80	40	20	645	310	45	9	762	195	42	14
control	**AplusDiagHack** 0 ; 0 ; 1 ; 2 ; 0,917492 ; 1,138074 ; 1,259564 ; 1,168299	780	172	47	23	672	292	35	10	726	232	41	16

Tabela 3: Heurística A e variantes versus Controlo usando Alfa-Beta com profundidade 4 (2 jogadas)

4.1.2 Testes de Heurísticas versus Heurísticas

Cada teste consiste em 200 partidas com tabuleiro inicial vazio e 200 com tabuleiros aleatórios. Dos 200 tabuleiros aleatórios uma metade são com um número de peças ímpares (jogador com pretas a iniciar a primeira jogada) e a outra metade com um número de peças par (jogador com brancas a iniciar a primeira jogada).

Joga com Brancas	Joga com Pretas	Tabuleio Inicial Vazio				Tabuleiro Inicial Aleatorio				Total			
		Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur
A 1,13 ; 1,15 ; 1,17 ; 1,19	A 1,13 ; 1,15 ; 1,17 ; 1,19	420	535	45	21	540	385	75	9	480	460	60	15
A 1,13 ; 1,15 ; 1,17 ; 1,19	Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	670	325	5	21	405	520	75	8	537	422	40	14
A 1,13 ; 1,15 ; 1,17 ; 1,19	one 0,5	900	55	45	20	745	180	75	9	822	117	60	14
A 1,13 ; 1,15 ; 1,17 ; 1,19	oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	815	120	65	22	765	185	50	9	790	152	57	15
A 1,13 ; 1,15 ; 1,17 ; 1,19	AplusDiagHack 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	0	990	10	16	500	415	85	9	250	702	47	12
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	A 1,13 ; 1,15 ; 1,17 ; 1,19	425	530	45	24	600	320	80	7	512	425	62	15
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	915	60	25	11	530	390	80	8	722	225	52	9
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	one 0,5	970	0	30	16	790	165	45	7	880	82	37	11
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	615	370	15	14	790	165	45	7	702	267	30	10
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	AplusDiagHack 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	685	315	0	14	605	300	95	7	645	307	47	10
one 0,5	A 1,13 ; 1,15 ; 1,17 ; 1,19	250	650	100	22	290	655	55	9	270	652	77	15
one 0,5	Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	5	995	0	12	160	790	50	7	82	892	25	9
one 0,5	one 0,5	460	165	375	29	475	335	190	14	467	250	282	21
one 0,5	oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	445	225	330	29	475	350	175	14	460	287	252	21
one 0,5	AplusDiagHack 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	345	600	55	18	235	695	70	9	290	647	62	13
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	A 1,13 ; 1,15 ; 1,17 ; 1,19	140	820	40	22	240	675	85	10	190	747	62	16
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	0	1000	0	13	185	760	55	7	92	880	27	10
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	one 0,5	440	150	410	30	480	305	215	14	460	227	312	22
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	400	170	430	30	480	355	165	14	440	262	297	22
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	AplusDiagHack 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	225	665	110	20	245	685	70	9	235	675	90	14
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	A 1,13 ; 1,15 ; 1,17 ; 1,19	740	215	45	19	485	405	110	9	612	310	77	14
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	175	765	60	18	390	565	45	8	282	665	52	13
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	one 0,5	960	25	15	17	785	180	35	7	872	102	25	12
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	925	30	45	17	730	235	35	8	827	132	40	12
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	AplusDiagHack 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	455	310	235	23	500	415	85	8	477	362	160	15

Tabela 4: Heurísticas contra Heurísticas usando Alfa-Beta com profundidade 4(2jogadas) e os pesos default

Joga com Brancas	Joga com Pretas	Tabuleio Inicial Vazio				Tabuleiro Inicial Aleatorio				Total			
		Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur	Vit‰	Der‰	Emp‰	Tur
oneDotTwo R:N ; DH:N ; 1 ; 1 ; 1 ; 1	control	885	85	30	23	720	220	60	12	802	152	45	17
oneDotTwo R:N ; DH:N ; 1 ; 1 ; 1 ; 1	A 1,13 ; 1,15 ; 1,17 ; 1,19	210	710	80	22	295	625	80	10	252	667	80	16
oneDotTwo R:N ; DH:N ; 1 ; 1 ; 1 ; 1	Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	5	995	0	11	205	740	55	8	105	867	27	9
oneDotTwo R:N ; DH:N ; 1 ; 1 ; 1 ; 1	one 0,5	350	170	480	30	495	280	225	14	422	225	352	22
oneDotTwo R:N ; DH:N ; 1 ; 1 ; 1 ; 1	oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	385	210	405	29	480	360	160	14	432	285	282	21
oneDotTwo R:N ; DH:N ; 1 ; 1 ; 1 ; 1	AplusDiagHack 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	105	805	90	19	270	685	45	9	187	745	67	14
control	**oneDotTwo** R:N ; DH:N ; 1 ; 1 ; 1 ; 1	450	475	75	24	460	460	80	12	455	467	77	18
A 1,13 ; 1,15 ; 1,17 ; 1,19	**oneDotTwo** R:N ; DH:N ; 1 ; 1 ; 1 ; 1	60	890	50	20	170	760	70	9	115	825	60	14
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	**oneDotTwo** R:N ; DH:N ; 1 ; 1 ; 1 ; 1	45	895	60	17	145	835	20	6	95	865	40	11
one 0,5	**oneDotTwo** R:N ; DH:N ; 1 ; 1 ; 1 ; 1	275	340	385	29	245	470	285	15	260	405	335	22
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	**oneDotTwo** R:N ; DH:N ; 1 ; 1 ; 1 ; 1	230	440	330	29	295	540	165	14	262	490	247	21
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	**oneDotTwo** R:N ; DH:N ; 1 ; 1 ; 1 ; 1	45	945	10	16	170	795	35	7	107	870	22	11
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	control	995	0	5	11	830	140	30	6	912	70	17	8
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	A 1,13 ; 1,15 ; 1,17 ; 1,19	485	500	15	23	665	275	60	7	575	387	37	15
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	920	45	35	11	555	390	55	7	737	217	45	9
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	one 0,5	965	10	25	16	810	165	25	7	887	87	25	11
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	650	335	15	15	765	170	65	7	707	252	40	11
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	AplusDiagHack 0 ; 0 ; 1 ; 2 ; 1,13 ; 1,15 ; 1,17 ; 1,19	690	310	0	14	650	305	45	7	670	307	22	10
control	**Astar** 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	955	35	10	12	780	175	45	7	867	105	27	9
A 1,13 ; 1,15 ; 1,17 ; 1,19	**Astar** 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	340	655	5	21	510	435	55	8	425	545	30	14
Astar 1,13 ; 1,15 ; 1,17 ; 1,19 ; Y	**Astar** 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	40	910	50	12	380	560	60	7	210	735	55	9
one 0,5	**Astar** 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	995	0	5	11	760	205	35	7	877	102	20	9
oneDotTwo R:Y ; DH:N ; 1 ; 1 ; 1 ; 1	**Astar** 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	1000	0	0	13	725	215	60	8	862	107	30	10
AplusDiagHack 0 ; 0 ; 1 ; 0 ; 1,13 ; 1,15 ; 1,17 ; 1,19	**Astar** 1,13 ; 1,15 ; 1,17 ; 1,19 ; N	695	235	70	19	535	385	80	8	615	310	75	13

Tabela 5: Heurística 1.2 não relaxada e A star sem rotações

4.2 Eficiência dos Algoritmos

Para testar a eficiência de cada um dos algoritmos, realizamos dois tipos de testes. Primeiro determinamos o tempo que cada algoritmo demora a efetuar uma jogada em função da nível de profundidade. Depois testamos o tempo que cada algoritmo demora a efetuar uma jogada em função do número de peças no tabuleiro.

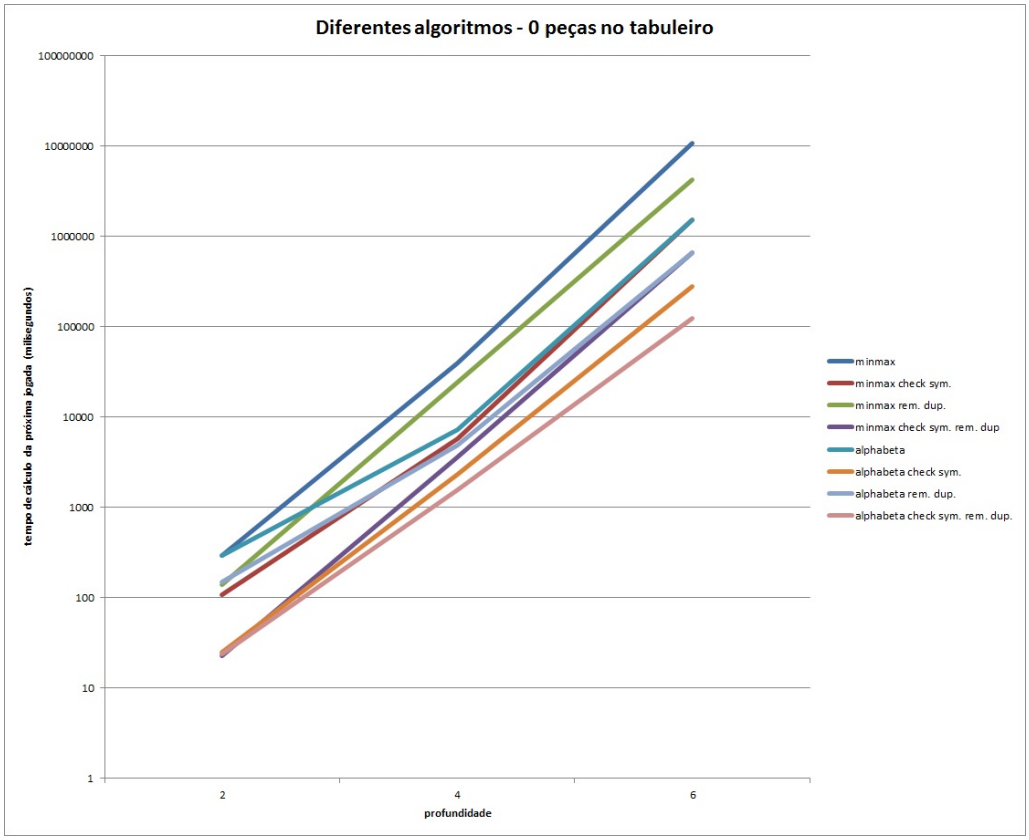
Cada teste individual realizado varia no algoritmo usado, profundidade usada e número de peças no tabuleiro. Para um só teste são iterados 100 tabuleiros aleatórios, os mesmos tabuleiros são usados para cada algoritmo para profundidade e número de peças iguais.

Os testes foram automatizados de modo a percorrerem todas as combinações de algoritmo, profundidade e número de peças delimitados. É apresentada nesta secção apenas uma versão integral dos resultados obtidos com a junção dos dados mais relevantes.

4.2.1 Variação do nível de profundidade

Estes testes foram efetuados com número de peças variável. No entanto, por questões de espaço, apenas apresentamos os resultados para a jogada inicial, isto é, quando o número de peças é 0. Dada a complexidade temporal dos algoritmos, apenas apresentamos profundidades até 6 (3 jogadas).

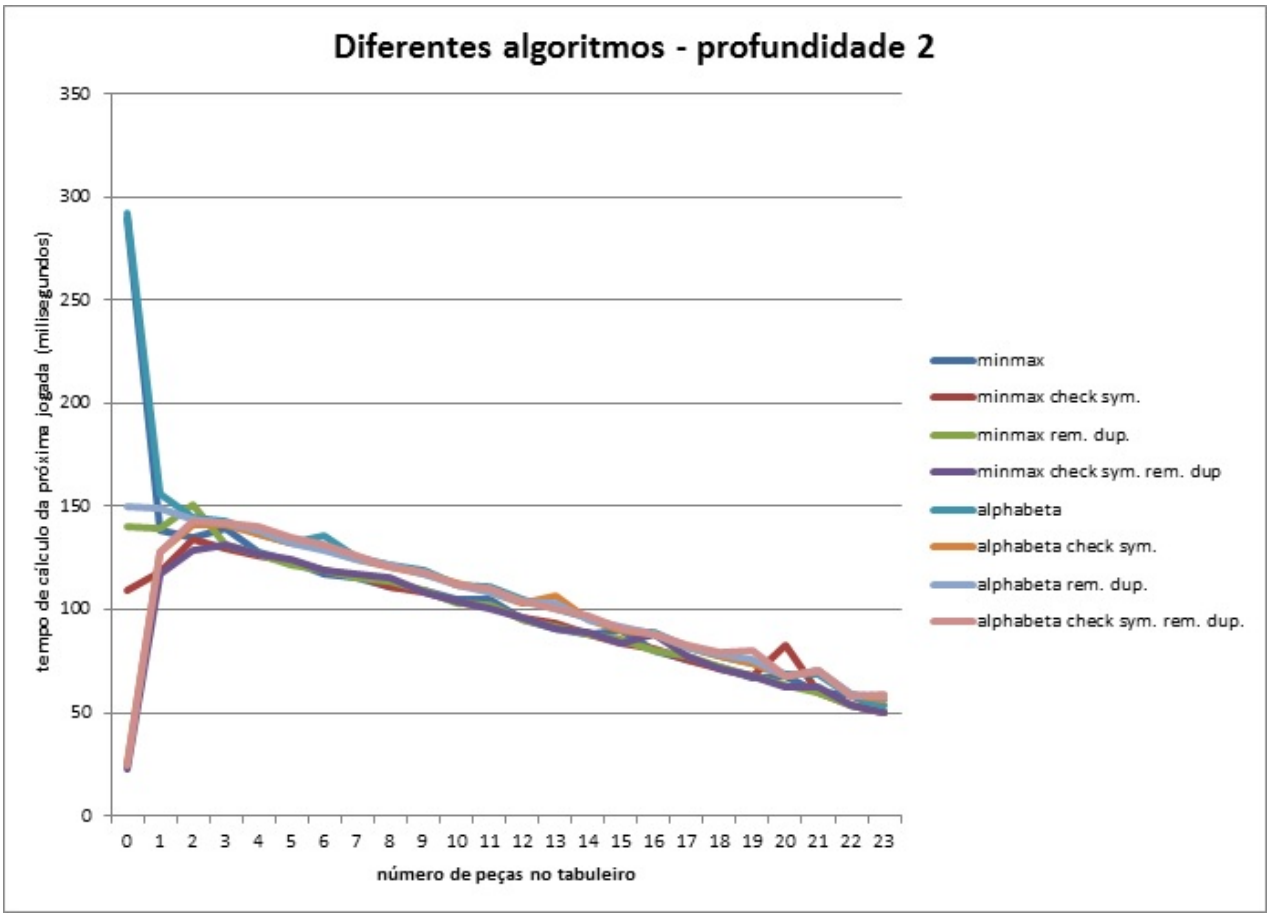
Relativamente à profundidade 8 e outras acima, não nos foi possível apresentar resultados devido ao tempo que os testes demoraram a completar. Uma semana, usando os testes automatizados para 100 tabuleiros, não foi suficiente para obter dados relativos aos algoritmos para uma profundidade de 8 (4 jogadas). Alterando o número de tabuleiros talvez fosse possível obter resultados a tempo, tal não foi realizado porque para o uso de simetrias, remoção de de duplicados e para a ordenação aleatória do alfa-beta a utilização de um número reduzido de testes poderia levar a resultados com menor fiabilidade.



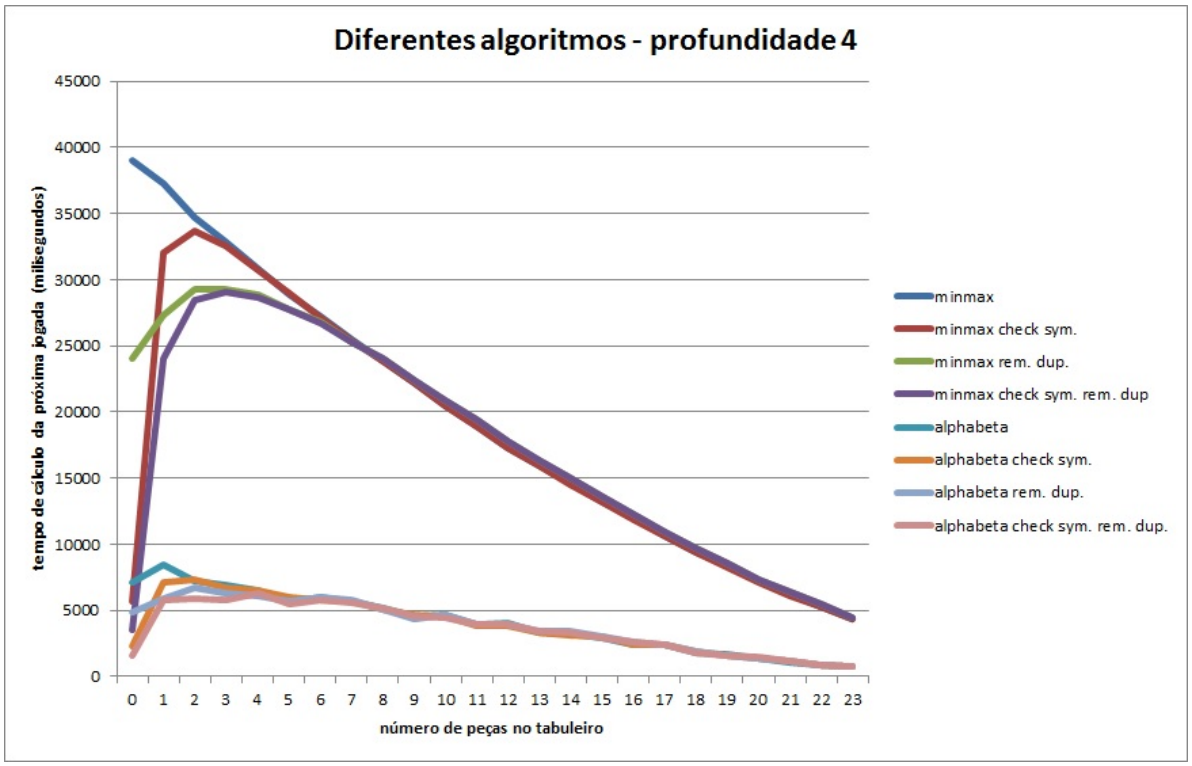
O gráfico tem escala logarítmica no eixo dos tempos, daí as retas aproximadas. Isto significa, como já era de esperar, que todos os algoritmos apresentam complexidade logarítmica. Também de esperar é a ordenação destes gráficos, sendo a melhor performance do alpha-beta com verificação de simetrias e remoção de duplicados e a pior do minmax sem qualquer melhoria.

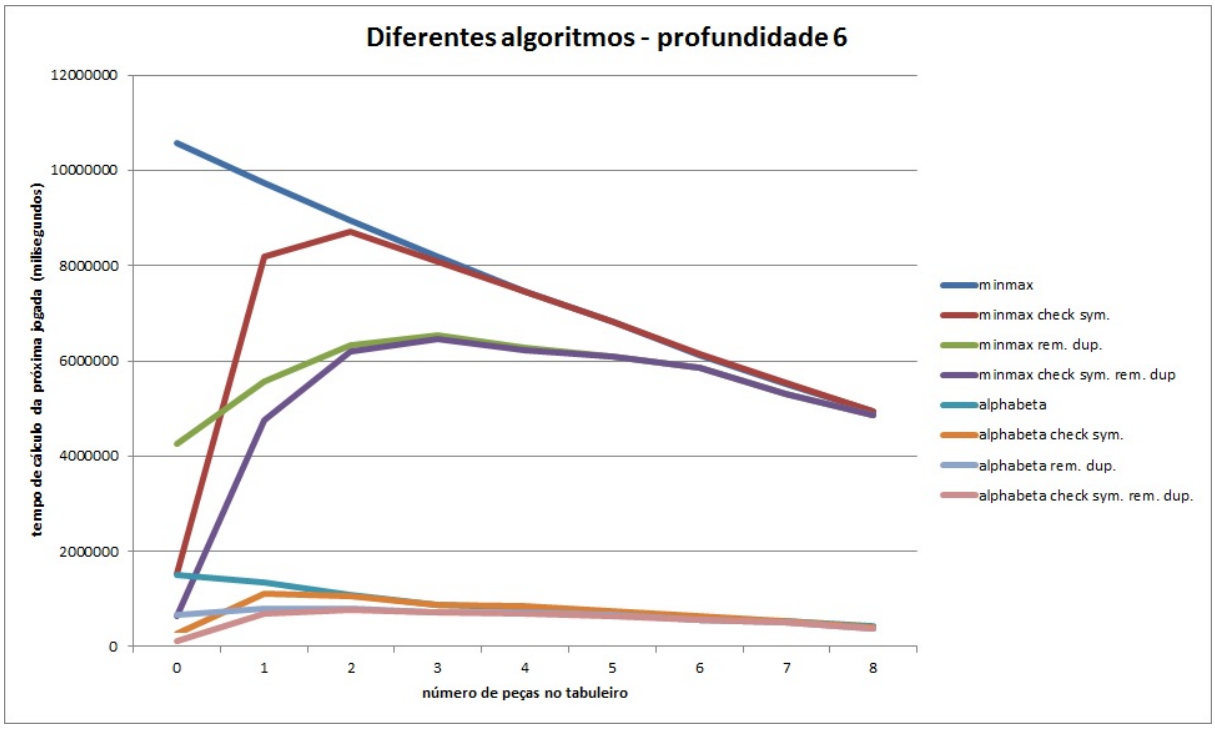
4.2.2 Variação do número de peças

A variação do número de peças é útil para perceber que se pode obter resultados imediatos com níveis de profundidade diferentes ao longo do jogo. Enquanto que inicialmente o nível de profundidade 4 é o melhor que conseguimos, ao fim de 8 peças, já se consegue praticamente o mesmo tempo com nível de profundidade 6. Além disso, estes testes também ajudam a perceber quando é útil verificar as simetrias e remover duplicados.



De reparar que a convergência total neste primeiro gráfico se deve ao facto de profundidade 2 corresponder apenas a uma jogada, não existindo por isso cortes alfa-beta.


Diferentes algoritmos - profundidade 4
tempo de cálculo da próxima jogada (milisegundos)
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
número de peças no tabuleiro
minmax
minmax check sym.
minmax rem. dup.
minmax check sym. rem. dup
alphabeta
alphabeta check sym.
alphabeta rem. dup.
alphabeta check sym. rem. dup.


Diferentes algoritmos - profundidade 6
tempo de cálculo da próxima jogada (milisegundos)
12000000
10000000
8000000
6000000
4000000
2000000
0
0 1 2 3 4 5 6 7 8
número de peças no tabuleiro
minmax
minmax check sym.
minmax rem. dup.
minmax check sym. rem. dup
alphabeta
alphabeta check sym.
alphabeta rem. dup.
alphabeta check sym. rem. dup.

5 Conclusões

Quando se pretende implementar um jogador artificial que recorra a algoritmos minmax, a maior dificuldade será, em princípio, conseguir uma boa heurística para avaliar os tabuleiros de jogo. Essa foi também uma das maiores dificuldades que encontramos neste projeto. De facto, para cada heurística baseada numa ideia original, as primeiras tentativas apresentavam defeitos graves de performance contra humanos. Mesmo contra a heurística de controlo, que escolhe um valor aleatório para o tabuleiro, os resultados não eram muito animadores. Foi necessário observarmos vários jogos concretos para percebermos o que devíamos mudar nas nossas heurísticas. No final, consideramos que, neste aspeto, o projeto se encontra sólido, dado que a heurística A star apresenta um comportamento bastante satisfatório contra jogadores humanos. Esta também foi a heurística que melhor se portou em geral nos testes automáticos de performance.

Além das heurísticas, a outra grande parte deste tipo de projetos é a escolha das jogadas seguintes. Na sua versão mais ingénua, o minmax percorre todas as jogadas seguintes à procura da mais adequada. No entanto, isso muitas vezes pode ser melhorado. Não apenas no sentido de usar cortes alfa-beta ou cortes devidos a considerarmos um determinado resultado suficientemente bom. Mas no sentido de perceber sem análise dos descendentes que um determinado tabuleiro conduzirá aos mesmos resultados que outro previamente analisado. Esse é precisamente o caso do Pentago em que as simetrias desempenham um papel fundamental. Apesar de ser desde sempre óbvio que este era um tema importante para explorar, demorou algum tempo a perceber como o fazer. Estamos convencidos que o resultado final é bastante interessante, nomeadamente o facto de nem sempre compensar esta verificação de simetrias.

6 Futuras Melhorias

Relativamente à componente de IA existem algoritmos e abordagens não exploradas. A implementação de `iterative deepening` em vez da realização de uma pesquisa única seria um ponto simples de implementar na continuação do projeto. A implementação de `negaScout` e a ordenação de tabuleiros por uma ordem não aleatória baseada numa classificação de modo a facilitar a poda também seriam pontos interessantes a explorar.

Face à mini `framework` implementada, seria interessante completá-la refinando e generalizando o PentagoPandora de modo a que consiga criar a árvore Minmax para qualquer jogo e para qualquer jogador.

O interface gráfico e a implementação em Unity3d pode ser ainda melhorada em diversos pontos, seja na apresentação, seja no código implementado. Com o intuito de disponibilizar parte do projeto na Asset Store do Unity3d, todo grafismo presente não original deverá ser substituído e o código do projeto será revisto após a entrega do trabalho sendo que deverá ficar disponível para download antes de Outubro, caso passe nos testes de qualidade e não existam restrições legais que impeçam a publicação.

7 Recursos

7.1 Bibliografia

[1] Geoffrey Irving. *Perfect Pentago Github*. 4 Apr 2014? URL: http://arxiv.org/pdf/1404.0743.pdf.

[2] MindtwisterUSA. *Pentago - Basic Play, Rules and Strategy Guide*. URL: https://webdav.info.ucl.ac.be/webdav/ingi2261/ProblemSet3/PentagoRulesStrategy.pdf.

[3] *Pentago Wiki*. URL: https://en.wikipedia.org/wiki/Pentago.

[4] *Perfect Pentago*. URL: https://perfect-pentago.net/.

[5] *Zobrist Hashing*. URL: https://chessprogramming.wikispaces.com/Zobrist+Hashing.

7.2 Lista de Software e Serviços usados

- Blender
- Eclipse Mars
- Github
- Overleaf
- Logmein
- Skype
- Unity3d
- Virtual Box
- Visual Studio
- WxMáxima

7.3 Balanceamento da carga de trabalho

- Ângela Cardoso
- Bruno Madeira %
- Francisco Veiga

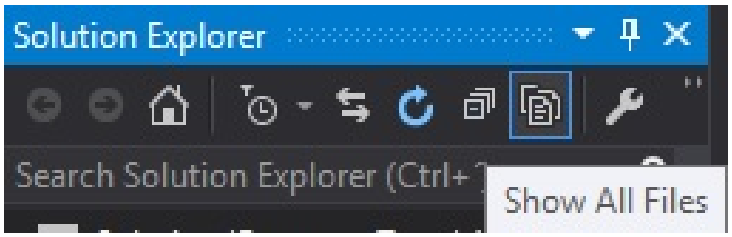
Appendices

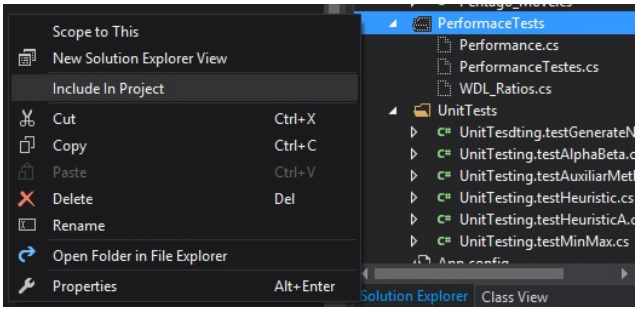
.1 Manual de utilização

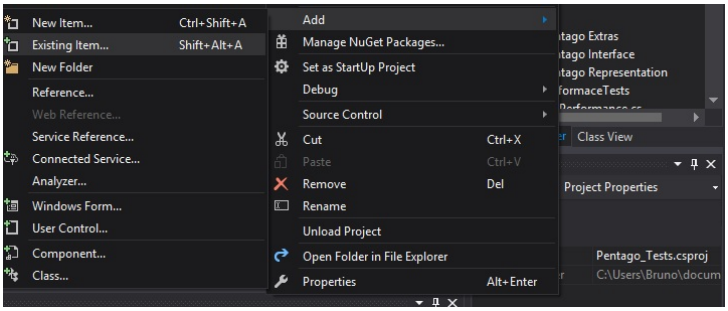
.1.1 Versão Consola

Na pasta "CODIGO SOURCE" está disponível quase todo o código desenvolvido para o projeto. Esta não inclui código para a interface gráfica nem testes manuais realizados pela equipa. No ficheiro Program.cs já se encontram duas funções relativas às experiências efetuadas comentadas. Para corre-las é preciso descomentá-las e compilar o código. Para especificar mais concretamente os testes a correr em vez de apenas usar estas linhas de código presentes no Program.cs ou para fazer uso de outras funcionalidades implementadas é necessário estar familiarizado com a `framework` implementada.

Se desejar compilar e correr esta versão do código pode usar uma versão do `Visual Studio` que suporte C#. Pode criar um novo projeto para consola em C# e arrastar os ficheiros manualmente para a pasta de projeto. Depois de arrastados é necessário inclui-los no projeto criado a partir do `Solution Explorer`. As imagens seguintes ilustram como pode realizar esta última etapa. Caso pretenda usar a ferramenta sugerida pode consultar o MSDN em caso de dúvidas. Alternativamente também pode optar por usar o Mono.





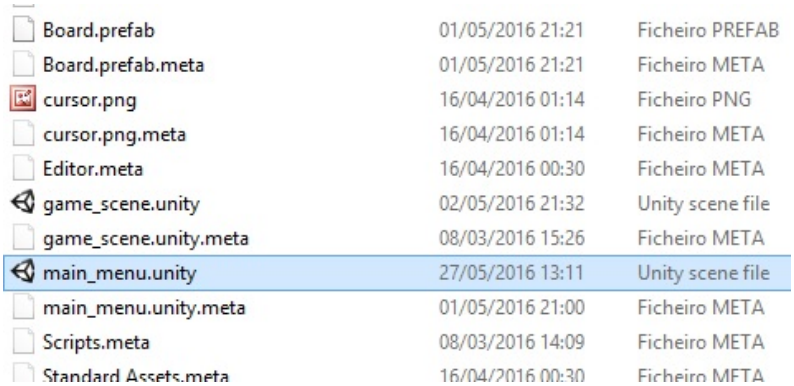


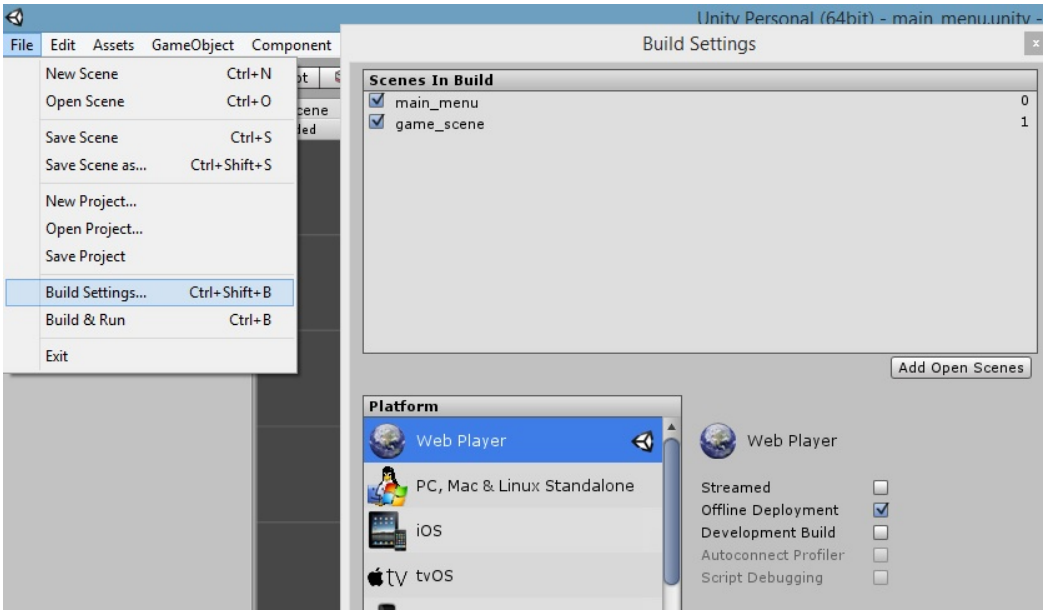
.1.2 Versão Unity

No "PentagoLastBuildAndSource.rar" contem uma pasta denominada "PENTAGO_ALL" onde encontram-se outras duas pastas.

Uma pasta denominada de "Pentago_LastBuild" contem o projeto já compilado para correr em browser independentemente do sistema operativo usado. Para correr esta versão do jogo com interface gráfica deve usar um browser que tenha instalado o plugin do Unity.

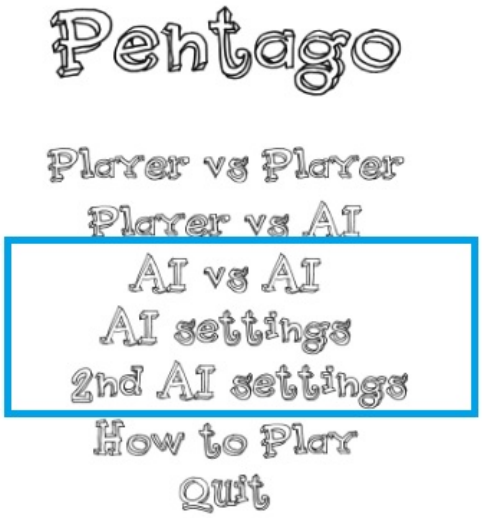
Uma outra pasta denominada de Pentago contem o projeto Unity com todo o código, modelos e conteúdo usado no desenvolvimento. O código desenvolvido especificamente para esta versão encontra-se dentro de "Assets Scripts". Além das componentes ligadas ao interface gráfico também é possível reparar em mudanças mínimas no código original. Caso pretenda editar ou compilar necessita de uma versão do Unity compatível com o mesmo (em principio todas as versões 5.x deverão funcionar). Para abrir o projeto abra uma `scene` do mesmo e para compilar é necessário que as `scenes` estejam incluídas nos `Build Settings`. As figuras seguintes ilustram estas etapas. Note que pode optar por compilar para um sistema operativo em vez de browser.





.1.3 Notas relativas ao Jogo

O jogo presenta uma naveção intuitiva e é bastante auto-explicativo. O único detalhe que talvez seja importante referir é que quando é usado o modo "IA vs IA" as "`IA setings`" são sempre aplicadas às IA de peças brancas e as de "`2nd IA`" à IA de peças pretas. Quando se joga no modo humano contra computador, independentemente das peças com que o jogador humano joga, o computador utiliza sempre as configurações definidas em `IA setings`. Este detalhe não está explicito no interface gráfico e será adicionado posteriormente à entrega para esclarecimento dos utilizadores.



Detalhes relativos a como jogar o jogo estão descritos no próprio interface gráfico na secção "`How to Play`".

.2 Code

.2.1 IGameRules

```
/// <summary>
/// Interface that defines all the methods that must be implemented in order to
    use Minimaz Algorithm
/// </summary>
/// <typeparam name="GAME_BOARD">Class with the representation of the game's
   board</typeparam>
/// <typeparam name="GAME_MOVE_DESCRIPTION">Class with that is used to
   represent the game's player allowed moves/plays</typeparam>
public interface IGameRules<GAME_BOARD,GAME_MOVE_DESCRIPTION> {

    GAME_MOVE_DESCRIPTION[] possible_plays(GAME_BOARD gb, int depth = 0);

    GAME_BOARD board_after_play(GAME_BOARD gb, GAME_MOVE_DESCRIPTION gmd);

    GAME_BOARD[] next_states(GAME_BOARD gb,int depth=0);

    /// <summary>
    /// checks if the game has ended.
    /// <para>returns null if game did not end or the value of utility if game
       ended</para>para>
    /// </summary>
    /// <param name="gb"></param>
    /// <param name="depth">optional usage, only usefull to calculate utility</
       param>
    /// <returns>null if game did not end or the value of utility if game ended
       </returns>
    float? game_over(GAME_BOARD gb,int depth);

    /// <summary>
    /// method to be used when minimax achieves max depth
    /// <para>may have a heuristics 'selector' in the implementation </para>
    /// <para>may return Float.POSITIVE_INFINITY or Float.NEGATIVE_INFINITY to
       force prioritization of certain plays</para>
    /// </summary>
    /// <param name="gb"></param>
    /// <returns></returns>
    float evaluate(GAME_BOARD gb);

    /// <summary>
    /// Should select the next node type to use in minimax.
    /// <para>Default behaviour for most games should be return !
       currentIterationNode.</para>
    /// <para>Some Games, like pentago, need some additional consierations due
       to 'half step/play' endgames</para>
    /// </summary>
    /// <param name="gb"></param>
    /// <param name="currentIterationNode"></param>
    /// <returns></returns>
    bool selectMINMAX(GAME_BOARD thisnode_gb, bool currentIterationNode);

    /// <summary>
    /// decide what depth to use on minimax depending on the game board
    /// </summary>
    /// <param name="gb"></param>
    /// <returns></returns>
    int smart_depth(GAME_BOARD gb);
```

```
    /// <summary>
    /// get information about 'Rules' to print/display
    /// </summary>
    /// <param name=""></param>
    /// <returns></returns>
    string toDisplayString();
}
```