MINISTERUL EDUCAȚIEI, CULTURII ȘI

CERCETĂRII AL REPUBLICII MOLDOVA

Universitatea Tehnică a Moldovei

Facultatea Calculatoare, Informatică şi

Microelectronică Departamentul Inginerie Software și

Automatică

Cuzmin Simion Faf-221

Report

Laboratory work n. 5.3

of Computer Graphics

Checked by:

Olga Grosu, *university assistant*

DISA, FCIM, UTM

Chișinău – 2023

Take your creature and build a system of creatures. How can they interact with each other? Can you use inheritance and polymorphism to create a variety of creatures, derived from the same code base? Develop a methodology for how they compete for resources (for example, food). Can you track a creature's "health" much like we tracked a particle's lifespan, removing creatures when appropriate? What rules can you incorporate to control how creatures are born?

The program code with relevant comments:

```
import processing.core.*;
import java.util.ArrayList;
import java.util.Iterator;
```

```
void setup() {
  size(800, 600);
  creatures = new ArrayList<Creature>();
  foods = new ArrayList<Food>();

  // Create 3 creatures
  for (int i = 0; i < 3; i++) {
    creatures.add(new
Creature(random(width), random(height), 30,
0.02, 50));
  }

  // Create 100 food items
  for (int i = 0; i < 100; i++) {
    foods.add(new Food(random(width),
random(height), 10));
  }

  // Create poison food
  for (int i = 0; i < 10; i++) {
```

```
    foods.add(new PoisonFood(random(width),
random(height), 10));
  }
}

void draw() {
  background(224, 158, 118);

  // Iterate over creatures using Iterator
  Iterator<Creature> creatureIterator =
creatures.iterator();
  while (creatureIterator.hasNext()) {
    Creature creature =
creatureIterator.next();
    creature.seek(foods);
    creature.update();
    creature.display();
    if (creatures.size() > 1) {
      creature.checkCollisions(creatures,
foods);
    }
```

```
  }

  // Iterate over foods using Iterator
  Iterator<Food> foodIterator =
foods.iterator();
  while (foodIterator.hasNext()) {
    Food food = foodIterator.next();
    food.display();
  }

  // Close the program if only one creature
is left
  if (creatures.size() == 1) {
    exit();
  }
}

ArrayList<Creature> creatures;
ArrayList<Food> foods;
```

```
class Food {
  PVector position;
  float radius;

  Food(float x, float y, float radius) {
    // Ensure the initial position is at
least 100 pixels away from all borders
    float adjustedX = max(radius + 100,
min(x, width - radius - 100));
    float adjustedY = max(radius + 100,
min(y, height - radius - 100));
    position = new PVector(adjustedX,
adjustedY);
    this.radius = radius;
  }

  void display() {
    fill(0, 255, 0);
    ellipse(position.x, position.y, radius
* 2, radius * 2);
  }
```

```
}

class PoisonFood extends Food {

  PoisonFood(float x, float y, float
radius) {
    super(x, y, radius);
  }

  @Override
  void display() {
    fill(139, 0, 0); // RGB color for
poison food
    triangle(
      position.x, position.y - radius,
      position.x - radius, position.y +
radius,
      position.x + radius, position.y +
radius
    );
  }
```

```
}


class Creature {
  PVector center;
  float radius;
  float angle;
  float angleVelocity;
  float speed;
  PVector acceleration;
  PVector velocity;
  PVector direction;
  ArrayList<Wings> wingsList;
  float lastMillis;


  Creature(float x, float y, float radius,
float angleVelocity, int numWings) {
    center = new PVector(x, y);
    this.radius = radius;
    this.angle = 0;
```

```
    this.angleVelocity = angleVelocity;
    this.speed = 20;
    this.acceleration = new PVector(0.1,
0.1);
    this.velocity = new PVector(this.speed,
0);
    this.direction = new PVector(0, 0);
    wingsList = new ArrayList<Wings>();
    lastMillis = millis();

    for (int i = 0; i < numWings; i++) {
      wingsList.add(new Wings());
    }
  }

  void seek(ArrayList<Food> foods) {
    if (foods.size() > 0) {
      Food target = findNearestFood(foods);
      PVector desired =
PVector.sub(target.position, center);
      desired.setMag(speed);
```

```
      // Calculate the steering force
      PVector steer = PVector.sub(desired,
velocity);
      steer.limit(0.1); // Limit the
steering force to avoid abrupt changes

      // Apply steering force to update the
direction

this.direction.add(steer).normalize();

      // Set the new velocity based on the
updated direction

this.velocity.set(PVector.mult(this.directi
on, this.speed));
    }
  }

  void applyForce(PVector force) {
    acceleration.add(force);
```

```
  }


  void update() {
    float deltaTime = (float) (millis() -
lastMillis) / 1000.0;
    lastMillis = millis();


    // Update speed based on acceleration
    this.speed += this.acceleration.mag() *
deltaTime;


    // Update velocity based on
acceleration

this.velocity.add(PVector.mult(this.acceler
ation, deltaTime));


    // Update position based on velocity

this.center.add(PVector.mult(this.velocity,
deltaTime));
```

```
    // Oscillate the angle of the
creature's wings based on speed
    this.angle += this.speed *
this.angleVelocity;


    // Update wings position
    for (Wings wings : wingsList) {
      wings.update(this.center,
this.angle);
    }


    // Check boundaries and reverse
direction if needed
    if (this.center.x > width -
this.radius) {
      this.center.x = width - this.radius;
      this.velocity.x *= -1; // Reverse x-
direction
    } else if (this.center.x < this.radius)
{
      this.center.x = this.radius;
```

```
      this.velocity.x *= -1; // Reverse x-
direction
    }

    if (this.center.y > height -
this.radius) {
      this.center.y = height - this.radius;
      this.velocity.y *= -1; // Reverse y-
direction
    } else if (this.center.y < this.radius)
{
      this.center.y = this.radius;
      this.velocity.y *= -1; // Reverse y-
direction
    }

    // Reset acceleration
    this.acceleration.mult(0);
  }

  void display() {
```

```
    pushMatrix();
    translate(this.center.x,
this.center.y);

    ellipse(0, 0, this.radius * 2,
this.radius * 2);

    fill(216, 240, 232);
    ellipse(-10, -10, 5, 5);
    ellipse(10, -10, 5, 5);

    arc(0, 5, 30, 20, 0, PI);
    fill(133, 122, 106);

    for (Wings wings : wingsList) {
      wings.display(this.radius);
    }

    popMatrix();
  }
```

```
void checkCollisions(ArrayList<Creature>
creatures, ArrayList<Food> foods) {
  Iterator<Creature> creatureIterator =
creatures.iterator();
  while (creatureIterator.hasNext()) {
    Creature otherCreature =
creatureIterator.next();
    if (this != otherCreature &&
collidesWith(otherCreature)) {
      if (this.radius >
otherCreature.radius) {
        this.grow(1.3);
        creatureIterator.remove(); // Use
iterator to remove the smaller creature
        if (this.radius >
MAX_CREATURE_RADIUS) {
          handleTooBigError();
        }
      } else if (this.radius <
otherCreature.radius) {
        otherCreature.grow(1.3);
```

```
        creatureIterator.remove(); // Use
iterator to remove the smaller creature
        if (otherCreature.radius >
MAX_CREATURE_RADIUS) {
          handleTooBigError();
        }
        break;
      } else {
        // Equal-sized creatures, randomly
choose one to survive
        if (random(1) < 0.5) {
          this.grow(1.3);
          creatureIterator.remove();
          if (this.radius >
MAX_CREATURE_RADIUS) {
            handleTooBigError();
          }
        } else {
          otherCreature.grow(1.3);
          creatureIterator.remove();
```

```
          if (otherCreature.radius >
MAX_CREATURE_RADIUS) {
            handleTooBigError();
          }
          break;
        }
      }
    }
  }

  Iterator<Food> foodIterator =
foods.iterator();
  while (foodIterator.hasNext()) {
    Food food = foodIterator.next();
    if (collidesWith(food)) {
      if (food instanceof PoisonFood) {
        this.grow(0.5); // Poisonous food,
reduce growth
      } else {
        this.grow(1.1); // Regular food
      }
```

```
      foodIterator.remove(); // Use
iterator to remove the food
      if (this.radius >
MAX_CREATURE_RADIUS) {
        handleTooBigError();
      }
      break;
    }
  }
}



  void handleTooBigError() {
    println("Error: Creature is too big!
Exiting program.");
    exit();
  }


  static final float MAX_CREATURE_RADIUS =
200; // Adjust the maximum allowed creature
radius as needed
```

```
  void grow(float scaleFactor) {
    this.radius *= scaleFactor;
    for (Wings wings : wingsList) {
      wings.changeSize(scaleFactor);
    }
  }

  boolean collidesWith(Creature
otherCreature) {
    float distance = dist(center.x,
center.y, otherCreature.center.x,
otherCreature.center.y);
    return distance < (radius +
otherCreature.radius) / 2;
  }

  boolean collidesWith(Food food) {
```

```
    float distance = dist(center.x,
center.y, food.position.x,
food.position.y);

    return distance < (radius +
food.radius) / 2;

  }


  Food findNearestFood(ArrayList<Food>
foods) {

    if (foods.size() > 0) {

      Food nearest = foods.get(0);

      float record =
dist(nearest.position.x,
nearest.position.y, center.x, center.y);

      for (Food food : foods) {

        float d = dist(food.position.x,
food.position.y, center.x, center.y);

        if (d < record) {

          record = d;

          nearest = food;

        }

      }
```

```
      return nearest;
    } else {
      return null;
    }
  }
}

class Wings {
  float wingLength;
  float oscillationAmplitude;
  float wingAngleOffset;

  Wings() {
    wingLength = 30;
    oscillationAmplitude = 20;
    wingAngleOffset = random(TWO_PI);
  }

  void update(PVector center, float angle)
{
```

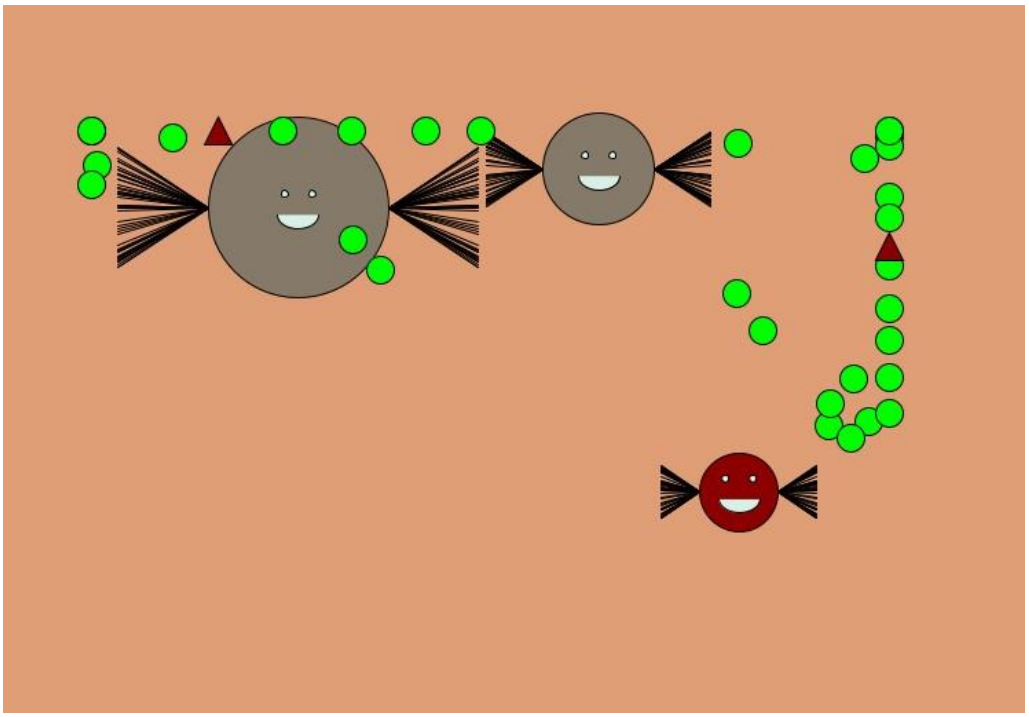
```
    // Wings update logic, if needed
  }

  void display(float creatureRadius) {
    float wingOscillation = sin(frameCount
* 0.05 + wingAngleOffset) *
oscillationAmplitude;

    line(creatureRadius, 0, creatureRadius
+ wingLength, wingOscillation);
    line(-creatureRadius, 0, -
creatureRadius - wingLength,
wingOscillation);
  }

  void changeSize(float scaleFactor) {
    this.wingLength *= scaleFactor;
    this.oscillationAmplitude *=
scaleFactor;
  }
}
```

- Screen printing of program execution:



- Student's conclusions and reflections:

The provided code creates a simulation in Processing where creatures seek and consume food while avoiding collisions with each other. The creatures exhibit autonomous movement, and their wings oscillate as they navigate the environment. The simulation includes regular food items and poison food items, with creatures growing in size when consuming either type.

The main functionality revolves around seeking and consuming food, handling collisions between creatures, and addressing the scenario where a creature grows too

large. If a collision occurs between two creatures, the larger one absorbs the smaller, and if the creatures are equal in size, one is randomly chosen to survive.

The code structure employs object-oriented programming, with classes for Creature, Food, and PoisonFood. Iterators are used to traverse through the lists of creatures and food items, facilitating dynamic updates during the simulation. The simulation terminates when only one creature remains.

However, there are potential issues with concurrent modification exceptions due to simultaneous modification of the creatures and food lists. These exceptions may arise when a creature consumes food and grows in size, leading to collisions with other creatures in the same iteration cycle.

To address this, additional checks and modifications have been introduced to ensure that the simulation runs smoothly without errors. These adjustments include using separate iterators for creatures and foods, removing elements through iterators, and handling scenarios where a creature grows too large, ensuring a stable and functional simulation.