



Software Engineering Department
Braude College of Engineering

Capstone Project Phase B

Plant Growth Dynamics: A Learning Experience

25-1-D-9

GitHub Repository: [Plant-Growth-Dynamics-A-Learning-Experience](#)

Advisor: **PhD. Sulamy Moshe**
moshesu@braude.ac.il

Students:

Amit Benita	-	amitbenita260996@gmail.com
Andy Lewis Sapner	-	Andy.Lewis.Sapner@e.braude.ac.il

Table of Contents:

Abstract.....	3
1. Introduction.....	3
2. Related Work	4
3. General Description	5
3.1 Background and Purpose of the System.....	5
3.2 System Goals.....	5
3.3 How the System Works	6
3.4 System Implementation – Technological Structure	6
3.5 Target Users	7
4. Solution Description	8
4.1 Project Structure	8
4.2 Menu Scene	9
4.3 Introduction Scene	9
4.4 Game Scene	10
5. Development Process	11
6. Tools Used in the Project.....	11
6.1 Unity Engine.....	11
6.2 The C# Programming Language.....	12
6.3 Java programming language and Spring Boot	12
6.4 Amazon DynamoDB	12
6.5 Amazon EC2	13
7. Challenges	13
7.1 Working with the Unity Engine and C#.....	13
7.2 Creating a Server Using Amazon EC2.....	14
7.3 Integrating the Server with the Client	15
8. Results and Conclusions.....	15
9. Lessons Learned	16
10. Meeting the Project Goals	17
11. User Guide	18
11.1 Description.....	18
11.2 Application’s Flow.....	18
11.2.1 Menu Scene	18

11.2.2 Introduction Scene	19
11.2.3 Game Scene.....	22
12. Maintenance Guide	33
12.1 Client Side - Programs.....	34
12.2 Server Side – Programs.....	34
12.3 Client Side – Key Files.....	36
12.4 Server Side – Key Files	37
12.5 Other Notes	38
13. References	38

Abstract

“Plant Growth Dynamics: A Learning Experience” is an educational game that teaches plant care and growth. Developed with the Unity engine, it uses real-time weather data via an API. Players grow plants in a simulated environment, learning how factors like temperature, humidity, and light affect growth. They manage plant care, receive feedback on plant health, and monitor weather impacts. Players can also upload real plant images for identification, disease detection, and integration into the game. These images can also be used for modifying the state of a plant. The goal is to teach proper plant care before or while applying it in real life.

1. Introduction

In recent years, people have become more aware of the importance of protecting the environment and living sustainably. Many are interested in growing plants at home, both as a hobby and to reconnect with nature. However, this process is not always simple it requires knowledge of different plant species, environmental factors, and proper care techniques. Without this understanding, plants may not grow properly or may become vulnerable to diseases.

Although people often search online for information about how to care for their plants, static resources like articles or videos do not always provide the interactive experience or personalized feedback needed to succeed in real life.

Our game provides a unique and practical solution to these challenges. It is a virtual plant-growing simulation that mimics real-world conditions. The system offers a fun, interactive, and educational environment where users can learn how to grow and care for plants effectively without taking any real-life risks.

The game integrates various environmental factors such as weather conditions, soil quality, and water availability, creating a dynamic and realistic plant-care experience. It also uses gamification elements, such as earning points and tracking progress to keep players engaged and motivated to improve their knowledge and skills.

Designed for users of all ages and backgrounds, the game is also suitable for people who have no prior experience with gardening. Players can even upload photos of real plants they own (or used to own) and receive care recommendations based on the status of the plant.

Through an intuitive user interface, players can perform real-time actions such as watering, fertilizing, and selecting different environments (greenhouse, garden, or indoor space). The system responds instantly to each action and provides clear explanations about how those actions affect plant growth.

In addition to being an entertaining game, our system can also serve as a powerful educational tool in classrooms, nature programs, or environmental workshops. It supports real-time weather data, disease detection using ML, and

cloud-based progress tracking, making it a technologically advanced learning experience.

By giving players control over plant growth and offering immediate feedback on their choices, the game encourages a sense of responsibility, exploration, and success while helping build environmental awareness in a modern, engaging way.

2. Related Work

One approach to plant learning is with mobile apps for plant identification and care. For example, PlantSnap [1] is a popular app that helps users identify plants using their smartphone cameras. Once identified, the app provides users with basic information about the plant, such as its care needs and environmental preferences. While this app is useful for plant identification, it lacks interactive elements and does not provide a simulation of the plant growth process.

Plant identification using computer vision has been a widely researched topic in recent years. Studies such as those by Wäldchen and Mäder [2] demonstrate the potential of deep learning models, including convolutional neural networks (CNNs), to accurately classify plant species based on image data. This type of technology is a cornerstone for modern plant-related applications like PlantSnap and serves as the basis for similar features in our project.

Another approach focuses on garden management. Gardenia [3] is a gardening app designed to help users plan and manage their gardens by tracking their plants, reminding them of watering schedules, and offering general care tips. However, the app does not simulate real-time weather impacts or allow users to experiment with virtual plant growth, limiting its educational potential for beginners.

Gardening management and simulation tools have also been explored in academic research. For instance, studies such as those by Taylor et al. [4] emphasize the importance of integrating real-time data into virtual environments for gardening simulations. These findings support the need for advanced tools like our Plant Growing Learning System, which incorporates real-time weather data to enhance educational impact.

Plant disease diagnosis has also been explored in apps like Plantix [5], which targets farmers by offering tools to identify plant diseases through photos. This app is highly effective for agricultural purposes but is not designed for learning or experimentation. It also does not offer a fun or engaging way for users to practice plant care in a low-risk environment.

Research into plant disease diagnosis, such as the work of Mohanty et al. [6], highlights the effectiveness of image-based deep learning models in detecting diseases across a wide variety of crops. This has been a significant

advancement for agricultural technology and provides inspiration for features in our project, such as disease recognition through uploaded images.

Unlike these existing tools, our Plant Growing Learning System offers a unique combination of features. By integrating real-time weather data, detailed plant care simulations, and the ability to upload images of real plants for conversion into virtual plants, our system bridges the gap between passive learning and interactive experimentation. The goal is not just to provide information but to allow users to actively engage with the growth process in a realistic and enjoyable way.

Computer games are an excellent way to teach plant care because they are interactive, engaging, and provide instant feedback [7]. In our game, players can experiment with different actions, such as watering schedules or plant placements, and immediately see how these decisions affect plant health and growth. This firsthand experience is crucial for learning, as it allows users to evaluate and refine their skills without real-world consequences.

Furthermore, studies such as those by Gee [8] emphasize the educational value of gamified learning systems in building practical skills. Gamification not only motivates users but also enhances retention and understanding through interactive problem-solving scenarios.

In addition, the use of gamified elements, like growth stages and rewards for healthy plants, keeps players motivated and ensures that learning is both fun and effective. By using a game, we can make plant care accessible to people of all ages and skill levels, preparing them to grow plants successfully in the real world.

3. General Description

3.1 Background and Purpose of the System

The Plant Growth Learning System is an interactive simulation developed in Unity that teaches users how to grow and care for plants in a fun and educational way. The system combines real science and technology with a game-like experience, making it easy for players to learn by doing.

3.2 System Goals

The main goals of the system are:

- To help users understand how light, temperature, humidity, and fertilizer affect plant growth.
- To teach responsibility and observation by letting players care for their own virtual plants.
- To introduce basic scientific thinking through firsthand experience.

- To use machine learning to detect plant diseases based on real images uploaded by the user.
- To make learning enjoyable through missions, rewards, and progress tracking.

3.3 How the System Works

The system begins with a short tutorial where the player meets a character who guides them inside a space station. After that, the player is asked to select a country and city based on their real-world location. This choice affects the weather conditions in real time. Only then does the player land on Earth and choose a location for growing their plant: indoors, outdoors, or in a greenhouse.

Players can select a plant from a predefined list or upload an image of their own. Each plant has specific needs and responds to environmental conditions based on time and location.

Players can:

- Control lights, sprinklers, and see real-time temperature and humidity.
- Advance time to see how the plant grows over hours or days.
- Earn points by completing missions like planting, watering, or checking the weather.
- Use points in the store to buy tools, seeds, and fertilizers.
- Upload photos of their plants to check for diseases using machine learning models.

3.4 System Implementation – Technological Structure

This system was created as part of a final project and was developed using the Unity game engine and C#. It follows a modular structure and includes a user-friendly interface, login and registration features, environmental control systems, a virtual store, a mission-based reward system, and a machine learning based plant disease detection module.

The implementation includes the following components:

- Weather API – used to retrieve real-time weather data (temperature, humidity, general conditions) based on the selected location.
- Day-Night Cycle – internal time management that affects the environment, including control of lighting and rain.
- Dynamic Store – allows users to purchase seeds, tools, and fertilizers using points earned from completing missions.

- Image Analysis System – based on a machine learning model (ONNX format) for detecting plant diseases. The model runs inside Unity using the Barracuda library.
- Planting Interface – where the user selects a location and a plant type and monitors its growth.
- Mission System – rewards users with points for completing educational tasks and supports progress in the game.
- Separate UI Layers – for displaying information, monitoring the plant's status, and reacting in real time to environmental conditions.

3.5 Target Users

The system is designed for:

- Young children and students who want to learn about nature in a fun way.
- Teachers and educators who are looking for interactive tools to teach science and environment topics.
- Parents who want to expose their kids to digital learning through nature and gardening.
- Tech students who want to explore how game engines and machine learning can work together in a real project.

4. Solution Description

4.1 Project Structure

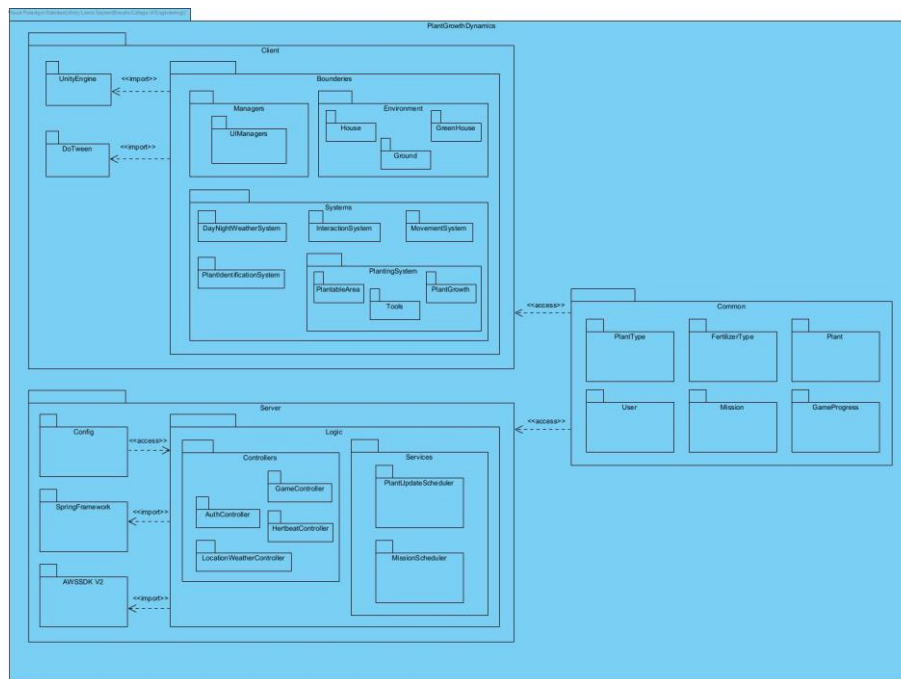


Image 1: Project Structure Package Diagram

The project is built based on a two-layer architecture – a client and a server. The client side is developed in the Unity Engine and its programming language of choice is C#. This side includes various classes, many of whom manage parts of the application, such as data, interactions within the game and the environment. It also features some systems, including the day-night cycle, weather system, plant identification system and the planting and growing system. The server side is responsible on saving data from the client, storing it in the database and continuing growth of plants when players are offline. It also has the capabilities of authenticating users and fetching data from various APIs – weather and geolocation. It is implemented using the Java programming language, and is a Spring-Boot server.

The client side, which is implemented in Unity, has various scenes, which each have their own functionalities and features. They are expanded upon in parts 4.2 to 4.4.

4.2 Menu Scene

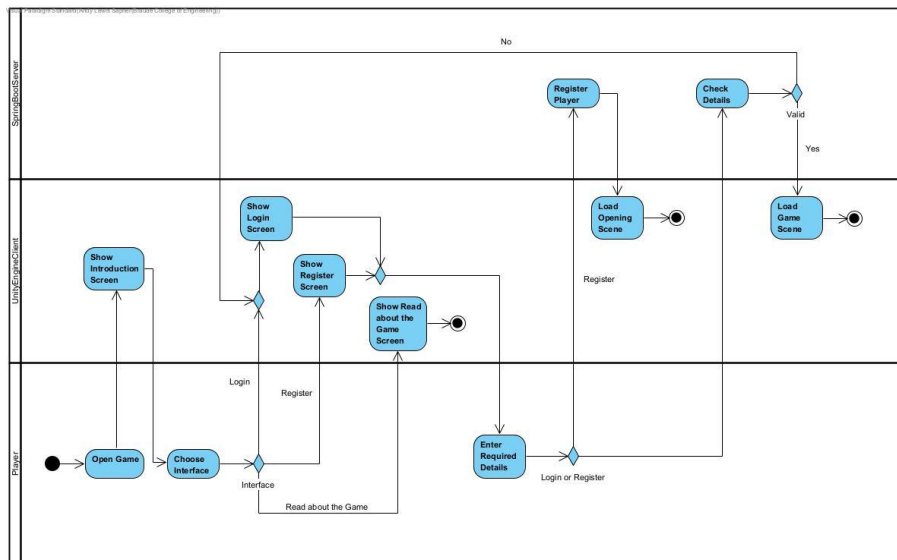


Image 2: Menu Scene Activity Diagram

The first scene in the game is the menu scene. It includes, as seen in the diagram, a login and register screens and a screen which allows reading about the game. Signing up and in are handled by the server.

4.3 Introduction Scene

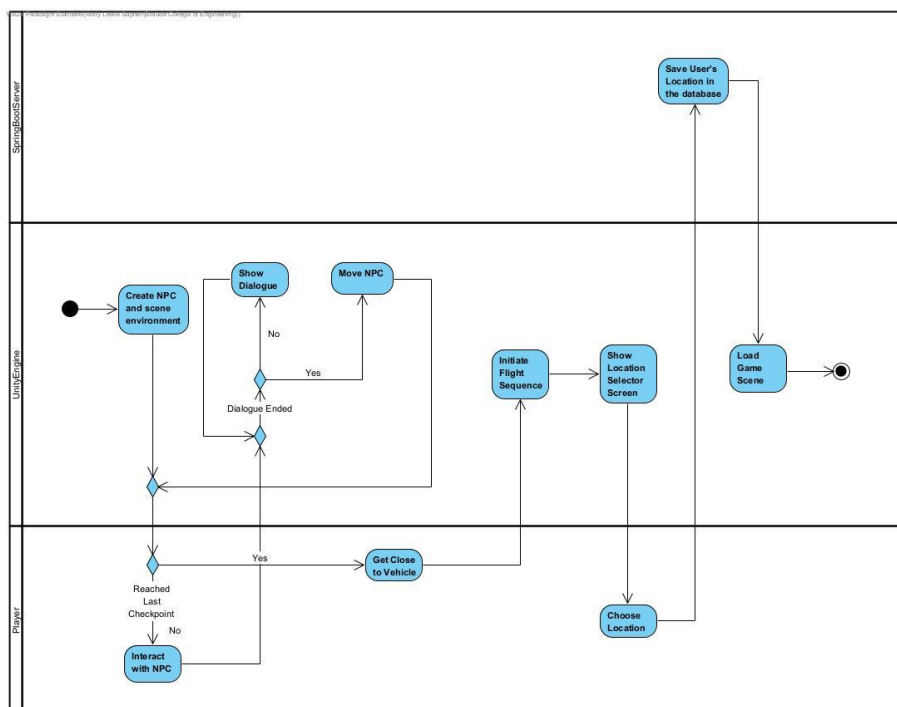


Image 3: Introduction Scene Activity Diagram

The second scene, which is available for players who just registered, is an introduction scene. It features a simple dialogue with a NPC, which is there to introduce the game in a storytelling aspect. It also includes

choosing a location, which is stored in a database and is used to fetch weather from an API for the exact location.

4.4 Game Scene

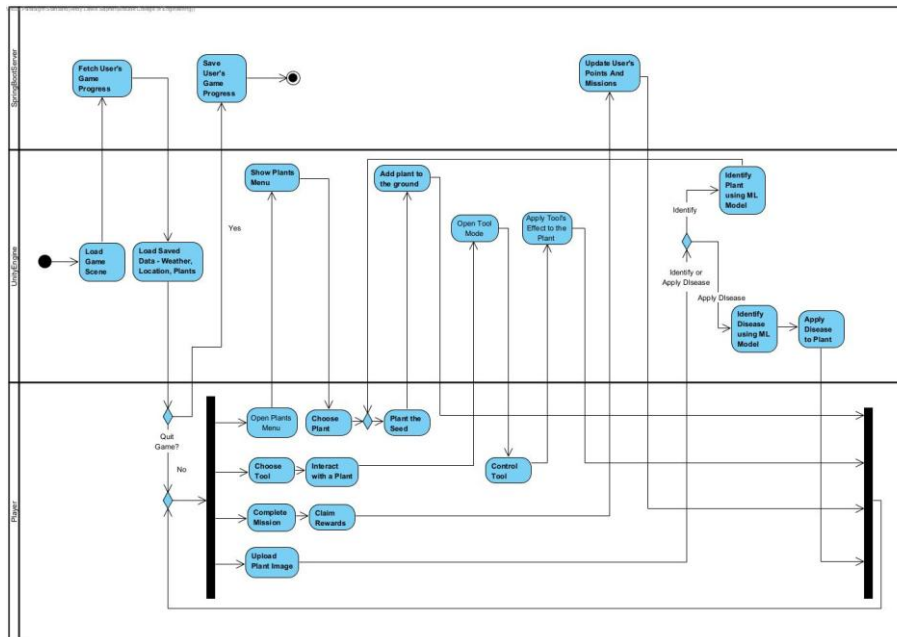


Image 4: Game Scene Activity Diagram

The third scene is the game scene. It includes the main functionality of the game – planting and growing plants. It has various environments where plants can be planted. It also features various tools that can be used by the player to care for plants, and includes a mission system, where players can complete missions to gain points and buy tools, seeds and fertilizers. The scene includes two identification systems, one that identifies the type of the plant and the other the disease of the plant.

5. Development Process

Our first step was to get familiar with the Unity game engine, which was completely new to us. We learned how to build scenes, define 3D objects, work with user interfaces (UI), create animations, and control the behaviour of different elements using C# code. We used tutorials, videos, and official documentation to understand the structure of the engine and to begin developing basic features of the game.

Alongside the technical learning, we conducted deep research about the world of plants. We read professional sources to understand what conditions such as light, humidity, temperature, and fertilization different plants need throughout their growth stages. In addition, we researched common plant diseases and learned how to recognize external symptoms that can be identified in images. We collected a dataset of images of both healthy and diseased plants, and organized them by plant type and disease type, to train a machine learning model to detect plant diseases.

Next, we compared several cloud services to choose the most suitable platform for our system's needs. We considered solutions such as Firebase, AWS, and Spring Boot. Eventually, we decided to develop a dedicated backend using Spring Boot:

- We used Spring Boot, hosted on Amazon EC2, to manage the entire backend logic including user registration, login, and data processing.
- Amazon DynamoDB was used as our NoSQL database for storing user data, plant information, and environmental parameters.

The server was developed to communicate with the client application (Unity), perform actions in real time, process environmental data (such as temperature and humidity from the weather), and send appropriate responses back to the game. We also integrated a Weather API to fetch real-time weather forecasts based on the location chosen by the user at the beginning of the game.

During development, we had to learn how to properly integrate Unity with Spring Boot, including sending secure requests through AWS CloudFront, converting data between JSON format and server models, and managing the correct loading order of game elements to avoid errors during execution.

6. Tools Used in the Project

6.1 Unity Engine

Unity is a popular development platform for creating video games, animations, virtual reality (VR) and augmented reality (AR) content [9]. It is a free-to-try platform commonly used by game developers for its game engine and ability to create 2D and 3D applications, such as games and simulations. It is popular for many reasons, including its robust suite of tools with an easy-to-use interface.

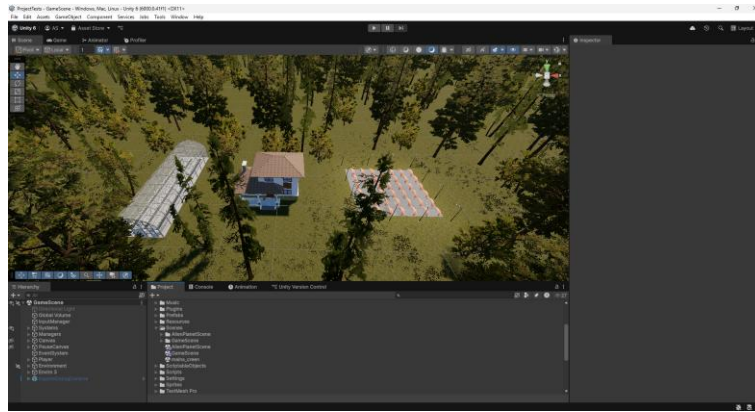


Image 5: The Unity Engine

The platform allows developers to create various games across different genres and platforms [10]. For example, action games, adventure games, role-playing games, simulation games, educational games and more can be created using Unity.

6.2 The C# Programming Language

Unity uses C#, an open-source, object-oriented, and cross-platform programming language [9]. It supports the core concepts, such as abstraction, encapsulation, inheritance and polymorphism. Its syntax is clean, and many complex features found in other languages, in particular pointers, are either minimized or avoided altogether [11]. It also has features like exception handling, garbage collection, and type safety which contribute to its reliability. The language supports dynamic programming, asynchronous programming and LINQ (Language Integrated Query).

6.3 Java programming language and Spring Boot

Java is a programming language and computing platform that powers a large share of today's digital world, by providing the reliable platform upon which many services and applications are built [12]. It is platform independent, which means that code can be written once and can be ran anywhere using the Java Virtual Machine (JVM). It is mostly used for building desktop applications, web applications, Android apps, and enterprise systems [13].

Spring Boot is an open-source Java framework used to create a Micro Service. It provides a faster way to set up, configure, and run both simple and web-based applications. It is a combination of Spring Framework and Embedded Servers. The main goal of Spring Boot is to reduce development, unit test, and integration test time and in Spring Boot, there is no requirement for XML configuration [14].

6.4 Amazon DynamoDB

Amazon DynamoDB is a serverless, NoSQL database that allows developing modern applications at any scale [15]. As a serverless

database, DynamoDB has no cold starts, no version upgrades, no maintenance windows, no patching, and no downtime maintenance. It offers a broad set of security controls and compliance standards. Its reliability is supported by with managed backups, point-in-time recovery and more.

6.5 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) offers the broadest and deepest compute platform, with over 750 instances and choice of the latest processor, storage, networking and operating system [16]. Benefits include reliable and scalable infrastructure on demand and providing secure compute for applications.

7. Challenges

7.1 Working with the Unity Engine and C#

Many challenges appeared when working with the Unity Engine, since this is our first project that incorporates using a game engine to develop it. Thanks to Unity's understandable structure and the vast number of tutorials on the internet, we could start developing the simple features in the game, mainly creating the UI, that is one of the primary elements of any game. Developing more complex features were more challenging than we thought.

The main feature of the game is plant growth. Although we planned to use 6 specific plants as a starting point for the game, not all the plants had 3D models on the internet. Therefore, we decided to change the plants to ones that do have such models. Moreover, it is not so easy to use these models to simulate real life growing, thus the growing was implemented by just changing the scale of the plant according to the time passed from when it was planted.

At first, C# as the main programming language posed a little challenge, since we didn't have enough experience with this language. However, because it has many similarities to the Java language, we got used to it.

Another problem that Unity has is the number of free assets available on the internet. We had to use assets (3D and 2D assets) that not always corresponded to real life representations of the same things. For example, since there is no free asset for fertilizer bags, we decided to use a fertilizer jerrycan instead. In addition, water isn't a common asset that can be found, especially for the case of sprinklers, irrigation and so on, so we used the simple particle system that Unity provides.

The C# language, that supports Unity, has many built-in methods for it. For instance, it includes the Start, Update, OnEnable (and OnDisable),

OnDestroy and many more methods. Learning on how to use these methods so we wouldn't encounter many exceptions wasn't an easy task, but with trial and error, we succeeded to create a couple of systems in the game.

Another challenge that occurs when developing a game is optimizing its performance. Many elements that are added into the Unity's scene may need the game to use more resources from the GPU when it is open. Therefore, some built-in solutions were used, including using Level of Detail (LOD) on objects in the game, turning off objects that are far from the player, not changing the scale of a plant until the player comes closer to it and so on.

This project also includes using machine learning to identify plants and their diseases. The challenge we encountered with this is that there are not enough images of the plants we chose on the internet. Therefore, we had to use all available images we could find to train the machine learning.

7.2 Creating a Server Using Amazon EC2

One of the goals defined for this game is to allow plants to grow in real time. This means that to implement such a feature, we will need to both store the plants somewhere and then apply the same logic as in the client side (the Unity application) to the plants, based on the environment. This posed some challenges, such as choosing the right server provider and a compatible database and then implementing the system.

After a comprehensive search on the internet, we found out that Amazon's services are quite reliable, and they also have some free tiers in some their services. We set up a server using Amazon EC2 and created a database with Amazon DynamoDB. Creating the server wasn't a hard task, because we've used the Spring Boot project creators, that could be downloaded into the local machine and be developed into the server.

The server itself includes many classes. It was a little bit confusing to create some of them, since we were not acquainted with how to use both Spring Boot's functionality and AWS SDK's functionality. With finding solutions on the internet, we were able to create the server, but it was a long task, because many methods that receive or send data to the client included transforming the data from a model class to a map (a JSON object) or from a map to the model class (model class refers to a table in the database).

7.3 Integrating the Server with the Client

The project is divided into two parts – the client which is the Unity application and the server which is the Spring Boot server written with Java. One of the main challenges that occur when trying to integrate both together is to keep the same functionality in both sides. For instance, it is important to use the right order of what to load into the game. Loading in the wrong order may create many exceptions, which could interfere with the game's execution.

Moreover, Unity doesn't support HTTP requests since they are not safe. However, the EC2 instance we've created can get only HTTP requests directly. We had to find a way to ensure that we can send requests to the server. Thankfully, we found the CloudFront service from Amazon that allows redirecting requests from a URL created by it to the server.

8. Results and Conclusions

Our project successfully achieved its main objective: to develop an interactive simulation that teaches users how to grow and care for plants in a fun, educational, and personalized way. Through our system, users can interact with realistic environmental conditions, make decisions, and see how those decisions affect plant growth, both visually and scientifically.

We created a fully functional and engaging Unity-based game that allows players to select a plant, monitor environmental factors such as light, humidity, and temperature, and observe the plant's growth over time. The system also includes a machine learning module that detects plant diseases from uploaded images, helping players better understand visual symptoms of plant health issues.

Additionally, the system integrates real-world data using a Weather API, dynamically adjusting the plant environment based on the user's selected geographic location. This feature adds depth and realism to the experience, bridging the gap between digital learning and real-life gardening knowledge.

From a technical standpoint, we implemented a backend infrastructure based entirely on Spring Boot, hosted on Amazon EC2. This server handles all core functionalities, including user authentication, data management, and communication with the Unity client. We used Amazon DynamoDB as our NoSQL database. This architecture allowed us to maintain performance, scalability, and cloud-based logic processing that supports plant growth simulation in real time.

Throughout development, we faced many challenges that required us to learn new tools and technologies. As a two-person team, we divided our tasks carefully using a shared task table, met weekly, and provided mutual support

throughout the process. This structured collaboration helped us maintain steady progress and quickly resolve problems as they appeared.

Key technologies we mastered include the Unity engine and C# scripting, Spring Boot for backend development, and the AWS SDK for secure communication and server operations. We also learned to integrate machine learning models inside Unity using Barracuda, and to manage game logic alongside real-time environmental data.

In conclusion, the project delivered a complete, multi-layered learning environment that combines game development, scientific education, and cloud technologies. We gained valuable hands-on experience in software development, team collaboration, and problem solving under real constraints.

Looking ahead, future improvements may include adding more plant types, improving visual feedback on plant health, and enabling direct interaction between users, such as plant sharing, challenge modes, or live progress comparisons.

9. Lessons Learned

During the project, we worked as a team of two and kept our work process organized, supportive, and efficient. At the beginning, we created a Google Sheets file where we listed all the tasks, we needed to complete the project. We included due dates for each task and assigned responsibility for each one. This file helped us stay on track and manage our progress.

We meet once a week to update the task status, divide new tasks, and solve any problems that come up. Whenever one of us needed help, the other was always ready to support. We tried to divide the work based on what each of us was good at, but we also switched tasks when needed and helped each other whenever necessary.

Did we work the right way?

Most of the time, we worked in a clear and organized way, communicated well, and supported each other. Each of us took responsibility for different parts of the project and made sure to finish tasks on time. We were both committed to the goal, and our teamwork really helped us succeed.

What would we change in hindsight?

- **The plant disease detection method** – At first, we decided to train our own model to detect plant diseases using images. We spent time collecting and organizing a dataset of plant images. But later, we realized that we didn't have enough images, and our model was not accurate enough. We understood that using an API could be a better solution, and

if we had realized this earlier, we would have saved a lot of time and effort.

- **The server technology** – In the beginning, we planned to use Firebase for user login and basic data. But later, we switched to a server built with Spring Boot, hosted on AWS EC2. This change required us to make a lot of adjustments. Looking back, it would have been better to research our options more deeply at the start and choose the final solution earlier.
- **Writing documentation earlier** – Most of our documents, including this one, were written at the end of the project. It would have been easier if we had written things during the process, not just at the last moment.

Conclusion

In conclusion, we had a good work process that was serious and focused. We learned not only about new technologies, but also how to make decisions under pressure, stay flexible, and work well as a team. We faced real challenges and still managed to build a project that we're proud of.

10. Meeting the Project Goals

At the beginning of the project, we set clear goals for what we wanted to achieve. These included:

- Creating a working game in Unity where users can grow and take care of plants.
- Including real-time weather data that affects the plants.
- Adding a feature to detect plant diseases from images.
- Building a backend server to manage data and user login.
- Making the system simple and fun to use.

We are happy to say that we met most of the goals we set for ourselves.

- The Unity game works well, and users can interact with plants, take care of them, and see how they grow over time.
- We successfully connected a weather API that updates the in-game environment based on real weather data.
- We trained our own model for plant disease detection. It doesn't give the best results because of scarcity of diseased plants images, but it still can identify some diseases successfully.
- We built a backend server using Spring Boot, hosted on AWS EC2, and used DynamoDB to store data. All communication between the game and the server works smoothly.

- We focused on user experience and made sure the system is clear, friendly, and responsive.

While we had to make some changes during the project, we believe that these choices helped us reach the goals in a better and smarter way.

Conclusion

We believe we did a good job meeting the project goals. We stayed flexible, solved problems, and improved our plan when needed. In the end, we created a full working system that does what we planned — and even more.

11. User Guide

11.1 Description

“Plant Growth Dynamics: A Learning Experience” was created to provide a solution for people who want to try to learn how to grow plants. It allows users to plant seeds, follow their growth, deal with diseases that occur to plants by using various tools and more. It also provides the user the ability to control some things in the environment such as lights, fans and irrigation. It includes missions to create a gamified experience, and to guide the player with what they can do in game. In addition, the game asks the player to choose a location so it will use real weather information for growing the plants.

11.2 Application’s Flow

To start with the game, it will be necessary to install it according to the guidelines that appear in the GitHub repository. After installing and running the exe, the following can be done.

11.2.1 Menu Scene

This is the first screen encountered when running the game file. It includes signing-up (2), signing-in (1), reading about the game (3) and exiting the game (4).



Image 6: Menu Scene – First Screen

Register Screen and Login Screen

The sign-up and sign-in screens include inserting a couple of details such as the email address (only for registering – 1), username (a unique one – 2) and a password (3). To register (or login), a button should be pressed accordingly (4).

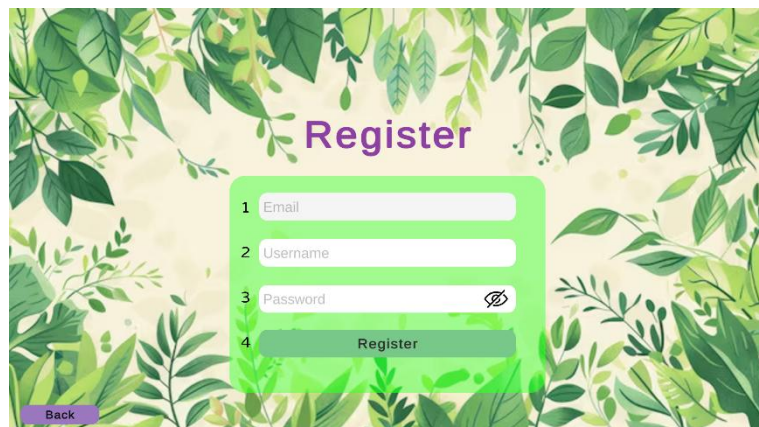


Image 7: Menu Scene – Register Screen

When signing-up, a special scenario will occur, which will not occur anymore when signing-in again. We will first describe this special scenario and then describe the common scenario for both situations.

11.2.2 Introduction Scene

The player first starts in the introduction scene, where they will be instructed to follow a friend of them, and go through a dialogue with it.



Image 8: Introduction Scene – Player's View

The player must interact with the NPC (1) and enter to a dialogue with it (using the E key – 2).



Image 9: Introduction Scene – NPC to Interact with

The dialogue can be advanced using the “next” button (1).

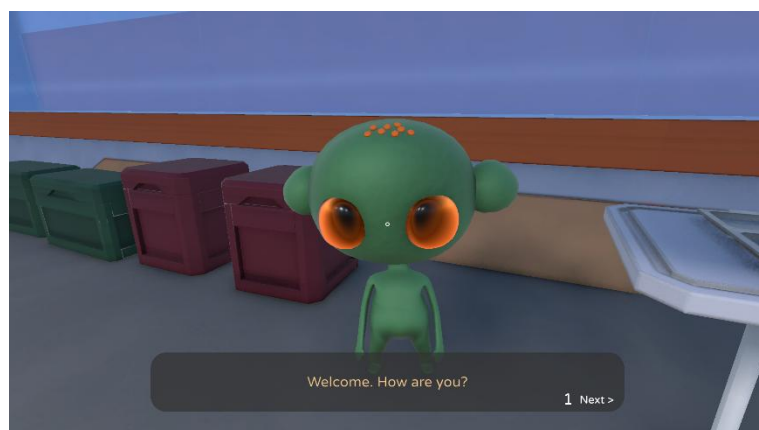


Image 10: Introduction Scene – After Interacting with the NPC

After ending the dialogue, the player must follow the friend and discuss a little with it, until they reach the door to exit the facility.



Image 11: Introduction Scene – Last Mission Checkpoint

The player will interact one last time with the friend and go to the spaceship by following the navigation line (1).

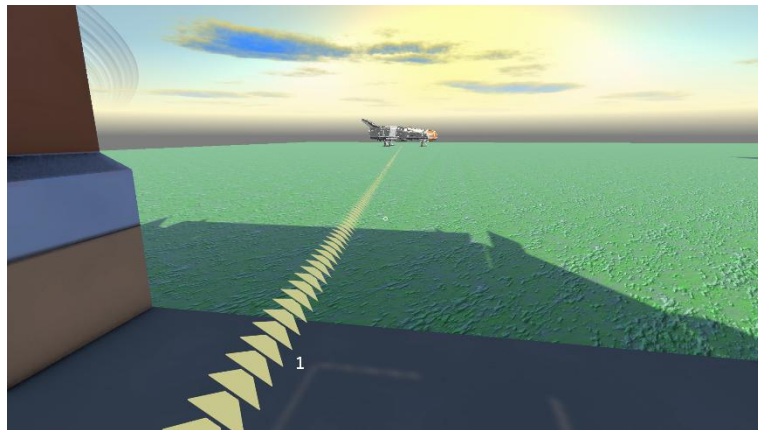


Image 12: Introduction Scene – Navigation to the Spaceship

When reaching the spaceship, it will automatically start flying. When reaching halfway to earth, a location selection UI will appear. In it, the player will need to choose their country (1) and city (2). They can type the values in the dropdowns so the search will be easier. Then, the “Select Location” button (3) should be pressed to select the chosen location.

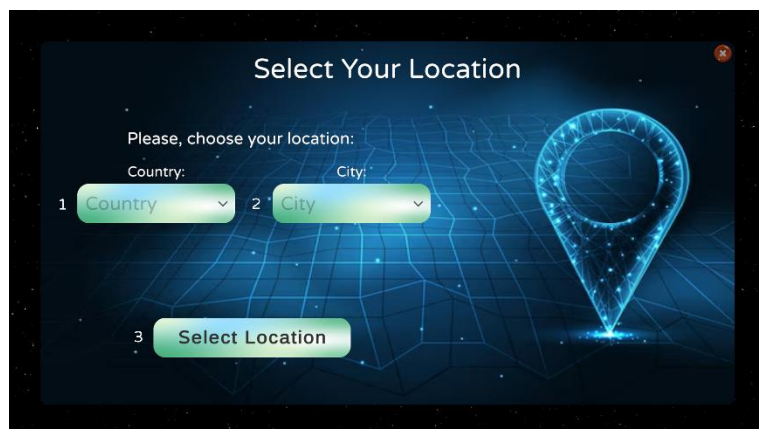


Image 13: Introduction Scene – Location Selection Screen

After selecting the location, the spaceship will continue towards earth and load the next scene. Please note that the selector may not include all cities (or even countries), so the player can choose the closest location to it.

From now on, the description for both new and already registered players is the same. Therefore, already registered players can follow the flow from here.

11.2.3 Game Scene

When entering the game, the player will be greeted with a notification (1) saying that the Alt key can be used to access the UI.



Image 14: Game Scene – Player's View at Start

The game includes various screens that can be accessed and some environments where players can plant their seeds.

Information Screen

By clicking on the  icon, players can access the information screen.

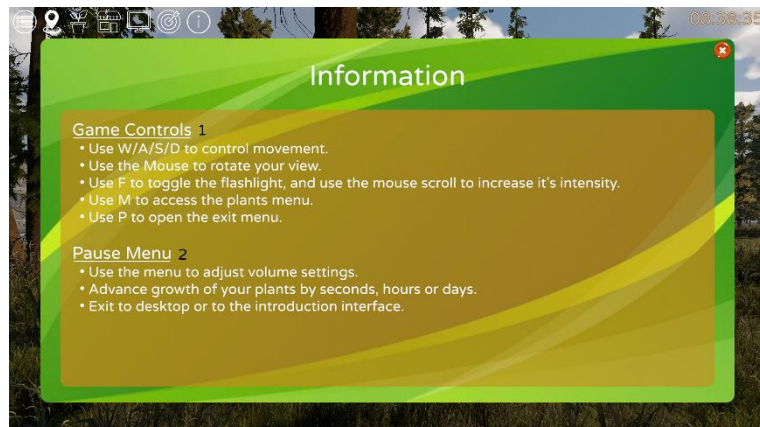



Image 15: Game Scene – Information Screen

This screen includes some information about game controls (1) and the pause menu (2).

Missions Screen

By clicking on the  icon, players can access the missions screen.

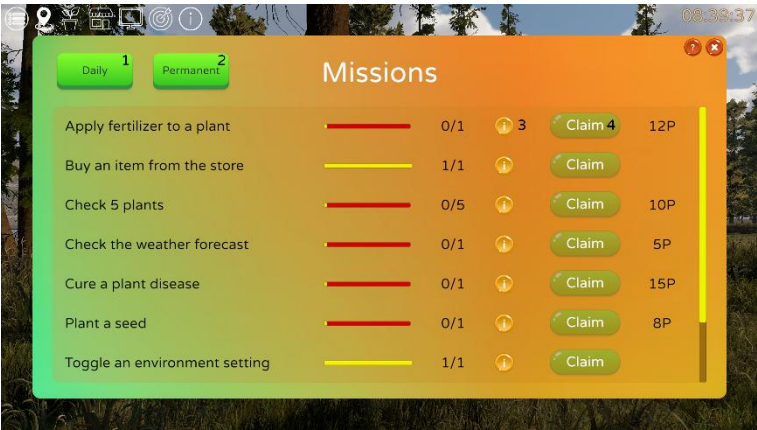


Image 16: Game Scene – Missions Screen

This screen has two tabs, available by choosing “Daily” (1) or “Permanent” (2). When choosing the appropriate tab, players may encounter multiple missions that they can complete. If a player isn’t familiar with how to complete a certain mission, they may click on the (i) icon (3) near the mission, which will provide a simple guide for them. To claim the rewards for completing a mission, the “Claim” button (4) can be pressed. It can be seen that for completed missions, the reward will not be displayed.

Manage Screen


By clicking on the  icon, the player accesses the manage screen.




Image 17: Game Scene – Manage Screen

This screen allows players to control some elements in the environment. It has 3 tabs, corresponding to 3 environments that the player can plant their seeds in. The ground tab (1) allows toggling sprinklers (4) and outdoor lights (5). The house tab (2) allows toggling lights in the house and air conditioning. The green house tab (3) allows toggling lights, irrigation system and fans. All actions affect the growth of the plants, since they change the environmental effects of light, humidity and temperature.

For example, toggling the sprinklers or irrigation system will apply water to plants (although in the greenhouse not all plants can be watered by the irrigation system). The screen also features some statistics about the current situation in each environment – the temperature (6), the humidity (7) and the light radiation (8).

Store Screen

By clicking on the  icon, the player can access the store screen.

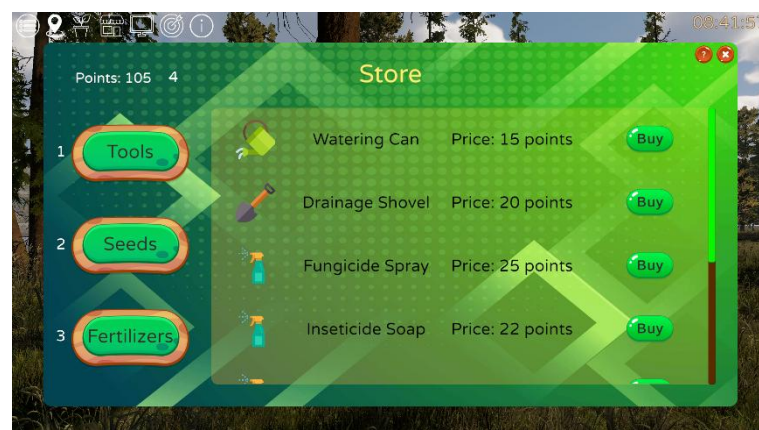


Image 18: Game Scene – Store Screen

The store features three tabs. When a player first registers, they get 100 points to use (the amount is indicated by a text – 4). By completing missions, the player can get more points to buy more things.

The tabs include:

- Tools tab (1) - features tools that they player can use to water the plants, fertilize them or cure them.
- Seeds tab (2) – features seeds that the player can buy from a predefined list of 6 plants.
- Fertilizers (3) tab – after purchasing the fertilizer jerrycan from the tools tab, to use fertilizers, the player can buy one of three types of fertilizers there.

Plants Menu Screen


By clicking on the  icon, the player can access the plants menu screen.



Image 19: Game Scene – Plants Menu Screen

This screen features various elements. First, with this screen, players can choose a seed to plant by pressing on the corresponding plant (1). After pressing on it, the player holds the seed, and they can plant it by interacting (pressing on the E key) with the ground:



Image 20: Game Scene – Player Interacting with the Ground

To purchase seeds, the player will need to earn points (by completing missions) and then enter the store screen.

Second, the players can access the store (Image 19 – 2) from this screen. It is the same store as the one mentioned before. Third, players can upload an image of their plant:

When pressing on the “Upload your Plant” button (Image 19 – 3), the file browser will open:

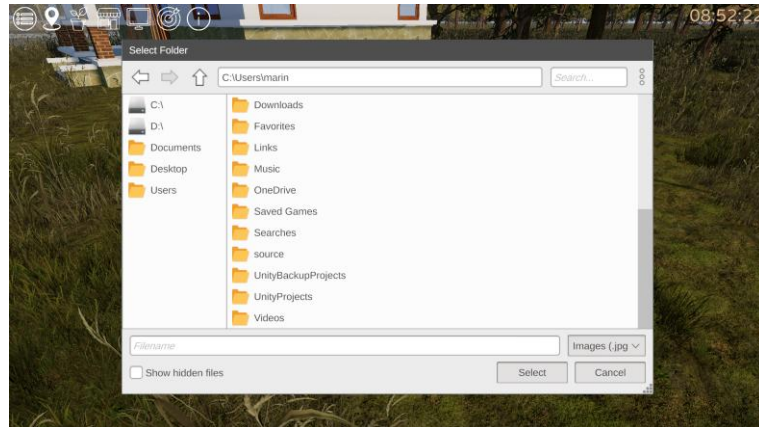


Image 21: Game Scene – File Browser

There, the player will be able to go to the directory in their device where the plant image is available. After choosing the image, a screen will appear:

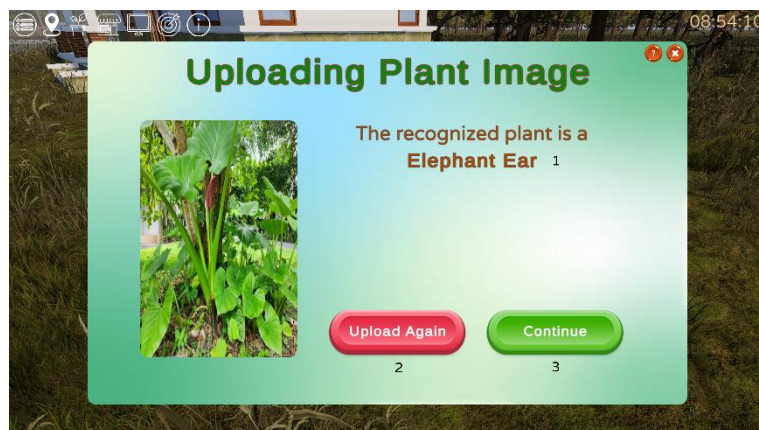



Image 22: Game Scene – Uploading Plant Screen

The plant will be categorized (1) into one of the 6 types of seeds the game features. With pressing on “Continue” (2), the player will hold a new seed of the plant, and then they can plant it. With pressing on “Upload Again” (3), the player can choose another image.

Select Location Screen

By clicking on the  icon, the player can access the location selection screen.

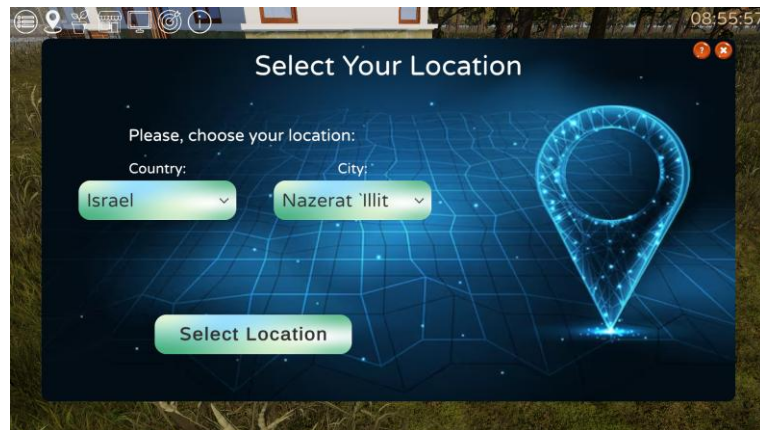



Image 23: Game Scene – Location Selection Screen

Same as the location selection screen in the opening scene, players can change their location. When opening the screen in the game, if a location is already set, it will be shown. To choose a new location, the player will need to erase the country and insert a new one.

Pause Menu

By clicking on the  icon, the players can access the pause menu.

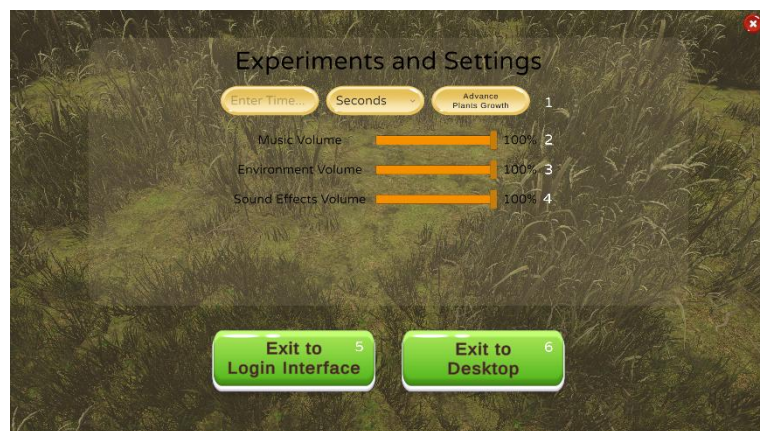


Image 24: Game Scene – Pause Menu Screen

This screen features some simple settings, an experimenting feature and two exit buttons. The screen allows advancing the plant growth (1) by seconds, hours or days. Please note that the weather that will be used for the advancement will be the same weather it is currently.

The settings include adjusting the volume of the background music (2), the environment (3) and sound effects (4). Moreover, there are two exit buttons – one to return to the login interface (5) (to login or register to another account) and exit to desktop (6), which will quit the application entirely.

Planting, Using Tools and Monitoring Plant

The game includes three environments where plants can be planted.

The first environment is the ground:



Image 25: Game Scene – The Ground Planting Environment

To interact with plants in the environment, or plant them, the player must get close to the appropriate ground. When the player is close enough and is pointing towards such an area, this will be shown:



Image 26: Game Scene – Interacting with the Ground

As seen in the image, the player can interact with this area. When pressing on the interact key, the plants menu will be open. After planting, as instructed before, the name of the plant will be shown when pointing to the ground.



Image 27: Game Scene – After planting a seed

From this scenario, there are some options. To monitor the plant, the player can interact again with the plant, and the statistics screen will be open:



Image 28: Game Scene – Statistics Screen

In this screen, the player can look at the moisture (1) of the ground, the nutrient (2) that was given to the plant (fertilizing) and the plant's disease (3). If the plant has a disease, it will show its name and how to cure it. The screen includes also recommendations (4) for the plant, which can be cycled through by pressing the previous (5) and next (6) buttons. Moreover, the player can check the weather's forecast and the current weather in the right side (7).

There are also two buttons – one to remove the plant from the ground permanently (9), and the second is to apply a disease to the plant (8). The second button is useful when the player's plant is diseased, and they want to check what is the cure recommended by the game. Therefore, they can upload an image of the plant, and if the disease is identified, it will apply it to the plant.

When pressing the button, a file browser will open (same as with uploading an image for choosing a seed):

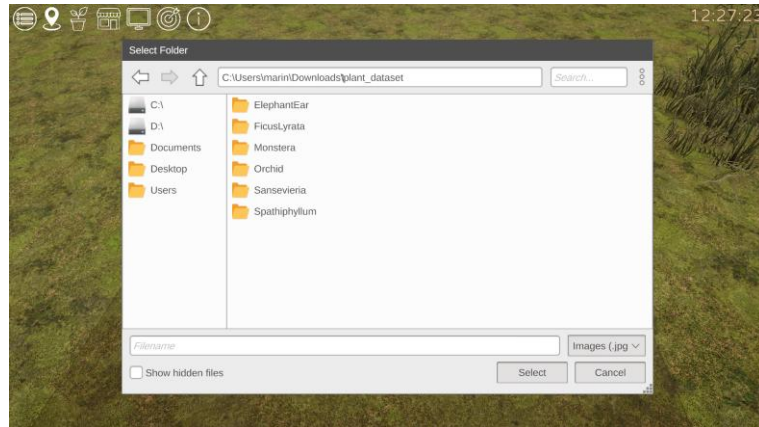


Image 29: Game Scene – File Browser for Disease Update

In the file browser, the player can choose any image of a plant they have and then press on “Select”. After choosing an image, a screen will pop up:



Image 30: Game Scene – Uploading Plant with Disease

In this screen, the player can either choose a disease from one of 3 options (1 – 3) that were detected by a machine learning model. The percentages show what are the probabilities of each disease being the disease the plant has. By pressing on either disease, it will be applied to the plant. (Pressing on “None” will remove the current disease, if the plant has one).

There is also an option to choose “Apply Manually” (4), where all diseases the current plant may have, will be displayed:



Image 31: Game Scene – Apply Disease Manually

Same as in the previous screen, pressing on either disease (1 – 4) will apply it to the plant.

To use tools with the plants, the player must first purchase them from the store screen, in the tools tab. Then, when pointing towards an interactable ground area, such as the one shown before, by using ‘<’ and ‘>’ keys (on the keyboard), the player can cycle through all their available tools.

As an example, we can find the shovel by cycling to it:



Image 32: Game Scene – Changing the Tool the Player Holds

When interacting with the ground, we will enter an activity state with the shovel:

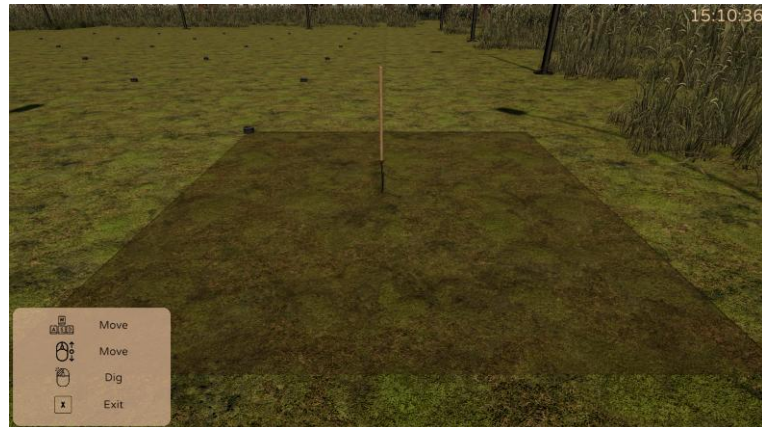


Image 33: Game Scene – Using the Shovel

In the activity state (with all tools), a guide will be shown in the left-bottom corner of the screen with the appropriate controls. The player can try to use the tool by using the controls.

The second environment is a house:



Image 34: Game Scene – The House Planting Environment

The player can enter it through the doors. In the house, the player can find many interactable ground areas where they can plant their plants:



Image 35: Game Scene – Interacting with a Planting Pot in the House

The player can point to objects in the environment to look for such objects. The functionality stays the same as with the ground outside.

The third environment is the green house:



Image 36: Game Scene – The Green House Planting Environment

The player can access the green house also throughout the doors. Same as before, the player can find the interactive ground areas in it.



Image 37: Game Scene – Interacting with a Planting Pot in the Green House

The player's progress – plants that were planted, state of lights, fans, irrigation, etc., tools that were purchased and more is saved only when using the one of the exit buttons in the pause menu. If the player exists the game in any other way, the progress will be lost.

12. Maintenance Guide

This project uses a two-layered architecture – client and server. We will explain about what is included in both sides. First, we will explain about the programs needed, then how to create the server. Finally, we will explain about key parts

of the client and the server, with mentions to important parts of them, such as some classes and unity assets.

12.1 Client Side - Programs

The client side includes a Unity project; therefore, Unity should be installed via the official site of Unity [17]. The project uses the 6000.0.41f1 version of the Unity Editor. It is recommended to install this version to continue developing this project.

In addition, a C# based IDE should be installed. The ones that are currently supported by Unity are Microsoft Visual Studio 2022 (installed with the Unity Editor if checked in installation) and JetBrains Rider (installed via JetBrains' official site [18]). It is recommended to install either of these.

12.2 Server Side – Programs

The server side includes a Java Spring Boot project. To continue development of the server, install any text-editor or IDE that works with Java. For instance, JetBrains IntelliJ (Community Edition) [19] or the Eclipse IDE [20] can be installed.

The server side also connects to Amazon DynamoDB and runs in an Amazon EC2 instance. To use these services, the developer should have an account in the AWS Console [21].

To create a database to work with, the developer can enter the DynamoDB service page [22] and create tables there. The names of the tables and their properties can be viewed in the server's `DynamoDbTableConfig.java` class and their corresponding classes in the "models" package.

To run an instance of the server, the developer should create a new Amazon EC2 instance through the service page [23], in the Instances tab. When creating such an instance, the developer will encounter a private key that should be downloaded and then be used to access the instance.

After creating the instance, the developer can use the "**mvn clean package**" command (in a windows machine, in the command line, where the server is developed) to create an executable (Jar) of the server. Then they can transfer the file to the EC2 instance by using WinSCP [24] (if on a windows machine) or the command line. After that, by connecting to the server, using PuTTY [25] or any other possible way, the server can be run there.

The developer can follow the following guide to run the server in the instance:

(1) Set up the server environment:

- (a) Update the system: use the **"sudo apt update && sudo apt upgrade -y"** command.
- (b) Install Java: use the **"sudo apt install openjdk-17-jdk -y"** command. Verify installation: use the **"java -version"** command.
- (c) Create a directory for the application: Use the commands:
"sudo mkdir -p /opt/plantgame"
"sudo chown \$USER:\$USER /opt/plantgame".
- (d) Copy the Jar to the server (can be done by using WinSCP as mentioned above): Use the command:
"scp target/plantgame-0.0.1-SNAPSHOT.jar user@your-server-IP:/opt/plantgame/plantgame.jar".
"IP" is the IP of the instance.
On the server, the JAR should have the following permissions:
"sudo chmod 644 /opt/plantgame/plantgame.jar"

(2) Create a systemd service:

- (a) Create the service file, using the command **"sudo nano /etc/systemd/system/plantgame.service"**.
- (b) Add the following content to the file:
[Unit]
Description=PlantGame Server
After=network.target

```
[Service]
User=your-username
WorkingDirectory=/opt/plantgame
ExecStart=/usr/bin/java -jar /opt/plantgame/plantgame.jar
SuccessExitStatus=143
Restart=always
RestartSec=10
Environment="OPENWEATHERMAP_API_KEY=apiKey"
Environment="WEATHERAPI_API_KEY= apiKey"
```

```
[Install]
WantedBy=multi-user.target
```

- (c) Reload systemd to recognize this service, by using the **"sudo systemctl daemon-reload"** command.
- (d) Enable the service to start on boot, by using **"sudo systemctl enable plantgame.service"**.

(3) Start and manage the server:

- (a) Start the service: **"sudo systemctl start plantgame.service"**.
- (b) Stop the service (shouldn't be used regularly, only when changing the server, if necessary): **"sudo systemctl stop plantgame.service"**.
- (c) Restarting the service (after updating the Jar or making configuration changes): **"sudo systemctl restart plantgame.service"**.
- (4) Check the service status:
 - (a) Check if running: **"sudo systemctl status plantgame.service"**. If it indicates "active (running)", then everything is alright. If not, there could be exceptions that should be fixed.
 - (b) View service logs: use **"sudo journalctl -u plantgame.service"** command. It is recommended to view only the last 1000 lines, so using **"journalctl -u plantgame.service -n 1000"** is better (since the last exception should appear there, if there is one).

12.3 Client Side – Key Files

The Unity project includes three scenes: `mains_screen`, `GameScene` and `AlienPlanetScene`. These scenes include all the features of the current game.

The `mains_screen` scene has the login interface. The `AlienPlanetScene` has the opening scene of the game, that initiates when registering and `GameScene` has the main functionality of the game, including plant growth.

There are some directories that are important in the client side:

- Managers – includes many managers (regular and UI) in the game. They are singletons, and each of them manage a certain aspect of the game. Key managers are:
 - `DataManager.cs` – includes the loading and saving logic of the game. Most of these methods use the `UnityEngine.Networking` package to send requests and receive responses from the server.
 - `InputManager.cs` – has all the input actions used in the game.
 - `LocationManager.cs` – initializes the location of the user.
 - `WeatherManager.cs` – fetches the weather of the user's location.
 - `UpdateManager.cs` – the project uses the observer pattern, where instead of using the built-in `Update` method in many

classes, they implement the `IUpdateObserver` interface by implementing the `ObservedUpdate` method, which is called in a single `Update` method here. It should improve performance of the application.

- **Serializable** – includes the common models between the client and the server. The models (classes) should have the same fields and types to work.
- **Systems** – includes some systems the game has. The main system is the planting system and is positioned in the `PlantingSystem` directory.

This directory has 3 directories:

- **PlantableArea** – controls areas where plants can be planted.
- **PlantGrowth** – includes parts of the plant object itself. It also has the **`PlantGrowthManager.cs`** class that similarly to the **`UpdateManager.cs`** class, it updates all plants together. This class has the **`CalculateGrowthModifier`** method, which calculates how much should the plant grow according to the weather in its environment and its optimal values, which are defined in a **`PlantSo.cs`** scriptable object. Moreover, it has a **`PlantObjectPool.cs`** class that creates a couple instances of the plants in the start of the game, to improve performance.
- **Tools** – includes the tools that can be used with the plant. Each tool has its separate class, which control its movement, appearance and so on.

12.4 Server Side – Key Files

There are some important packages in the server Java project. Here is some description about them:

- **config:**
 - **`DynamoDBConfig.java`** – includes configuration for the DynamoDB database. It should be noted that there are some private fields there that should be configured in the **`resources/application.properties`** file.
 - **`DynamoDbTableConfig.java`** – includes configurations for all tables in the database. It has all the tables with references to the classes that define them (from the models package).
- **controllers** – this directory defines the API endpoints, when **`GameController.java`** includes the loading and saving endpoints, **`AuthController.java`** the authentication ones and

LocationWeatherController.java all endpoints regarding the weather and location of the user.

- models – contains the same models (and some more unique to the server) as the ones in the client and the database.
- services – includes all the backend functionality for growing the plants while the user is inactive. **PlantUpdateScheduler.java** has the main function that is called every five minutes and updates all available plants in the database.

12.5 Other Notes

We will note that Amazon AWSSDK V2 is used as part of the server. In addition, to use the server from the client side, the developer should create an IAM role in the IAM service [26] that gives a full access to the DynamoDB database. Also, a CloudFront [27] distribution should be created to use an https link for the server, since it is not recommended to use http requests in Unity.

The client side includes some assets from the Unity Asset Store, including SuperBehaviour [28] which improves performance by caching the transform component of game objects, DOTween [29] which provides animations through code, Runtime File Browser [30] that provides access to files in the device when running the game, Enviro 3 [31] which provides excellent weather and day night systems based on user's location and Serialized Dictionary [32] that serializes dictionary to the Unity Editor to make development easier. It also uses the Unity Mesh Simplifier [33] to simplify meshes of the plants to reduce GPU workload (can be used as part of the editor, not in runtime).

13. References

1. PlantSnap Inc. "PlantSnap: Identify Plants, Flowers, Trees & More" – An app designed for identifying plants and providing basic care tips.
<https://www.plantsnap.com/>
2. Wäldchen, J., & Mäder, P. (2018). Plant species identification using computer vision techniques: A comprehensive review.
<https://doi.org/10.1016/j.ecoinf.2018.07.006>
3. Gardenia Technologies. "Gardenia: Gardening, Plants Care" – A gardening management tool to track and organize plant care schedules.
<https://www.mygarden.com/>
4. Taylor, J., & Francis, R. (2020). Integrating weather data into gardening simulations: A framework for virtual environments.
<https://doi.org/10.1016/j.envsoft.2020.104678>

5. Simone Strey et al. "Plantix: Diagnose Plant Problems" – A plant health diagnostic tool aimed at helping farmers identify and treat plant diseases.
<https://plantix.net/en/>
6. Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016). Using deep learning for image-based plant disease detection. Frontiers in Plant Science.
<https://doi.org/10.3389/fpls.2016.01419>
7. A. Cliffe. (n.d.). Teaching through Games: Educational Video Games in Interactive Learning. Springer Journal. Retrieved from
<https://link.springer.com/article/10.1007/BF02504866>
8. Gee, J. P. (2005). Why Video Games Are Good for Learning. Computers in Entertainment (CIE).
<https://dl.acm.org/doi/10.1145/1077246.1077253>
9. What is Unity?, Coursera
<https://www.coursera.org/articles/what-is-unity>
10. Technical Overview of Unity Game Engine, PubNub
<https://www.pubnub.com/guides/unity/>
11. Introduction to C#, GeeksforGeeks
<https://www.geeksforgeeks.org/introduction-to-c-sharp/>
12. What is Java technology and why do I need it?, Java
https://www.java.com/en/download/help/whatis_java.html
13. Introduction to Java, GeeksforGeeks
<https://www.geeksforgeeks.org/introduction-to-java/>
14. Spring Boot Tutorial, GeeksforGeeks
<https://www.geeksforgeeks.org/spring-boot/>
15. Amazon DynamoDB, Amazon
<https://aws.amazon.com/dynamodb/>
16. Amazon EC2, Amazon
<https://aws.amazon.com/ec2/>
17. Download Unity
<https://unity.com/download>
18. Download Rider
<https://www.jetbrains.com/rider/download/>
19. Download IntelliJ IDEA Community Edition
<https://www.jetbrains.com/idea/download/>
20. Download Eclipse IDE
<https://www.eclipse.org/downloads/>
21. AWS Console
<https://console.aws.amazon.com/>

- 22. AWS DynamoDB Service
<https://console.aws.amazon.com/dynamodbv2>
- 23. Amazon EC2 Service
<https://console.aws.amazon.com/ec2>
- 24. Download WinSCP
<https://winscp.net/eng/download.php>
- 25. Download PuTTY
<https://www.putty.org/>
- 26. AWS IAM Service
<http://console.aws.amazon.com/iam>
- 27. AWS CloudFront Service
<http://console.aws.amazon.com/cloudfront>
- 28. Super Behaviour – Unity Asset Store
<https://assetstore.unity.com/packages/tools/utilities/super-behaviour-258290>
- 29. DOTween (HOTween v2) – Unity Asset Store
<https://assetstore.unity.com/packages/tools/animation/dotween-hotween-v2-27676>
- 30. Runtime File Browser – Unity Asset Store
<https://assetstore.unity.com/packages/tools/gui/runtime-file-browser-113006>
- 31. Enviro 3 – Sky and Weather – Unity Asset Store
<https://assetstore.unity.com/packages/tools/particles-effects/enviro-3-sky-and-weather-236601>
- 32. Serialized Dictionary – Unity Asset Store
<https://assetstore.unity.com/packages/tools/utilities/serialized-dictionary-243052>
- 33. Unity Mesh Simplifier – GitHub
<https://github.com/Whinarn/UnityMeshSimplifier>