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

Especificação do Trabalho
Jogo da Vida

1 Introdução

Esse trabalho tem como objetivo avaliar, de forma prática, os conceitos ensinados no curso de Programação II. O trabalho pode ser feito de forma incremental, isto é, à medida que o conteúdo for ensinado. A meta desse trabalho é implementar um simulador de Jogo da Vida (Conway's Game of Life) seguindo as especificações descritas nesse documento e utilizando os conceitos ensinados em sala para manter boas práticas de programação.

O Jogo da Vida é um simulador onde existe um tabuleiro quadriculado de células (a princípio, tal tabuleiro se estende infinitamente em todas as direções). Cada célula pode estar viva ou morta, e ao longo das iterações da simulação (chamadas "gerações") certas regras são aplicadas a essas células para que elas vivam ou morram. A morte e ressurreição das células causam padrões visuais que se espalham pelo tabuleiro. Um exemplo de implementação do Jogo da Vida pode ser visto em http://pmav.eu/stuff/javascript-game-of-life-v3.1.1/.

2 Descrição do Trabalho

Nesse trabalho, você deverá implementar um programa para simular o Jogo da Vida. O programa deverá rodar no terminal, assim como todos os outros exercícios feitos em sala. Considere que as simulações podem ser realizadas com configurações variadas (i.e., diferentes tamanhos de tabuleiro, diferentes estados iniciais, etc.), e que esses dados (chamados de configuração do jogo) serão passados para o programa antes do jogo iniciar através de um arquivo de configuração.

Perceba que o Jogo da Vida que você irá implementar será diferente do Jogo da Vida original em alguns aspectos. O tabuleiro não será infinito, mas terá uma altura e largura determinada. Ao longo das gerações algumas células terão seus estados alterados, conforme especificado no arquivo de configuração. E você coletará alguns dados durante a simulação que, a princípio, não são necessários para a simulação em si, mas são necessários para ganhar os pontos do trabalho. Perceba também que o jogo é completamente determinístico, portanto não há elementos aleatórios a serem considerados. Ao rodar uma mesma simulação várias vezes, os resultados sempre serão os mesmos.

De maneira geral, o programa irá iniciar pela linha de comando (terminal) e irá ler um arquivo contendo as configurações do jogo, como por exemplo o tamanho do tabuleiro, quais células estão vivas inicialmente, número de gerações a processar, entre outros dados. A localização desse arquivo será passada como parâmetro (pela linha de comando) para o programa. O primeiro passo do jogo é ler o arquivo de configuração e criar o tabuleiro de células com o tamanho especificado. Em seguida, a simulação iniciará e, assim, o estado inicial (definido também nas configurações) será criado no tabuleiro, contando como a primeira geração. A partir disso, o estado atual do tabuleiro será alterado, para criar a próxima geração, de acordo com as seguintes regras:

- Qualquer célula viva com menos de dois vizinhos vivos deverá morrer;
- Qualquer célula viva com dois ou três vizinhos vivos deverá continuar viva;
- Qualquer célula viva com mais de três vizinhos vivos deverá morrer;

- Qualquer célula morta com exatamente três vizinhos vivos deverá reviver.

Você deve considerar que cada célula tem oito vizinhos (laterais e diagonais), exceto aquelas células que estão nos cantos do tabuleiro, estas células têm menos vizinhos. Cada nova geração será criada pela aplicação das quatro regras no estado da geração anterior. A simulação deverá durar pelo número de gerações determinado no arquivo de configuração. Ao final do jogo, você deverá criar alguns arquivos contendo estatísticas da simulação.

2.1 Funcionamento Detalhado do Programa

O programa será chamado de *trabalhoprog2* (após a compilação), portanto o projeto no Netbeans também deverá ser chamado de *trabalhoprog2*. Sua execução será feita através da linha de comando (i.e. pelo cmd, console, ou terminal) e permitirá a passagem de parâmetros, descritos mais adiante.

Ao iniciar, o programa deverá executar uma série de operações na sequência, e informar ao usuário o estado atual da simulação. As operações a serem realizadas pelo programa são descritas a seguir (obedecendo a ordem): criar jogo, realizar simulação, gerar arquivo de estatísticas, gerar arquivo com a contagem de "Still Lifes", além de uma função bônus opcional. A descrição dos parâmetros de inicialização e das operações a serem realizadas pelo programa são apresentadas em detalhes a seguir.

Parâmetros de inicialização (passados na chamada do programa pela linha de comando): Para garantir o correto funcionamento do programa, o usuário deverá informar, ao executar o programa pela linha de comando, o caminho do diretório de teste que contém o arquivo de configuração (ex. de execução: "*./trabalhoprog2 /maquinadorprofessor/test_1*"). Nesse mesmo diretório, serão gerados os arquivos de saída, porém, estes deverão ir para uma pasta específica, chamada *saída*, dentro desse diretório informado (ex. da pasta dos arquivos de saída "*./trabalhoprog2 /maquinadorprofessor/test_1/saida*"). Considere um caminho com tamanho máximo de 1000 caracteres. O programa deverá verificar a presença do parâmetro. Caso o usuário tenha esquecido de informar o nome do diretório, o programa deverá exibir (ex. "ERRO: O diretório de arquivos de configuração não foi informado!"), e finalizar sua execução.

Criar jogo: Nessa operação, o programa deverá ler informações do arquivo de configuração do jogo para a memória e preparar o ambiente de jogo. O arquivo de configuração terá sempre o nome *config.txt*, e estará sempre no diretório de teste recebido pela linha de comando. Caso o programa não consiga ler o arquivo, ele deverá ser finalizando e imprimir uma mensagem informando que não conseguiu ler o arquivo (OBS: a mensagem deve conter o caminho e nome do arquivo que ele tentou ler).

- Organização do arquivo de configuração do jogo (*config.txt*): A primeira linha do arquivo estará no formato shape=<W>,<H>, onde <W> é a largura do tabuleiro, e <H> é a altura do mesmo, ambos são número inteiros positivos (até no máximo 2500). A segunda linha terá o formato max_gen=<MAX_GEN>, onde <MAX_GEN> é um número inteiro positivo (máximo 1000) descrevendo o número de gerações da simulação. A terceira linha codifica uma lista de células que estarão vivas no estado inicial (todas as outras células devem iniciar mortas), essa linha terá o seguinte formato start: <N> <X1>,<Y1> <X2>,<Y2>..., em que <N> é o número de células vivas e cada par <X>,<Y> representa as coordenadas de uma célula que deve estar viva no estado inicial (N <= 100). Alguns arquivos de configuração ainda terão mais algumas linhas comandando que algumas

células tenham seu estado trocado antes de uma geração determinada. As linhas comandando as trocas de estado (respeitando um máximo de 200 linhas, ou seja, ocorrerão trocas em no máximo 200 gerações) terão um formato semelhante ao do estado inicial, e este é gen <I>: <N> <X1>,<Y1> <X2>,<Y2>..., em que <I> é a geração a ser processada, <N> é o número de células a serem trocadas (N <= 100) e cada par <X>,<Y> representa as coordenadas de uma célula. Imediatamente antes de processar a geração <I>, o estado das células listadas devem ser trocados. Um exemplo de arquivo de configuração completo está a seguir:

```
shape=30,10
max_gen=100
start: 5 1,0 2,1 0,2 1,2 2,2
gen 25: 5 1,0 2,1 0,2 1,2 2,2
gen 50: 5 1,0 2,1 0,2 1,2 2,2
gen 75: 5 1,0 2,1 0,2 1,2 2,2
```

Caso a leitura do arquivo de configuração seja bem-sucedida, em seguida, o programa deverá iniciar a simulação considerando um tabuleiro com as dimensões especificadas pelo shape do arquivo de configurações. As únicas células vivas do tabuleiro serão aquelas especificadas pelo start do arquivo de configuração. Esse estado inicial deve ser impresso como a primeira geração, ou seja, geração 0. Leia mais a frente como o estado deve ser impresso.

Realizar simulação: Em seguida, o programa deverá aplicar as quatro regras descritas acima para gerar a segunda geração. Esse processo deverá se repetir até que <MAX_GEN> gerações tenham sido geradas.

- Organização da informação do estado a ser impresso na tela (representado nos exemplos de teste pelo arquivo *output.txt*, que foi gerado redirecionando-se a saída padrão da tela para o referido arquivo): Ao imprimir o estado de um geração, você deverá imprimir três informações: número da geração (começando por 0, que é o estado inicial); número de células vivas; e número de células mortas. Após essas informações você deve imprimir o estado atual do tabuleiro. Use o caractere '-' (hífen) para representar as células mortas, e o caractere 'O' (letra "o" maiúscula) para representar as células vivas. Exemplo de impressões feitas pelo programa:

```
Geracao: 0
N. celulas vivas: 5
N. celulas mortas: 295
-O----------------------------
--O----------------------------
OOO----------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
Geracao: 1
N. celulas vivas: 5
N. celulas mortas: 295
------------------------------
O-O----------------------------
-OO----------------------------
-O----------------------------
------------------------------
------------------------------
------------------------------
------------------------------
```

```
------------------------------
------------------------------
Geracao: 2
N. celulas vivas: 5
N. celulas mortas: 295
------------------------------
--O---------------------------
O-O---------------------------
-OO---------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
------------------------------
...
...
...
```

Ainda, quando o arquivo de configuração especificar trocas de estado de uma lista de células, você deve trocar os estados deles imediatamente **antes** de processar a geração especificada. Trocar o estado da célula significa que: se ela estiver morta, você deve marcá-la como viva; e se ela estiver viva você deve marcá-la como morta. Por exemplo, vamos supor que o arquivo de configuração especifica gen 5: 2 1,1 2,2, então, após imprimir e coletar os dados da geração número 4, o programa deverá inverter o estado das células 1,1 e 2,2, e somente então deverá aplicar as regras para criar a geração número 5 e imprimi-la. Lembrando que o estado inicial conta como geração e é a geração 0.

Gerar arquivo de estatísticas para análise: Ao final do jogo, o programa deverá escrever um arquivo (*stats.txt*) contendo algumas estatísticas básicas sobre a simulação. Para isso, será necessário que ao longo da simulação o programa colete alguns dados a cada geração, para que seja possível gerar essas estatísticas. Esse arquivo deverá conter: a média de células vivas ao longo de todas as gerações (usar duas casas decimais); qual geração teve a maior quantidade de células vivas e qual é essa quantidade; qual geração teve a maior quantidade de células mortas e qual é essa quantidade; e também, por fim, uma lista com todas as gerações e a quantidade de células vivas em cada uma delas. Essa lista deve estar ordenada pela quantidade de células vivas em ordem **decrescente**, e o desempate será feito pelo número da geração em ordem **crescente**. Durante a contagem de máximo de células vivas (e mortas), se houver mais uma geração com o número máximo, então a geração impressa deverá ser a menor geração entre essas. Veja abaixo um exemplo de como deve ser o arquivo:

```
A media de celulas vivas ao longo das geracoes foi 6.19.
A maior quantidade de celulas vivas foi de 11 na geracao 43.
A maior quantidade de celulas mortas foi de 300 na geracao 46.
Lista de geracoes ordenada pela quantidade de celulas vivas:
 - Geracao 43: 11 celulas
 - Geracao 93: 11 celulas
 - Geracao 25: 10 celulas
 - Geracao 26: 10 celulas
 - Geracao 27: 10 celulas
 - Geracao 28: 10 celulas
 - Geracao 75: 10 celulas
 - Geracao 76: 10 celulas
 - Geracao 77: 10 celulas
 - Geracao 29: 9 celulas
 - Geracao 31: 9 celulas
 - Geracao 32: 9 celulas
```

Gerar arquivo com a contagem de Still Lifes: Durante o jogo certos padrões aparecem ao longo das gerações. Esses padrões podem ser: Still Lifes (padrões estáveis que permanecem o mesmo ao longo das gerações); Oscillators (padrões cíclicos que repetem sua forma periodicamente ao longo das gerações); Spaceships (padrões que se movimentam pelo tabuleiro); entre outros. Exemplos de todos esses padrões podem ser vistos na página do Wikipédia:

- https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life#Examples_of_patterns

Os Still Lifes que vocês irão utilizar estão representados na tabela a seguir. Alguns desses Still Lifes poderiam originalmente se apresentar em diferentes orientações, isto é, espalhados horizontalmente ou verticalmente. Mas você não precisa se preocupar com diferentes orientações. Use os Still Lifes exatamente da forma como eles estão representados abaixo.

```
Block  Beehive  Loaf     Boat    Tub
----   ------   ------   -----   -----
-OO-   --OO--   --OO--   -OO--   --O--
-OO-   -O--O-   -O--O-   -O-O-   -O-O-
----   --OO--   --O-O-   --O--   --O--
       ------   ---O--   -----   -----
                ------
```

Ao final da simulação, você deverá criar um arquivo (*still_lifes.txt*), listando quantas vezes cada Still Life apareceu ao longo de todas as gerações. Essa lista deve estar ordenada pelo número de aparições em ordem decrescente, e, em caso de empate, pela ordenação alfabética dos nomes dos Still Lifes. É importante notar que, ao procurar pelos Still Lifes no tabuleiro, você deve procurar levando em consideração exatamente cada célula viva e também cada célula morta do Still Life, exatamente como exibido na tabela acima. Veja, a seguir, um exemplo do tipo de conteúdo que deverá existir no arquivo *still_lifes.txt*:

```
 - Block: 6
 - Beehive: 5
 - Boat: 4
 - Loaf: 4
 - Tub: 4
```

Função bônus (opcional): Ao final do jogo você deve criar um arquivo, chamado *ghost.txt*, contendo um tabuleiro que mostra todas as células que estiveram vivas em alguma geração. Ou seja, se em alguma geração uma certa célula esteve viva ao ser impressa, esta mesma célula deve estar representada como viva nesse arquivo. Atente para o fato de uma célula poder trocar de estado conforme definido no arquivo de configurações, portanto, ele só deverá ser considerada viva, se ela foi impressa como viva em alguma geração. Veja o exemplo:

```
-O--------------------------
OOO-------------------------
OOOO------------------------
-OOOO-----------------------
--OOOO----------------------
---OOOO---------------------
----OOOO--------------------
-----OOOO-------------------
------OOOO------------------
-------OOO-------------------
```

Alguns arquivos de entrada e seus respectivos arquivos de saída serão fornecido para o aluno como exemplo (os arquivos de saída esperada estarão na pasta *saidaesperada* dentro do diretório do respectivo teste). O aluno deverá utilizar tais arquivos para testes durante a implementação do trabalho. É 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 fornecidos, outros arquivos (específicos para a correção e não disponibilizados para os alunos) seguindo a formatação descrita nesse documento e utilizada nos arquivos exemplos. 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 (nota zero). Portanto, siga estritamente o formato estabelecido para as entradas e saídas.

2.2 Implementação

A implementação deverá seguir os conceitos de modularização e abstração apresentados 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 utilizar algum programa para fazer as comparações do resultado final do programa, isto é, os arquivos de saída gerados, poderão ser comparados com o arquivo de saída esperada (fornecido 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.

3 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. O aluno deverá necessariamente utilizar o Netbeans para desenvolvimento do trabalho!

Cada aluno deverá trabalhar independente dos outros não sendo permitido a cópia ou compartilhamento de código. O professor irá fazer verificação automática de plágio. Trabalhos identificados como iguais, em termos de programação, 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 (ex. laboratório).**

3.1 Entrega do Trabalho

O trabalho deverá ser entregue por email (para: todsantos@inf.ufes.br) até o dia **23/06/2018**. O assunto da mensagem deverá seguir o padrão: [TRABALHO_PROG2_2018_1] <NomeDoAluno>.

Exemplo do assunto da mensagem para o aluno Fulano da Silva:

- [TRABALHO_PROG2_2018_1] FulanoDaSilva

Todos os arquivos do programa (*.c e *.h), incluindo o diretório “nbproject” e o arquivo “Makefile” gerados pelo netbeans deverão ser compactados em um único arquivo <nome_do_aluno>.zip e enviado como anexo da mensagem. Lembrando que o anexo não deverá conter arquivos compilados ".o" ou executáveis ".exe" sob o risco de bloqueio de email contendo arquivos executáveis. O código será compilado posteriormente em uma outra máquina. A correção poderá ser feita de forma automática, portanto a entrega incorreta do trabalho, ou seja, fora do padrão, poderá acarretar na impossibilidade de correção do trabalho e consequentemente na

atribuição de nota zero. A pessoa corrigindo não terá a obrigação de adivinhar nome de arquivos, diretórios ou outros. Siga estritamente o padrão estabelecido!

Dica para teste de envio do trabalho: siga os passos anteriores; envie um email para você mesmo; descompacte o conteúdo para uma pasta; abra um terminal e mude o diretório corrente para a pasta de descompactação; execute o comando "make all" que deverá compilar todo o projeto e gerar um arquivo executável em algum lugar da pasta "./dist/Release/.../*trabalhoprog2.exe*"; coloque os arquivos de teste em um diretório qualquer (ex. "/minhapasta/test_1"); execute o programa gerado passando o caminho do diretório com os arquivos de teste, e veja se funciona corretamente. Por fim, abra o projeto com o Netbeans para ver se está tudo como esperado. **Se não funcionou para você, não vai funcionar para mim!**

Os dados necessários para executar o comando "make" são gerados automaticamente pelo Netbeans dentro da pasta do projeto.

3.2 Pontuação

Trabalhos entregues após o prazo não serão corrigidos (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 a simulação corretamente. Código com falta de legibilidade e modularização pode perder ponto 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
Realizar simulação	Carregar o estado inicial. Processar as gerações usando as regras. Imprimir os estados de cada geração na tela.	4
Criar estatísticas	Gerar corretamente o arquivo com as estatísticas.	2
Trocar estados	Obedecer as trocas de estado especificadas no arquivo de configuração.	1
Contagem de Still Lifes	Gerar arquivo ordenado com a quantidade de aparições dos Still Lifes.	3
Legibilidade e Modularização	Falta de: • Uso de comentários; • Identação do código; • Uso de funções;	-2

	• Uso de tipos de dados definidos pelo usuário e respectivos arquivos (.c e .h).	
Função bônus	Gerar corretamente o tabuleiro das células que viveram.	1

Com intuito de identificar alunos que tiveram o trabalho feito por terceiros ou tiveram ajuda demasiada, a nota obtida com a avaliação acima (NI) será ponderada com uma nota de entrevista (NE) sobre o trabalho. A nota final do trabalho será calculada com NI x NE, onde NE vai de zero a um. A entrevista avaliará o conhecimento do aluno a respeito do programa desenvolvido. 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 entregue. A necessidade da entrevista será avaliada no decorrer do semestre, portanto sejam honestos e façam o trabalho com seu próprio esforço.

4 Considerações Finais

Não utilizar acentos no trabalho.

5 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 as especificações do trabalho. Caso seja notada qualquer tipo de inconsistência nos arquivos de testes disponibilizados, comunique imediatamente ao professor para que ela seja corrigida.