MC302
Primeiro semestre de 2017

Trabalho 1

Professores: Esther Colombini (esther@ic.unicamp.br) e Fábio Luiz Usberti (fusberti@ic.unicamp.br)
PEDs: (Turmas ABCD) Elisangela Santos (ra149781@students.ic.unicamp.br), Lucas Faloni (lucasfaloni@gmail.com), Lucas David (lucasolivdavid@gmail.com), Wellington Moura (wellington.tylon@hotmail.com)
PEDs (Turmas EF) Natanael Ramos (naelr8@gmail.com), Rafael Arakaki (rafaelkendyarakaki@gmail.com)
PAD: (Turmas ABCD) Igor Torrente (igortorrente@hotmail.com)
PAD: (Turmas EF) Bleno Claus (blenoclaus@gmail.com)

---

1 Objetivo

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

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. Existem três tipos de magias: (i) dano em alvo, (ii) dano em área, (iii) magia de buff. As magias de dano em alvo causam dano em um único lacaio do oponente à escolha do jogador utilizador ou no herói do oponente. As magias de área causam dano em todos os lacaios e também no herói do oponente. Por fim, as magias de buff podem aumentar o ataque e a vida de um dos lacaios do jogador que a utilizar.
- 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 sete cartas na mão. Se já possuir sete cartas ao início de seu turno, a nova carta que foi comprada será descartada.
- Quando o jogador fica sem cartas para comprar do baralho chamamos esse estado de jogo de *fadiga*. A *fadiga* inicia quando em um dado turno não for possível a um jogador comprar cartas porque o baralho já está esgotado, então o jogador recebe um de dano no início daquele turno. No próximo início de turno receberá dois de dano, depois três, e assim por diante.
- **Cada jogador só pode ter até sete lacaios vivos em seus respectivos campos.**

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:

```
1 public JogadorRAxxxxxx (ArrayList<Carta> maoInicial, boolean primeiro){
2     primeiroJogador = primeiro;
3     mao = maoInicial;
4
5     // Mensagens de depuracao:
```

```
6       System.out.println("Sou o " + (primeiro?"primeiro":"segundo") + " jogador");
7       System.out.println("Mao inicial:");
8       for(int i = 0; i < mao.size(); i++)
9           System.out.println("ID " + mao.get(i).getID() + ": " + mao.get(i));
10  }
```

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:

```
1   public ArrayList<Jogada> processarTurno (Mesa mesa, Carta cartaComprada,
        ArrayList<Jogada> jogadasOponente){
2       int minhaMana, oponenteMana;
3       ArrayList<Carta> lacaiosMeus;
4       ArrayList<Carta> lacaiosOponente;
5
6       if(cartaComprada != null)
7           mao.add(cartaComprada);
8
9       if(primeiroJogador){
10          minhaMana = mesa.getManaHeroi1();
11          oponenteMana = mesa.getManaHeroi2();
12          lacaiosMeus = mesa.getLacaiosHeroi1();
13          lacaiosOponente = mesa.getLacaiosHeroi2();
14      }
15      else{
16          minhaMana = mesa.getManaHeroi2();
17          oponenteMana = mesa.getManaHeroi1();
18          lacaiosMeus = mesa.getLacaiosHeroi2();
19          lacaiosOponente = mesa.getLacaiosHeroi1();
20      }
21
22      ArrayList<Jogada> minhasJogadas = new ArrayList<Jogada>();
23      // Aqui decide quais serao as jogadas
24      return minhasJogadas;
25  }
```

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 vidaAtual, então o método get correspondente se chamará getVidaAtual (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
custoMana	int	Custo de mana da carta	Inteiro positivo

Note que a classe Carta (Tabela 1) é uma classe abstrata. Existem duas classes que herdam da classe Carta: CartaLacaio e CartaMagia que serão descritas a seguir.

4.2 Classe CartaLacaio

A classe CartaLacaio é descrita na Tabela 2. Uma vez que a classe CartaLacaio herda da classe Carta, a classe CartaLacaio possui também os atributos e métodos da classe abstrata Carta. Na Tabela 2 são mostrados apenas os atributos específicos da classe CartaLacaio. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set.

Tabela 2: Classe CartaLacaio

Atributo	Tipo	Descrição	Domínio
ataque	int	Ataque da carta	Inteiro positivo
vidaAtual	int	Vida da carta durante o jogo	Inteiro positivo
vidaMaxima	int	Vida da carta sem contar danos	Inteiro positivo
efeito	TipoEfeito	(Não será cobrado no Trabalho 1)	Enumeração
turno	int	(Utilizado pelo Motor)	Inteiro positivo

4.3 Classe CartaMagia

A classe CartaMagia é descrita na Tabela 3. Uma vez que a classe CartaMagia herda da classe Carta, a classe CartaMagia possui também os atributos e métodos da classe abstrata Carta. Na Tabela 3 são mostrados apenas os atributos específicos da classe CartaMagia. Todos os atributos são privados e seus valores podem ser recuperados por meio dos métodos get e set.

4.4 Classe Mesa

A classe Mesa é descrita na Tabela 4. 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.

Tabela 3: Classe CartaMagia

Atributo	Tipo	Descrição	Domínio
magiaTipo	TipoMagia	Define o tipo da magia	Enumeração: ALVO, AREA ou BUFF
magiaDano	int	Valor de dano ou buff	Inteiro positivo

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:

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

Tabela 4: 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]

4.5 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

Tabela 5: Atributos relevantes por tipo de Jogada

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

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 5 mostra 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).

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 magia de buff:

```
// card = objeto Carta da magia (de buff) que quero usar
// cardAlvo = objeto Carta de um lacaio do utilizador que recebera o buff
```

```
Jogada mag = new Jogada(TipoJogada.MAGIA, card, cardAlvo);
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:

```
int minhaMana = primeiroJogador ? mesa.getManaJog1() : mesa.getManaJog2();
for(int i = 0; i < mao.size(); i++){
    Carta card = mao.get(i);
    if(card instanceof CartaLacaio && 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 Comportamentos

Dentro do jogo LaMa os jogadores costumam adotar certas estratégias típicas, chamadas aqui de comportamentos, para tentar vencer o jogo. Dentre outras, três comportamentos se destacam: (i) agressivo, (ii) controle, (iii) curva de mana. *A sua classe **JogadorRA** deverá possuir os três comportamentos implementados.* Além disso, a classe deverá adotar algum critério para determinar em qual comportamento irá jogar a cada momento (Seção 6.4).

6.1 Agressivo

O comportamento agressivo objetiva causar dano no herói do oponente o mais rápido possível. Esse comportamento pode ser entendido como a abordagem mais direta e de curto prazo ao objetivo do jogo. As prioridades desse comportamento são baixar lacaios de ataque alto, usar magias de alvo no herói do oponente e quase nunca realizar trocas com os lacaios do oponente (obs: “trocas” é uma expressão para designar quando um lacaio ataca um outro lacaio do adversário).

6.2 Controle

O comportamento de controle objetiva manter um controle do campo acumulando mais lacaios vivos do que o oponente e assim podendo realizar "trocas favoráveis". Entende-se por "troca favorável" quando um lacaio x ataca um lacaio y do oponente onde: (i) o lacaio x sobreviverá e o lacaio y não; (ii) ambos os lacaios morrerão, porém o custo de mana do lacaio y é maior do que o custo de mana do lacaio x; (iii) ambos os lacaios morrerão, porém o lacaio x já estava danificado e possui menos vida do que o lacaio y. Quando não há trocas favoráveis, a estratégia controle usa seus lacaios para atacar o herói do oponente.

O uso das magias é muito importante no comportamento de controle. Procura-se utilizar as magias para matar lacaios do oponente de maneira eficiente. Por eficiente considera-se que: (i) magias de área são usadas somente se houver dois ou mais lacaios do oponente em campo; (ii) magias de dano em alvo são usadas apenas para eliminar lacaios e onde a diferença entre o dano da magia e a vida atual do lacaio é pequena (no máximo 1, por exemplo), em outras palavras, não "desperdiça-se" dano de magias de alvo. O comportamento controle prioriza manter sob controle os lacaios do oponente e por isso prefere eliminar lacaios do oponente com magias ao invés de descer lacaios mais poderosos.

6.3 Curva de mana

O comportamento de curva de mana é um comportamento meio termo entre o agressivo e o controle. Esse comportamento procura usar as cartas da mão de maneira a utilizar toda a mana disponível em cada turno. Lacaios que possuem o custo exato do total de mana daquele turno são tidos como prioridade. Por exemplo, no turno 5 uma boa jogada de curva de mana é: (i) baixar um lacaio de custo 5; (ii) baixar um lacaio de custo 3 de mana e utilizar uma magia de custo 2 para matar um lacaio do adversário. As trocas são realizadas de maneira parecidas ao do controle.

As magias somente são utilizadas se forem eliminar lacaio(s) que somem mais do que o custo de mana da própria magia. Essa estratégia procura ganhar vantagem fazendo um uso mais eficiente da mana do que o oponente. O comportamento de curva de mana prefere descer lacaios de custo de mana alto (mais poderosos) ao invés de eliminar lacaios do oponente com magias. Um problema dessa estratégia é que ao chegar no décimo turno ela não se torna tão viável uma vez que haverá muita mana para ser utilizada em todos os turnos seguintes.

6.4 Critério de comportamento

A classe **JogadorRA** deverá implementar algum critério para escolher entre qual comportamento irá jogar a cada momento do jogo. Esse critério poderá levar em conta fatores como: (i) estágio do jogo (início, meio, fim), (ii) vida do seu herói e do oponente, (iii) quais cartas existem na mão, (iv) quais cartas já foram utilizadas pelo adversário, entre muitos outros possíveis. O critério adotado deve ser claramente e sucintamente apresentado no cabeçalho do código-fonte (Seção 8.1).

7 Saída do programa

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

8 Critério de Avaliação

A nota do trabalho será composta por dois componentes, conforme mostra a equação a seguir:

$$NT_1 = \min\{10, Q + C\}$$

Onde:

- NT_1: Nota do Trabalho 1
- $Q \in [0, 10]$: Componente de qualidade do trabalho de acordo com os critérios: corretude, aderência, estratégia, comentários e convenções.
- $C \in [0, 2]$: Componente de competitivade do jogador, ou seja, o quão eficiente é esse jogador ao disputar com os demais jogadores. Cabe observar que este componente entrará como **bonificação** à nota do Trabalho 1.

8.1 Componente de qualidade da implementação

1. **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.
2. **Aderência ao Enunciado:** A implementação deve realizar o que é requisitado no enunciado.
3. **Estratégia do jogador:** No cabeçalho do código-fonte deve haver um texto comentado, de pelo menos 30 e no máximo 60 linhas, descrevendo organizadamente a estratégia adotada pelo seu jogador *para cada comportamento*, ou seja, quais critérios são utilizados para baixar lacaios, utilizar magias, utilizar poder heróico, escolher os alvos e quando utilizar magias de área. Além disso, descreva em poucas palavras qual o critério que seu jogador utiliza para *trocar entre os comportamentos*. Você leva em consideração quantos pontos de vida seu adversário ou você tem para decidir uma jogada? Você considera quais cartas possui na mão e quais ainda estão no baralho ? Você utiliza algum tipo de memória, por exemplo para avaliar as cartas já baixadas pelo seu adversário? Você utiliza memória para avaliar o comportamento do seu adversário durante o jogo? (Obs: a organização e clareza do seu texto também implicará na nota deste componente).
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. Dê preferência para o uso de comentários no padrão Javadoc.
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.

8.2 Componente de competitividade do jogador

O componente de competitividade refere-se ao número de vitórias do jogador relativo ao número de vitórias dos demais jogadores. O jogador mais competitivo (maior número de vitórias) receberá a bonificação máxima (2). Será introduzido um jogador "ingênuo" no campeonato que fornecerá um limitante inferior de competitividade. A bonificação mínima (0) corresponderá ao número de vitórias do jogador "ingênuo". O componente de competitividade será calculado de acordo com a seguinte equação:

$$C = 2 \times \left(\frac{V_{jogador} - V_{min}}{V_{max} - V_{min}} \right) \tag{1}$$

- $V_{jogador}$: Número de vitórias obtidas pelo jogador no campeonato.
- V_{min}: Número de vitórias obtidas pelo jogador "ingênuo" do campeonato.
- V_{max}: Número máximo de vitórias obtidas pelo melhor jogador do campeonato.

9 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.

10 Submissão

O trabalho deverá ser submetido até as 23:55 do dia 28 de abril de 2017, através do Moodle. A submissão deve ser exclusivamente um arquivo nomeado JogadorRAxxxxxx.java (onde "'xxxxxx' é o RA do aluno). Após submeter, certifique-se de que o arquivo enviado é o correto.