Relatório Projeto PCD

Introdução:

Este relatório serve para detalhar as dificuldades que o grupo teve em cada uma das fases de desenvolvimento e quais foram as soluções encontradas para essas dificuldades.

Fases de Desenvolvimento

**Colocação Inicial dos Jogadores:**

Na colocação inicial dos jogadores optou-se por seguir a sugestão dos professores usando variáveis condicionais para tornar a operação sincronizável. Desta forma garantimos que não existe mais de um jogador a tentar colocar-se sobre uma posição.

Aquando da colocação dos jogadores, é primeiro avaliado se a posição escolhida já está ocupada por um jogador e, se for esse o caso, através da variável condicional **isFull**, o jogador irá aguardar que a posição fique vazia. Assim que for notificado que a posição se encontra vazia, continua o processo de colocação.

Notámos que através desta solução existiam jogadores que não estavam a ser colocados em jogo, devido às células que os jogadores estavam a tentar colocar-se conterem jogados mortos. Para tal, implementámos uma condição que verifica se o jogador que está a ocupar a posição se encontra morto. Caso assim seja, lançamos uma **InterruptedException**, que é tratada pelo jogador e que faz com que se tente novamente colocar noutra célula aleatória. O processo só termina quando for encontrada uma célula disponível.

**Movimentação básica dos jogadores automáticos:**

A movimentação dos jogadores automáticos é efetuada por intervalos de tempos diferenciados pela energia inicial dos jogadores. O intervalo de tempo usado foi o **Game.REFRESH\_INTERVAL**, que multiplicado pela energia inicial gera 3 tempos distintos de atualização. A cada intervalo de tempo, é gerado uma direção aleatória sempre tendo em atenção às limitações do jogo.

**Movimentação completa dos jogadores:**

Para complementar a anterior fase de desenvolvimento, incluímos métodos de sincronização para garantir que jogadores distintos não comunicavam com células que já estavam a ser usadas por outros. Os métodos de sincronização escolhidos foram o **lock** e o **synchronized**.

**Movimento para cima de outro jogador:**

Quando acontece um movimento para cima de um jogador vivo, é despoletado um confronto entre ambos ganhando aquele que tem um nível de energia superior, ou, no caso das energias serem iguais, o jogador que ganha o confronto é definido aleatoriamente.

Neste confronto, o jogador derrotado muda de estado para inativo (morto) e vitorioso consome a energia do derrotado.

**Movimento para cima de um jogador morto:**

No caso de um jogador tentar-se mover para cima de uma posição onde tenha um jogador morto a ação resultante será diferenciada pelo tipo de jogador. **Jogador Humano**, neste caso nada acontece, a movimentação é dada como nula, ou seja, o jogador humano não se move para cima de o jogador morto, simplesmente ficando a aguardar nova direção. **Jogador Automático,** neste caso a movimentação gera um bloqueio utilizando o método **wait**, que faz com que a Thread fique a aguardar um desbloqueio. Este desbloqueio é tratado na fase de desenvolvimento que se segue.

**Jogador atinge o máximo nível de energia:**

Sempre que existe um confronto entre jogadores, é validado se a energia do jogador vitorioso ultrapassa ou é igual à máxima energia permitida e, caso assim seja, o jogador é um dos 3 possíveis winners do jogo interrompendo a sua função.

**Resolução da imobilização no movimento:**

Conforme comentando na fase anterior de desenvolvimento, o Jogador Automático irá ficar no método **wait**, até que seja desbloqueado. Para que este desbloqueio aconteça, criou-se uma Thread Autónoma que tem como função verificar se o movimento do jogador é efetuado no máximo em 2 segundos.

Caso o movimento do jogador demore 2 segundos, a Thread irá interromper o jogador em questão, fazendo com que o mesmo gere novo movimento.

**Final do Jogo:**

Para o final do jogo acontecer implementámos o mecanismo de **countdownlatch**, que vai aguardar por pelo menos 3 jogadores serem vitoriosos e, assim que isso aconteça, interromper todos os jogadores.

**Implementação dos jogadores humanos como aplicações remotas:**

Criado uma folder com as classes necessárias para correr o jogo com os dados minmos (sem acesso a logica do jogo, que pode ser transportado para outro pc mas decidimos manter tudo no mesmo projecto). Este vai ligar-se ao servidor que foi facultado (IP + Porto) e se o jogo é multiplayer (Permite até dois jogadores por cliente)

O Cliente recebe os player do servidor (só com a informação mínima), e o estado do jogo e envia para o servidor a direção do player de acordo com as teclas.

Do lado do servidor ao receber uma conexão cria uma thread para lidar com esse cliente. Ao criar a thread cria um player ou 2 caso seja mutiplayer. Depois ele envia a lista de players para os clientes (só com a informação mínima), envia o estado do jogo e recebe a direção para mover o player.

**Deteção de possíveis situações de bloqueio e conflito:**

Durante o projeto identificámos algumas situações de conflito que foram sendo resolvidas ao longo do mesmo. Detalhamos aqui as situações onde notámos mais dificuldade em resolver o conflito:

* Recurso Partilhado: A escolha do recurso partilhado não foi direta, nos primórdios do projeto começamos por usar o Game mas após alguma ponderação encontrámos na Cell o recurso ideal para o efeito.
* Colocação de Jogadores: Ao princípio notámos alguma dificuldade em colocar os jogadores através de variáveis condicionais, principalmente no momento de avisar os jogadores que a posição estava livre. Resolvemos ao colocar uma notificação sempre que um jogador se move com sucesso, avisando assim que deixou aquela célula livre.
* Confrontos: O conflito que nos tomou mais tempo foi o confronto entre jogadores. Ao princípio tínhamos vários jogadores bloqueados sempre que existia um confronto, para resolver, usámos o método Lock onde decidimos que sempre que um jogador se move ele irá (por esta ordem) bloquear a célula onde se encontra e só depois bloquear a célula para onde se quer mover. Se se mover com sucesso, o desbloqueio será efetuado pela ordem contrária.