Geometry Friends: um jogo de cooperação

Relatório Final

4º ano do Mestrado Integrado de Engenharia Informática e Computação

Grupo T12_1
Catarina Correia - up201405765 - up201405765@fe.up.pt
José Aleixo Cruz - up201403526 - up201403526@fe.up.pt
José Carlos Coutinho - up201404293 - up201404293@fe.up.pt

Dezembro 2017

Objetivo

Descrição do cenário

Geometry Friends é um jogo de plataformas num ambiente 2D com um motor de simulação física (com gravidade e atrito). O objetivo dos jogadores é colecionar um conjunto de diamantes no menor tempo possível. Cada jogador (agente) controla uma das duas personagens (formas geométricas): um círculo amarelo e um retângulo verde.

Os agentes controlam a sua personagem, que dispõe de um número restrito de movimentos. O retângulo é capaz de mudar a sua forma para horizontal ou vertical (com a mesma área), andar para o lado direito ou esquerdo. O círculo pode andar para os lados, saltar ou crescer.

Num nível podem também existir obstáculos. Os obstáculos podem obstruir o caminho de apenas um dos agentes ou de ambos. Os obstáculos com a cor preta bloqueiam tanto o círculo como o retângulo. Os obstáculos com a cor verde apenas permitem que o retângulo os atravesse. Os obstáculos com a cor amarela apenas permitem que o círculo os atravesse.

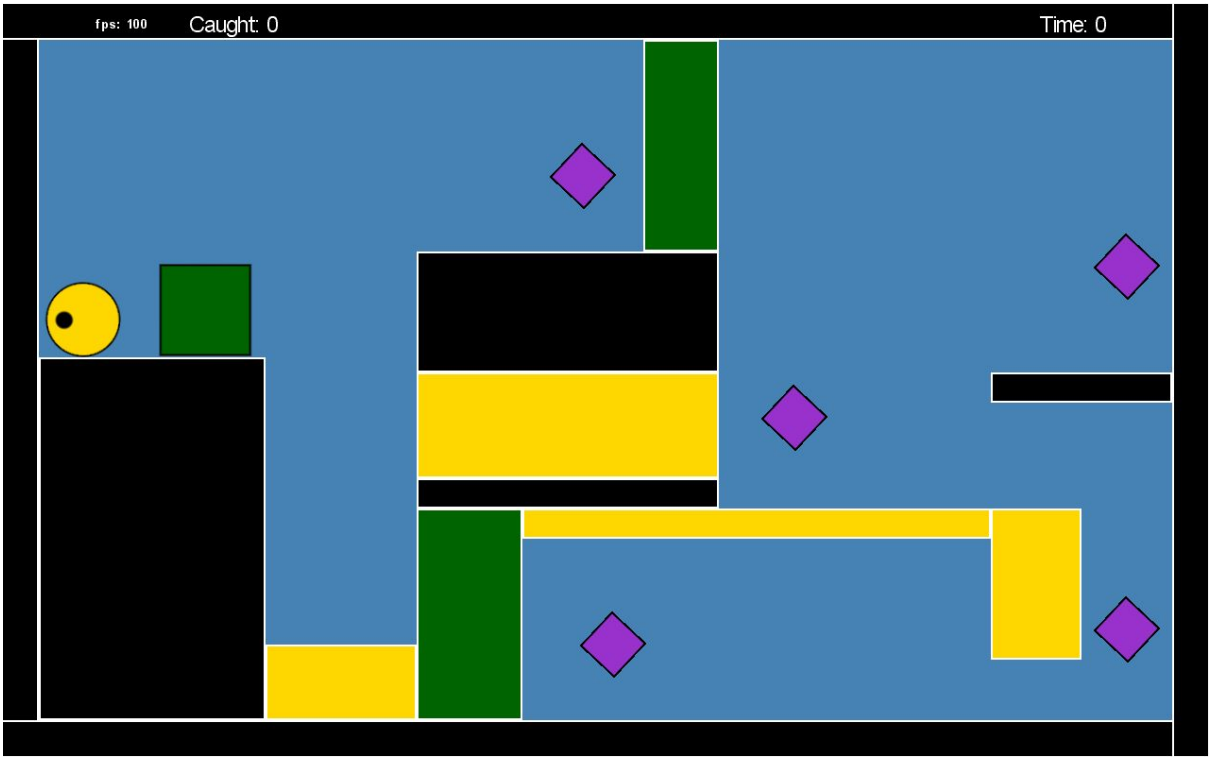


Figura 1 - Exemplo de um nível e dos vários tipos de obstáculos

No jogo existem dois tipos de níveis: cooperativos e individuais. Nos níveis cooperativos, os agentes devem utilizar os seus movimentos característicos e interagirem de forma a apanhar os diamantes. Nos níveis individuais apenas uma das personagens é posta à prova. Para este projeto serão focados apenas os níveis cooperativos.

Objetivos do trabalho

O objetivo deste trabalho consiste na implementação de um algoritmo e de um forma de comunicação que permita aos dois agentes cooperar entre si. Esta cooperação é fundamental para que os agentes consigam apanhar todos os diamantes dos diferentes níveis cooperativos.

Visto que uma das principais metas da competição é conseguir passar um nível no menor tempo possível, um dos propósitos do grupo, e o maior desafio do projeto, é conseguir implementar um algoritmo eficiente que tenha em conta a posição dos agentes, dos obstáculos e dos vários diamantes do nível e determine qual a ordem pela qual estes devem ser apanhados, bem como, para cada um dos diamantes, qual o agente que mais facilmente o consegue apanhar e se é necessário que ambos utilizem movimentos em conjunto para apanhar um diamante.

Especificação

Identificação e caracterização dos agentes

Na competição Geometry Friends, existem dois agentes: o círculo e o retângulo. O círculo é um objeto da classe CircleAgent e o retângulo é um objeto da classe RectangleAgent. Ambos derivam de AbstractAgent e implementam os seus métodos.

As únicas ações que os agentes podem tomar reduzem-se ao seu movimento e estão descritas na classe Moves.

Movimentos do retângulo	**Movimentos do círculo**
Moves.MOVE_LEFT	Moves.ROLL_LEFT
Moves.MOVE_RIGHT	Moves.ROLL_RIGHT
Moves.MORPH_UP	Moves.JUMP
Moves.MORPH_DOWN	Moves.GROW

A solução para este projeto passa por executar em cadeia uma série de movimentos que leve à maior pontuação possível no nível.

Propriedades dos Agentes

Tanto o CircleAgent como o RectangleAgent possuem várias características em comum. Tal pode ser observado na tabela que se segue, que descreve as propriedades destes dois agentes.

Tabela 1 - Propriedades dos agentes CircleAgent e RectangleAgent

Propriedades	**Domínio de Valores**	**Justificação**
Tempo de Vida	Permanente	O tempo de vida dos agentes corresponde à totalidade do tempo de jogo.
Nível de Cognição	Deliberativo	Os agentes mantêm uma representação do ambiente, tomam decisões com base nessa representação e em informações transmitidas pelo outro agente.
Implementação	Procedimental	As informações de controlo necessárias ao

		uso do conhecimento estão embutidas no próprio conhecimento.
Mobilidade	Estacionário	Os agentes trocam informação entre eles através de mensagens. Não existe um agente que faça a recolha de informação.
Adaptabilidade	Fixo	As capacidades dos agentes são independentes do ambiente e respetivas interações.
Localização	Local	Os agentes não conseguem correr em máquinas diferentes.
Autonomia Social	CircleAgent: Independente RectangleAgent: Controlado	O círculo toma decisões por ele e pelo retângulo.
Sociabilidade	CircleAgent: Responsável RectangleAgent: Atento	O retângulo aguarda mensagens enviadas pelo círculo.
Colaboração	Cooperativos	Os agentes trocam entre si mensagens para atingir um objetivo comum.

Entre os dois agentes, o CircleAgent será o líder. Ele interpretará as suas informações e as informações do RectangleAgent para calcular a melhor estratégia para completar o nível e depois transmiti-la e executá-la.

Protocolos de interação

Cada agente possui uma lista de mensagens que são objetos da classe AgentMessage, que consiste numa string e num anexo opcional que pode ser qualquer objeto. Esta lista é verificada sempre que é feita uma atualização ao estado de jogo. Se um agente tiver adicionado uma nova mensagem à lista, a mensagem é transmitida ao outro agente. A cada atualização, um agente verifica se recebeu novas mensagens e invoca o método para lidar com elas. Este mecanismo já vinha implementado no jogo.

Para que cada agente possa interpretar a informação transmitida pelo outro, foi criada uma classe Message, que possui um identificador próprio e que é anexada a uma AgentMessage.

Da classe Message derivam as classes Request e Answer. Um objeto Request representa um pedido feito por um agente. Cada Request incluirá um objeto Command, que indica a ação que o agente invocador pede que o outro agente execute. Um objeto Answer representa a resposta de um agente a um determinado Request e pode ter um objeto anexado.

Para cada Request enviado por um agente, espera-se uma Answer enviada pelo outro, confirmando ou não a execução do Request.

Por exemplo, se o CircleAgent precisar que o RectangleAgent se mova para a esquerda, gerará um Request com um Command que invoque a função MoveLeft() do retângulo. De seguida, anexará o Request a uma AgentMessage e adiciona-a à sua lista de mensagens. Do outro lado, o RectangleAgent receberá a mensagem e ao interpretá-la corre a função MoveLeft(), que retornará se foi bem sucedida ou não. Caso tenha sido, envia uma Answer com o tipo "YES", caso contrário, envia "NO". O CircleAgent recebe a mensagem e decide como prosseguir.

Também é possível associar a uma Answer um Object "attachment" quando a resposta é do tipo "YES". Por exemplo, no início do jogo, o círculo quer saber qual o diamante e respetivo caminho mais perto do retângulo para poder decidir a que diamantes é que cada um dos agentes vai. Assim, gerará um Request com um Command que invoque a função getCheapestPath(). Por sua vez, o retângulo receberá a resposta e, ao interpretá-la, corre a função getCheapestPath() que retornará se foi bem sucedida ou não e, se bem sucedido, anexará ao Object "attachment" o caminho (Path) para o diamante mais próximo.

Desenvolvimento

Plataforma/ferramenta utilizada

O Geometry Friends é desenvolvido pelo laboratório GAIPS INESC-ID. A comunicação entre agentes já está estabelecida através de uma interface de mensagens. Como tal, este trabalho não requer a utilização de umas das plataformas propostas, pois é desenvolvido utilizando o software específico da competição, que requer programação em C#. O kit da competição inclui um projeto em Visual Studio, que permite adicionar a implementação de agentes.

Estrutura da aplicação

O projeto de Visual Studio utilizado para desenvolver os agentes é composto por duas partes: o executável do jogo e as classes que compõem os agentes. O código fonte é compilado para um ficheiro ".dll" que é copiado para o diretório do jogo. Dentro do jogo pode-se escolher que implementação de agentes deve ser utilizada.

Módulos

O executável do jogo possui código desenvolvido por terceiros e que não deve ser de qualquer forma adulterado para implementar os agentes. É tratado como uma caixa negra e o seu funcionamento não será abordado.

Toda a implementação é feita no código fonte dos agentes. Considerando cada ficheiro ".cs" do código fonte como um módulo, segue-se uma breve descrição do papel de cada um na execução dos agentes:

Módulo	**Papel**
CircleAgent.cs	Contém tudo aquilo que define o processo de decisão do círculo.
RectangleAgent.cs	Contém tudo aquilo que define o processo de decisão do retângulo.
AgentType.cs	Contém um enum que define o tipo de agente.
AStar.cs	Dado um grafo, um nó inicial e um nó final, devolve o trajeto mais curto entre esses nós através do algoritmo A*.
Communication.cs	Contém a definição das classes Message, Request e Answer, assim como a definição de todos os Commands que o CircleAgent pode enviar ao RectangleAgent.
Graph.cs	Cria um grafo a partir da informação do nível e dos agentes, de forma a permitir o cálculo de caminhos.

Matrix.cs	Gera uma matriz bidimensional que possui informação relativa à posição de todos os elementos de um nível no nível.
Movement.cs	Inclui todas as restrições que movimento que devem ser aplicadas aos caminhos obtidos pela execução do A*.
Node.cs	Define o nó que faz parte do grafo que é percorrido pelo A*.
Pixel.cs	Contém informação relativa ao elemento que ocupa um determinado pixel no ecrã de jogo.
Status.cs	Possui funções e variáveis que permitem executar com maior precisão o movimento de cada agente.
Utils.cs	Contém funções com utilidades diversas, como por exemplo o cálculo de áreas e de distâncias.

Implementação

Análise do nível

Matriz de jogo

Ao ser selecionado um nível, cada agente executa uma função *Setup()* que recebe como argumentos as coordenadas e as dimensões de cada elemento do nível.

Nos agentes implementados, esta função começa por gerar a matriz do nível, recorrendo à função *GenerateMatrixFromGameInfo()*, que criaum array bidimensional com as dimensões em da área de jogo e inicializa cada posição com o tipo de elemento que a ocupa. Os tipos de elementos estão descritos na enumeração *Pixel.Type*:

- *Space*: representa um espaço livre;
- *Circle*: representa uma posição ocupada pelo CircleAgent;
- *Rectangle*: representa uma posição ocupada pelo RectangleAgent;
- *Diamond*: representa uma posição ocupada por um diamante;
- *Obstacle*: representa uma posição ocupada por um obstáculo;
- *RectanglePlatform*: representa uma posição ocupada por uma plataforma que apenas o RectangleAgent consegue penetrar.
- *CirclePlatform*: representa uma posição ocupada por uma plataforma que apenas o CircleAgent consegue penetrar.

Representação poligonal

Com a matriz gerada, os agentes constroem o grafo de caminhos. Uma primeira abordagem consistiu em considerar que cada pixel de jogo poderia ser um nó, no entanto existem cerca de 700 mil pixéis na área de jogo, pelo que os cálculos desse grafo seriam absurdamente complexos. Assim, considerou-se que cada agente e diamante seria representado por um único nó, enquanto os obstáculos seriam representados por quatro nós: um em cada canto.

Assim, o grafo corresponde a uma representação poligonal do nível, que converte cada polígono (cada obstáculo) em nós que representam os seus vértices.

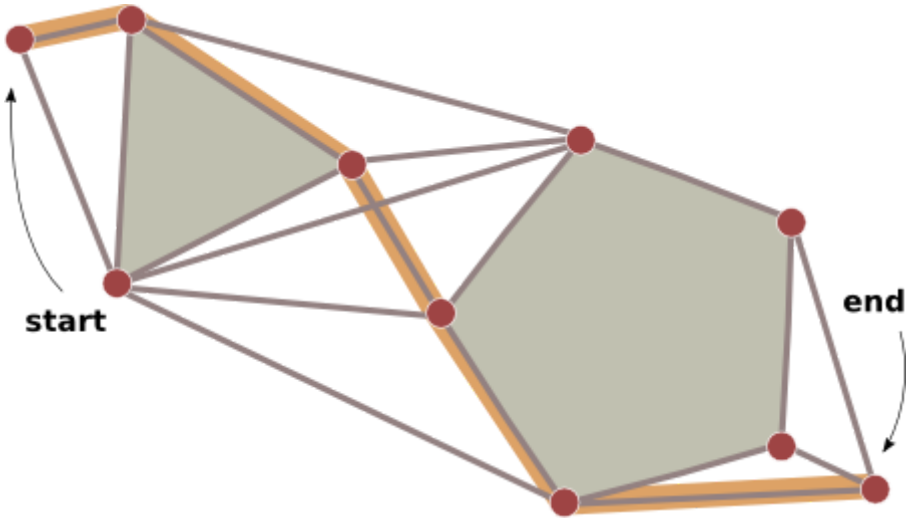


Figura 2 - Representação poligonal

Para determinar se um determinado nó é ou não adjacente a outro, é utilizada a função *walkableLine(Point begin, Point end, AgentType agentType)*, que segundo o tipo de agente e a informação sua matriz, verifica se existe uma linha que o agente possa percorrer entre a posição dos dois nós. Esta função tem em conta a obstrução que os obstáculos causam aos agentes e que os agentes causam um ao outro.

A função *walkableLine()* é corrida para todas as combinações de nós possíveis no grafo. Se a função retornar um valor verdadeiro, os nós definem-se como adjacentes, ou seja, é possível que o agente navegue entre os dois recorrendo aos seus movimentos. Caso contrário, não são adjacentes.

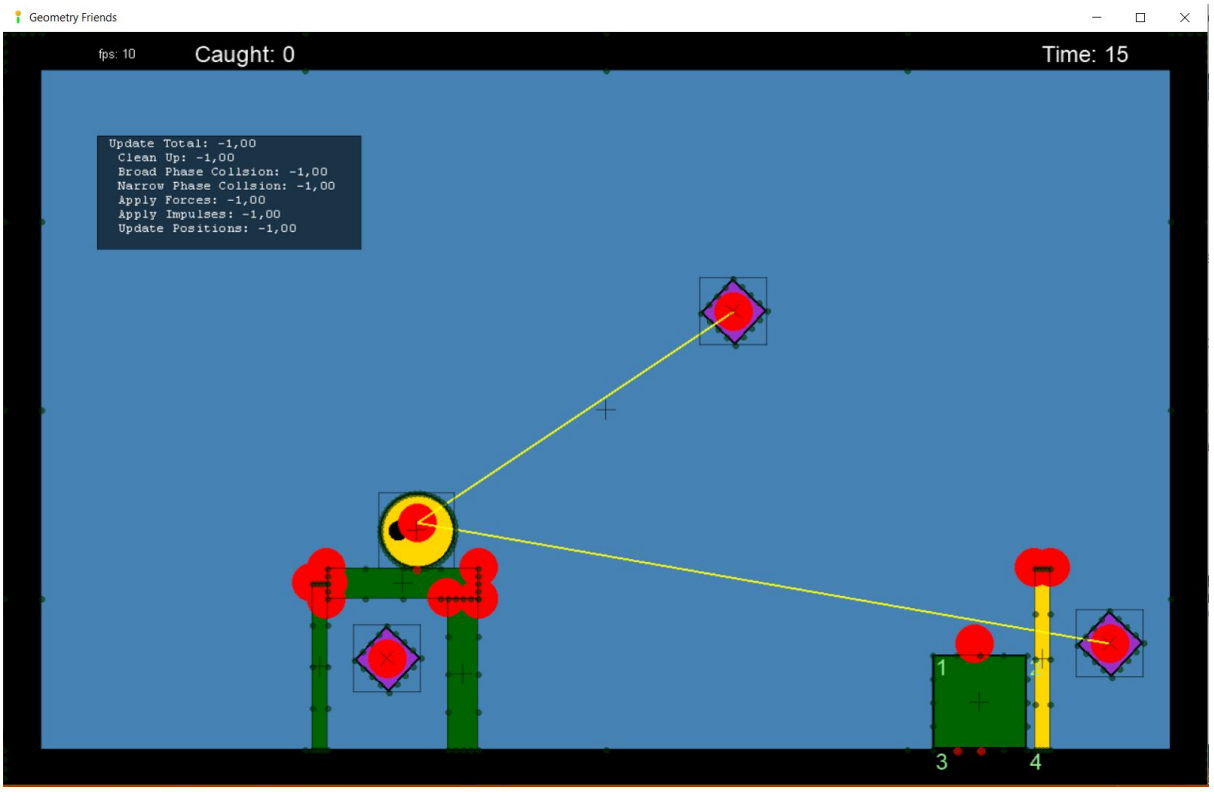


Figura 3 - Exemplo de um grafo do CircleAgent

Cada círculo vermelho representa um nó do grafo. As linhas amarelas representam a adjacência entre o nó que representa o círculo e os nós que representam os diamantes. Neste exemplo em particular, o círculo não possui uma linha para o diamante exatamente abaixo dele, pois não há nenhum caminho possível para ele o apanhar, já que os obstáculos verdes apenas podem ser atravessados pelo retângulo.

Caminhos mais curtos e A*

Depois de estar criado o grafo e determinadas todas as suas adjacências, executa-se o algoritmo A* para determinar quais são os caminhos mais curtos que cada agente pode tomar para apanhar um diamante.

Como heurística para o A* é usada a distância em linha reta entre o nó de origem e o nó de destino. Como o custo real nunca será menor que a distância em linha reta, a heurística é admissível e é sempre encontrado o caminho mais curto entre dois nós.

O A* corre para cada agente e guarda numa lista os caminhos mais curtos para cada diamante no momento inicial.

Se um nó que representa um diamante não for adjacente a nenhum nó do grafo do agente, ele não pode ser apanhado individualmente e é adicionado à lista de diamantes que devem ser apanhados cooperativamente. O CircleAgent é responsável por recolher os diamantes que o RectangleAgent não consegue apanhar sozinho e separar os aqueles que podem ser apanhados independemente dos diamantes que necessitam de movimentos cooperativos.

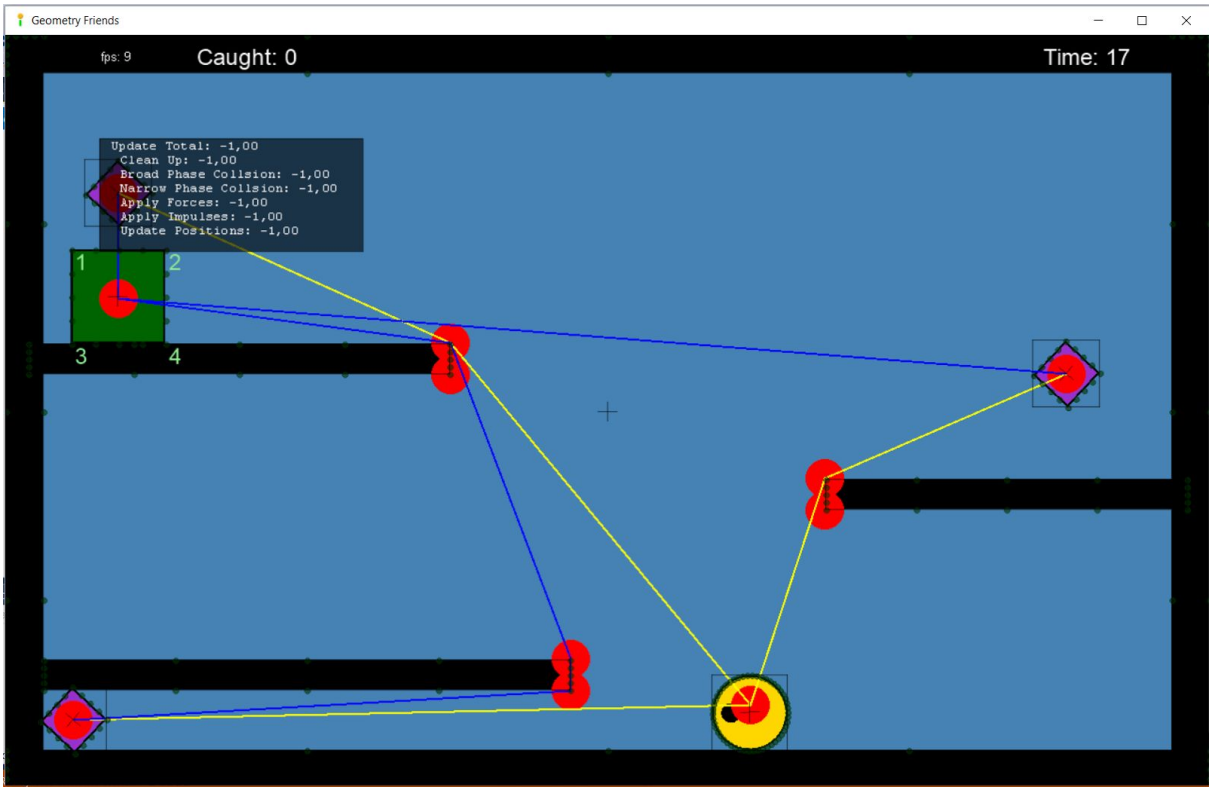


Figura 4 - Caminhos mais curtos de cada agente até cada diamante

Primeiras decisões

Depois do *setup*, os agentes estão prontos para partilhar um com o outro quais são os diamantes que devem ir buscar. Para isso, é executada uma função pelo CircleAgent (que é o agente líder) com o objetivo de decidir que diamantes os agentes devem tentar apanhar inicialmente. O método funciona da seguinte forma:

```
circDiamond = CircleAgent.CheapestPath();

/* recebe uma resposta do RectangleAgent com o diamante mais próximo*/
rectDiamond = CircleAgent.sendRequest(new Request.CheapestPath());

if(rectDiamond.totalCost >= circDiamond.totalCost)
{
	/* o RectangleAgent deverá apanhar o diamante seguinte com menor custo e o
CircleAgent fica encarregue de apanhar o seu diamante */

	Circle.sendRequest(new Request.CatchNextNode());
	Circle.CatchCurrentNode();
}
else
{
	/* o inverso da situação anterior */

	Circle.sendRequest(new Request.CatchCurrentNode());
	Circle.CatchNextNode();
}
```

Assim os agentes optam por uma estratégia gananciosa, em que a ordem pela qual apanham os diamantes depende apenas do diamante cujo caminho tem menor custo total.

Os diamantes que necessitam de movimentos cooperativos para ser apanhados são tratados depois de todos os que podem ser apanhados individualmente. Assim, caso o tempo limite do nível seja curto, otimiza-se a quantidade de diamantes que se consegue apanhar.

Movimentos

Cada movimento dos agentes é descrito da seguinte forma:

- CircleAgent:
  - ROLL_LEFT - O CircleAgent aplica uma força para rolar para a esquerda
  - ROLL_RIGHT - O CircleAgent aplica uma força para rola para a direita
  - JUMP - O CircleAgent aplica um impulso para cima
- RectangleAgent:
  - MOVE_LEFT - O RectangleAgent aplica uma força para deslizar para a esquerda.
  - MOVE_RIGHT - O RectangleAgent aplica uma força para deslizar para a direita.
  - MORPH_UP - O RectangleAgent redistribui a sua área de modo a ficar com mais altura e menos largura.
  - MORPH_DOWN - O RectangleAgent redestribui a sua área de modo a ficar com menos altura e mais largura.

Para os agentes efetuarem o movimento pretendido, é necessário atribuir o valor correspondente a esse movimento ao atributo *currentAction*. O programa base do GeometryFriends tratará de ler esse atributo e de fazer o agente executar o movimento correspondente.

O uso simples destes movimentos não é suficiente para conseguir que os agentes possuam um controlo preciso, por isso foi necessária a implementação de funções que facilitassem e melhorassem o movimento dos agentes, bem como o estabelecimento de *flags* quanto ao posicionamento e velocidade dos agentes.

Para facilitar a representação do posicionamento e das velocidades, optou-se dividir os valores em escalões com a enumeração *Quantifier*.

- Quantifier:
  - NONE:
    - Posicionamento: Significa que não se encontra nesta posição relativa em relação a outro objeto. (ex:

LEFT_FROM_TARGET = NONE significa que o agente não se encontra à esquerda do alvo)
        - Velocidade: Significa que se movimenta 0-9 pixels/segundo numa dada direção.
    - SLIGHTLY:
        - Posicionamento: Significa que se encontra a 0-49 pixels de distância do objeto em causa numa dada direção.
        - Velocidade: Significa que se movimenta 10-39 pixels/segundo numa dada direção.
    - A_BIT:
        - Posicionamento: Significa que se encontra a 50-199 pixels de distância do objeto em causa numa dada direção.
        - Velocidade: Significa que se movimenta 40-99 pixels/segundo numa dada direção.
    - A_LOT:
        - Posicionamento: Significa que se encontra a mais de 200 pixels de distância do objeto em causa numa dada direção.
        - Velocidade: Significa que se movimenta a mais de 100 pixels/segundo numa dada direção.

Para a implementação das *flags* foi criada a classe Status, sendo que cada um dos agente possui um atributo *agentStatus* que pertence a esta classe. Um Status assume que existem sempre apenas dois agentes (um CircleAgent e um RectangleAgent) e um alvo/obstáculo, de modo a haver sempre a comparação simultânea entre o agente em questão e o outro agente, e entre o agente em questão e o alvo/obstáculo.

- Status:
    - Atributos:
        - AgentType agentType: Indica se o agente a que se refere este Status é um CircleAgent ou um RectangleAgent.
        - Moves actualMove: Indica a *currentAction* do agente a que se refere este Status.
        - Quantifier Left_From_Target: Indica o quão à esquerda o agente se encontra do alvo.
        - Quantifier Right_From_Target: Indica o quão à direita o agente se encontra do alvo.
        - Quantifier Above_Target: Indica o quão acima o agente se encontra do alvo.
        - Quantifier Below_Target: Indica o quão abaixo o agente se encontra do alvo.
        - bool Near_Target: Indica, com base nos quatro Quantifiers anteriores, se o agente se encontra próximo do alvo.

        - Quantifier Above_Other_Agent: Indica o quão acima o agente está do outro agente.
        - Quantifier Below_Other_Agent: Indica o quão abaixo o agente está do outro agente.
        - Quantifier Right_From_Other_Agent: Indica o quão à direita o agente está do outro agente.
        - Quantifier Left_From_Other_Agent: Indica o quão à esquerda o agente está do outro agente.
        - bool Near_Other_Agent: Indica, com base nos quatro Quantifiers anteriores, se o agente se encontra próximo do outro agente.
        - bool Moving: Indica se o agente tem uma velocidade superior a 10 pixels/segundo em qualquer direção.
        - bool Other_Agent_Moving: Indica se o outro agente tem uma velocidade superior a 10 pixels/segundo em qualquer direção.
        - Quantifier Moving_Right: Indica a velocidade do agente para a direita.
        - Quantifier Moving_Left: Indica a velocidade do agente para a esquerda.
        - Quantifier Moving_Up: Indica a velocidade do agente para cima.
        - Quantifier Moving_Down: Indica a velocidade do agente para baixo.
        - Quantifier Moving_Towards_Target: Indica a velocidade do agente na direção do alvo.
        - bool Blocked: Indica se o agente se está a tentar mover, mas a sua velocidade não ultrapassa os 10 pixels/segundo.
    - Métodos:
        - Update(): Atualiza todos os atributos de Status.

Para o CircleAgent, foram implementadas as seguintes funções:

- Roll(*direção*, *velocidade*)
    - Devolve o movimento necessário para o CircleAgent se mover dentro da velocidade necessária.
    - Caso a velocidade seja inferior à pretendida, o CircleAgente irá rolar na direção pretendida. Se for superior à pretendida, o CircleAgente irá realizar força para rolar na direção oposta, de modo a travar.
- RollToPosition(*posição)*
    - Recorrendo a Roll(), devolve o movimento necessário para o CircleAgent rolar até à posição pretendida.

    - Quanto mais próximo da posição pretendida, menor será a velocidade do agente, de modo a aumentar a precisão.
- JumpOnto(*alvo*)
    - Recorrendo a Roll(), devolve o movimento necessário para o CircleAgent se colocar em cima do alvo.
    - Caso esteja demasiado próximo, rola na direção oposta. Quanto tiver espaço suficiente para ganhar velocidade para saltar, o CircleAgent rola na direção do alvo. Se o movimento for na direção do alvo, e estiver próximo, então o CircleAgent salta.
- JumpOntoRectangle()
    - Recorrendo a JumpOnto(), devolve o movimento necessário para o CircleAgent se colocar em cima do RectangleAgent.
- JumpAboveObstacle(*obstacle*)
    - Semelhante ao JumpOnto(), devolve o movimento necessário para o CircleAgent saltar por cima de um obstáculo.
- HoldGround()
    - Recorrendo a Roll(), devolve o movimento necessário para o CircleAgent se manter no mesmo sítio.
- Lauch()
    - Recorrendo a JumpOntoRectangle(), devolve o movimento necessário para o CircleAgent se colocar em cima do RectangleAgent, e comunica com o RectangleAgent para o auxiliar se projetar mais alto no ar.

Para o RectangleAgent, foram implementadas as seguintes funções:

- Move(*direção, velocidade*)
    - Semelhante ao Roll() do CircleAgent, devolve o movimento necessário para o RectangleAgent se mover na direção pretendida com a velocidade pretendida.
- MoveToPosition(*posição*):
    - Semelhante ao RollToPosition(), recorre a Move() para devolver o movimento necessário para o RectangleAgent se mover para a posição pretendida.
    - Tem um *overload* que faz com que o RectangleAgent faça MorphUp/MorphDown quando chega à posição pretendida.
- HoldGround()
    - Semelhante ao HoldGround() do CircleAgent, recorre a Move() para devolver o movimento necessário para o RectangleAgent se manter no mesmo sítio.
    - Tem um *overload* que faz com que o RectangleAgent faça MorphUp/MorphDown quando ficar imóvel.

Decisões ao longo do nível

Consoante a posição do diamante que um agente tem como função apanhar ele executa os movimentos pré-definidos de forma a poder alcançar o diamante.

Quando um diamante é apanhado por um agente, ele volta a calcular os caminhos mais curtos para os restantes diamantes. O CircleAgent deteta que um diamante foi apanhado e inicia o processo de comunicação e decisão com o RectangleAgent. Isto ocorre sempre, enquanto a lista de diamantes que um dos agentes consegue apanhar sozinho não estiver vazia.

Quando um agente apanha todos os diamantes atribuídos a si, espera que o outro agente termine de apanhar os seus e só depois é que prosseguem para a execução de movimentos cooperativos.

O único movimento cooperativo incorporado é o em que o círculo usa o retângulo como plataforma de lançamento. Assim, sempre que existem diamantes na lista de diamantes que devem ser apanhados cooperativamente, os agentes procuram sempre executar este movimento para apanhá-lo.

Experiências

Objetivo

Em termos de jogo, a pontuação é calculada da seguinte forma:

$$Pontuação = Nível\ Completo * \frac{(Tempo\ Limite - Tempo)}{Tempo\ Limite} + (Diamantes\ Apanhados * Pontuação\ do\ Diamante)$$

Como as experiências comparam níveis que tenham sido completados, a função usada para a avaliação de resultados é o tempo necessário para que todos os diamantes sejam apanhados. Quanto menos tempo for necessário, melhor é o resultado.

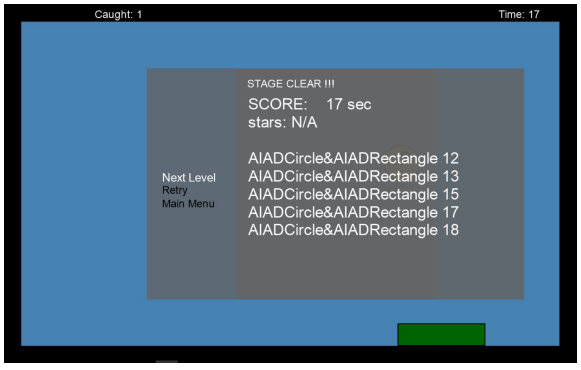
O objetivo destas experiências é provar que os agentes implementados são capazes de apanhar todos os diamantes existentes no nível, dentro do tempo limite, da forma mais rápida possível, usando a comunicação e cooperando entre si. Caso isso não seja possível, o objetivo passa a ser apanhar o maior número de diamantes.

Pretende-se também com isto provar que os agentes já referidos são mais inteligentes que agentes implementados de forma aleatória e agentes que não cooperam entre si.

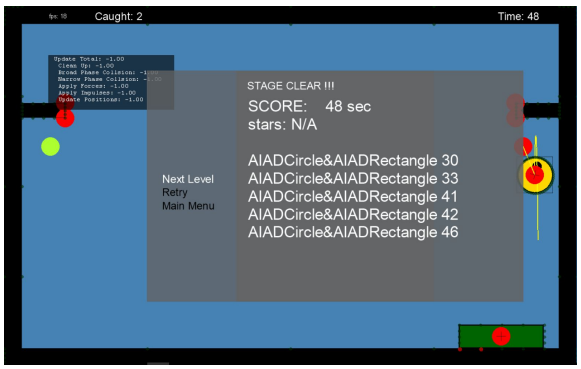
Resultados

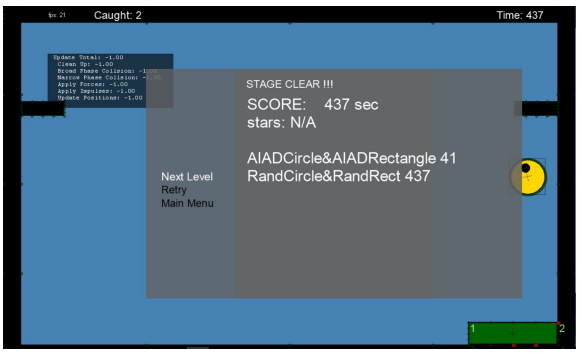
Nas seguintes imagens, pode observar-se a comparação dos agentes implementados com os agentes aleatórios:

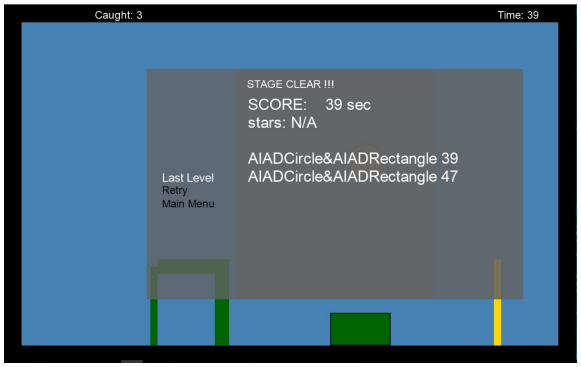
Agentes implementados

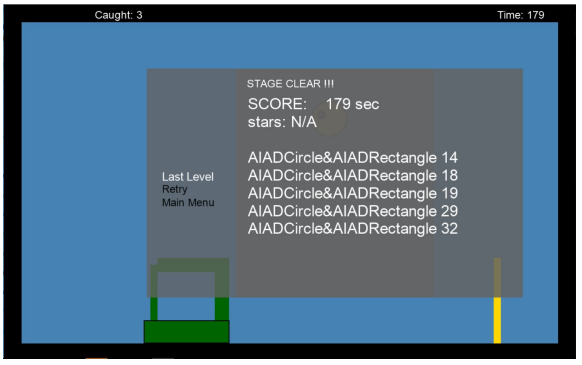


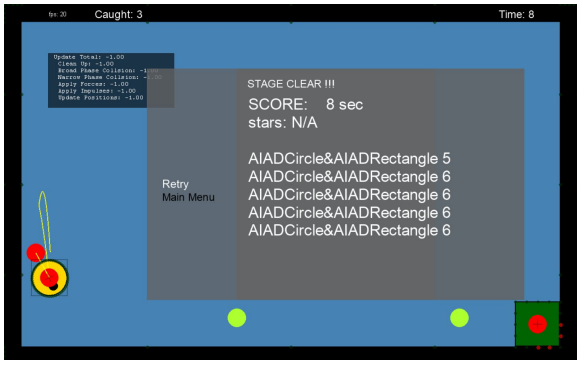
Agentes aleatórios

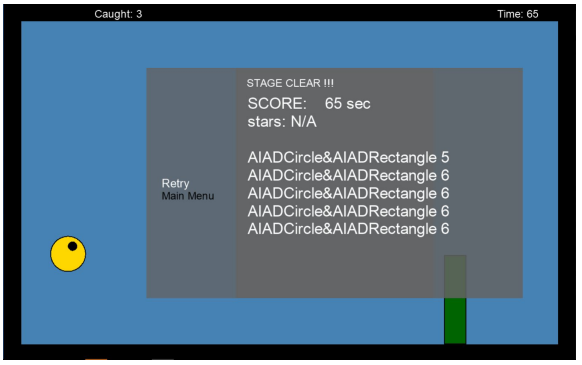

Caught: 2
Time: 48
STAGE CLEAR !!!
SCORE: 48 sec
stars: N/A
AIADCircle&AIADRectangle 30
AIADCircle&AIADRectangle 33
AIADCircle&AIADRectangle 41
AIADCircle&AIADRectangle 42
AIADCircle&AIADRectangle 46
Next Level
Retry
Main Menu


Caught: 2
Time: 437
STAGE CLEAR !!!
SCORE: 437 sec
stars: N/A
AIADCircle&AIADRectangle 41
RandCircle&RandRect 437
Next Level
Retry
Main Menu


Caught: 3
Time: 39
STAGE CLEAR !!!
SCORE: 39 sec
stars: N/A
AIADCircle&AIADRectangle 39
AIADCircle&AIADRectangle 47
Last Level
Retry
Main Menu


Caught: 3
Time: 179
STAGE CLEAR !!!
SCORE: 179 sec
stars: N/A
AIADCircle&AIADRectangle 14
AIADCircle&AIADRectangle 18
AIADCircle&AIADRectangle 19
AIADCircle&AIADRectangle 29
AIADCircle&AIADRectangle 32
Last Level
Retry
Main Menu


Caught: 3
Time: 8
STAGE CLEAR !!!
SCORE: 8 sec
stars: N/A
AIADCircle&AIADRectangle 5
AIADCircle&AIADRectangle 6
AIADCircle&AIADRectangle 6
AIADCircle&AIADRectangle 6
AIADCircle&AIADRectangle 6
Retry
Main Menu


Caught: 3
Time: 65
STAGE CLEAR !!!
SCORE: 65 sec
stars: N/A
AIADCircle&AIADRectangle 5
AIADCircle&AIADRectangle 6
AIADCircle&AIADRectangle 6
AIADCircle&AIADRectangle 6
AIADCircle&AIADRectangle 6
Retry
Main Menu

Conclusões

Análise dos resultados das experiências

Pelas imagens demonstradas nos resultados, é possível observar-se que tanto os agentes aleatórios como os agentes implementados (agentes cooperativos) conseguem apanhar os todos os diamantes e, assim, completar o nível. Contudo, é possível verificar-se que os agentes cooperativos são mais eficientes que os aleatórios e, deste modo, conseguem muito mais rapidamente concluir o nível.

Também é possível observar-se que, em alguns níveis, os agentes implementados, apesar de mais eficientes, apresentam algumas disparidades entre execuções de um mesmo nível. Por exemplo, para o terceiro par de imagens, pode ver-se que os tempos variam entre 14 a 32 segundos.

A plataforma Geometry Friends já vinha com alguns agentes implementados (não cooperativos). Devido à falta de outro tipo de agentes cooperativos implementados, não foi possível comparar-se a eficiência da nossa implementação com outras abordagens.

Aplicabilidade de SMA ao *Geometry Friends*

O *GeometryFriends* possui níveis em que os diamantes apenas conseguem ser apanhados se houver cooperação entre as duas personagens do jogo: o círculo e o retângulo. Se apenas um dos personagens jogasse, o nível nunca seria completado.

Um sistema multi-agente permite não só que os dois personagens possam ser controlados ao mesmo tempo, mas também que cooperem um com o outro de forma a completar o nível e fazê-lo no menor tempo possível. Os dois agentes do Geometry Friends têm características complementares que contribuem para esse objetivo.

Além disso, para níveis mais complexos, um SMA permite a distribuição da tarefa de computação de cálculos por mais do que um processo, podendo até ser executada em paralelo. Permite também a utilização de algoritmos e técnicas diferentes para cada agente, desde que haja um canal de comunicação entre os processos. Isto contribui para uma resolução ainda mais rápida do problema e exploração de diferentes abordagens.

Melhoramentos

Travelling Salesman Problem

Para decidir a ordem pela qual os diamantes são apanhados, foi aplicada uma estratégia gananciosa, tendo em conta o custo total (distância) que um agente tem para chegar a um determinado diamante, que nem sempre será a melhor opção. Um possível melhoramento seria incorporar um algoritmo capaz de resolver o *Travelling Salesman Problem* com dois agentes, de forma a obter a sequências mais rápida de apanhar os diamantes.

Movimentos limitados

A maior limitação destes agentes encontra-se na diversidade de movimentos que podem fazer. Por o jogo se basear num motor de física, sobre os quais não é divulgada qualquer informação, os movimentos dos agentes têm de ser executados de uma forma muito muito dinâmica. Em níveis mais complexos, o CircleAgent e o RectangleAgent demorarão mais tempo a colecionar os diamantes todos, se é que o conseguirão fazer, pois não estão preparados para ultrapassar todo o tipo de obstáculos.

Recursos

Bibliografia

INESC. “Geometry Friends | Cooperation Puzzle Game.” Geometry Friends Cooperation Puzzle Game, INESC, http://gaips.inesc-id.pt/geometryfriends/.

D'Inverno, Mark, et al. Foundations and Applications of Multi-Agent Systems. Springer, 2002.

Vlassis, Nikos. A Concise Introduction to Multiagent Systems and Distributed Artificial Intelligence. Morgan & Claypool, 2007.

Oliveira, Eugénio. Agentes e Inteligência Artificial Distribuída, http://paginas.fe.up.pt/~eol/AIAD/aiad1718.html.

Software

Sistema operativo: Windows
IDE: Visual Studio
Linguagem: C#

Elementos do grupo

- Catarina Correia - 1/3
- José Aleixo Cruz - 1/3
- José Carlos Coutinho - 1/3

Apêndice

Executar o jogo com os agentes desenvolvidos

Instalar o jogo:

Pelo facto de os ficheiros de jogo ocuparem muito espaço, não é possível submetê-los juntamente com o código fonte.

Para experimentar os agentes é necessário seguir a instalação do GeometryFriends, copiar ambos os ficheiros dentro da pasta 'src/Agents' submetida e colá-los na pasta 'Release/Agents' do diretório para onde foi instalado o GeometryFriends.

Correr o jogo com os agentes implementados:

1. Executar 'GeometryFriends.exe'.
2. Premir a tecla 'Esc' quando é pedido para ligar o *Wiimote*.
3. Navegar com as teclas 'seta para cima' e 'seta para baixo' no menu até a opção 'Options'. De seguida, premir 'Enter'.
4. Dentro do menu 'Options', navegar até à opção 'Agents Implementation' e premir 'Enter' até a opção 'Geometry Friends Agents' estar selecionada.
5. Clicar na tecla 'seta para baixo' e escolher 'Save and Exit'.

Para jogar um nível:

1. Selecionar no menu inicial a opção 'Agents Only'.
2. Com o título do mundo selecionado com uma cor vermelha (o título do mundo é algo semelhante a "GF 2017 Circle Public"), navegar com as setas esquerda e direita até o mundo "GF 2017 Cooperative Public" estar selecionado.
3. Premir a tecla para baixo e selecionar o nível que se pretende executar com 'Enter'.
4. Clicar em 'Start' para os agentes começarem a jogar o nível.

Visualizar o código fonte

Todo o código fonte relativo aos agentes implementados está disponível na pasta 'GeometryFriendsAgents'. Cada ficheiro com a extensão ".cs" poderá representar mais do que uma classe.