

1 Sobre a entrega do trabalho

São requisitos para atribuição de notas a este trabalho:

- Uso de um arquivo Makefile para facilitar a compilação. Os professores rodarão “make” e deverão obter o arquivo executável funcional com a sua solução. Este executável deverá estar no subdiretório t1;
- Opções de compilação: deve incluir `-Wall` e `-std=c90`. Haverá desconto na nota se compilador mostrar algum “warning”;
- Os professores também irão rodar com o valgrind, cada erro também vai gerar desconto na nota;
- Arquivo de entrega:
 - deve estar no formato tar comprimido (.tar.gz);
 - O tar.gz deve ser criado considerando-se que existe um diretório com o nome do trabalho. Por exemplo, este trabalho é o t1;
 - Então seu tar.gz deve ser criado no diretório pai do subdiretório t1, o qual deve conter todos os arquivos que serão entregues (tar zcvf t1.tar.gz t1), de maneira que os professores, ao abrir o tar.gz com o comando “tar zxvf t1.tar.gz” obterão um diretório t1 com as suas respostas;
 - Os professores testarão seus programas em uma máquina do departamento de informática (por exemplo, cpu1), por isso, antes de entregar seu trabalho faça um teste em máquinas do dinf para garantir que tudo funcione bem.

2 O trabalho

Você vai baixar o t1.tar.gz anexo a este enunciado e vai abri-lo para poder fazer o trabalho, você vai precisar de todos os arquivos. Ele contém os seguintes arquivos sob um diretório de nome t1:

libconjunto.h: arquivo (read only) de *header* para o TAD CONJUNTO;

libfila.h: arquivo (read only) de *header* para o TAD FILA;

liblef.h: arquivo (read only) de *header* para o TAD LEF;

makefile: sugestão de um Makefile que você pode usar (ou adaptar, se quiser).

Os arquivos `.h` não podem ser alterados em nenhuma hipótese. Na correção, os professores usarão os arquivos originais.

- Use boas práticas de programação, como indentação, bons nomes para variáveis, comentários no código, bibliotecas, ... Um trabalho que não tenha sido implementado com boas práticas vale zero.
- Quaisquer dúvidas com relação a este enunciado devem ser solucionadas via sistema moodle sendo que os professores devem receber o questionamento. Na dúvida não tomem decisões sobre a especificação, perguntem!
- Pode ser durante as aulas também.

3 Bibliotecas necessárias

Você vai programar todos os módulo que serão necessárias para a implementação do problema descrito abaixo, esses módulos fazem parte de sua solução e serão testados e avaliados. Estes módulos são os seguintes:

- TAD conjunto de inteiros;
- TAD fila de inteiros;
- TAD a lista de eventos futuros (LEF).

Você deve implementar esses três módulos do seu programa considerando os arquivos de cabeçalho (*headers*) fornecidos: `libconjunto.h`, `libfila.h` e `liblef.h`. Esses arquivos de cabeçalho não devem ser modificados em hipótese alguma.

Lembre-se de testar cada um desses módulos.

4 O problema

Uma simulação é uma experimentação com um modelo que imita de forma aproximada a realidade e sua evolução ao longo do tempo. A simulação é usada em muitos contextos: na modelagem científica de sistemas naturais; como técnica de ensino, oferecendo para os alunos um ambiente onde é possível experimentar e errar sem as consequências do ambiente real; e também em um grande número de jogos eletrônicos recreativos.

A simulação de eventos discretos (SED) modela a operação de um sistema como uma sequência de eventos discretos no tempo. O modelo possui uma estrutura estática e uma estrutura dinâmica. A estrutura estática especifica os possíveis estados do modelo e é geralmente descrita como uma coleção de entidades e seus atributos. O estado do sistema é a coleção de dados que descreve o sistema em um determinado momento.

A estrutura dinâmica especifica como o estado muda ao longo do tempo e geralmente é definida em termos de eventos que ocorrem em um determinado momento e mudam o estado do sistema. Entre eventos consecutivos o estado do sistema não sofre mudança alguma e a simulação pode saltar diretamente do instante de ocorrência de um evento para o próximo. Uma atividade é o estado de um objeto durante um intervalo.

Um simulador deve manter um relógio lógico global e uma lista ordenada por tempo dos eventos que vão ocorrer no futuro (LEF - Lista de Eventos Futuros). O relógio costuma ser um número inteiro que representa a quantidade de tempo do sistema real. Dependendo do modelo da simulação, cada unidade do relógio virtual pode representar pequenas frações de tempo (por exemplo, em uma simulação de partículas) ou muito grande (por exemplo, simulando a evolução do clima em uma determinada era).

O ciclo básico de simulação retira o primeiro evento da lista de eventos, atualiza o estado do sistema, agenda os novos eventos na lista e continua o ciclo.

Nós vamos implementar um simulador de vida real simplificado. Os habitantes de nosso mundo virtual são heróis que passam seu tempo lutando contra bandidos em diversas missões.

O restante desse texto especifica as entidades e atributos do sistema; os eventos que alteram o estado do nosso mundo virtual; o esqueleto geral da simulação e as mensagens de saída esperadas.

5 Entidades e atributos

Essa seção apresenta as entidades e atributos do nosso mundo virtual.

Herói: Representa os Heróis com seus atributos

id: Inteiro que identifica de maneira única um herói;

paciência: Inteiro entre 0 e 100 que indica quanto paciente uma pessoa é. Em nosso modelo isso afeta as decisões de permanência em locais e filas;

idade: Inteiro entre 18 e 100 indicando a idade em anos de uma pessoa. Em nossa simulação a idade afeta o tempo de deslocamento entre lugares;

experiência: Inteiro maior ou igual à 0 que indica o número de missões em que o herói já participou;

conjunto de habilidades (CH): Habilidades que um herói possui

Local: Representam os locais frequentados pelos heróis para formar equipes; os locais têm uma localização em um plano cartesiano, podem estar cheios e possuem uma fila de espera. Os atributos dos locais são:

id: Inteiro que identifica o local;

lotação máxima: Número máximos de heróis do lugar;

heróis no lugar: Conjunto de identificadores dos heróis que estão atualmente no lugar, representam as equipes disponíveis para realizar missões;

fila: Fila onde os heróis esperam para poderem entrar caso o mesmo esteja lotado;

localização: Par de inteiros (x, y) indicando uma coordenada em um plano cartesiano. Vamos considerar que o mapa de nossa supercidade é representado por um plano cartesiano de tamanho tal que cada unidade representa 1 metro.

Mundo: Nosso mundo é definido por essas entidades e algumas informações gerais.

TempoAtual: Inteiro positivo indicando o tempo atual da simulação. Cada unidade representa 15 minutos de tempo real;

Herois: Vetor representando todos os heróis;

Locais: Vetor representando todos os locais;
Habilidades: Conjunto de inteiros que representam as habilidades existentes no mundo;
NHerois: Número total de heróis no mundo;
NLocais: Número total de locais no mundo;
NTamanhoMundo: Coordenadas máximas do plano cartesiano dos locais.

6 Eventos

Nosso relógio (**TEMPO_ATUAL**) inicia em 0 e cada unidade representa 15 minutos da vida real. Os eventos são registrados em uma Lista de Eventos Futuros (**LEF**) ordenados pelo seu tempo lógico. Cada vez que um evento é lido o **TEMPO_ATUAL** é atualizado para o tempo do evento.

Em nossa simulação temos 4 tipos de eventos diferentes: **CHEGADA**, **SAIDA**, **MISSAO** e **FIM**. Cada evento possui diferentes atributos que permitem os processamentos na simulação. Portanto, a **LEF** deve suportar diferentes tipos de eventos, a nossa implementação oferece dois inteiros que são utilizados para essas informações. O significado desses inteiros depende do evento.

As próximas subseções especificam os atributos de cada um destes eventos e qual seus comportamento esperado.

6.1 CHEGADA

Representa um herói (**ID_HEROI**) entrando em um local (**ID_LOCAL**). Ao chegar, um herói deve saber se o local está ou não lotado, decidir esperar na fila ou sair imediatamente. Caso o herói entre no local ele deve decidir seu tempo de permanência e escalonar sua saída criando um evento de **SAIDA** no futuro.

Parte do comportamento de nossa população tem componente aleatório. Nas especificações deste trabalho vamos utilizar a notação **ALEAT(x,y)** para representar números inteiros gerados aleatoriamente entre x e y ¹.

Abaixo mostramos uma especificação em alto nível de como a simulação deve tratar o evento de **CHEGADA**.

¹Em C você pode usar simplesmente a função **rand()**. Em uma única vez do seu código você deve chamar a função **srand(mente)**. Se você usar a mesma semente durante toda sua simulação e em toda execução o comportamento será sempre o mesmo. O que pode ser útil para efeitos de depuração.

```

Se o ID_LOCAL está lotado o ID_HEROI,
decidir entre ficar na fila ou ir embora:

    Se (HEROIS[ID_HEROI].PACIENCIA/4 - tamanho_fila(ID_LOCAL) > 0
        Adiciona ID_HEROI em ID_LOCAL.fila
    Senão
        Cria um evento de SAIDA de ID_HEROI com TEMPO_ATUAL+0 e insere na LEF

Se ID_LOCAL não está lotado, calcular o Tempo de Permanência no Local (TPL):
    TPL = MAX(1,HEROI[ID_HEROI].paciencia/10+ALEAT(-2,6))

*obs: o MAX é para garantir que o tempo mínimo seja 1
      TPL já é dado em unidades de tempo de simulação (1 == 15 min)

Criar SAIDA para ID_HEROI e inserir na LEF em TEMPO_ATUAL + TPL

```

6.2 SAIDA

Representa a saída de um herói (ID_HEROI) de um local (ID_LOCAL). A saída de um herói pode gerar uma vaga no local tirando uma pessoa que estava na fila. Durante a saída é determinado um novo local de destino para a pessoa (ID_PESSOA) como o tempo necessário para sua chegada.

Abaixo uma especificação em alto nível de como a simulação deve tratar o evento de SAÍDA.

```

Se ID_LOCAL.fila não estiver vazia (existe um herói ID_HEROI_FILA na fila)
- criar evento de CHEGADA para ID_HEROI_FILA em ID_LOCAL
- inserir esse evento em TEMPO_ATUAL+0 e INICIO da LEF
obs: - inserir no INICIO evita que uma CHEGADA com tempo=TEMPO_ATUAL
      na LEF possa "fure" a fila.
      - nem toda saída gera lugar na fila, um herói impaciente sai do
        local sem nunca ter entrado (não quis ficar na fila).

Criar evento de CHEGADA para ID_HERÓI
- ID_LOCAL_DEST deve ser escolhido aleatoriamente entre os locais
  obs: podendo inclusive decidir entrar novamente no mesmo local)
- inserir o evento na LEF considerando em TEMPO_ATUAL+TDL/15
- TDL(Tempo de Deslocamento entre Locais):
  TDL = Distância (D) / Velocidade (V)

onde:
  V = 100-MAX(0,pessoas[ID_HEROI].idade-40);
  D = Distância cartesiana entre ID_LOCAL e ID_LOCAL_DESTINO

```

6.3 MISSAO:

Representa uma missão ID_MISSAO para ser cumprida pelos nossos heróis. Um missão é definida por um subconjunto das habilidades existentes no mundo. O conjunto de todos os heróis de um lugar formam as equipes que podem ser escolhidas para realizar uma missão. Uma equipe apta para

realizar a missão é aquela em que a união das habilidades de todos os heróis contém as habilidades da missão. Em caso de mais de uma equipe apta a equipe com menor número de heróis será escolhida.

Abaixo uma especificação em alto nível de como a simulação deve tratar o evento de SAÍDA.

```
MISSAO = subconjunto de CH (Conjunto de Habilidades) com tamanho ALEAT(3,6)
ID_LOCAL_EQUIPE_ESCOLHIDA = escolhe menor equipe apta para realizar a MISSAO
Se existe LOCAL_EQUIPE_ESCOLHIDA
    Atualiza experiencia de todos os heróis da equipe
Senão
    Recriar evento missão ID_MISSAO entre o tempo atual e o FIM_DO_MUNDO
```

6.4 FIM

Esse evento determina o final da simulação.

7 Esqueleto da simulação

No início da simulação devemos criar nosso mundo virtual com todas suas entidades e um conjunto inicial de eventos na LEF. Você tem vários parâmetros gerais para a simulação:

```
0 tempo inicial é 0
0 tamanho do nosso mundo (N_TAMANHO_MUNDO) é 20000
0 tamanho do conjunto de habilidades (N_HABILIDADES) é 10
0 número de heróis (N_HEROIS) é N_HABILIDADES*5
0 número de locais (N_LOCAIS) é N_HEROIS/6
0 número de missões (N_MISSÕES) é FIM_DO_MUNDO/100
0 final da simulação (FIM_DO_MUNDO) é no instante 34944
```

Para teste você pode mudar esses parâmetros, eles foram escolhidos para garantir que todas as condições da simulação ocorram. Lugares cheios, missões impossíveis, filas, etc...

Criar uma estrutura MUNDO que contém pelo menos:

```
tempo da simulação
tamanho do mundo
número de heróis
número de locais
vetor de N_HEROIS
vetor de N_LOCAIS
CH_HABILIDADES conjunto de N_HABILIDADES
    habilidades (de 0 até N_HABILIDADE-1)
```

No vetor de N_HEROIS, cada HEROI é inicializado com:

```
id = número sequencial 0-N_HEROI-1
experiencia = 0
paciencia    = ALEAT(0,100)
idade = ALEAT (18,100)
habilidades = subconjunto de CJ_HABILIDADES
              com tamanho ALEAT(2,5)
```

No vetor de N_LOCAIS, cada LOCAL é inicializado com:

```
id = número sequencial 0-N_LOCAIS-1
lotacao_max = ALEAT (5,30)
localizacao =
    ALEAT(0,N_TAMANHO_MUNDO-1), ALEAT(0,N_TAMANHO_MUNDO-1)
publico = conjunto vazio
```

Criar para cada HEROI da população:

```
EVENTO de CHEGADA em ID_LUGAR_DEST=ALEAT(0,N_LOCAIS-1)
inserir na LEF com tempo: ALEAT(0,96*7)
```

Criar N_MISSOES eventos de MISSOES com tempos aleatórios entre 0 e FIM_DO_MUNDO.

Criar e inserir o evento de FIM no tempo FIM_DO_MUNDO.

Fiat Lux! Nosso mundo foi criado e podemos começar a processar os eventos. Os eventos vão atualizar o estado do nosso mundo e também criar novos eventos. Esse ciclo termina com o evento de final (em determinadas culturas chamado de apocalipse). O laço principal é relativamente simples, e fica elegante usando o comando `switch/case`.

```
ENQUANTO retirar EVENTO da LEF
  Atualizar TEMPO_ATUAL com EVENTO.tempo
  switch(EVENTO->tipo)
  {
    case CHEGADA:
      tratar evento
      break;
    case SAIDA:
      tratar evento
      break;
    case MISSAO:
      tratar evento
      break
    case FIM
      finalizar simulacao
  }
```


8 Mensagens da simulação

Nenhuma divindade virtual se sente realizada sem conhecimento do que as pessoas de seu mundo estão fazendo. Por isso nossa simulação deve imprimir algumas informações durante o processamento dos eventos. Essas mensagens também podem ser úteis no processo de depuração do código.

8.1 Mensagens para o evento de CHEGADA

Exemplos das mensagens no evento de CHEGADA quando o local não está lotado:

```
34936:CHEGA HEROI 29 Local 1 ( 1/13), ENTRA
34936:CHEGA HEROI 28 Local 1 ( 2/13), ENTRA
34937:CHEGA HEROI 38 Local 7 ( 5/11), ENTRA
```

A primeira mensagem nos diz que no instante 34936 ocorreu um evento de chegada do herói 29 no local 1; no local 1 estavam 1 herói; e sua lotação máxima de 13. Portanto o herói 29 "ENTRA" no local.

Foram usados para o print as seguintes formatações:

```
%6d para o tempo
%d para o ID_LOCAL
%2d para os demais inteiros
```

Exemplos de mensagem no evento de CHEGADA quando o local está lotado:

```
1467:CHEGA HEROI 38 Local 5 ( 6/ 6), FILA 2
1475:CHEGA HEROI 46 Local 5 ( 6/ 6), FILA 1
1476:CHEGA HEROI 13 Local 5 ( 6/ 6), DESISTE
```

Nas mensagens acima vemos que, no instante 1467, o herói 38 chegou ao local 5, entretanto esse estava lotado, a pessoa decidiu entrar na fila do local 5 e foi colocada na posição 2; Na próxima mensagem no instante 1475 temos uma situação semelhante, o herói 46 também chega em um local lotado e entra na fila na posição 1. No instante 1476 vemos um caso em que o herói 13 não teve paciência para esperar na fila e por isso DESISTE de entrar no local.

8.2 Mensagens para o evento de SAÍDA

```
78:SAIDA HEROI 16 Local 5 ( 2/ 6)
80:SAIDA HEROI 16 Local 5 ( 3/ 6)
81:SAIDA HEROI 29 Local 2 ( 1/19)
```

As mensagens acima são análogas às de CHEGADA.

```
19871:SAIDA HEROI 20 Local 5 ( 6/ 6), REMOVE FILA HEROI 29
19871:CHEGA HEROI 29 Local 5 ( 5/ 6), ENTRA
```

No exemplo acima, vemos que a saída do HEROI 20 do local 5 abriu um espaço para a entrada do herói 29, vemos na mensagem seguinte sua entrada.

8.3 Saída para MISSÃO

```
692:MISSAO 157 HAB_REQ 0 3 6
692:MISSAO 157 HAB_EQL 0:0 1 3 4 5 6 7 9
692:MISSAO 157 HAB_EQL 1:2 5 6 7 8 9
692:MISSAO 157 HAB_EQL 2:0 2 3 4 6 8
692:MISSAO 157 HAB_EQL 3:0 1 2 3 4 7 9
692:MISSAO 157 HAB_EQL 4:0 2 3 4 6 7
692:MISSAO 157 HAB_EQL 5:0 1 2 3 4 6 7 8
692:MISSAO 157 HAB_EQL 6:0 5 6 7 8 9
692:MISSAO 157 HAB_EQL 7:0 1 2 4 5 6 8 9
692:MISSAO 157 HER_EQS 2:1 20
```

Acima que a missão MISSAO_ID 157 foi composta pelas habilidades 0, 3 e 6; as próximas linhas mostram a união das habilidades dos heróis que estavam em cada um dos locais (HAB_EQL). Finalmente vemos que a equipe escolhida para a missão (HER_EQS) foi a do lugar 2, composta pelos heróis 1 e 20.

Abaixo, um exemplo da missão MISSAO_ID 329 que foi impossível no instante 16016 mas que posteriormente foi reescalada.

```
16016:MISSAO 329 HAB_REQ 1 2 3 4
16016:MISSAO 329 HAB_EQL 0:0 1 2 4 5 7 9
16016:MISSAO 329 HAB_EQL 1:1 3 4 7 9
16016:MISSAO 329 HAB_EQL 2:0 1 2 4 5 8 9
16016:MISSAO 329 HAB_EQL 3:2 3 4 5 6 7 8
16016:MISSAO 329 HAB_EQL 4:1 3 6
16016:MISSAO 329 HAB_EQL 5:0 2 3 4 5 7 9
16016:MISSAO 329 HAB_EQL 6:2 4 6
16016:MISSAO 329 HAB_EQL 7:2 3 4 7
16016:MISSAO 329 IMPOSSIVEL
29405:MISSAO 329 HAB_REQ 0 2 3 5
29405:MISSAO 329 HAB_EQL 0:conjunto vazio
29405:MISSAO 329 HAB_EQL 1:5 9
29405:MISSAO 329 HAB_EQL 2:0 1 2 4 6 7
29405:MISSAO 329 HAB_EQL 3:0 1 2 3 4 5 6 7 9
29405:MISSAO 329 HAB_EQL 4:0 1 2 3 4 9
29405:MISSAO 329 HAB_EQL 5:0 1 2 4 5 6 8
29405:MISSAO 329 HAB_EQL 6:3 7 9
29405:MISSAO 329 HAB_EQL 7:0 1 2 3 5 6 7 9
29405:MISSAO 329 HER_EQS 7:11 12 18 27 34
```

8.4 Saída experiência

Ao final da simulação você deverá imprimir a experiência de todos os heróis:

```
34944:CHEGA HEROI 8 Local 4 ( 6/29), ENTRA
34944:FIM
HEROI 0 EXPERIENCIA 21
HEROI 1 EXPERIENCIA 59
HEROI 2 EXPERIENCIA 12
HEROI 3 EXPERIENCIA 25
HEROI 4 EXPERIENCIA 26
HEROI 5 EXPERIENCIA 25
HEROI 6 EXPERIENCIA 15
HEROI 7 EXPERIENCIA 31
<--cortado intencionalmente -->
HEROI 43 EXPERIENCIA 20
HEROI 44 EXPERIENCIA 42
HEROI 45 EXPERIENCIA 39
HEROI 46 EXPERIENCIA 60
HEROI 47 EXPERIENCIA 14
HEROI 48 EXPERIENCIA 18
HEROI 49 EXPERIENCIA 22
```

8.5 Dicas

A saída vai ser útil para você validar o comportamento da sua simulação. Uma das maneiras de fazer isso será usando comandos do bash para filtrar essa saída e permitir análises. Por exemplo, acompanhar entradas saída de um dado herói:

```
$> ./mundo |grep "HEROI 10" |more
577:CHEGA HEROI 10 Local 3 ( 3/ 8), ENTRA
589:SAIDA HEROI 10 Local 3 ( 2/ 8)
593:CHEGA HEROI 10 Local 4 ( 3/29), ENTRA
601:SAIDA HEROI 10 Local 4 ( 3/29)
607:CHEGA HEROI 10 Local 6 ( 3/ 7), ENTRA
614:SAIDA HEROI 10 Local 6 ( 5/ 7)
616:CHEGA HEROI 10 Local 3 ( 1/ 8), ENTRA
```

Ou verificar a lotação e fila de um local:

```
$> ./mundo |grep "Local 5" |more
680:CHEGA HEROI 39 Local 5 ( 4/ 6), ENTRA
682:CHEGA HEROI 31 Local 5 ( 5/ 6), ENTRA
682:CHEGA HEROI 10 Local 5 ( 6/ 6), FILA 1
683:SAIDA HEROI 28 Local 5 ( 6/ 6), REMOVE FILA HEROI 10
683:CHEGA HEROI 10 Local 5 ( 5/ 6), ENTRA
```

```
683:SAIDA HEROI 14 Local 5 ( 6/ 6)
\begin{verbatim}
```

Ou ver todos heróis escolhidos nas missões:

```
\begin{verbatim}
$> ./mundo |grep HER_EQS
161:MISSAO 261 HER_EQS 0:16
428:MISSAO 136 HER_EQS 1:4 6 12
534:MISSAO 267 HER_EQS 6:16 28 36
548:MISSAO 344 HER_EQS 7:30 34 37
692:MISSAO 157 HER_EQS 2:1 20
764:MISSAO 176 HER_EQS 5:4 10 21 29 36
831:MISSAO 75 HER_EQS 1:4 12 44
888:MISSAO 134 HER_EQS 2:1 12 30 32 44
```

Esse programa vai utilizar 3 módulos, libconjunto, liblef e libfila, você precisa ter certeza que esses módulos funcionam. Caso contrário você vai ficar pensando que o problema é em seu programa principal, quando na verdade é em um dos módulos anteriores.

Outra dica é que você de implementar aos poucos seu trabalho. Primeiro crie o mundo e os heróis com pelo menos partes dos seus atributo. Veja se consegue gerar os eventos de ENTRADA, mesmo que não verifique a lotação e coloque heróis na fila. Depois tente acrescentar o evento de SAIDA, e se der certo, daí sim pense em cuidar de gerenciar a lotação e fila dos lugares. Com isso funcionando preocupe-se então com as MISSOES. Para cada passo deste verifique as saídas, pense em teste que permitam verificar que o que você fez está correto. Ou seja, você deve começar implementando os eventos de maneira simplificada. Durante esse esforço pode-se perceber que decisões ruins foram tomadas, não hesite em reescrever código, quase sempre é menos trabalhoso do que conviver com o custo das decisões equivocadas passadas.

9 Como entregar o trabalho?

Entregue um único arquivo de nome t1.tar.gz no sistema moodle contendo pelo menos estes arquivos:

- libconjunto.h, libfila.h e liblef.h, mesmo que você não possa modificá-los;
- libconjunto.c, libfila.c e liblef.c;
- mundo.c, contendo a sua implementação da simulação (este arquivo contém o seu main());

- `makefile`. Os professores rodarão “make” para compilar e gerar os executáveis;
- Se você usou ou implementou mais arquivos que são importantes para o funcionamento do programa, entregue-os no mesmo pacote.

10 Considerações finais

Esse trabalho vai oferecer a oportunidade de amadurecimento na Linguagem C e em especial no uso de tipos de dados abstratos. É esperado que essa implementação além do programa principal tenha pelo menos 3 módulos independentes implementando os tipos abstratos de dados LEF, conjunto e fila. Esses módulos devem ser previamente testados e validados caso contrário poderá ser muito difícil encontrar erros na sua implementação.

A implementação é relativamente simples, mas exige bastante cuidado. Se o código começar a ficar muito complicado melhor repensar! A beleza deste tipo de simulação é justamente conseguir simular diversos cenários com facilidade. A implementação feita para validar a especificação tem menos de 500 linhas e o total com bibliotecas na ordem de mil linhas.

Foram determinados vários parâmetros na simulação. O objetivo foi dar dinamicidade para a simulação, os lugares serem distantes o suficiente para o tempo de deslocamento ser variável, o número de heróis ser compatível com a lotação dos locais de forma a eventualmente termos fila, entre outros tantos *apertos de parafusos*. Durante a fase de testes vocês podem experimentar outros valores para ver o comportamento de seu mundo!

Todos vocês já interagiram com muitas simulações eletrônicas! Essa especificação serve para os requisitos da disciplina, mas os mais audazes poderão ter curiosidade de implementar novos eventos em nosso mundo. Heróis podem perder poderes, ficarem velhos e se aposetarem, enfim literalmente um universo de possibilidades. Um programador pode criar os mais diversos universos e isso é um privilegio espetacular, nossas possibilidades são infinitas.

I etch a pattern of geometric shapes onto a stone. To the uninitiated, the shapes look mysterious and complex, but I know that when arranged correctly they will give the stone a special power, enabling it to respond to incantations in a language no human being has ever spoken. I will ask the stone questions in this language, and it will answer by showing me a vision: a world created by my spell, a world imagined within the pattern on the stone.

A few hundred years ago in my native New England, an accurate description of my occupation would have gotten me burned at the stake. Yet my work involves no witchcraft; I design and program computers. The stone is a wafer of silicon, and the incantations are software. The patterns etched on the chip and the programs that instruct the computer may look complicated and mysterious, but they are generated according to a few basic principles that are easily explained The Pattern on The Stone: The Simple Ideas That Make Computers Work, W. Daniel Hillis

11 Considerações finais

Mais detalhes você encontrará nos comentários dos próprios arquivos fonte fornecidos.

Bom trabalho!