	Centro Tecnológico Departamento de Informática
Disciplina: Programação I	Código: INF15927
Trabalho Prático	

Especificação do Trabalho

Freeway

Introdução

Este trabalho tem como objetivo avaliar, de forma prática, os conceitos ensinados no curso de Programação I. A meta deste trabalho é implementar uma versão simplificada clássico jogo de Atari chamado *Freeway*[1], popularmente conhecido no Brasil como Jogo da Galinha, seguindo as especificações descritas neste documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.

O jogo consiste em um desafio de um jogador, no qual o mesmo controla uma galinha que se move verticalmente em um mapa bidimensional. Logo, o objetivo do jogo é guiar a galinha através de uma rodovia movimentada, desviando dos carros, para chegar em segurança ao outro lado do mapa antes que todas as vidas acabem. Uma vida é perdida a cada vez que a galinha é atropelada.

Descrição do Trabalho

Neste trabalho, você deverá implementar um jogo similar ao Jogo da Galinha. O programa deverá rodar no terminal, assim como todos os outros exercícios feitos em sala. De maneira geral, o programa é iniciado via linha de comando (terminal) e realiza a leitura do estado inicial do jogo, que inclui a definição do mapa, das configurações das pistas, dos carros e da galinha.

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

[1] https://en.m.wikipedia.org/wiki/Freeway_%28video_game%29

até o jogo terminar, momento em que será apresentado o estado final e alguns arquivos com informações sobre a partida serão criados.

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 mapa de calor* e a *função bônus (para gerar iterações diferentes no jogo)*. 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 e, caso o usuário tenha esquecido de informar o nome do diretório, o programa deverá exibir uma mensagem de erro (ex. "ERRO: Informe o diretorio com os arquivos de configuracao."), finalizando sua execução. Dentro desse diretório, espera-se encontrar dois arquivos, um com as configurações iniciais do jogo (`config_inicial.txt`) e outro com o desenho dos personagens do jogo (`personagens.txt`). O programa deverá ler ambos os arquivos e preparar o ambiente de jogo. Caso o programa não consiga ler um deles (*e.g.*, porque alguns deles não existem), 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 fornecido. Considerar que a pasta saida já estará criada nos casos de teste.

Abaixo é apresentado um modelo de como o `config_inicial.txt` é escrito:

config_inicial.txt

```
animacao
largura_mapa qtd_pistas

direcao velocidade num_carro X1 X2 … Xn
direcao velocidade num_carro X1 X2 … Xn

galinha X_galinha vidas
```

Com o arquivo modelo é possível ver que a primeira linha possui um valor inteiro chamado *animacao* que determina se a funcionalidade bônus de animação vai ser ativada ou não (sendo 0 para a execução normal e 1 para o modo com ponto bônus), isso será detalhado posteriormente. A segunda linha do arquivo contém dois números inteiros: *largura_mapa* e

qtd_pistas, que representam, respectivamente, a quantidade de colunas do mapa e o número de pistas que ele terá. É importante destacar que a quantidade de linhas do mapa não é fornecida diretamente nesse arquivo e deve ser calculada pelo código, como será explicado adiante. Ainda assim, o mapa deve respeitar os limites máximos de 35 linhas e 100 colunas, podendo conter até 12 pistas e no máximo 10 carros por pista.

O restante do arquivo contém as informações referentes às pistas, sendo que o número de linhas restantes é igual ao valor de *qtd_pistas*. Neste jogo, a pista do topo (ou seja, a primeira pista) representa o ponto final do mapa e, por isso, nunca possui carros, assim, a terceira linha do arquivo é sempre vazia. Da mesma forma, a pista onde a galinha aparece no início do jogo (última pista do mapa) também nunca contém carros e sempre aparece na última linha do arquivo de configurações. Essa linha possui o seguinte formato: *galinha X_galinha* vidas, em que galinha é sempre representada pela letra "G", *X_galinha* é um número inteiro que indica a posição central da galinha no eixo X, e *vidas* corresponde à quantidade de vidas que a galinha terá no jogo.

As demais linhas do arquivo, que aparecem entre a terceira linha e a última, definem a configuração das pistas intermediárias, que podem ou não conter carros. Quando uma pista não possui carros, sua linha será completamente em branco, assim como ocorre na linha da primeira pista. Caso a pista contenha carros, a linha seguirá o seguinte formato: *direcao velocidade num_carro X1 X2 … Xn*, onde

- *direcao* é um caractere que indica o sentido de movimentação dos carros da pista, podendo assumir os valores "D" ou "E". Quando o valor é "D", os carros se deslocam da esquerda para a direita no mapa; quando é "E", o movimento ocorre da direita para a esquerda, ou seja, no sentido oposto.
- *velocidade* é representada por um número inteiro que indica quantas posições todos os carros da pista devem avançar a cada jogada.
- *numCarros* é um número inteiro que representa a quantidade total de carros presentes na pista.
- *X1 X2 … Xn* são números inteiros que representam as posições no eixo X correspondentes ao ponto central de cada carro na pista (sem sobreposições). O valor de *n* é igual ao número total de carros (*num_carro*), e *Xn* representa a posição central do último carro daquela pista.

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

config_inicial.txt

```
0
50 6

D 2 5 2 7 15 27 35
E 1 2 5 25

E 2 3 10 25 32
G 13 3
```

No exemplo acima, o jogo não é executado na funcionalidade bônus, uma vez que o valor é 0. A segunda linha especifica um mapa com 50 colunas e 6 pistas. A terceira linha vazia

representa a primeira pista, consequentemente o ponto final do jogo. As linhas 4 até 7 apresenta as configurações das pistas 2 até 5. Note que a pista 4 não possui carros, uma vez que a linha que representa essa pista está vazia. Por fim, a última linha que começa com a letra “G” representa a configuração da pista em que a galinha nasce, sendo o X igual a 13 e a galinha possuindo 3 vidas.

É importante destacar que não haverá sobreposição entre os carros e as coordenadas dos arquivos consideram a posição (1 1) como o canto superior esquerdo do mapa.

O arquivo *personagens.txt* define os desenhos dos personagens do jogo. Este arquivo contém 5 desenhos, cada um com 2 linhas e 3 colunas. O primeiro desenho (linhas 1 e 2) é o que representa a galinha. O primeiro desenho do carro (linhas 3 e 4) representa o desenho padrão do carro, o restante do arquivo representa os desenhos que serão utilizados na funcionalidade bônus. Os jogos sem o ponto extra devem sempre desenhar o desenho padrão.

É importante frisar que a posição central (citada anteriormente) de todos os personagens é o segundo caractere da primeira linha do desenho. Considerando a galinha, por exemplo, a posição central é o caractere “_”.

Um exemplo desse arquivo de definição dos personagens é mostrado a seguir.

personagens.txt

```
^ ^
(o)
[-]
---
[-]
\-\
[-]
|-|
[-]
/-/
```

Abaixo é mostrado como deve ser exibido no mapa o desenho da galinha.

Desenho da galinha:

```
^ ^
(o)
```

Para compreender melhor como tratar as animações do ponto extra, leia o tópico "Animação (Bônus)" dessa especificação.

Como resultado do passo de *Inicializar o Jogo*, o programa deve gerar o arquivo `inicializacao.txt`. Esse arquivo deve conter o desenho do estado inicial do jogo com as configurações fornecidas. Depois da representação do mapa, deve haver a linha de texto “A posicao central da galinha iniciara em (X Y).”, substituindo "X" pela coluna e "Y" pela linha da posição de onde a galinha. Como o valor do eixo Y não é fornecido diretamente no arquivo de configuração, ele precisa ser calculado dentro do código. Abaixo, é apresentado

um exemplo de um estado inicial do jogo que pode auxiliar na compreensão de como esse valor deve ser calculado.

`inicializacao.txt`

```
|-------------------------------------------------|
|                                                 |
|                                                 |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- |
|[-]  [-]     [-]         [-]     [-]             |
|---  ---     ---         ---     ---             |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- |
|   [-]                 [-]                       |
|   ---                 ---                       |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- |
|           ^ ^                                   |
|           (o)                                   |
|-------------------------------------------------|
A posicao central da galinha iniciara em (13 10).
```

Observe que o mapa possui uma borda meramente estética, que precisa ser desenhada, mas não ocupa o espaço interno útil do mapa. Desconsiderando essa borda, é possível identificar um padrão na organização das pistas: cada pista é composta por duas linhas em branco, que podem ser utilizadas pelos personagens do jogo, seguidas por uma linha com o padrão “-- ” (dois traços e um espaço em branco) localizado entre as pistas. A galinha não pode ficar entre as pistas, ou seja, a linha tracejada não é uma posição válida para a galinha. Com base nesse padrão, é possível calcular o número total de linhas do mapa e, a partir disso, determinar corretamente o valor do eixo Y correspondente à posição central de cada personagem.

***Realizar jogo*:** Uma vez que o jogo esteja preparado, as jogadas poderão começar a ser processadas. O jogo terminará quando a galinha cruzar todas as pistas, consequentemente chegando no topo do mapa, ou quando perder todas as vidas.

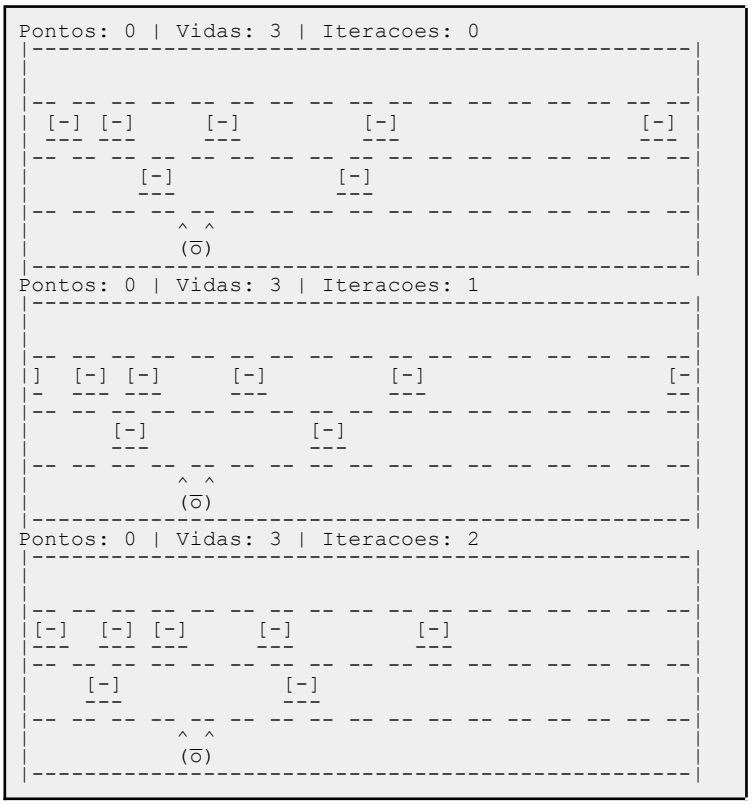
As jogadas deverão ser lidas da entrada padrão, permitindo o redirecionamento a partir de um arquivo (`entrada.txt`), ou seja, conforme feito nos exercícios do BOCA e sem precisar abrir o arquivo em código. Cada iteração de uma partida deve seguir, necessariamente na mesma ordem, as seguintes etapas:

- Verifique se a galinha chegou ao topo do mapa (vitória) ou se ela perdeu todas as vidas (derrota);

- Leia a jogada fornecida pelo usuário no formato “@”, onde “@” é um caractere que define o movimento, podendo ser:
  - "w" para mover para frente.
  - "s" para mover para trás.
  - “ ” (espaço), a galinha deve ficar imovel e passar a vez.

- Realize o movimento da galinha conforme o comando recebido, mudar de pista ou não fazer nada. Dessa forma, a galinha nunca ficará parcialmente entre duas pistas, ela estará sempre completamente posicionada em uma única pista.

- Desloque todos os carros conforme a velocidade e o sentido da pista. Caso algum carro ultrapasse a borda do mapa durante o movimento, ele deverá reaparecer na primeira posição do lado oposto da mesma pista. Abaixo, é apresentada uma sequência de telas que exemplifica esse comportamento. Observe que os carros da segunda pista estão se movendo para a direita. Quando o carro mais à direita dessa pista ultrapassa a borda direita do mapa, ele reaparece no início da mesma pista. Não é necessário verificar se essa posição está livre, pois nunca haverá colisão entre os carros.



- Verifique se houve colisão entre a galinha e algum carro. Em caso positivo, a galinha deve perder uma vida, zerar sua pontuação e retornar à posição inicial. Caso não ocorra colisão, adicione 1 ponto à pontuação atual sempre que a galinha atravessar uma pista para frente, e 10 pontos ao atingir o topo do mapa. Movimentos para trás, mesmo sem colisão, não devem conceder pontos. Importante notar que o X da galinha ficará sempre fora das bordas esquerda e direita do mapa, de forma que não seja necessário verificar a colisão entre a galinha e carros que estejam tocando o início ou final da pista.

- Incrementar o contador de iterações da partida (começando na iteração 0 logo após a etapa de inicialização).

- Exibir na saída padrão (ou seja, na tela) o estado atual do jogo (incluindo o mapa com a galinha e os carros) além da pontuação acumulada até o momento, a quantidade de vidas da galinha e número da iteração do momento, no seguinte formato:

```
Pontos: # | Vidas: $ | Iteracoes: @
|-------------------------------------------------|
|                                                 |
|                                                 |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
| [-] [-]     [-]         [-]                 [-] |
| --- ---     ---         ---                 --- |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|         [-]             [-]                     |
|         ---             ---                     |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|            ^ ^                                  |
|            (o)                                  |
|-------------------------------------------------|
```

Onde #, $ e @ podem assumir o valor de um número inteiro positivo qualquer. Abaixo é apresentado um exemplo de uma tela:

```
Pontos: 0 | Vidas: 3 | Iteracoes: 0
|-------------------------------------------------|
|                                                 |
|                                                 |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
| [-] [-]     [-]         [-]                 [-] |
| --- ---     ---         ---                 --- |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|         [-]             [-]                     |
|         ---             ---                     |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|            ^ ^                                  |
|            (o)                                  |
|-------------------------------------------------|
```

No caso das verificações de fim de jogo (vitória ou derrota), deve ser exibida a mensagem "Parabens! Voce atravessou todas as pistas e venceu!" em caso de vitória, e "Voce perdeu todas as vidas! Fim de jogo." em caso de derrota.

Para facilitar a implementação, será fornecido também, com cada caso de teste, um arquivo de movimentos (denominado `entrada.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.

entrada.txt

```
W
W
S
W
W
W
W
S
S
W
```

Gerar resumo: 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 de onde ocorreu alguma variação significativa, sendo elas: a galinha foi atropelada por um carro e fim de jogo. Cada linha deve descrever a alteração causada, sendo que cada uma possui um padrão específico de saída.

Nos exemplos a seguir, os símbolos possuem significados específicos: "@" indica a iteração em que o evento ocorreu, "#" representa o índice da pista, "$" o índice do carro e "X" e "Y" devem ser substituídos pela posição central da galinha no momento em que foi atropelada. Os índices das pistas e dos carros devem obedecer à ordem em que aparecem no arquivo *config_inicial.txt*, sendo que a primeira pista lida possui índice 1 , e o primeiro carro lido daquela pista também possui índice 1, e assim sucessivamente. Os índices dos carros reiniciam em 1 em cada pista. Com isso, os registros devem ser formatados conforme os padrões estabelecidos, respeitando essa numeração e substituindo os valores correspondentes de forma adequada.

- Colisão entre a um carro e a galinha:

```
[@] Na pista # o carro $ atropelou a galinha na posicao (X,Y).
```

- Fim de jogo:

```
[@] Fim de jogo.
```

Um exemplo completo do arquivo de resumo pode ser visto em:

resumo.txt

```
[6] Na pista 5 o carro 2 atropelou a galinha na posicao (10,3).
[8] Na pista 2 o carro 1 atropelou a galinha na posicao (10,11).
[10] Na pista 2 o carro 1 atropelou a galinha na posicao (10,11).
[10] Fim de jogo
```

Observe que os registros do resumo devem seguir a ordem de processamento de eventos fornecida em "Realizar Jogo". Além disso, a posição atingida do inimigo utiliza a linha e a coluna de cima para baixo e da esquerda para direita, respectivamente.

Gerar ranking: O programa também deve salvar um arquivo na pasta de saída do caso de teste em questão, chamado (*ranking.txt*), que conterá um ranking dos atropelamentos que a galinha sofreu durante o jogo. Esse arquivo deve ser ordenado em ordem crescente pelo índice da pista, seguindo a mesma sequência de leitura do arquivo *config_inicial.txt*. Em caso de empate no índice da pista, o critério de desempate será o índice do carro (ordem crescente), que também deve respeitar a ordem de leitura do arquivo de configuração. Se ainda houver empate, a iteração em que ocorreu o atropelamento será usada para definir a ordem final (ordem decrescente).

O arquivo de ranking deve começar com a primeira linha contendo o texto “id_pista,id_carro,iteracao”, seguido pelos dados dos inimigos em cada linha subsequente separado com uma vírgula. Abaixo é apresentado um exemplo de um arquivo gerado:

`ranking.txt`

```
id_pista,id_carro,iteracao
2,4,18
5,1,22
5,2,5
```

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 a serem coletadas incluem: o número de movimentos (para frente e para trás), a altura máxima que a galinha chegou, a altura máxima que a galinha foi atropelada, a altura mínima que a galinha foi atropelada e o número de movimentos na direção oposta (para trás). Para a altura da galinha, considere o valor de Y do ponto central dela. Conforme mencionado no tópico "Inicializar jogo", os arquivos consideram a posição superior esquerda do mapa como o ponto (1,1). Como a galinha inicia na pista mais inferior, o valor de Y do seu ponto central diminui à medida que ela avança para cima. Por isso, é necessário converter esse valor para que a impressão represente corretamente a altura que a galinha alcançou, ou seja, é preciso ajustar o ponto central inicial da galinha para que ele aumente conforme ela se aproxima do topo.

O padrão deste arquivo segue o formato mostrado no exemplo abaixo:

`estatisticas.txt`

```
Numero total de movimentos: 10
Altura maxima que a galinha chegou: 16
Altura maxima que a galinha foi atropelada: 14
Altura minima que a galinha foi atropelada: 6
Numero de movimentos na direcao oposta: 0
```

Gerar Mapa de Calor: Ao final da execução do jogo, o programa deve gerar um arquivo chamado heatmap.txt, localizado na pasta de saída referente ao caso de teste em execução. Este arquivo representa um mapa de calor que contabiliza a quantidade de vezes que a galinha passou por cada célula do mapa, bem como as linhas onde ela foi atropelada. A cada iteração, incluindo a iteração 0 (logo após a inicialização do jogo), que galinha permaneça em uma célula, deve-se incrementar em 1 o seu valor. É importante destacar que deve ser considerado todas as posições da galinha, sendo assim, seis células atualizadas por iteração. Quando ocorrer um atropelamento, todas as linhas ocupadas pela galinha (lembrando que ela ocupa sempre duas linhas) devem ser marcadas com o caractere asterisco (*), sinalizando o evento. Além disso, é importante destacar que, ao ser atropelada, a galinha retorna à posição inicial, e esse retorno à posição original também deve ser registrado no mapa de calor. Os valores registrados no mapa de calor não devem ultrapassar o limite de 99; caso esse valor seja excedido, ele deve ser mantido forçadamente em 99. Para facilitar a visualização do arquivo, cada número no mapa deve ser formatado com exatamente três caracteres de largura, utilizando a notação "%2d".

Um exemplo do arquivo pode ser visto abaixo:

`heatmap.txt` de exemplo

```
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  0  0  0  0  0  0  0  0  0
```

***Animação (Bônus)*:** Haverá uma pontuação extra para quem implementar a funcionalidade de animação dos carros. O arquivo *config_inicial.txt* contém, em sua primeira linha, uma flag que indica se a animação está ativada (1) ou desativada (0). Caso a flag esteja definida como 1, os desenhos dos inimigos devem ser alternados a cada iteração. Por exemplo, o primeiro desenho é exibido na primeira iteração, seguido pelo segundo desenho na

próxima iteração, o terceiro desenho na seguinte, e o quarto desenho na próxima, recomeçando o ciclo quando os quatro padrões de carros forem desenhados. Além disso, ao ocorrer um atropelamento, a velocidade da pista correspondente deve ser reduzida em uma unidade. No entanto, se a velocidade atual da pista já for igual a 1, ela deve permanecer inalterada. Essa funcionalidade não é obrigatória, mas será considerada como critério de pontuação extra na avaliação do trabalho.

Um exemplo parcial pode ser visto nas quatro iterações abaixo:

Saída com animação de exemplo para o caso 1:

```
Pontos: 0 | Vidas: 3 | Iteracoes: 0
|--------------------------------------------------|
|                                                  |
|                                                  |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
| [-] [-]     [-]         [-]                 [-]  |
| --- ---     ---         ---                 ---  |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|         [-]             [-]                      |
|         ---             ---                      |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|           ^ ^                                    |
|           (o)                                    |
|--------------------------------------------------|
Pontos: 0 | Vidas: 3 | Iteracoes: 1
|--------------------------------------------------|
|                                                  |
|                                                  |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|]  [-] [-]     [-]         [-]                 [-|
|\   \-\ \-\     \-\         \-\                 \-|
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|      [-]             [-]                         |
|      \-\             \-\                         |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|           ^ ^                                    |
|           (o)                                    |
|--------------------------------------------------|
Pontos: 0 | Vidas: 3 | Iteracoes: 2
|--------------------------------------------------|
|                                                  |
|                                                  |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|[-]  [-] [-]     [-]         [-]                  |
||-|  |-| |-|     |-|         |-|                  |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|    [-]             [-]                           |
|    |-|             |-|                           |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|           ^ ^                                    |
|           (o)                                    |
|--------------------------------------------------|
Pontos: 0 | Vidas: 3 | Iteracoes: 3
|--------------------------------------------------|
|                                                  |
|                                                  |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|   [-]  [-] [-]     [-]         [-]               |
|   /-/  /-/ /-/     /-/         /-/               |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|   [-]             [-]                            |
|   /-/             /-/                            |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|           ^ ^                                    |
|           (o)                                    |
|--------------------------------------------------|
Pontos: 0 | Vidas: 3 | Iteracoes: 4
|--------------------------------------------------|
|                                                  |
|                                                  |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|    [-]  [-] [-]     [-]         [-]              |
|    ---  --- ---     ---         ---              |
|-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --|
|[-]            [-]                                |
```

	Centro Tecnológico Departamento de Informática
Disciplina: Programação I	Código: INF15927
Trabalho Prático	

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 na incompatibilidade durante a correção do trabalho e consequentemente na impossibilidade de sua correção (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 corretamente, 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 e feito 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 e é muito útil no desenvolvimento do trabalho. Diferenças na formatação poderão impossibilitar a comparação e consequentemente impossibilitar a correção do trabalho. O programa será considerado correto se gerar a saída esperada idêntica à fornecida com os casos de teste.

O trabalho será corrigido com um script de correção automática. Os casos de testes visíveis serão disponibilizados juntamente com esta especificação do trabalho, sendo os casos ocultos disponibilizados em um momento posterior. A estrutura de diretórios e forma de execução do script de correção será disponibilizada em um arquivo README.md na documentação do script a ser fornecida juntamente com o trabalho.

	Centro Tecnológico Departamento de Informática
Disciplina: Programação I	Código: INF15927
Trabalho Prático	

Regras Gerais

O trabalho deverá ser feito individualmente e pelo próprio aluno (sem ajuda de terceiros ou uso de IA para geração de código). 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 entre outros não faz dois trabalhos serem diferentes), serão penalizados com a nota zero e submetidos para penalidades adicionais em instâncias superiores. 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. Caso necessário, o professor poderá realizar uma entrevista para verificar o domínio sobre o código.

Entrega do Trabalho

O trabalho deverá ser entregue pelo classroom (turma de Engenharia da Computação) e AVA (turma de Ciência da Computação) 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, sem espaços e sem acentos, 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. Faça backups constantes do seu trabalho, uma vez que não é raro perder tudo perto da data de entrega. É responsabilidade do aluno ter cópias de segurança na nuvem. 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. A pontuação de um item será dada pelo número de casos de testes corretos para aquele item. Haverá o mesmo número de casos visíveis e invisíveis (que serão conhecidos no dia da correção).

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 o arquivo do mapa Gerar corretamente o arquivo de Inicialização (`inicializacao.txt`)	1
Realizar jogo	Receber corretamente as entradas do jogador Mostrar as informações do jogo como especificado Mostrar o resultado do jogo Usar a saída padrão	3
Gerar resumo do resultado	Gerar arquivo com o resumo dos resultados (`resumo.txt`)	1
Gerar Ranking	Gerar arquivo com a ordenação solicitada (*ranking.txt*)	1
Gerar estatísticas	Gerar corretamente o arquivo com as estatísticas (`estatisticas.txt`)	1
Gerar mapa de calor	Gerar corretamente o arquivo com o mapa de calor (`heatmap.txt`)	1
Manutenibilidade	Permitir modificações surpresas e serem passadas no dia da correção do trabalho pelos alunos.	2

Função bônus	Animação dos inimigos	1
Legibilidade e Modularização	Falta de: • Uso de comentários; • Identação do código; • Uso de funções; • Uso de tipos de dados definidos pelo usuário	-1

Processo de correção do trabalho: O trabalho será corrigido, considerando-se 3 turnos: Primeira Entrega, Tempo de Correção e Manutenção e Entrevista (sob demanda).

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 as funcionalidades entregues nesta etapa.

O Tempo de Correção e Manutenção ocorrerá no dia indicado no classroom. Nesse dia, o aluno terá a possibilidade, dentro do tempo disponibilizado, de corrigir erros encontrados no trabalho. Funcionalidades novas já especificadas anteriormente não serão contabilizadas nesta etapa, portanto não adianta usar este tempo para implementar o que já havia sido pedido e não foi entregue. Para fazer a correção, o aluno receberá todos os casos de teste escondidos do trabalho. No final deste tempo de correção, o aluno deverá reenviar o programa corrigido 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. 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. Neste mesmo dia, será testada a manutenibilidade do código, sendo exigida a implementação de novas funcionalidades para incorporação de pequenas mudanças no comportamento do jogo. Isso faz parte da pontuação do trabalho (ver tabela

de pontuação). Quanto mais modular estiver o código, mais fácil será de implementar as mudanças. As modificações necessárias para as novas funcionalidades não precisarão da tag “//CORRECAO:”.

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 quando necessário. Portanto, ela será feita sob demanda para avaliar o conhecimento do aluno a respeito do seu próprio programa.

Considerações Finais

Não utilizar caracteres especiais (como acentos) em lugar nenhum do trabalho.