| | |
|---|---|
| | Centro Tecnológico<br>Departamento de Informática |
| Disciplina: Programação II | Código: INF09330 |
| Trabalho Prático | Prof. Dr. Thiago Oliveira dos Santos |

# Especificação do Trabalho
# Batalha Naval

## Introdução

Este trabalho tem como objetivo avaliar, de forma prática, os conceitos ensinados no curso de Programação II. A meta deste trabalho é implementar uma batalha naval seguindo as especificações descritas neste documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.

O jogo de Batalha Naval (também conhecido como *Battleship*) é um jogo de estratégia e adivinhação para dois jogadores, onde cada jogador possui um tabuleiro (um exemplo pode ser visto na Figura 01) com seus navios posicionados, ocultos do outro jogador[1]. Os jogadores se alternam dando "tiros" no campo inimigo, e o vencedor é quem acaba com a frota adversária primeiro.

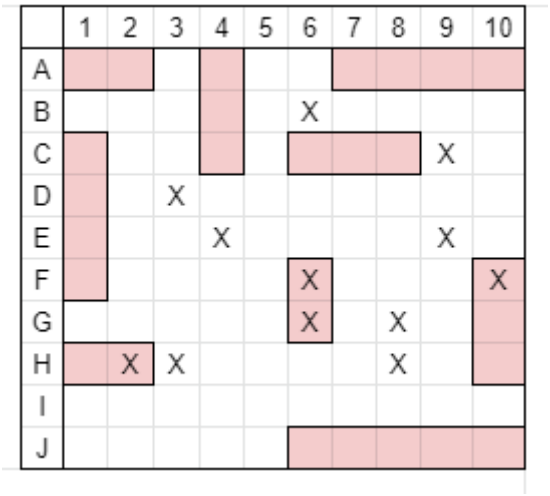


Figura 01 - Exemplo de um tabuleiro 10x10 de um jogo de batalha naval.

A batalha naval se popularizou nos anos 30, mas foi só em 1967 que ganhou uma versão de tabuleiro convencional. Na sua forma mais básica, baseada na versão de 1990 de Milton Bradley, existem 5 tipos de navios[2] descritos na Tabela 01.

| Nome do Navio | Tamanho |
|---|---|
| Carrier | 5 |

[1] https://en.wikipedia.org/wiki/Battleship_(game)
[2] https://www.hasbro.com/common/instruct/battleship.pdf

| | |
|---|---|
| Battleship | 4 |
| Cruiser | 3 |
| Submarine | 3 |
| Destroyer | 2 |

Tabela 01 - Informações dos navios

## Descrição do Trabalho

Neste trabalho, você deverá implementar um jogo de batalha naval para dois jogadores. O programa deverá rodar no terminal, assim como todos os outros exercícios feitos em sala. Para simplificar o programa a ser criado, não será obrigatório implementar a disposição aleatória dos navios no tabuleiro, sendo esses fornecidos como dados de entrada.

De maneira geral, o programa irá iniciar pela linha de comando (terminal) e irá ler o tabuleiro inicial de cada um dos jogadores. Com os tabuleiros iniciados, os jogadores farão suas jogadas alternadamente, seguindo a ordem jogador 1 e jogador 2, até que um dos dois ganhe ou o jogo empate. O programa apresentará os resultados parciais na saída padrão até o jogo terminar, hora em que ele apresentará o resultado final e salvará as estatísticas do jogo em um arquivo.

## Funcionamento Detalhado do Programa

A execução do programa será feita através da linha de comando (i.e. pelo cmd, console, ou terminal) e permitirá a passagem de parâmetros. As funcionalidades a serem realizadas pelo programa são descritas a seguir: *inicializar jogo*, *verificar tabuleiro*, *realizar jogo*, *gerar resumo de resultado, gerar estatísticas e gerar tabuleiro válido*. A descrição dos parâmetros de inicialização, da exibição do estado atual do programa, assim como, das operações a serem realizadas pelo programa, são apresentadas em detalhes a seguir.

*Inicializar jogo:* Para garantir o correto funcionamento do programa, o usuário deverá informar, ao executar o programa pela linha de comando, o caminho do diretório contendo os arquivos a serem processados (exemplo assumindo um programa compilado para o executável *prog*, “./prog /maquinadoprofessor/diretoriodeteste”). Considere um caminho com tamanho máximo de 1000 caracteres. O programa deverá verificar a presença desse parâmetro informado na linha de comando. Caso o usuário tenha esquecido de informar o nome do diretório, o programa deverá exibir (ex. “ERRO: O diretorio de arquivos de configuracao nao foi informado”), e finalizar sua execução. Dentro desse diretório, espera-se encontrar os arquivos dos tabuleiros dos jogadores, *tabu_1.txt* e *tabu_2.txt*, sendo alguns exemplos fornecidos com esta especificação. O programa deverá ler informações dos arquivos dos tabuleiros e preparar o ambiente de jogo. Caso o programa não consiga ler os arquivos, ele deverá ser finalizado e imprimir uma mensagem informando que não conseguiu ler os arquivos (OBS: a mensagem deve conter o caminho e nome do arquivo que ele tentou ler).

Os arquivos de tabuleiro definirão as configurações de do tabuleiro de cada jogador, incluindo o posicionamento dos seus navios no campo de batalha. Cada linha conterá: o tipo do navio (literal igual ao apresentado na tabela anterior), a identidade do navio representada por um número único (inteiro), a orientação do navio no tabuleiro (0 para vertical e 1 para horizontal), a coordenada do primeiro ponto do navio, expressa com uma letra e um número, todos separados por ";". Considere a origem do tabuleiro no canto superior esquerdo da matriz com um crescimento positivo para a direita e para baixo dos eixos horizontais e verticais respectivamente. Os tabuleiros deverão ser **válidos** para o correto funcionamento do jogo (definição de tabuleiro válido é dada adiante, no tópico verificar tabuleiro), ou seja, deve-se verificar a validade do tabuleiro antes de prosseguir. Veja um exemplo de arquivo de tabuleiro abaixo.

*tabu_1.txt*

```
Submarine;1;1;a2
```

Caso a leitura dos tabuleiros seja bem-sucedida (i.e., eles sejam válidos), em seguida, o programa deverá perguntar o nome dos jogadores na entrada/saída padrão, com a mensagem "Nome do Jogador #:", em que # representa o número do jogador, 1 ou 2. Assumir que os nomes serão simples (ou seja, apenas uma palavra e conter no máximo 16 caracteres). Um exemplo de entrada e saída esperada é dado abaixo, sendo as linhas 1 e 3 exemplos das saídas e as linhas 2 e 4 exemplo das entradas:

| L | Terminal |
|---|---|
| 1 | Nome do Jogador 1: |
| 2 | Leticia |
| 3 | Nome do Jogador 2: |
| 4 | Thiago |

A inicialização do jogo será salva, na pasta de saída do caso de teste em questão (a pasta de saída do caso de teste é uma pasta denominada *saida*, já criada e localizada dentro do diretório contendo os arquivos de tabuleiro do jogo daquele teste), em um arquivo de nome inicializacao.txt. Esse arquivo conterá o nome do jogador # na primeira linha, seguido do tabuleiro do jogador # nas linhas seguintes. O tabuleiro será descrito pela letra "o" (minúscula) para marcar espaços de "água" e a letra "X" (maiúscula) para marcar a posição dos navios. Esses elementos estarão sempre separados por um espaço. Cada linha da matriz será dada em uma linha diferente do arquivo. Após as informações do jogador 1, será disposto da mesma forma para o jogador 2, separando ambos por uma linha vazia. Um exemplo do arquivo de inicialização pode ser visto abaixo:

inicializacao.txt

```
Leticia
o X X X o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o

Thiago
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o o o o o o o o o
o o X o o o o o o o
o o X o o o o o o o
o o X o o o o o o o
o o o o o o o o o o
```

*Verificar tabuleiro*: Após ler os tabuleiros, o programa deverá verificar se cada um dos tabuleiros é válido e salvar, na pasta de saída do caso de teste em questão, um arquivo de nome validacao_tabuleiros.txt. Esse arquivo conterá o nome do tabuleiro, um ponto e vírgula, e uma descrição literal da validação (“valido” se o tabuleiro for válido e “invalido” se ele for inválido). Isto é, um navio não deve encostar em outro, não deve invadir além das bordas do tabuleiro, e um tabuleiro sempre tem um tamanho 10x10 e deve conter ao menos 1 navio. Se um tabuleiro não cumprir esses requisitos, ele será considerado inválido. Adicionalmente, o programa deverá verificar se os tabuleiros são compatíveis entre si, ou seja, ambos os jogadores têm o mesmo número de navios de cada tipo. Essa verificação deverá ser feita caso os dois tabuleiros sejam válidos. Quando ambos tabuleiros forem válidos, o arquivo deverá conter uma terceira linha com a mensagem “Tabuleiros compativeis entre si” ou “Tabuleiros incompativeis entre si” dependendo de cada caso. Abaixo, veja um exemplo de um arquivo de validação de tabuleiro em que os dois tabuleiros são válidos e compatíveis:

validacao_tabuleiros.txt

```
tabu_1.txt;valido
tabu_2.txt;valido
Tabuleiros compativeis entre si
```

*Realizar jogo*: Uma vez preparado o jogo, as jogadas poderão começar (sempre com o jogador 1) e seguirão se alternando com o jogador 2) até o fim do jogo. O jogo terminará quando um dos jogadores destruir todos os navios do outro. As jogadas deverão ser lidas da entrada padrão de dados, permitindo dessa forma, um redirecionamento a partir de arquivo. O fluxo da jogada segue:

- Antes de cada jogada, o programa deverá pedir a jogada do jogador da vez exibindo a mensagem “Jogada de @@@:”, em que @@@ é o nome do jogador da vez.
- A jogada será fornecida pelo usuário no formato “@#”, em que @ é uma letra (iniciando de “a” minúsculo e indo até a letra correspondente ao tamanho do tabuleiro, por exemplo, “j” para o tamanho 10) e # é um número (iniciando de 1 e indo até o tamanho do tabuleiro).
- Caso a jogada seja repetida, isto é, o mesmo jogador já houver feito aquela mesma jogada anteriormente, ou caso a jogada seja inválida, isto é, se a linha digitada com a jogada não seguir o padrão “@#\n” com o @ e # sendo valores contidos no intervalo permitido para o tabuleiro 10x10 do jogo (exemplos de jogadas inválidas: “aa2”, “b0”, “w3”, “a30”, “b2w”, “C3”, entre outras) o programa deverá imprimir “$$$:Jogada inválida! Jogue novamente @@@:”, em que $$$ é a jogada inválida e @@@ é o nome do jogador que deu a entrada inválida.
- Após uma jogada válida, o programa deverá exibir em uma linha a jogada (“@#:”) seguido do resultado da jogada ("Tiro!", "Agua", ou "Afundou @@", em que @@ é o tipo do navio destruído) e a condição atual do tabuleiro do adversário na saída padrão, sendo essas informações separadas por uma linha vazia. A condição atual do tabuleiro será dada pela frase indicando qual o tabuleiro “Tabuleiro atual de @@@ apos a jogada de ***”, em que “***” é o nome do jogador que acabou de jogar e “@@@” é o nome do jogador adversário, e o tabuleiro explícito abaixo, com pontos finais (“.”) representando os lugares onde não houveram jogadas (ou seja, posições fechadas), a letra “X” (maiúscula) representando onde os tiros acertaram um navio e a letra “o” (minúscula) representando onde os tiros erraram os navios (ou seja, que acertaram água). Um exemplo da saída de um jogo pode ser visto abaixo:

saida.txt

```
Nome do Jogador 1:
Nome do Jogador 2:

Jogada de Leticia:

b3:Agua

Tabuleiro atual de Thiago apos a jogada de Leticia
. . . . . .
. . o . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Jogada de Thiago:

a2:Tiro!

Tabuleiro atual de Leticia apos a jogada de Thiago
. X . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

```
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Jogada de Leticia:

b1:Agua

Tabuleiro atual de Thiago apos a jogada de Leticia
. . . . . .
o . o . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Jogada de Thiago:

a3:Tiro!

Tabuleiro atual de Leticia apos a jogada de Thiago
. X X . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Jogada de Leticia:

b2:Agua

Tabuleiro atual de Thiago apos a jogada de Leticia
. . . . . .
o o o . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Jogada de Thiago:

a4:Afundou Submarine

Tabuleiro atual de Leticia apos a jogada de Thiago
. X X X . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
```

```
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Vencedor: Thiago
```

O resultado final do jogo será descrito por uma das seguintes mensagens: "Empate" ou "Vencedor: @@@", em que @@@ é o nome do jogador vencedor. Para tornar o jogo mais justo, sempre que o jogador 1 ganhar, o programa deverá permitir que o jogador 2 dê mais um tiro para tentar empatar o jogo. Caso ele também destrua todos os navios do jogador 1, então o jogo será considerado empate. Os resultados deste passo poderão ser redirecionados para um arquivo, a fim de possibilitar a comparação com os arquivos fornecidos como exemplo.

*Gerar resumo de resultado*: Ao terminar o jogo, o programa deverá salvar, na pasta de saída do caso de teste em questão, um arquivo (denominado resultado.txt) contendo o resumo do resultado do jogo. O resumo será uma lista de jogadas de cada jogador ordenadas de forma crescente seguida do vencedor do jogo. A primeira linha será referente ao nome literal do primeiro jogador, seguido nas linhas posteriores pela coordenada da jogada (expressa com uma letra e um número) e o resultado da jogada. As jogadas serão apresentadas na ordem que foram sendo feitas. Se a jogada acertou, será expresso o nome do navio que acertou, no formato “@# - Tiro - @@@ - ID $$”, em que @# se refere a coordenada da jogada, @@@ se refere ao nome do navio atingido e $$ se refere ao ID do navio. Se a jogada não acertou um navio, ela será especificada no formato “@# - Agua”, onde @# também se refere a coordenada da jogada. Após as informações do jogador 1 e uma linha vazia, serão informadas as jogadas da mesma forma para o jogador 2. Após as informações dos dois jogadores e saltar uma linha, será informado o vencedor da partida, no formato “Vencedor: @@@”, em que @@@ é o nome literal do vencedor, ou a palavra “Empate” no caso de empate. Um exemplo do arquivo do resultado pode ser visto abaixo:

resultado.txt

```
Leticia
b3 - Agua
b1 - Agua
b2 - Agua

Thiago
a2 - Tiro - Submarine - ID 01
a3 - Tiro - Submarine - ID 01
a4 - Tiro - Submarine - ID 01

Vencedor: Thiago
```

Para facilitar a implementação, será fornecido também, com cada caso de teste, um arquivo de jogadas (denominado jogadas.txt), que poderá ser redirecionado pela entrada padrão para gerar uma saída esperada. Isso evitará ter que digitar todas as jogadas na mão. O arquivo

conterá uma lista de jogadas que irão começar com o jogador 1 e se alternarão com o jogador 2. Veja abaixo um exemplo do conteúdo de um arquivo de jogadas.

jogadas.txt

```
Leticia
Thiago
b3
a2
b1
a3
b2
a4
```

*Gerar arquivo de estatísticas para análise*: Ao final do jogo, o programa deverá também escrever, na pasta de saída do caso de teste em questão, um arquivo (estatisticas.txt) contendo algumas estatísticas básicas sobre a partida. Entre as informações coletadas para cada jogador, deverão estar o número de tiros errados ("água"), o número de tiros certos ("tiro"), a localização média das suas jogadas válidas e não-repetidas (centróide das jogadas $\bar{t}$), o desvio padrão em torno da localização média das suas jogadas válidas e não-repetidas. O desvio padrão é dado fórmula abaixo, sendo $t_{li}$ e $t_{ci}$ as coordenadas (linha a coluna respectivamente) da jogada $i$, $\bar{t}_l$ e $\bar{t}_c$ as coordenadas (linha a coluna respectivamente) do centróide das jogadas $\bar{t}$ e $N$ o número de jogadas válidas não-repetidas do jogador em questão:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N} \left( (t_{li} - \bar{t_l})^2 + (t_{ci} - \bar{t_c})^2 \right)}$$

Este arquivo deverá conter também a lista dos navios atingidos (não necessariamente afundados) do jogador 1 ordenada pelo nome dos navios (em ordem alfabética), tendo a ordem do navio que levou o primeiro tiro primeiro como critério de desempate. A lista de navios atingidos do jogador 2 seguirá o mesmo critério de ordenação. O arquivo conterá na sequência em cada linha: o nome do jogador literal; a informação de tiros que o jogador errou no formato “Tiros Errados: ##”; a informação de tiros que o jogador acertou no formato “Tiros Acertados: ##”; a localização média com duas casas decimais dos tiros no formato “Localizacao Media: (#.##,#.##)”, sendo o primeiro #.## referente a linha (com a posição *a* equivalendo a 1, a *b* a 2 e assim por diante) e o segundo #.## referente a coluna; e, o desvio padrão da localização média, também com duas casas decimais e no formato “Desvio Padrao da Localizacao: #.##”. Em seguida, serão apresentados os navios do primeiro jogador, iniciados com o título “Navios de @@@” com o nome do jogador literal. Nas linhas seguintes, os navios com o índice da jogada de seu primeiro tiro serão apresentados, no formato “## -

@@@ - ID $$" em que "##" representa o índice da jogada do adversário que atingiu o navio pela primeira vez (ou seja, 01 caso o adversário tenha atingido o navio em sua primeira jogada, 02 na segunda e assim por diante), "@@@" representa o nome literal do navio atingido e "$$" representa o ID do navio. Após as informações sobre o jogador 1 e uma linha vazia, as mesmas informações sobre o jogador 2 serão apresentadas. Um exemplo do arquivo de estatísticas pode ser visto abaixo:

estatisticas.txt

```
Leticia
Tiros Errados: 3
Tiros Acertados: 0
Localizacao Media: (2.00,2.00)
Desvio Padrao da Localizacao: 0.82
Navios de Leticia:
01 - Submarine - ID 01

Thiago
Tiros Errados: 0
Tiros Acertados: 3
Localizacao Media: (1.00,3.00)
Desvio Padrao da Localizacao: 0.82
Navios de Thiago:
```

*Gerar Tabuleiro Válido (Bônus)*: Haverá uma pontuação extra para a funcionalidade de gerar um tabuleiro aleatório. Essa funcionalidade será executada a parte do programa, ou seja, quando o programa for executado na linha de comando com um parâmetro "-gt", ele não deverá realizar o jogo, mas sim gerar um tabuleiro aleatório. Para isso, o programa deverá receber pela linha de comando após o parâmetro "-gt" o nome do arquivo de tabuleiro a ser gerado incluindo a sua localização (assumindo um programa compilado como *prog*, a execução seria algo como "./prog -gt /maquinadoprofessor/meudiretorio/meu_tabu.txt"). Deve-se usar a função geradora de números aleatórios da linguagem C. Assuma que o tabuleiro a ser gerado será sempre 10x10, contendo um navio de cada tipo. O arquivo de saída será idêntico ao modelo do arquivo de tabuleiro já informado anteriormente, ou seja, ele poderia fazer parte de um dos casos de teste.

## Informações Gerais

Alguns arquivos de entrada e respectivos arquivos de saída serão fornecidos para o aluno. O aluno deverá utilizar tais arquivos para testes durante a implementação do trabalho. É de responsabilidade do aluno criar novos arquivos para testar outras possibilidades do programa e garantir seu correto funcionamento. O trabalho será corrigido usando, além dos arquivos

dados, outros arquivos (específicos para a correção e não disponibilizados para os alunos) seguindo a formatação descrita nesse documento. Em caso de dúvida, pergunte ao professor. O uso de arquivos com formatação diferente poderá acarretar em incompatibilidade durante a correção do trabalho e consequentemente na impossibilidade de correção do mesmo (sendo atribuído a nota zero). Portanto, siga estritamente o formato estabelecido.

## Implementação e Verificação dos Resultados

A implementação deverá seguir os conceitos de modularização vistos em sala. O trabalho terá uma componente subjetiva que será avaliada pelo professor para verificar o grau de uso dos conceitos ensinados. Portanto, além de funcionar, o código deverá estar bem escrito para que o aluno obtenha nota máxima.

É extremamente recomendado (para não dizer obrigatório) utilizar algum programa para fazer as comparações do resultado final do programa, isto é, os arquivos de saída gerados, poderão ser comparados com os arquivos de saídas esperadas (fornecidos pelo professor) utilizando o comando *diff*, como visto em sala. O *meld* é uma alternativa gráfica para o *diff*, se você preferir. Esse programa faz uma comparação linha a linha do conteúdo de 2 arquivos. Diferenças na formatação poderão impossibilitar a comparação e consequentemente a correção do trabalho. O programa será considerado correto se gerar a saída esperada idêntica à fornecida com os casos de teste.

Os casos de testes visíveis serão disponibilizados juntamente com esta especificação do trabalho. A pasta com os casos de teste visíveis terá uma subpasta (denominada “Gabarito”) contendo todos os casos fornecidos. Além disso, ela terá outra subpasta (denominada “Testes”) contendo o mesmo conteúdo da pasta Gabarito, porém sem os respectivos arquivos de saída. Caso o programa do aluno esteja funcionando adequadamente, após executá-lo utilizando os casos da pasta de Teste, esta deverá ficar idêntica a pasta Gabarito.

## Regras Gerais

O trabalho deverá ser feito individualmente e pelo próprio aluno, isto é, o aluno deverá necessariamente conhecer e dominar **todos** os trechos de código criados. Cada aluno deverá trabalhar independente dos outros, não sendo permitido a cópia ou compartilhamento de código. O professor irá fazer verificação de plágio. Trabalhos identificados como iguais, em termos de programação (por exemplo, mudar nomes de variáveis e funções não faz dois trabalhos serem diferentes), serão penalizados com a nota zero. Isso também inclui a pessoa que forneceu o trabalho, sendo portanto, de sua obrigação a proteção de seu trabalho contra cópias ilícitas. Proteja seu trabalho e não esqueça cópias do seu código nas máquinas de uso comum.

| | |
|---|---|
|  | Centro Tecnológico<br>Departamento de Informática |
| Disciplina: Programação II | Código: INF09330 |
| Trabalho Prático | Prof. Dr. Thiago Oliveira dos Santos |

## Entrega do Trabalho

O trabalho deverá ser entregue pelo classroom até a data e hora definidas para tal. O trabalho deve ser entregue todo em um único arquivo ".c", nomeado com o nome do aluno e sem espaços, por exemplo "ThiagoOliveiraDosSantos.c". Ele será necessariamente corrigido no sistema operacional linux, qualquer incompatibilidade devido ao desenvolvimento em um sistema operacional diferente é de responsabilidade do aluno. Isso pode inclusive levar a nota zero, no caso de impossibilidade de correção. A correção será feita de forma automática (via *script*), portanto trabalhos não respeitando as regras de geração dos arquivos de saída, ou seja, fora do padrão, poderão impossibilitar a correção. Consequentemente, acarretar na atribuição da nota zero. A pessoa corrigindo não terá a obrigação de adivinhar nome de arquivos, diretórios ou outros. **Siga estritamente o padrão estabelecido!**

## Pontuação e Processo de Correção

*Pontuação*: Trabalhos entregues após o prazo não serão corrigidos (recebendo a nota zero). O trabalho será pontuado de acordo com sua implementação e a tabela abaixo. Os pontos descritos na tabela não são independentes entre si, isto é, alguns itens são pré-requisitos para obtenção da pontuação dos outros. Por exemplo, gerar o arquivo de estatísticas depende de realizar o jogo corretamente. Código com falta de legibilidade e modularização pode perder pontos conforme informado na tabela. Erros gerais de funcionamento, lógica ou outros serão descontados como um todo.

Percebam que no melhor dos casos os pontos da tabela abaixo somam 11 ao invés de 10. Isso foi feito propositalmente para ajudar os alunos esforçados com um ponto extra. Esse ponto, caso obtido, irá complementar uma das notas, do trabalho ou das provas parciais do semestre. Prioridade será dada para a nota com maior peso, porém não ultrapassando a nota máxima.

| Item | Quesitos | Ponto |
|---|---|---|
| Inicializar jogo | Ler os arquivos dos tabuleiros<br>Gerar corretamente o arquivo de Inicialização | 1 |
| Verificar tabuleiro | Verificar se o tabuleiro é válido | 2 |
| Realizar jogo | Receber corretamente as entradas dos jogadores<br>Mostrar as informações do jogo como especificado<br>Mostrar o resultado da partida | 3 |
| Gerar resumo do resultado | Gerar arquivo com o resumo dos resultados | 1 |
| Criar estatísticas | Gerar corretamente o arquivo com as estatísticas. | 2 |

| | | |
|---|---|---|
| | | |
| Tratamento de jogadas não ordinárias | Tratamento de jogadas repetidas<br>Tratamento de jogadas inválidas | 1 |
| Legibilidade e Modularização | Falta de:<br>• Uso de comentários;<br>• Identação do código;<br>• Uso de funções;<br>• Uso de tipos de dados definidos pelo usuário | -2 |
| Função bônus | Gerar Tabuleiro Válido | 1 |

*Processo de correção do trabalho:* O trabalho será corrigido, considerando-se 3 turnos: primeira entrega, tempo de correção e entrevista. A Primeira Entrega ocorrerá na data definida para a Entrega do Trabalho. Ela gerará uma nota da Primeira Entrega (NPE) que servirá como base para a nota final do trabalho, ou seja, o trabalho será pontuado de acordo com o que for entregue nesta etapa. O Tempo de Correção ocorrerá no dia indicado no classroom, e serão aceitas as correções até o prazo também definido no classroom. Neste dia, o aluno terá a possibilidade, dentro do tempo disponibilizado, de corrigir erros encontrados no trabalho. Funcionalidades novas implementadas nessa etapa não serão contabilizadas. As partes corrigidas ganharão um percentual (0% a 100%) do ponto cheio, de acordo com o tipo de correção feita pelo aluno, ou seja, ela gerará a nota do Tempo de Correção (NTC), em que NPE ≤ NTC ≤ 10. Para fazer a correção, o aluno receberá todos os casos de teste escondidos do trabalho. Ele receberá também o *script* que rodará o programa de forma automática (seguindo as regras definidas nesta especificação) para que ele possa consertar possíveis erros de implementação (por exemplo, formatação das saídas, nomes de arquivos, lógica, etc.). No final deste tempo de correção, o aluno deverá reenviar o programa corrigido, desta vez na atividade de correção do trabalho, com comentários explicando cada alteração feita. Cada comentário deverá ser iniciado com a tag "//CORRECAO:" para indicar o que foi alterado. Modificações sem as devidas explicações e tags poderão ser desconsideradas e não pontuadas. Vale a pena ressaltar que não será possível aceitar entregas fora do prazo, dado que os casos de teste escondidos serão liberados no Tempo de Correção. Portanto, se não quiser ter problemas, faça o trabalho e a entrega com antecedência. A Entrevista ocorrerá em uma data posterior a aula de correção e fora do horário de aula (a ser agendado). Ela tem o intuito de levantar o conhecimento dos alunos sobre o próprio trabalho. A nota, NTC, obtida ao final das duas primeiras etapas acima será ponderada com uma nota de entrevista (NE) sobre o trabalho. A nota final do trabalho será calculada com NTC x NE, em que 0 ≤ NE ≤ 1. A entrevista avaliará o conhecimento do aluno a respeito do programa desenvolvido. Como pode ser visto, a nota da entrevista não servirá

para aumentar a nota do trabalho, mas sim para diminuir quando for identificada uma falta de conhecimento sobre o trabalho (i.e., código) entregue.

## Considerações Finais

Não utilizar caracteres especiais (como acentos) no trabalho.

## Erratas

Esse documento descreve de maneira geral as regras de implementação do trabalho. É de responsabilidade do aluno garantir que o programa funcione de maneira correta e amigável com o usuário. Qualquer alteração nas regras do trabalho será comunicada em sala e no portal do aluno. É responsabilidade do aluno frequentar as aulas e se manter atualizado em relação às especificações do trabalho. Caso seja notada qualquer tipo de inconsistência nos arquivos de testes disponibilizados, comunique imediatamente ao professor para que ela seja corrigida e reenviada para os alunos.