Snaaake

Snaaake is a take on the old nibbles game. Maneuver your snake around eating tokens, growing longer with every point you gain, your tail becoming the greatest threat to your own existence. However this one puts a different spin on it by having a small, boarder less playing field that wraps top to bottom and left to right and by keeping the view centered on the player. As you play you will quickly find the already claustrophobic space getting more cramped as you navigate the labyrinthine passages formed from your own tail.

Snaaake was written by Andreas Jörgensen

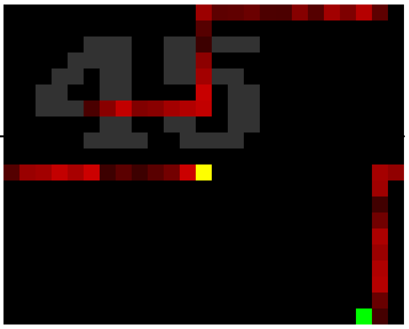
.

/* snake.h listing begins: */

```
#ifndef SNAKE_H
#define SNAKE_H
#include <allegro.h>
#include <string>
#include <stdio.h>

class Snake //The snake's head. I think most of this is fairly self-explanatory.
{
  int x, y, oldx, oldy;
  public:
  int score;
  Snake(); //CONSTRUCTAH
  void move();
  int getOldX();
  int getOldY();
  int getX();
  int getY();
  void reset();
  void setX(int X);
  void setY(int Y);
  void draw(BITMAP *destination);
  void check();
  std::string direction;
};

class SnakeTail
{
  int x, y;
  int oldx, oldy;
  bool active;
  public:
  SnakeTail();
  int getOldX();
  int getOldY();
  int getX();
  int getY();
  void setXY(int X, int Y);
  void draw(BITMAP *destination);
  bool getActive();
  void setActive(bool stat);
};
#endif
```



/* snake.cpp listing begins: */

```
#include "snake.h"

Snake::Snake()
{
  x = 10;
  y = 10;
  oldx = 9;
  oldy = 10;
  score = 0;
  direction = "right";
}

void Snake::move()
{
  if (key[KEY_LEFT] && (direction != "right")) direction = "left";
  else if (key[KEY_RIGHT] && (direction != "left")) direction = "right";
```

Listing continued on next page...

/* Listing continued from previous page */

```
  else if (key[KEY_UP] && (direction != "down")) direction = "up";
  else if (key[KEY_DOWN] && (direction != "up")) direction = "down";

  oldx = x;
  oldy = y;

  if (direction == "left") x--;
  else if (direction == "right") x++;
  else if (direction == "up") y--;
  else if (direction == "down") y++;
}

void Snake::draw(BITMAP *destination)
{
   putpixel(destination, x, y, makecol(255, 255, 0)); //Draw a YELLOW PIXEL.
}

//Most of this stuff is very self-explanatory. Have fun reading.
int Snake::getOldX() { return oldx; }
int Snake::getOldY() { return oldy; }
int Snake::getX() { return x; }
int Snake::getY() { return y; }
void Snake::setX(int X) { x = X; }
void Snake::setY(int Y) { y = Y; }

int SnakeTail::getX() { return x; }
int SnakeTail::getY() { return y; }

void SnakeTail::draw(BITMAP *destination)
{
  int randColor = (rand()%150) + 60;
  putpixel(destination, x, y, makecol(randColor, 0, 0));
}

bool SnakeTail::getActive() { return active; }
void SnakeTail::setActive(bool stat) { active = stat; }

void SnakeTail::setXY(int X, int Y)
{
  oldx = x;
  oldy = y;
  x = X;
  y = Y;
}

int SnakeTail::getOldX() { return oldx; }
int SnakeTail::getOldY() { return oldy; }

SnakeTail::SnakeTail()
{
  x = -1;
  y = -1;
}

void Snake::reset()
{
  direction = "right";
  score = 0;
  x = 10;
  y = 10;
}
```

/* main.cpp listing begins */

```
#include <allegro.h> //Graphics!
#include <stdio.h> //Sprintf! And rand()!
#include <ctime> //Random seed!
#include "snake.h" //Snake and SnakeTail classes!

#define SCR_H 20 //Resolutions!
#define SCR_W 25 //Resolutions again!

void init();
void deinit();

SnakeTail TailArray[256];
Snake Player;

BITMAP* BOARD = NULL; //Double BOARDing, ofc.
BITMAP* BUFFER = NULL;

int AppleX, AppleY;

int main() {
  init();
  std::string gamestate = "menu";
  std::string oldstate = "none";
  srand(time(0));

  AppleX = rand()%(SCR_W);
  AppleY = rand()%(SCR_H);

  char textbuf[50] = "HELLOOOOO"; //Temporary text FTW
  BOARD = create_bitmap(SCR_W, SCR_H); //Ahh, double BOARDing <3
  BUFFER = create_bitmap(SCR_W, SCR_H);
  std::string selectedItem = "S"; //For the menu!

  int selectCounter = 0;  //Counter for the menu options
  while (true) {

    if (gamestate == "game")
    {

      if (oldstate == "menu")
      {
        Player.reset();
        AppleX = rand()%(SCR_W);
        AppleY = rand()%(SCR_H);
        oldstate = "game"; //So we don't reset again and again...
        for (int j = 0; j < 256; ++j)
        {
          if (j > 2)
          {
            TailArray[j].setActive(false);
          }
          else
          {
            TailArray[j].setActive(true);
            TailArray[j].setXY(Player.getX(), Player.getY());
          }
        }
      }

      Player.move();
      TailArray[0].setXY(Player.getOldX(), Player.getOldY());

  if (Player.getX() > SCR_W - 1) Player.setX(0); //Wrap player around screen
```

/* Listing continued on next page… */

/* Listing continued from previous page */

```
  if (Player.getX() < 0 ) Player.setX(SCR_W - 1);
  if (Player.getY() > SCR_H - 1) Player.setY(0);
  if (Player.getY() < 0) Player.setY(SCR_H - 1);

  sprintf(textbuf, "%d", Player.score);
  textout_ex(BUFFER, font, textbuf, 2, 2, makecol(50, 50, 50), -1);

    if (key[KEY_ESC]) {gamestate = "menu";} //Go back to the menu

    for (int i = 0; i < 256; ++i)
    {
      if (TailArray[i].getActive())
      {

        if (i > 0)
        {
          TailArray[i].setXY(TailArray[i-1].getOldX(), TailArray[i-1].getOldY());
        }
        TailArray[i].draw(BOARD); //Now DRAW THEM.
      }
    }

    for (int i = 0; i < 256; ++i)
    {
      if ((Player.getX() == TailArray[i].getX())
        && (Player.getY() == TailArray[i].getY()) && TailArray[i].getActive())
      {
        gamestate = "lose"; //OH NOES, I HIT MYSELF
        break;
      }
    }

    if ((Player.getX() == AppleX) && (Player.getY() == AppleY))
    {
      AppleX = rand()%(SCR_W);
    AppleY = rand()%(SCR_H);
    Player.score++;
    for (int i = 0; i < 256; ++i)
    {
      if (!TailArray[i].getActive())
      {
        TailArray[i].setActive(true); //Grow by one tile-thing
        break;
      }
    }
    }

    putpixel(BOARD, AppleX, AppleY, makecol(0, 255, 0)); //Draw apple!
    Player.draw(BOARD);

    int sx = (SCR_W + Player.getX() - SCR_W/2) % SCR_W;
    int sy = (SCR_H + Player.getY() - SCR_H/2) % SCR_H;
    for (int xx = 0; xx < SCR_W; xx++)
      for (int yy = 0; yy < SCR_H; yy++) {
      int col = getpixel(BOARD, (xx + sx) % SCR_W, (yy + sy) % SCR_H);
      if (col) putpixel (BUFFER, xx, yy, col);
      }
    }

    else if (gamestate == "menu")
    {
      textout_ex(BUFFER, font, "S", 2, 2, makecol(255, 255, 255), -1);
      textout_ex(BUFFER, font, "Q", 2, 12, makecol(255, 255, 255), -1);
```

/* Listing continued on next page… */

/* Listing continued from previous page */

```
      if (selectedItem == "S")
        textout_ex(BUFFER, font, "<", 10, 2, makecol((rand()%50)+180, 0, 0),-1);
      if (selectedItem == "Q")
        textout_ex(BUFFER, font, "<", 10, 12, makecol((rand()%50)+180, 0, 0),-1);

      selectCounter++; //Just so the item selection won't flail around wildly
      if ((key[KEY_UP] || key[KEY_DOWN]) && selectedItem=="S" && selectCounter>2)
        { selectedItem = "Q"; selectCounter = 0; }
      else if ((key[KEY_UP] || key[KEY_DOWN]) && selectedItem=="Q"
        && selectCounter>2)
        { selectedItem = "S"; selectCounter = 0; }

      if (selectedItem == "S" && key[KEY_ENTER])
        { gamestate = "game"; oldstate = "menu"; }
      else if (selectedItem == "Q" && key[KEY_ENTER]) gamestate = "quit";
    }

    if (gamestate == "lose")
    {
      textout_ex(BUFFER, font, "scr", 1, 2, makecol(255, 255, 255), -1);
      textout_ex(BUFFER, font, textbuf, 1, 12, makecol(80, 80, 80), -1);

      if (key[KEY_ESC]) gamestate = "menu";
    }

    if (gamestate == "quit") { deinit(); return 0; }
    stretch_sprite(screen, BUFFER, 0, 0, SCR_W*8, SCR_H*8);
    clear_bitmap(BOARD);
    clear_bitmap(BUFFER);
    vsync(); //Vsync, no tearing!
    rest(50);
  }

  deinit();
  return 0;
}
END_OF_MAIN()

void init() {
  int depth, res;
  allegro_init();
  depth = desktop_color_depth();
  if (depth == 0) depth = 32;
  set_color_depth(depth);
  res = set_gfx_mode(GFX_AUTODETECT_WINDOWED, SCR_W*8, SCR_H*8, 0, 0);
  if (res != 0) {
    allegro_message(allegro_error);
    exit(-1);
  }

  install_timer();
  install_keyboard();
  install_mouse();
  set_window_title ("SNAAAKE");
}

void deinit() {
  clear_keybuf();
  destroy_bitmap(BOARD);
  /* add other deinitializations here */
}
```