MC202 — ESTRUTURAS DE DADOS

Laboratório 04 — Caixas de Colisão

Tarefa

Um problema comum na programação de jogos é a detecção de colisões entre objetos do jogo. Uma forma de detectar essas colisões é simplificar os objetos como um conjunto de “caixas de colisão”, como ilustrado na figura a seguir:

Na figura, a caixa de colisão do golpe sobrepôs a caixa de colisão do adversário, gerando assim o dano. No computador, um retângulo pode ser representado por dois pontos no plano cartesiano, o ponto superior esquerdo do retângulo e o ponto inferior direito.

Em um jogo, os objetos são dinâmicos e movimentam-se de um frame para o outro. Geralmente, as colisões são detectadas de um frame para o outro após a aplicação de um movimento representado por um vetor.

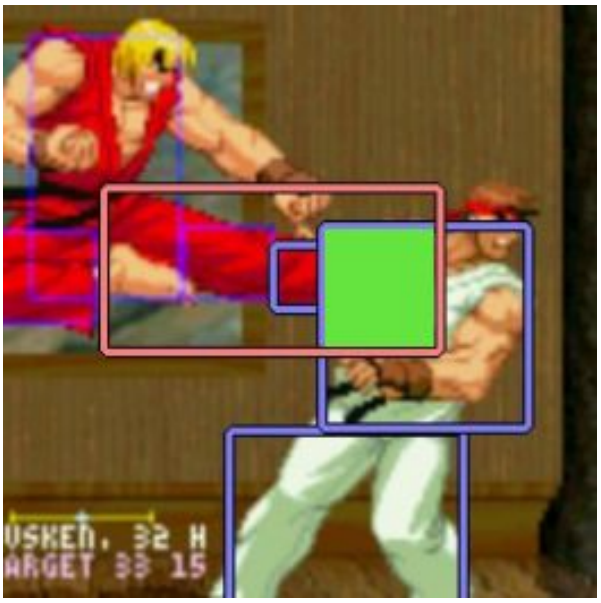
Em geometria, os pontos de um retângulo correspondem aos pontos do interior, excluindo-se os pontos da fronteira. Assim, dois retângulos interceptam-se somente se a área da interseção for maior que 0.

Para detectar uma colisão, recebemos os seguintes dados:

- um retângulo r representando o tamanho da tela do jogo,
- um conjunto A de retângulos que representam as caixas de colisão do ataque e seus vetores de movimento,
- um conjunto B de retângulos que representam as caixas de colisão do adversário e seus vetores de movimento,

Seu trabalho será encontrar a maior área de intersecção, **após a aplicação dos movimentos**, entre um retângulo de A e um retângulo de B considerando-se apenas as áreas de interseção que também interceptam r.

Por exemplo, digamos que foi aplicada a movimentação que resultou na imagem acima. A maior área de intersecção é a destacada a seguir:



Na figura, podemos ver apenas um retângulo de ataque que intersecta dois retângulos do adversário. A intersecção de maior área corresponde ao retângulo verde.

Testando e encontrando as intersecções

Após ler a entrada e aplicar o movimento devido de cada retângulo, será necessário testar se dois retângulos se intersectam e construir o retângulo resultante da intersecção.

Dado dois retângulos $r_0 = (p_0, p_1)$ e $r_1 = (p_2, p_3)$, sendo p_0 e p_2 pontos do canto superior esquerdo e p_1 e p_3 pontos do canto inferior direito dos retângulos, uma forma de checar se há uma intersecção entre r_0 e r_1 é primeiramente projetar o retângulo no eixo x e no eixo y, formando os intervalos $(p_0.x, p_1.x)$ e $(p_2.x, p_3.x)$ no eixo x; e $(p_0.y, p_1.y)$ e $(p_2.y, p_3.y)$ no eixo y. Depois, basta verificar se há uma intersecção não vazia s_0 nos intervalos projetados no eixo y e se há uma intersecção não vazia s_1 dos intervalos projetados no eixo x. Veja a Figura 1.

A maneira mais fácil de verificar se dois intervalos (l_0, l_1) e (m_0, m_1) se Intersectam é simplesmente testar se $l_0 < m_1$ e $l_1 > m_0$.

Podemos construir o intervalo s que é a intersecção de dois intervalos (l_0, l_1) e (m_0, m_1) que se intersectam, atribuindo $s = (\max(l_0, l_1), \min(m_0, m_1))$.

Voltando para a intersecção do retângulo, a intersecção r_3 entre os retângulos r_1 e r_2 pode ser construída facilmente através dos intervalos $s_0 = (a, b)$ e $s_1 = (c, d)$ vistos anteriormente, dado que $r_3 = (p, q)$ tal que $p = (c, a)$ e $q = (d, b)$. Como ilustrado na Figura 1.

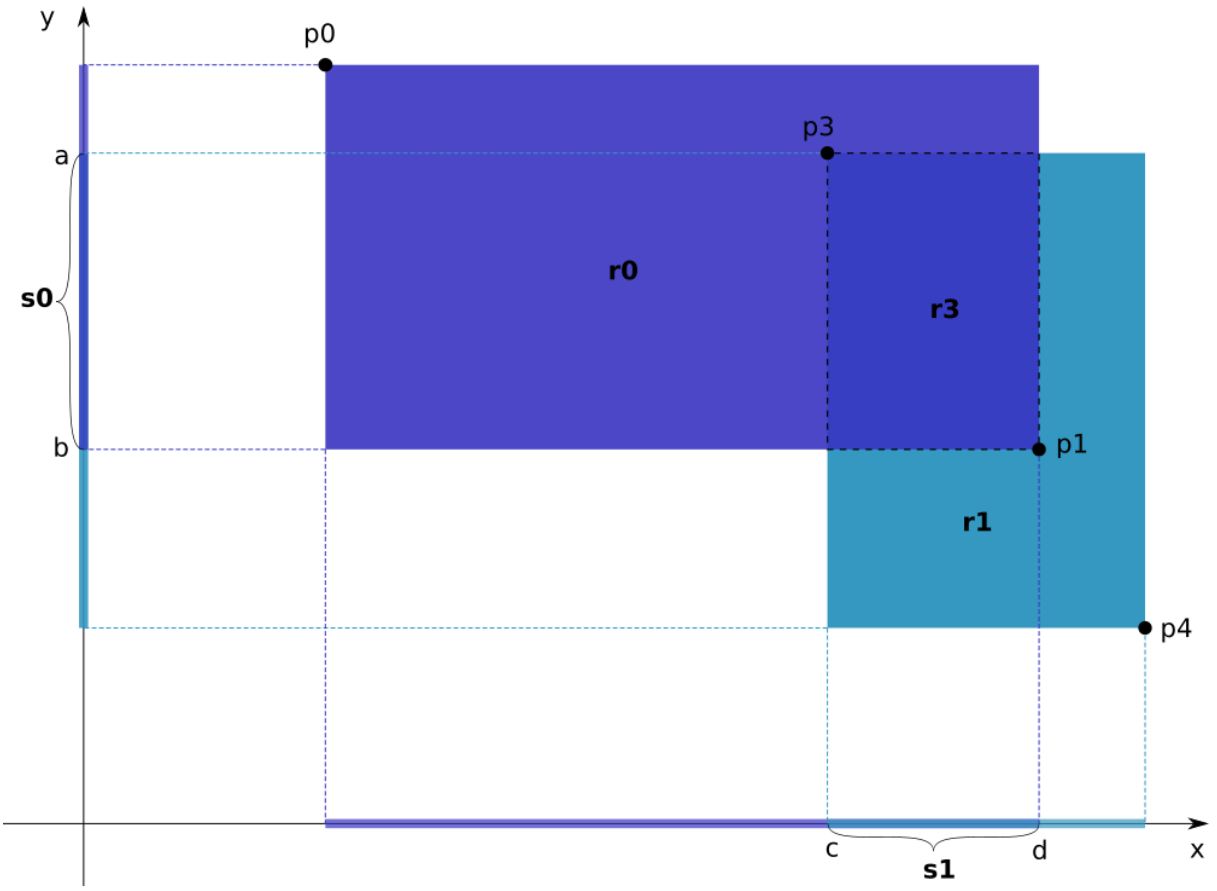


Figura 1: Intersecção entre dois retângulos e suas projeções no eixo x e y. Os intervalos s_0 e s_1 não são vazios nesse exemplo.

Entrada

Um retângulo é representado por uma sequência de quatro números de ponto flutuante x_0, y_0, x_1, y_1 que correspondem ao ponto superior esquerdo (x_0, y_0) e ao ponto inferior direito (x_1, y_1) do retângulo. Um vetor de movimento é representado por dois pontos flutuantes x_v e y_v, que indicam a movimentação no eixo x e no eixo y, respectivamente.

A entrada será composta primeiramente pelo retângulo que representa a tela.

Depois, há um inteiro $N\ (0 \leq N \leq 1000)$ que é o número de caixas de colisão do ataque. Seguem N caixas de colisão de ataque e movimentações correspondentes. As caixas e as movimentações são representadas respectivamente por um retângulo e um vetor.

Por fim, há um inteiro $M\ (0 \leq M \leq 1000)$, que é o número de caixas de colisão do adversário. Seguem M caixas de colisão do adversário e movimentações correspondentes. As caixas e as movimentações são representadas respectivamente por um retângulo e um vetor.

Saída

A maior área de intersecção, **após a aplicação dos movimentos**, entre uma caixa de colisão de ataque e uma caixa de colisão do adversário considerando-se apenas as áreas de interseção que também interceptam r. A área deverá ser representada por um número de ponto flutuante com precisão de duas casas decimais após o ponto. Caso não seja detectada nenhuma intersecção válida, a saída deve conter o texto “null”.

Exemplo

Entrada

```
0 100 100 0
1
20 60 40 40
0 0
1
30 50 50 30
0 0
```

Saída

```
100.00
```

Entrada

```
0 100 100 0
1
120 160 140 140
0 0
1
130 150 150 130
0 0
```

Saída

```
null
```

Entrada

```
-1 1 1 -1
1
-0.3 0.3 -0.1 0.1
0.15 -0.15
1
0.1 -0.1 0.3 -0.3
-0.15 0.15
```

Saída

```
0.01
```

Entrada

```
0 8 8 0
2
1 7 2 6
-4 -3
7 5 8 6
-2.5 0
2
5 7 7 6
0 -2
2 4 4 3
-5 3
```

Saída

```
0.50
```

Critérios específicos

- Utilize apenas a precisão do tipo double do C, não utilize float
- Você receberá os seguintes arquivos prontos e não deverá modificá-los:
    - retangulo.h: interface do retangulo

    - vetor.h: interface do vetor
- Deverão ser submetidos os seguintes arquivos:
    - retangulo.c: implementação da interface do retangulo
    - vetor.c: implementação da interface vetor
    - lab04.c programa principal
- Você deve implementar as funções dos TADs vetor e retângulo utilizando passagem por referência.
- Tempo máximo de execução: 1 segundos.

Dicas

Para imprimir a saída com precisão de dois números após o ponto, você pode utilizar a seguinte formatação do printf

```
double area = calcular_area(...);
printf("%.2f", area);
```

Testando

Para compilar usando o Makefile fornecido e verificar se a solução está correta basta seguir o exemplo abaixo.

```
make
./lab04 < arq01.in > arq01.out
diff arq01.out arq01.res
```

onde `arq01.in` é a entrada (casos de testes disponíveis no SuSy) e `arq01.out` é a saída do seu programa. O Makefile também contém uma regra para testar todos os testes de uma vez; nesse caso, basta digitar:

```
make testar_tudo
```

Isso testará o seu programa com os casos abertos. Após o prazo, os casos de teste fechados serão liberados e podem ser baixados digitando-se:

```
make baixar_fechados
```

Para ver quanto tempo seu programa demora em cada teste, digite:

```
make tempo
```

Observações gerais

No SuSy, haverá 3 tipos de tarefas com siglas diferentes para cada laboratório de programação. Todas possuirão os mesmos casos de teste. As siglas são:

1. **DRAFT**: Esta tarefa serve para testar o programa no SuSy antes de submeter a versão final. Nessa tarefa, tanto o prazo quanto o número de submissões são ilimitados, porém arquivos submetidos aqui não serão corrigidos.

2. **ENTREGA**: Esta tarefa tem limite de *uma única submissão* e serve para entregar a versão final dentro do prazo estabelecido para o laboratório. Não use essa tarefa para testar o seu programa: submeta aqui quando não for mais fazer alterações no seu programa.

3. **FORAPRAZO**: Esta tarefa tem limite de *uma única submissão* e serve para entregar a versão final após o prazo estabelecido para o laboratório, mas com nota reduzida (conforme a ementa). O envio nesta tarefa irá substituir a nota obtida na tarefa ENTREGA apenas se o aluno tiver realizado as correções sugeridas no feedback ou caso não tenha enviado anteriormente em ENTREGA.

Observações sobre SuSy:

- Versão do GCC: gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-36)

- Flags de compilação:
  `-std=c99 -Wall -Werror -Werror=vla -pedantic-errors -g -lm`

Além das observações acima, esse laboratório será avaliado pelos critérios gerais:

- Indentação de código e outras boas práticas, tais como:
    - uso de comentários (úteis e apenas quando forem relevantes);
    - código simples e fácil de entender;
    - sem duplicidade (partes que fazem a mesma coisa).
- Organização do código:
    - tipos de dados criados pelo usuário e funções bem definidas e tão independentes quanto possível.
- Corretude do programa:
    - programa correto e implementado conforme solicitado no enunciado;
    - inicialização de variáveis sempre que for necessário;
    - dentre outros critérios.