Relatório da Terceira Fase

17/11/2017

César Gasparini Fernandes, 10297630

Daniel Silva Nunes, 10297612

Artur M. R. dos Santos, 10297734

**1. Resumo**

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

Em suma, implementamos o vetor de robôs na arena, adicionamos uma arma, implementamos a parte gráfica ao jogo e "limpamos" 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 nem um pouco o código e muito menos o jogador, sendo apenas um detalhe de implementação na conversão.

Fizemos uma reunião para discutir alguns tópicos do projeto, e uma das pautas era sobre o gerador da arena, antes implementado por seções de tipo de terreno aleatórias e as posições dos robôs eram totalmente aleatórias, e entramos em um consenso, decidimos então que, para melhorar a experiência do jogador, o terreno da arena deve ser gerado totalmente aleatório, célula por célula, sendo menos provável que o jogador fica em desvantagem. Decidimos também que a posição do robô deve continuar sendo aleatória, entretanto perto de sua base, a qual ficará distante da base inimiga. Chegamos à essas conclusões por meio de cálculos estatísticos e 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 energia", não podendo mais executar por um número determinado de rodadas. Ou seja, a energia é análoga a uma penalidade. A quantidade de energia que ele ganha, quando ataca, é três, equivalendo a três rodadas sem poder executar. A quantidade de energia que ele ganha, quando se move, é determinada pelo tipo de terreno que ele foi, 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

AGUA, azul , 5, 5

Adicionamos um recurso extra ao jogo, uma arma. A arma é tratada, assim como o cristal, como um coletável, entretanto, quando coletada, não pode ser depositada e não marca pontos ao time. Apenas duas armas são colocadas na arena, com posições aleatórias. Ao coleta-la, o robô aumento o seu dano de *10* para *30*, porém, se o mesmo robô coletar duas armas, ele terá seu dano igual a *30*.

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 a saúde deste. Quando a saúde de um robô chega a 0, ou menos, ele morre, ou seja, some da arena e não executa mais.

Agora, há, no jogo, 10 robôs sendo executado, dois de cada time. Para facilitar os testes, fizemos o "*geraProg()"*, uma função 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.

Para melhorar a experiência do jogador, outro recurso implementado, foi a 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 qual, se for possível, a empilhará em sua pilha de dados, sem penalidades.

O jogo tem, no máximo, 500 rodadas. Caso o jogo não termine até esta 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, retestamos todos os recursos oferecidos pelo jogo e houveram melhoras em alguns trechos de código, relatado melhor na seção *(Testes)*

**3. Testes**

Foram feitos diversos testes, testamos todos os comandos e recursos do jogo.

Observação: As instruções que usamos para testar o jogo está disponíveis na pasta "Testes".

Para testar, primeiro deixamos a arena com um só robô e o fizemos executar as instruções presentes em "mov.txt", verificamos que todas as direções em todos os possíveis casos estavam certos, logo, pudemos concluir que a transformação da direção da matriz hexagonal para a quadrada estava certa e que a instrução de movimento estava certa também.

Para testar o ataque, botamos dois robôs na arena e manipulamos suas posições, colocando-os lado a lado e mandamos 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 se não apresentava erro de segmentação, acabou apresentando um erro, mas consertamos e refizemos todos os testes de novo, dando todos certo. A instrução utilizada foi "ataca.txt".

A fim de testar a coleta de cristais, botamos um robô e um cristal na arena e manipulamos as posições de ambos, o resultado foi esperado, o robô coletou o cristal e tocou o 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", e uma arma na matriz, 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).

Para testar o fim, mudamos o numero de cristais que precisaria ser depositado na base para vencer, botamos 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.