Relatório 3ª fase

César Gasparini, Daniel Nunes e Artur Santos
10297630, 10297612, 10297734

November 19, 2017

Introdução

Dados e informações sobre a 3^a fase do projeto de MAC216. As implementações e features são descritas nesse documento.

1 Resumo

Nesta fase do projeto, focamos principalmente em implementar graficamente a arena e ampliar os recursos do jogo.

De forma geral, implementamos o vetor de robôs na arena, adicionamos uma arma como item coletável, implementamos a parte gráfica ao jogo, buscamos organizar melhor o código feito até então e testamos todos os recursos.

2 Implementação

2.1 Arena e Sistema

A priori, "rodamos" a arena para ser compatível com o apres, disponibilizado no enunciado, e com isso, mudamos o algoritmo que transforma a direção de uma matriz hexagonal em uma quadrada e os respectivos nomes das direções, que, agora, se chamam: NORTHEAST, EAST, SOUTHEAST, SOUTHWEST, WEST, NORTHWEST e CURRENT.

Com essa mudança, por questões técnicas, tivemos que transpor a matriz que representa a arena ao fazer a conversão. Entretanto isso não afeta significativamente o jogador, já que é apenas um detalhe da implementação na conversão.

Fizemos uma reunião para discutir alguns tópicos do projeto. Uma das pautas levantadas foi o gerador da arena, antes implementado de modo a gerar o cenário por seções de terrenos. Entramos em um consenso e decidimos então que, para melhorar a experiência do jogador, o terreno da arena deve ser gerado de modo totalmente aleatório, célula por célula, sendo menos provável que o jogador fique em desvantagem. Decidimos também que a posição do robô deve continuar sendo aleatória, mas com a condição de que ele sempre comece a partida perto de sua base, que por sua vez ficará distante da base inimiga. Chegamos a essas conclusões por meio de testes empíricos.

Outra implementação importante nesta fase foi a energia. Embora sua implementação não seja intuitiva, ela funciona da seguinte maneira: quando o robô se locomove ou ataca, ele "ganha pontos de energia", não podendo mais executar por um número determinado de rodadas. Assim, a energia é análoga a uma penalidade. Quando ataca, o robô obtém três pontos de energia, o que equivale a três rodadas sem poder executar. A quantidade de energia que ele ganha, quando se move, é determinada pelo tipo de terreno em que o movimento ocorreu, segundo o que segue abaixo:

Terreno	Cor	Energia	Rodadas sem executar
Grama	Verde	1	1
Areia	Bege	2	2
Pedra	Cinza	3	3
Gelo	branco	4	4
Água	Azul	5	5

Adicionamos um recurso extra ao jogo: uma arma. A arma é tratada, assim como o cristal, como um item coletável pelo robô. Entretanto, quando coletada, não pode ser depositada e não marca pontos ao time. Esse detalhe será revisto futuramente. Apenas duas armas são colocadas na arena, com posições aleatórias, de modo a ter uma arma relativamente mais perto de cada base. Ao coleta-la, o robô aumenta o seu dano de 10 para 30, porém, se o mesmo robô coletar duas armas, ele terá seu dano igual a 60.

Nota: A Saúde inicial do robô é igual a 100. Se o robô deu 10 de dano a um robô, significa que ele abaixou em 10 pontos a saúde deste. Quando a saúde de um robô chega a 0 pontos ou menos ele morre, ou seja, é removido da arena e não executa mais.

Agora há no jogo 10 robôs sendo executados, cinco de cada time. Definimos um número fixo de dois times por partida. Para facilitar os testes durante a produção desta fase, fizemos o método "geraProg()", uma função escrita no arquivo maq.c que gera um vetor de instruções para o robô executar. Entretanto, para facilitar a correção desse projeto, todos os robôs executam as instruções presentes no arquivo "tprog.c", gerado pelo montador e a função geraProg() foi removida.

Para melhorar a experiência do jogador e o andamento do jogo em geral, há uma pausa de um segundo a cada chamada de sistema que o robô faz.

Para um robô obter informação sobre a célula atual ou vizinha ele deve solicitar ao sistema (o jogador realiza tal ação com o código INF direção), o resultado, se for possível, será empilhado em sua pilha de dados, sem penalidades.

O jogo tem, no máximo, 500 rodadas. Caso o jogo não termine até a última rodada, o sistema acabará o jogo contando quantos pontos cada time marcou e anunciando o(s) vencedor(es).

Importante: um time recebe ponto apenas quando deposita um cristal em sua base, sendo que cada cristal depositado equivale a um ponto!

Como agora podemos ver os robôs, testamos novamente todos os recursos que implementamos no jogo e houve melhoras em alguns trechos de código, relatado melhor na seção (Testes).

2.2 Interface Gráfica

Como sugerido no enunciado desta parte do projeto, a interface gráfica do jogo foi construída e manipulada a partir da interação entre os códigos em controle.c e apres, escritos respectivamente em linguagem C e Python. De forma geral, escrevemos diversas funções em controle.c que podem ser chamadas por todos os códigos que implementam a interface controle.h e que enviam instruções para o programa em Python a partir do protocolo sugerido pelo professor.

O protocolo conta com nove instruções implementadas (até agora) e que apresentam o comportamento descrito a seguir:

Instrução e argumentos	Descrição
rob img	Carrega a imagem de um robô na arena mas não a posiciona em uma célula ainda. É chamada em controle.c antes da função move(), que trata de posicionar finalmente a imagem do robô carregada na célula definida pelo sistema.
oi oj di dj	Essa instrução é um conjunto de quatro inteiros sem um rótulo. Basicamente move o robô que está em uma célula de origem (oi,oj) para uma célula de destino nas coordenadas (di,dj)
d_cel i j terreno	Desenha uma célula na posição (i,j) com o terreno dado pelo número natural terreno, que pode variar de 0 a 6. Terreno 0: Grama; Terreno 1: Areia; Terreno 2: Pedra; Terreno 3: Gelo; Terreno 4: Água; Terreno 5: Íngreme; Terreno 6: Base
base img i j	Desenha uma base na célula (i,j). O desenho da base é uma bandeira com cor correspondente a cada time, dado em img (flag1.png ou flag2.png)
cristal i j n	Desenha um cristal na célula (i,j). O argumento n, embora não esteja sendo útil agora, pode servir posteriormente para mostrar na interface gráfica quantos cristais estão na célula.
gun i j	Desenha uma arma na célula (i,j). Duas armas podem ser coletadas por partida.
remitem i j	Remove um item (arma ou cristal) da célula (i,j) e executa um efeito sonoro para indicar a coleta.
atk	Por enquanto, somente executa um som quando é chamado, indicando que o ataque de um robô foi bem sucedido.
fim	Finaliza a construção da arena. Será utilizado para emitir notificação quando o jogo acabar.

Todas as instruções descritas acima são enviadas ao programa apres a partir de chamadas de função em arena.c.

Temos no jogo um número fixo de dois exércitos, cada um com cinco robôs. Os robôs desenhados indicam a que time pertencem. O robô pode ser do time vermelho ou do time verde. As duas imagens foram produzidas por nós e são exibidas a seguir:

Figure 1: Robô vermelho e Robô verde

Cada um dos dois exércitos possui uma base. Cada base é representada em uma célula marcada com uma bandeira de cor correspondente ao exército ao qual a base pertence. As bandeiras são exibidas a seguir:

Figure 2: Bandeira vermelha e verde

As imagens foram redimensionadas. Cada imagem foi obtida no seguinte endereço: Bandeira vermelha: `https://commons.wikimedia.org/wiki/File:Red_flag_waving.svg`e a bandeira verde: `https://commons.wikimedia.org/wiki/File:Dark_green_flag_waving.png`.

As duas armas colocadas em posições aleatórias da arena são representadas pelo mesmo arquivo de imagem.

Figure 3: Laser Gun

Até o momento, temos três arquivos de áudio que são executados ao longo do jogo: uma música de introdução, um som que indica que um item foi coletado e um som que indica que um robô atacou outro. Os três sons estão nos arquivos 'battle.wav', 'collect.wav' e 'atk.wav', respectivamente. Os arquivos utilizados foram pagos por um dos integrantes do grupo e estão disponíveis para compra em `https://www.assetstore.unity3d.com/en/#!/content/17256`

Por ora, o término de uma partida não é representado significativamente na interface gráfica. O indicador de que um jogo terminou são mensagens no terminal. Estamos estudando uma maneira de emitir na interface do jogo uma notificação de que o jogo acabou, apontando os resultados da partida.

Portanto, a interface gráfica construída segue o princípio de se enviar os códigos (instruções) do protocolo entre o programa controle.c e apres a partir de um 'pipe interno' gerenciado pela função fprintf, como sugerido pelo professor. Essa parte do projeto apresentou bons desempenhos nos testes, sendo crucial para a execução dos mesmos. Os testes descritos a seguir foram úteis também para testar a interface gráfica por si só.

3 Testes

Foram feitos diversos testes. Todos os comandos e recursos do jogo foram testado. ***Observação:*** **As instruções que usamos para testar o jogo estão disponíveis na pasta "Testes". Os testes lá utilizados são aplicados para múltiplos robôs no jogo.**

Para testar a movimentação dos robôs, primeiro deixamos a arena com um só robô que executou as instruções presentes em "mov.txt". Verificamos que todas as direções em todos os possíveis casos estavam certas. Logo, pudemos concluir que a transformação da direção da matriz hexagonal para a quadrada e que a instrução de movimento estavam certas.

Para testar o ataque, botamos dois robôs na arena e manipulamos suas posições, colocando-os lado a lado e mandando ambos coletarem um cristal e depois um atacar o outro até a morte. O resultado foi o esperado. Ambos coletaram um cristal e um deles matou o outro e o que morreu "derrubou" um cristal. Então botamos todos os robôs para se atacarem, a fim de avaliarmos eventuais erros de segmentação. Nos deparamos com alguns erros, mas consertamos e refizemos todos os testes de novo, obtendo êxito. A instrução utilizada foi "ataca.txt".

A fim de testar a coleta de cristais, colocamos um robô e um cristal na arena e manipulamos as posições de ambos. O resultado foi o esperado: o robô coletou o cristal e tocou um efeito sonoro. O arquivo com as instruções se chama "coleta.txt".

Ao testarmos o deposita, botamos um robô, um cristal e uma arma na arena e manipulamos suas posições. Executamos a instrução "deposita.txt" e o resultado foi o esperado: o robô coletou a arma e o cristal, mas só depositou o cristal.

A fim de testar a arma, colocamos dois robôs, um deles "não fazia nada". Colocamos também uma arma na arena, novamente manipulando suas posições. O robô que "fazia algo" executava as instruções presentes em "arma.txt" e o resultado foi novamente o esperado: o robô pegou a arma e matou o outro em menos ataques do que normalmente mataria (no caso, matou em 4 ataques, sem

arma mataria em 10 ataques). 4 ataques com uma arma = 4 x 30 (120) pontos de vida perdidos pelo robô que sofreu o ataque.

Para testar o fim do jogo, mudamos o numero de cristais que precisaria ser depositado na base para que um dos times vencesse. Colocamos um robô, um cristal e uma base em locais manipulados e executamos a instrução "coleta.txt". O robô coletou o cristal e depositou em sua base, acabando o jogo.

Todos os efeitos sonoros foram testados também, entretanto, não com uma instrução especifica, mas junto com todos os testes já citados.

Esses foram os principais testes realizados, houveram algumas repetições, como teste das bordas da arena (utilizando as instruções "bordas.txt"), e alguns menos significantes, os quais foram omitidos deste relatório.