Plant Cell Simulation Development Report

This development report describes the process of creating the **Plant Cell Simulation**, a Unity-based interactive educational tool designed to help students understand the structure and functions of plant cells.
The simulation visualizes key organelles, allowing students to interact with the cell and observe how different factors affect cellular processes.

How the Simulation Works:

The simulation is fully interactive and runs **directly in a web browser**. Students can define their own rules and input parameters, which dynamically affect how the simulation behaves.

To enhance the learning experience, the simulation offers three levels of visualization:
Macro View – Displays multiple flower cells, helping students understand how different plant cells interact.
Micro View – Focuses on a single cell, allowing a detailed examination of internal processes such as vacuole expansion and cellular respiration.
Structure View – where students assemble a cell piece by piece.

To make the simulation accessible to a wider audience, all in-game text has been translated into multiple languages, including:
- English (default language).
- 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 cell growth and structural changes.
- **Custom Shader** (Shader Graph & HLSL) – used to create a realistic liquid effect inside the cell.
- **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.

The main interactive features of the simulation include:

- The simulation includes various interactive mechanics that allow students to explore plant cell functions in a dynamic way:
- Water Absorption Mechanic – Demonstrates vacuole expansion when water is added. The vacuole grows and changes in size, affecting other cellular components.
- Cellular Respiration Visualization – Highlights Mitochondrion activity when oxygen and glucose are introduced.
- Cell Wall & Membrane Structure Adaptation – The cell wall and membrane dynamically change their structure based on vacuole size. When the vacuole expands due to water absorption, the cell

membrane stretches, and the cell wall slightly deforms, demonstrating turgor pressure in plant cells.

Challenges and Solutions

1. Performance Issues in WebGL (Optimization)

Problem:
When deploying to **WebGL**, performance dropped significantly due to the **high GPU load caused by complex shaders and textures**.

Solution:

- Optimized **PNG textures using the RGBA Crunched DXT5 (BS3) format**, which **reduced the build size by 30%** without noticeable quality loss.
- Replaced **heavy dynamic shaders** with **pre-baked textures**, reducing GPU workload.

2. Cell Animation and Structural Changes (Visual Effects)

Problem:
When animating **vacuole expansion**, the **cell membrane and wall did not properly adapt** to the changes in size.

Solution:

- Used **sprite-based animations** for **smooth transitions between different cell states**.
- Linked **vacuole expansion** with **cell wall stretching** to visually **demonstrate turgor pressure**.

3. Macro & Micro View Transitions (UX Challenge)

Problem:
Switching between **Macro View (multiple cells) and Micro View (single-cell focus)** was not intuitive, causing users to lose orientation.

Solution:

- Implemented **a configuration file** allowing users to **choose the preferred view mode** before starting the simulation.
- Maintained **a unified UI layout** to ensure **easy transitions between modes**.
- Added **smooth animation effects** between Macro and Micro View to create **a more natural transition experience**.

Conclusion

These challenges helped refine the **Plant Cell Simulation**, making it **more interactive, optimized, and user-friendly for educational purposes**.

The integration of **improved WebGL performance, multi-language support, advanced animations, and dynamic UI transitions** resulted in an **efficient and adaptable learning tool**.