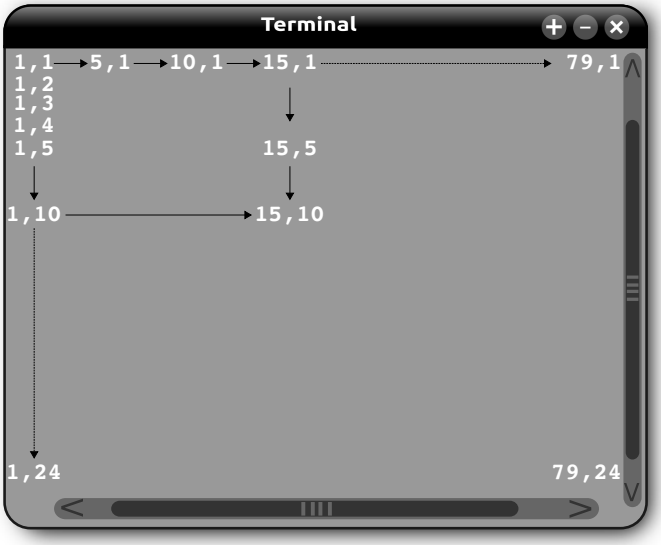
> Os que sonham de dia são conscientes de muitas coisas que escapam aos que sonham apenas à noite.
>
> Allan Kardec

1 Algumas coisas que precisamos saber sobre a tela

Estamos trabalhando com um ambiente conhecido como *modo texto*. Os jogos feitos para este ambiente são chamados de *ASCII Games*, pois utilizam apenas caracteres da tabela ASCII (letras, números ou símbolos que estão no seu teclado). A tela do terminal utiliza o conceito cartesiano de X,Y. Sendo X a coluna na tela e o Y a linha na tela. O terminal padrão trabalha com até 24 linhas e 79 colunas. A posição inicial da tela é 1,1 e a final 24,79 (tela maximizada). Observe a figura 1:

Figura 1: Coordenadas do terminal.



Para imprimir uma mensagem ou símbolo em alguma parte da tela, é necessário posicionar o cursor antes. A nossa biblioteca fornece a função *gotoXY(posiçãoX, posiçãoY)* para posicionar o cursor no terminal. Qualquer impressão na tela (mensagem por exemplo), será dado a partir da posição inicial informada pelo programador. Observe que programa 1 irá posicionar nas mesmas posições que constam na figura 1 e imprimirá o um asterisco em cada posição.

Programa 1: Conceitos de tela

```
1  //programa_001.cpp
   #include "biblaureano.h"

   int main()
   {
6    gotoXY(1,1);
     cout << "*";
     gotoXY(1,24);
```

```
   cout << "*";
   gotoXY(1,2);
   cout << "*";
   gotoXY(1,3);
   cout << "*";
   gotoXY(1,4);
   cout << "*";
   gotoXY(1,5);
   cout << "*";
   gotoXY(1,10);
   cout << "*";

   gotoXY(5,1);
   cout << "*";
   gotoXY(10,1);
   cout << "*";
   gotoXY(15,1);
   cout << "*";

   gotoXY(15,5);
   cout << "*";
   gotoXY(15,10);
   cout << "*";

   gotoXY(79,1);
   cout << "*";

   gotoXY(79,24);
   cout << "*";

   getch();
   return 0;
}
```

Trabalhar com coordenadas de tela para desenhar figuras implica em atribuir valores específicos para a posição *X* e a posição *Y* na função *gotoXY(posiçãoX,posiçãoY)*. A figura 2 exemplifica quando devemos aumentar ou diminuir o valor das variáveis correspondentes as posições:

Combinando as funções de cores e posicionamento de tela é possível mostrar quadrados coloridos na tela. O programa 2 desenha um conjunto de quadrados, começando pelas bordas mais externas e indo em direção ao centro da tela. Cada quadrado é impresso a cada 1 segundo. Para pausar o programa utilizamos a função *espera(tempo)* para determinar a parada, no caso o valor informado (100) refere-se a 1 segundo.

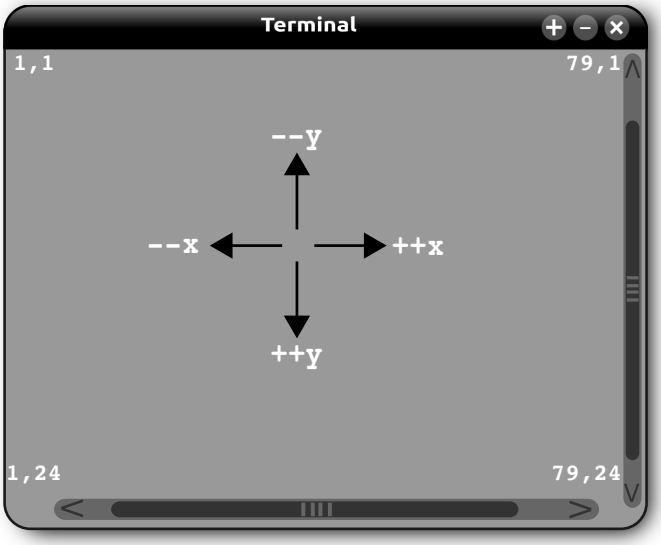
Programa 2: Desenhando quadrados na tela para demonstrar as trocas de coordenadas

```
//programa_002.cpp
#include "biblaureano.h"

int main()
{
   int xInicial, xFinal, yInicial, yFinal;

   xInicial = 1; //posição inicial na tela
   xFinal = 79; //posição final na tela
```

Figura 2: O pulo do gato.




```
yInicial = 1; //posição inicial na tela
yFinal = 24; //posição final na tela

int cor = 1; // começa com a cor vermelha (linux) ou azul (windows)
while( yInicial <= yFinal )
{
   mudaCor((COR)cor); //muda a cor

   //imprime as colunas (linhas da esquerda e direita) do quadrado
   int Y = yInicial;
   while( Y<=yFinal )
   {
      gotoXY(xInicial, Y);
      cout << "#";

      gotoXY(xFinal, Y);
      cout << "#";
      ++Y;
   }
   //imprime as linhas de cima e baixo do quadrado
   int X = xInicial;
   while( X<xFinal )
   {
      gotoXY(X,yInicial);
      cout << "#";

      gotoXY(X,yFinal);
      cout << "#";

```

```
            ++X;
        }
        cor++; //muda para a próxima cor
        if(cor > QTY_COR )
        {
          cor=1; // volta para a cor inicial
        }
        xInicial+=2; //aumenta a coluna inicial em 2
        xFinal-=2; //diminui a coluna final em 2
        ++yInicial; //aumenta a linha inicial
        --yFinal; //diminui a linha final
        espera(100); //pausa 1 segundo
    }
    return 0;
}
```

2 Verificando as teclas pressionadas

Todo jogo tem algum tipo de controle: teclado, mouse ou *joystick*. Como estamos trabalhando com o terminal precisamos verificar se alguma tecla foi pressionada para podermos tomar uma ação (mover para cima, para direita ou atirar, por exemplo). A programação de um jogo na sua grande maioria, segue alguns passos definidos:

```
enquanto verdadeiro faça
  se foi pressionada alguma tecla então
     verifica qual tecla foi pressionada
     se tecla = A
         ação A
     senão se tecla = B
         ação B
     senão
        ...
     fim
  fim
  ação C
  ação D
  ...
fim
```

Isto é: o seu jogo não pode ficar *preso* esperando que algo seja pressionado. Na biblioteca encontramos a função *verificaKB(char tecla)* que recebe uma variável para receber qual tecla foi pressionada. Esta função retorna verdadeiro se alguma tecla foi pressionada ou falso senão.

Programa 3: Desenhando na tela

```
//programa_003.cpp
#include "biblaureano.h"

int main()
{
```

```
6   int linha,coluna;
    linha = 1;
    coluna = 1;
    bool ligado = true;
    tiraCursor(false); //desliga o cursor
11  while(true)
    {
       gotoXY(coluna,linha);
       if( ligado )
       {
16       cout << "*";
       }
       else
       {
         cout << " ";
21     }

       char tecla;
       if( verificaKB(tecla) )
       {
26        if( tecla == 'Q' || tecla == 'q') // finaliza o programa
          {
            break;
          }
          else if( tecla == K_RIGTH)
31        {
            coluna--;
            if( coluna < 1)
            {
              coluna = 1;
36          }
          }
          else if( tecla == K_LEFT )
          {
            coluna++;
41          if( coluna > 79)
            {
              coluna = 79;
            }
          }
46        else if( tecla == K_UP)
          {
            linha--;
            if( linha < 1)
            {
51            linha = 1;
            }
          }
          else if( tecla == K_DOWN)
          {
56          linha++;
            if( linha > 24)
            {
```

```
            linha = 24;
          }
        }
        else if( tecla == 'L' || tecla == 'l')
        {
           //se ligado = false, então ligado recebe true e o programa volta a desenhar na tela
           //se ligado = true, então ligado recebe false e o programa apaga os
           // caracteres já desenhados
           ligado = !ligado;
        }
     }
  }
  return 0;
}
```

2.1 Entendendo o programa

O programa 3 apresenta algumas novidades em relação ao que já vimos até o momento. Primeiro temos uma variável *ligado* que será utilizada para controlar quando um caractere será desenhado ou apagado na tela. Logo após temos a função *tiraCursor(estadoCursor)* que habilita o cursor (*true*) ou desabilita (*false*) o cursor na tela. Desabilitar o cursor acaba se tornando uma vantagem pois após imprimir um caractere na coluna 10 o cursor se posicionará automaticamente na coluna 11:

```
...
bool ligado = true;
tiraCursor(false); //desliga o cursor
...
```

Na sequência temos uma estrutura de repetição infinita, necessário para que nosso programa execute continuamente, e a verificação se vamos imprimir um caractere ou limpar aquela posição da tela:

```
...
while(true)
{
   gotoXY(coluna,linha);
   if( ligado )
   {
     cout << "*";
   }
   else
   {
     cout << " ";
   }
   ...
```

E finalmente temos a verificação se alguma tecla foi pressionada. Se o teclado foi acionado é verificado qual tecla foi utilizada (se alguma letra ou as setas direcionais). Se alguma seta direcional foi acionada, então o programa incrementa ou decrementa a linha ou coluna e verifica se não foi passado dos limites da tela:

```
   ...
   char tecla;
```

```
    if( verificaKB(tecla) )
    {
       if( tecla == 'Q' || tecla == 'q') // finaliza o programa
       {
         break;
       }
       else if( tecla == K_RIGTH)
       {
         coluna--;
         if( coluna < 1)
         {
           coluna = 1;
         }
       }
       ...
```

A função *verificaKB() prende* o programa, isto, ele pausa o programa até que o usuário pressione alguma tecla. Para verificar se alguma tecla foi pressionada e ao mesmo tempo não prender o programa é necessário utilizar a função *kbhit()* em conjunto com *getch()*.

3 Verificando teclas pressionadas e o tempo que passou

Naturalmente um jogo não pode *pausar* esperando que o jogador pressione alguma tecla. Em jogos de corrida, por exemplo, os outros carros irão continuar correndo mesmo que o piloto esteja indeciso sobre frear, acelerar ou trocar de marcha. Com as funções *kbhit()* e *getch()* é possível verificar se uma tecla foi pressionada e testar a tecla, a função *tempoDecorrido()* permite verificar e calcular quanto tempo passou desde um determinado momento e finalmente a função *noecho(habilita)* permite desligar ou ligar a visualização do que se está digitando (útil para digitação de senhas ou para evitar caracteres *perdidos* quando se está jogando). No programa 4 podemos observar o uso de todas essas funções:

Programa 4: Controle de teclado e tempo

```
//programa_004.cpp
#include "biblaureano.h"

int main(void)
{
  time_t horaAntes; //quantidade de segundos
  horaAntes = tempoDecorrido(); //pega hora inicial

  int quantidadeInimigos = 10;

  limparTela();
  gotoXY(10,10);

  mudaCor(GREEN);
```

```
  cout <<"Pressione f para matar os inimigos..";
  limpaEfeito(); //limpa os efeitos de cores
  tiraCursor(false);
  noecho(true); //desliga impressão do que foi digitado
  while(true)
  {
    if( kbhit() )
    {
       char tecla = getch();
       if( tecla == 'f')
       {
          quantidadeInimigos--;
          if( quantidadeInimigos == 0)
          {
            gotoXY(10,14);
            mudaCor(CYAN);
            cout << "Parabéns... você matou todos os inimigos" << endl;
            limpaEfeito();
            break; //encerra o programa
          }
       }
    }
    gotoXY(10,11);
    mudaCor(RED);
    cout << "Faltam alguns segundos para acabar o seu tempo:";
    cout << tempoDecorrido( horaAntes ) << " " ;
    gotoXY(10,12);
    mudaCor(BLUE);
    cout << "Inimigos Restantes:";
    cout << quantidadeInimigos << " ";
    limpaEfeito();

    //verifica quando tempo se passou
    if(tempoDecorrido( horaAntes ) > 10 && quantidadeInimigos >0)
    {
      gotoXY(10,13);
      cout << "Passou 10 segundos e você não matou os inimigos!!!" << endl;
      cout << "Recomeçando a contagem..." << endl;
      quantidadeInimigos = 10;
      //pega nova hora inicial
      horaAntes = tempoDecorrido();
    }
  }

  cout << "Game over!!" << endl;
  return 0;
}
```

3.1 Entendendo o programa - coisas novas estão surgindo

Como o programa 4 trabalha com controle de tempo, precisamos criar uma variável compatível para guardar essa informação na memória do computador. A função *tempoDecorrido()* quando chamada sem passar um valor, irá retornar a hora atual do computador para uso na sequência:

```
...
time_t horaAntes; //quantidade de segundos
horaAntes = tempoDecorrido(); //pega hora inicial
...
```

Desligamos os efeito de cores (função *limpaEfeito()*), desabilitamos o cursor na tela (função *tiraCursor(false)*) e finalmente informamos que qualquer valor digitado não será apresentado na tela (função *noecho(true)*):

```
...
limpaEfeito(); //limpa os efeitos de cores
tiraCursor(false);
noecho(true); //desliga impressão do que foi digitado
...
```

Caso você precise que algum valor digitado seja mostrado na tela, basta utilizar uma sequência similar a:

```
...
noecho(false);
string nome = readString("Entre com seu nome:");
noecho(true);
...
```

Agora uma sequência importante do nosso programa. O programa está em um laço continuo (*loop* infinito). É verificado alguma tecla foi pressiona (função *kbhit()*) e caso verdadeiro, a tecla é capturada na sequência:

```
...
while(true)
{
   if( kbhit() )
   {
      char tecla = getch();
      if( tecla == 'f')
      {
         quantidadeInimigos--;
         if( quantidadeInimigos == 0)
         {
            gotoXY(10,14);
            mudaCor(CYAN);
            cout << "Parabéns... você matou todos os inimigos" << endl;
            limpaEfeito();
            break; //encerra o programa
         }
```

```
    }
}
...
```

Lembre-se que a função *kbhit()* não pausa o programa esperando que você digite alguma coisa. A função *verificaKB(tecla)* faz uma pausa no programa para verificar e retornar uma tecla pressionada.

```
...
gotoXY(10,11);
mudaCor(RED);
cout << "Faltam alguns segundos para acabar o seu tempo:";
cout << tempoDecorrido( horaAntes ) << " " ;
gotoXY(10,12);
mudaCor(BLUE);
cout << "Inimigos Restantes:";
cout << quantidadeInimigos << " ";
...
```

Lembrando que o objetivo era que o jogador destruísse 10 inimigos em menos de 10 segundos. Finalmente verificamos se o limite de tempo de 10 segundos foi ultrapassado. Para verificar o tempo decorrido entre o tempo inicial (pego anteriormente) e a hora atual, basta passar o tempo inicial para a função *tempoDecorrido()*. Se o limite de tempo foi ultrapassado e o objetivo não foi cumprido, recomeçamos a contagem de inimigos e do tempo:

```
...
//verifica quando tempo se passou
if((tempoDecorrido( horaAntes ) > 10 && quantidadeInimigos >0)
{
  gotoXY(10,13);
  cout << "Passou 10 segundos e você não matou os inimigos!!!" << endl;
  cout << "Recomeçando a contagem..." << endl;
  quantidadeInimigos = 10;
  //pega nova hora inicial
  horaAntes = tempoDecorrido();
}

...
```

4 Pausar o programa ou não? *kbhit()* ou *verificaKB()* ?

Atualmente temos jogos de todos os tipos disponíveis. Alguns exigem interação contínua e mesmo que o jogador fique parado ou AFK - *Away from keyboard* (longe do teclado) o jogo prossegue, por exemplo, no *Counter Strike* o jogador pode ficar parado, mas os demais irão continuar jogando e provavelmente quando o jogador que estava AFK retornar terá um grata surpresa (estará morto). Outros jogos tem um tempo determinado para que o jogador

tome uma ação (*Jewels* por exemplo). Para esses tipos de jogos devemos utilizar as funções *kbhit()* e *getch()* para determinar um evento no teclado.

Jogos como campo minado, truco, batalha naval, entre outros não tem a mesma rigorosidade e podem esperar que o jogador tome uma ação. A fim de evitar processamento desnecessário e economizar energia do computador, devemos utilizar a função *verificaKB()* para determinar os eventos do teclado.

5 Para fixar - imprimindo um triângulo na tela

O programa 5 demonstra como trabalhar com linhas diagonais. O *segredo* aqui é alterar a linha e a coluna simultaneamente. Lembre-se sempre de guardar as coordenadas iniciais da sua figura.

Programa 5: Impressão de um triângulo

```
//programa_005.cpp
#include "biblaureano.h"

int main()
{
  int x, y, tamanho;

  const int TEMPO = 10;

  x = readInt("Valor de X:");
  y = readInt("Valor de Y:");
  tamanho = readInt("Tamanho:");

  //int contadora = 0;
  while( true )
  {
    int cor = randomico(1,7);
    int xInicial = x,
       yInicial = y;

    mudaCor((COR)cor);
    while( yInicial >= (y-tamanho))
    {
       gotoXY( xInicial, yInicial);
       cout << "*";
       //subindo
       ++xInicial;
       --yInicial;
       espera(TEMPO);
    }
    while( yInicial <= y )
    {
       gotoXY( xInicial, yInicial);
       cout << "*";
       //descendo
       ++xInicial;
       ++yInicial;
       espera(TEMPO);
```

```
    }
    --xInicial;
    while( xInicial >= x)
    {
      gotoXY( xInicial, yInicial);
      cout << "*";
      --xInicial;
      espera(TEMPO);
    }
    //quando o teclado for pressionado, termina o programa
    if( kbhit() )
      break;
  }
  cout << endl << "Game over!!" << endl;
}
```

6 Nosso primeiro jogo - simples é claro

Finalmente, o programa 6 implementa um jogo simples de espaçonave. Observe neste programa que temos basicamente 3 itens principais de controle:

1. O movimento da espaçonave inimiga, dada pelas variáveis *xInimigo* e *yInimigo* que indicam as coordenadas atuais, o cálculo aleatório do rumo da espaçonave (direita ou esquerda) e os limites de tela;
2. O movimento da espaçonave amiga, verificando o teclado para alterar as variáveis *xAmigo* e *yAmigo*;
3. O caminho a ser percorrido pelo tiro.

Programa 6: Nosso primeiro jogo

```
//programa_006.cpp
#include "biblaureano.h"

int main()
{
  const string INIMIGO="|-|";
  const string AMIGO="_#_";

  int yInimigo; // linha do alvo
  int xInimigo; // coluna do alvo
  int yAmigo; //linha do aviao
  int xAmigo; //coluna do avião

  //posições iniciais das naves
  yInimigo = 5;
  xInimigo = randomico(1,80);

  yAmigo = 20;
  xAmigo = randomico(1,80);

  noecho(true);
  tiraCursor(false);
```

```
while(true)
{
  gotoXY(xInimigo,yInimigo);
  cout << INIMIGO << endl;
  gotoXY(xAmigo,yAmigo);
  cout << AMIGO << endl;
  gotoXY(1,23);
  cout << "F-fogo | S-sai do programa";

  espera(10);

  if( kbhit() )
  {
    char tecla = getch();

    gotoXY(xAmigo,yAmigo);
    cout << " ";

    switch(tecla)
    {
     //não podemos ultrapassar os limites
     //da tela
     case K_LEFT:
       xAmigo++;
       if( xAmigo > 77 )
       {
         xAmigo=1;
       }
       break;
     case K_RIGTH:
       xAmigo--;
       if( xAmigo < 1 )
       {
        xAmigo = 77;
       }
       break;
     case 's':
     case 'S':
       return 1;
     case 'f':
     case 'F':
     {
       int xFogo; //posicao coluna tiro;
       int yFogo; //posicao linha tiro;
       xFogo = xAmigo+1; // o tiro sai do meio do aviao
       yFogo = yAmigo-1; // o tiro sai uma linha acima do aviao

       //loop para controlar o tiro que vai para o inimigo
       while(true)
       {
         gotoXY(xInimigo,yInimigo);
         cout << INIMIGO;
```

```
            gotoXY(xFogo, yFogo);
77          cout << "+";

            gotoXY(xAmigo,yAmigo);
            cout << AMIGO;

82          //se tiro alcançou a mesma linha do inimigo
            if( yFogo == yInimigo )
            {
              break;
            }
87          yFogo--;

            espera(5);
            //apaga o rastro do tiro e do inimigo
            gotoXY(xFogo, yFogo+1);
92          cout << " " << endl;
            gotoXY(xInimigo,yInimigo);
            cout << " ";

            //inimigo continua se movimentando
97          if( randomico()%2 == 0)
            {
              //não podemos ultrapassar os limites
              //da tela
              xInimigo++;
102           if( xInimigo > 77 )
              {
                xInimigo=1;
              }
            }
107         else
            {
              //não podemos ultrapassar os limites
              //da tela
              xInimigo--;
112           if( xInimigo < 1 )
              {
                xInimigo = 77;
              }
            }
117       }
          //verificando onde o tiro pegou
          if( xFogo == (xInimigo+1) ) // acertou no meio
          {
            cout << endl << "Parabéns !! Detonou !!!";
122       }
          else if( (xInimigo+2) == xFogo ) // acertou asa direita
          {
            cout << endl << "Você só arranhou a minha asa direita...";
          }
127       else if( xFogo == (xInimigo) ) //acertou asa esquerda
          {
```

```
            cout << endl << "Você só arranhou a minha asa esquerda...";
          }
          else
          {
            cout << endl << "Como você é ruim !!!";
          }
          espera(100);
        }
      }
    }

    gotoXY(xInimigo,yInimigo);
    cout << " ";

    //movimentacao da nave
    if( randomico()%2 == 0)
    {
      //não podemos ultrapassar os limites
      //da tela
      xInimigo++;
      if( xInimigo > 77 )
      {
        xInimigo=1;
      }
    }
    else
    {
      //não podemos ultrapassar os limites
      //da tela
      xInimigo--;
      if( xInimigo < 1 )
      {
        xInimigo = 77;
      }
    }
  }
  cout << "Game over!!!" << endl;
  return 0;
}
```

7 Alterando o tamanho da tela

Como colocado anteriormente, o terminal padrão trabalha com até 24 linhas e 79 colunas. Mas é possível alterar o tamanho da tela para obter melhores resultados. A função *mudaTamanhoTerminal(int x,int y)* permite ajustar o tamanho do terminal em tempo de execução do programa. O ideal é ajustar o tamanho da tela no início do seu jogo e evitar demais alterações no decorrer. O programa 7 demonstra o funcionamento dessa função:

Programa 7: Ajuste de tamanho de tela

```
//lembrar de entrar em Settings -> Enviroment.. ->
//Terminal to launch console programns e colocar "gnome-terminal -t $TITLE -x"
```

```
//programa_007.cpp
#include "biblaureano.h"

int main ()
{
  int x=readInt("Valor de X:");
  int y=readInt("Valor de Y:");

  while(x&&y)
  {
    mudaTamanhoTerminal(x,y);
    limparTela();
    //imprime as informações em todas as
    //novas coordenadas
    gotoXY(1,1);
    cout << "(1,1)";
    gotoXY(x-8,1);
    cout << "(" << numeroToString(x) << ",1)";
    gotoXY(1,y);
    cout << "(1," << numeroToString(y) << ")";
    gotoXY(x-8,y);
    cout << "(" << numeroToString(x) <<"," << numeroToString(y) << ")";
    //centraliza uma mensagem na tela
    string mensagem="Teste de tamanho! Mensagem centralizada! Dormindo 5 segundos....";
    gotoXY((x-mensagem.size())/2,y/2);
    cout << mensagem << endl;
    espera(500);
    limparTela();
    x=readInt("Valor de X:");
    y=readInt("Valor de Y:");
  }
  return 0;
}
```

8 O nosso primeiro jogo - ajustado para uma tela maior

Utilizando a função *mudaTamanhoTerminal(int x,int y)*, podemos alterar o programa 6 para perguntar ao usuário se ele deseja jogar com uma tela maior (ou menor). Mas é necessário tomar apenas uma precaução em nosso: respeitar os novos limites da tela. O programa 8 reflete essas alterações:

Programa 8: Nosso primeiro jogo - ajustado para uma tela maior

```
//programa_008.cpp
#include "biblaureano.h"

int main()
{
  const string INIMIGO="|-|";
  const string AMIGO="_#_";

  int xTam = readInt("Informe X:");
```

```
  int yTam = readInt("Informe Y:");
  mudaTamanhoTerminal(xTam,yTam);

  int yInimigo; // linha do alvo
  int xInimigo; // coluna do alvo
  int yAmigo; //linha do aviao
  int xAmigo; //coluna do avião

  //posições iniciais das naves
  yInimigo = 5;
  xInimigo = randomico(1,xTam);

  yAmigo = yTam-4;
  xAmigo = randomico(1,xTam);

  noecho(true);
  tiraCursor(false);
  while(true)
  {
    gotoXY(xInimigo,yInimigo);
    cout << INIMIGO << endl;
    gotoXY(xAmigo,yAmigo);
    cout << AMIGO << endl;
    gotoXY(1,yTam-2);
    cout << "F-fogo | S-sai do programa";

    espera(10);

    if( kbhit() )
    {
      char tecla = getch();

      gotoXY(xAmigo,yAmigo);
      cout << " ";

      switch(tecla)
      {
       case K_LEFT:
          xAmigo++;
          if( xAmigo > (xTam-3) )
          {
            xAmigo=1;
          }
          break;
       case K_RIGTH:
          xAmigo--;
          if( xAmigo < 1 )
          {
           xAmigo = xTam-3;
          }
          break;
       case 's':
       case 'S':
```

```
      return 1;
    case 'f':
    case 'F':
    {
      int xFogo; //posicao coluna tiro;
      int yFogo; //posicao linha tiro;
      xFogo = xAmigo+1; // o tiro sai do meio do aviao
      yFogo = yAmigo-1; // o tiro sai uma linha acima do aviao

      //loop para controlar o tiro que vai para o inimigo
      while(true)
      {
        gotoXY(xInimigo,yInimigo);
        cout << INIMIGO;

        gotoXY(xFogo, yFogo);
        cout << "+";

        gotoXY(xAmigo,yAmigo);
        cout << AMIGO;

        //se tiro alcançou a mesma linha do inimigo
        if( yFogo == yInimigo )
        {
          break;
        }
        yFogo--;

        espera(5);
        //apaga o rastro do tiro e do inimigo
        gotoXY(xFogo, yFogo+1);
        cout << " " << endl;
        gotoXY(xInimigo,yInimigo);
        cout << " ";

        //inimigo continua se movimentando
        if( randomico()%2 == 0)
        {
          xInimigo++;
          if( xInimigo > (xTam-3) )
          {
            xInimigo=1;
          }
        }
        else
        {
          xInimigo--;
          if( xInimigo < 1 )
          {
            xInimigo = xTam-3;
          }
        }
      }
```

```
            //verificando onde o tiro pegou
            if( xFogo == (xInimigo+1) ) // acertou no meio
            {
              cout << endl << "Parabéns !! Detonou !!!";
            }
            else if( (xInimigo+2) == xFogo ) // acertou asa direita
            {
              cout << endl << "Você só arranhou a minha asa direita...";
            }
            else if( xFogo == (xInimigo) ) //acertou asa esquerda
            {
              cout << endl << "Você só arranhou a minha asa esquerda...";
            }
            else
            {
              cout << endl << "Como você é ruim !!!";
            }
            espera(100);
          }
        }
      }

      gotoXY(xInimigo,yInimigo);
      cout << " ";

      //movimentacao da nave
      if( randomico()%2 == 0)
      {
        xInimigo++;
        if( xInimigo > (xTam-3) )
        {
          xInimigo=1;
        }
      }
      else
      {
        xInimigo--;
        if( xInimigo < 1 )
        {
          xInimigo = xTam-3;
        }
      }
    }
    cout << "Game over!!!" << endl;
    return 0;
}
```

8.1 Entendendo o programa

Inicialmente pedimos ao jogador para informar as novas dimensões da tela e chamamos a função *mudaTamanhoTerminal()* passando as novas dimensões:

```
...
int xTam = readInt("Informe X:");
int yTam = readInt("Informe Y:");
mudaTamanhoTerminal(xTam,yTam);
...
```

Na sequência, utilizamos as novas dimensões para calcular as posições iniciais do inimigo e da nave do jogador:

```
...
//posições iniciais das naves
yInimigo = 5;
xInimigo = randomico(1,xTam);

yAmigo = yTam-4;
xAmigo = randomico(1,xTam);
...
```

E o principal, os controles de limite da nave inimiga e da nave do jogador:

```
...
     switch(tecla)
     {
       case K_LEFT:
          xAmigo++;
          if( xAmigo > (xTam-3) )
          {
            xAmigo=1;
          }
          break;
       case K_RIGTH:
          xAmigo--;
          if( xAmigo < 1 )
          {
            xAmigo = xTam-3;
          }
          break;
...
...
  //movimentação da nave
  if( randomico()%2 == 0)
  {
    xInimigo++;
    if( xInimigo > (xTam-3) )
    {
      xInimigo=1;
    }
  }
  else
  {
    xInimigo--;
    if( xInimigo < 1 )
    {
      xInimigo = xTam-3;
```

```
        }
    }
...
```

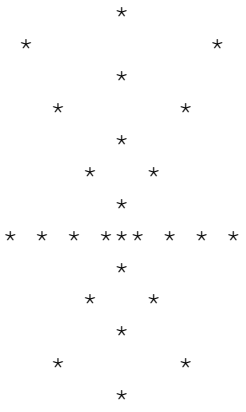
9 Exercícios

1. Altere o programa 6 para que o inimigo também possa se movimentar por linhas diferentes de forma aleatória.
2. Realizar um programa que cria uma estrela com altura = N, sendo N informado pelo usuário. Seja criativo com o uso de cores. Exemplos para:

   N=3:

   N=5:

   N=7:



```
*          *
     *
```

3. Escreva um programa que gere, para um valor $N \geq 0$ fornecido pelo usuário, um *quadrado* de N linhas e N ':' nas posições da diagonal principal e os caracteres '+' nas demais posições. Sendo $N \geq 3$ e $N \leq 13$ e N sendo um número ímpar. É possível resolver este programa sem utilizar a função *gotoXY()*, use a sua imaginação. Por exemplo, para $N = 5$ o programa deve gerar colunas que tenha caracteres:

```
:++++
+:+++
++:++
+++:+
++++:
```

4. Faça um programa para simular a contagem regressiva para o lançamento de um foguete.