**AMRITA SCHOOL OF ENGINEERING**



**21AIE112**

**ELEMENTS OF COMPUTING SYSTEM-II**

**BRANCH: CSE-AIE**

**SEMESTER-2**

****

**UNDER THE GUIDANCE OF**: Ms. Meena Belwal

TEAM MEMBERS:

M Ramya Sree – BL.EN.U4AIE21072

Mettu Siddhartha- BL.EN.U4AIE21080

Poli Vamsi Vardhan Reddy – BL.EN.U4AIE21101

**INDEX**

1. Abstract

2. Introduction

3. Game Operation

4. Code

5. Output

6. Future Scope

7. Conclusion

Abstract

Jack bird is a game in which a player controls a bird’s flight height to avoid obstacles. Pressing on the screen for a longer period of time allows the bird to fly higher, while letting go causes the bird to fly lower. This project will bring the game to life using a video camera to detect a player’s motion, and controls the bird based on the speed at which a player flaps her arms.

Introduction

Playing games is fun and exciting. It gives us relief from stress and unwinds from our stressful work. Many of us spend our free time with others that use most of our time playing and exploring new games. Today with the rapid development of technology we have a implemented a game using jack language.

Jack Bird is an interesting single player game where the player controls a bird, attempting to fly between the columns without hitting them. If the bird touches the columns, top and bottom part of screen game ends and player loses the game. In our Implementation we intend to make this a single player game where, the bird is controlled with the help of the key binds.

Game Operation

* Press ‘Spacebar’ key to start the game
* Use ‘Spacebar’ to control the bird: If you press spacebar then bird goes up or else it falls down.
* Use ‘Q’ key to quit the game.

Classes we used in our project are:

**FlappyBird**- Models the FlappyBird object and creates/updates the bird image.

**FlappyBirdGame-** Models the Flappy Bird game at a high level by creating in-game objects, handling user inputs, and moving through game states.

**Main-** It intialises the game and start running it

**MyMath-** Modulus operator function.

**Pipe-** Models an individual Pipe object and creates/updates the pipe image.

**ScoreBoard-** Displays scoreboard with designated score.

Code

Jack Bird:

class FlappyBird {

/\*

Models the FlappyBird object and creates/updates the bird image.

\*/

field int y\_offset; // Y-coordinate on screen

field int x\_offset; // X-coordinate on screen

field int size\_x; // Width of bird image.

field int size\_y; // Height of bird image.

field int wing\_state; // Which wing image is currently displayed.

field Array up\_state; // Image coords for wing `up` state.

field Array mid\_state; // Image coords for wing `middle` state.

field Array down\_state; // Image coords for wing `down` state.

field Array bird\_vector; // Image coords for current wing state.

/\*

FlappyBird object constructor.

int altitude Y coordinate to first show bird image.

\*/

constructor FlappyBird new(int altitude) {

let y\_offset = altitude;

let x\_offset = 160;

let size\_x = 16;

let size\_y = 12;

let wing\_state = 1;

do buildUpArray();

do buildMidArray();

do buildDownArray();

do draw();

return this;

}

/\*

Move bird image coordinates up.

\*/

method void moveUp()

{

do erase();

if (y\_offset > 30) {

let y\_offset = y\_offset - 30;

} else {

let y\_offset = 0;

}

return;

}

/\*

Move bird image coordinates down.

\*/

method void moveDown()

{

do erase();

if (y\_offset < (255 - size\_y - 5)) {

let y\_offset = y\_offset + 5;

} else {

let y\_offset = 255 - size\_y;

}

return;

}

/\*

Erase bird image.

\*/

method void erase()

{

do Screen.setColor(false);

do Screen.drawRectangle(x\_offset, y\_offset, x\_offset + size\_x, y\_offset + size\_y);

return;

}

/\*

Build Array of wing 'up' state image coordinates.

\*/

method void buildUpArray() {

let up\_state = Array.new(12);

let up\_state[0] = 4032;

let up\_state[1] = 4656;

let up\_state[2] = 8456;

let up\_state[3] = 20766;

let up\_state[4] = 20769;

let up\_state[5] = 16961;

let up\_state[6] = -959;

let up\_state[7] = -32222;

let up\_state[8] = -740;

let up\_state[9] = -32252;

let up\_state[10] = 31768;

let up\_state[11] = 992;

return;

}

/\*

Build Array of wing 'middle' state image coordinates.

\*/

method void buildMidArray() {

let mid\_state = Array.new(12);

let mid\_state[0] = 4032;

let mid\_state[1] = 4656;

let mid\_state[2] = 8456;

let mid\_state[3] = 20740;

let mid\_state[4] = 20740;

let mid\_state[5] = 16958;

let mid\_state[6] = -959;

let mid\_state[7] = -32191;

let mid\_state[8] = -706;

let mid\_state[9] = -32252;

let mid\_state[10] = 31768;

let mid\_state[11] = 992;

return;

}

/\*

Build Array of wing 'down' state image coordinates.

\*/

method void buildDownArray() {

let down\_state = Array.new(12);

let down\_state[0] = 4032;

let down\_state[1] = 4656;

let down\_state[2] = 8456;

let down\_state[3] = 20740;

let down\_state[4] = 20740;

let down\_state[5] = 16898;

let down\_state[6] = -962;

let down\_state[7] = -32191;

let down\_state[8] = -735;

let down\_state[9] = -32239;

let down\_state[10] = 31774;

let down\_state[11] = 992;

return;

}

/\*

Cycle bird wing state to next in loop.

\*/

method void cycleWingState()

{

if (wing\_state = 3) {

let wing\_state = 1;

} else {

let wing\_state = wing\_state + 1;

}

return;

}

/\*

Return image coordinates for current wing state.

\*/

method Array getBirdVector()

{

if (wing\_state = 1) {

return up\_state;

}

if (wing\_state = 2) {

return mid\_state;

}

return down\_state;

}

/\*

Draw bird.

\*/

method void draw() {

var int memAddress;

let memAddress = (16384 + (y\_offset \* 32) + (x\_offset/16));

let bird\_vector = getBirdVector();

do Memory.poke(memAddress+0, bird\_vector[0]);

do Memory.poke(memAddress+32, bird\_vector[1]);

do Memory.poke(memAddress+64, bird\_vector[2]);

do Memory.poke(memAddress+96, bird\_vector[3]);

do Memory.poke(memAddress+128, bird\_vector[4]);

do Memory.poke(memAddress+160, bird\_vector[5]);

do Memory.poke(memAddress+192, bird\_vector[6]);

do Memory.poke(memAddress+224, bird\_vector[7]);

do Memory.poke(memAddress+256, bird\_vector[8]);

do Memory.poke(memAddress+288, bird\_vector[9]);

do Memory.poke(memAddress+320, bird\_vector[10]);

do Memory.poke(memAddress+352, bird\_vector[11]);

return;

}

/\*

Get image X-coordinate.

\*/

method int getXOffset()

{

return x\_offset;

}

/\*

Get image Y-coordinate.

\*/

method int getYOffset()

{

return y\_offset;

}

/\*

Get image width.

\*/

method int getSizeX()

{

return size\_x;

}

/\*

Get image height.

\*/

method int getSizeY()

{

return size\_y;

}

/\*

Dispose of Bird object.

\*/

method void dispose() {

do Memory.deAlloc(this);

return;

}

}

Jack Bird Game:

class FlappyBirdGame

{

/\*

Models the Flappy Bird game at a high level by creating in-game objects,

handling user inputs, and moving through game states.

\*/

field FlappyBird bird; // FlappyBird object.

field Scoreboard scoreboard; // Scoreboard object.

field int bird\_wing\_state\_count; // Track Bird wing image angle.

field int bird\_fall\_count; // Track Bird's downward movement.

field int bird\_height\_default; // Default Bird starting height.

field Array pipes; // Visible pipes on screen.

field int count\_pipes; // Number of pipes visible.

field int dist\_between\_pipes; // Interval between successive pipes.

field int dist\_after\_last\_pipe; // Amount of space following last created pipe.

field int jumps; // Number of jumps made in game.

field int total\_pipes; // Number of pipes created.

/\*

Flappy Bird Game constructor.

\*/

constructor FlappyBirdGame new() {

let bird\_height\_default = 150;

let scoreboard = Scoreboard.new();

let pipes = Array.new(10);

let count\_pipes = 0;

let dist\_between\_pipes = 100;

let dist\_after\_last\_pipe = 0;

let jumps = 1;

let total\_pipes = 0;

return this;

}

/\*

Display instructions on the screen.

\*/

method void displayInstructions()

{

do Output.moveCursor(0, 0);

do Output.printString("Welcome to Nand Bird.");

do Output.println();

do Output.printString("To jump, press the space bar.");

do Output.println();

do Output.printString("Ramya Sree E4");

do Output.println();

do Output.printString("To quit, press the 'q' key.");

do Output.println();

do Output.printString("To win, stay alive.");

return;

}

/\*

Main loop for running game.

\*/

method void run() {

var char key;

var char last\_key;

do initializeGameState();

do waitForGameToStart();

let last\_key = 32;

while (~(key = 81)) { // 'q' key

// Determine if new jump has occurred.

if ((key = 32) & ~(last\_key = 32)) { // space bar

let jumps = jumps + 1;

do bird.moveUp();

} else {

do updateBirdFallState();

do updateBirdWingState();

}

// If collision found, go to score and reset state.

if (findCollision()) {

do displayScore();

do Sys.wait(3000);

do initializeGameState();

do waitForGameToStart();

}

// Draw updated game to screen.

do draw();

// Check key pressed.

let last\_key = key;

let key = Keyboard.keyPressed();

do Sys.wait(10);

}

return;

}

/\*

Animate Bird before game begins.

\*/

method void waitForGameToStart()

{

var char key;

let key = Keyboard.keyPressed();

while (key = 0) {

do Sys.wait(10);

let key = Keyboard.keyPressed();

do bird.draw();

do updateBirdWingState();

}

return;

}

/\*

Set all fields to pre-game state.

\*/

method void initializeGameState()

{

var Pipe pipe;

var int i;

let i = 0;

do whiteout();

do displayInstructions();

let bird = FlappyBird.new(bird\_height\_default);

// Remove existing pipes.

while (i < count\_pipes) {

let pipe = pipes[i];

do pipe.dispose();

let i = i + 1;

}

let count\_pipes = 0;

let total\_pipes = 0;

let bird\_wing\_state\_count = 1;

let bird\_fall\_count = 1;

let dist\_after\_last\_pipe = dist\_between\_pipes;

return;

}

/\*

White out screen.

\*/

method void whiteout()

{

do Screen.setColor(false);

do Screen.drawRectangle(0, 0, 511, 255);

return;

}

/\*

Display final score to screen.

\*/

method void displayScore()

{

var int score;

let score = 0;

let score = calculateScore();

do scoreboard.draw(score);

return;

}

/\*

Update bird's rising/falling status.

\*/

method void updateBirdFallState()

{

// Only move bird after every five 'draw' cycles to prevent dropping too fast.

if (bird\_fall\_count = 5) {

do bird.moveDown();

let bird\_fall\_count = 1;

} else {

let bird\_fall\_count = bird\_fall\_count + 1;

}

return;

}

/\*

Update bird wing display state.

\*/

method void updateBirdWingState()

{

// Only cycle wings after every five 'draw' cycles to prevent changing too fast.

if (bird\_wing\_state\_count = 5) {

do bird.cycleWingState();

let bird\_wing\_state\_count = 1;

} else {

let bird\_wing\_state\_count = bird\_wing\_state\_count + 1;

}

return;

}

/\*

Draw onscreen objects and update their coordinates.

\*/

method void draw()

{

var Pipe pipe;

// Moves pipes left.

do shiftPipesLeft();

// Remove first pipe if no longer on screen.

do disposeUnseenPipe();

// Create new pipe object if existing last pipe has moved far enough.

do createNewPipe();

do bird.draw();

return;

}

/\*

Draw pipes and shift their coordinates to the left.

\*/

method void shiftPipesLeft()

{

var Pipe pipe;

var int i;

let i = 0;

while (i < count\_pipes) {

let pipe = pipes[i];

do pipe.draw();

do pipe.moveLeft();

let i = i + 1;

}

return;

}

/\*

Remove Pipe object once it has exited the screen.

\*/

method void disposeUnseenPipe()

{

var int i;

var Pipe pipe;

if (count\_pipes = 0) {

return;

}

let pipe = pipes[0];

if (pipe.getWidthCurrent() < 1) {

do pipe.dispose();

let count\_pipes = count\_pipes - 1;

let i = 0;

while (i < count\_pipes) {

let pipes[i] = pipes[i+1];

let i = i + 1;

}

}

return;

}

/\*

Create new pipe to go onscreen.

\*/

method void createNewPipe()

{

var Pipe pipe;

if (dist\_after\_last\_pipe = dist\_between\_pipes) {

let pipes[count\_pipes] = Pipe.new(jumps);

let count\_pipes = count\_pipes + 1;

let total\_pipes = total\_pipes + 1;

let dist\_after\_last\_pipe = 0;

} else {

let pipe = pipes[0];

let dist\_after\_last\_pipe = dist\_after\_last\_pipe + pipe.getWidthIncrement();

}

return;

}

/\*

Calculate final score.

\*/

method int calculateScore()

{

var Pipe pipe;

var int i;

var int score;

let score = total\_pipes;

let i = 0;

while (i < count\_pipes) {

let pipe = pipes[i];

if (bird.getXOffset() < (pipe.getXOffset() + pipe.getWidthCurrent() + 1)) {

let score = score - 1;

}

let i = i + 1;

}

return score;

}

/\*

Determine if there been a collision between bird and borders or pipes.

\*/

method boolean findCollision()

{

var Pipe pipe;

var int i;

let i = 0;

while (i < count\_pipes) {

let pipe = pipes[i];

if (collides(pipe)) {

return true;

}

let i = i + 1;

}

return false;

}

/\*

Has Bird collided with specific pipe.

\*/

method boolean collides(Pipe pipe)

{

// Has bird hit the top of the screen.

if (bird.getYOffset() = 0) {

return true;

}

// Has bird hit the bottom of the screen.

if ((bird.getYOffset() + bird.getSizeY()) = 255) {

return true;

}

// Is bird still in front of pipe.

if (pipe.getXOffset() > (bird.getXOffset() + bird.getSizeX() + 1)) {

return false;

}

// Has bird passed pipe.

if ((pipe.getXOffset() + pipe.getWidthCurrent()) < bird.getXOffset()) {

return false;

}

// Has bird hit top section of pipe.

if (pipe.getGapTop() > bird.getYOffset()) {

return true;

}

// Has bird hit bottom section of pipe.

if (pipe.getGapBottom() < (bird.getYOffset() + bird.getSizeY())) {

return true;

}

return false;

}

/\*

Dispose of FlappyBirdGame object and objects created as properties.

\*/

method void dispose() {

var Pipe pipe;

var int i;

let i = 0;

while (i < count\_pipes) {

let pipe = pipes[i];

do pipe.dispose();

let i = i + 1;

}

do scoreboard.dispose();

do bird.dispose();

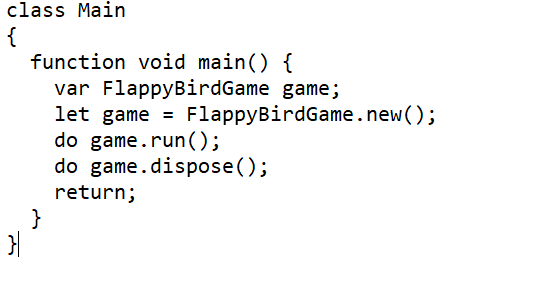
do Memory.deAlloc(this);

return;

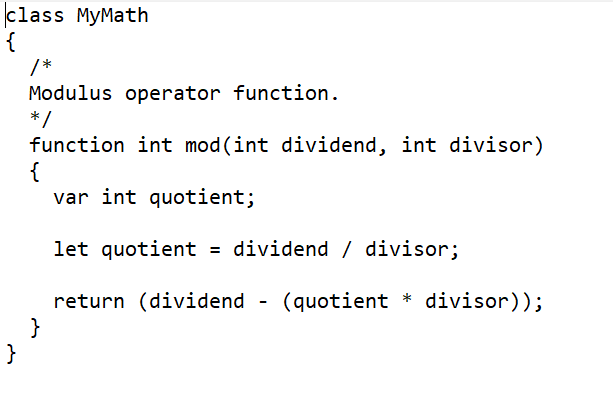
}

}

Main:



MyMath:



Pipe:

class Pipe

{

/\*

Models an individual Pipe object and creates/updates the pipe image.

\*/

field int width; // Image full-visibility width.

field int width\_current; // Current width (for entering/leaving screen).

field int width\_increment; // Amount to shift pipe for each step.

field int x\_offset; // Current x-coordinate for pipe image.

field int gap\_size; // Distance between top and bottom pieces.

field int gap\_top; // Coordinate for bottom of top piece.

field int gap\_bottom; // Coordinate for top of bottom piece.

field int screen\_width; // Screen width.

field int screen\_height; // Screen height.

/\*

Construct new Pipe object.

@param int gap\_offset Offset of pipe's gap.

\*/

constructor Pipe new(int gap\_offset) {

let width = 12;

let width\_increment = 1;

let width\_current = width\_increment;

let x\_offset = 509;

let screen\_width = 512;

let screen\_height = 256;

let gap\_size = 65;

let gap\_top = calculateGapTop(gap\_offset);

let gap\_bottom = (gap\_top + gap\_size);

return this;

}

/\*

Calculate coordinate for bottom of top piece.

\*/

method int calculateGapTop(int offset)

{

var int mod;

let mod = MyMath.mod(offset, 6);

return (mod + 1) \* 30;

}

/\*

Shift complete pipe image to the left.

\*/

method void shift()

{

if (x\_offset < (screen\_width - width )) {

do Screen.setColor(false);

do Screen.drawRectangle(x\_offset + width\_current - width\_increment, 0, x\_offset + width\_current, gap\_top);

do Screen.drawRectangle(x\_offset + width\_current - width\_increment, gap\_bottom, x\_offset + width\_current, 255);

}

return;

}

/\*

Move pipe image to the left. Deleting image entirely if coords no longer

on screen.

\*/

method void moveLeft()

{

do shift();

// Width\_current is either growing or shrinking

if (width\_current < width) {

if (x\_offset > 0) {

let width\_current = width\_current + width\_increment;

}

}

if (x\_offset > 1) {

let x\_offset = x\_offset - width\_increment;

} else {

let x\_offset = 0;

let width\_current = width\_current - width\_increment;

}

return;

}

/\*

Get current width.

\*/

method int getWidthCurrent()

{

return width\_current;

}

/\*

Get width shift increment.

\*/

method int getWidthIncrement()

{

return width\_increment;

}

/\*

Get current x-coordinate.

\*/

method int getXOffset()

{

return x\_offset;

}

/\*

Get y-coordinate of top of gap.

\*/

method int getGapTop()

{

return gap\_top;

}

/\*

Get y-coordinate of bottom of gap.

\*/

method int getGapBottom()

{

return gap\_bottom;

}

/\*

Draw pipe at current x-offset.

\*/

method void draw()

{

if (x\_offset - width\_increment > 0) {

do Screen.setColor(true);

do Screen.drawRectangle(x\_offset - width\_increment, 0, x\_offset, gap\_top);

do Screen.drawRectangle(x\_offset - width\_increment, gap\_bottom, x\_offset, 255);

}

return;

}

/\*

Dispose of pipe object.

\*/

method void dispose() {

do Memory.deAlloc(this);

return;

}

}

Score Board:

class Scoreboard

{

/\*

Models the Scoreboard object and creates/displays scoreboard with designated score..

\*/

field int score; // Final score.

field int kerning; // Space between characters.

field int default\_score\_offset\_x; // Default x-coord for start of Scoreboard image.

field int score\_offset\_x; // Current scoreboard draw cursor x-coord.

field int score\_offset\_y; // Current scoreboard draw cursor y-coord.

field int score\_pixel\_size; // Abstract pixel size.

/\*

Scoreboard constructor.

\*/

constructor Scoreboard new()

{

let kerning = 15;

let default\_score\_offset\_x = 50;

let score\_offset\_y = 100;

let score\_pixel\_size = 4;

return this;

}

/\*

Draw score.

\*/

method void draw(int score)

{

let score\_offset\_x = default\_score\_offset\_x;

do drawPrefix();

do drawScore(score);

return;

}

/\*

Draw 'SCORE: ' to prefix actual score value image.

\*/

method void drawPrefix()

{

do blackout();

do drawS();

do drawC();

do drawZero(); // Re-use zero image for letter 'O'.

do drawR();

do drawE();

do drawColon();

return;

}

/\*

Draw actual score number.

\*/

method void drawScore(int score)

{

var Array digits;

var int digit;

var int i;

let digits = getScoreDigits(score);

let i = 1;

while (i < (digits[0] + 1)) {

let digit = digits[i];

if (digit = 0) {

do drawZero();

}

if (digit = 1) {

do drawOne();

}

if (digit = 2) {

do drawTwo();

}

if (digit = 3) {

do drawThree();

}

if (digit = 4) {

do drawFour();

}

if (digit = 5) {

do drawFive();

}

if (digit = 6) {

do drawSix();

}

if (digit = 7) {

do drawSeven();

}

if (digit = 8) {

do drawEight();

}

if (digit = 9) {

do drawNine();

}

let i = i + 1;

}

return;

}

/\*

Get array of specific score digits.

\*/

method Array getScoreDigits(int score)

{

var int remainder;

var Array digits;

let digits = Array.new(5);

if (score > 999) {

let score = 1000; // Maximum score.

let digits[0] = 4; // Number of digits to draw.

let digits[1] = 1;

let digits[2] = 0;

let digits[3] = 0;

let digits[4] = 0;

return digits;

}

if (score > 99) {

let digits[0] = 3; // Number of digits to draw.

let digits[1] = score/100; // Calculate hundreds-place digit.

let remainder = MyMath.mod(score, 100);

let digits[2] = remainder/10; // Calculate tens-place digit.

let digits[3] = MyMath.mod(remainder, 10); // Calculate ones-place digit.

return digits;

}

if (score > 9) {

let digits[0] = 2; // Number of digits to draw.

let digits[1] = score/10; // Calculate tens-place digit.

let digits[2] = MyMath.mod(score, 10); // Calculate ones-place digit.

return digits;

}

let digits[0] = 1; // Number of digits to draw.

let digits[1] = score; // Calculate ones-place digit.

return digits;

}

/\*

Black out screen.

\*/

method void blackout()

{

do Screen.setColor(true);

do Screen.drawRectangle(0, 0, 511, 255);

return;

}

/\*

Draw 'S' character.

\*/

method void drawS()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 1)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw 'C' character.

\*/

method void drawC()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 1)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw 'R' character.

\*/

method void drawR()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + score\_pixel\_size,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + (score\_pixel\_size \* 7)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw 'E' character.

\*/

method void drawE()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 1)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 7)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw colon character.

\*/

method void drawColon()

{

var int width;

var int height;

let width = 1 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 0),

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 2)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 0),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 5)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '0' character.

\*/

method void drawZero()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(score\_offset\_x, score\_offset\_y, score\_offset\_x + width, score\_offset\_y + height);

do Screen.setColor(true);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x + width - score\_pixel\_size,

score\_offset\_y,

score\_offset\_x + width,

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + height - score\_pixel\_size,

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + height

);

do Screen.drawRectangle(

score\_offset\_x + width - score\_pixel\_size,

score\_offset\_y + height - score\_pixel\_size,

score\_offset\_x + width,

score\_offset\_y + height

);

do Screen.drawRectangle(

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + score\_pixel\_size,

score\_offset\_x + width - score\_pixel\_size,

score\_offset\_y + height - score\_pixel\_size

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '1' character.

\*/

method void drawOne()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(score\_offset\_x, score\_offset\_y, score\_offset\_x + width, score\_offset\_y + height);

do Screen.setColor(true);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 2),

score\_offset\_y + score\_pixel\_size

);

do Screen.setColor(true);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 2),

score\_offset\_x + (score\_pixel\_size \* 2),

score\_offset\_y + height - score\_pixel\_size

);

do Screen.setColor(true);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 3),

score\_offset\_y,

score\_offset\_x + width,

score\_offset\_y + height - score\_pixel\_size

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '2' character.

\*/

method void drawTwo()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + score\_pixel\_size,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + score\_pixel\_size,

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + (score\_pixel\_size \* 2)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + score\_pixel\_size,

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 2)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 3),

score\_offset\_y + (score\_pixel\_size \* 2),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 2),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 3),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 2),

score\_offset\_y + (score\_pixel\_size \* 5)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 5),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '3' character.

\*/

method void drawThree()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + score\_pixel\_size,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + score\_pixel\_size,

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + (score\_pixel\_size \* 2)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 2),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 0),

score\_offset\_y + (score\_pixel\_size \* 5),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '4' character.

\*/

method void drawFour()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 5)

);

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 5)

);

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 2),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 3),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '5' character.

\*/

method void drawFive()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 1)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '6' character.

\*/

method void drawSix()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + score\_pixel\_size,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + score\_pixel\_size,

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 2)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '7' character.

\*/

method void drawSeven()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 2)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 3),

score\_offset\_y + (score\_pixel\_size \* 2),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 2),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 3),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '8' character.

\*/

method void drawEight()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + score\_pixel\_size,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + score\_pixel\_size,

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 0),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Draw '9' character.

\*/

method void drawNine()

{

var int width;

var int height;

let width = 5 \* score\_pixel\_size;

let height = 7 \* score\_pixel\_size;

do Screen.setColor(false);

do Screen.drawRectangle(

score\_offset\_x + score\_pixel\_size,

score\_offset\_y,

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + score\_pixel\_size

);

do Screen.drawRectangle(

score\_offset\_x,

score\_offset\_y + score\_pixel\_size,

score\_offset\_x + score\_pixel\_size,

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 1),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 3)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 3),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 4),

score\_offset\_x + (score\_pixel\_size \* 5),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 0),

score\_offset\_y + (score\_pixel\_size \* 5),

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6)

);

do Screen.drawRectangle(

score\_offset\_x + (score\_pixel\_size \* 1),

score\_offset\_y + (score\_pixel\_size \* 6),

score\_offset\_x + (score\_pixel\_size \* 4),

score\_offset\_y + (score\_pixel\_size \* 7)

);

let score\_offset\_x = score\_offset\_x + width + kerning;

return;

}

/\*

Dispose of Scoreboard object.

\*/

method void dispose() {

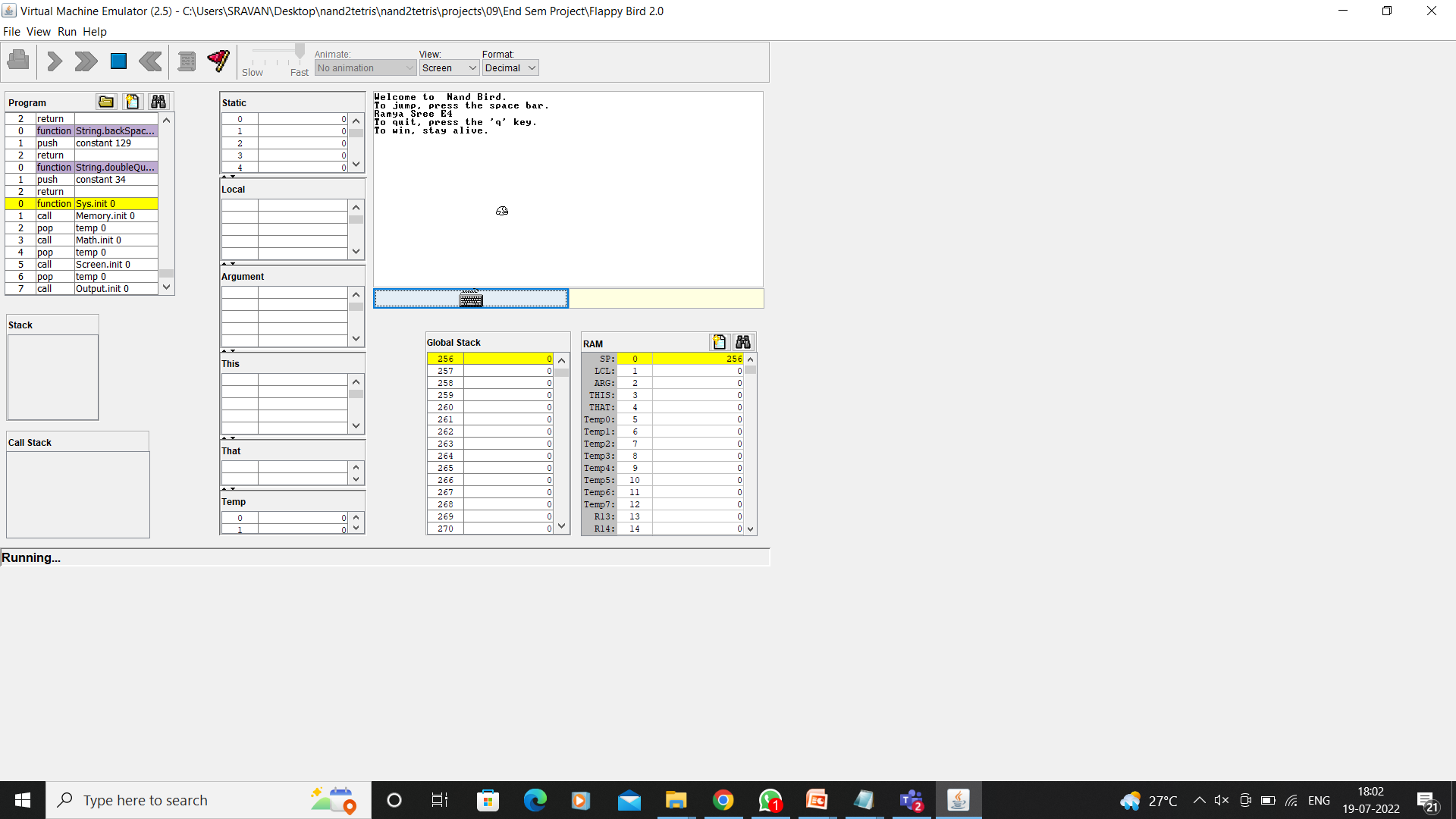
do Memory.deAlloc(this);

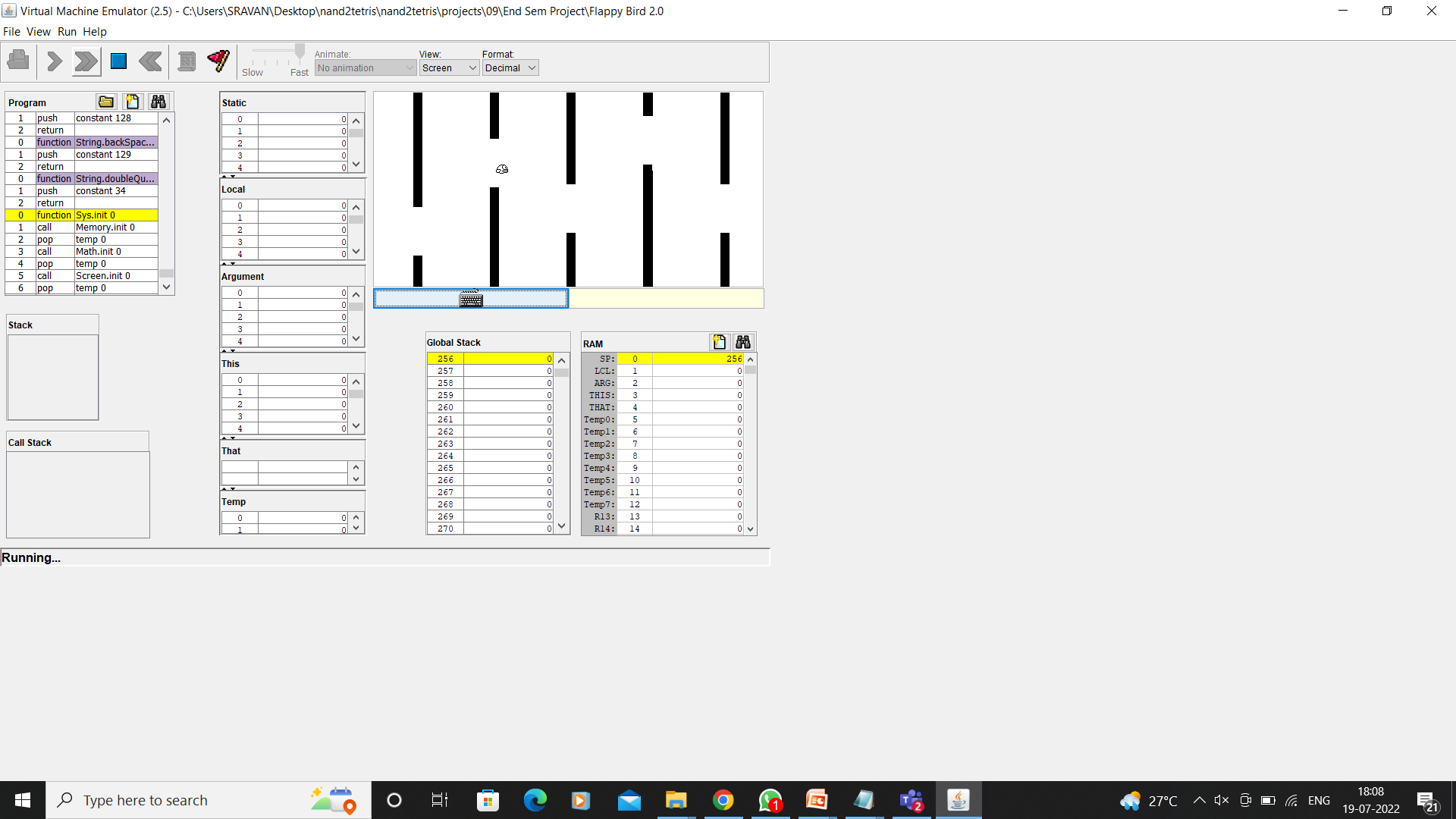
return;

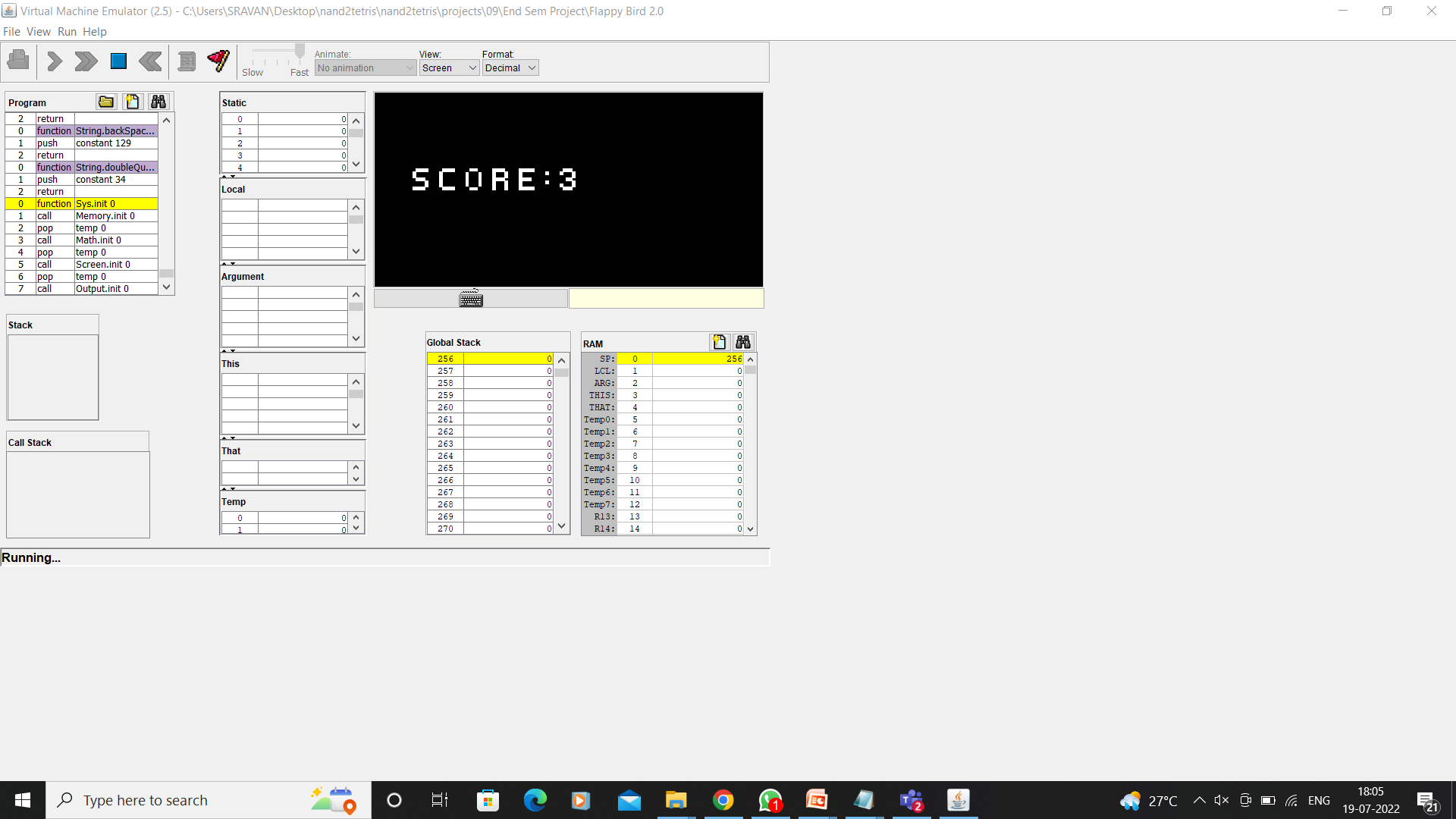
}

}

OUTPUT:







**FUTURE SCOPE**

* We can make it more user-friendly.
* We can add more options like top score and players’ profiles.
* We can add other obstacles.
* We can add a multiplayer option.

**CONCLUSION**

This project is done in JACK programming. In this project, you can play the popular “Jack Bird” just like you played it elsewhere.

You have to use the Spacebar to control the bird. The code was written in jack language and it was compiled using jack compiler which converts jack file into VM file. The final simulation was done in a VM emulator where we can play the game.