Inteligência Artificial
Snake

Tiago Costa (114629)
Tiago Almeida (113106)

Arquitetura do Agente - Funcionamento Básico

No primeiro momento do jogo, o servidor passa a informação ao Agente relativa ao mapa (i.e. tamanho do mapa e posições das paredes). Essa informação é imediatamente guardada num *collision map* para uso posterior.

Durante o jogo, em cada *tick*, o Agente toma uma decisão baseada em toda a informação a que tem acesso, criando uma lista de ações possíveis, que vai filtrando de acordo com uma lista de critérios:

- Remove a ação contrária à última ação que executou, para não ir contra si mesma
- Remove as ações de colisão, quer seja contra uma parede, *snake*, ou borda do mapa (no caso do *traverse* estar *false*)
- Remove as ações que levariam a um *loop*, isto é, ações que fariam com que a snake se encurralasse
- Se vir uma pelo menos uma maçã normal ou super no mesmo eixo X ou Y, tenta apanhá-la se a ação que a leva à maçã estiver dentro destas ações filtradas
- Se não houver maçãs à vista, a *snake* faz o movimento descrito no slide seguinte

De forma a encontrar movimentos que levem a que a *snake* se encurrale, o Agente calcula, em todos os *ticks*, a área em todas as direções que não levem a *Game Over*. Isto pode levar a duas situações:

- O Agente determina áreas iguais em todos os lados. Isto significa que não há áreas fechadas, ou seja, nenhuma ação leva a encurralamento
- O Agente determina áreas superiores umas às outras. Neste caso, é evidente que existem áreas fechadas, e evitam-se

Arquitetura do Agente - Com Traverse

Quando o Agente consegue passar pelas paredes e pelas bordas do mapa (*Traverse == True*), depois de ter o seu movimento filtrado:

- Se houver pelo menos uma maçã à vista (descrito no funcionamento básico) e a ação que leva a essa maçã tiver passado pelos filtros, a *snake* vai na direção da maçã
- Se não houver, a *snake tem o seguinte movimento default:*
  - Durante 40 *steps*, a *snake* percorre uma direção horizontal em linha reta
  - No fim desses 40 *steps*, a *snake* sobe durante 7 *steps* e volta a percorrer uma direção horizontal
  - Este movimento é interrompido sempre que a *snake* é forçada a executar uma ação, para evitar *Game Over*, *loop*, ou para pegar uma maçã.

- Este movimento assegura que a *snake* cobre uma grande área do mapa em pouco tempo, tendo em conta que o mapa é um retângulo com um eixo Y menor que o eixo X.

Arquitetura do Agente - Sem Traverse

Quando o Agente não consegue passar pelas paredes nem pelas bordas do mapa (*Traverse == False*), depois de ter o seu movimento filtrado:

- Se houver pelo menos uma maçã à vista (descrito no funcionamento básico) e a ação que leva a essa maçã tiver passado pelos filtros, a *snake* vai na direção da maçã
- Se não houver, a snake tem o seguinte movimento *default:*
  - Durante 12 *steps*, a *snake movimenta-se em linha reta*
  - *No fim desses 12 steps, a snake escolhe uma ação aleatória que não leve a Game Over nem a loop.*
  - *Sempre que a snake percorre a linha reta durante 12 steps, se for forçada a mudar de direção a meio dos 12 devido a uma ação que leve a Game Over*, *loop* ou maçã, *o counter dá reset.*

Algoritmos Usados

No projeto desenvolvido, os principais algoritmos usados são os algoritmos que verificam as ações que levam a *Game Over*, as ações que levam a *loop* e a ação que leva a uma maçã.

Para as ações que levam a *Game Over*, foi criada uma função *will_hit_test()* que chama funções *move_snake_head()* e *hits_(...)_test()* para verificar se, executando um movimento, a cabeça da *snake* fica na mesma coordenada que um corpo de cobra, parede ou borda (no caso do traverse estar desligado).

No caso das ações que levam a loop, foi criada uma função calculate_and_compare() que, calcula as áreas à volta da snake (como descrito no funcionamento básico) e retorna as ações que levam às maiores áreas. Esta função chama a função flood_fill() que é um algoritmo conhecido de cálculo de áreas (neste caso não usamos a parte do fill, apenas calculamos a área). Este algoritmo escolhe um quadrado inicial e pega os 4 quadrados vizinhos, adicionando à área os quadrados que não são paredes ou corpos de cobra. Fazendo esta pesquisa até serem visitados todos os quadrados, encontramos uma área fechada.

O algoritmo de pegar maçãs é simples: verifica se a cobra vê uma maçã, e se a maçã está num mesmo eixo X ou Y da cabeça da cobra. Caso esteja, a cobra escolhe a direção que leva à maçã, fazendo comparação de coordenadas.

Benchmarks

Nas primeiras fases de desenvolvimento, a cobra limitava-se a escolher movimentos aleatórios até encontrar uma maçã. Se o *traverse* estivesse desligado, o Agente apenas era capaz de evitar as bordas do mapa, uma vez que ainda não tinha sido criado o *collision map* com as paredes. Nesta fase, a pontuação média rondava os 20 pontos. A maior e principal causa de *Game Over* eram colisões com paredes.

Posteriormente foram implementados os algoritmos de evitar paredes, e a pontuação média subiu para os 50 pontos. A principal causa de *Game Over* nesta fase eram colisões com o próprio corpo vindas de encurralamento.

Durante um bom tempo foram testados vários métodos de evitar *loops*, através de pesquisas recursivas pelo corpo da cobra, entre outros. Eventualmente surgiu a ideia do *flood fill*, e a pontuação média subiu para os 75 pontos.

Finalmente, a movimentação da cobra foi melhorada, tendo-se experimentado métodos como por exemplo explorar o mapa uniformemente, através de um dicionário de *viewed_nodes*. No fim, o método final é o que se encontra explicado na arquitetura do agente. A pontuação média subiu para os 100 pontos.

Conclusões

Em retrospectiva, o projeto evoluiu de forma constante e consistente, devido à frequente discussão de ideias por parte dos 2 membros da equipa, bem como uma boa distribuição de tarefas ao longo do semestre.

A arquitetura do Agente não tem algoritmos específicos para o modo *multiplayer*, uma vez que não surgiram ideias relevantes que pudessem melhorar este aspeto do Agente. No entanto, o grupo reconhece que é possível com certeza melhorar o modo *multiplayer*, possivelmente através de alguma heurística calculada com o que se pode ver do corpo da cobra inimiga. Este tipo de algoritmos seriam mais abstratos, visto que o Agente não tem acesso a todas as coordenadas da cobra inimiga, e este é um obstáculo dinâmico e imprevisível.

A maior dificuldade no desenvolvimento do projeto foi sem dúvida a implementação do algoritmo que evita os *loops*. Depois de várias tentativas com diferentes abordagens, através de muita pesquisa e tentativa e erro foi possível implementar o algoritmo atual que funciona sem erros.

Os outros aspetos do Agente, como evitar obstáculos e encontrar maçãs foram triviais.

Para além disso, o algoritmo de movimento teve também uma série de modificações, mas a última versão é relativamente eficiente, apesar de ser muito melhorável ainda.