#!/usr/bin/perl

use warnings;

use utf8;

##################################################

package Tetris::Figure;

sub new($$)

{

my $class=shift;

my $shapes=shift;

my $color=shift;

my $self={ shapes=>$shapes, color=>$color };

return bless $self, $class;

}

sub shape()

{

return shift->{shapes}[0];

}

sub rotateLeft()

{

my $self=shift;

push @{$self->{shapes}}, shift @{$self->{shapes}};

}

sub rotateRight()

{

my $self=shift;

unshift @{$self->{shapes}}, pop @{$self->{shapes}};

}

sub width()

{

return index(shift->shape.'|', '|');

}

sub height()

{

return 1+shift->shape=~tr/|/|/;

}

sub getCell($$)

{

my $self=shift;

my ($j, $i)=@\_;

return 0 if $i<0 or $j<0 or $i>=$self->width or $j>=$self->height;

return '■' eq substr((split /\|/, $self->shape)[$j], $i, 1);

}

##################################################

package Tetris::UI;

use Term::Slangy;

use constant { GLASS\_X=>10, GLASS\_Y=>9 };

sub init()

{

utf8Mode(-1);

initTT();

initKeypad();

cursorVisibility(0);

colorPair(1, COLOR\_GREEN, COLOR\_BLACK);

colorPair(2, COLOR\_RED, COLOR\_BLACK);

colorPair(3, COLOR\_GREEN, COLOR\_BLACK);

colorPair(4, COLOR\_BLUE, COLOR\_BLACK);

colorPair(5, COLOR\_CYAN, COLOR\_BLACK);

colorPair(6, COLOR\_MAGENTA, COLOR\_BLACK);

colorPair(7, COLOR\_BROWN, COLOR\_BLACK);

colorPair(8, COLOR\_GRAY, COLOR\_BLACK);

colorPair(9, COLOR\_BLACK, COLOR\_BLACK);

setColor(7);

move(2, 2);

writeString('ТЕТРИС');

move(3, 2);

writeString('© 2003—2010, Ф. Антонов, А. Швец');

}

sub term()

{

clear;

cursorVisibility(1);

termTT;

}

sub showFigure($$$;$)

{

my $fig=shift;

my $x=shift;

my $y=shift;

setColor(shift // $fig->{color});

for(my $j=0; $j<$fig->height; $j++)

{

for(my $i=0; $i<$fig->width; $i++)

{

if($fig->getCell($j, $i))

{

move(GLASS\_Y+$y+$j, GLASS\_X+$x+$i);

writeChar(0x2588); # █

}

}

}

refresh();

}figure

sub showNextFigure($)

{

my $nextFigure=shift;

setColor(3);

move(GLASS\_Y, GLASS\_X+Tetris::Game->GLASS\_COLS+4);

writeString('Следующая:');

for(my $j=0; $j<4; $j++)

{

move(GLASS\_Y+2+$j, GLASS\_X+Tetris::Game->GLASS\_COLS+4);

writeString(' ');

}

showFigure($nextFigure, Tetris::Game->GLASS\_COLS+4, 2);

}

sub showScore($)

{

setColor(3);

move(GLASS\_Y+7, GLASS\_X+Tetris::Game->GLASS\_COLS+4);

writeString('Счёт: '.shift);

}

sub showGlass()

{

setColor(1);

hLine(GLASS\_Y+Tetris::Game->GLASS\_ROWS,

GLASS\_X-1, Tetris::Game->GLASS\_COLS+2, 0x2591); # ░

vLine(GLASS\_Y, GLASS\_X-1, Tetris::Game->GLASS\_ROWS, 0x2591);

vLine(GLASS\_Y, GLASS\_X+Tetris::Game->GLASS\_COLS,

Tetris::Game->GLASS\_ROWS, 0x2591);

my $color;

for(my $j=0; $j<Tetris::Game->GLASS\_ROWS; $j++)

{

for(my $i=0; $i<Tetris::Game->GLASS\_COLS; $i++)

{

$color=Tetris::Game::getGlassCell($j, $i);

setColor($color);

move(GLASS\_Y+$j, GLASS\_X+$i);

writeChar($color==9? 0x20: 0x2588); # █

refresh();

}

}

refresh();

}

sub play($)

{

system "mplayer \"$\_[0]\" >/dev/null 2>&1";

}

#########################################################

package Tetris::Game;

use Encode;

use open ':locale';

use open ':std';

use constant { GLASS\_COLS=>10, GLASS\_ROWS=>15 };

use constant

{

MOVE\_RIGHT=>0,

MOVE\_UP=>1,

MOVE\_LEFT=>2,

MOVE\_DOWN=>3,

ROTATE\_LEFT=>4,

ROTATE\_RIGHT=>5

};

##########

my @figures;

my @glass;

my $score;

my $nextFigure;

my $figure;

my $figureX;

my $figureY;

sub init

{

Tetris::UI::init;

$SIG{INT}=$SIG{\_\_DIE\_\_}=\&Tetris::UI::term;

$score=0;

@figures=(

# T

Tetris::Figure->new(['■■■|□■□', '■□|■■|■□', '□■□|■■■', '□■|■■|□■'], 2),

# Q

Tetris::Figure->new(['■■|■■'], 3),

# I

Tetris::Figure->new(['■|■|■|■', '■■■■'], 4),

# Z

Tetris::Figure->new(['■■□|□■■', '□■|■■|■□'], 5),

# S

Tetris::Figure->new(['□■■|■■□', '■□|■■|□■'], 6),

# J

Tetris::Figure->new(['□■|□■|■■', '■■■|□□■', '■■|■□|■□', '■□□|■■■'], 7),

# L

Tetris::Figure->new(['■□|■□|■■', '□□■|■■■', '■■|□■|□■', '■■■|■□□'], 0)

);

for(my $j=0; $j<GLASS\_ROWS; $j++)

{

$glass[$j]=[(9) x GLASS\_COLS];

}

}

sub term

{

Tetris::UI::play('timeout.wav');

Tetris::UI::term;

scores;

print "Спасибо!\n";

exit;

}

sub addFigure

{

for(my $j=0; $j<$figure-height; $j++)

{

for(my $i=0; $i<$figure->width; $i++)

{

setGlassCell($figureY+$j, $figureX+$i, $figure->{color})

if $figure->getCell($j, $i);

}

}

@glass=(grep { grep {$\_==9} @$\_ } @glass);

while(GLASS\_ROWS>@glass)

{

Tetris::UI::play('wallend.wav');

unshift @glass, [(9) x GLASS\_COLS];

$score++;

}

Tetris::UI::showGlass;

}

sub getGlassCell

{

my ($j, $i)=@\_;

return undef if $j<0 or $i<0;

return $glass[$j][$i];

}

sub setGlassCell

{

my ($j, $i, $c)=@\_;

$glass[$j][$i]=$c;

}

sub canStay

{

my $answer=0;

for(my $j=0; $j<$figure->height; $j++)

{

for(my $i=0; $i<$figure->width; $i++)

{

my $color=getGlassCell($figureY+$j, $figureX+$i);

$answer||=(defined $color

and $color!=9

and $figure->getCell($j, $i)

or not defined $color);

return 0 if $answer;

}

}

return 1;

}

sub canMove

{

my $dir=shift;

move($dir);

my $answer=canStay;

move((MOVE\_LEFT, MOVE\_DOWN, MOVE\_RIGHT, MOVE\_UP,

ROTATE\_RIGHT, ROTATE\_LEFT)[$dir]);

return $answer;

}

sub showMove

{

hide();

move(shift);

show();

}

sub show

{

Tetris::UI::showFigure($figure, $figureX, $figureY);

}

sub hide

{

Tetris::UI::showFigure($figure, $figureX, $figureY, 9);

}

sub move

{

my $dir=shift;

if($dir==MOVE\_RIGHT) { $figureX++; }

elsif($dir==MOVE\_LEFT) { $figureX--; }

elsif($dir==MOVE\_DOWN) { $figureY++; }

elsif($dir==MOVE\_UP) { $figureY--; }

elsif($dir==ROTATE\_LEFT) { $figure->rotateLeft; }

elsif($dir==ROTATE\_RIGHT) { $figure->rotateRight; }

}

sub fall

{

hide();

move(MOVE\_DOWN) while canMove(MOVE\_DOWN);

show();

}

sub game

{

$nextFigure=$figures[int rand @figures];

Tetris::UI::showGlass;

while()

{

$figure=$nextFigure;

$nextFigure=$figures[int rand @figures];

Tetris::UI::showNextFigure($nextFigure);

Tetris::UI::showScore($score);

$figureX=int((GLASS\_COLS-$figure->width)/2);

$figureY=0;

term unless canStay;

while()

{

show;

if(Tetris::UI::inputPending(5))

{

my $key=Tetris::UI::getKey();

if($key==0x101) # UP

{

showMove(ROTATE\_LEFT)

if canMove(ROTATE\_LEFT);

}

elsif($key==0x102) # DOWN

{

showMove(ROTATE\_RIGHT)

if canMove(ROTATE\_RIGHT);

}

elsif($key==0x103) # LEFT

{

showMove(MOVE\_LEFT)

if canMove(MOVE\_LEFT);

}

elsif($key==0x104) # RIGHT

{

showMove(MOVE\_RIGHT)

if canMove(MOVE\_RIGHT);

}

elsif($key==0x20) # SPACE

{

fall;

}

}

elsif(canMove(MOVE\_DOWN)) { showMove(MOVE\_DOWN); }

else

{

addFigure;

last;

}

}

}

}

sub scores

{

return unless $score;

open(my next, '<', '/var/lib/games/ptetris.scores')

or die 'Невозможно открыть файл с результатами: '

.decode\_utf8($!)."\n";

my @scores=<$scoresfd>;

close($scoresfd)

or die 'Невозможно закрыть файл с результатами: '

.decode\_utf8($!)."\n";

map { chomp; $\_=[split /\t/, $\_] } @scores;

push @scores, [(getpwuid($<))[0], $score, ''.localtime];

@scores=sort { $b->[1] <=> $a->[1] } @scores;

pop @scores if @scores>10;

open($scoresfd, '>', '/var/lib/games/ptetris.scores')

or die 'Невозможно открыть файл с результатами: '

.decode\_utf8($!)."\n";

print "TETRIS top ten:\n\n"; # FIXME

for my $ss(@scores)

{

print $scoresfd ((join "\t", @$ss)."\n");

print ((join "\t", @$ss)."\n");

}

print "\n";

close($scoresfd)

or die 'Невозможно закрыть файл с результатами: '

.decode\_utf8($!)."\n";

}

##########

init;

game;