Enunciado de PCD - 2023/24
Snakes: jogo concorrente e distribuído

Luís Mota

Versão 2 *- 31 de outubro de 2023

Resumo

Pretende-se desenvolver como projeto de PCD um jogo distribuído, aqui designado Snakes, uma versão simplificada (e no plural) do popular jogo Snake[1]. Apelando ao caráter lúdico deste tipo de jogos, pretende-se assim motivar os alunos a dedicar o esforço necessário à aprendizagem dos conteúdos da cadeira.

Para bem servir os propósitos de aprendizagem, as características do jogo foram adaptadas, por forma a serem um bom cenário de aplicação de questões habituais no domínio da concorrência. Optou-se também por ter os participantes humanos a funcionar remotamente, para impor um caráter distribuído ao projeto.

1 Descrição genérica do projeto

1.1 1ª parte: concorrência e coordenação

O jogo Snakes joga-se online, entre um ou mais jogadores ligados remotamente, e algumas cobras controladas automaticamente. Cada jogador controla um elemento do jogo, representado por uma instância da classe `Snake`. O objetivo de cada jogador é capturar um prémio existente no tabuleiro, representado por um algarismo entre 1 e 9, que é o seu valor. Como consequência da captura do prémio, a cobra vai acrescentar ao seu comprimento atual o valor do prémio capturado. O crescimento da cobra ocorre gradualmente, uma posição por vez, a partir da cabeça da cobra, a cada novo

*__Nota:__ o enunciado pode ser alterado para melhorar a clareza ou adequar-se melhor às necessidades de aprendizagem. Qualquer nova versão será publicada no moodle e será feita uma notificação por email.

[1] `https://en.wikipedia.org/wiki/Snake_(video_game_genre)`

movimento. Existem também jogadores controlados automaticamente, que vão deambular pelo tabuleiro de forma a dirigir-se pelo caminho mais curto para o prémio, evitando as posições ocupadas pela própria cobra.

O objetivo de todos os jogadores (no caso dos humanos, apenas previstos na 2ª parte do projeto) é ir conquistando prémios, até capturar o valor máximo de 9. Nessa altura, acaba o jogo e ficam imóveis na posição que ocupavam.

Naturalmente, no movimento, uma cobra pode tentar ocupar uma posição já ocupada por outra cobra. Neste caso, ao contrário do habitual em jogos deste tipo, a cobra não morrerá e a ocupação dessa posição deve ficar em espera, até que a posição seja desocupada.

A movimentação dos jogadores automáticos deve orientar-se para a posição atual do prémio, não devendo, em particular, evitar as posições com outros jogadores ou obstáculos. O movimento deve, porém, evitar células ocupadas pela própria cobra.

Existirão também obstáculos no tabuleiro, colocados aleatoriamente no início do jogo, em posições que estejam livres. Estes obstáculos serão manipulados por *threads* próprias: cada obstáculo terá uma *thread* associada. Estas *threads associadas* não poderão estar todas em execução simultânea: apenas poderão estar em execução `NUM_SIMULTANEOUS_MOVING_OBSTACLES`, p.ex. no número de três. Esta limitação à execução deve ser conseguida com auxílio de uma `ThreadPool`. A ação das *threads associadas* é muito simples: após um curto período de espera (`Obstacle.OBSTACLE_MOVE_INTERVAL`), a localização do obstáculo deve ser alterada para uma posição livre do tabuleiro, escolhida aleatoriamente. Este reposicionamento do obstáculo será repetido `Obstacle.NUM_MOVES`, após o que a *thread associada* terminará. Se uma cobra tentar deslocar-se para a posição onde se encontre um obstáculo, o movimento não poderá prosseguir e a cobra ficará bloqueada. Este desbloqueio poderá acabar se o obstáculo se deslocar para outra localização.

Os prémios, representados pela classe `Goal` serão manipulados pelas cobras, que se encarregarão de relocalizar o prémio e incrementar o seu valor de cada vez que o prémio for capturado, o que será provocado pela invocação do método `Goal.captureGoal` pela cobra que capturar o prémio.

Dado o movimento relativamente pouco inteligente das cobras automáticas, é bastante provável que estas fiquem bloqueadas em obstáculos ou noutras cobras, sobretudo quando estes existirem em número elevado. Para isso existe na GUI um botão `Reset snakes' directions` que enviará um sinal às cobras (através de uma interrupção), que forçará as cobras a escolherem um novo destino, resolvendo momentaneamente o bloqueio.

1.2 2ª parte: Programação em rede

A 2ª parte apenas deve ser desenvolvida após a conclusão da primeira, depois da apresentação intermédia. Serão criados jogadores humanos remotos, que se ligarão em qualquer momento a um jogo já existente. Estes clientes remotos servirão de controlador, lendo comandos a partir do teclado que servirão para dar a direção do movimento a um jogador humano, não automático. Também terão uma interface gráfica que exibirá remotamente o estado do jogo. Estes clientes remotos serão independentes e poderão coexistir em qualquer número. Atempadamente, serão dadas mais indicações sobre esta parte do trabalho.

2 Requisitos do projeto

O projeto é uma ferramenta de aprendizagem, pelo que deve servir para adquirir conhecimentos em vários tópicos do programa de PCD, que passamos a elencar e que devem ser escrupulosamente seguidos.

Todos os mecanismos de coordenação devem ser desenvolvidos pelo próprio grupo, não devendo ser usados os equivalentes disponibilizados nas bibliotecas padrão do Java.

Colocação dos jogadores Aquando do início do jogo, os jogadores participantes, em número a definir em constante no programa, devem ser colocados numa posição aleatória na primeira coluna da esquerda, ocupando uma única célula. Caso a posição escolhida para um jogador já esteja ocupada por outro jogador colocado antes, a escolha da posição do segundo jogador deve ser repetida até ser encontrada uma célula livre.

Sincronização de acesso às secções críticas Como sempre necessário em situações de acesso concorrente a informação partilhada, devem ser identificadas as secções críticas e devidamente protegidas de interferências, usando o mecanismo de sincronização ou equivalente, que assegurará a exclusão mútua de acessos por diferentes *threads*. A sincronização deve, naturalmente, ser tão localizada quanto possível, para permitir o máximo de concorrência. Deve ser dada particular atenção ao movimento das cobras: estes envolvem as células de origem e destino. A solução ideal deve aplicar exclusão mútua a ambas as células, evitando, naturalmente, situações de bloqueio (*deadlock* ou equivalente).

Coordenação para a ocupação de células Uma cobra pode pretender ocupar uma célula transitoriamente ocupada por outra cobra ou por um obstáculo. Deve ser implementado um mecanismo de coordenação que coloque a cobra ocupante em espera, aguardando que a célula destino fique livre.

Aplicação de `ThreadPool` Os obstáculos devem ser movimentados por `threads` dedicadas: cada obstáculo deve ter um `thread` associada, da classe `ObstacleMover`. A execução destas `threads` deve ser gerida por uma `ThreadPool`: em cada momento apenas devem estar em execução simultânea um número de `threads` definido em `LocalBoard.NUM_SIMULTANEOUS_MOVING_OBSTACLES`.

Coordenação para o fim do jogo Como descrito na secção anterior, o jogo termina quando uma cobra capture o prémio com o valor 9. Quando tal acontecer,o jogo termina para todos os participantes. Para implementar esta coordenação, devem ser interrompidas todas as cobras em execução: se o jogo de facto tiver terminado, a sua execução deve terminar

Resolução de imobilização Os jogadores, na sua movimentação, poderão tentar mover-se para uma posição ocupada, quer por outra cobra quer por um obstáculo. Por vezes, estes bloqueios serão irresolúveis, por serem originados por um obstáculo que já não se vai mover, ou por outra cobra também ela bloqueada. Para evitar que o jogo fique bloqueado para sempre, deve ser feito um mecanismo, desencadeado por um botão na GUI, que interrompa todas as cobras e as leve a escolher uma célula de destino alternativa.

Ligação remota dos jogadores No sentido de aplicar os conceitos de programação distribuída, devem os jogadores humanos ser implementados como aplicações remotas, que se ligam a um servidor existente, associado ao jogo. Cada jogador remoto deve ser lançado e controlado por uma aplicação diferente e autónoma, que se pode juntar ao jogo a qualquer momento.

Mais detalhes sobre estes clientes remotos serão dadas atempadamente.

3 Detalhes de implementação

Para poder focar o desenvolvimento nas matérias relevantes de PCD, a interface gráfica já é fornecida, com todas as capacidades para representar o jogo em todos os seus momentos. Também é dada a implementação, completa ou em esqueleto conforme os casos, de quase todas as classes necessárias ao desenvolvimento do projeto.

Sempre que, durante o jogo, se proceda a alguma alteração, como p.ex. a energia ou posição de um jogador, deve ser invocado o método `Board.setChanged()`, que assegurará que a interface gráfica é atualizada.

Para assegurar a consistência dos dados, sugere-se fortemente que a localização dos jogadores seja mantida **preferencialmente** nas células mantidas no atributo `cells` da classe `Board`.

A temporização dos movimentos dos jogadores não deve ser centralizada: todos deverão respeitar um tempo de espera fixo, definido na constante `Board.PLAYER_PLAY_INTERVAL`. Assim, após efetuar o seu movimento, cada jogador deve adormecer por este período de tempo.

Para permitir a ligação de jogadores remotos, o início dos movimentos do jogo apenas deve ocorrer cerca de 10 segundos depois de este ser lançado: as threads dos jogadores tentarão colocar imediatamente o jogador na posição inicial, mas apenas iniciarão a sua movimentação normal após decorrido este tempo. Sugere-se que se coloque um `sleep` no método `run` logo após a colocação na posição inicial.

4 Fases de desenvolvimento

Nesta secção sugere-se um encadeamento de fases que permitem um desenvolvimento sustentado deste projeto. As primeiras duas fases devem imperiosamente ser cumpridas no início, pois vão constituir a entrega intercalar.

1. **Colocação inicial dos jogadores:** tratar da colocação inicial dos jogadores automáticos, segundo as indicações anteriores.

2. **Movimentação básica dos jogadores automáticos:** Os jogadores devem realizar a sua deslocação apenas a intervalos regulares. Por isso, use a classe `AutomaticSnake` para os jogadores automáticos, que devem poder funcionar enquanto processo ligeiro (*Thread*). A sua execução deve conter uma repetição regular do envio para o jogo do pedido de movimentação numa direção. O intervalo entre repetições do envio deve ser de acordo com a constante `Board.PLAYER_PLAY_INTERVAL`, que está inicialmente configurada para 100ms. Nesta fase não considere obstáculos nem outras limitações de movimentos. Sempre que for feito um movimento,deve ser desencadeada a atualização da interface gráfica, invocando o método `setChanged()`.

3. **Movimentação completa dos jogadores:** desenvolva a fase anterior, considerando as possíveis consequências dos movimentos:

(a) se o movimento for para cima de outro jogador ou obstáculo, deverá ficar em espera segundo as regras enunciadas acima. Recomenda-se, como oportunidade extra de aprendizagem, a utilização de variáveis condicionais:

(b) se o jogador atingir o prémio máximo, deve acabar a sua execução e registar este facto no estado do jogo.

4. **Resolução da imobilização no movimento:** como descrito anteriormente, se um jogador automático tentar deslocar-se para cima de um obstáculo, pelo que o movimento não será possível. Neste caso, o jogador deve bloquear, usando as habituais operações de `wait` ou `await`, conforme estejam a usar cadeados implícitos ou explícitos. Como o obstáculo poderá nunca sair desta posição, é necessário encontrar uma forma de desbloquear o movimento suspenso. Para tal, deve ser aplicada a técnica descrita na secção 2.

5. **Final do jogo:** como descrito na secção anterior, também é necessária coordenação, através de uma interrupção, para processar o final do jogo, conforme detalhadamente descrito acima.

6. **Análise de possíveis situações de bloqueio e conflito** Analise a sua resolução para o projeto e tente identificar possíveis situações de bloqueio, como p.ex. *deadlock*, *livelock* ou *starvation*. Caso identifique riscos de ocorrência destas situações, tente aplicar medidas que as impeçam. Deve ser feita uma descrição das medidas tomadas, em relatório autónomo a incluir na entrega final.

7. **Gestão dos movimentos dos obstáculos**: Aplicação da `ThreadPool` segundo indicações acima.

8. **Implementação dos jogadores humanos como aplicações remotas:** para aprofundar os conhecimentos de programação distribuída, deve, nesta última fase, ser desenvolvida uma versão remota dos jogadores, que serão controlados por um humano. Estes ligar-se-ão ao jogo através de um servidor que estará disponível num endereço e porto conhecidos. Note-se que nada impede que estes jogadores remotos coexistam no jogo com os jogadores automáticos referidos no ponto 3.

   Serão atempadamente dados mais detalhes da comunicação com os clientes remotos.

5 Entregas

Para favorecer o início atempado do desenvolvimento do projeto, existe uma entrega intermédia. Pede-se que seja feita uma apresentação das fases iniciais, referentes à versão concorrente, no turno de laboratório da semana de 20 a 24 de novembro, onde deverá ser feita uma breve apresentação do trabalho feito. Esta entrega é obrigatória e eliminatória, e feita presencialmente.

A entrega final será às 09:00 de 11.12.2023, em página no moodle a disponibilizar oportunamente. As discussões finais, de que alguns alunos poderão eventualmente ser dispensados, decorrerão nos laboratórios de 12, 13 e 14 de dezembro e, se necessário, estender-se-ão eventualmente para 15.12.2023.

O modelo da avaliação é o habitual nas cadeiras de programação: o projeto é obrigatório mas não conta diretamente para a nota, podendo apenas limitar a nota final, conforme critérios apresentados na primeira aula e constante da FUC.

6 Avaliação

Todas as fases descritas na secção 4 devem ser implementadas, pois o projeto é primordialmente uma ferramenta de aprendizagem e essas fases correspondem a partes importantes da matéria a aprender e a ser avaliada.

Caso haja lacunas menores nesta implementação, os alunos poderão mesmo assim ter aprovação com uma valoração qualitativa mais baixa.

7 Histórico de versões

1 Versão original

2 Enumeração e descrição das fases de desenvolvimento e enumeração dos requisitos do projeto. Foi retirada a indicação de que o `Goal` seria manipulado por uma `thread` própria: tal passa a ser opcional.