

Computer Animation Practical 2 – Crowd Simulation

Introduction

Program Explanation

Compiling the Program

The Interface

The Classes

The Person Class

The Universe Class

Tasks

Task 1: Collision Detection And Reaction (10 Marks)

Task 2: Flocking (30 Marks)

Task 3: War Scene

Basic Requirements (30 Marks):

Enhancement (30 Marks):

Task 4: Written Report

Acknowledgement

Introduction

In this practical, you are required to implement a crowd simulation system using Java. A template program is given to you. You are only requested to focus on programming the movement function of the 2D avatars in the scene. Upon completion of the program, you will be able to generate a scene in which multiple avatars fighting with each others. The project will require 10-15 hours of work.

Program Explanation

Compiling the Program

You will be given a Java application. To compile the program, type the following command:

```
del *.class
del actor\*.class
del geometry\*.class
javac *.java actor\*.java geometry\*.java
java crowd
```

The Interface

The program consists of a control panel and a simulation panel. In the control panel, you can add different type of “actors”, including “person”, “square obstacle”, “triangle obstacle”, and “goal point”. One you add the actors, you will immediately see them in the simulation panel. In the simulation panel, you can left click and drag the actors to set their positions. You can also right click and drag to set the sizes of them. Finally, you could use mouse scroll to set the scale of the scene. When you place your mouse over any of the actors, the relative information is show in the control panel.

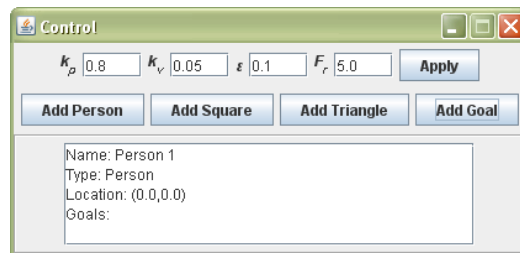


Figure 1: The control panel

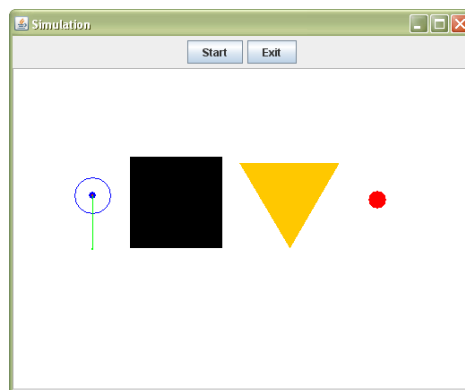


Figure 2: The simulation panel with four actors, namely “Person”, “Square”, “Triangle”, and “Goal” from left to right

The Classes

In the Java application you will have different Java files. This section will explain their hierarchical structures and uses.

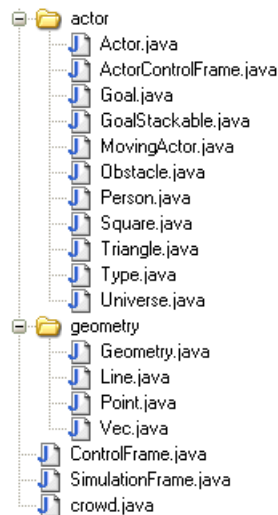


Figure 3: The classes in the project

Some important classes are highlighted below:

- crowd.java ← The class to launch the application
- ControlFrame.java ← The class to draw the control panel and react to user input to the panel
- SimulationFrame.java ← The class to draw the simulation panel and react to user input to the panel
- actor/Universe.java ← The class to contain every entity in the program, including “Person”, “Square”, “Triangle” and “Goal”
- actor/Actor.java ← The base class for every entity in the program
- actor/Goal.java ← “Goal” in the program, extended from the Actor class
- actor/Obstacle.java ← Extended from the Actor class, only used for the Square and Triangle classes
- actor/Square.java ← “Square Obstacle” in the program, extended from the Obstacle class
- actor/Triangle.java ← “Triangle Obstacle” in the program, extended from the Obstacle class
- actor/MovingActor.java ← Extended from the Actor class, only used for the Person class
- actor/Person.java ← “Person” in the program, extended from the MovingActor class

In general, when you launch the class crowd.java, it calls the two classes ControlFrame.java and SimulationFrame.java to draw the control panel and the simulation panel. In the background, the universe is setup using the Universe.java. In the universe, there may be different entities such as “Person” (Person.java), “Square” (Square.java), “Triangle” (Triangle.java) and “Goal” (Goal.java).

The Person Class

The person class is where you will spend most of your time on. In this section more details about the person class will be explained.

In the person class, you could:

- Call “position()” to get the 2D position of the person
e.g. position().xValue(), position().yValue()

- Call “velocity()” and “acceleration()” to get the 2D velocity and acceleration of the person e.g. velocity().xValue(), velocity.yValue()
- Call “personalSpaceRadius()” to get the size occupied by the person
- Call “setPosition()” to set the position the person

There is a variable call “goals” which holds a stack of goals for this person. You could:

- Call “goals.empty()” to check if the goal stack is empty
- Call “goals.clear()” to clear all existing goals
- Call “pushGoal()” to push a goal in the stack

The Universe Class

In the universe class you can initialize the scene. You could new any type of actors and placed them in the scene before the program starts.

You should initialize the scene in the “initialize()” function. The following code shows you how to add a person to the scene:

```
P = new Person(center(), new Vec(0, 0), new Vec(), Color.blue, 1.0);
Universe.add(p);
```

You could also add other actors into the scene as well.

Tasks

There are four tasks in this project. The first three of them are to program the system, while the last one is a written report. You need to submit THREE SEPARATE RUNABLE PROGRAMME for task 1 to 3, and a one page report for task 4.

Task 1: Collision Detection And Reaction (10 Marks)

The objective of this task is to help you to get familiar with the program. In this class, you will implement the code to avoid two persons overlap to each other.

First, you need to initialize the scene. Update the “initialize()” function in the Universe class. Place two persons into the scene.

Then, you need to update the “movementFunction()” function in the Person class. First, you need to detect collision between the current person and other persons. To do so, you need to check the distance between the current person and all other persons. And compare the distance with the sum of “personalSpaceRadius()” for every pair of them. The following code could be useful for you to start with:

```
// Find collided person
for (int i = 0; i < Universe.actors.size(); i++)
{
    // Get actor
    Actor a = (Actor)(Universe.actors.elementAt(i));
    // If a is a person
```

```

    if (a instanceof Person)
        if ( a != this)
        {
            // Do collision detection here
        }
    }
}

```

Then, you need to move the person away such that it is not overlapped with the others. You need to calculate the direction and distance for the person to move, and the call “setPosition()” to set the new position of the avatar.

To test your function, you could drag one person in the simulation panel towards the other one to see if they react to the collision.

Task 2: Flocking (30 Marks)

Next, you will need to implement a flocking system.

First, initialize your scene in the universe class such that you have some persons, some obstacles, and one goal. Assign the goal to the goal queue of ONE OF THE PERSONS using the “pushGoal” function. Indicate this person as a leader by setting the “bLeader” variable. Based on the code available, the leader will move towards the goal. You interact with the system by dragging the goal in the simulation panel such that the leader moves to follow the goal.

The movement of OTHER PERSONS in the scene is decided by the following forces:

- Separation: Avoid colliding with neighbors and objects
- Alignment: Steer towards average heading of neighbors
- Cohesion: Steer towards average position of neighbors

You need to update the movement function create the flocking effect. If the person is a leader, it will simply move towards the goal. However, if the person is not a leader, it obeys the flocking behaviors.

Task 3: War Scene

In this task you will create a scene in a war, in which two groups of persons fighting with each other.

Basic Requirements (30 Marks):

- Initialize the scene such that there are two groups of persons and some obstacles. Set the “iGroup” variable for each person to indicate its group. For each group, there is one leader. The two groups are initially far apart.
- The leaders will lead the other persons to fight the enemy. Apply the results of Task 2 to create a realistic effect.
- When the person contact with the enemy (apply Task 1), there will be a random chance that it is being hit. If it is being hit, increment the counter “iCountBeingHit” and create a movement effect for being hit.
- When a person is being hit for a specific number of times, it dies. Call “Universe.remove(this)” to remove the person in the scene.

Enhancement (30 Marks):

You are encouraged to apply any algorithm you know to create a more realistic scene. For example, you could generate a more intelligent person using a state machine, or model the behavior of the person such that it reacts like real human. You are also encouraged to create a more interesting scene with more objectives look more similar to a real war scene. For example, you could add defensive soldiers to defend a castle while the others are attacking.

Task 4: Written Report

Write one page of report to describe what you have done. Highlight and discuss any algorithm you have used for the project.

Acknowledgement

The program is modified based on the work of Rod McFarland:

<http://www.cs.ubc.ca/~van/cpsc526/Vjan2003/projects/mcfarland/index.html>