MC322/MC336
Segundo semestre de 2015

Trabalho 1

Professor: Fábio Luiz Usberti (fusberti@ic.unicamp.br)
PED: Rafael Arakaki (rafaelkendyarakaki@gmail.com)

---

1 Objetivo

Estudo e implementação dos conceitos de programação orientada a objetos, abordados em aula, para uma aplicação na área de jogos.

2 Regras do Jogo

Chamaremos o nosso jogo de **LaMa** (**La**caios e **Ma**gias), as regras do jogo são:

- Cada jogador inicia com um Herói com trinta pontos de vida. Vence o jogador que reduzir a vida do Herói do oponente a zero.
- O baralho possui trinta cartas, sendo 22 Lacaios e 8 Magias.
- O primeiro jogador inicia com 3 cartas, enquanto o segundo jogador inicia o jogo com 4 cartas.
- Todo o início de turno cada jogador compra uma carta do baralho, de maneira aleatória.
- No turno i, cada jogador possui $\min(i, 10)$ de mana para ser utilizado em uso de cartas ou poder heróico, não podendo ultrapassar este valor. O limite de mana é acrescido até o décimo turno e então não aumenta mais. **Exceção**: O segundo jogador possui dois de mana no primeiro turno ao invés de um de mana.
- Existem dois tipos de carta: Lacaios e Magias.
  - Lacaios são baixados à mesa e não podem atacar no turno em que são baixados, apenas no turno seguinte (se ainda estiverem vivos!).
  - Magias possuem efeitos imediatos, ao serem utilizadas causam dano na vida de um único alvo inimigo ou em todos os alvos inimigos (efeito em área). As magias de alvo podem ter como alvo o Herói do oponente ou um dos Lacaios na mesa do oponente. As magias de efeito em área causam dano nos Lacaios do oponente e no Herói do oponente.
- O jogador pode utilizar, uma vez por turno, o poder heróico de seu Herói ao custo de duas unidades de mana. O poder heróico faz o Herói atacar um alvo qualquer com um de dano. Se o alvo do poder heróico for um Lacaio, o Herói que atacou receberá de dano o ataque do Lacaio alvo.
- Cada Lacaio pode escolher atacar um alvo por turno. Os possíveis alvos são: o Herói do oponente e os Lacaios em mesa do oponente. Se o Lacaio **x** atacar o Lacaio **y**, o Lacaio **y** tem sua vida diminuída pelo poder de ataque do Lacaio **x**, e o Lacaio **x** tem sua vida diminuída pelo poder de ataque do Lacaio **y**. Se a vida de um Lacaio chegar a zero, ele morre e é retirado da mesa.

- Se um Lacaio atacar um Herói, a vida do Herói é reduzida pelo poder de ataque do Lacaio mas o Lacaio não sofre nenhum tipo de dano.
- O jogador pode possuir até dez cartas na mão. Se já possuir dez cartas ao início de seu turno, a nova carta que foi comprada será descartada.
- Se em um dado turno não for possível a um jogador comprar cartas porque o Baralho já está esgotado, o jogador recebe um de dano. No próximo turno receberá dois de dano, depois três, e assim por diante.

3 Descrição do Trabalho

É esperado do aluno que implemente uma classe de **Jogador**, que irá analisar o jogo e tomar decisões para tentar vencer o jogo. Deste modo, uma das tarefas que se espera é que a classe **Jogador** implementada pelo aluno consiga ser competitiva, isto é, que vença o máximo de partidas que for possível ao jogar contra outras classes jogadoras.

A classe do jogador deve ser chamada **JogadorRA**xxxxxx (onde "xxxxxx" é o RA do aluno). A classe de jogador do aluno deve herdar da classe **Jogador** abstrata. Dois atributos são herdados da classe **Jogador** abstrata:

- ArrayList<Carta> mao: É um ArrayList de objetos da classe Carta. Possui as cartas que estão na mão do jogador. Deve ser inicializada no método construtor.
- boolean primeiroJogador: É um atributo booleano que diz se a classe foi escolhida como primeiro ou segundo jogador na partida. Se primeiroJogador é verdadeiro, a classe foi escolhida para ser o primeiro, senão foi escolhido para ser o segundo. Deve ser inicializada no método construtor.

Dois métodos deverão ser obrigatoriamente implementados para interagir com o Motor do jogo:

1. public JogadorRAxxxxxx (ArrayList<Carta> maoInicial, boolean primeiro)
2. public ArrayList<Jogada> processarTurno (Mesa mesa, Carta cartaComprada, ArrayList<Jogada> jogadasOponente)

O primeiro método é o método construtor, que deverá ser responsável por inicializar todos os atributos que você utilizar em sua classe. Além disso, este método deve preparar a sua classe para iniciar uma partida, recebendo como argumento uma lista de cartas da mão inicial em **maoInicial** e uma variável booleana **primeiro** que diz se esta classe foi escolhida para ser o primeiro jogador ou não. Um exemplo da implementação do método construtor é mostrado a seguir:

```
public JogadorRAxxxxxx (ArrayList<Carta> maoInicial, boolean primeiro){
    primeiroJogador = primeiro;
    mao = maoInicial;

    // Mensagens de depuracao:
    System.out.println("Sou o " + (primeiro?"primeiro":"segundo") + " jogador");
    System.out.println("Mao inicial:");
    for(int i = 0; i < mao.size(); i++)
        System.out.println("ID " + mao.get(i).getID() + ": " + mao.get(i));
}
```

O segundo método, nomeado **processarTurno()**, será chamado a cada turno do jogador para que este faça suas jogadas devolvendo um objeto **ArrayList<Jogada>**, ou seja, um ArrayList de objetos da classe Jogada. Como entrada este método recebe:

- Um objeto da classe **Mesa** que possui todas as informações do estado corrente da mesa (isto é, lacaios e suas vidas, vida dos heróis, número de cartas de cada jogador, mana disponível para este turno para cada jogador, etc).
- Um objeto da classe Carta que possui a carta que foi comprada neste turno.
- Um ArrayList de objetos da classe Jogada que contém as jogadas feitas pelo oponente no último turno dele.

Deste modo, com essas informações a classe **JogadorRA** deverá ser capaz de decidir suas jogadas deste turno. Uma maneira de começar a implementar o método **processarTurno()** é criar objetos para diferenciar entre quais são os seus Lacaios e quais são os do oponente, com base nas informações do objeto Mesa. Veja o exemplo a seguir:

```
public ArrayList<Jogada> processarTurno (Mesa mesa, Carta cartaComprada,
    ArrayList<Jogada> jogadasOponente){
    int minhaMana, oponenteMana;
    ArrayList<Carta> lacaiosMeus;
    ArrayList<Carta> lacaiosOponente;

    if(cartaComprada != null)
        mao.add(cartaComprada);

    if(primeiroJogador){
        minhaMana = mesa.getManaHeroi1();
        oponenteMana = mesa.getManaHeroi2();
        lacaiosMeus = mesa.getLacaiosHeroi1();
        lacaiosOponente = mesa.getLacaiosHeroi2();
    }
    else{
        minhaMana = mesa.getManaHeroi2();
        oponenteMana = mesa.getManaHeroi1();
        lacaiosMeus = mesa.getLacaiosHeroi2();
        lacaiosOponente = mesa.getLacaiosHeroi1();
    }

    ArrayList<Jogada> minhasJogadas = new ArrayList<Jogada>();
    // Aqui decide quais serao as jogadas
    return minhasJogadas;
}
```

4 Descrição do Motor

O Motor irá se comunicar com a classe **JogadorRA** através dos métodos construtor e **processarTurno()** supracitados. O método construtor é executado somente uma vez, fornecendo a mão inicial e a informação de qual jogador a classe é (primeiro ou segundo). Em seguida serão feitas várias chamadas ao método **processarTurno()** para cada turno daquela partida.

A seguir serão apresentadas as principais classes que serão utilizadas no trabalho.

4.1 Classe Carta

A classe Carta é descrita na Tabela 1. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set. Por convenção, se o nome de um atributo é composto como vidaMesa, então o método get correspondente se chamará getVidaMesa (note que a letra v virou maiúscula).

Tabela 1: Classe Carta

Atributo	Tipo	Descrição	Domínio
ID	int	ID único de uma carta	Inteiro positivo
nome	String	Nome da carta	String
tipo	TipoCarta	Define se a carta é do tipo lacaio ou magia	Enum: TipoCarta.LACAIO, TipoCarta.MAGIA
ataque	int	Ataque da carta (lacaio)	Inteiro positivo
vida	int	Vida da carta (lacaio)	Inteiro positivo
vidaMesa	int	Vida na mesa da carta (lacaio)	Inteiro positivo
mana	int	Mana da carta	Inteiro positivo
magiaTipo	TipoMagia	Define se a magia é do tipo alvo ou área	Enum: TipoMagia.ALVO, TipoMagia.AREA
magiaDano	int	Valor de dano (magia)	Inteiro positivo
turno	int	(Utilizado pelo Motor)	Inteiro positivo

4.2 Classe Mesa

A classe Mesa é descrita na Tabela 2. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set. A convenção para nomes de get e set é a mesma da aplicada para a classe Carta.

A Mesa é um objeto que será fornecido no início de todo turno para a classe **JogadorRA** através do método **processarTurno()**. A Mesa irá fornecer o **estado do jogo** imediatamente antes do início do turno do jogador correspondente. É necessário consultar a Mesa para descobrir, por exemplo, se um dado lacaio ainda está vivo no jogo ou não.

Exemplo da consulta de Lacaios vivos do oponente através da classe Mesa:

```
1 ArrayList<Carta> lacaiosOponente = primeiroJogador ? mesa.getLacaiosJog2() :
      mesa.getLacaiosJog1();
2 for(int i = 0; i < lacaiosOponente.size(); i++){
3     Carta lacaioOpo = lacaiosOponente.get(i);
4     System.out.println("Lacaio do oponente descoberto: "+ lacaioOpo);
5 }
```

Tabela 2: Classe Mesa

Atributo	Tipo	Descrição	Domínio
lacaiosJog1	ArrayList<Carta>	Lacaios na mesa do herói 1	Cartas que são lacaios vivos na mesa
lacaiosJog2	ArrayList<Carta>	Lacaios na mesa do herói 2	Cartas que são lacaios vivos na mesa
vidaHeroi1	int	Vida do herói 1	Inteiro entre [0,30]
vidaHeroi2	int	Vida do herói 2	Inteiro entre [0,30]
numCartasJog1	int	Número de Cartas jogador 1	Inteiro entre [0,10]
numCartasJog2	int	Número de Cartas jogador 2	Inteiro entre [0,10]
manaJog1	int	Mana disponível neste turno para jogador 1	Inteiro entre [1,10]
manaJog2	int	Mana disponível neste turno para jogador 2	Inteiro entre [1,10]

4.3 Classe Jogada

A classe Jogada é descrita no texto abaixo. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set. A convenção para nomes de get e set é a mesma da aplicada para a classe Carta. Os atributos da classe Jogada são os seguintes:

- *tipo*: Um atributo do tipo **TipoJogada** (enumeração) que define se a jogada trata-se de baixar um lacaio à mesa (TipoJogada.LACAIO), utilizar uma magia (TipoJogada.MAGIA), atacar com um lacaio (TipoJogada.ATAQUE) ou utilizar um poder heróico (TipoJogada.PODER).
- *cartaJogada*: Um atributo da classe **Carta** que define qual carta está sendo utilizada. No caso de baixar um lacaio (TipoJogada.LACAIO) é o lacaio que será baixado, no caso de utilizar uma magia é a carta da magia (TipoJogada.MAGIA), no caso de atacar com um lacaio é o lacaio que irá atacar (TipoJogada.ATAQUE). Toma valor **null** no caso de Poder Heróico.
- *cartaAlvo*: Um atributo da classe **Carta** que define qual carta será utilizada como alvo. Ou então toma o valor **null** para ter como alvo o herói do oponente. Se for realizar um ataque (TipoJogada.ATAQUE) ou utilizar uma magia de alvo (TipoJogada.MAGIA) o campo deve conter a carta do lacaio do oponente que será alvo desse dano, ou então **null** para ter o herói como alvo.

Note que alguns campos não são utilizados dependendo da jogada, a Tabela 3 auxilia ao verificar quais campos são utilizados para cada tipo de jogada (X) e quais não são (vazio). Por exemplo, se a jogada é de baixar um lacaio, o tipo será TipoJogada.LACAIO, e o atributo cartaJogada será o lacaio que será baixado à mesa. Porém o atributo cartaAlvo será ignorado e como convenção deve-se utilizar **null** nesta situação.

O construtor da classe Jogada é: public **Jogada**(TipoJogada tipo, Carta cartaJogada, Carta cartaAlvo).

Tabela 3: Atributos relevantes por tipo de Jogada

Tipo	cartaJogada	cartaAlvo
TipoJogada.LACAIO	X	
TipoJogada.MAGIA de alvo	X	X
TipoJogada.MAGIA de área	X	
TipoJogada.ATAQUE	X	X
TipoJogada.PODER		X

5 Exemplos de Código

Aqui mostraremos alguns exemplos de como criar uma **Jogada** de Baixar Lacaio, Usar Magia, Poder Heróico e Ataque de Lacaio.

Exemplo de Jogada para baixar um lacaio à mesa:

```
// card = objeto Carta do lacaio que quero baixar
// null = nao ha alvo nesta jogada
Jogada lac = new Jogada(TipoJogada.LACAIO, card, null);
minhasJogadas.add(lac);
```

Exemplo de Jogada para utilizar magia de alvo:

```
// card = objeto Carta da magia (de alvo) que quero usar
// cardAlvo = objeto Carta de um lacaio do oponente que e o alvo desta magia
Jogada mag = new Jogada(TipoJogada.MAGIA, card, cardAlvo);
minhasJogadas.add(mag);
```

Exemplo de Jogada para utilizar magia de área:

```
// card = objeto Carta da magia (de area) que quero usar
// null = nao ha alvo especifico nesta jogada
Jogada mag = new Jogada(TipoJogada.MAGIA, card, null);
minhasJogadas.add(mag);
```

Exemplo de Jogada para utilizar um poder heróico:

```
// null = nao ha carta sendo usada
// cardAlvo = objeto Carta do lacaio que quero atacar com o poder
Jogada pod = new Jogada(TipoJogada.PODER, null, cardAlvo);
minhasJogadas.add(pod);
```

Exemplo de Jogada para atacar com um lacaio em outro lacaio:

```
// card = objeto Carta de meu lacaio de ataque
// cardAlvo = objeto Carta que serÃ¡ alvo do ataque
Jogada atk = new Jogada(TipoJogada.ATAQUE, card, cardAlvo);
minhasJogadas.add(atk);
```

Exemplo de Jogada para atacar com um lacaio no herói do oponente:

```
// card = objeto Carta de meu lacaio de ataque
// null significa que o lacaio esta atacando o heroi do oponente.
Jogada atk = new Jogada(TipoJogada.ATAQUE, card, null);
minhasJogadas.add(atk);
```

Exemplo de como percorrer as cartas da mão em ordem, e baixar à mesa todos os lacaios que a mana permitir:

```java
int minhaMana = primeiroJogador ? mesa.getManaJog1() : mesa.getManaJog2();
for(int i = 0; i < mao.size(); i++){
    Carta card = mao.get(i);
    if(card.getTipo() == TipoCarta.LACAIO && card.getMana() <= minhaMana){
        Jogada lac = new Jogada(TipoJogada.LACAIO, card, null);
        minhasJogadas.add(lac);
        minhaMana -= card.getMana();
        System.out.println("Jogada: Baixei o lacaio: "+ card);
        mao.remove(i);
        i--;
    }
}
```

6 Saída do programa

Atenção: a classe **JogadorRA** não deve imprimir nada na saída do programa.

7 Critério de Avaliação

O trabalho será avaliado através de dois critérios, cada um correspondente a 50% da nota: (i) critério de qualidade da implementação, (ii) critério de competitividade do jogador.

$$NT_1 = 0.5 \times NP_1 + 0.5 \times NP_2 \tag{1}$$

Onde:

- NT_1: Nota do Trabalho 1
- NP_1: Nota parcial do Trabalho 1 correspondente ao critério de qualidade da implementação.
- NP_2: Nota parcial do Trabalho 1 correspondente ao critério de competitividade.

7.1 Critério de qualidade da implementação

1. **Estratégia do jogador:** No cabeçalho do código-fonte deve haver um texto comentado, de pelo menos 30 linhas, descrevendo a estratégia adotada pelo seu jogador, ou seja, quais critérios são utilizados para baixar lacaios, utilizar magias, utilizar poder heróico, escolher os alvos e quando utilizar magias de área. Você leva em consideração quantos pontos de vida seu adversário ou você tem para decidir uma jogada? Você utiliza algum tipo de memória, por exemplo para avaliar as cartas já baixadas pelo seu adversário? Ou por exemplo para avaliar o comportamento do seu adversário durante o jogo?
2. **Corretude:** O código não deve possuir *warnings* ou erros de compilação. O código não deve emitir *exceptions* em nenhuma situação. As jogadas realizadas pelo jogador devem ser válidas, segundo as regras do jogo, em todas as situações.
3. **Aderência ao Enunciado:** A implementação deve realizar o que é requisitado no enunciado.
4. **Comentários:** Os comentários devem ser suficientes para explicar os trechos mais importantes da implementação, utilizando-se de terminologias corretas vistas em sala de aula quando isso se aplicar.

5. **Convenções:** Os nomes das entidades de seu código devem seguir as convenções Java. Além disso, seu código deve estar corretamente identado e bem apresentado segundo as boas práticas da linguagem.

7.2 Critério de competitividade do jogador

Quanto ao critério de competitividade, o jogador mais competitivo (maior número de vitórias) receberá a nota máxima (10) neste critério e o jogador menos competitivo (menor número de vitórias) receberá a nota mínima (0). Será introduzido um jogador ingênuo no campeonato que fornecerá um limitante inferior de competitividade. A nota do critério competitividade será dada de acordo com a seguinte equação:

$$NP_2 = 10 \times \left(\frac{V_{jogador} - V_{min}}{V_{max} - V_{min} + \delta} \right) \tag{2}$$

- $V_{jogador}$: Número de vitórias obtidas pelo jogador no campeonato.
- V_{min}: Número mínimo de vitórias obtidas pelo pior jogador do campeonato.
- V_{max}: Número máximo de vitórias obtidas pelo melhor jogador do campeonato.
- $\delta = 10^{-6}$: infinitesimal para evitar divisão por zero.

8 Observações

- O trabalho é individual.
- Não é permitido nenhum tipo de compartilhamento de código entre os alunos ou o uso de códigos de terceiros. Uma única exceção permitida consiste nas APIs da linguagem Java. Em caso de plágio, todos os alunos envolvidos serão reprovados com média zero.
- Não é permitido explorar nenhuma brecha de segurança (*exploit*) do Motor. Se você encontrar alguma brecha, relate imediatamente ao professor.

9 Submissão

O trabalho deverá ser submetido até o dia 30 de setembro através do Ensino Aberto. A submissão deve ser exclusivamente um arquivo JogadorRAxxxxxx.java (onde “’xxxxxx’ é o RA do aluno). Crie uma pasta denominada “Trabalho 1” em seu portfólio do Ensino Aberto e inclua seu código-fonte nessa pasta. **Importante**: Não se esqueça de deixar a visibilidade da pasta do Portfólio como "Compartilhado somente com Formadores".