IPL instituto politécnico de leiria
escola superior de tecnologia e gestão

Inteligência Artificial

"SNAKE"

Rúben Lopes Nº. 2160852; Tiago Batista Nº. 2161353

Inteligência Artificial | 11/06/2018

Docentes: Carlos Grilo | José Carlos Ribeiro

Objetivos

Como projeto da cadeira de Inteligência Artificial foi fornecido um documento com os requerimentos para uma aplicação em java. A partir de um projeto base, completá-lo de forma a atingir os objetivos descritos no documento.

Partindo da ideia básica do famoso jogo *"Snake"* construir várias formas de o jogar através de controladores baseados em redes neuronais. Alguns dos principais objetivos são:

- Começar por construir controladores de jogo sem qualquer *inteligência*, simplesmente que faça o básico. Tanto jogar de forma aleatória como perseguir a comida. Estes seriam o *SnakeRandom* e o *SnakeAdHoc*. Que serviriam para referência/comparação mais tarde;
- Construir dois controladores diferentes baseados em redes neuronais;
- Possibilidade de ter mais que um controlador a correr ao mesmo tempo na grelha de jogo. Tanto dois controladores com o mesmo funcionamento como de controladores diferentes;
- Realização de testes para alguns tipos de algoritmos de forma a testar alguns aspetos como o efeito da variação do tamanho da população, o efeito da variação das probabilidades dos operadores genéticos utilizados, etc.

Snake Random e Ad-Hoc

Tendo como base um projeto realizado anteriormente na cadeira começamos por implementar o agente *RANDOM*. Sabendo os objetivos do jogo e uma grelha com tamanho fixo, não toroidal (sair de um lado e entrar do outro) foi simples introduzir o nosso controlador neste ambiente. Apenas a "cabeça" da cobra, e uma comida na grelha de jogo. Para ir até à comida é gerado um número aleatório e a partir dele damos uma direção ao agente.

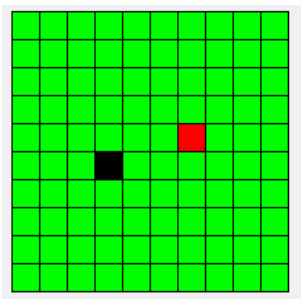

Figura 1 - Ad-Hoc campo jogo, comida (vermelho) e agente (preto)

Viria a ser de seguida implementado o mecanismo de "comer" de forma eficaz, ou seja, o agente *AD-HOC*. O agente passaria a saber a posição da comida e iria sempre até lá "guiado" por nós através de uma série de condições no código.

Assim, ainda sem cauda o agente come sempre a comida até acabar o número de iterações.

```
            action = Action.SOUTH;
        }
    }
}

if (this.getCell().getColumn() == food.getColumn()) {
    if (this.getCell().getLine() > food.getLine()) {
        if (n != null &&  !n.hasTail() && !n.hasAgent() && !n.hasWall()) {
            action = Action.NORTH;
        } else if ( e != null &&  !e.hasTail() && !e.hasAgent() && !e.hasWall()) {
            action = Action.EAST;
        } else if (w != null &&  !w.hasTail() && !w.hasAgent() && !w.hasWall()) {
            action = Action.WEST;
        } else {
            action = Action.SOUTH;
        }
    } else if (this.getCell().getLine() < food.getLine()) {
        if (s != null &&  !s.hasTail() && !s.hasAgent() && !s.hasWall()) {
            action = Action.SOUTH;
        } else if (e != null &&  !e.hasTail() && !e.hasAgent() && !e.hasWall()) {
            action = Action.EAST;
        } else if (w != null &&  !w.hasTail() && !w.hasAgent() &&  !w.hasWall()) {
            action = Action.WEST;
        } else {
            action = Action.NORTH;
        }
    }
}
```

Figura 2 - Excerto de código Ad-Hoc

Sempre que é encontrado algum obstáculo, o agente é desviado para outra direção.

O próximo passo seria então criar a cauda da cobra. Funcionando como um aumento de uma célula cada vez que é comida uma "comida". O agente continuaria a desviar-se dos obstáculos, mas agora a própria cauda também conta como obstáculo.

Feito isto teríamos o nosso agente básico implementado com todos os requisitos funcionais para este agente. A forma como funcionaria a cauda é de uma maneira simples. Da mesma forma como se move a cabeça, mover a última cauda para a posição onde a cabeça está antes de se move para a próxima célula, sendo essa ação feita primeiro.

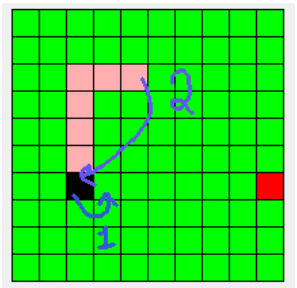

Figura 3 – Movimento da Cauda

Algoritmo genético aplicado ao contexto do projeto

Para criar este agente já nos foi fornecido no projeto base os elementos base e incompletos à implementação do agente. O algoritmo de aprendizagem de inteligência artificial dado é um algoritmo genético que vamos passar a explicar abaixo no contexto do projeto.

O algoritmo é constituído de 5 fases:

1. População inicial
2. Função de *Fitness*
3. Processo de seleção
4. Recombinação
5. Mutação

1. POPULAÇÃO INICIAL

Nesta primeira fase é gerado um conjunto de indivíduos a que chamamos população. Cada individuo é uma resposta ao nosso problema.

Cada um desses indivíduos é caracterizado por um conjunto de parâmetros ou genes.

Esses genes formam um cromossoma, ou o que queremos aqui, a solução.

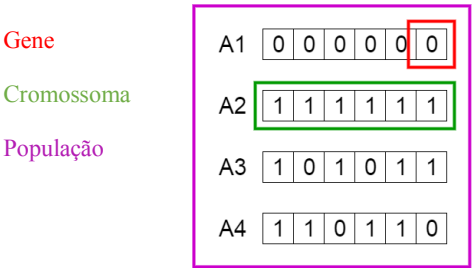

Figura 4 – Visualização de uma população

No contexto do nosso problema a população é um conjunto de indivíduos que tem um genoma. Este é inicializado com valores aleatórios.

2. FUNÇÃO DE FITNESS

A função de fitness serve para avaliar um determinado individuo, ou seja, a capacidade que tem para "comer" ou "andar" na grelha sem perder. É atribuído um fitness a cada individuo. Usamos dois tipos de fitness para avaliar os agentes, quando tínhamos um só agente a avaliação era só contabilizado o número de comidas e de movimentos, sendo sempre valorizado, mais o número de comidas, no entanto quando havia dois agentes tivemos de mudar a função de fitness pois havia um agente que ficava só no canto e não comia, passando essa tarefa só a ser feita por outro agente, logo este efeito foi combatido com uma penalização ao fitness se um dos agentes tivesse uma performance melhor que a outra (comer mais).

```
fitness = ((food * 500) + (movements * 0.2))
```

Figura 5 – Função fitness sem penalização

```
penalty = Math.abs(snake1Food - snake2Food) * 500;
fitness = ((food * 500) + (movements * 0.2)) - penalty;
```

Figura 6 – Função fitness com penalização

3. PROCESSO DE SELEÇÃO

Nesta fase a ideia é selecionar os melhores indivíduos e passar esses genes para uma próxima geração. Indivíduos com melhor fitness têm uma maior probabilidade de serem escolhidos. No nosso problema foram usados dois métodos, o *tournament* e o *roulette*.

4. Recombinação

No ponto de recombinação são escolhidos pontos aleatórios de entre o cromossoma de dois indivíduos e são trocados os genes criando indivíduos diferentes para a população.

No projeto temos 3 métodos de recombinação, *One Cut*, *Two Cuts* ou *Uniform* e uma probabilidade para tal acontecer.

5. MUTAÇÃO

Dos indivíduos alguns irão estar sujeitos a uma mutação, isto é, uma probabilidade baixa que se pode alterar na aplicação que altera os genes de um indivíduo selecionado. É feito para que seja criada uma certa diversidade na população e que não comecem todos os indivíduos a serem muito semelhantes uns aos outros. Nós usamos uma mutação Random para esta probabilidade.

```
for (int i = 0; i < ind.getNumGenes(); i++) {
    if (GeneticAlgorithm.random.nextDouble() < probability) {
        ind.setGene(i,  newValue: GeneticAlgorithm.random.nextDouble()*2-1);
    }
```

Figura 7 – Atribuição de mutações

O algoritmo é terminado quando terminarem as gerações definidas na aplicação ainda que chegue a um certo ponto em que a população já não evolui mais. É nesse momento que temos uma solução para o problema, neste contexto um individuo com um conjunto de genes que permitem "comer" mais e ficar mais tempo no campo de jogo sem perder.

SNAKE A.I.

O primeiro agente que implementamos denominamos de "Snake A.I.", este agente assenta num controlador baseado em redes neuronais com 11 *inputs*, 8 *hidden layers* e 4 outputs.

Os inputs do agente são valores que indicam se este têm comida, cauda ou parede (limite da grelha) a norte, sul, este ou oeste. São baseados em três valores -1, 0 ou 1, onde os primeiros quatro inputs são para informar se é possível o agente andar para alguma posição adjacente, os outros quatro permitem saber se a comida está numa comida adjacente e os últimos dois permitem saber a direção da comida mesmo esta estando inalcançável de momento.

Figura 8 – Exemplo de inputs (Snake A.I.)

Estes valores são depois são inseridos na rede neuronal mais propriamente na função *forwardPropagation* até termos um resultado final que no nosso caso é um *array* com tamanho de 4. Seguidamente o agente usa a sua função de decide para escolher a ação que vai executar, não é garantido que o agente faça a melhor decisão, desse modo o agente tem de ser treinado para ele poder ajustar os seus pesos e poder obter uma melhor performance.

Depois de implementadas todas as funcionalidades acima descritas, passámos para a realização de experiências, que consistem na simulação do agente. Foram testados os parâmetros acima referidos com diversos valores para podermos obter o melhor agente possível e menor custo computacional.

Para todos os agentes, seguimos o mesmo método de testes, começando com o primeiro parâmetro com diversos valores e todos os outros fixos. No fim da primeira ronda de experiências já sabemos qual o "melhor" valor e passamos a testar o próximo já com o anterior definido e assim sucessivamente.

EXPERIÊNCIAS COM SNAKE A.I.

Na primeira experiência realizada conseguimos perceber que a população inicial de indivíduos afeta a capacidade do agente, visto que a diferença entre o menor e maior resultado é de cerca de 20%, decidimos continuar os testes com uma população inicial de 200 indivíduos.

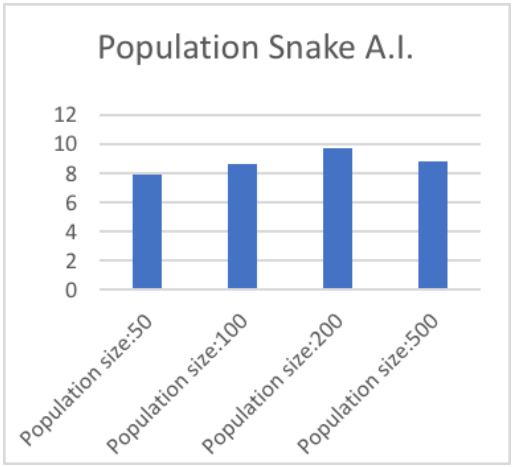

Figura 9 – Testes com população

De seguida, testámos a influência que o número de gerações tem no agente e verificámos que o Snake A.I. tem melhores resultados com gerações de 500 ou mais indivíduos, no entanto não apresentando melhorias para gerações mais altas, por isso decidimos continuar os testes com 500 gerações pois é o valor que apresenta melhor custo benefício.

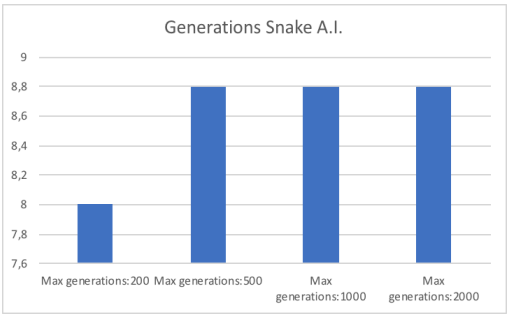

Figura 10 – Testes com gerações

O torneio teve uma melhoria com valores mais altos, de cerca de 20%, por isso escolhemos usar um torneio de 32 indivíduos nos restantes testes.

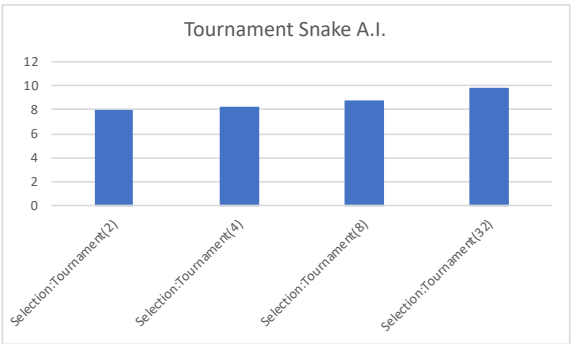

Figura 11 – Testes com torneios

Na recombinação escolhemos o método uniforme, pois foi o que apresentou melhores resultados, cerca de 15% a mais do que os outros. No teste da probabilidade de recombinação a

diferença entre as probabilidades testadas foi de cerca de 20%, tendo apresentado melhores resultados probabilidades de combinação mais baixa, visto isto continuamos o teste com o menor valor (0,25).

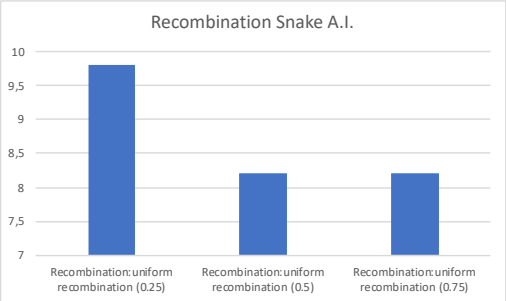

Figura 12 – Testes com recombinação

Por último, testámos a mutação e obtivemos resultados demasiado semelhantes, visto que probabilidades de mutação baixas (0,2) ou altas (0,8) obtiveram resultados idênticos e tendo os outros valores intermédios apresentado fitness mais baixas na casa dos 20%, assumimos que o valor mais baixo de mutação seria o ideal.

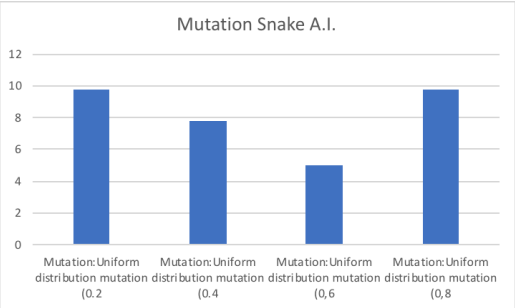

Figura 13 – Testes com mutações

SNAKE A.I. 2

Após implementado o primeiro agente, fomos implementar o segundo que denominamos de "Snake A.I.2", este agente tem por base um controlador diferente baseado em redes neuronais com 9 inputs, 8 hidden layers e 4 outputs.

O funcionamento deste agente é em tudo idêntico ao Snake A.I., a diferença está nos *inputs*, visto que juntamos os dois últimos *inputs* do agente anterior que verificavam desde a posição da comida à verificação se a comida está adjacente ao agente.

RESULTADOS COM SNAKE A.I. 2

O método de teste foi o mesmo de agente anterior.

O primeiro parâmetro a ser testado foi a população, o melhor resultado foi obtido com uma população inicial de 200 indivíduos á semelhança do que aconteceu com o agente anterior.

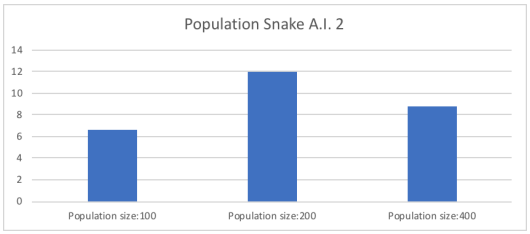

Figura 14 – Testes com populações para o outro controlador

De seguida, testámos as gerações, e ao contrário do que aconteceu com a Snake A.I. a variação do número de gerações na Snake A.I.2 não teve qualquer efeito no fitness do agente, para reduzir o custo computacional decidimos continuar os testes com 200 gerações de indivíduos.

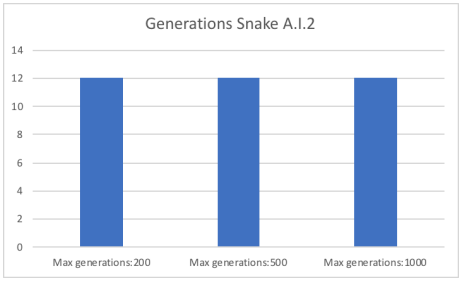

Figura 15 – Testes com gerações para o outro controlador

O torneio foi o próximo teste que realizámos e o torneio com tamanho 32 teve uma performance muito acima dos outros nomeadamente mais de 400% em relação ao menor valor, números muito significativos.

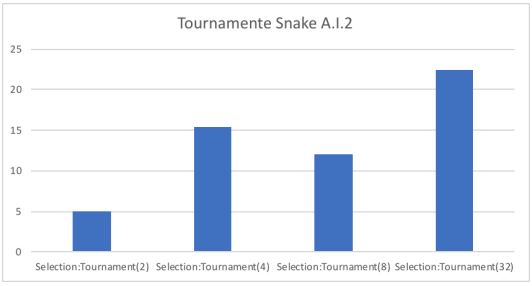

Figura 16 – Testes com torneios para o outro controlador

Na recombinação, tal como aconteceu com a Snake A.I. quanto menor a recombinação melhores foram os fitnesses da Snake A.I.2, tendo mesmo uma diferença de mais de 150% em relação ao menor valor.

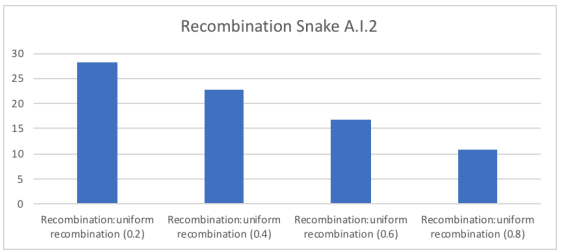

Figura 17 – Testes com recombinações para o outro controlador

Na mutação da Snake A.I.2, ao contrário do que aconteceu com a Snake A.I. quanto menor a mutação melhores foram os fitnesses da Snake A.I.2, tendo mesmo uma diferença de quase de 200% em relação ao menor valor.

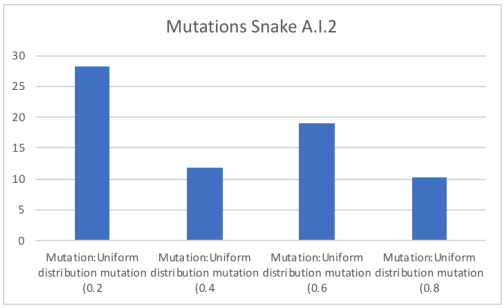

Figura 18 – Testes com mutações para o outro controlador

SNAKE HOMOGÉNEA

Após implementado dois tipos de agentes diferentes, foi-nos pedido que evoluíssemos um controlador onde juntaríamos dois "cérebros" de agentes iguais e denominámo-la de "Snake Coop"

Visto a Snake A.I.2 ter tido melhores resultados no geral decidimos escolhe-la. Tivemos, porém, de alterar a função de fitness dado que como teríamos dois agentes ambos teriam de se movimentar e comer logo a avaliação foi baseada nesse aspeto.

RESULTADOS HOMOGÉNEA

A população na "Snake Coop" teve bastante influência no fitness do agente, notando melhorias na casa dos 400%, sendo o crescimento mais notório quando temos uma população inicial de 400 indivíduos.

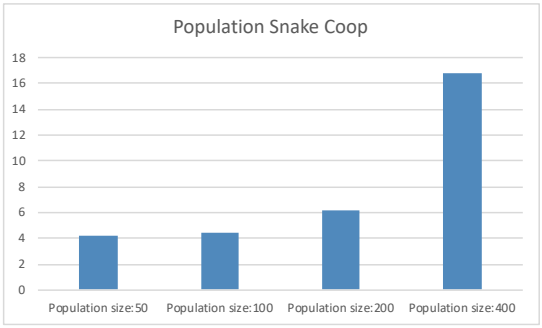

Figura 19 – Testes com populações para controladores homogéneos

A variação do número de gerações neste agente teve um efeito nulo, já que todas as gerações testadas tiveram fitness idênticos.

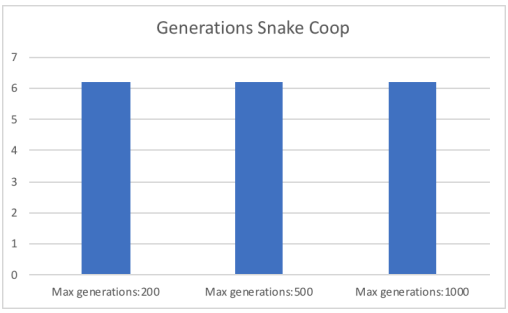

Figura 20 – Testes com gerações para controladores homogéneos

O torneio novamente teve efeitos muito bastante positivos no agente, tendo uma melhoria de mais de 200% no aumento do desempenho do agente.

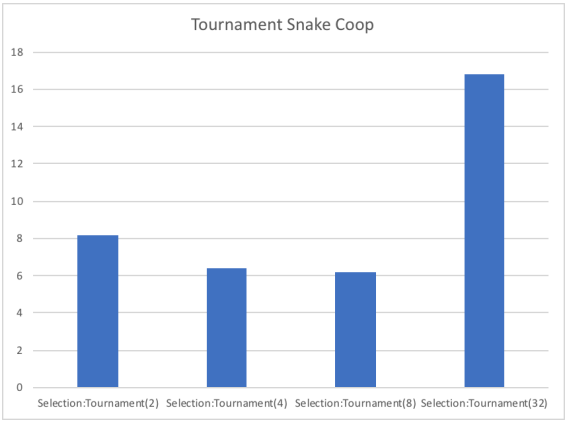

Figura 21 – Testes com torneios para controladores homogéneos

A recombinação neste agente apresenta melhores resultados quando mais elevada, ao contrário de quando temos só um agente.

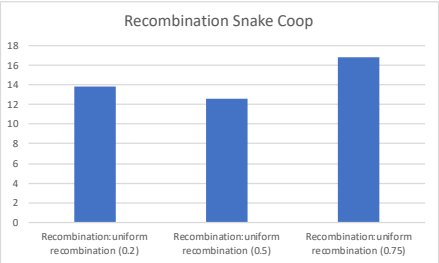

Figura 22 – Testes com recombinações para controladores homogéneos

A Snake Coop apresenta melhores resultados com uma mutação mais baixa o que segue a tendência dos outros agentes.

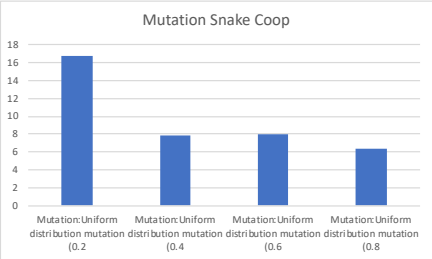

Figura 23 – Testes com mutações para controladores homogéneos

RESULTADOS HETEROGÉNEA

Para este agente baseado num controlador homogéneo, verificamos que a população tem pouca relevância visto que entre o maior e menor valor apenas verificamos uma diferença de 10% não sendo esta significativa e custando 4 vezes mais tempo.

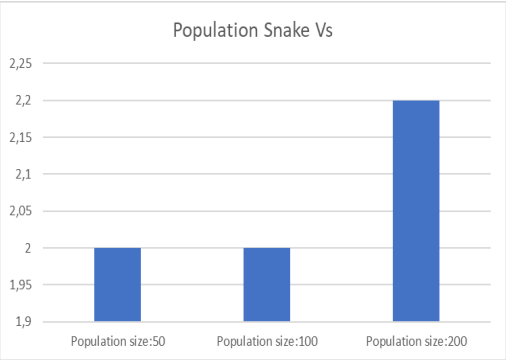

Figura 24 – Testes com populações para controladores heterogéneos

Nos teste com gerações tivemos um aumento bastante razoável até às 5000 gerações, a partir daí o aumento não se verificou, em gerações mais baixas a diferença para os agentes com maior fitness é de cerca de 1/3.

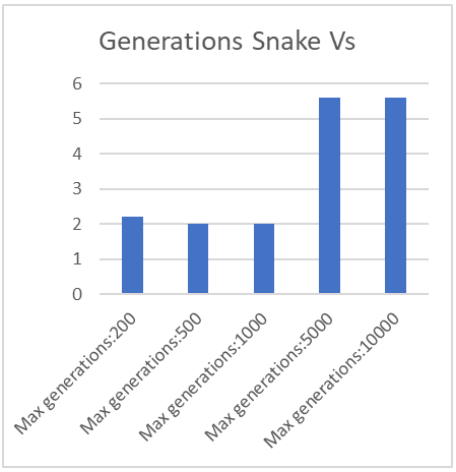

Figura 25 – Testes com gerações para controladores heterogéneos

Como em outras experiências verificamos que a mutação mais baixa produz melhores efeitos no agente, no entanto para valores mais altos não se verifica nenhuma queda abrupta de valores.

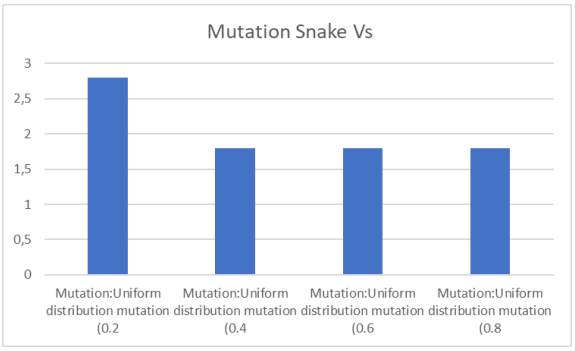

Figura 26 – Testes com mutações para controladores heterogéneos

O controlador heterogéneo apresentou melhores resultados com probabilidades de recombinação mais elevadas, valores mais baixos não apresentam grande diferença entre si, porém a diferença para valores mais elevados é de cerca de 20%.

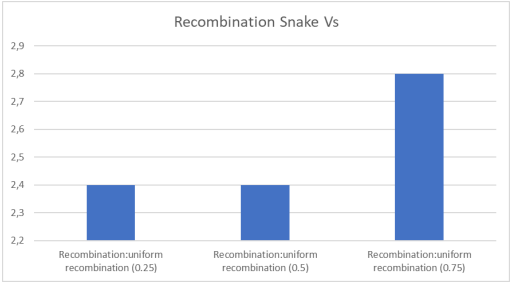

Figura 27 – Testes com recombinações para controladores heterogéneos

No torneio obtivemos resultados equilibrados no máximo e no mínimo de fitness, porém a diferença entre eles é de cerca de 20%.

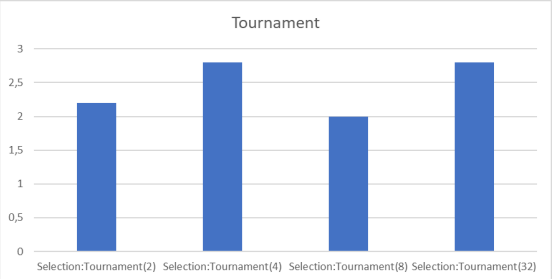

Figura 28 – Testes com torneios para controladores heterogéneos

Usamos em todos os agentes 4 *hidden units* pois foi o que apresentou melhores resultados, os nossos outputs foram um array de 4 posições onde na função *decide* verificávamos qual a posição com o valor mais elevado e aí, o agente decidia a sua ação.

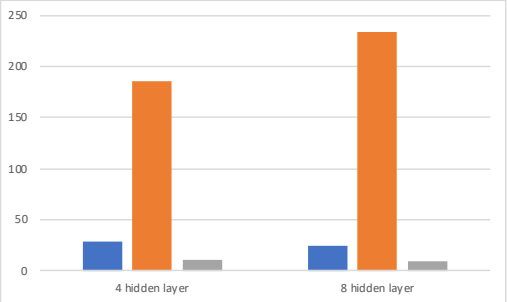

Figura 29 – Testes com layers para Snake A.I. 2

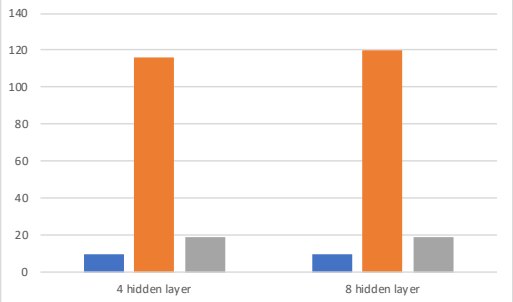

Figura 30 – Testes com layers para Snake A.I.

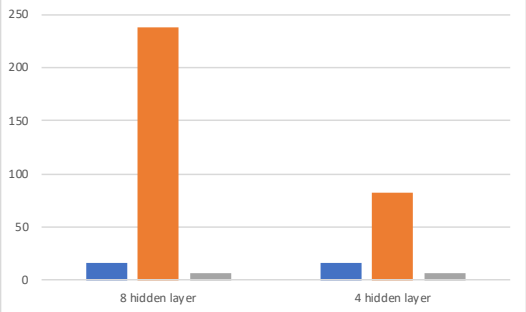

Figura 31 – Testes com layers para Snake Coop

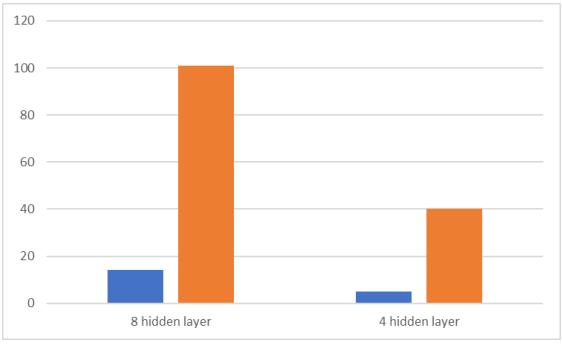

Figura 32 – Testes com layers para Snake Vs

Também testamos outros aspetos dos agentes, tais como a fórmula com a mutação é calculada, testamos o modelo *Gaussin* em que a mutação é muito baixa e também uma mutação Random, esta última com melhores resultados e foi a que utilizámos.

Outro dos aspetos testados nos diversos agentes foi o método de seleção, onde o *tournament* se evidenciou com muito melhores resultados que o outro método de seleção, o *roulette*.

COMPARAÇÃO ENTRE OS VÁRIOS AGENTES

Como podemos verificar na imagem abaixo, com todas as cobras desenvolvidas. Sendo a Snake A.I. 2 aquela com melhores movimentos (laranja), número de comidas obtidas (cinzento), superando o outro agente singular que não se conseguiu desenvolver tanto.

Na comparação entre o controlador homogéneo e heterogéneo, apesar de ambos os controladores terem tido resultados semelhantes, o controlador heterógeno não teve o comportamento desejado, dado que um dos agentes realizou o necessário, mas o outro ficou pouco apenas num canto a fazer movimentos para à frente e para trás a tentar não morrer. Tentámos ainda resolver este problema sem as penalizações, mas tal não sucedeu.

Por sua vez o controlador heterogéneo mesmo não conseguindo comer e movimentar-se, tanto ambos os controladores faziam o desejado tentando ir atrás da comida. Apesar de morrer bastante cedo e nunca ultrapassar as 15 comidas e pouco mais do que 100 movimentos.

No fim de todas as experiências nos diversos agentes, podemos verificar que houve parâmetros com bastante mais eficácia no melhoramento dos agentes que outros.

O tamanho do torneio e o tamanho da população inicial foram os parâmetros que mais afetaram a performance dos agentes tendo sempre grandes crescimentos com um número adequado, por outro lado podemos verificar que o crescimento nos outros parâmetro embora exista não é tão pronunciado.

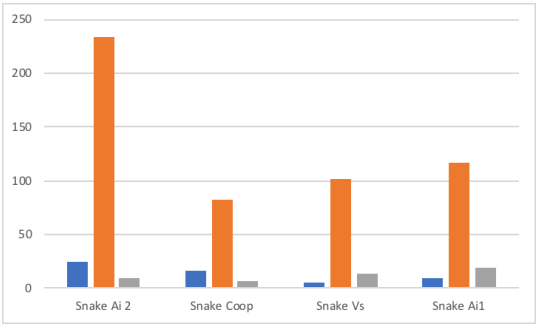

Figura 33 – Comparação entre fitnesses dos vários agentes usando os melhores parâmetros

CONCLUSÃO

Com este projeto, apesar das dificuldades sentidas na realização foi um trabalho que nos agradou fazer. Nunca tendo tido contacto com a área da inteligência artificial para além da experiência de utilizador tornou-se um objetivo interessante ver como iriamos aplicar um algoritmo genético ao contexto deste projeto.

Apesar de termos realizados alguns exercícios de inteligência artificial nas aulas práticas, o pouco conhecimento que tínhamos com a linguagem usada no trabalho provou ser uma barreira a ultrapassar, mas que certamente iremos utilizar no futuro depois de ter conseguido terminar o projeto. Sendo esta uma barreira para completar o projeto tivemos pena que não pudesse ter sido resolvida mais cedo apesar do nosso esforço e do professor, o que fez com que tivéssemos menos tempo para melhorar os nossos controladores baseados em redes neuronais.

Certamente foi interessante perceber e implementar mudanças tais como os inputs nos agentes e melhorar tanto os resultados.

A inteligência artificial tem vindo a ser uma tecnologia cada vez maior nos últimos anos. Tendo aplicações em quase todas as áreas, nomeadamente na área multimédia. Foi do nosso agrado trabalhar num projeto que nos traz mais próximo de uma tecnologia que está neste momento a ser tão intensivamente explorada por todo o mundo.