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

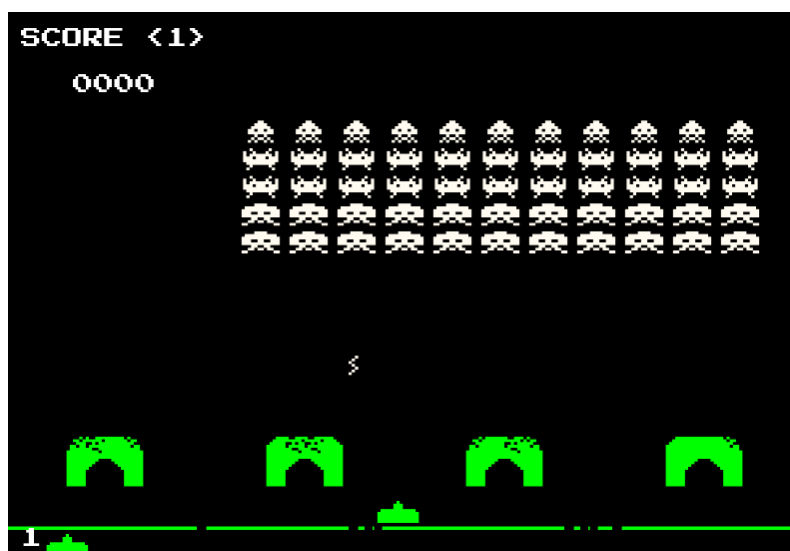
## Especificação do Trabalho

### *Space Invaders*

#### 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 jogo *Space Invaders*<sup>1</sup>, seguindo-se as especificações descritas neste documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.

*Space Invaders* é um jogo *single-player* no qual o jogador controla uma nave espacial que se move horizontalmente sobre um mapa bidimensional. O objetivo do jogo é eliminar toda a frota da invasão alienígena antes que cheguem até a nave do jogador.




#### Descrição do Trabalho

Neste trabalho, você deverá implementar um jogo do gênero *Space Invaders*. 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, a posição inicial da nave e os inimigos das 3 fileiras disponíveis. 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, momento em que será apresentado o estado final e alguns arquivos com informações sobre a partida serão criados.

#### Funcionamento Detalhado do Programa

<sup>1</sup> [https://pt.wikipedia.org/wiki/Space\\_Invaders](https://pt.wikipedia.org/wiki/Space_Invaders)

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

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 animar os inimigos). 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 diretório com os arquivos de configuracao.”), finalizando sua execução. Dentro desse diretório, espera-se encontrar dois arquivos, um com a definição do mapa do jogo (*mapa.txt*) e outro com o desenho do inimigo (*inimigo.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 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 fornecido.

O arquivo *mapa.txt* definirá o tamanho do mapa, bem como as posições do jogador e dos inimigos nas três fileiras disponíveis. O mapa será representado como uma matriz com N linhas e M colunas, com um tamanho máximo de 40 linhas e 100 colunas.

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


*mapa.txt*

```
25 11
(13 10)
(10 2) (16 2)
(13 6)
```

No exemplo acima, a primeira linha especifica um mapa com 25 colunas e 11 linhas. A segunda linha define a posição inicial central do jogador, que está na coluna 13 e linha 10. As três linhas seguintes detalham as fileiras de inimigos, onde cada inimigo é descrito pelo par de coordenadas (*coluna linha*), seguindo o mesmo formato usado para o jogador. Observa-se que há apenas duas linhas com posições de inimigos, enquanto a terceira está vazia, indicando que não é obrigatório haver inimigos em todas as fileiras. Cada linha, apresentará os inimigos pertencentes àquela fileira e não haverá sobreposição entre os inimigos. As coordenadas dos arquivos consideram a posição (1 1) como o canto superior esquerdo do mapa.

O arquivo *inimigo.txt* começa com uma flag na primeira linha que indica se o jogo deve ser executado com ou sem o ponto bônus (0 para execução normal e 1 para o ponto bônus). Nas



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

jogador, devem usar o caractere "-" em vez de barras verticais, indicando a posição onde o jogador perderá se os inimigos a ultrapassarem.

Por fim, note que as posições fornecidas nos arquivos se referem ao caractere central, tanto para o jogador quanto para os inimigos.

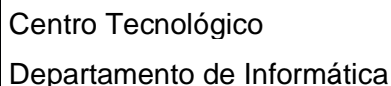
**Realizar jogo:** Uma vez que o jogo esteja preparado, as jogadas poderão começar a ser processadas. O jogo terminará quando o jogador eliminar todos os inimigos ou quando qualquer inimigo cruzar a linha limite (a linha imediatamente acima do jogador).

As jogadas deverão ser lidas da entrada padrão, permitindo o redirecionamento a partir de um arquivo (entrada.txt). Cada iteração de uma partida deve seguir, necessariamente na mesma ordem, as seguintes etapas:

- Verifique se todos os inimigos foram eliminados (vitória) ou se algum inimigo cruzou a linha limite (derrota);
- Caso exista um tiro ativo, verifique se ele colidiu com algum inimigo. Em caso de acerto, elimine o inimigo atingido, remova o tiro e adicione os pontos à partida. Os pontos são calculados multiplicando-se a coluna do inimigo pela sua linha no momento de sua morte, considerando as linhas dos inimigos de baixo para cima;
- Mova todos os inimigos vivos uma posição para o lado. Se algum inimigo ultrapassar a lateral do mapa com a movimentação, em vez de transladá-los horizontalmente, todos os inimigos devem descer uma posição e começar a se mover para o lado oposto na próxima iteração. No início do jogo, os inimigos movem-se para a direita;
- Caso exista um tiro ativo, mova-o uma posição para cima. Se o tiro atingir a borda superior, ele deve ser desabilitado;
- Incrementar o contador de iterações da partida (começando na iteração 0 logo após a etapa de inicialização).
- Leia a jogada fornecida pelo usuário no formato "@", onde "@" é um caractere que define o movimento, podendo ser:
  - "a" para mover para a esquerda.
  - "d" para mover para a direita.
  - " " (espaço) para atirar.
  - "s" para passar a vez.

Se o movimento lateral fizer o jogador ultrapassar uma borda, ele não se moverá e terá o mesmo efeito que "passar a vez" (tecla "s").

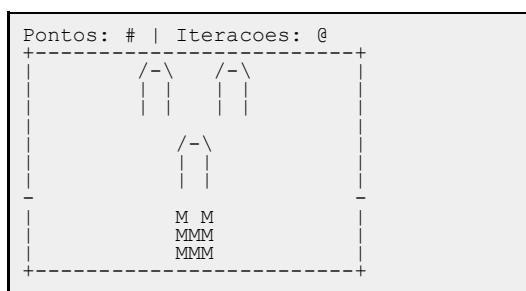
Atirar faz surgir um tiro na linha imediatamente acima do jogador, na sua posição central. Só é permitido atirar se não houver outro tiro ativo. Caso a entrada seja o



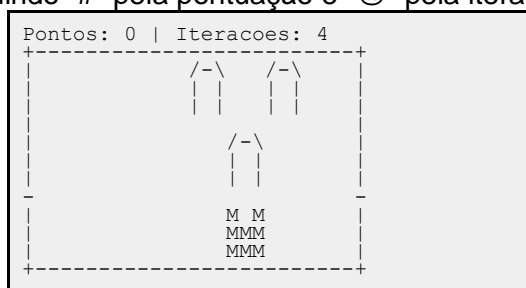
Código: INF15927

## Trabalho Prático

- Após cada movimento, o programa deverá exibir na saída padrão (ou seja, na tela) o estado atual do jogo (incluindo o mapa com o jogador e os inimigos) além da pontuação acumulada até o momento, no seguinte formato:



Substituindo "#" pela pontuação e "@" pela iteração do momento. Exemplo:




Um detalhe importante sobre a impressão do estado atual do jogo que deve ser considerado é a ordem em que os elementos do jogo são desenhados devido a problemas de sobreposição, portanto, é necessário atribuir prioridade da mais alta para a mais baixa na seguinte ordem: tiro ativo, inimigos e, por fim, o jogador.

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

d  
d  
d  
d

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

d  
a  
d  
d  
a

**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: o inimigo colidir com a lateral direita ou esquerda, jogador atingir um inimigo com um tiro ou o jogador colidir em alguma lateral. Cada linha deve descrever a alteração causada, sendo que cada uma possui um padrão específico de saída.

Nos exemplos abaixo, "@" indica a iteração que ocorreu o evento, "#" o índice do inimigo na fileira (de acordo com a ordem do arquivo `mapa.txt`, ou seja, o primeiro inimigo lido de uma fileira é o de índice 1, o segundo é o de índice 2 e assim por diante), "\$" a fileira (podendo ser 1, 2 ou 3, representando respectivamente a primeira, segunda ou terceira fileiras lidas do arquivo `mapa.txt`), "%" indica lateral em que ocorreu a colisão (direita ou esquerda) e "X" e "Y" devem ser trocados pela posição do inimigo atingido no momento de sua morte. Portanto, os registros devem seguir os seguintes padrões:

- Colisão do inimigo com alguma lateral:

```
[@] Inimigo de indice # da fileira $ colidiu na lateral %.
```

- Jogador atingir algum inimigo:

```
[@] Inimigo de indice # da fileira $ foi atingido na posicao (X Y).
```

- Jogador colidir com alguma lateral:


```
[@] Jogador colidiu na lateral %.
```

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

`resumo.txt`

```
[5] Inimigo de indice 1 da fileira 3 foi atingido na posicao (25 9).
[9] Inimigo de indice 2 da fileira 1 colidiu na lateral direita.
[9] Inimigo de indice 2 da fileira 2 colidiu na lateral direita.
[16] Inimigo de indice 1 da fileira 1 foi atingido na posicao (22 4).
[27] Inimigo de indice 1 da fileira 2 colidiu na lateral esquerda.
[45] Inimigo de indice 2 da fileira 1 colidiu na lateral direita.
[45] Inimigo de indice 2 da fileira 2 colidiu na lateral direita.
[52] Inimigo de indice 2 da fileira 3 foi atingido na posicao (32 10).
```

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.

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

**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 das mortes dos inimigos ao final do jogo. Os inimigos serão ordenados pela linha em que foram mortos (linha em que o tiro os acertou), começando de baixo para cima. Em caso de empate na linha de morte, será considerado como critério de desempate quem morreu primeiro, utilizando a iteração da morte.

O arquivo de ranking deve começar com a primeira linha contendo o texto "indice,fileira,linha,iteracao", seguido pelos dados dos inimigos em cada linha subsequente. O índice refere-se à ordem de leitura do inimigo do arquivo de configuração da partida (`mapa.txt`), conforme definido acima em gerar resumo. A linha e iteração indicam respectivamente a posição vertical (Y) do inimigo no mapa e o momento da sua morte.

Exemplo do arquivo de ranking:

`ranking.txt` do caso de exemplo 1

```
indice,fileira,linha,iteracao
1,2,9,14
1,1,10,9
2,1,10,37
```


**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 a esquerda (A) ou para a direita (D), o número de tiros efetivos (tiros que não foram anulados por já existir um tiro na iteração), o número de tiros que não acertaram nenhum inimigo e o número de vezes que os inimigos desceram uma posição.

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

`estatisticas.txt` do caso de exemplo 1

```
Numero total de movimentos (A ou D): 34;
Numero de tiros efetivos: 3;
Numero de tiros que nao acertaram: 0;
Numero de descidas dos inimigos: 2;
```

**Gerar Mapa de Calor:** Ao final do jogo, o programa deve criar um arquivo chamado (`heatmap.txt`) na pasta de saída do caso de teste atual. Este arquivo contém um mapa de calor que registra o número vezes que o jogador ou o tiro passaram por cada célula. Ou seja, cada iteração deve incrementar 1 nas células ocupadas pelo jogador ou pelo tiro no mapa. A

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

iteração 0, imediatamente após a inicialização do jogo, deve ser a primeira contabilizada na contagem do mapa de calor.

O valor máximo que pode ser registrado no mapa é 999; caso exceda esse valor, ele permanecerá forçadamente como 999. É fundamental destacar que as nove posições adjacentes ao jogador devem ser atualizadas no mapa de calor, e não apenas a posição central. Além disso, para melhor visualização, cada número registrado deve ser formatado com três caracteres de tamanho (utilizar a notação "%3d").

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 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 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 1 0 0 0 0 0 0 0 0
4 6 10 8 8 6 4 2 0 0 0 0
4 6 10 8 8 6 4 2 0 0 0 0
4 6 10 8 8 6 4 2 0 0 0 0

```

**Animação (Bônus):** Haverá uma pontuação extra para a implementação da funcionalidade de animação dos inimigos. O arquivo `inimigo.txt` especifica uma flag na primeira linha (0 para desativar a animação e 1 para ativá-la). Caso o aluno opte por implementar essa funcionalidade, quando a flag for 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, recomeçando o ciclo quando os três padrões de inimigos forem desenhados. Note que os desenhos serão os mesmos para todos os inimigos.

Um exemplo parcial pode ser visto nas quatro iterações abaixo:

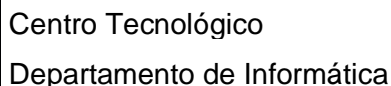
Saída com animação de exemplo para o caso 1:

```

Pontos: 0 | Iteracoes: 0
+-----+
|          |
|  /-\  /-\ |
|  |  |  |  |
|          |
|  /-\  |
|  |  |  |
|  M M  |
|  M M  |
|  M M  |
+-----+
Pontos: 0 | Iteracoes: 1
+-----+
|  <->  <-> |
|  |  |  |  |
|  <->  |
|  |  |  |
|          |
+-----+

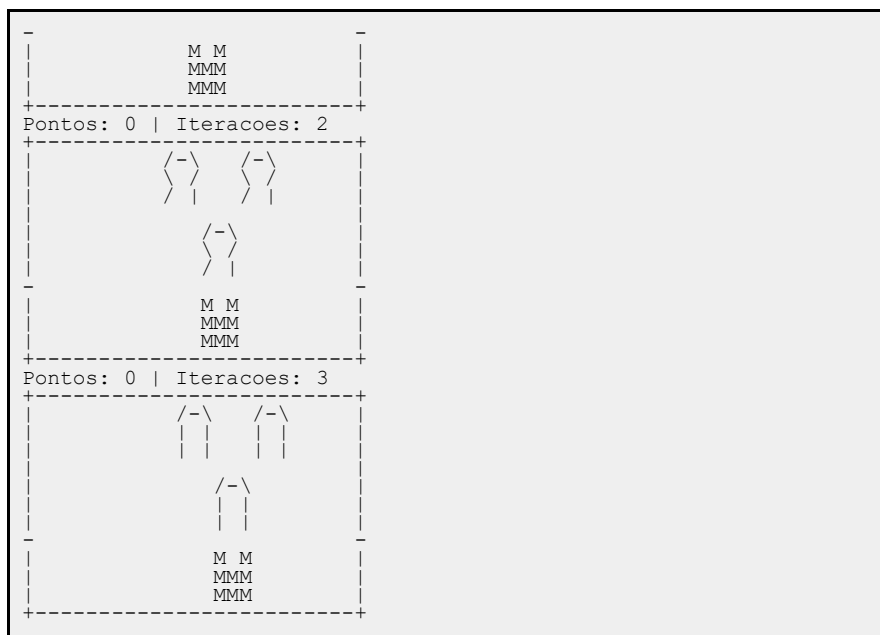
```





Código: INF15927


## Trabalho Prático



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.

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

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

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.

## 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 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.

## 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, 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. 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

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

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

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


Legibilidade e Modularização	Falta de: <ul style="list-style-type: none"> <li>• Uso de comentários;</li> <li>• Identação do código;</li> <li>• Uso de funções;</li> <li>• Uso de tipos de dados definidos pelo usuário</li> </ul>	-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  $NPE \leq NTC \leq 10$ . 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.

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

## Considerações Finais

Não utilizar caracteres especiais (como acentos) em lugar nenhum do 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 nos canais de comunicação da disciplina. 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.

### Errata 1

Os arquivos `ranking.txt` dos casos de teste apresentavam um erro no cálculo das linhas dos inimigos. Portanto, os arquivos foram atualizados para refletir a especificação do trabalho.

### Errata 2

- O máximo de linhas e colunas estavam trocados;
- Os eventos de colisão dos inimigos dos arquivos de `resumo.txt` foram atualizados para respeitar a ordem de eventos fornecida em "Realizar Jogo", pois, estavam sendo mostradas antes do evento de morte do inimigo;
- Os Inimigos mortos estavam sendo considerados nos casos 8, 14 e 21, agora os testes os ignoram;
- Os arquivos de entrada do caso 9, 18, `bonus_3`, `bonus_4`, `bonus_5` tinham caracteres em sequência na mesma linha. Os caracteres foram movidos de forma que cada linha tenha apenas um caractere seguidos de uma quebra de linha;
- O arquivo `mapa.txt` do caso 18 tinha um espaço extra no fim da linha 3, sendo removido;