Systems Programming

2025/2026 - The Project – Part A

🗑️ Space Trash 🚀

In a parallel universe, various planets face the perils of the trash roaming the space. These planets face doom if trash reaches a high level.

To respond to this threat, each planet decided to build and launch a **trash ship** to collect the trash.

In this game, each user controls one of these **trash ships** to collect trash and deposit it safely for recycling.

1 The Universe

The universe is facing extinction due to the trash that follows a set of special rules (some natural, some other anthropogenic:

- The universe is a square;
- The number of planets is fixed, and each one is named with a single letter;
- Periodically, new trash is generated in the universe; (**PART B**)
- There are ships trying to collect trash;
- Collected trash should be deposited in the recycling planet.
- If a trash ship collides with a planet that is not the recycling, the whole trash cargo is spilled in the universe;
- When a trash hits a planet, new trash is generated;
- Trash is affected by gravity and has inertia;
- Trash ships are affected by gravity and inertia. (**PART B**)

This parallel universe is ruled by laws similar to those of our own universe, but some other para-physical rules also apply.

1.1 Physics rules

Only trash and **trash ships** are affected by gravitation force generated by the planets and the planets are stationary. The following constants should be used to calculate forces, acceleration, velocity, and location of trash:

- planet mass – 10 mass units
- trash mass – 1 mass units

- distance units – 1 pixel
- time unit – 10ms
- gravitational constant – 1
- trash friction – reduces velocity by 1% in every time unit

The code to calculate the gravitational force for each trash is the following:

```
void new_trash_acceleration(planet_stucture planets[], int total_planets,
                            trash_stucture trash[], int total_trash){
   vector total_vector_force;
   vector local_vector_force
   for (int n_trash = 0; n_trash < N_TRASH; n_trash ++){
      total_vector_force.amplitude = 0;
      total_vector_force.angle = 0;
      for (int n_planet = 0; n_planet < N_PLANETS; n_planet ++){
         float force_vector_x = planets[n_planet].x – trash[n_trash].x;
         float force_vector_y = planets[n_planet].y – trash[n_trash].y;
         local_vector_force = make_vector(force_vector_x, force_vector_y);
         local_vector_force.amplitude = planets[n_planet].mass /
                                        pow(local_vector_force.amplitude, 2);
         total_vector_force = add_vectors(local_vector_force,
                                        total_vector_force);
         trash[n_trash].acceleration = total_vector_force;
      }
   }
}
```

The velocity and position can then be computed using the following functions:

```
void new_trash_velocity(trash_stucture trash[], int total_trash){
   for (int n_trash = 0; n_trash < total_trash; n_trash ++){
     trash[n_trash].velocity.amplitude *= 0.99;
     trash[n_trash].velocity= add_vectors(trash[n_trash].velocity,
                                        trash[n_trash].acceleration);
   }
}

void new_trash_position( trash_stucture trash[], int total_trash){
   for (int n_trash = 0; n_trash < total_trash; n_trash ++){
      trash[n_trash].x += trash[n_trash].velocity.amplitude *
```

```
                                                cos(trash[n_trash].velocity.angle);
        trash[n_trash].y += trash[n_trash].velocity.amplitude *
                                               sin(trash[n_trash].velocity.angle);;
        correct_position(&trash[n_trash].x);
        correct_position(&trash[n_trash].y);
    }
}
```

By calling these 3 functions every 10ms, it is possible to simulate the basic gravitational mechanics of this universe.

1.2 Para-physics rules

When a trash or **trash ship** reaches the edge of the universe, it is teleported to the opposite side at the same speed (amplitude and direction).

Although planets have a radius of 20, trash can enter the interior of the planet, and continue moving as if the planet is a single point.

When the trash touches the center of the planet (distance between centers lower than 1.0), a new trash emerges in a random place of the universe.

When the amount of trash roaming the universe reaches a limit, the whole universe collapses and is destroyed.

1.3 Anthropic rules

Trash, ships, and recycling units also have some rules:

- Periodically, humanity generates trash that appears in random places in the universe. (**PART B**)
- Each planet has one trash ship that is operated by a human. These ships try to collect all roaming trash before the universe collapses.
- Trash is collected when the ship comes into contact with it.
- Ships have a limit on the capacity of collected trash. Before reaching that limit, collected trash needs to be deposited on the recycling planet.
- At any given time, only one planet can recycle trash collected by trash ships.
- If a trash ship hits a non-recycling planet, all of the trash cargo is spilled in the universe.
- The planet capable of recycling trash is randomly assigned and changes periodically. (**PART B**)

2 Project – Part A

In the first part of the project, students will implement two independent systems:

- simulator of the universe without trash ships.
- simulator of a **trash ship** maneuvering and operation.

The rules of the universe marked with **PART B** can not be implemented now.

2.1 Universe simulator

This application (called **universe-simulator)** should simulate the universe over time with the planets and roaming trash. The evolution of the universe (planets and trash location) is shown in a graphical window and calculated over time.

This application should calculate the new position for each trash every 10ms by applying the previously presented code.

Only the rules described earlier and not related to **trash ships** should be implemented:

- trash movements;
- trash collision with the planets.

As such, it is expected that this universe will be destroyed when it reaches the limit of trash.

Students should decide on the following:

- how to store the planets;
- how to store the trash;
- how to implement the simulation loop;
  - to apply the physics rules;
  - to draw the universe;
- how to implement the rules to generate more trash;
- how to verify the end of the universe.

Periodic trash generation should not be implemented.

2.2 Trash ship simulator

Students should implement a simple client-server system that allows the control of a single trash ship in a simple version of the universe:

- universe-client
  - opens a socket and connects to the server;
  - reads cursor keys from the keyboard;
  - sends movement orders to the server;
  - does not display universe;
- universe-server
  - creates a socket and receives client connections;
  - receives movement order from one client;

- updates universe;
- display universe (with planets, trash, and trash ships);
- does not calculate gravitational forces nor move trash.

The **universe-serve**r should implement all interactions between **trash ships** and trash, and planets:

- allow multiple clients/trash ships;
- move the trash ships and draw them;
- collect trash when a trash ship touches it;
- deposit trash in a recycling planet;
- spill the collected trash if the trash ship touches a non-recycling planet.

The **universe-server** should NOT implement the following:

- movement of the trash;
- change of the recycling planet.

In the **universe-server**, students should use exactly the same solution as in the **universe-simulator** for:

- planet storage (data structures);
- trash storage (data structures);
- representation of the planets and trash (functions to draw the universe).

But, they must decide on:

- sockets to be used;
- messages' structure;
- storage of the trash ships;
- representation of the trash ships;
- operation of the trash ships (movement and trash collection, deposition, and spill).

2.3 Universe configurations file

The two versions of the universes to be implemented (**universe-server** and **universe-simulator**) can be configured. These parameters should not be hard-coded but provided as a configuration file with the following information:

- Dimensions of the universe;
- Number of planets;
- Maximum number of trash before the universe collapses;
- Initial number of trash;

- Capacity of the trash ships.

3 Software architecture

Students should try to implement layers on this project and data abstraction. As such, for the universe simulator, there should be the following layers and C files.

Universe-simulator-main	Universe-simulator.c
display processing	display.c display.h
Gravitational rules	physics-rules.c physics-rules.h
Data management	universe-data.c universe-data.h

The **ship-simulator** will have a new layer but will reuse some layers from the universe-simulator

universe-client-main	universe-server-main	Universe-client.c Universe-server.c
Keyboard processing	Display processing	Display.c Display.h
Communication	Communication	Universe-data.c Universe-data.h
	Data management	Communication.c Communication.h

4 Development technologies

Besides Standard C, students should only use the following extra libraries:

- **SDL2** for displaying the universe (universe-simulator and universe-server) and reading the cursor keys (**universe-client**);
- **libconfig++** for reading the configuration files; (documentation provided in **libconfig** folder)
- **ZeroMQ TCP** socket for communication between client and server;
- **protocol-buffers** for the encoding of messages.

Students should implement the system using the C language **WITHOUT** using the following:

- threads;
- select;
- non-blocking communication;
- active wait;
- signals.

5 Order of development

For success in the development (higher grade :), better results, and more learned knowledge, it is advised that users follow the presented order of development:

1. Universe simulator:
    a) Read the configurations;
    b) Initialize SDL and create a window;
    c) Implement a simple loop (Quit event and SDL_Wait);
    d) Define the planets' data structures;
    e) Create the planets in the universe;
    f) Draw the planets;
    g) Define the trash data structures;
    h) Create the trash in the universe;
    i) Draw the trash without moving it;
    j) Add gravitational force, acceleration, speed, and position of trash;
    k) Implement the crash of trash on the planets;
    l) Verify the end of the world.
2. Trash ship simulator:
    a) Create a simple client that reads the cursor keys;
    b) Create a simple server by stripping the universe simulator of all trash movement (there should be planets and immobile trash);
    c) Define the ZMQ sockets;
    d) Implement a simple protocol between the client and server, including all sends and receives, but with no information exchanged;
    e) Define the protocol-buffer messages;
    f) Implement the encoding and encoding of the messages;

g) Implement the addition of a new player (assignment of a ship);

h) Implement the drawing of ships;

i) Implementation of simple ship movement;

j) Implementation of the collection of trash by ships;

k) Implementation of the deposition of trash on the recycling planet;

l) Implement the ship's crash into other planets and the spill of trash throughout the universe.

6 Error treatment / Cheating

When implementing a distributed/network-based system, servers cannot guarantee that clients will lawfully adhere to the defined protocol.

If the communication protocol permits it, malicious programmers can exploit the devised messages and interactions for cheating.

In addition to verifying all messages received on the server to detect communication errors, the protocol and data exchanged between clients and the server should ensure that a malicious client cannot cheat by subverting the semantics and order of messages.

Here, we are not addressing hacking concerns that could be solved using cryptography. The code must ensure that programmers with a C compiler and knowledge of the protocol cannot disrupt the game (for instance, by moving other trash ships, …).

7 Project submission

The deadline for submitting part A of the project will be **Friday, 5th December at 19h00** on FENIX.

Before submission, students should create the project group and register at FENIX.

Students should submit a unique **zip** file containing the code for all the components with two different folders:

- One for the universe simulator;
- Another for the trash ship simulator (client and server).

The students should also provide multiple Makefiles to compile the various programs.

8 Project evaluation

The grade for this project will be given taking into consideration the following:

- Number of functionalities implemented;
- Communication;
- Code structure and organization;

- Error validation and treatment;
- Cheating robustness;
- Comments.