


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

Especificação do Trabalho

Snake

Introdução

Este trabalho tem como objetivo avaliar, de forma prática, os conceitos ensinados no curso de Programação II. A meta deste trabalho é implementar um jogo do gênero *Snake*¹ seguindo as especificações descritas neste documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.

O jogo, também conhecido informalmente como Jogo da Cobrinha e comumente disponível em celulares antigos, é um jogo *single-player* no qual o jogador controla uma cobra que se move sobre um mapa bidimensional. O objetivo do jogo é obter a maior pontuação possível sem colidir com obstáculos ou com si mesmo. No mapa (exemplificado na Figura 1) existem itens especiais que, ao serem tocados pela cobra são consumidos, aumentando a pontuação e possivelmente aumentando o tamanho da cobra. Com o passar do tempo, a dificuldade do jogo aumenta, pois quanto maior a cobra mais difícil é traçar um caminho em que ela não colida com si mesma. Nas versões mais populares, o jogo termina quando a cobra alcançar o tamanho máximo, ou a cobra colidir com algum obstáculo ou nela mesma.

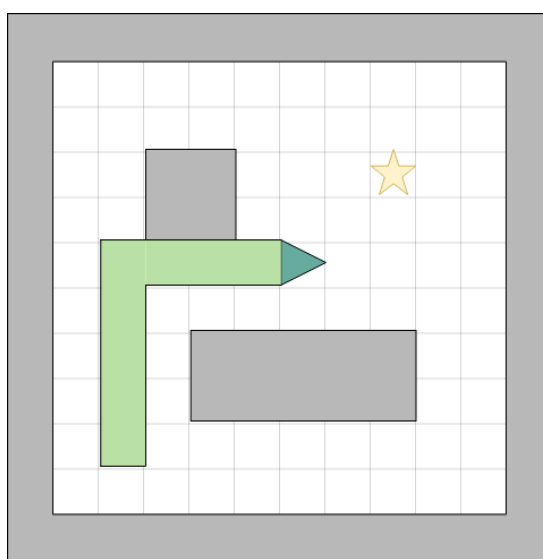



Figura 1- Exemplo de um mapa 12x12 do jogo. Em cinza as paredes (obstáculos), em verde escuro a cabeça da cobra, em verde claro o corpo, e a estrela representa um item especial.

¹ [https://pt.wikipedia.org/wiki/Serpente_\(jogo_eletr%C3%B4nico\)](https://pt.wikipedia.org/wiki/Serpente_(jogo_eletr%C3%B4nico))

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

Descrição do Trabalho


Neste trabalho, você deverá implementar um jogo do gênero *Snake*. O programa deverá rodar no terminal, assim como todos os outros exercícios feitos em sala. De maneira geral, o programa irá iniciar pela linha de comando (terminal) e irá ler o estado inicial do jogo, que consiste na definição do mapa e a posição inicial da cobra. Com o jogo inicializado, serão executados movimentos dados pelo usuário na entrada padrão. Durante a execução, o programa apresentará os estados parciais na saída padrão até o jogo terminar, hora em que ele apresentará o resultado final e salvará arquivos do jogo contendo outros resultados.

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*, *realizar jogo*, *gerar resumo de resultado*, *gerar estatísticas*, *gerar ranking*, *gerar heat map* e a *função bônus (para tratar túneis)*. 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 o arquivo de definição do mapa do jogo: *mapa.txt*, sendo alguns exemplos fornecidos com esta especificação. O programa deverá ler o arquivo e preparar o ambiente de jogo. Caso o programa não consiga ler o arquivo (*e.g.*, porque ele não existe), 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). Todas as saídas em forma de arquivo do trabalho devem ser escritas na pasta *saida* dentro do mesmo diretório.

O arquivo *mapa.txt* definirá todos os parâmetros do mapa: seu tamanho e o conteúdo de cada célula, ou seja, o que deve estar em cada posição. A direção inicial do movimento da cobra será sempre para a direita. Um mapa é definido como um arranjo de N linhas contendo M colunas, semelhante à uma matriz. Cada posição dessa matriz (denominada célula) pode ser de diferentes tipos como mostrado na tabela abaixo.

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

| Tipo de célula | Representação | Em caso de colisão |
|--------------------------|--------------------|--|
| Vazio | (espaço em branco) | Nada acontece |
| Parede | # | Fim de jogo |
| Comida | * | Cobra cresce uma unidade Ganha +1 ponto |
| Dinheiro | \$ | Ganha +10 pontos |
| Posição inicial da cobra | > | – |
| Corpo da cobra | o | Fim de jogo |

Um exemplo desse arquivo de definição do mapa é mostrado a seguir.

`mapa.txt`


```
6 6
#####
#   #
#> * #
#*  *#
#   $#
#####
```

O exemplo acima define um mapa com 6 linhas e 6 colunas, paredes (#) em suas bordas, um dinheiro (\$) e três comidas (*) espalhadas pelo mapa. Além disso, a posição inicial da cobra está na terceira linha e segunda coluna (posição do ">").

Como resultado do passo de *Inicializar o Jogo*, o programa deve gerar o arquivo `inicializacao.txt`. Esse arquivo deve conter o mapa que foi lido. Depois da representação do mapa, deve haver a linha de texto "A cobra começará o jogo na linha @ e coluna #", substituindo @ pela linha e # pela coluna da posição de onde a cobra foi definida no arquivo `mapa.txt`. Um exemplo desse arquivo é mostrado a seguir.

`inicializacao.txt`

```
#####
#   #
#> * #
#*  *#
#   $#
#####
A cobra comecara o jogo na linha 3 e coluna 2
```

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

Realizar jogo: Uma vez preparado o jogo, as jogadas poderão começar a serem processadas. O jogo terminará quando não houver mais comidas pelo mapa ou a cobra morrer (*i.e.*, quando ela colidir com alguma parede ou em si mesma). As jogadas deverão ser lidas da entrada padrão de dados, permitindo dessa forma, um redirecionamento a partir de arquivo. O fluxo da jogada segue:

- A jogada será fornecida pelo usuário no formato “@”, em que @ é um caractere que define o movimento: “h” para movimento horário, “a” para movimento anti-horário e “c” para continuar (*i.e.*, não alterar a direção).
- Após ler a jogada do jogador, ela deve ser processada. O movimento que a cobra fará depende da direção em que ela estava e do movimento que o jogador definiu. Por exemplo, caso a cobra estivesse indo para a direita e o movimento do jogador fosse horário (“h”), a cobra deveria movimentar-se uma célula para baixo e alterar sua direção para baixo. É importante notar que somente a cabeça da cobra que se movimenta instantaneamente. O resto do corpo da cobra somente segue a posição da parte da cobra à sua frente, como mostrado na Figura 2. Além disso, caso a cobra saia do mapa, ela deve reaparecer do outro lado do mapa. Por exemplo, caso a cobra saia pela borda de cima, ela deve aparecer na parte de baixo do mapa. Assuma que o tamanho máximo que a cobra pode alcançar são 100 partes.

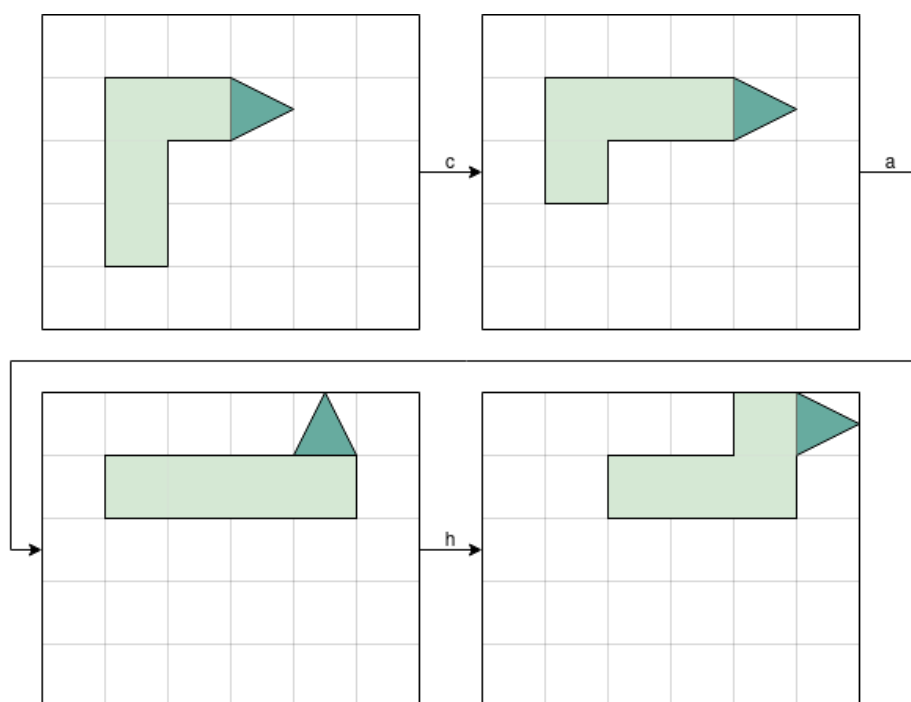



Figura 2 - Exemplos de movimentos. O primeiro quadro representa o estado inicial, onde a partir deles são feitos uma sequência de movimentos: “c”, “a” e “h”.

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

- Após um movimento, o programa deverá exibir na saída padrão (ou seja, na tela) o estado atual do jogo (o mapa com a cobra) e a pontuação até o momento no seguinte padrão:

```
Estado do jogo apos o movimento '#':
<estado do mapa>
Pontuacao: @
```

Substituindo “#” pelo movimento feito (“c”, “a” ou “h”), “<estado do mapa>” pelo estado do mapa e “@” pela pontuação no momento. Exemplo:

```
Estado do jogo apos o movimento 'c':
#####
#      #
# >*  #
#*   *#
#    $#
#####
Pontuacao: 0
```

No caso de a cobra também possuir corpo (ou seja, seu tamanho ser maior que um), cada parte dele deve ser representada pelo caractere “o”. Além disso, a representação da cabeça da cobra depende de sua direção: “>” para esquerda, “<” para direita, “^” para cima e “v” para baixo.

- Caso após o movimento executado não haja mais nenhuma comida no mapa (mesmo que haja célula de dinheiro), o jogo encerra com a seguinte mensagem:

```
Voce venceu!
Pontuacao final: #
```

Substituindo “#” pela pontuação final.

Caso o movimento executado resulte na colisão com um obstáculo ou com a própria cobra, o estado a ser impresso deve apresentar todas as partes da cobra com “X” incluindo a célula da colisão. Depois disso, exibe-se a mensagem a seguir:

```
Game over!
Pontuacao final: #
```

Novamente substituindo “#” pela pontuação final.

Para facilitar a implementação, será fornecido também, com cada caso de teste, um arquivo de movimentos (denominado `movimentos.txt`), que poderá ser redirecionado pela entrada padrão para gerar uma saída esperada. Isso evitará ter que digitar todas as jogadas na mão. Veja abaixo um exemplo do conteúdo de um arquivo de movimentos. Considere que os casos dados sempre serão suficientes para finalizar o jogo, com uma vitória ou uma derrota. Cabe ao aluno testar outros casos.

`jogadas.txt`

```
c
c
c
h
c
```



Centro Tecnológico
Departamento de Informática

Disciplina: Programação I

Código: INF09330

Trabalho Prático

Prof. Dr. Thiago Oliveira dos Santos

h
h
a
c

Gerar resumo de resultado: O programa deverá também salvar na pasta de saída do caso de teste em questão, um arquivo (denominado `resumo.txt`) contendo o resumo do resultado do jogo. O arquivo de resumo deverá conter uma descrição do que houve em cada movimento onde houve alguma variação significativa: crescimento da cobra, dinheiro ou fim de jogo. Cada linha deve descrever a alteração causada por um movimento, sendo que cada alteração possui um padrão específico de saída, como mostrado a seguir.

No caso de dinheiro:

```
Movimento # (@) gerou dinheiro
```

No caso de crescimento da cobra sem terminar o jogo (ou seja, ainda existe comida no mapa):

```
Movimento # (@) fez a cobra crescer para o tamanho ##
```

No caso de crescimento da cobra onde o jogo é terminado (ou seja, comeu a última comida do mapa):

```
Movimento # (@) fez a cobra crescer para o tamanho ##, terminando o jogo
```

No caso de fim de jogo por colisão:


```
Movimento # (@) resultou no fim de jogo por conta de colisao
```

Exemplo completo do arquivo de resumo:

`resumo.txt`

```
Movimento 2 (c) fez a cobra crescer para o tamanho 2  
Movimento 4 (h) fez a cobra crescer para o tamanho 3  
Movimento 5 (c) gerou dinheiro  
Movimento 9 (c) fez a cobra crescer para o tamanho 4, terminando o jogo
```

Gerar ranking de posições mais frequentes: O programa deverá também salvar na pasta de saída do caso de teste em questão, um arquivo (denominado `ranking.txt`) contendo o as posições do mapa (que foram visitadas pelo menos uma vez) ordenadas (decrementalmente) pela quantidade vezes que a cabeça da cobra passou por ali. Usar a linha e depois a coluna como critério de desempate, ambos em ordem crescente. Por exemplo, em caso de células com frequências iguais, a célula com linha menor deve vir

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

antes. Caso a linha também seja igual, a célula com coluna menor deve vir antes. O arquivo de ranking deverá apresentar cada célula em uma linha seguida do número de vezes que ela foi visitada, como mostrado a seguir.

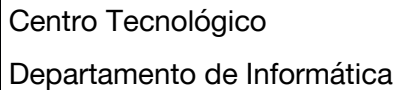
```
(#L, #C) - #Q
(#L, #C) - #Q
(#L, #C) - #Q
```

Exemplo do arquivo de ranking:

ranking.txt do caso simples/3 (primeiras 25 linhas)

```
(5, 8) - 3
(5, 9) - 3
(5, 10) - 3
(5, 11) - 3
(5, 12) - 3
(5, 13) - 3
(5, 14) - 3
(5, 15) - 3
(5, 16) - 3
(5, 17) - 3
(5, 18) - 3
(5, 19) - 3
(5, 20) - 3
(5, 1) - 2
(5, 3) - 2
(5, 5) - 2
(5, 6) - 2
(5, 7) - 2
(5, 21) - 2
(5, 22) - 2
(1, 1) - 1
(1, 2) - 1
(1, 3) - 1
(1, 4) - 1
(1, 5) - 1
```

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. As informações que devem ser coletadas são: número de movimentos, número de movimentos sem pontuar, número de movimentos para baixo, número de movimentos para cima, número de movimentos para a esquerda e número de movimentos para a direita. O padrão desse arquivo segue o seguinte formato:




Código: INF09330

Prof. Dr. Thiago Oliveira dos Santos

Um exemplo do arquivo de estatísticas pode ser visto abaixo:

Gerar heat map: Ao final do jogo, o programa deverá também escrever, na pasta de saída do caso de teste em questão, um arquivo (`heat_map.txt`) contendo o mapa que descreve o número de vezes com que a cobra passou por cada célula. O número de passagens por cada espaço será mostrado como uma matriz, onde o valor na linha i e coluna j denota o número de vezes em que a cobra passou pela linha i e coluna j do mapa. Um exemplo do arquivo de *heat map* pode ser visto abaixo:

Tratar túnel (Bônus): Haverá uma pontuação extra para a funcionalidade de túneis, que será avaliada por um conjunto de casos de testes separados. Nesses casos, haverá mais um tipo de célula: o túnel. Em todos os casos esses casos haverá duas células de túnel no mapa, representadas pelo caractere “@”. Quando a cobra colidir com uma dessas células, ela deve ser teleportada para a posição da outra célula. Além disso, a cobra deve continuar na mesma direção em que estava antes. Portanto, se a cobra colide com o lado direito de um dos túneis (indo para a esquerda), ela deve ser teleportada para o lado esquerdo do outro túnel

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

(continuando a ir para a esquerda). Perceba que a cobra nunca vai ocupar a posição de um túnel. Portanto, em mapas com túnel, o valor no heat map dessas posições será sempre zero.

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 neste 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

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

“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.

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 das máquinas do laboratório, 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.

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

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 | Pont o |
|------------------------------|--|-----------|
| Inicializar jogo | Ler o arquivo do mapa Gerar corretamente o arquivo de Inicialização (inicializacao.txt) | 2 |
| Realizar jogo | Receber corretamente as entradas do jogador Mostrar as informações do jogo como especificado Mostrar o resultado do jogo | 3 |
| Gerar resumo do resultado | Gerar arquivo com o resumo dos resultados (resumo.txt) | 1 |
| Gerar ranking | Gerar arquivo com o resumo dos resultados (ranking.txt) | 2 |
| Gerar estatísticas | Gerar corretamente o arquivo com as estatísticas (estatisticas.txt) | 1 |
| Gerar heat map | Gerar corretamente o arquivo com as estatísticas (heat_map.txt) | 1 |
| Legibilidade e Modularização | Falta de: <ul style="list-style-type: none"> • Uso de comentários; • Identação do código; • Uso de funções; • Uso de tipos de dados definidos pelo usuário • Manutenibilidade (permitir modificações surpresas) | -3 |
| Função bônus | Tratar os túneis | 1 |

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


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. 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. Neste mesmo dia, será testada a manutenibilidade do código, sendo exigida a incorporação de pequenas mudanças no comportamento do jogo. Quanto mais modular estiver o código, mais fácil será de implementar as mudanças. 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 $NPE \leq NTC \leq 10$. 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 $NTC \times NE$, em que $0 \leq NE \leq 1$. 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

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

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.