# Relatório 3ª fase

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

November 18, 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: $https://commons.wikimedia.org/wiki/File:Red_flag_waving.svg$ Bandeira verde: $https://commons.wikimedia.org/wiki/File:Dark_green_flag_waving.png$