Among Us

Sistemas Multi-Agente

Ano Letivo 2020/21

Francisco Cavaco, 51105

Marta Correia, n.º 51022

Miguel Tavares, n.º 51966

Grupo 02

1. Among Us

O Among Us consiste num jogo de estratégia que pode acomodar entre 4 e 10 jogadores, dos quais 1 a 3 podem ser *imposters*, enquanto que os restantes são *crewmates*. O objetivo de cada *crewmate* é fazer um conjunto de *tasks* que lhe são atribuídas, enquanto que o *imposter* tem de os ir matando (existindo um tempo de espera para que o mesmo *imposter* possa fazer outra vítima). Ao *imposter* é dada também uma lista de *tasks*, as quais ele pode fingir fazer. Tanto um *crewmate*, como um *imposter* têm uma distância de visão na qual conseguem ver, sendo tipicamente a do *imposter* maior que a do *crewmate*.

O *imposter* pode também acionar sabotagens: o Reactor; o Oxigénio; as Luzes; e Comunicações. No caso da sabotagem Oxigénio e Reactor, existe um tempo limite para que sejam arranjadas. Tal como para matar, existe também um tempo de espera entre a chamada de cada sabotagem para que o *imposter a* possa acionar novamente.

Sempre que um jogador encontra um corpo, pode decidir reportá-lo e uma *meeting* é convocada. Na *meeting*, cada jogador, que ainda estiver vivo, pode trocar informações e discutir quem suspeitam ser *imposter.* No final, cada um pode votar em quem acha que é o *imposter* ou escolher dar *Skip* (não votar em ninguém). O jogador com mais votos, é ejetado.

Quando um jogador é morto ou ejetado, torna-se um fantasma. No caso dos *crewmates*, estes podem continuar a fazer as suas *tasks.* Os *imposters* podem continuar a fazer sabotagens para ajudar outros *imposters* que ainda estejam vivos.

O jogo termina e os *crewmates* ganham se terminarem todas as suas tarefas ou se todos os *imposters* forem ejetados. O jogo termina e os *imposters* ganham se o número de *imposters* vivos for igual ao número de *crewmates* vivos ou se as sabotagens Reactor e Oxigénio não forem arranjadas a tempo.

1.1. Adaptações na Implementação

Em relação à nossa implementação, fizemos algumas adaptações às regras, de forma não só a reduzir a complexidade do programa e sincronização de agentes, como também para simplificar toda a lógica que o próprio jogo envolve e que não era necessariamente relevante para o objetivo do projeto.

Primeiro, alterámos o mapa para que não existissem paredes dentro do próprio tabuleiro, uma vez que o objetivo principal do projeto era a comunicação entre agentes e aplicação de *behaviors*. Isto permitiu simplificar os deslocamentos de cada agente, sem existir demasiada procura.

No mapará, para representar os *crewmates* foi usado o caracter "C", para os *imposters* o "I", para fantasma o "G" e para corpos o "B", cada um com a respetiva cor daquele agente. Para as *tasks* foi utilizada a letra "T", para o Reactor o "R", para o Oxigénio o "O", para as Luzes o "L", e para o ponto de partida o caracter "o".

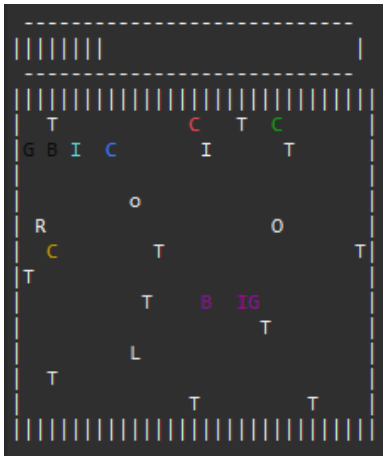

Figura 1 – Mapa com os agentes e a barra de tarefas já feitas

Na nossa implementação, apenas os *crewmates* reportam corpos e não incluímos a sabotagem de Comunicações.

Nas *meetings*, quando ocorre a votação, não incluímos a opção de *Skip,* tendo que todos os agentes votar antes da *meeting* terminar.

Quando um *imposter* é ejetado, na nossa implementação não o colocamos como fantasma a ajudar os outros *imposters*, mas apenas terminamos o agente.

2. Agentes

2.1. Game

O agente *Game* é quem gere o jogo, sendo responsável por criar os agentes, no seu *setup*, atribuindo-lhes 4 *tasks,* e imprimir o mapa e a barra de tarefas. É também responsável por imprimir as mensagens transmitidas entre agentes durante as *meetings* e terminar o jogo, com exceção de quando termina por uma sabotagem explodir.

Este agente é constituído por três *behaviours*: um *CyclicBehaviour*, que verifica as condições de término do jogo e, quando estas acontecem, avisa os outros agentes que o jogo terminou e finaliza; um *TickerBehaviour*, responsável por imprimir o mapa e a barra de tarefas, a cada segundo; um *FSMBehaviour*, que representa os vários estados do jogo, sendo eles *Playing*, *Meeting*, *Emergency* ou *Over*.

Relativamente aos estados do *FSMBehaviour*, o estado *Playing* itera sobre si mesmo até receber uma mensagem que um corpo foi reportado, mudando o estado para *Meeting*, ou que uma emergência foi acionada, mudando o estado para *Emergency*. No estado *Meeting,* este agente recebe a sua própria mensagem, de quem e onde é que ocorreu a reportagem de corpo, depois recebe as mensagens com informação relevante de cada agente e imprimi-as, e por fim, recebe os votos de cada agente e calcula o agente a ser ejetado. No final da *Meeting,* volta para o estado de *Playing.* No estado *Emergency,* o agente espera que um corpo seja reportado (indo para *Meeting*), a emergência tenha sido arranjada (indo para *Playing)* ou de receber uma mensagem que o jogo terminou porque a emergência rebentou (indo para *Over).* No estado *Over,* o agente interrompe os seus *behaviors* e termina.

2.2. Crewmate

Para o agente *Crewmate*, fizemos duas classes *Crewmate* e *OtherCrewmate*, que apenas diferem no seu comportamento quando deixam de ter *tasks* para fazer. A primeira, desloca-se para uma nova *task*, de forma a maximizar a procura por corpos, enquanto que o agente *OtherCrewmate* move-se de forma randomizada.

O agente *Crewmate*, é responsável por representar o comportamento descrito acima para um *crewmate,* fazendo as *tasks* que lhe são atribuídas e reportando corpos. Este agente é constituído por dois *behaviours*: um *behaviour FSMBehaviour*, que pode estar nos estados *Playing*, *DoingTask*, *Meeting*, *Emergency* e *Over*; e um *CyclicBehaviour*, que recebe mensagens e muda a variável *states* de forma a que o agente possa saber quando mudar de estado no *FSMBehaviour* (excepto nas *meetings,* em que o *FSMBehaviour* é que trata das mensagens).

Sobre o *behaviour FMSBehaviour*, no estado *Playing* o agente recolhe informação sobre os agentes que consegue ver e o sítio do mapa onde se encontra. O local é ditado pela *task* ao qual está mais próximo. Depois de recolher informação, verifica se existem corpos a serem reportados e, se não, procura fazer as suas *tasks*. Se já estiver na posição para fazer uma *task,* passa ao estado *DoingTask*. Se não, muda a sua posição em direção à *task* mais próxima. Se já não tiver *tasks,* move-se como foi explicado acima de acordo com o tipo de *Crewmate* que é.

No *DoingTask,* o agente itera 3 vezes sobre o estado, até terminar a tarefa. Depois de terminada a tarefa, volta para o estado *Playing.* Durante este estado, também é verificado se não ocorreu, entretanto, uma emergência, uma *meeting* ou se o jogo acabou.

No estado *Meeting*, é recebido quem morreu e onde ocorreu a morte para depois ser filtrada a informação recolhida durante o *Playing* de forma a obter apenas informação que tenha sido recolhida no local do crime. Decidimos ter em conta apenas esta informação de forma a simplificar a decisão e análise de dados por parte de cada agente, sendo que outras abordagens teriam sido possíveis nesta fase. Depois de processada, a informação é enviada para os outros agentes, e o próprio recebe também informação de todos os outros. Com essa informação recebida, e com a sua, é feita uma análise dos dados, que adiciona ou retira pontos a cada suspeito. No final, envia para o *Game* a sua decisão, que será o agente que tinha mais pontos de suspeita.

No estado *Emergency*, se já tiver morrido, o *crewmate* continua a fazer as suas *tasks* como fantasma. Se não, este agente vai tentar arranjar a emergência, reportando um corpo se o encontrar pelo caminho. Arranjar a emergência passa por enviar uma mensagem à emergência em como está arranjada, sendo que cada sabotagem é também um agente na nossa implementação.

O estado *Over,* tal como no *Game*, serve para interromper os *behaviours* e terminar o agente.

2.3. Imposter

O agente *Imposter*, é responsável por representar o comportamento descrito acima para um *imposter,* matando *crewmates* e fingindo fazer as *tasks* que lhe são atribuídas (que não contribuem para as *tasks* já feitas). Este agente é constituído por dois *behaviours*, *FSMBehaviour* e *CyclicBehaviour*, que funcionam de forma semelhante ao agente *Crewmate*.

No estado *Playing*, o *Imposter* também recolhe informações sobre o que está a ver para depois informar os outros agentes nas *meetings*. Na nossa implementação, poderíamos ter colocado mentiras de forma a ilibar os *imposters*, mas escolhemos não o fazer de forma a não ser impossível para os *crewmates* ganharem. Desta forma, toda a informação que é passada nas reuniões, é verdadeira, sendo mais fácil chegar a uma dedução.

Após recolher informação, o agente verifica se alguém está ao seu alcance para matar e se não existem *crewmates* à volta a ver. Se tiver mais que um agente que pode matar, o *imposter* aciona a sabotagem de luzes e mata. Se tiver apenas uma pessoa que pode matar, e não existirem *crewmates* que conseguem ver o ato, o *imposter* mata esse *crewmate* e chama o *reactor* ou o oxigénio, dependendo do lado do mapa em que se encontra. Se estiver a ver apenas uma pessoa, mas não a conseguir matar, vai atrás dela. Se nenhuma destas condições se cumprirem, o *imposter* move-se para uma das *tasks* que tem de fingir fazer ou tenta procurar outra *task* de forma a otimizar encontrar *crewmates*, uma vez que estes estão, com maior probabilidade, perto de *tasks*.

O estado *DoingTask* funciona da mesma forma que o do *Crewmate*, com exceção de que se estiver a meio de uma *task* e a situação o permitir matar um *crewmate*, o *imposter* para de fazer a *task* para matar. O estado *Meeting* é também igual ao do *Crewmate*, com exceção de que o *imposter* nunca suspeita outros *imposters*. No estado *Emergency,* o *imposter* também tenta arranjar a emergência, mas continua a tentar matar *crewmates*. O estado *Over* é igual ao do *Crewmate*.

2.4. Reactor, Oxygen e Ligths

Os agentes *Reactor*, *Oxygen* e *Ligths* representam as sabotagens possíveis na nossa implementação. O *Reactor* e *Oxygen* possuem dois *behaviours*: um *TickerBehaviour*, que diminui o tempo que os agentes têm para arranjar a sabotagem e envia a mensagem de termino quando o tempo se esgota; um *CyclicBheaviour* que recebe as mensagens e envia-as para os outros agentes. O agente *Lights* tem apenas o *CyclicBehaviour*, uma vez que não requer um tempo para ser arranjado.

3. Outras classes

Outras classes implementadas que são relevantes de explicar são a class *Blackboard* e *DistanceUtils*.

A primeira é um *Singleton* que serve como armazenamento de todas as informações comuns entre agentes. É esta classe que contém as posições atuais de cada agente, bem como os jogadores que ainda estão vivos e os que já morreram e o número de *tasks* já feitas, entre outras informações.

A classe *DistanceUtils* é uma classe auxiliar que trata da informação relativa a distâncias entre agentes e posições, e contém os métodos sobre o próximo movimento dos agentes quando já não existem *tasks* para fazem.