Particle Simulator: Explorer Mode
STDISCM Problem Set 2 Specifications
Term 2 AY 2023 - 2024

Ryan Austin Fernandez
2401 Taft Avenue
Manila, Philippines
ryan.fernandez@dlsu.edu.ph

ABSTRACT
This document details the specifications of the second problem set in distributed computing. General instructions are provided for an explorer mode augmentation to the first problem set, which will be implemented using concurrent programming.

Author Keywords
concurrent programming, load balancing

CCS Concepts
•**Computing methodologies** $\rightarrow$ **Shared memory algorithms; Concurrent algorithms;**

GENERAL PROGRAMMING INSTRUCTIONS
This project is intended for 3 - 4 members. You are to create an "explorer mode" augmentation to your particle physics simulator from problem set 1 using either C++ or Java. The interface of problem set 1 will henceforth be known as "developer mode." In developer mode, a user should be able to add particles to the canvas. A user can no longer add walls.

In "explorer mode", the user spawns a sprite that can travel around the space. When rendering the canvas, ensure the frame rate does not drop while updating the environment.

RESTRICTIONS
Only either C++ or Java and their threading support library and UI libraries should be used in this problem set.

Other libraries or game engines could be used. You may request approval in our Discord server.

SPECIFIC INSTRUCTIONS
The program begins in "developer mode." A user should be able to add particles to the environment with an initial position (x, y), an initial angle Θ (0 degrees is east, and degrees increase in an anticlockwise manner, e.g., 90 degrees is north), and a velocity V (in pixel per second). The particle will travel in a straight line, bouncing off the four walls of the canvas. Particles do not collide with other particles. All collisions are elastic, which means particles do not slow down or speed up after a collision.

The canvas should be $1280x720$ pixels. Coordinate $(0,0)$ is the southwest corner of the canvas. Coordinate $(1280,720)$ is the northeast corner.

Particles can be added in batches. This is in three forms:

- Provide an integer n indicating the number of particles to add. Keep the velocity and angle constant. Provide a start point and end point. Particles are added with a uniform distance between the given start and end points.
- Provide an integer n indicating the number of particles to add. Keep the start point and velocity constant. Provide a start Θ and end Θ. Particles are added with uniform distance between the given start Θ and end Θ.
- Provide an integer n indicating the number of particles to add. Keep the start point and angle constant. Provide a start velocity and end velocity. Particles are added with a uniform difference between the given start and end velocities.

The user can also switch to "explorer mode." A sprite will spawn in the space (at a location of your choosing), and the user can control this sprite using onscreen arrow keys, WASD keys, or arrow keys. The canvas used for viewing developer mode will now transform into a zoomed in version of the sprite's periphery. The dimensions of the periphery are 19 rows by 33 columns, with the sprite rendered in the very center. This leaves 16 columns of pixels to the sprite's left and right and 9 rows above and below the sprite to render. As the sprite moves around the space, any balls that enter its periphery must be rendered in motion on the screen until they leave the periphery.

For "developer mode," ensure your screen resolution is high enough to show all particles on-screen. For "explorer mode," you only need to ensure you can display all the particles in the sprite's periphery. Show the FPS counter on-screen every 0.5 seconds for both modes.

You are also to write a **1 - 2 page technical report** containing the following:

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

- Discuss how you implemented the dependencies of the sprite's periphery and the rendering of the particles within the periphery.
- Provide code snippets supporting your technical explanation.

The technical report must have one-inch margins and be typeset in Calibri 11-point font. **DO NOT** exceed two pages. Any pages after the second page will be ignored. At the end of the technical report, provide a record of contribution for each major activity for the case study.

1. Each row represents an activity, and each cell represents the percentage each member contributed to that activity. Each row's contributions must total 100.
2. The **TOTAL** row must be normalized to add up to 100.

Example:

Group Members:

- P1: Percy Jackson
- P2: Annabeth Chase
- P3: Grover Underwood

Activity	**P1**	**P2**	**P3**
Topic Formulation	50.0	25.0	25.0
Developer Mode UI	33.3	33.3	33.3
Explorer Mode UI	25.0	50.0	25.0
Periphery Rendering and Dependencies	10.0	10.0	80.0
Raw Total	118.3	118.3	163.3
TOTAL	**29.6**	**29.6**	**40.8**

Have each member sign below the table above their full name and the date.

GRADING RUBRICS

Grading rubrics will be posted on Canvas.

FINAL DELIVERABLES AND DEADLINE

- A 1 - 5 min video demo showing the sample program in MP4.
- Technical report
- Submit a GitHub/GDrive link containing your source code. Include a README.md file for instructions on how to run your program.
- For C++, the program should be compatible with VS 2019 (x64) and C++ 20 compiler and ready for compilation. The program should already include any external dependencies.
- For Java, include a JAR file to be run.

Deadline for submission of all deliverables: March 22, 2024, 11:59 PM

REFERENCES

[1] Allen B Downey. 2005. *The little book of semaphores*.

[2] Maurice Herlihy, Nir Shavit, Victor Luchangco, and Michael Spear. 2020. *The art of multiprocessor programming*. Newnes.

[3] Maarten Van Steen and Andrew S Tanenbaum. 2017. *Distributed systems*. Maarten van Steen Leiden, The Netherlands.