STDISCM - Technical Report

1. Dependencies of the sprite's periphery

In the particle simulator, the "sprite's periphery" denotes the visible area surrounding the explorer sprite on the screen. This periphery dynamically adjusts based on the explorer's position, consistently displaying nearby particles.

The simulator computes the dimensions and position of this periphery relative to the explorer's current coordinates on the canvas. As the explorer navigates the environment, the simulator continuously recalculates and modifies the viewport, ensuring that particles within this vicinity remain visible. This dynamic adjustment facilitates an enhanced exploration and interaction experience, as it consistently presents the particles in proximity to the explorer. The simulator leverages concurrent programming techniques to manage and render these elements efficiently [1].

```
public class Explorer {
    private double x, y;
    private double vx, vy;
    private static final int SIZE = 10; // Adjusted size for a smaller sprite

    public Explorer(int x, int y) {
        this.x = x;
        this.y = y;
        this.vx = 0;
        this.vy = 0;
    }

    public void update(int canvasWidth, int canvasHeight) {
        x += vx;
        y += vy;

        if (x < 0) x = 0;
        if (x > canvasWidth - SIZE) x = canvasWidth - SIZE;
        if (y < 0) y = 0;
        if (y > canvasHeight - SIZE) y = canvasHeight - SIZE;
    }

    public void draw(Graphics g) {
        g.setColor(Color.BLUE);
        g.fillOval((int) x, (int) y, SIZE, SIZE);
    }
```

The update method in the Explorer class ensures that the explorer's position is updated based on its velocity and keeps it within the canvas boundaries. This method increments the explorer's position (x and y) by its velocity (vx and vy). It then checks if the explorer has moved outside the canvas boundaries and adjusts its position to ensure it remains within the valid range (0 to canvasWidth - SIZE for the x-coordinate, and 0 to canvasHeight - SIZE for the y-coordinate).

As for the user input for the explorer's movement, the handleKeyPress method in MainScreenUI handles user input to move the explorer and updates its velocity based on the arrow key pressed:

```
private void handleKeyRelease(KeyEvent e) {
    if (explorerMode) {
        Explorer explorer = threadManager.getExplorer();
        if (explorer != null) {
            switch (e.getKeyCode()) {
                case KeyEvent.VK_UP:
                case KeyEvent.VK_DOWN:
                    if (explorer.getVelocityX() == 0) explorer.setVelocity(vx:0, vy:0);
                    else explorer.setVelocity(explorer.getVelocityX(), vy:0);
                    break;
                case KeyEvent.VK_LEFT:
                case KeyEvent.VK_RIGHT:
                    if (explorer.getVelocityY() == 0) explorer.setVelocity(vx:0, vy:0);
                    else explorer.setVelocity(vx:0, explorer.getVelocityY());
                    break;
            }
        }
    }
}
```

This method checks if the 'E' key is pressed to toggle explorer mode. If explorer mode is active and the explorer exists, it updates the explorer's velocity based on the arrow key pressed (VK_UP, VK_DOWN, VK_LEFT, VK_RIGHT), enabling the explorer to move in the specified direction.

Additionally, the dynamic viewport calculation ensures that the viewport remains centered around the explorer by adjusting the viewport dimensions. The drawParticlesInView method in the Controller class renders particles that fall within this dynamically calculated viewport, ensuring efficient rendering and maintaining focus on the particles near the explorer.

2. Rendering of the particles within the periphery

The particle simulator only draws particles that are within the visible area (viewport) around the explorer. This ensures that the simulator

uses computer resources efficiently while keeping the simulation responsive and visually clear.

Every frame, the simulator checks where each particle is in relation to the viewport. It only draws particles that are within this area on the screen. This helps the simulator run smoothly and avoids using too much computing power. This way of drawing particles not only makes the simulator respond faster but also helps users focus on interactions nearby where they're exploring. The drawParticlesInView method in the Controller class is responsible for drawing particles that are within the calculated viewport:

```
public void drawParticlesInView(Graphics g, int canvasHeight, int viewLeft, int viewTop, int viewRight, int viewBottom) {
    for (Ball particle : particles) {
        if (particle.getX() >= viewLeft && particle.getX() <= viewRight && particle.getY() >= viewTop && particle.getY() <= vi
            particle.draw(g, canvasHeight);
        }
    }
    if (explorer != null) {
        explorer.draw(g);
    }
}
```

This method iterates through the list of particles and checks if each particle's position (particle.getX() and particle.getY()) falls within the specified viewport boundaries (viewLeft, viewTop, viewRight, viewBottom). If a particle is within the viewport, it is drawn on the canvas. The explorer is also drawn if it exists.

In the drawExplorerMode method within MainScreenUI.java, the viewport is dynamically calculated based on the explorer's position. This involves defining the grid size and calculating the viewport's dimensions relative to the explorer's current location whilst applying a zoom based on the scaling of the particles:

```
private void drawExplorerMode(Graphics2D g2d) {
    Explorer explorer = threadManager.getExplorer();
    Color saiyanBlue = new Color(r:173, g:216, b:230, a:255);
    g2d.setColor(saiyanBlue);
    if (explorer != null) {
        int explorerX = (int) explorer.getX();
        int explorerY = (int) explorer.getY();

        // Define the grid size based on the spec (19 rows by 33 columns)
        int cellSize = 38; // Adjusted cell size
        int gridColumns = 33;
        int gridRows = 19;
        int gridWidth = gridColumns * cellSize;
        int gridHeight = gridRows * cellSize;
        int halfGridWidth = gridWidth / 2;
        int halfGridHeight = gridHeight / 2;

        // Calculate viewport dynamically
        int left = explorerX - halfGridWidth;
        int top = explorerY - halfGridHeight;

        // Determine dynamic scaling based on explorer position
        double scale = 2.0; // Example scaling factor for zooming in

        // Calculate the translation to center the explorer
        int panelWidth = getWidth();
        int panelHeight = getHeight();
        int translateX = (int) ((panelWidth / scale) / 2 - explorerX);
        int translateY = (int) ((panelHeight / scale) / 2 - explorerY);

        // Apply scaling and translation
        g2d.scale(scale, scale);
        g2d.translate(translateX, translateY);
```

This ensures that the scaling and translation are applied to center the viewport around the explorer. After this, the drawParticlesInView method is called to draw the particles within the calculated viewport boundaries, followed by drawing the explorer itself

Group 4:
[P1] Russel Campol, **[P2]** Andres Clemente, **[P3]** Paolo Flores, **[P4]** Mel Geoffrey Racela, and **[P5]** Kenn Michael Villarama 07/12/2024

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
Periphery Rendering & Dependencies	5	5	5	42.5	42.5
Raw Total	80	80	80	80	80
TOTAL	20	20	20	20	20

References

[1] Lea, D. (2000). "Concurrent Programming in Java: Design Principles and Patterns." Addison-Wesley.