The Circulatory System Development Report

The **Circulatory System Simulation** is an interactive tool that demonstrates how the **circulatory, respiratory, and digestive systems** work together to transport oxygen and glucose to cells and remove carbon dioxide. Students can explore **blood circulation, oxygen exchange in the lungs, and the role of the heart in maintaining cellular function**. The simulation allows them to observe how changes in **heart rate and breathing rate** impact oxygen delivery and overall system efficiency.

How the Simulation Works:

The **Circulatory System Simulation** features a **flexible system** that dynamically controls the **movement and transition of particles** between organs based on the rules set by students. This allows for an interactive learning experience where students can define how substances like **oxygen, glucose, and carbon dioxide** circulate through the body.

To ensure **progressive learning**, not all objects are shown at once. Instead, components can be **gradually introduced** as the lesson progresses. For example:

- Initially, only **blood particles and oxygen molecules** are visible, demonstrating how **oxygen binds to blood cells and is transported**.
- Later, additional elements like **glucose, carbon dioxide, and other metabolic byproducts** can be introduced, allowing students to observe their role in cellular respiration and system interactions.

This structured approach ensures that students **build their understanding step by step**, reinforcing key biological concepts through **interactive exploration**.

To make the simulation accessible to a wider audience, all in-game text has been translated into multiple languages, including:
- English (default language).
- Portuguese (standard left-to-right adaptation)

- Hebrew (LTR, right-to-left adaptation).
- Arabic (RTL, using ArabicSupport for proper text rendering).
When switching to **Arabic**, the system correctly adjusts text alignment, word order, and UI layout, ensuring proper readability and usability.

The simulation was developed using the following technologies:

- **Unity 2021.3.30 (LTS)** – main game engine.
- **C#** – scripting language for logic implementation.
- **TextMeshPro** – for rendering high-quality text descriptions.
- **Unity WebGL** – for web deployment.
- **Optimized PNG textures** (RGBA Crunched DXT5, BS3) – used to reduce build size while maintaining image quality.
- **Sprite-based animations** – used for visual effects such as animation of heart and lungs and animation of eating an apple.
- **Custom Shader (Shader Graph & HLSL)** – simulates energy consumption in muscle tissue through a dynamic liquid effect.
- **External API Communication** – The simulation interacts with an external API written in TypeScript.
  Uses System.Runtime.InteropServices to send messages between C# (Unity) and JavaScript/TypeScript.
  Receives instructions from the API, allowing dynamic simulation behavior based on user-defined parameters.
- **Optimized Particle Management** – Implemented Dictionary<TKey, TValue>, **HashSet<T>, and Object Pooling to handle a large number of particles efficiently.

The main interactive features of the simulation include:

The simulation includes several interactive features that allow students to explore how the body responds to different activities and energy consumption. Key interactive elements include:

- **Activity Selection Buttons** – Students can switch between **walking, running, and resting**, observing how each state affects oxygen consumption, glucose usage, and muscle energy.
- **Interactive Eating Mechanic** – A button allows students to **consume an apple**, adding glucose to the body.
- **Glucose Toggle Mode** – Instead of manually clicking the apple each time, students can enable a **continuous glucose intake mode**, where glucose is automatically supplied, and the apple animation loops without repeated input.

This system provides an **engaging, hands-on approach** to understanding how energy intake and physical activity influence metabolism.

Challenges and Solutions

1. Managing a Large Number of Particles and Their Transitions

Problem:
The simulation involves **a large number of particles** moving between multiple locations, requiring a **clear structure** to track their **positions, transitions, and interactions** efficiently. Without proper management, tracking which particles exist in which locations would become complex and error-prone.

Solution:
To ensure **efficient data handling**, the following optimized data structures were implemented:

- **Dictionary<TKey, TValue>** – Used for **quick lookup and categorization** of particles in different locations.
- **HashSet<T>** – Utilized for **fast existence checks and efficient state management**.
- **Object Pooling** – Implemented to **reuse particle instances** instead of frequently instantiating and destroying them, reducing **memory allocation overhead** and improving performance.

This approach significantly optimized **particle management**, ensuring smooth transitions and real-time updates.

2. Visual Optimization for Better Readability

Problem:
Since the simulation displays **a large number of particles at once**, excessive visual clutter made it difficult for students to focus on key elements.

Solution:

- **Adjustable Transparency Levels** – Some particles were rendered with **high transparency** to maintain visibility **without overwhelming the scene**.
- **Layering Techniques** – Important particles remain **fully visible**, while background particles appear **less prominent**, creating a clearer distinction.

These optimizations improved **visual perception**, making the simulation more **intuitive and accessible** without sacrificing complexity.

3. Reducing WebGL Build Size Without Losing Quality

Problem:
The initial simulation build, using **3D objects for particles**, resulted in a **30 MB WebGL build**, which was too large for smooth web deployment.

A more optimized approach was needed to maintain visual fidelity while significantly reducing file size.

Solution:

- **Switched from 3D models to animated sprites**, dramatically reducing asset size.
- **Optimized texture compression** to further decrease file weight.
- **Final Build Size Reduction:**
    - **Before optimization:** 30 MB (with 3D objects).
    - **After switching to sprite animations and optimizations: 4 MB**.

This resulted in a **7.5x reduction in build size**, making the simulation **fast and lightweight for web browsers** while maintaining high visual quality.

Conclusion:

The **Circulatory System Simulation** effectively combines **interactive learning, real-time particle tracking, and optimized visual representation** to enhance student engagement.

Through **efficient data structures** (**Dictionary, HashSet, Object Pooling**), the simulation manages **large-scale particle transitions smoothly**, ensuring **high performance in WebGL**. Visual optimizations, such as **adjustable transparency and layering**, improve readability, preventing clutter.

A key achievement was **reducing build size from 30 MB to 4 MB** by replacing **3D models with animated sprites**, making the simulation **fast, lightweight, and accessible for web deployment**.