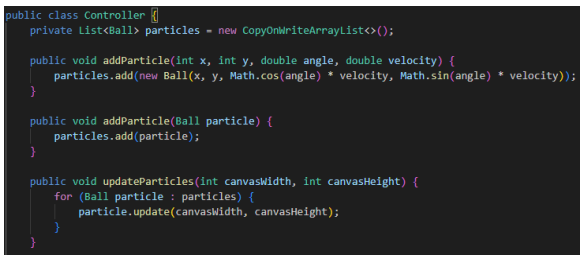
STDISCM - Technical Report

Message passing was implemented through a multi-threaded approach using the ThreadController class which manages multiple threads each handled by a particleprocessor, responsible for a subset of particles running concurrently to update their states. updateParticles triggers parallel execution using a ForkJoinPool.



```
public class Controller {
    private List<Ball> particles = new CopyOnWriteArrayList<>();

    public void addParticle(int x, int y, double angle, double velocity) {
        particles.add(new Ball(x, y, Math.cos(angle) * velocity, Math.sin(angle) * velocity));
    }

    public void addParticle(Ball particle) {
        particles.add(particle);
    }

    public void updateParticles(int canvasWidth, int canvasHeight) {
        for (Ball particle : particles) {
            particle.update(canvasWidth, canvasHeight);
        }
    }
```

The system dynamically adjusts the number of threads based on the processing load, adding new processors and redistributing particles as needed. This is to ensure efficient parallel processing and maintain the performance in real time simulations.

```
public void checkAndAdjustThread() {
    if (shouldAddThread()) {
        redistributeParticles();
    }
}

private boolean shouldAddThread() {
    if (processingTimesHistory.isEmpty()) {
        return false; // Skipping if not enough data or particles
    }

    long currentAverageProcessingTime = processingTimesHistory.get(processingTimesHistory.size() - 1);
    boolean processingTimeIncreasing = currentAverageProcessingTime > lastAverageProcessingTime;
    boolean significantParticleIncrease = particleSize >= lastParticleSizeAtThreadAddition * 1.10;

    return processingTimeIncreasing &&
        processors.size() < Runtime.getRuntime().availableProcessors() &&
        significantParticleIncrease;
}
```

To achieve and sustain a consistent 60 frames per second (FPS) during client-side sprite rendering, our system employs precise frame rate monitoring, optimized rendering loops, and effective thread management. The FPS class monitors and displays the frame rate, using System.nanoTime() to track frame intervals and update the FPS count every second. This updated FPS count is rendered within the paintComponent method of the MainScreenUI class, ensuring continuous and real-time updates during each repaint cycle. Furthermore, the ThreadController class manages the main game loop within a dedicated thread, effectively separating it from the primary application flow. This loop targets a 60 FPS rate by calculating the time difference between frames and dynamically adjusting the sleep duration using Thread.sleep(2). This approach ensures the synchronization of game logic and rendering processes, maintaining smooth and consistent visual performance.

Here we have the StartBroadcaster() function. This function simply checks for updates in the explorers as well as the particles then sends these updates to the explorer clients. Each ExplorerClient handles its own Explorer which sends its state to the other Explorers.

```
private void startBroadcaster() {
    new Thread(() -> {
        while (true) {
            List<String> particleMessages = getDeltaParticleData();
            List<String> explorerMessages = getDeltaExplorerData();

            for (String particleMessage : particleMessages) {
                broadcast(particleMessage);
            }

            for (String explorerMessage : explorerMessages) {
                broadcast(explorerMessage);
            }

            try {
                Thread.sleep(10); // Reduce to 10ms or an appropriate value
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }).start();
}
```

```
private List<String> getDeltaExplorerData() {
    List<String> deltaMessages = new ArrayList<>();
    for (ClientHandler client : clients) {
        Explorer explorer = client.getExplorer();
        String newState = "EXPLORER: " + explorer.getId() + ", " + explorer.getX() + ", " + explorer.getY();
        if (!newState.equals(lastExplorerState.get(explorer.getId()))) {
            deltaMessages.add(newState);
            lastExplorerState.put(explorer.getId(), newState);
        }
    }
    return deltaMessages;
}
```

On the client side, we have the handleServerMessage() function which handles messages from the server and updates its respective explorer's state.

```
private void handleServerMessage(String message) {
    System.out.println("Received message: " + message);

    String[] parts = message.split(": ");
    if (parts.length == 2) {
        if (parts[0].equals("CONTROLLED_EXPLORER")) {
            controlledExplorerId = parts[1];
        } else {
            String[] data = parts[1].split(", ");
            if (data.length == 3 && parts[0].equals("EXPLORER")) {
                String id = data[0];
                double x = Double.parseDouble(data[1]);
                double y = Double.parseDouble(data[2]);
                explorers.put(id, new Explorer(id, x, y));
            } else if (data.length == 5) {
                String id = data[0];
                double x = Double.parseDouble(data[1]);
                double y = Double.parseDouble(data[2]);
                double vx = Double.parseDouble(data[3]);
                double vy = Double.parseDouble(data[4]);
                particles.put(id, new Ball(id, (int) x, (int) y, vx, vy));
            }
        }
    }
    explorerPanel.repaint();
}
```

Group 4:
[P1] Russel Campol, **[P2]** Andres Clemente,
[P3] Paolo Flores, **[P4]** Mel Geoffrey Racela, and
[P5] Kenn Michael Villarama 08/08/2024

Activity	**P1**	**P2**	**P3**	**P4**	**P5**
Topic Formulation	25	25	25	12.5	12.5
Developer Mode UI	25	25	25	12.5	12.5
Explorer Mode UI	25	25	25	12.5	12.5
Client-Server Communication	5	5	5	42.5	42.5
Raw Total	80	80	80	80	80
TOTAL	20	20	20	20	20

References

[1] Harischandra, G. (2023, November 10). Java Fork-Join pool - Gathila Harischandra - Medium. *Medium*. https://medium.com/@gathilaharism/java-fork-join-pool-with-an-example-320a0d3d5b4c

[2] Stack Overflow. (2009). Calculating frames per second in a game. *Stack Overflow*. https://stackoverflow.com/questions/87304/calculating-frames-per-second-in-a-game

[3] Stack Overflow. (2012, July 20). *How to make a rendering loop run at a consistent rate?* Stack Overflow. https://stackoverflow.com/questions/11552717/how-to-make-a-rendering-loop-run-at-a-consistent-rate