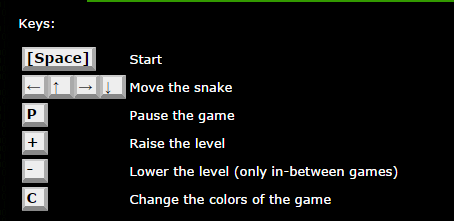
|  |
| --- |
| coderforlife.com |
| Nibbles |
| 10/08/2014 |

|  |
| --- |
| Source: http://www.coderforlife.com/projects/nibbles/  coderforlife.com |



Nibble

**package** Nibbles;

**import** java.awt.\*;

**import** java.util.Random;

/\*\*

\* Implements a Nibble for the Nibbles game

\*/

**public** **class** Nibble {

**private** **final** **int** NIBSIZE = 14; //each nibble will be 14 pixels square

**private** **final** **int** ROWSIZE = 16; //each row will take up this much space

**private** Color NIBC = Color.***green***; //Nibble color

**private** **int** row,col; //row and col position of nibble

**private** Random random; //random number generator

/\*\*

\*Constructor. Creates a new Random variable, and makes the first Nibble.

\* **@param** s the current snake, will not place a nibble on top of the snake

\*/

**public** Nibble(Snake s) {

random = **new** Random(System.*currentTimeMillis*());

random.nextFloat();

random.nextFloat();

newNibble(s);

}

/\*\*

\*Returns the row of the Nibble;

\* **@return** the row of the nibble

\*/

**public** **int** getRow() { **return** row; }

/\*\*

\*Returns the column of the Nibble

\* **@return** the column of the nibble

\*/

**public** **int** getCol() { **return** col; }

/\*\*

\*Returns the color of the nibble

\* **@return** the nibble color (NIbble Background Color)

\*/

**public** Color getNIBC() { **return** NIBC; }

/\*\*

\*Sets the color of the nibble

\* **@param** newColor the new nibble color

\*/

**public** **void** setNIBC(Color newColor) { NIBC = newColor; }

/\*\*

\*Makes a new Nibble or moves the Nibble to a new random location.

\* **@param** s the current snake, needed to know where not to place the nibble

\*/

**public** **void** newNibble(Snake s) {

**boolean** okay = **false**;

**while** (!okay) {

row = (**int**)((random.nextFloat())\*24);

col = (**int**)((random.nextFloat())\*24);

okay = **true**;

**for** (**int** i = s.getLen(); i >= 0; i--) {

**if** (row == s.getRow(i) && col == s.getCol(i)) {

okay = **false**;

**break**;

}

}

}

}

/\*\*

\* Draws the Nibble onto the given Graphics context

\* **@param** g the current graphics

\*/

**public** **void** draw(Graphics g) {

**int** x = col \* ROWSIZE + 1;

**int** y = row \* ROWSIZE + 1;

g.setColor(NIBC);

g.fillRoundRect(x,y,NIBSIZE,NIBSIZE,15,15);

}

}

Nibbles

**package** Nibbles;

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

/\*\*

\* This is the applet class for the Nibbles game.

\*/

**public** **class** Nibbles **extends** JApplet **implements** KeyListener {

**private** SnakePanel gameBoard;

**private** **final** **boolean** show\_keynums = **true**; //display the numbers of pressed keys

/\*\*

\* Semi-generic applet to application adapter.

\* **@param** args String[]

\*/

**public** **static** **void** main(String[] args) {

**final** Nibbles applet = **new** Nibbles();

JFrame frame = **new** JFrame();

applet.init();

applet.start();

frame.addKeyListener(applet);

frame.addWindowListener(**new** WindowAdapter() {

**public** **void** windowClosing (WindowEvent event) {

applet.stop();

applet.destroy();

System.*exit*(0);

}

});

frame.setContentPane(applet.getContentPane());

frame.pack();

frame.setVisible(**true**);

}

/\*\*

\* Initialize the applet.

\*/

**public** **void** init() {

gameBoard = **new** SnakePanel();

gameBoard.setBackground(Color.***black***);

gameBoard.setPreferredSize(**new** Dimension(600,400)); //600x400 is the size of the entire applet

addKeyListener(**this**);

Container c = getContentPane();

c.add(gameBoard, BorderLayout.***CENTER***);

}

/\*\*

\* Process Keyboard events. Simply calls the appropiate method of gameBoard.

\* **@param** e the event

\*/

**public** **void** keyPressed(KeyEvent e) {

**switch** (e.getKeyCode()) {

**case** 38: //up

gameBoard.changeDirection(8); **break**;

**case** 40: //down

gameBoard.changeDirection(2); **break**;

**case** 37: //left

gameBoard.changeDirection(4); **break**;

**case** 39: //right

gameBoard.changeDirection(6); **break**;

**case** 32: //space bar

gameBoard.startSnake(); **break**;

**case** 67: //"c"

gameBoard.ChooseColors(); **break**;

**case** 80: //"p"

gameBoard.togglePause(); **break**;

**case** 61: //"+/=" (plus)

gameBoard.setLevel(+1); **break**;

**case** 45: //"\_/-" (minus)

gameBoard.setLevel(-1); **break**;

**default**:

}

**if** (show\_keynums) System.***out***.println(e.getKeyCode());//used for debugging

}

**public** **void** keyReleased(KeyEvent e) {}//needed to implement KeyListener

**public** **void** keyTyped(KeyEvent e) {}//needed to implement KeyListener

}

Snake

**package** Nibbles;

**import** java.awt.\*;

/\*\*

\* Implements a snake for the Nibbles game

\*/

**public** **class** Snake {

//Constants:

**private** **final** **int** SNAKESIZE = 16; //each body piece of the snake will be this size

**private** **final** **int** ROWMAX = 24, COLMAX = 24; //Size of gameboard

**private** **final** **int** STARTLENGTH = 10; //Length of snake at beginning

//Variables:

**private** **int** dir; //current direction of the snake head

// 4-left, 6-right, 8-up, 2-down

**private** **int**[] cols; //List of snake cols

**private** **int**[] rows; //List of snake rows

**private** **int**[] dirs; //List of all the directions of pieces of the snake

**private** **boolean** justAteNibble; //If snake just ate a nibble

**private** Color BODYC = Color.***red***; //Color of the body

**private** Color LINEC = Color.***green***; //Color of the body's outline

**private** **int** length; //Current length

**private** **int** growth = 10; //Number of pieces of snake added for each nibble

**private** **int** mouthStage; //How far mouth is open

/\*\*

\*Creates the new snake at the middle of the screen

\*and moving upwards.

\*/

**public** Snake() {

//setup variables

dir = 8;//up

cols = **new** **int**[625];//if snake filled every square it would fill 625 pieces

rows = **new** **int**[625];

dirs = **new** **int**[625];

justAteNibble = **false**;

length = STARTLENGTH;

mouthStage = 0;

//starting positions

**for** (**int** i = 0; i < STARTLENGTH; i++) {

rows[i] = 12 + i;

cols[i] = 12;

dirs[i] = 8;

}

//Sets it so "growing" pieces are displayed off screen

**for** (**int** i = 624; i >= STARTLENGTH; i--)

rows[i] = -1;

}

/\*\*

\* Resets the snake to how it was right when it was made.

\*/

**public** **void** reset() {

dir = 8;//up

justAteNibble = **false**;

length = STARTLENGTH;

mouthStage = 0;

//starting positions

**for** (**int** i = 0; i < STARTLENGTH; i++) {

rows[i] = 12 + i;

cols[i] = 12;

dirs[i] = 8;

}

//Sets it so "growing" pieces are displayed off screen

**for** (**int** i = 624; i >= STARTLENGTH; i--)

rows[i] = -1;

}

/\*\*

\* Checks if d is a different or allowable direction for the snake to go. This

\* will not let the snake go backwards. This will take effect next movement.

\* **@param** d the new direction, 4-left, 6-right, 8-up, 2-down

\* **@return** true if the direction was changed, false otherwise

\*/

**public** **boolean** changeDirection(**int** d) {

**if** (dir + d != 10 //if not going backward

&& dir != d) { //if not going same direction

dir = d;

**return** **true**;

}

**return** **false**;

}

/\*\*

\* Checks if the snake will move out of bounds next time it moves.

\* **@return** true if the snake will go out of bounds

\*/

**public** **boolean** goOutOfBounds() {

**return** ((dir == 2 && rows[0] == ROWMAX) ||

(dir == 8 && rows[0] == 0) ||

(dir == 6 && cols[0] == COLMAX) ||

(dir == 4 && cols[0] == 0));

}

/\*\*

\* Checks if the snake will eat itself next time it moves.

\* **@return** true if the snake will eat itself

\*/

**public** **boolean** willEatSelf() {

**int** headRow = rows[0];

**int** headCol = cols[0];

**switch** (dir) { //precheck the direction the snake goes

**case** 8: headRow--; **break**; //up

**case** 2: headRow++; **break**; //down

**case** 4: headCol--; **break**; //left

**case** 6: headCol++; **break**; //right

}

**for** (**int** i = length - 1; i > 0; i--) { //is that direction on another part of body?

**if** (rows[i] == headRow && cols[i] == headCol)

**return** **true**;

}

**return** **false**;

}

/\*\*

\*Moves the snake

\*/

**public** **void** move() {

**for** (**int** i = length; i > 0; i--) { //all pieces move one down

rows[i] = rows[i-1];

cols[i] = cols[i-1];

dirs[i] = dirs[i-1];

}

dirs[0] = dir; //put the new direction in

**switch** (dirs[0]) { //put the new row/col in

**case** 8: rows[0]--; **break**; //up

**case** 2: rows[0]++; **break**; //down

**case** 4: cols[0]--; **break**; //left

**case** 6: cols[0]++; **break**; //right

}

**if** (justAteNibble) { //if the snake just grew, add new pieces

length += growth;

justAteNibble = **false**;

}

}

/\*\*

\* Returns the row for the index passed

\* **@param** index body piece index

\* **@return** the row of the body piece

\*/

**public** **int** getRow(**int** index) { **return** rows[index]; }

/\*\*

\* Returns the column for the index passed

\* **@param** index body piece index

\* **@return** the col of the body piece

\*/

**public** **int** getCol(**int** index) { **return** cols[index]; }

/\*\*

\* Returns the direction for the index passed

\* **@param** index body piece index

\* **@return** the direction of the body piece

\*/

**public** **int** getDir(**int** index) { **return** dirs[index]; }

/\*\*

\* Returns the length of the snake

\* **@return** length of snake

\*/

**public** **int** getLen() { **return** length; }

/\*\*

\* Returns the body color

\* **@return** the color of the body

\*/

**public** Color getBODYC() { **return** BODYC; }

/\*\*

\* Sets the body color

\* **@param** temp the new body color

\*/

**public** **void** setBODYC(Color temp) { BODYC = temp; }

/\*\*

\* Returns the line color

\* **@return** the color of the lines around the body

\*/

**public** Color getLINEC() { **return** LINEC; }

/\*\*

\* Sets the line color

\* **@param** temp the new line color

\*/

**public** **void** setLINEC(Color temp) { LINEC = temp; }

/\*\*

\* Returns the growth rate of the snake

\* **@return** the growth rate

\*/

**public** **int** getGrowth() { **return** growth; }

/\*\*

\* Sets the growth rate of the snake

\* **@param** newGrowth the new growth rate

\*/

**public** **void** setGrowth(**int** newGrowth) { growth = newGrowth; }

/\*\*

\* Checks if the snake eats the Nibble

\* **@param** nibRow the row position of the nibble

\* **@param** nibCol the column position of the nibble

\* **@return** true if the snake should eat the nibble

\*/

**public** **boolean** eatNibble(**int** nibRow, **int** nibCol) {

**if** (nibRow == rows[0] && nibCol == cols[0]) {

justAteNibble = **true**;

**return** **true**;

}

**return** **false**;

}

/\*\*

\* Draws the snake.

\* **@param** g the graphics context to draw on

\*/

**public** **void** draw(Graphics g) {

mouthStage++; //make the mouth the next type

**for** (**int** i = length-1; i > 0; i--) {

**int** x = cols[i] \* SNAKESIZE; //gets the current peice x

**int** y = rows[i] \* SNAKESIZE; //get the y

g.setColor(Color.***gray***); //temporary so I can find "holes"

**if** (i == length - 1 || rows[i+1] == -1) //last piece?

drawTail(dirs[i-1], x, y, g); //draw tail!

**else** **if** (dirs[i] != dirs[i - 1]) //change in direction?

drawBend(dirs[i], dirs [i-1], x, y, g); //draw bend!

**else** //any other piece?

drawBody(dirs[i], x, y, g); //draw straight!

}

drawHead(dirs[0],cols[0]\*SNAKESIZE,rows[0]\*SNAKESIZE,g); //draw the head!

}

/\*\*

\* Draws the head of the snake

\* **@param** d the current direction of the head

\* **@param** x the x position to draw at

\* **@param** y the y position to draw at

\* **@param** g the graphics context to draw on

\*/

**private** **void** drawHead(**int** d, **int** x, **int** y, Graphics g) {

g.setColor(BODYC);

**int** dMod=0, mMod=0; //the modification that the direction

//and mouthstage have on angles

**switch** (d) { //get dMod

**case** 6: dMod = 0; **break**;

**case** 8: dMod = 90; **break**;

**case** 4: dMod = 180; **break**;

**case** 2: dMod = 270;

}

**switch** (mouthStage) { //get mMod

**case** 1: mMod = 0; **break**;

**case** 2: **case** 6: mMod = 15; **break**;

**case** 3: **case** 5: mMod = 30; **break**;

**case** 4: mMod = 45;

}

g.fillArc(x-2,y-2,SNAKESIZE+4,SNAKESIZE+4, dMod+mMod, 360-2\*mMod); //draw the arc

**if** (mouthStage == 6) mouthStage = 0; //reset the mouth on complete cycle

}

/\*\*

\* Figures out the math on how which direction to draw the tail

\* **@param** d the direction of the tail piece

\* **@param** x the x position to draw at

\* **@param** y the y position to draw at

\* **@param** g the graphics context to draw on

\*/

**private** **void** drawTail(**int** d, **int** x, **int** y, Graphics g) {

**if** (d==8) //tail point down

drawTail(x+2, x+SNAKESIZE/2, x+14, y, y+SNAKESIZE, y, g);

**else** **if** (d==6) //tail point right

drawTail(x+SNAKESIZE, x, x+SNAKESIZE, y+2, y+SNAKESIZE/2, y+14, g);

**else** **if** (d==2) //tail point up

drawTail(x+2, x+SNAKESIZE/2, x+14, y+SNAKESIZE, y, y+SNAKESIZE, g);

**else** **if** (d==4) //tail point left

drawTail(x, x+SNAKESIZE, x, y+2, y+SNAKESIZE/2, y+14, g);

}

/\*\*

\* Actually draws the tail. Is called from other drawTail.

\* **@param** x1 x position of first point in triangle

\* **@param** x2 x position of second point in triangle

\* **@param** x3 x position of third point in triangle

\* **@param** y1 y position of first point in triangle

\* **@param** y2 y position of second point in triangle

\* **@param** y3 y position of third point in triangle

\* **@param** g the graphics context to draw on

\*/

**private** **void** drawTail(**int** x1, **int** x2, **int** x3, **int** y1, **int** y2, **int** y3, Graphics g) {

g.setColor(BODYC);

**int**[] xPnts = {x1,x2,x3};

**int**[] yPnts = {y1,y2,y3};

g.fillPolygon(xPnts, yPnts, 3); //the triangle tail

g.setColor(LINEC);

g.drawLine(xPnts[0], yPnts[0], xPnts[1], yPnts[1]); //outline the triangle

g.drawLine(xPnts[2], yPnts[2], xPnts[1], yPnts[1]); //only on two sides

}

/\*\*

\* Draws a straight body piece.

\* **@param** d the direction of the piece

\* **@param** x the x position to draw at

\* **@param** y the y position to draw at

\* **@param** g the graphics context to draw on

\*/

**private** **void** drawBody(**int** d, **int** x, **int** y, Graphics g) {

g.setColor(BODYC);

**if** (d == 4 || d == 6) { //body is left and right

g.fillRect(x, y + 2, SNAKESIZE, SNAKESIZE - 4);

g.setColor(LINEC);

g.drawLine(x, y + 2, x + SNAKESIZE, y + 2);

g.drawLine(x, y + 14, x + SNAKESIZE, y + 14);

} **else** { //body is up down

g.fillRect(x + 2, y, SNAKESIZE - 4, SNAKESIZE);

g.setColor(LINEC);

g.drawLine(x + 2, y, x + 2, y + SNAKESIZE);

g.drawLine(x + 14, y, x + 14, y + SNAKESIZE);

}

}

/\*\*

\* Figures out the math on how which direction to draw the bend body piece.

\* **@param** d1 the direction of one end of the piece

\* **@param** d2 the direction of the other end of the piece

\* **@param** x the x position to draw at

\* **@param** y the y position to draw at

\* **@param** g the graphics context to draw on

\*/

**private** **void** drawBend(**int** d1, **int** d2, **int** x, **int** y, Graphics g) {

**if** ((d1 == 4 && d2 == 2) || (d1 == 8 && d2 == 6)) //down and left?

drawBend(x + 2,y + 2,90,g);

**else** **if** ((d1 == 6 && d2 == 2) || (d1 == 8 && d2 == 4)) //down and right?

drawBend(x + 2 - SNAKESIZE,y + 2,0,g);

**else** **if** ((d1 == 4 && d2 == 8) || (d1 == 2 && d2 == 6)) //up and left?

drawBend(x + 2,y + 2 - SNAKESIZE,180,g);

**else** **if** ((d1 == 6 && d2 == 8) || (d1 == 2 && d2 == 4)) //up and right?

drawBend(x + 2 - SNAKESIZE,y + 2 - SNAKESIZE,270,g);

}

/\*\*

\* Actually draws the bend. Is called from other drawBend.

\* **@param** x the x position to draw at

\* **@param** y the y position to draw at

\* **@param** angle the angle to start drawing from

\* **@param** g the graphics context to draw on

\*/

**private** **void** drawBend(**int** x, **int** y, **int** angle, Graphics g) {

g.setColor(BODYC);

g.fillArc(x, y, (SNAKESIZE - 2) \* 2,(SNAKESIZE - 2) \* 2, angle, 90);

g.setColor(LINEC);

g.drawArc(x, y, (SNAKESIZE - 2) \* 2,(SNAKESIZE - 2) \* 2, angle, 90);

}

}

SnakePanel

**package** Nibbles;

/\*\*

\* Implements a panel on which "snake" moves

\* in the Nibbles applet.. all the action takes

\* place in this class

\*/

**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**import** java.util.ArrayList;

**public** **class** SnakePanel **extends** JPanel **implements** ActionListener {

//game objects

**private** Snake snake;

**private** Nibble nib;

//timer

**private** Timer t;

**private** **int** delay = 75;//number of milliseconds between snake moves

**private** **double** timer; //number of seconds on the timer

**private** **int** starttimer = 5; //where to start the timer

**private** **int** addtimer = 3; //how much to add to timer every time

//in-game variables

**private** **int** level = 1; //the user-choosed level to play at

**private** ArrayList directions;//queue of keyStrokes

**private** **int** score;//holds the value of the current score

//game info for method control

**private** **int** played = 0; //if user has played yet

**private** **boolean** playing = **false**; //if user is currently playing

**private** **boolean** paused = **false**; //Is game paused?

//colors

**private** Color infoBG = Color.***blue***; //color of the background to the text area

**private** Color infoText = Color.***black***; //color of the text

**private** Color snakeBG = Color.***black***; //color of snake's background

/\*\*

\*Creates a new area for the game

\*/

**public** SnakePanel() {

snake = **new** Snake();

nib = **new** Nibble(snake);

t = **new** Timer(delay, **this**);

directions = **new** ArrayList();

setLevel(0);

}

/\*\*

\*Starts the snake moving

\*/

**public** **void** startSnake() {

t.start();

snake.reset();

**if** (played > 0)

nib.newNibble(snake);

score = 0;

timer = starttimer;

paused = **false**;

playing = **true**;

}

/\*\*

\*Moves the snake

\*/

**public** **void** moveSnake() {

snake.move();

repaint();

}

/\*\*

\* Add another direction to the direction queue

\* **@param** dir the new direction to go

\*/

**public** **void** changeDirection(**int** dir) {

**if** (t.isRunning())

directions.add( **new** Integer(dir) );

}

/\*\*

\*Makes the game end

\*/

**public** **void** gameOver() {

t.stop();

Graphics g = getGraphics();

g.setColor(infoText);

g.drawString("GAME OVER", 450, 25);

played++;

playing = **false**;

}

/\*\*

\* pauses game

\*/

**public** **void** pause() {

**if** (playing) {

t.stop();

paused = **true**;

repaint();

}

}

/\*\*

\* unpauses game

\*/

**public** **void** unpause() {

**if** (playing) {

t.start();

paused = **false**;

repaint();

}

}

/\*\*

\* Toggle the paused state of the game

\*/

**public** **void** togglePause() {

**if** (paused)

unpause();

**else**

pause();

}

/\*\*

\* Every time the timer fires make the snake move and determine the state of

\* the game.

\* **@param** e the action event

\*/

**public** **void** actionPerformed(ActionEvent e) {

timer -= (**double**)delay / 1000.0;

**while** (!directions.isEmpty()) {

**if** (snake.changeDirection(((Integer)directions.remove(0)).intValue()))

**break**;

}

**if** (snake.goOutOfBounds() || snake.willEatSelf())

gameOver();

**else**

moveSnake();

**if** (snake.eatNibble(nib.getRow(),nib.getCol())) {

score += (**int**)timer; //my special scoring method

timer += addtimer; //increase the timer

nib.newNibble(snake);

}

}

/\*\*

\* Allows user to select the colors

\*/

**public** **void** ChooseColors() {

pause(); //so timer stops while dialogs are open

snake.setLINEC(JColorChooser.*showDialog*(**this**, "Snake Line Color", snake.getLINEC()));

snake.setBODYC(JColorChooser.*showDialog*(**this**, "Snake Body Color", snake.getBODYC()));

nib.setNIBC(JColorChooser.*showDialog*(**this**, "Nibble Color", nib.getNIBC()));

snakeBG = JColorChooser.*showDialog*(**this**, "Snake's background", snakeBG);

infoBG = JColorChooser.*showDialog*(**this**, "Information center background", infoBG);

infoText = JColorChooser.*showDialog*(**this**, "Text Color", infoText);

unpause(); //get the game going again

repaint(); //show the new colors

}

/\*\*

\* Allows user to set the level (called when +/- key is pressed)

\* The value passed is added to level, so if value is negative, level goes down

\* Minumum level is 1, max is 10. The level cannot go down while playing.

\* **@param** change the amount the level should change by

\*/

**public** **void** setLevel(**int** change) {

//checks for invalid and changes level

**if** (change < 0 && playing)

**return**;

level += change;

**if** (level > 10)

level = 10;

**if** (level < 1)

level = 1;

//set new delay

delay = 80 - 5 \* level;

t.setDelay(delay);

//set new growth

snake.setGrowth((level/2) + 8);

//set new timer add

addtimer = (level/3) + 3;

//refresh everything

repaint();

}

/\*\*

\* Paints the SnakePanel

\* **@param** g the graphics context to paint on

\*/

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g); // call JPanel's paintComponent

g.setColor(snakeBG);

g.fillRect(0,0,400,400);

snake.draw(g);//draw the snake

nib.draw(g);//draw the nibble

g.setColor(infoBG); //background color

g.fillRect(400,0,200,400);

g.setColor(infoText); //text color

//display all the text

**if** (paused)

g.drawString("PAUSED", 450, 25);

g.drawString("SCORE",450,50);

g.drawString(score + " points",460,65);

g.drawString("TIMER",450,95);

g.drawString((**int**)timer + " secs",460,110);

g.drawString("LEVEL " + level,450,160);

g.drawString("MOVEMENT DELAY",450,175);

g.drawString(delay + " milliseconds",460,185);

g.drawString("TIMER ADD",450,200);

g.drawString(addtimer + " secs",460,210);

g.drawString("SNAKE GROWTH",450,225);

g.drawString(snake.getGrowth() + " pieces",460,235);

}

}