Snake

Slither your avatar's sinuous form around dodging obstacles and eating food. Proceed with caution. The more you eat the longer you get and you don't want to accidently bite your own tail. Use the arrow keys to change direction and if you want to speed things up press the TAB key.

This snake game is a rather relaxed game. The area is large and sparsely populated. It takes a long time before your tail becomes a real issue. But if the game is not exciting enough for you there are many things you can do to break the boredom. You can increase the number of obstacles (and food) by changing the values of NUMBARRIERS (and NUMFOOD) at the beginning of the code. If the pace is too slow for you look for the line:

```
if((DoSnake % 8) == 0)
```

and change the 8 to a lower number.

Snake was written by Robert Ferris (aka BAF), originally written in one hour as an entry to the MinorHack challenge, and slightly modified since then.

// Snake.cpp linsting begins:

```
#include <allegro.h>
#include <stdlib.h>
#include <time.h>
#include <vector>
#include <list>
#include <iostream>

const int NUMBARRIERS = 30;
const int NUMFOOD = 15;

volatile int GameTicker = 0;
void Ticker()
{
  ++GameTicker;
}
END_OF_FUNCTION(Ticker);


std::vector<int> RandomData;

void LoadExeData()
{
  char ThisFile[512];
  get_executable_name(ThisFile, 512);
  PACKFILE *Self = pack_fopen(ThisFile, "rb");

  int RandAmt = file_size(ThisFile) / 16;

  for(int i = 0; i < RandAmt; ++i)
    RandomData.push_back(pack_igetw(Self));

  pack_fclose(Self);
}

int GetRand()
{
  if(RandomData.size() <= 0)
    LoadExeData();

  int Rand = rand() % RandomData.size();
  int Ret = RandomData[Rand];
  RandomData.erase(RandomData.begin() + Rand);
  return Ret;
}

struct SnakePixel
{
  int x, y;

  SnakePixel(int X, int Y) : x(X), y(Y) { }
};


int main(int argc, char *argv[])
{
  srand(time(NULL));

  allegro_init();
  install_keyboard();
  install_timer();
```

// Listing continued on next page...

// Listing continued from previous page

```
  LOCK_VARIABLE(GameTicker);
  LOCK_FUNCTION(Ticker);
  install_int_ex(Ticker, BPS_TO_TIMER(60));

  set_color_depth(desktop_color_depth());
  if(set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0))
    if(set_gfx_mode(GFX_AUTODETECT, 640, 480, 0, 0))
      return -1;

  LoadExeData();

  BITMAP *buffer = create_bitmap(SCREEN_W, SCREEN_H);
  enum eSD
  {
    NORTH = 0,
    WEST,
    EAST,
    SOUTH
  } SnakeDirection;


  // set up board
  int BoardWidth = SCREEN_W / 5;
  int BoardHeight = SCREEN_H / 5;
  char **Board = new char*[BoardWidth];
  for(int i = 0; i < BoardWidth; ++i)
    Board[i] = new char[BoardHeight];

  // board edge barriers
  for(int x = 0; x < BoardWidth; ++x)
    Board[x][0] = Board[x][BoardHeight - 1] = 2;
  for(int y = 0; y < BoardHeight; ++y)
    Board[0][y] = Board[BoardWidth - 1][y] = 2;

  for(int x = 1; x < BoardWidth - 1; ++x)
    for(int y = 1; y < BoardHeight - 1; ++y)
      Board[x][y] = 0;

  for(int i = 0; i < NUMBARRIERS; ++i) // place barriers
    Board
                [(GetRand() % (BoardWidth - 2)) + 1]
                [(GetRand() % (BoardHeight - 2)) + 1] = 2;
  for(int i = 0; i < NUMFOOD; ++i) // place food
    Board
                [(GetRand() % (BoardWidth - 2)) + 1]
                [(GetRand() % (BoardHeight - 2)) + 1] = 1;
  // set up snake
  std::list<SnakePixel*> Snake;
  Snake.push_back(new SnakePixel(BoardWidth / 2, BoardHeight / 2));
  Board[BoardWidth/2][BoardHeight/2] = 3;
  SnakeDirection = (eSD)(GetRand() % 4);

  int DoSnake = 0;
  int moved = 0;
  int Score = 0;
  while(!key[KEY_ESC]) // main loop
  {
    while(GameTicker > 0)
    {
      if (!moved) {
        poll_keyboard();

        if(key[KEY_LEFT] && SnakeDirection!=EAST && SnakeDirection!=WEST)
```

// Listing continued on next page...

// Listing continued from previous page

```
        {
          SnakeDirection = WEST;
          moved = 1;
        }
        else if(key[KEY_RIGHT] && SnakeDirection!=WEST && SnakeDirection!=EAST)
        {
          SnakeDirection = EAST;
          moved = 1;
        }
        else if(key[KEY_UP] && SnakeDirection!=SOUTH && SnakeDirection!=NORTH)
        {
          SnakeDirection = NORTH;
          moved = 1;
        }
        else if(key[KEY_DOWN] && SnakeDirection!=NORTH && SnakeDirection!=SOUTH)
        {
          SnakeDirection = SOUTH;
          moved = 1;
        } else if(key[KEY_TAB]) DoSnake = 0;
      }

      if((DoSnake % 8) == 0)
      {
        moved = 0;
        switch(SnakeDirection)
        {
          case NORTH:
            Snake.push_front
              (new SnakePixel(Snake.front()->x, Snake.front()->y - 1));
            break;
          case WEST:
            Snake.push_front
              (new SnakePixel(Snake.front()->x - 1, Snake.front()->y));
            break;
          case EAST:
            Snake.push_front
              (new SnakePixel(Snake.front()->x + 1, Snake.front()->y));
            break;
          case SOUTH:
            Snake.push_front
              (new SnakePixel(Snake.front()->x, Snake.front()->y + 1));
            break;
        }
        if (Snake.size() > Score + 2) {
          Board[Snake.back()->x][Snake.back()->y] = 0;
          Snake.pop_back();
        }

        if(Board[Snake.front()->x][Snake.front()->y] > 1) // Hit Barrier
        {
          std::cout << "\nGame Over! Score: " << Score << "\n\n";
          return 0;
        }
        else if(Board[Snake.front()->x][Snake.front()->y] == 1) // FOOD!
        {
          ++Score;
          int xx,yy;
          do {
            xx = (GetRand() % (BoardWidth - 2)) + 1;
            yy = (GetRand() % (BoardHeight - 2)) + 1;
          } while (Board[xx][yy]);
          Board[xx][yy] = 1; // new food
        }
```

// Listing continued on next page...

// Listing continued from previous page

```
        Board[Snake.front()->x][Snake.front()->y] = 3;
      }

      ++DoSnake;
      --GameTicker;
    }

    while(GameTicker < 0)
    {
      rest(1);
    }

    clear_bitmap(buffer);

    for(int x = 0; x < BoardWidth; ++x)
    {
      for(int y = 0; y < BoardHeight; ++y)
      {
        if(Board[x][y] == 2) // Barrier
          rectfill(buffer, x * 5, y * 5, ((x + 1) * 5) - 1, ((y + 1) * 5) - 1,
            makecol(192, 0, 0));
        if(Board[x][y] == 1) // Food
          rectfill(buffer, x * 5, y * 5, ((x + 1) * 5) - 1, ((y + 1) * 5) - 1,
            makecol(0, 192, 256));
      }
    }

    float g = 255;
    for(std::list<SnakePixel*>::iterator i=Snake.begin(); i!=Snake.end(); ++i)
    {
      rectfill(buffer,(*i)->x*5, (*i)->y*5, (((*i)->x+1)*5)-1, (((*i)->y+1)*5)-1,
        makecol(96, (int)g, 256-(int)g));
      g -= 128.0/Snake.size();
    }

    textprintf_right_ex(buffer, font, SCREEN_W, 0, makecol(255, 0, 0),
      makecol(255, 255, 0), " Score: %d ", Score);

    blit(buffer, screen, 0, 0, 0, 0, SCREEN_W, SCREEN_H);
  }

  destroy_bitmap(buffer);

  return 0;
}
END_OF_MAIN()
```