Game of Thrones

Caio Matheus, Christian Luis, Fillype Nascimento,
Paulo Borges, José Fortes, Rodrigo L. Rocha, Natalia O. Borges

Prof. Dr. Marcos F. Caetano
CIC116319 Estrutudas de Dados, Turma A.
Departamento de Ciência da Computação
Universidade de Brasília

6 de Novembro de 2017

1 Introdução

A batalha pelo Trono de Ferro está acirrando-se cada vez mais. Em cada reino existem alguns que se consideram dignos e reais donos do trono. **Você é um desses!** Mas a guerra está longe do fim. Sempre há mais alguma batalha a ser enfrentada e o seu papel será implementar um torneio pelo

Trono de Ferro. Com persoangens pré-definidos, você terá que lutar contra alguns personagens e vencer o torneio para se tornar o dono do Trono de Ferro e o líder dos sete reinos.

2 Objetivos

A representação dos torneiros será feita utilizando a estrutura de dados do tipo árvore binária. Nesta estrutura, cada nó representará um personagem. O combate acontecerá através da comparação entre as habilidades de dois personagens, sendo o vencedor aquele que tiver maior pontuação. O vencedor do torneio será aquele que conseguir vencer todos os combates e se tornará, enfim, o dono do tão sonhado Trono de Ferro.

2.1 Árvore Binária - Torneio

A ordem dos combates que fazem parte do torneio é definida pela árvore do tipo binária, representada pela Figura 1. Conforme pode ser observado, a árvore é formada por 4 níveis (do nível 0 ao 4). Nesta estrutura, inicialmente, os galhos representam a ordem em que os combates serão realizados e os nós do tipo folha representam os personagens. Neste caso, teremos 16 personagens diferentes que participarão do torneio. Ao final do torneio, o nó raiz representará o personagem vencedor.

Cada nó da árvore é do tipo `Character *` (vide Seção 2.2). No início do jogo, apenas os nós-folha estarão preenchidos com os personagens que participarão do torneio. As batalhas do torneio serão realizadas entre 2 nós filhos de um mesmo nó-pai. O vencedor de uma batalha passa a ocupar a posição de seu nó-pai e "sobe"na árvore para a próxima luta, onde enfrentará o outro nó-filho de mesmo pai. Este procedimento será repetido até que o vencedor ocupe a posição do nó-raíz.

Os vencedores do 1º combate serão promovidos para o nível 3 da árvore (*round* 1, Figura 1); os vencedores do 2º combate irão para o nível 2 (*round* 2, Figura 1); os vencedores do 3º combate irão para o nível 1 (*round* 3, Figura 1); por fim, o vencedore do 4º combate irá para o nível 0 (*vencedor*, Figura 1) e vence o Torneio.

2.2 Personagens

A árvore binária que representa o torneio é formada pela estrutura do tipo `t_node`, conforme apresentado pela Figura 2. Cada elemento `t_node`, é formado por três ponteiros. Os ponteiros `left` e `right` contém endereços

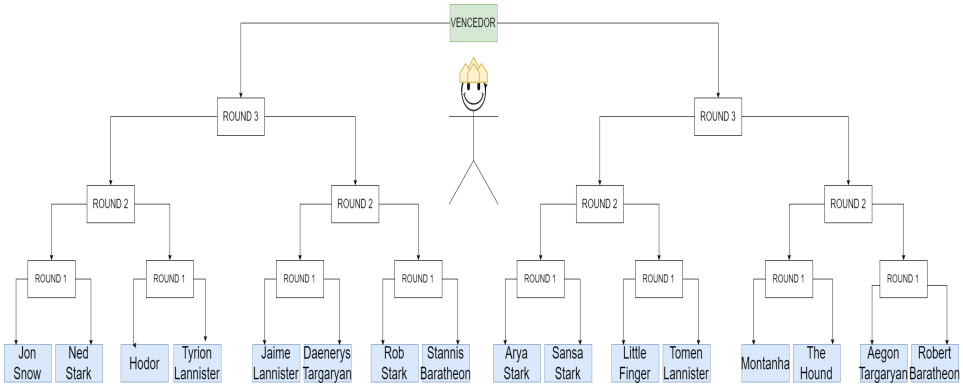

Figura 1: Árvore binária que representa um torneio.

para os elementos da esquerda e direita, respectivamente, caso eles existam. Se o nó representado for do tipo folha, os ponteiros `left` e `right` estarão apontados para `NULL`. O ponteiro `character` contém o endereço para estrutura do tipo `Character`, conforme definido pela Figura 3. Caso um nó-galho ainda não armazene um personagem (*round* 1, *round* 2, *round* 3 ou *vencedor*), o ponteiro `character` apontará para `NULL`.

Os personagens serão representados pela estrutura do tipo `Character` (Figura 3). Cada elemento de `Character` representa um atributo do personagem. Os atributos inteiros (*agility*, *strength*, *inteligence* e *health*) armazenarão valores entre 0 e 100. Os atributos *name* e *house* são nominais e imutáveis.

2.3 Escolhendo os 16 personagens

Para inicialização do jogo, será fornecido um arquivo de entrada (**personagens.txt**) contendo diversos personagens. O seu programa deverá ler este arquivo e

```
1    typedef struct node {
2        Character* character;
3        struct node* left;
4        struct node* right;
5    } t_node;
6
```

Figura 2: Definição da estrutura nó que forma a árvore binária que representa o torneio.

```c
typedef struct {
    char* name;
    char* house;
    int agility;       // 0 a 100
    int strength;      // 0 a 100
    int intelligence; // 0 a 100
    int health;        // 0 a 100
} Character;

```

Figura 3: Definição da estrutura `Character` que representa um personagem.

escolher 16 personagens, de forma aleatória, e armazená-los em memória em uma **lista duplamente encadeada**.

Após armazenar os 16 personagens na lista, você poderá escolher um desses 16 personagens para batalhar por você. Na tela de escolha de personagem, na qual o jogador escolherá o seu combatente para o torneio, o usuário poderá ver apenas **um dos quatro atributos de cada personagem**. A escolha do atributo que estará visível deverá ser feita de maneira aleatória e os personagens deverão estar listados por ordem em que apareceram na lista duplamente encadeada. É importante destacar que na relação de personagens que será apresentada ao jogador, as informações de nome (*name*) e casa (*house*) serão omitidas. A Figura 6 apresenta um exemplo de como esta relação deverá ser apresentada ao jogador (Seção 3.2).

2.4 Arquivo de Entrada

A árvore binária será formada com base no arquivo `personagens.txt`. Este é um arquivo de texto simples, sem formatação, que apresenta a relação de personagens que será utilizada na elaboração do torneio. Conforme definição, a árvore binária é formada, inicialmente, por 16 personagens distintos. Contudo, o arquivo `personagens.txt` pode conter uma lista muito maior de personagens possíveis. Cada linha do arquivo representa um personagem distinto, que forma a base de personagens possíveis a serem utilizados no jogo. No arquivo, cada personagem é descrito da seguinte forma:

```
Jon Snow, Stark, 30, 47, 63, 70
Stannis Baratheon, Baratheon, 40, 70, 100
```

Onde, os campos listados por cada linha, segundo a estrutura `Character` (Figura 3), apresentam os seguintes significados:

```
name, house, agility, strength, intelligence, health
```

2.5 Construindo a Árvore Binária

Conforme descrição apresentada na Seção 2.3, os 16 personagens selecionados de forma aleatória, a partir do arquivo `personagens.txt`, deverão ser armazenados em memória em uma lista duplamente encadeada. Cada elemento da lista armazenará um personagem, representado no formato `Character` (Figura 3). Esta estrutura será utilizada para a criação da árvore binária.

Conforme descrição apresentada na Seção 2.7, a função `tree_create()` retorna um ponteiro do tipo `t_node` que aponta para o nó raiz da árvore binária, cujos os nós contém o ponteiro `character` apontando para `NULL`. Este é o estado inicial do processo de construção da árvore binária. O processo de construção da árvore continua, envolvendo a utilização da lista duplamente encadeada (devidamente inicializada) e a árvore binária retornada pela função `tree_create()`. Neste ponto, o ponteiro `character`, da estrutura do tipo `t_node`, de cada **nó folha** da árvore binária, precisa ser apontado para seu respectivo personagem (`Character`) armazenado na lista duplamente encadeada.

Observando-se a Figura 1, o resultado final do procedimento descrito são os nós folhas (em azul) apresentados pela figura.

2.6 Combate

A mecânica do combate entre personagens funciona como um *super-trunfo*: você escolhe um atributo (*agility*, *strength*, *inteligence* ou *health*) para atacar seu oponente. Vence o personagem que possuir o maior valor do atributo escolhido. Durante o combate, deverá ser mostrado ao jogador todas as informações do seu personagem. Com base nesta informação, o usuário poderá escolher qual atributo irá utilizar para atacar o seu oponente.

Os demais *rounds* do jogo serão controlados pelo "computador". Isto é, o seu programa sorteará de forma aleatória os atributos para que os demais personagens lutem, definindo assim o vencedor de cada *round*. Seguindo a definição de ordem apresentada pela árvore binária, o personagem vencedor subirá para o próximo nível do torneio e enfrentará o vencedor do "round vizinho". Por exemplo, observando-se a Figura 1, o vencedor do combate entre *Jon Snow* e *Ned Stark* será referenciado (`character`, linha 2, Figura 2) no nó pai denominado *round* 1 (Figura 1).

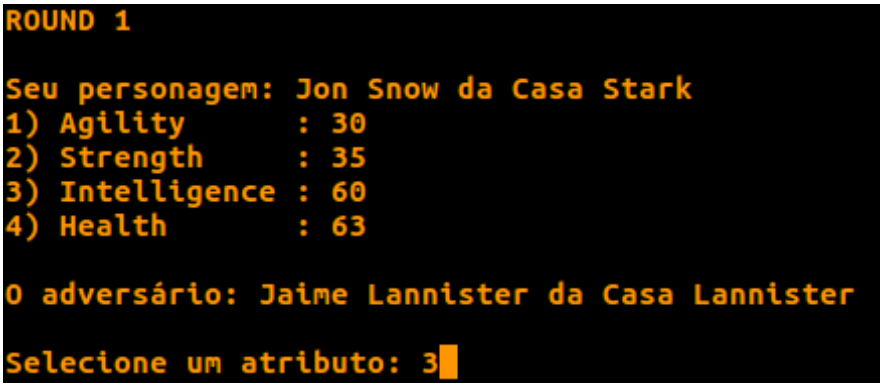

Figura 4: Tela do Combate, *round* 1.

A Figura 4 apresenta um exemplo de como a tela de batalha que deverá ser apresentada a cada combate realizado. Conforme pode ser observado, a informação do *round* em que o combate está sendo realizado precisa ser apresentada. Em seguida, as informações do seu personagem escolhido (Seção 2.3) serão listadas, bem como, as do seu adversário[1]. Note que, do adversário selecionado, somente as informações de nome e de casa são apresentadas. Por fim, é apresentada a opção de escolha de qual habilitade será utilizada no combate. No exemplo apresentado pela Figura 4, a habilidade escolhida foi *intelligence* (opção 3).

Observação: Como regra importante do jogo, NÃO É POSSÍVEL ESCOLHER PARA UM *ROUND* O MESMO ATRIBUTO UTILIZADO NO *ROUND* ANTERIOR. A ideia aqui é que um mesmo atributo não pode ser utilizado por dois *rounds* consecutivos, uma vez que é necessário um *round* de intervalo para o seu reestabelecimento. O valor do atributo usado no *round* anterior deverá ser representado como inutilizável no *round* seguinte (ver Figura 9); entretanto, o mesmo deve ser disponibilizado novamente após passado o *round* de reestabelecimento requerido.

2.7 Assinatura das Funções a Serem Implementadas

Nesta seção serão relacionadas as assinaturas da funções que deverão ser implementadas neste trabalho.

- `t_node* node_create();`

[1] É importante ressaltar que a ordem das batalhas é definida pela árvore binária.

  - retorna endereço para estrutura do tipo `t_node` alocada dinamicamente. Os ponteiros `character`, `left` e `right` são inicializados com o valor `NULL`;

- `Character* character_create(char* _name, char* _house, int _agility, int _strenght, int _intelligence, int _health);`
  - aloca dinamicamente memória para estrutura do tipo `Character`. Inicializa **POR CÓPIA** os atributos `name`, `house`, `agility`, `strength`, `intelligence` e `health`, utilizando, respectivamente, os parâmetros `_name`, `_house`, `_agility`, `_strength`, `_intelligence` e `_health`. Ao final, a função retorna o endereço para estrutura `Character` alocada;

- `void character_free(Character* character);`
  - libera a memória alocada e referenciada por `character`;

- `t_node* tree_create();`
  - retorna o endereço para o nó raiz de uma da árvore binária completa de quatro níveis (Figura 1). **TODOS** os nós das árvores apresentam o atributo `character` apontado para `NULL`.

- `void tree_free(t_node* tree);`
  - remove de forma recursiva todos os nós presentes da árvore. A memória referente ao atributo `character` também deve ser liberada;

- `Character* fight(Character* fighter_one, Character* fighter_two, int atribute);`
  - Compara o valor do atributo definido por `atribute` do `fighter_one` e do `fighter_two`, retornando o ponteiro para o personagem vencedor. Em caso de empate, o ponteiro para `fighter_one` deverá ser retornado;

- `void tree_print_preorder(t_node* root);`
  - percore a árvore binária em pré-ordem, imprimindo os personagens referenciados em `character`;

3 Funcionamento do programa

Nesta seção serão apresentadas algumas telas de funcionamento da mecânica do jogo *Game of Thrones*.

3.1 Tela inicial

Inicialemnte, o jogo deverá apresentar uma tela inicial contendo as opções definidas pela Figura 5. Escolhendo a opção ***Start New Game*** o usuário será direcionado para a tela de **Escolha de Personagem** (Seção 3.2). A opção ***Quit Game*** faz o programa finalizar a sua execução.

Figura 5: Tela inicial do jogo *Game of Thrones*.

3.2 Escolha de personagem

Nesta tela serão apresentados ao jogador os **16 personagens**, escolhidos de forma aleatória do arquivo **personagens.txt** e armazenados em memória na estrutura de lista duplamente encadeada. O jogador poderá escolher um dos personagens relacionados. **É importante destacar que o nome e a casa de cada personagem deverão ser omitidos e apenas um atributo por personagem deverá ser exibido ao jogador**, conforme apresentado pela Figura 6. A escolha de qual atributo a ser apresentado é feito de forma aleatória. Após feita a sua escolha, o jogador irá para o campo de batalha!

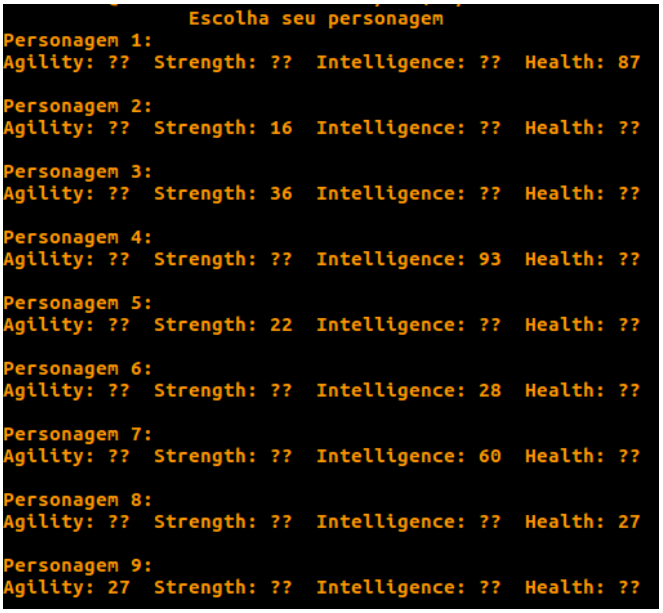

Figura 6: Tela para escolha do personagem.

Observação: Por questões de limitação de espaço, a Figura 6 apresenta somente 9 personagens. Contudo, é mandatória a exibição de 16 personagens, conforme especificação.

3.3 Mecânica de Funcionamento do Torneio

A Figura 7 apresenta um exemplo de como a tela de batalha que deverá ser apresentada a cada combate realizado. Conforme pode ser observado, a informação do *round* em que o combate está sendo realizado precisa ser apresentada. Em seguida, as informações do seu personagem escolhido (Seção 2.3) serão listadas, bem como, as do seu adversário[2]. Note que, do adversário selecionado, somente as informações de nome e de casa são apresentadas. Por fim, é apresentada a opção de escolha de qual habilitade será utilizada no combate. No exemplo apresentado pela Figura 7, a habilidade escolhida foi *intelligence* (opção 3).

[2]É importante ressaltar que a ordem das batalhas é definida pela árvore binária.

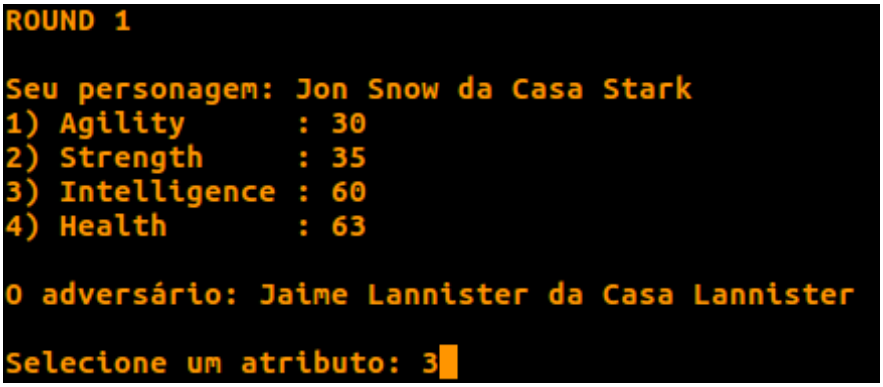

Figura 7: Tela do combate, no *round* 1. O atributo 3 (*Intelligence*) foi escolhido para a luta.

Finalizado o combate, deverá ser mostrado ao jogador o seu resultado:

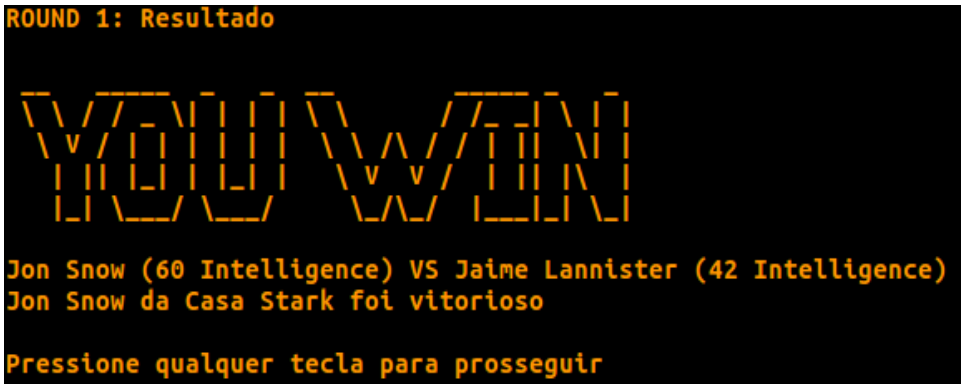

Figura 8: Tela de vitória do combate, *round* 1.

Conforme definição (Seção 2.6), no próximo *round*, o jogador não pode selecionar o mesmo atributo utilizado no combate anterior. A Figura 9 apresenta esta restrição.

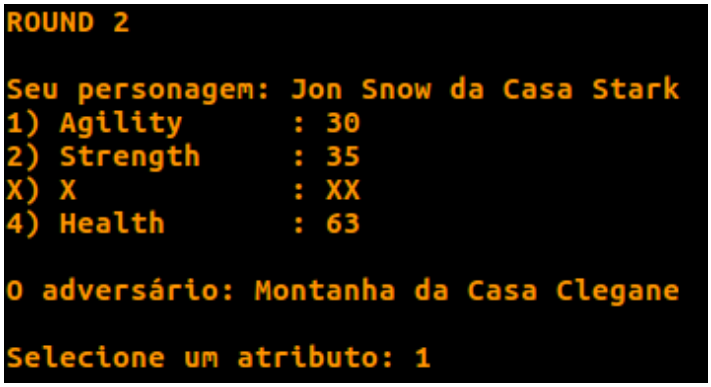

Figura 9: Tela do combate, no *round* 2. O atributo 1 (*agility*) foi escolhido para a luta.

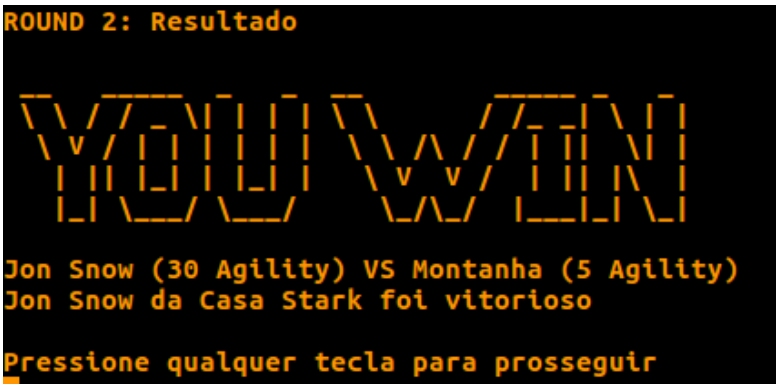

Figura 10: Tela de vitória do combate, no *round* 2.

De acordo com Figura 11, no 3^o *round*, o atributo que não pode ser escolhido é o 1 (*agility*), pois foi escolhido no combate anterior. Entretando, o atributo 3 (*intelligence*) foi utilizado no *round* 1 já pode ser reutilizado.

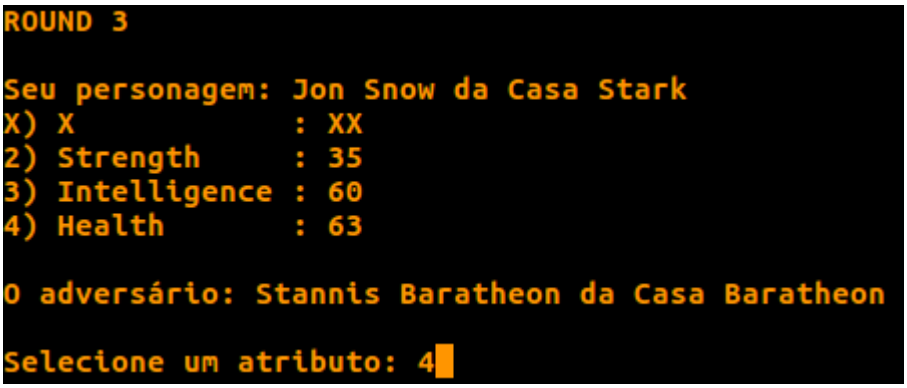

Figura 11: Tela do combate, no *round* 3. O atributo 4 (*health*) foi escolhido para a luta.

A Figura 12 apresenta a tela quando o valor do atributo escolhido pelo jogador é menor que o valor do seu oponente.

A partir da tela apresentada pela Figura 12, caso o jogador selecione a opção 1, o programa deve retornar para a tela inicial. Caso seja selecionada a opção 2, o programa deve encerrar sua execução. Além disso, devem ser mostradas todas as batalhas que ocorreram **até a partida em que o usuário morreu**, no caso do exemplo foram mostradas todas as batalhas até o 3^o *round* (no qual o jogador morre).

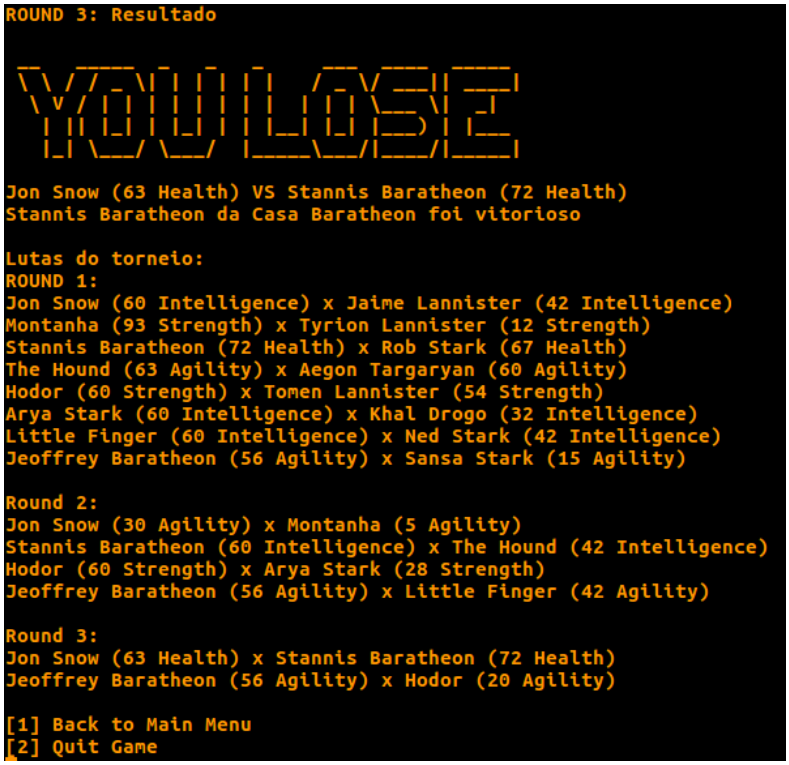

Figura 12: Tela de derrota, no *round* 3. O jogador tem a opção de voltar ao menu principal (1) ou sair do jogo (2).

Mas é claro que não vamos deixar nosso querido Jon Snow morrer nesse exemplo. Digamos que ele tenha vencido o combate anterior e tenha ido para o ultimo *round*. Neste caso, A Figura 13 apresenta o resultado desta continuação.

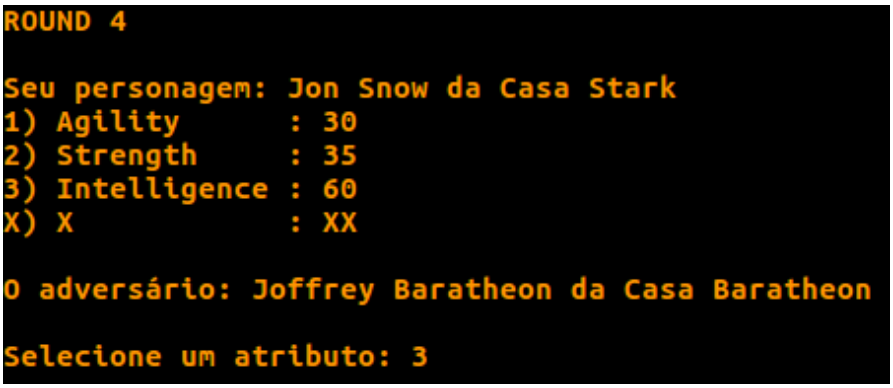

Figura 13: Tela do combate, no *round* 4. O atributo 3 (*intelligence*) foi escolhido para a luta.

Caso o usuário vença a batalha final, **ele será o rei dos 7 reinos**. Nessa tela (Figura 14) ele tem a opção de voltar ao menu inicial (1) ou sair do jogo (2). Note que será exibido ao jogador todas as lutas que ocorreram no torneio até a vitória, com isso, existem apenas duas condições em que as lutas do torneio aparecem: quando o jogador ganha o jogo e quando ele perde.

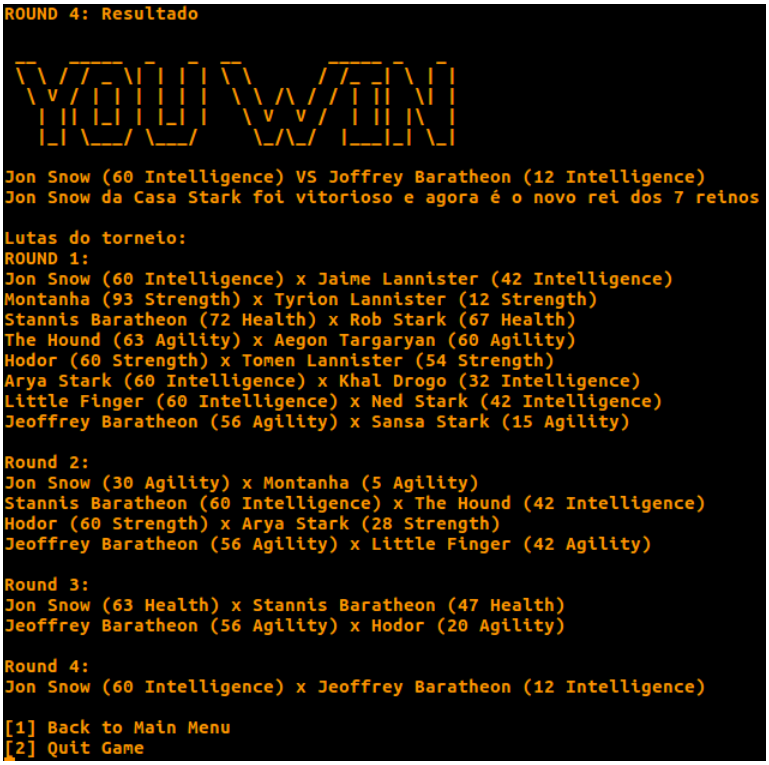

Figura 14: Tela de vitória de batalha, no Round 4 (final).

4 Avaliação

Este projeto deverá OBRIGATORIAMENTE ser feito em dupla. A avaliação do projeto será composta por duas partes.

4.1 Código (70%)

- (10%) Código modularizado (makefile) e documentado;

- (20%) Implementação da estrutura arvore binária;
- (10%) Implementação da estrutura lista duplamente encadeada;
- (20%) Implementação dos combates;
- (5%) Recuperação das informações contidas no arquivo de entrada (`personagens.txt`);
- (5%) Implementação de forma correta das funções definidas na Seção 2.7.

4.2 Relatório (30%)

O relatório a ser entregue deve estar OBRIGATORIAMENTE no formato PDF. Ele deve conter as seguintes informações:

- Documentação doxygen
- Descrição de como os problemas foram solucionados e implementados;
- Toda informação necessária para compilação e execução do seu programa.

5 Observações

As seguintes observações deverão ser consideradas pelos alunos que forem fazer o trabalho:

1. O trabalho deverá ser feito (OBRIGATORIAMENTE) em DUPLA;
    - Uma dupla é formada por dois alunos;
    - Trabalhos individuais ou em grupo não serão aceitos;
2. Para ser considerado entregue, a dupla fará uma apresentação do projeto implementado. No fórum da disciplina serão publicados os horários disponíveis para a apresentação dos trabalhos;
    - **Trabalhos não apresentados ganharão nota zero**;
    - **Ambos os alunos da dupla** deverão ter o conhecimento de **TODO o trabalho**. Ou seja, ambos os alunos da dupla deverão ter condições de responder a quaisquer questionamento referente a solução implementada e a documentação feita;

3. Utilize nomes coerentes e auto-explicativos para variáveis, funções e arquivos;

4. Todo arquivo deve conter um cabeçalho explicativo;

5. Seu código deve estar todo comentado;

6. Identação faz parte o código;

    - Será descontado 1,0 ponto do trabalho que apresentar código não identado;

7. Serão descontados pontos dos trabalhos que apresentarem vazamento de memória (utilize o *valgrind* para verificação);

8. O trabalho deve rodar **OBRIGATORIAMENTE** na plataforma GNU/Linux;

    - Trabalhos que não rodarem na plataforma GNU/Linux levarão nota zero;

9. Deve ser utilizado **OBRIGATORIAMENTE** a sintaxe ANSI C;

10. No relatório do trabalho deve conter as instruções para compilação do código e a relação de todas as bibliotecas utilizadas;

    - A nota do trabalho que não compila é zero. É sua responsabilidade se certificar que o código submetido está correto e compila na plataforma GNU/Linux;

11. O relatório deverá ser entregue **OBRIGATORIAMENTE** no formato de arquivo PDF;

12. Códigos copiados (entre alunos, retirados da Internet ou do respositório da disciplina) serão considerados "cola"e todos os alunos envolvidos ganharão nota zero;

13. Dúvidas sobre o trabalho deverão ser tiradas **NO FÓRUM DE DÚVIDAS DO AMBIENTE APRENDER**. Desta forma, dúvidas comuns e esclarecimentos poderão ser respondidos uma única vez para toda a turma.