import rules import ui/main_ui import ui/widgets import ui/types import ui/settings as settings import constants import util

import nimraylib_now

import random import math import system/iterators import os as os

proc triangleVertices(t: Triangle, ui: Ui): (Vector2, Vector2, Vector2) = let (w, h) = triangleSize.tuple (t.pos + Up.rotate(t.heading + PI/2) * h * 2/3, t.pos + Left.rotate(t.heading + PI/2) * w * 0.5 + Down.rotate(t.heading + PI/2) * h * 1/3, t.pos + Right.rotate(t.heading + PI/2) * w * 0.5 + Down.rotate(t.heading + PI/2) * h * 1/3)

proc generateTriangles(ui: Ui, n: int): seq[Triangle] = for i in 0..<n: let heading = rand(2 * PI) speedRange = ui.get(MinSpeed)..ui.get(MaxSpeed) speed = rand(speedRange) (w, h) = triangleSize.tuple x = rand(w..(screenWidth.float - w)) y = rand(h..(screenHeight.float - h)) result.add(Triangle(pos: Vector2(x: x, y: y), vel: Vector2.fromRad(heading) * speed,))

func moveTriangles(triangles: var seq[Triangle], dt: float) = for t in triangles.mitems: t.pos += t.vel * dt

proc avoidEdges(triangles: var seq[Triangle], ui: Ui) = let screen = Rectangle(x: 0.0, y: 0.0, width: screenWidth.float, height: screenHeight.float) for t in triangles.mitems: let vr = ui.get(ViewRadius) f = ui.get(EvadeEdges) distanceLeft = t.pos.x - screen.x distanceRight = screen.x + screen.width - t.pos.x distanceTop = t.pos.y - screen.y distanceBottom = screen.y + screen.height - t.pos.y

```
if distanceLeft < vr:
  t.vel.x += (vr - distanceLeft) * f
elif distanceRight < vr:
  t.vel.x += -(vr - distanceRight) * f
elif distanceTop < vr:
  t.vel.y += (vr - distanceTop) * f
elif distanceBottom < vr:
  t.vel.y += -(vr - distanceBottom) * f
```

proc drawTriangles(triangles: seq[Triangle], ui: Ui) = for t in triangles: let (a, b, c) = triangleVertices(t, ui) cHeading = PI - abs(t.heading) cval = uint8(cHeading * 255.0 / PI) color = Color(r: 0, g: 255 - cval, b: cval, a: 255) drawTriangle(a, b, c, color)

func dt(): float = getFrameTime()

proc main*() = case os.paramCount() of 0: discard # Use defaults of 2: screenWidth = os.paramStr(1).parseInt screenHeight = os.paramStr(2).parseInt else: echo "usage: boids_sim [screen_width screen_height]" quit(1)

initWindow(screenWidth, screenHeight, "Boids Sim") setTargetFPS(60)

var mainUi = setupMainUi() triangles = mainUi.generateTriangles(mainUi.get(NumTriangles).int) while not windowShouldClose(): let dt = dt()

```
### Input ###
mainUi.update()

### Rules ###
apply_rules(triangles, mainUi, dt)
avoidEdges(triangles, mainUi)
moveTriangles(triangles, dt)

### Update Triangles ###
let delta = mainUi.get(NumTriangles).int - triangles.len
if delta > 0:
  triangles &= mainUi.generateTriangles(delta)
elif delta < 0:
  triangles.setLen(triangles.len + delta)

### Draw ###
beginDrawing()

# Clear
clearBackground(Black)

# Triangles
drawTriangles(triangles, mainUi)

if settings.debugMode:
  # Protected zone
  drawCircleLines(triangles[0].pos.x.int, triangles[0].pos.y.int, mainUI.get(ProtectedZone),

  # View radius
  drawCircleLines(triangles[0].pos.x.int, triangles[0].pos.y.int, mainUI.get(ViewRadius), Wh

  # Color the first triangle differently
  let (a, b, c) = triangleVertices(triangles[0], mainUi)
  drawTriangle(a, b, c, White)

  drawFPS(margin.int, int(widgetHeight + 2 * margin))

# User interface
mainUi.draw()

endDrawing()
```

closeWindow() echo “<>”