BLOOM

Installation Guide, User Manual and Maintenance Guide

Udit Bhatia

Pang Hoi (Eddie) Chan

Marat Danyarov

Siqi (Claire) He

Alec Mason

Daniel-Favour Oshidero

Yan Chun (Ivan) Yeung

15 December 2023

Contents

1 About this document

1.1 About Bloom

Bloom is an edutainment mobile application developed by MSc Computer Science students at the University of Bath for the CM50109 Software Engineering unit. It focuses on teaching users about indoor plant care through an engaging blend of education and entertainment. Featuring an interactive hub, educational quizzes, and real-time plant maintenance tasks, the app allows users to learn about different indoor plants, manage their care, and earn rewards through interactive gameplay.

For detailed information about Bloom's history, purpose and game mechanics, see Overview of Bloom.

1.2 How to use this document

1.2.1 As a user

If you want to learn more about houseplants, or if you want help caring for your own houseplants, then Bloom is the app for you!

Begin with the installation guide for users.

Once you've got the app, have a look at the Tutorial. You'll get to know the Bloom interface, and you'll learn how to take proper care of a cactus! After that, you can refer to the Detailed user manual for information about all of Bloom's different features.

1.2.2 As a developer or modder

The installation guide for developers contains instructions to download the app's source code and build it on your computer.

For anything else you want to do, refer to Useful chapters for different readers.

1.3 Sharing

This document may be freely accessed by and shared between any interested parties, with no restrictions whatsoever.

2 Version history

This is a complete list of every revision made to this document. Previous versions can be retrieved from the Overleaf repository:
https://www.overleaf.com/read/kjkzvhzrsgzh#8ea4ce

Version	Date	Parts	Changes
1.10	15/12/23	III	Added structure diagrams
1.9	15/12/23	III	Added maintenance guide meta-information, dev tools, dependencies, build instructions, troubleshooting and expansion
1.8	14/12/23	III	Completed source code documentation; added extension points and feature recommendations
1.7	13/12/23	II	Updated detailed user manual
1.6	13/12/23	II	Added getting started tutorial
1.5	13/12/23	III	Added source code documentation templates for components, screens, states and navigation
1.4	07/12/23	I	Added user installation guide
1.3	04/12/23	I	Added list of tested devices
1.2	04/12/23	III	Added maintenance guide template
1.1	29/11/23	I	Added developer installation guide
1.0	29/11/23	All	Created document

3 Acknowledgements

The authors would like to thank the following individuals, whose specific advice and support helped us to create this document:

- Dr Julian Padget, Reader in Artificial Intelligence at University of Bath
- Fahid Mohammed, Postgraduate Research Student at University of Bath
- Elena Safrygina, Postgraduate Research Student at University of Bath
- Jinha Yoon, Postgraduate Research Student at University of Bath

The Bloom logo featured on the title page was created using DALL·E 3 by OpenAI. The authors claim ownership of this logo as per OpenAI's terms of use.

A complete list of credits and acknowledgements for Bloom is available at https://github.com/dfoshidero/Bloom#acknowledgments

Part I

Installation Guide

1 Android

We recommend installing Bloom on an Android device. You can also access Bloom on Windows or Mac, but this will require you to create an Android virtual device.

A demonstration build of the application can be found in the GitHub repository, specifically within the "test" file. This build is available for download from the repository and can be subsequently installed on your mobile device.

For the convenience of users, we provide the following QR code, to facilitate the direct download of the APK onto an Android device:

Bloom APK

2 iOS

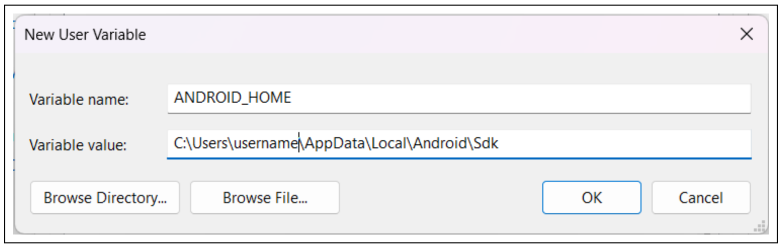
Bloom is not currently available for iOS devices. If you do not have an Android device, we recommend installing the app on your Windows or Mac computer instead.

3 Windows or Mac

If you don't have an Android device available to test with, we recommend using the default emulator that comes with Android Studio. If you run into any problems setting it up, follow the steps in this guide.

Windows:

1. Download Android Studio.
2. Open **Android Studio Setup**. Under **Select components to install**, select Android Studio and Android Virtual Device. Then, click **Next**.
3. In the Android Studio Setup Wizard, under **Install Type**, select **Standard** and click **Next**.
4. The Android Studio Setup Wizard will ask you to verify the settings, such as the version of Android SDK, platform-tools, and so on. Click **Next** after you have verified.
5. In the next window, accept licenses for all available components.
6. After the tools installation is complete, configure the `ANDROID_HOME` environment variable. Go to **Windows Control Panel** > **User Accounts** > **User Accounts** (again) > **Change my environment variables** and click **New** to create a new `ANDROID_HOME` user variable. The value of this variable will point to the path to your Android SDK. For example:


New User Variable

Variable name: ANDROID_HOME

Variable value: C:\Users\username\AppData\Local\Android\Sdk

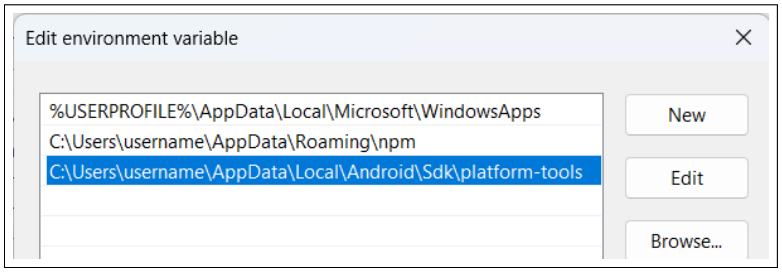
By default, the Android SDK is installed at the following location:

```
%LOCALAPPDATA%\Android\Sdk
```

7. To verify that the new environment variable is loaded, open **PowerShell**, and copy and paste the following command:

```
Get-ChildItem -Path Env:
```

8. To add platform-tools to the Path, go to **Windows Control Panel** > **User Accounts** > **User Accounts** (again) > **Change my environment variables** > **Path** > **Edit** > **New** and add the path to the platform-tools to the list as shown below:



```
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
C:\Users\username\AppData\Roaming\npm
C:\Users\username\AppData\Local\Android\Sdk\platform-tools
```

By default, the platform-tools are installed at the following location:

```
%LOCALAPPDATA%\Android\Sdk\platform-tools
```

9. Finally, make sure that you can run adb from the PowerShell. For example, run the adb –version to see which version of the adb your system is running.

macOS:

1. Download and install Android Studio.
2. Open the **Android Studio** app, click **More Actions** and select **SDK Manager**.
3. Open Android Studio, go to **Settings** > **Languages & Frameworks** > **Android SDK**. From the **SDK Platforms** tab, select the latest Android version (API level).

   Then, click on the **SDK Tools** tab and make sure you have at least one version of the **Android SDK Build-Tools** and **Android Emulator** installed.
4. Copy or remember the path listed in the box that says **Android SDK Location**.
5. Click **Apply** and **OK** to install the Android SDK and related build tools.
6. If you are on macOS or Linux, add an environment variable pointing to the Android SDK location in:

```
/.bash_profile (or .../.zshrc if you use Zsh).
```

For example:

```
export ANDROID_HOME=/your/path/here.
```

Add the following lines to your `.../.zprofile` or `.../.zshrc` (if you are using bash, then `.../.bash_profile` or `/.bashrc`) config file:

```
export ANDROID_HOME=$HOME/Library/Android/sdk
export PATH=$PATH:$ANDROID_HOME/emulator
export PATH=$PATH:$ANDROID_HOME/platform-tools
```

7. Reload the path environment variables in your current shell:

```
# for bash
source $HOME/.bashrc

# for zsh
source $HOME/.zshrc
```

8. Finally, make sure that you can run adb from your terminal.

You must then set up a virtual device as follows:

1. On the Android Studio main screen, click **More Actions**, then **Virtual Device Manager** in the dropdown.
2. Click the **Create device** button.
3. Under **Select Hardware**, choose the type of hardware you'd like to emulate. We recommend the **Google Pixel 7** or the **Google Pixel 7 Pro**, as the Bloom application has been thoroughly tested using those devices.
4. Select an OS version to load on the emulator (probably one of the system images in the **Recommended** tab), and download the image.
5. Change any other settings you'd like, and press **Finish** to create the virtual device. You can now run this device anytime by pressing the Play button in the AVD Manager window.

Upon completion of these steps, you can simply drag and drop the Bloom APK file onto the emulator and it will automatically start installing.

Part II

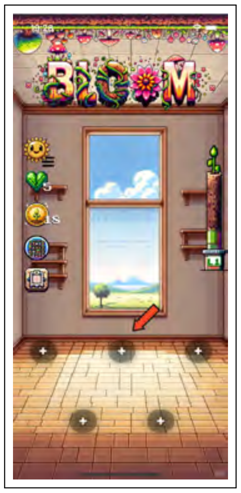
User Manual

1 Getting started

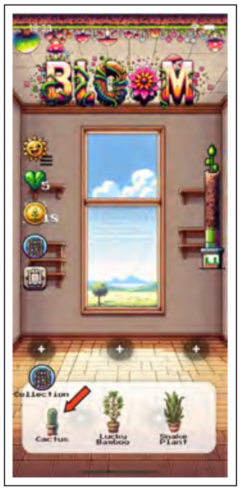
First, follow the instructions in the installation guide for users to install the app on your device.
This chapter provides a brief tutorial for users to familiarise themselves with the game. It gives users the essentials of the main functionalities and actions that users could do throughout the game. For more detailed descriptions of each game features please see Detailed User Manual.

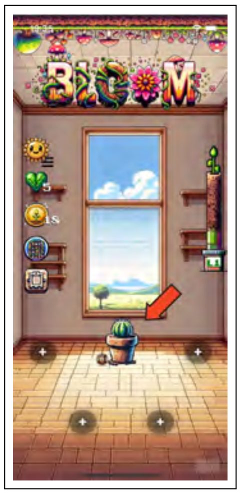
1.1 Tutorial

1.1.1 How to place a plant



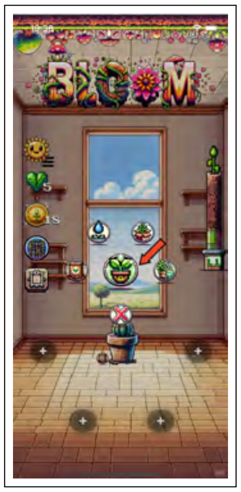
1. On the main screen, select a place for your plant and press the button "+".



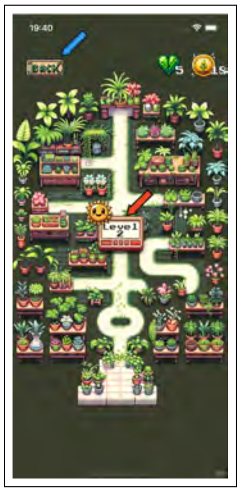


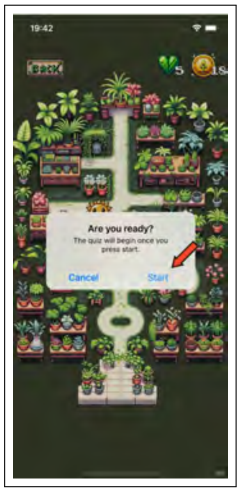
2. Select a plant you want to learn and/ or manage by tapping on the one of the item shown in the menu at the bottom of the screen.

1.1.2 How to start the quiz for a plant



1. Tap a plant icon to open a plant menu. Click the learn button to move to the quiz screen.



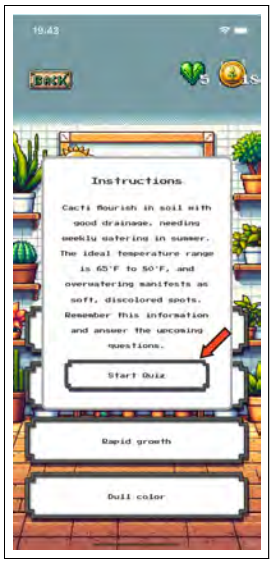


Are you ready?
The quiz will begin once you press start.

2. If you want to return to the main menu, tap the back button or select the quiz button to start the quiz:

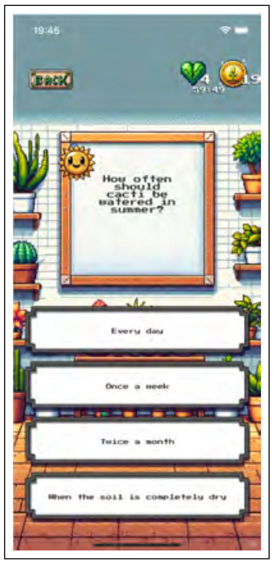
3. Tap the start button:

4. Read the instructions carefully and tab "Start Quiz":
5. Read the questions and choose the correct answer:



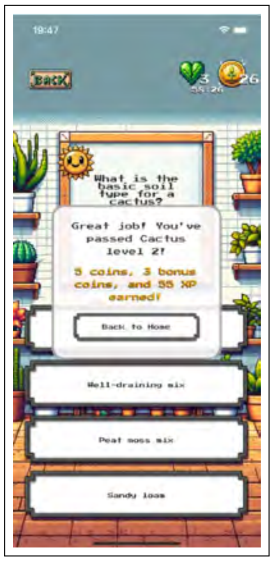
Instructions

Cacti flourish in soil with good drainage, needing weekly watering in summer. The ideal temperature range is 65'F to 50'F, and overwatering manifests as soft, discolored spots. Remember this information and answer the upcoming questions.



How often should cacti be watered in summer?

6. You will receive coins and XP, and your Mastery level will increase upon completion of the quiz.

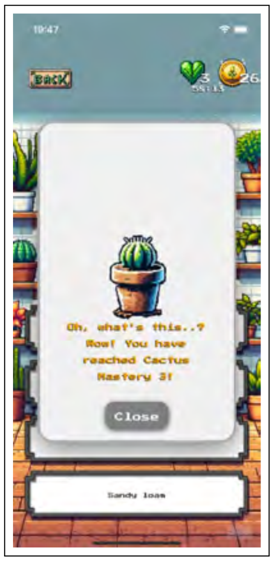


What is the basic soil type for a cactus?

Great job! You've passed Cactus level 2!

5 coins, 3 bonus coins, and 55 XP earned!

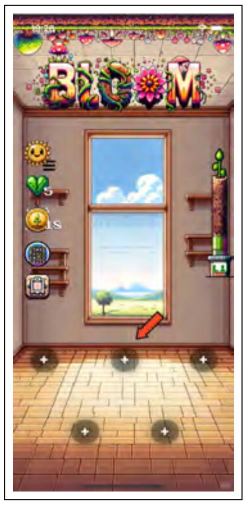
$\longrightarrow$

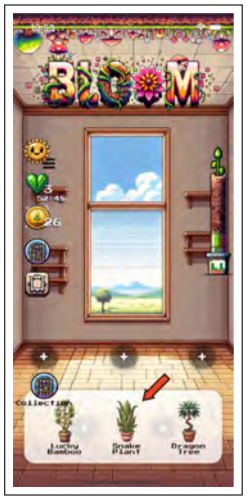


Oh, what's this..? Wow! You have reached Cactus Mastery 3!

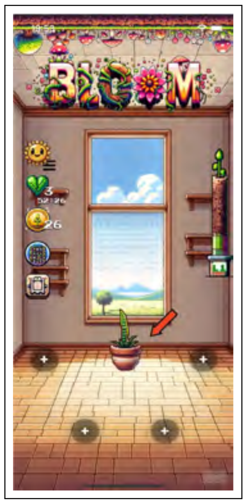
1.1.3 How to link a real-life plant to Bloom

1. On the main screen, select a place for your real plant and press the button “+”:
2. From the menu, select a plant you want to link from real life:

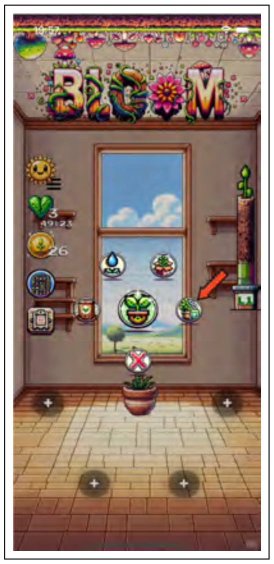


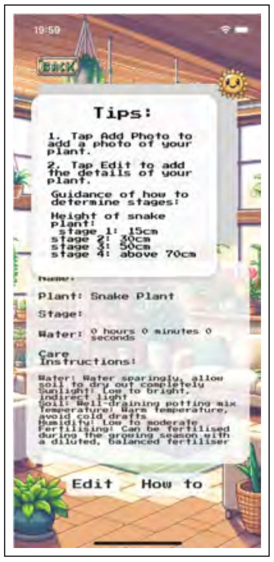


$\longrightarrow$



3. Tap a plant icon to open a plant menu and select the “link the plant option”:
4. Read the tips carefully:





Tips:

1. Tap Add Photo to add a photo of your plant.

2. Tap Edit to add the details of your plant.

Guidance of how to determine stages:

Height of snake plant:
stage 1: 15cm
stage 2: 30cm
stage 3: 50cm
stage 4: above 70cm

Name:

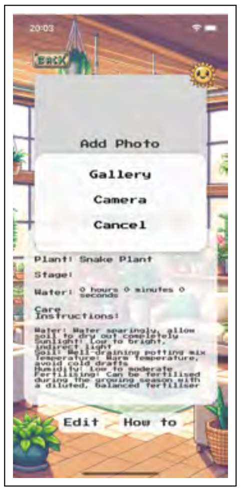
Plant: Snake Plant

Stage:

Water: 0 hours 0 minutes 0 seconds

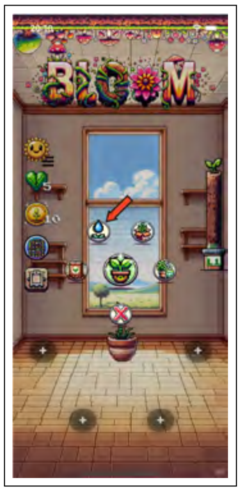
Care Instructions:

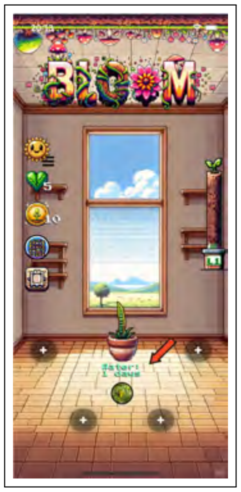
Water: Water sparingly, allow soil to dry out completely
Sunlight: Low to bright, indirect light
Soil: Well-draining potting mix
Temperature: Warm temperature, avoid cold drafts
Humidity: Low to moderate
Fertilising: Can be fertilised during the growing season with a diluted, balanced fertiliser



Plant: Snake Plant
Stage:
Water: 0 hours 0 minutes 0 seconds
Care Instructions:
Water: Water sparingly, allow soil to dry out completely
Sunlight: Low to bright, indirect light
Soil: Well-draining potting mix
Temperature: Warm temperature, avoid cold drafts
Humidity: Low to moderate
Fertilising: Can be fertilised during the growing season with a diluted, balanced fertiliser

5. Tap the "Add Photo" then select option "Gallery" if you want to upload the photo of your plant from your smartphone gallery or select option "Camera" if you want to take a photo of your plant:



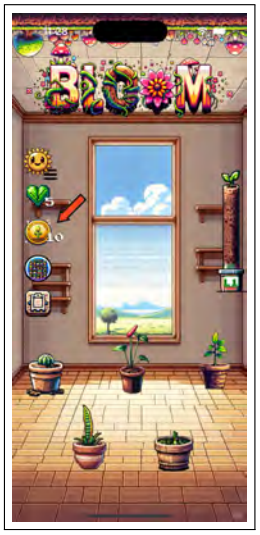


6. To track the watering record tab on the real plant avatar select the watering icon:

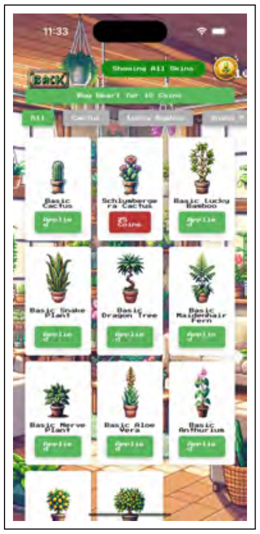
7. Then you can observe the recommended time for the next watering session:

1.1.4 How to buy skins or hearts

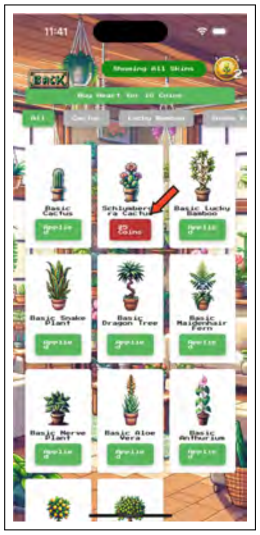
1. On the main screen, select the shop button to access the shop. The number shown on the shop button icon is the amount of coins the player currently has:



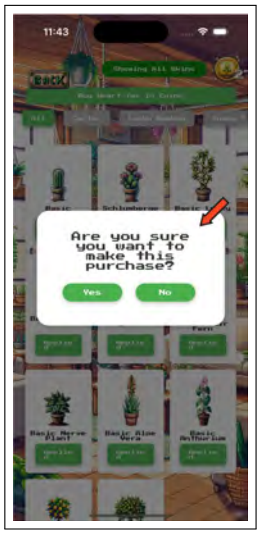
⟶



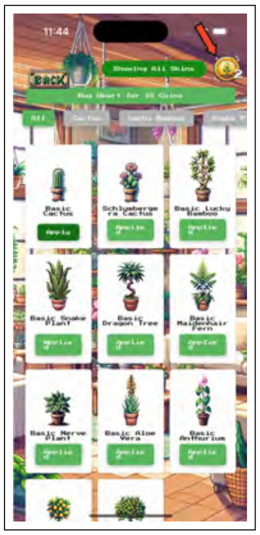
2. To purchase the skin, select the plant skin (stated with the cost of the skin), then press the "yes" button to confirm the purchase. Then you will see that coins are updated after the purchase:



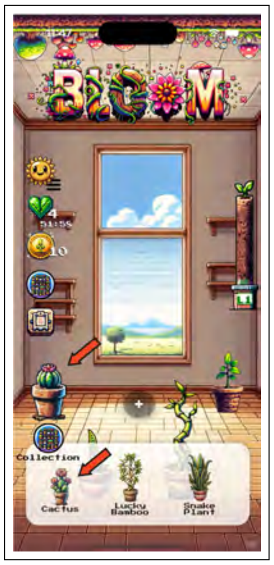
⟶

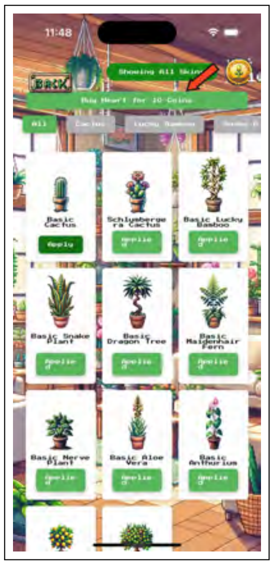


⟶

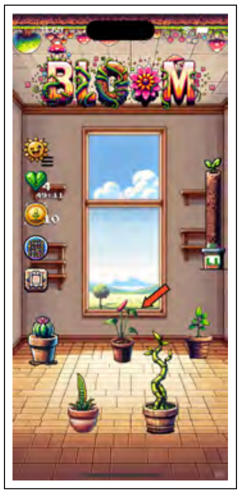
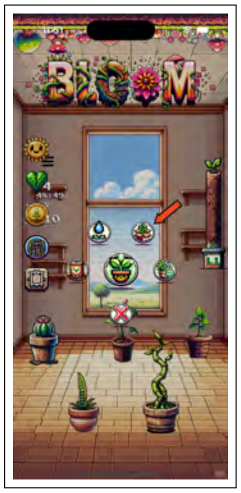


3. Plant skin updated after purchase:
4. To purchase a heart, select the "Buy Heart for 10 Coins" button on the shop screen:



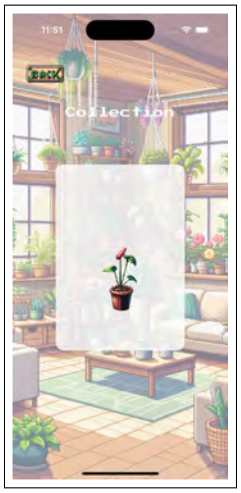


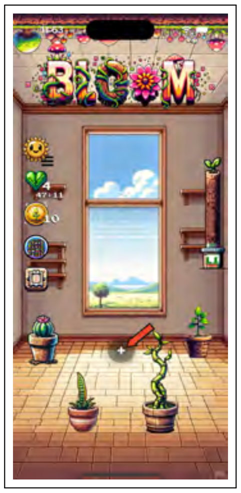
1.1.5 How to add to/place from the collection

1. To place the plant into the collection, select and press on the target plant on the main screen:

2. Press the “archive plant” button:

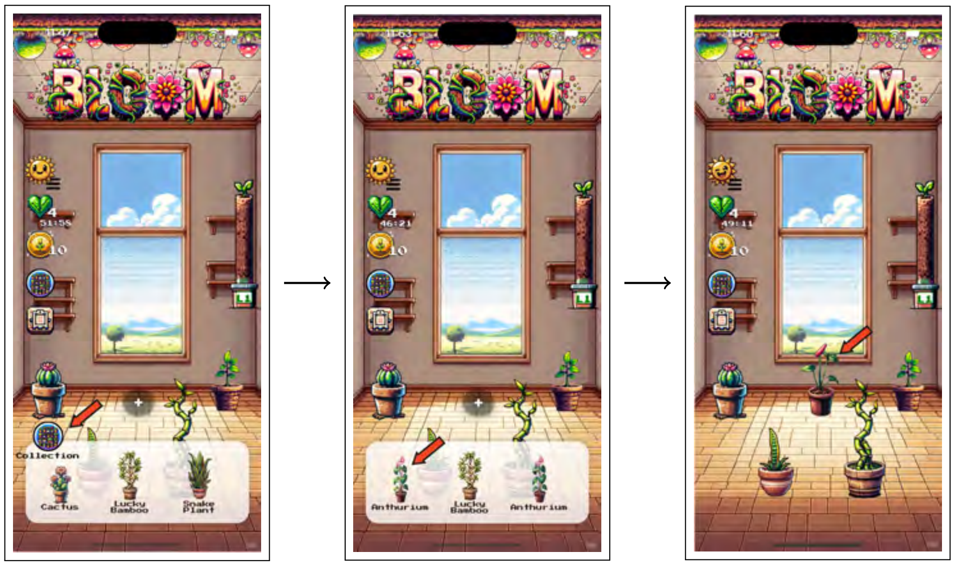




3. The archived plant can be viewed in the collection:

4. To place archived plant from collection to main screen, On the main screen, select a place for your plant and press the button “+” on the main screen:

5. Press the “collection” button on the top left of the plant inventory, then select the archived plant:



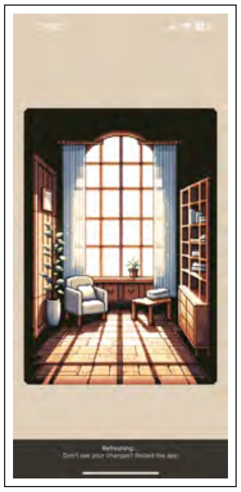
2 Detailed user manual

This chapter provides a detailed catalog to the various features of the game, designed to allow both a guided learning process about indoor plants, as well as the management and care of real-life plants. The games features include, but are not limited to:

- **Sun Oracle:** A character that serves as a guide, offering advice and information to the user throughout the game.
- **Educational Quizzes:** These are interactive quizzes aimed at teaching users about various types of indoor plants, enhancing their knowledge in a fun and engaging manner.
- **Plant Maintenance:** This feature simulates the virtual care of plants, reflecting the responsibilities associated with real-life plant maintenance. It provides users with a practical understanding of plant care.
- **Interactive Hub Interface:** A virtual bedroom acts as the central hub for user interactions within the game. This space allows users to access different elements of the game in a cohesive environment.
- **Difficulty Levels:** Plants are categorized into Easy, Medium, and Hard levels. This classification aids in progressive learning, enabling users to gradually increase their knowledge and skills in plant care.
- **Real-time Management:** The game includes real-time features such as watering timers, which remind players of essential plant care tasks in their real life, reinforcing the learning experience.
- **Rewards and Achievements:** Users earn experience points, unlock mastery levels, and receive achievements for reaching various milestones. These rewards serve to motivate and engage users, making the learning process more rewarding.

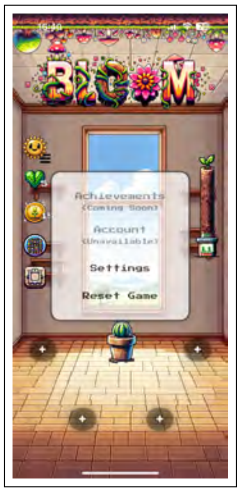
2.1 General

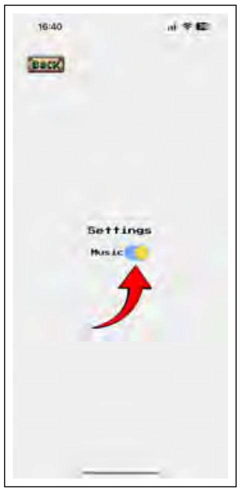
2.1.1 Loading screen



The Loading Screen enhances user experience, appearing at game startup to ensure seamless transitions by preloading assets, minimizing disruptions caused by loading times. This approach maintains optimal performance for smooth gameplay.

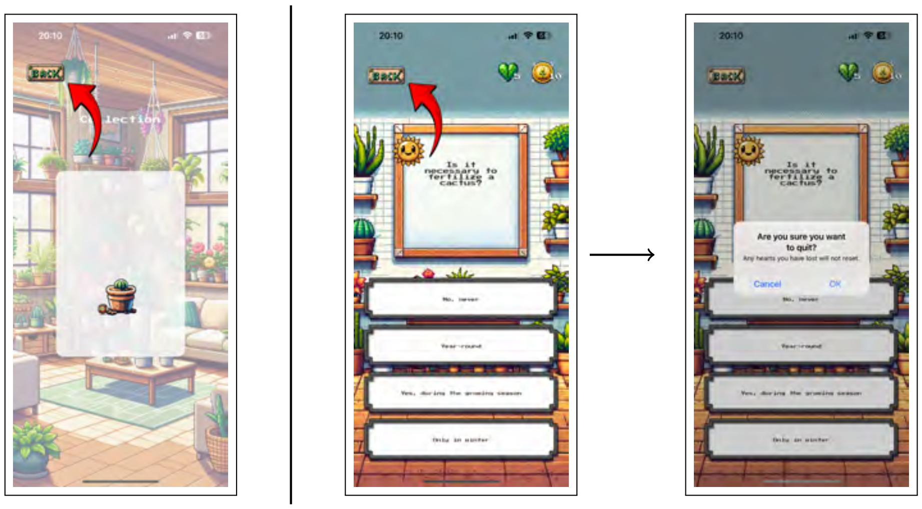
2.1.2 Main menu





In the main menu, options to adjust settings or reset data are available. Current settings only allow music to be switched on or off. Functions for accounts and achievements are not yet implemented.

2.1.3 Back button



Is it necessary to fertilize a cactus?

No, never

Year-round

Yes, during the growing season

Only in winter

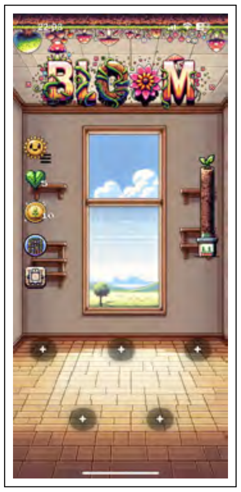
Are you sure you want to quit?

Any hearts you have lost will not reset.

Cancel OK

The game interface features a page-by-page layout with a 'Back' button on all but the home page, ensuring easy navigation through sections like Shop, Collections, Stats, Settings, Levels, and Quiz.

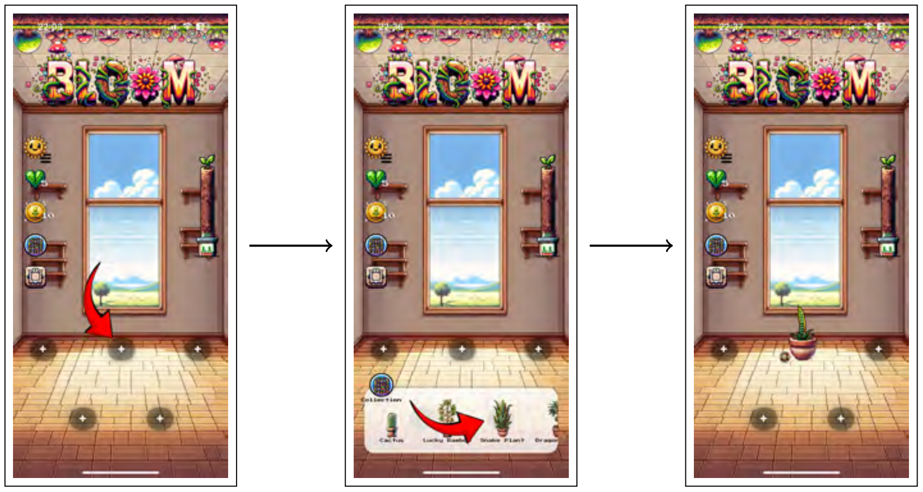
2.2 Main screen



After the initial loading, the game's hub becomes accessible. Designed as an interactive virtual "room," it evolves as users progress through levels, with unlockable new, customisable rooms. The main screen includes:

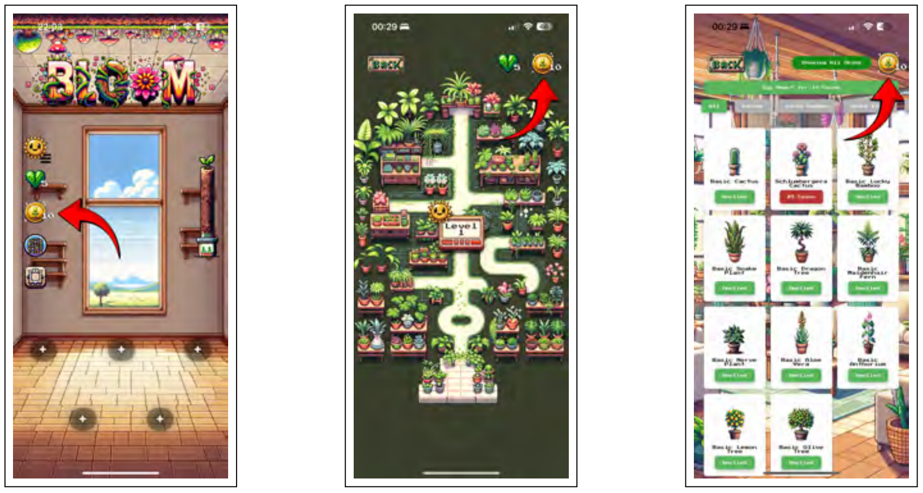
- **Oracle Icon:** Interactive button leading to the main menu, also shows reactions.
- **Hearts Count and Timer:** Displays heart count and time remaining for heart regain.
- **Coin Icon:** Displays coin count and navigates to the shop.
- **Collections Icon:** Accesses archived plants.
- **Game Stats Icon:** Displays plant statistics.
- **Player XP Bar:** Indicates the user's current experience points.
- **Fixed Plant Locations:** Varies by room, showing placed and linked plants, as well as positions to place plants.

2.2.1 Add plant from inventory



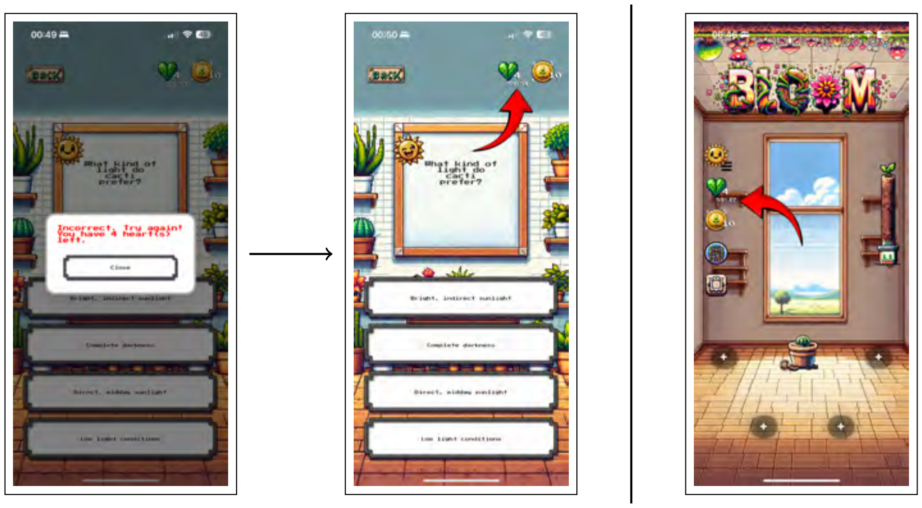
To place a plant from the mastery list, a user should simply press the "+" icon and choose their desired plant. It will be positioned in their chosen location.

2.2.2 View coin count



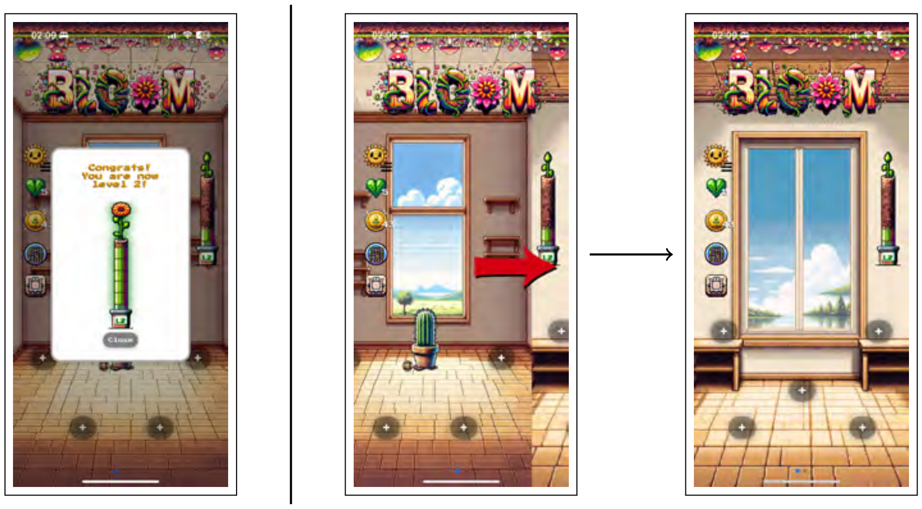
Coins are displayed on most screens in the top-right corner, but on the left icon tower in the main screen.

2.2.3 View hearts count & countdown


BACK
What kind of light do cacti prefer?
Incorrect, Try again! You have 4 heart(s) left.
Close
Bright, indirect sunlight
Complete darkness
Direct, midday sunlight
Low light conditions
BLOOM

Hearts are usually shown near the coin icon. When the user has fewer than 5 hearts, a countdown starts for the next free heart. The countdown stops at 5 hearts, regardless of purchases from the shop.

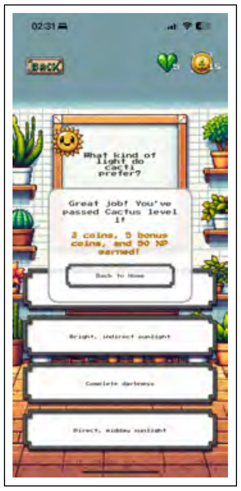
2.2.4 XP & new rooms



Each time users levels up, new rooms may become available, with extra spaces for plants. Presently, these unlock at specific milestone levels: 2, 5, and 8.

2.3 Shop

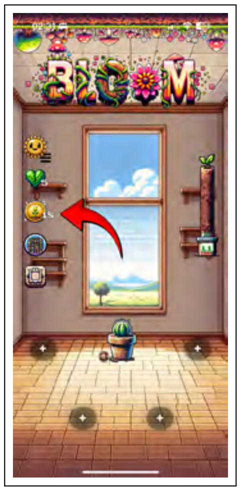
2.3.1 Earning coins


What kind of light do cacti prefer?

Great job! You've passed Cactus level 1!

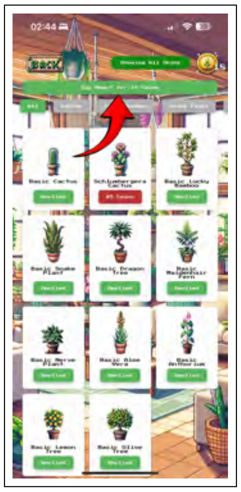
3 coins, 5 bonus coins, and 50 XP earned!

$\longrightarrow$

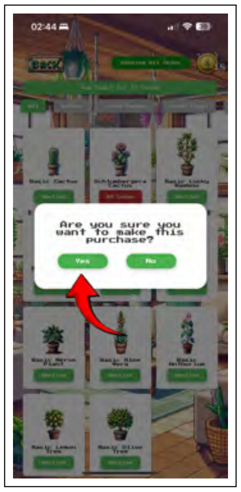


Users earn coins for each correct answer in quizzes. E.g. Answering all questions in a level with three questions grants three coins. On first completion of a level, up to five bonus coins are possible, with one subtracted per incorrect answer.

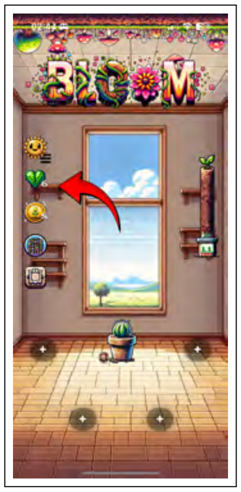
2.3.2 Purchase extra hearts



$\longrightarrow$


Are you sure you want to make this purchase?

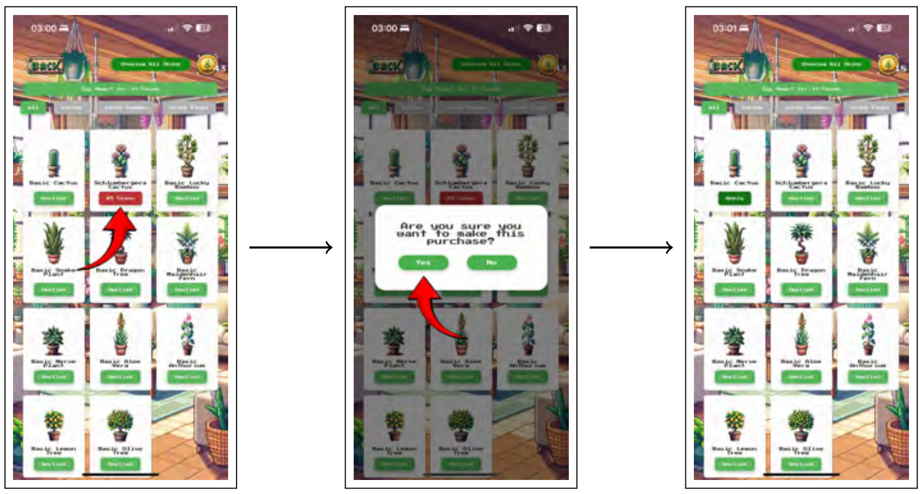
$\longrightarrow$



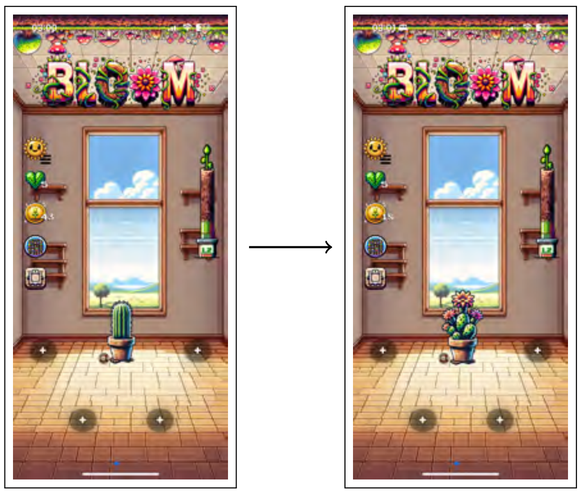
Hearts can be purchased for the price of 10 coins. A user simply has to tap and confirm the purchase in the shop. The purchased hearts will be added to the players heart count. If a player has fewer than 5 hearts, the heart restore timer will still stop at five regardless.

2.3.3 Purchasing & applying skins

Skins can each be purchased at their respective prices. They are named after real life plant species. Users can easily browse and select skins for specific plants using the scroll bar at the top of the screen. Once a skin is selected and the purchase is confirmed, it is automatically applied to the plant. Changing skins is then a simple process: users simply find a different skin and press 'Apply' to switch to the new choice.



Purchase Skin.

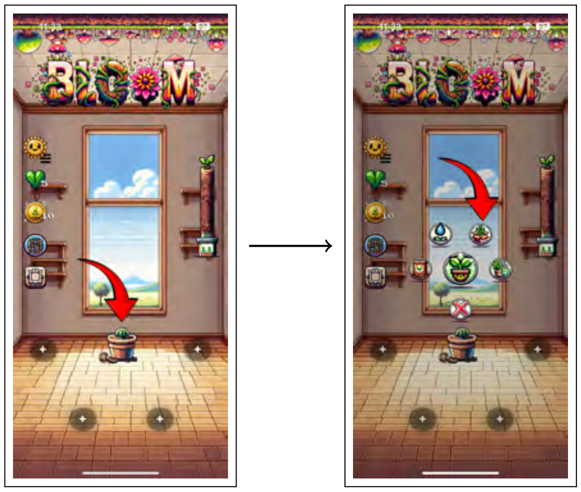


Apply Skin.

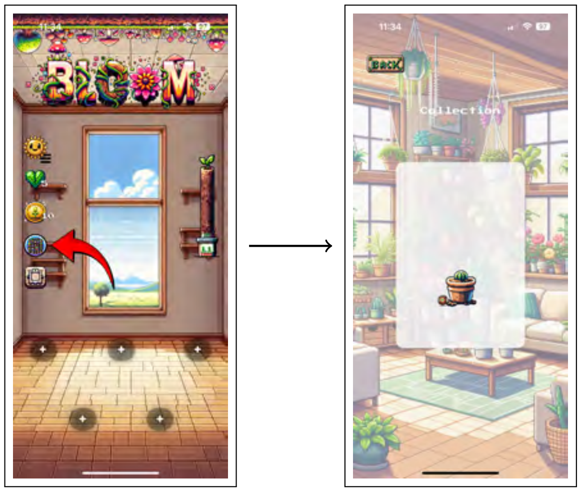
2.4 Collections

The collection feature is designed to help users manage their space efficiently. It provides a storage option for plants that cannot fit in the original hub, while still allowing users to monitor and track their real-life plants conveniently.

2.4.1 Add plant to collection



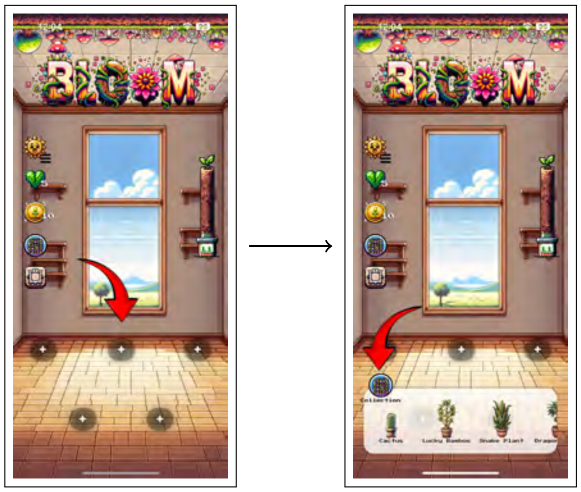
Press "Add to Archive" Icon.



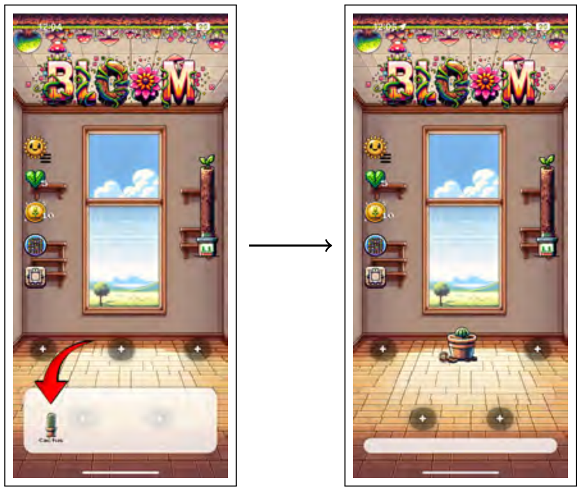
Press Collections Icon.

2.4.2 Add plant from collection

Similarly, plants can be re-placed from the archive if a user wishes to see their linked plant in the hub again.

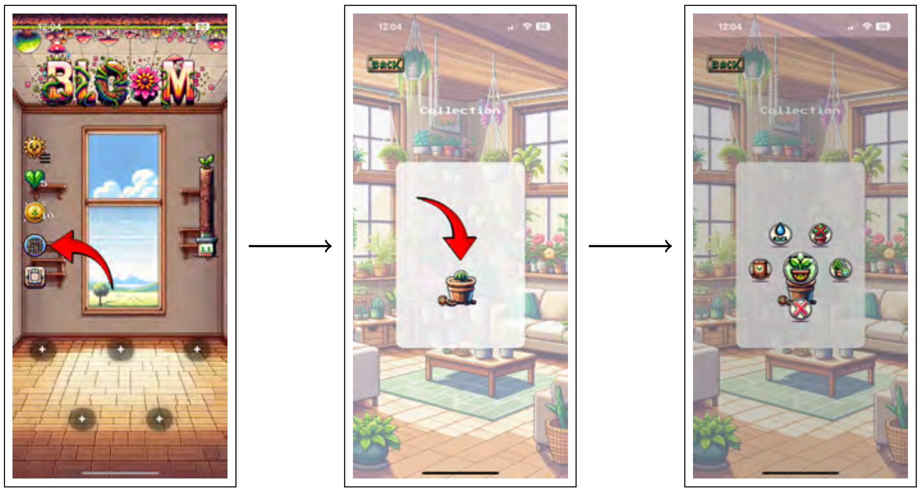


Press "+" icon, and select Collections button.



Select chosen plant, and it will remove it from archive and re-place it in hub.

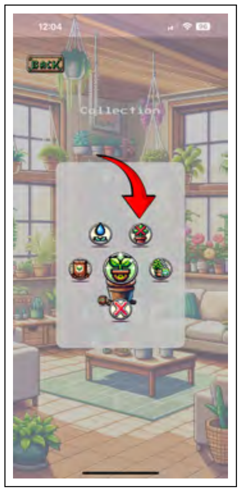
2.4.3 Interacting with archived plant



Within the collection, users are able to interact with their plants in the exact same way as they were able to in the hub. For more information, refer to section 2.6.

2.4.4 Delete plant

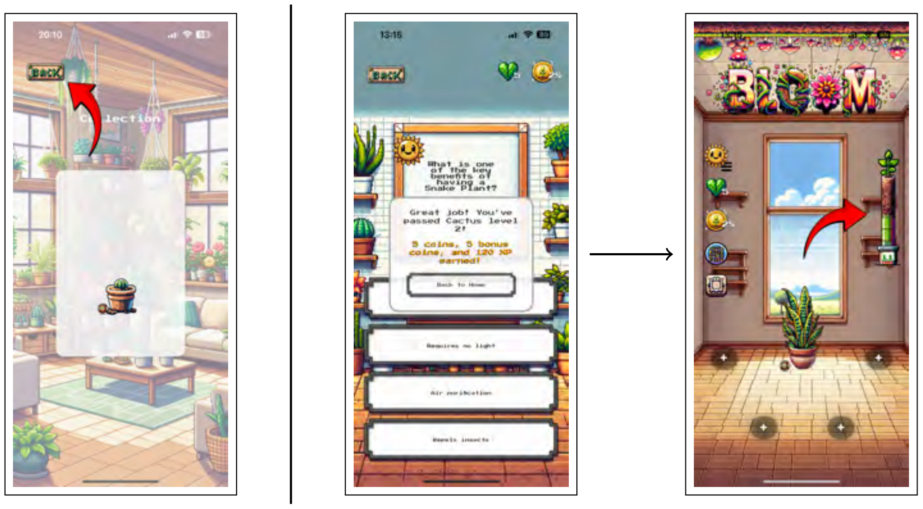
The main difference between plant interactions within the Collections Screen and the main hub is that the archive button changes to a delete button (see above figure). Pressing this will remove a plant from stored data permanently.



2.5 Masteries

A variety of indicators are available to keep players informed of progress:

2.5.1 Gaining XP & levelling up



What is one of the key benefits of having a Snake Plant?

Great job! You've passed Cactus level 2!

3 coins, 3 bonus coins, and 120 XP earned!

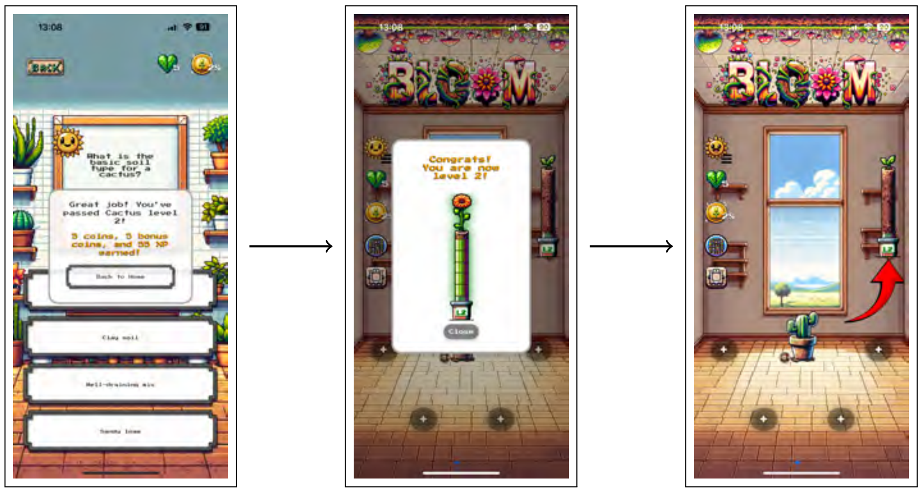
Back to Home

Requires no light

Air purification

Repels insects

XP can be earned through quiz completion or plant interactions, with the progress displayed on the right-side XP Bar.



What is the basic soil type for a cactus?

Great job! You've passed Cactus level 2!

3 coins, 3 bonus coins, and 55 XP earned!

Back to Home

Clay soil

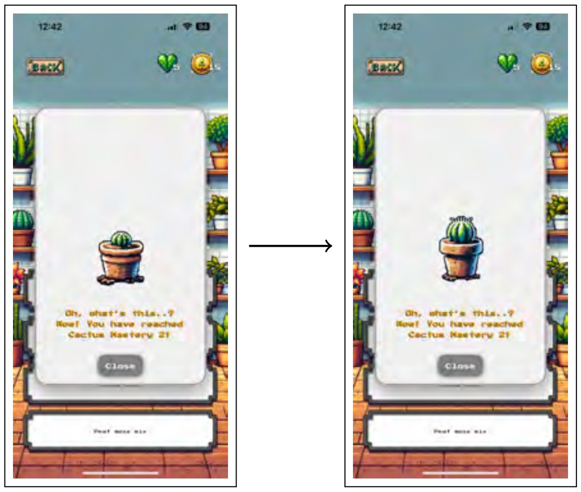
Well-draining mix

Sandy loam

Congrats! You are now level 2!

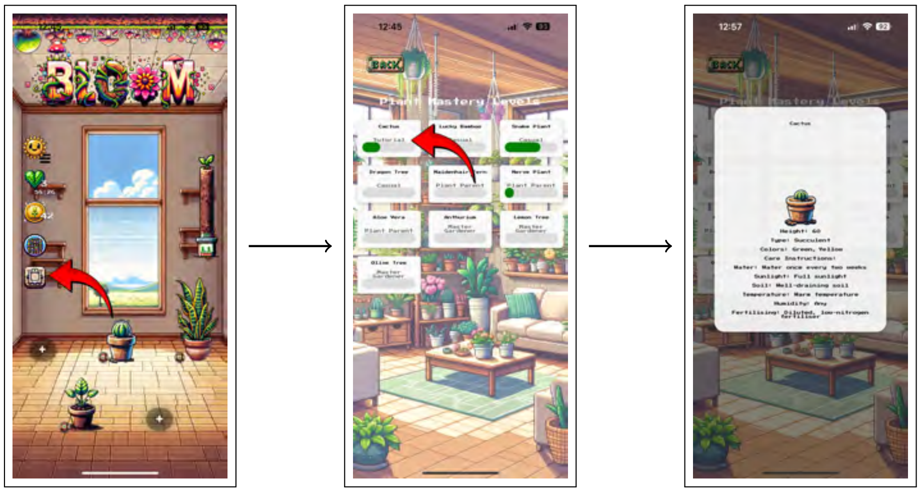
When sufficient XP is gained, a Level Up animation plays and the level increases on the XP Bar.

2.5.2 Mastery stats screen



Oh, what's this..?
Wow! You have reached
Cactus Mastery 2!

Masteries are gained by completing quizzes.



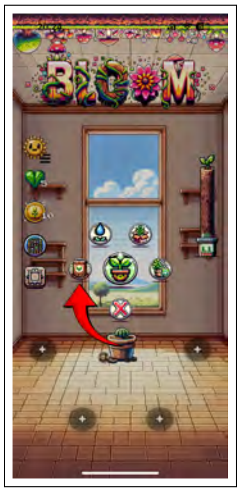
Height: 60
Type: Succulent
Colors: Green, Yellow
Care Instructions:
Water: Water once every two weeks
Sunlight: Full sunlight
Soil: Well-draining soil
Temperature: Warm temperature
Humidity: Any
Fertilising: Diluted, low-nitrogen fertiliser

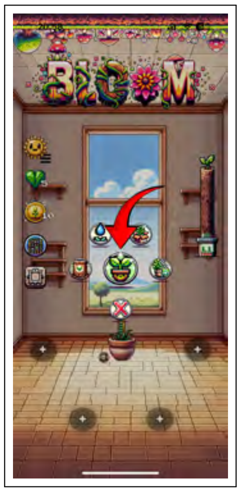
Press the Game Stats Icon to view user progress for each plant. You can then press a specific plant to view a quick reference handbook of its data.

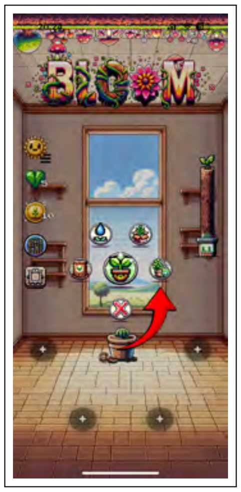
2.6 Plants

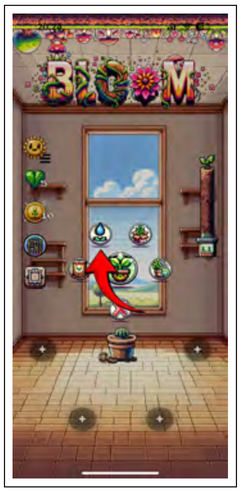
Plants may be interacted with in the following ways:

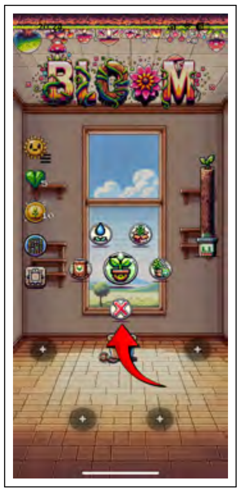
2.6.1 Plants menu

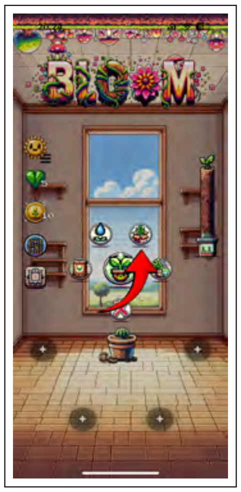

Fertilize Plant.


Learn!

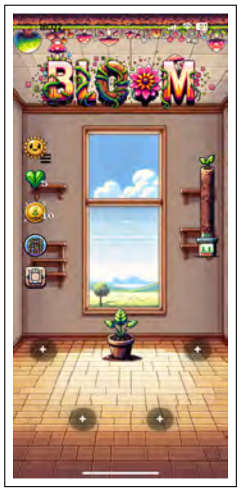

Link Plant.

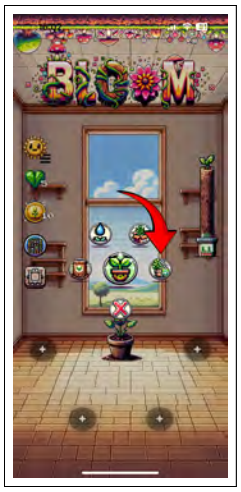

Water Plant.

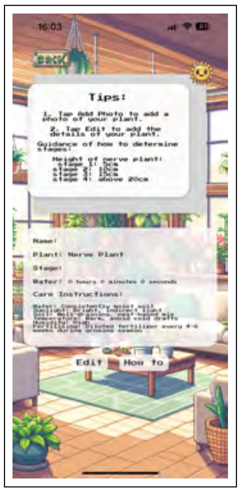

Close Plant.


Archive Plant.

2.6.2 Real plant link







Tips:

1. Tap Add Photo to add a photo of your plant.

2. Tap Edit to add the details of your plant.

Guidance of how to determine stages:

Height of nerve plant:
stage 1: 5cm
stage 2: 10cm
stage 3: 15cm
stage 4: above 20cm

Name:

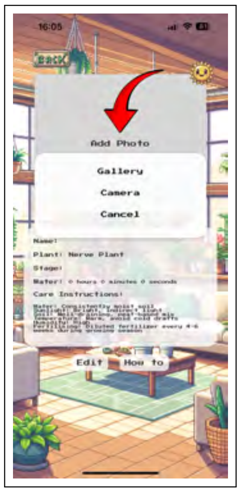
Plant: Nerve Plant

Stage:

Water: 0 hours 0 minutes 0 seconds

Care Instructions:

Water: Consistently moist soil
Sunlight: Bright, indirect light
Soil: Well-draining, peat-based mix
Temperature: Warm, avoid cold drafts
Humidity: High
Fertilizing: Diluted fertilizer every 4-6 weeks during growing season



Name:

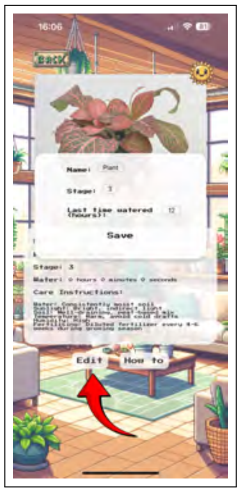
Plant: Nerve Plant

Stage:

Water: 0 hours 0 minutes 0 seconds

Care Instructions:

Water: Consistently moist soil
Sunlight: Bright, indirect light
Soil: Well-draining, peat-based mix
Temperature: Warm, avoid cold drafts
Humidity: High
Fertilizing: Diluted fertilizer every 4-6 weeks during growing season



Name: Plant

Stage: 3

Last time watered (hours): 12

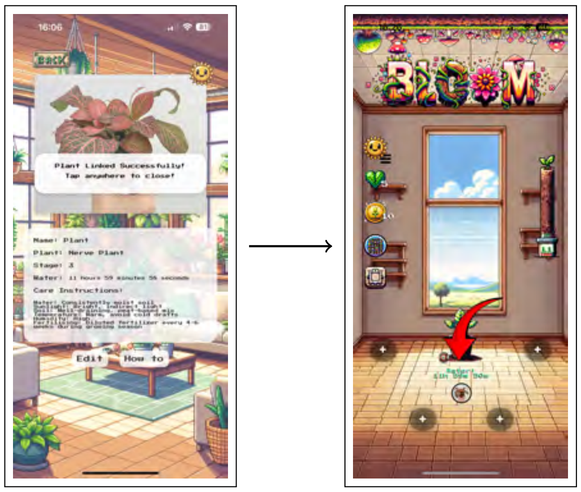
Stage: 3

Water: 0 hours 0 minutes 0 seconds

Care Instructions:

Water: Consistently moist soil
Sunlight: Bright, indirect light
Soil: Well-draining, peat-based mix
Temperature: Warm, avoid cold drafts
Humidity: High
Fertilizing: Diluted fertilizer every 4-6 weeks during growing season

To link a plant, click the link icon on the plant menu's bottom right. Users should follow the on-screen tips for data entry, upload a plant image, and add real-life details. A confirmation message will verify the successful link.

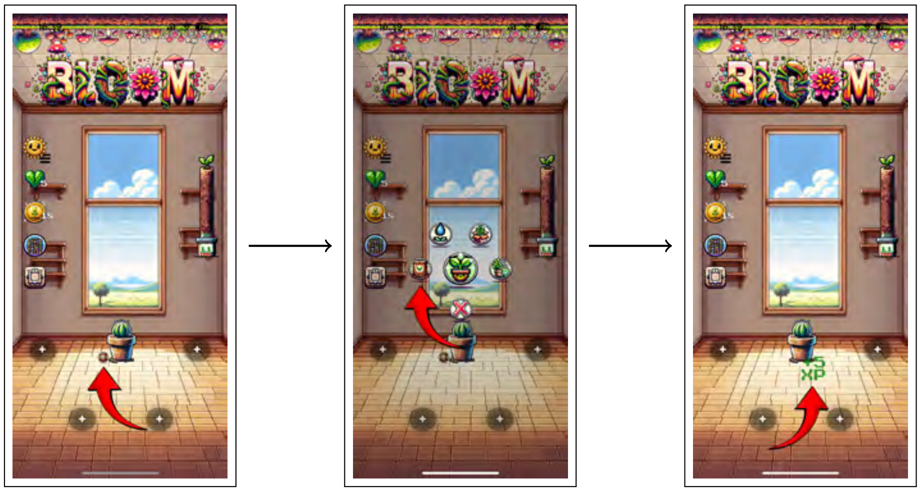


Plant Linked Successfully!
Tap anywhere to close!

Name: Plant
Plant: Nerve Plant
Stage: 3
Water: 11 hours 59 minutes 56 seconds
Care Instructions:
Water: Consistently moist soil
Sunlight: Bright, indirect light
Soil: Well-draining, peat-based mix
Temperature: Warm, avoid cold drafts
Humidity: High
Fertilizing: Diluted fertilizer every 4-6 weeks during growing season

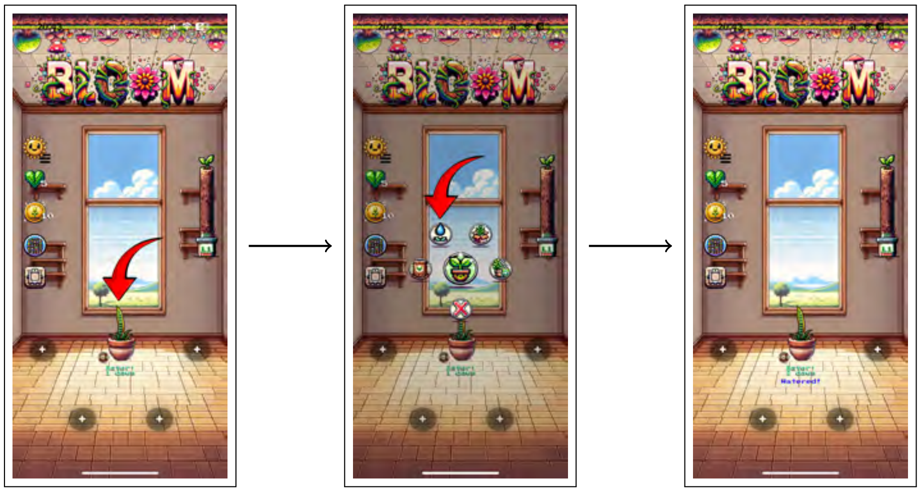
Following a successful link, a watering timer, as well as the linked plants image will be placed below the plant, to indicate the next watering time. This timer will be reset every time a user waters the plant (see sub-section 2.6.3).

2.6.3 Fertilise



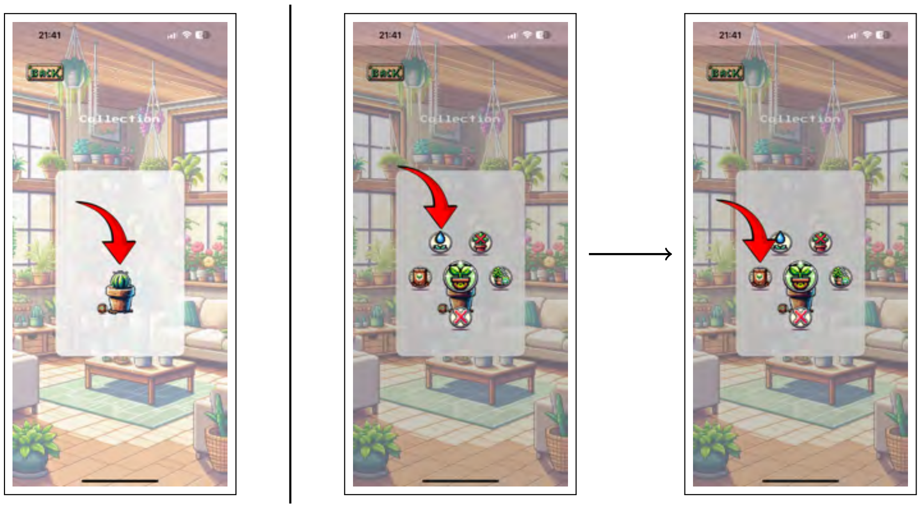
To link a plant, click the link icon on the plant menu's bottom right. Users should follow the on-screen tips for data entry, upload a plant image, and add real-life details. A confirmation message will verify the successful link.

2.6.4 Watering



Users have the ability to water a plant with or without linking it. The initial watering starts a countdown to the next required session for the plant. Subsequent waterings reset this timer, with players earn a small amount of XP each time.

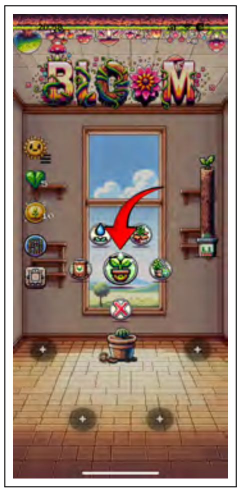
Watering and fertilizing tasks can also be conveniently performed via the Collections Screen.

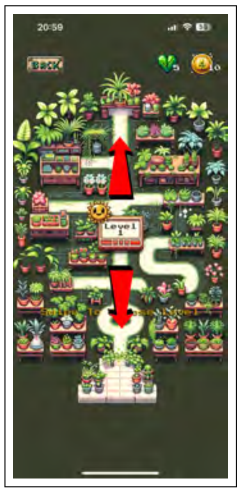


2.7 Quiz

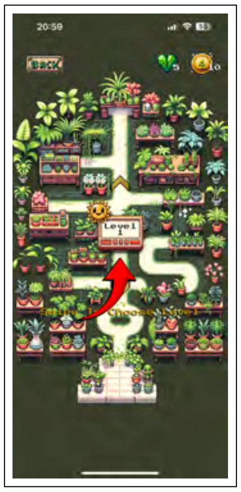
The Oracle offers engaging educational quizzes and practical tips to improve indoor gardening skills.

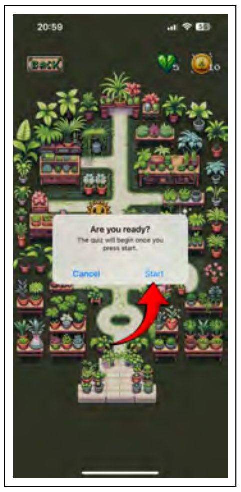
2.7.1 Select levels



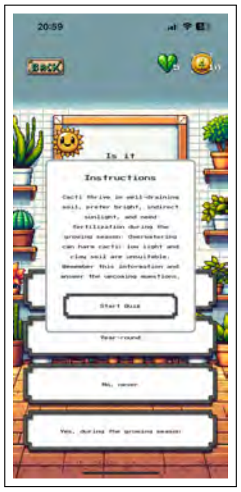


Press the central button to access the trivia screen, and swipe up or down to switch levels.





Are you ready?
The quiz will begin once you press start.
Cancel Start



Is it

Instructions

Cacti thrive in well-draining soil, prefer bright, indirect sunlight, and need fertilization during the growing season. Overwatering can harm cacti; low light and clay soil are unsuitable. Remember this information and answer the upcoming questions.

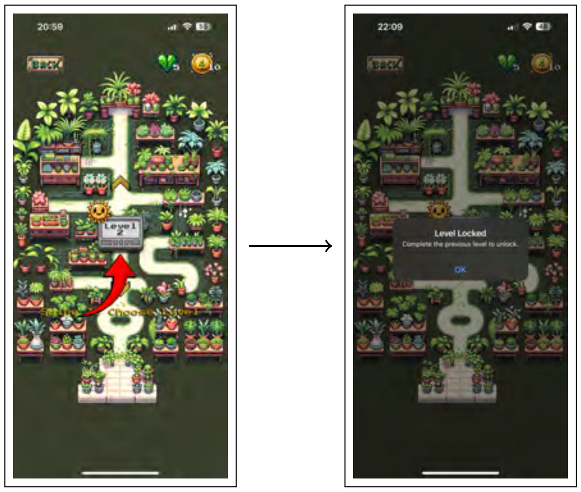
Start Quiz

Year-round

No, never

Yes, during the growing season

Select level and confirm to begin a quiz.

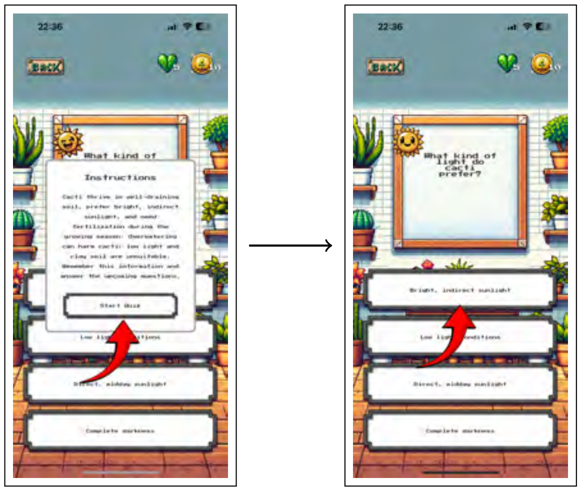


Level Locked

Complete the previous level to unlock.

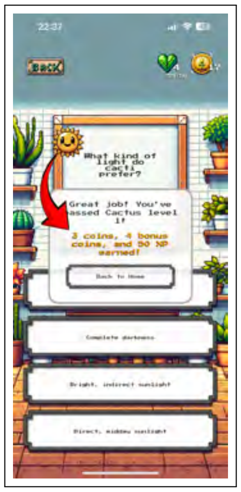
If a level is locked, the user will not be able to play it.

2.7.2 Quiz pass



Cacti thrive in well-draining soil, prefer bright, indirect sunlight, and need fertilization during the growing season. Overwatering can harm cacti; low light and clay soil are unsuitable. Remember this information and answer the upcoming questions.

Start quiz and navigate through answers.



What kind of light do cacti prefer?

Great job! You've passed Cactus level 1!

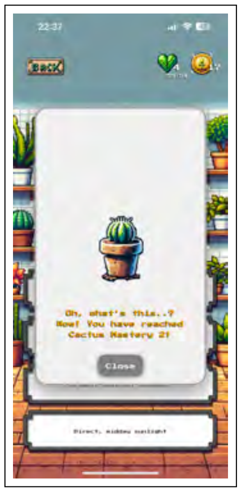
3 coins, 4 bonus coins, and 50 XP earned!

Back to Home

Complete darkness

Bright, indirect sunlight

Direct, midday sunlight



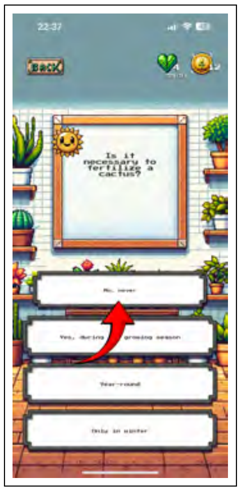
Oh, what's this..? Wow! You have reached Cactus Mastery 2!

Close

Direct, midday sunlight

Upon completion, experience and coins are earned. User also increases mastery.

2.7.3 Quiz fail



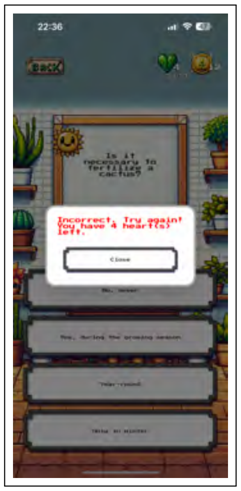
Is it necessary to fertilize a cactus?

No, never

Yes, during the growing season

Year-round

Only in winter



Incorrect. Try again! You have 4 heart(s) left.

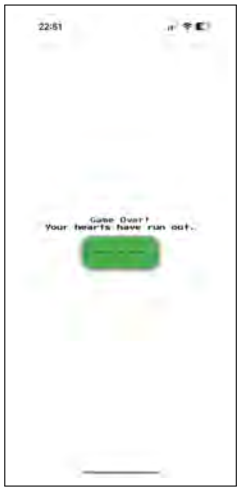
Close

No, never

Yes, during the growing season

Year-round

Only in winter



Game Over! Your hearts have run out.

If a user selects a wrong answer, they receive a prompt and lose a heart. If all hearts are lost, the game is over.

Part III

Maintenance Guide

1 About the maintenance guide

This chapter contains useful information about the maintenance guide itself.

How to use is the first thing anyone looking at the maintenance guide for the first time should read. It contains an overview of the entire guide, and highlights important sections depending on the reader's role.

Intended audience lists the people who might benefit from this guide.

Motivation explains the circumstances under which the guide was produced.

Purpose explains the purpose the authors intended the guide to serve, and the criteria it should fulfil.

Updating gives detailed instructions for updating the maintenance guide as development continues.

Support contains contact information for the authors.

Resources is a list of online reference guides and tutorials which might be generally helpful to developers on this project.

Glossary defines project-specific terminology used in the maintenance guide.

How to update

Any new terminology used in the project should be defined in the Glossary.

If the nature of the maintenance guide fundamentally changes, this should be reflected in the relevant sections: How to use, Intended audience, Motivation and Purpose.

Primary responsibility for the maintenance guide rests with Alec Mason. If he leaves the development team, or if another team member takes over the maintenance guide, Alec Mason's contact details should be removed from Support and replaced with the new lead author of the guide.

If the project framework changes, or if any major libraries are added, relevant resources should be added to the Resources section.

1.1 How to use

1.1.1 Overview

About the maintenance guide contains information about the guide itself: its history, and how a reader can get the most use out of it.

Overview of Bloom contains general information about the Bloom app.

Development tools lists the non-essential tools which a developer might use to streamline their work.

Dependencies lists the libraries and other software which is required to build the app.

Source code contains detailed documentation of each module and asset in the source code.

Structure gives an overview of how the app's various modules and libraries interact with one another.

Installation guide for developers gives first-time instructions for downloading the source code, so that you can begin making changes and additions.

Build instructions should be followed when the a version of the app is ready to be released to production.

Troubleshooting lists the issues which have been encountered when building the app, along with any known fixes.

Expansion lays out a roadmap for future development, along with advice for how its goals can be achieved.

1.1.2 Useful chapters for different readers

The chapters of the guide which are of most use will vary depending on who the reader is and what their interest is in in the project.

Current members of the development team will most often find themselves referring to the Source code documentation for all the information necessary to import and use a module or asset.

New members of the development team should, in time, familiarise themselves with the entire document, but will derive an immediate benefit from: the Glossary, which defines all project-specific terminology used within the development team; the Development ethos, which describes principles and practices to which developers should adhere; Development tools, which recommends useful tools including an IDE; and the Installation guide for developers, which will walk them through the process of downloading and testing the app on their computer.

1.2 Intended audience

This maintenance guide is freely available from the project's GitHub repository, and can be used by any party interested in how the Bloom app functions. This includes:

- The project's original developers, who may wish to return to it in future.
- Any independent developer who wishes to fork this project and work on it themselves.
- Any developer who wishes to reuse parts of this project for some other project.

1.3 Motivation

This maintenance guide, along with the rest of the Bloom project, was submitted for Coursework 2 of the Software Engineering unit at University of Bath. However, given the amount of time and effort invested in the project, the authors feel that it has value beyond being a coursework assignment. Some members of the development team may wish to include the project as part of a portfolio. They may also wish to return to this project and continue work on it after their studies are complete. In any case, the authors' intention is that this guide will offer comprehensive and practically useful documentation, written to a professional standard.

1.4 Purpose

As per the coursework specification, the maintenance guide's primary purpose is to provide: "detailed sign-posting of how to navigate the code and where and how to make extensions."

However, for the reasons discussed in the Motivation section above, the authors have chosen to expand the purpose of the guide. If development continues, this guide will act as a point of reference for the development team, and it should be continually updated as the project grows and changes.

The authors' intention is that any new member of the development team, or any interested third party, should be able to learn everything they need to know about the Bloom app from this guide. However, the guide assumes a basic knowledge of programming and app development; it is not intended as a replacement for training or education. Nor should members of the development team consider this guide a substitute for communication with other team members.

1.5 Updating

Keeping the maintenance guide up to date is the collective responsibility of the entire development team. If a developer makes a significant change or addition to the project source code, it is that developer's responsibility to reflect that change or addition in the maintenance guide. The guide should also be subject to periodic reviews by the entire team.

Each chapter in this guide should contain instructions for updating itself. These should detail the circumstances under which updates need to be made, the content that should be changed or added, and the structure and format of new material.

1.6 Support

The main point of contact for all matters relating to this maintenance guide should be Alec Mason, Bloom developer and maintenance guide author. Alec Mason can be contacted via GitHub (https://github.com/AlecSoren) or by email at asm213@bath.ac.uk

Other project developers can be contacted via GitHub. Their profiles are listed at https://github.com/dfoshidero/Bloom/graphs/contributors

1.7 Resources

Useful resources for developers on this project include:

- JavaScript documentation
  The official
  documentation for JavaScript is the ECMAScript standard, which is defined by
  https://ecma-international.org/publications-and-standards/standards/ecma-262/
  A more accessible guide to JavaScript is available at
  https://developer.mozilla.org/en-US/docs/Web/JavaScript
- React documentation
  React is the main library used for this project. Documentation is available at
  https://react.dev/
- React Native documentation
  React Native is a framework which allows React projects to be built as mobile apps. Documentation is available at
  https://reactnative.dev/docs/getting-started

1.8 Glossary

The glossary should define terms used in the maintenance guide which meet the following criteria:

- Technical language which might not be obvious to a generalist developer.
- Terminology relating to React, React Native or any other library used in the project.
- Ambiguous terminology (i.e. terms whose meaning varies by field) which has a specific, consistent meaning in this project.
- Language used idiosyncratically within the project to refer to app mechanics or code elements.

Archive The act of moving a plant to the collection.

Background

1. The background image on a screen.
2. A **room**.

Bloom The app which is the subject of this maintenance guide.

Coin Bloom's in-game currency used to purchase skins and hearts.

Collection Contains all plants which are not currently inside a room.

Component A React component. See https://react.dev/reference/react/Component

Developer Any individual making changes to or inspecting the source code, whether they are a member of the development team or not.

Development team The official developers for Bloom. See Initial development for a full list.

Fertilise The user can tap a button to fertilise a plant, which rewards them with XP.

Heart Resource which allows the user to complete quizzes. Each wrong answer loses a heart.

Inventory Contains each species of plant, allowing the user to add new plants to their room.

Link (to real life) See Real plant link

Mastery Each plant species has a mastery level which tracks how many quizzes the user has completed for that species.

Navigate See https://reactnavigation.org/docs/navigating

Navigator See https://reactnavigation.org/docs/stack-navigator

Module An object, function or source code file. See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules for more information about modules in JavaScript.

Plant interaction When the user taps a plant, it displays a menu with various interaction options.

Player See **User**

Real-life link See Real plant link

Room The main game environment, which simulates a room in a house.

Screen See Screens

Skin An alternative set of images which change the appearance of a plant.

Slot A position where a plant can be placed inside a room.

Species

1. A type of in-game plant (e.g. cactus, olive tree, snake plant) with its own trivia, mastery level and skin family.
2. A real plant species, used as the basis for a purchasable skin.

State See States

Trivia See **Quiz**

User An individual using Bloom for its intended purpose (i.e. not for developing or testing).

Quiz A multiple-choice quiz on caring for a particular species of plant, to be completed by the user.

XP Resource earned by the user when they complete quizzes and fertilise plants. See Gaining XP & levelling up

Water If a plant is linked to real life, it will require watering at regular intervals. The user does this by tapping a button in the plant interaction menu.

2 Overview of Bloom

This chapter contains general information about the Bloom app.

Motivation and purpose describes Bloom's raison d'être: the circumstances under which it was envisioned and the requirements it meets.

Game mechanics lists Bloom's core mechanics.

Development history is a record of how and when Bloom was developed.

Development ethos considers the development philosophy of Bloom. Members of the development team should strive to incorporate these principles in their work.

How to update

Motivation and purpose and Game mechanics contain historical information about Bloom which will not change (unless it is found to be inaccurate), but may no longer be relevant if the nature of the project changes. In that case, it will likely be necessary to completely restructure these sections.

Any new production releases should be added to Stable releases.

If development continues, it will be necessary to add a new subsection to Development history for the new development phase.

The principles described in Development ethos are flexible and subject to change according to the collective will of the development team. The documentation should be changed to reflect the actual practices used by the team, rather than forcing the team to follow the documented practices even if they are no longer helpful.

2.1 Motivation and purpose

Bloom was created as part of Coursework 2 of the Software Engineering unit at University of Bath. Coursework 2 was intended to provide practical experience of the software engineering process, teaching students essential collaboration skills and assessing their ability to document, review and improve their performance.

The requirements for the project were set by members of the teaching staff playing the role of customers. These requirements evolved throughout the duration of the project, and were negotiated in weekly customer meetings.

The initial project specification was to create a serious game (i.e. a game with a purpose other than pure entertainment, such as teaching or advertising) with "tasks that vary or increase in difficulty" and "an oracle so I [the user] can get advice when I am stuck".

Bloom meets these criteria in the following ways:

- It offers gameplay in the form of quizzes.
- These quizzes teach the user how to take care of houseplants.
- Subsequent quizzes increase in difficulty, as new questions are introduced along with repeated questions from previous quizzes.
- The oracle is the sun, which appears in the app in several different places and offers advice on a range of game mechanics.

2.2 Game mechanics

Bloom's core mechanics are listed below. This is not intended to be a comprehensive list, but rather the bare minimum functionality Bloom must have to fulfil its purpose as a serious game. For comprehensive and detailed information on game mechanics, see the Detailed user manual.

- The main game environment is a room, which contains several slots for plants.
- The sun is visible from the room, and offers advice when selected.
- New plants can be added from a menu which offers a choice of species.
- Each plant presents an option for the user to attempt a quiz.
- The user can choose which level of quiz they want to attempt. Previously completed quizzes can be re-attempted to 'revise' for the next quiz, but users must complete new quizzes sequentially - they cannot skip ahead.
- Before the quiz begins, the sun appears and shows the user information which is needed for the quiz.
- A quiz consists of multiple-choice questions about caring for houseplants. When the user answers a question correctly, the game will tell them so, and proceed to the next question. If their answer is incorrect, they will lose a heart, and must repeat the question.

- The user cannot attempt quizzes if they have no hearts remaining. Hearts replenish over time.
- When a quiz is completed successfully, all plants of the corresponding species will grow to a new stage.

2.3 Development history

2.3.1 Stable releases

Version	Date	Notes
1.0	11/12/23	First stable release, submitted for Software Engineering Coursework 2

2.3.2 Initial development

Bloom was developed over a 9 week period in October-December 2023. The development team consisted entirely of MSc Computer Science students at the University of Bath, whose names are as follows:

- Udit Bhatia
- Pang Hoi (Eddie) Chan
- Marat Danyarov
- Siqi (Claire) He
- Alec Mason
- Daniel-Favour Oshidero
- Yan Chun (Ivan) Yeung

For further details concerning the initial development phase, consult the Process Document which is available at
https://github.com/dfoshidero/Bloom/tree/delivery/deliverables

2.4 Development ethos

This project follows the principles of agile software development (see http://agilemanifesto.org/), and uses the Scrum framework for planning and project management.

Development should focus on making deliverable builds which satisfy user requirements. Documentation for documentation's sake is strongly discouraged, and any redundant documentation should be deleted (but preserved in version history).

When making changes or additions to the source code, nothing should be considered off-limits to a knowledgeable developer who understands the code base. Any part of the project, from its core purpose to implementation details, should be open for discussion and review by the whole team.

Decisions should be made collaboratively, although team members may temporarily take leadership roles for specific aspects of the project. When a consensus cannot be reached, it is the duty of each team member to work towards a compromise, bearing in mind that progress is more important than taking a perfect approach.

3 Development tools

This chapter describes tools which are not an essential part of storing or building the app, but may nevertheless be useful to developers.

How to update

Any and all tools used by the development team should be documented somewhere in the maintenance guide - if not elsewhere, then here. A new section should be created for each tool. Avoid removing tools from this chapter, unless all files and assets created with those tools have been removed from the project.

3.1 DALL·E

DALL·E by OpenAI is used to create image assets for Bloom. Any team member using DALL·E should ensure that they are familiar with OpenAI's terms of use, accessible at https://openai.com/policies/terms-of-use

Image assets generated in this way should be checked for visual consistency with the rest of the app. It may be necessary to modify the image using image editing software, e.g. to make the background transparent.

3.2 Visual Studio Code

Visual Studio Code is the recommended IDE for this project, due to its suitability for JavaScript and built-in version control support. Instructions for use are available at https://code.visualstudio.com/learn

4 Dependencies

This chapter lists the libraries and programming languages used for the project.

How to update

For all sections in this chapter, the library version must be updated to match what is actually being used for the project (which is often not the most recent version).

Libraries, Development dependencies and Programming languages should be added or removed as necessary, depending on whether they are actually used in the source code.

It is unlikely that the project framework will change, but if it does, then the React Native section must be rewritten accordingly.

4.1 React Native

Bloom uses the React library (Version 18.2.0) for its user interface. React Native (Version 0.72.7) is used to build the app for Android.

4.2 Libraries

Other libraries used for the app are listed below:

- `@react-native-async-storage/async-storage`
  Version 1.19.6
- `@react-navigation/native`
  Version 6.08
- `@react-navigation/stack`
  Version 6.2.0

- `expo`
  Version 49.0.15
- `expo-av`
  Version 13.4.1
- `expo-font`
  Version 11.4.0
- `expo-image-picker`
  Version 14.5.0
- `expo-status-bar`
  Version 14.5.0
- `react-countdown`
  Version 2.3.5
- `react-native-animatable`
  Version 1.4.0
- `react-native-gesture-handler`
  Version 2.14.0
- `react-native-linear-gradient`
  Version 2.8.3
- `react-native-pager-view`
  Version 6.2.2
- `react-native-reanimated`
  Version 3.6.1
- `react-native-responsive-fontsize`
  Version 0.5.1
- `react-native-restart`
  Version 0.0.27
- `react-native-safe-area-context`
  Version 4.5.3
- `react-native-screens`
  Version 3.21.0
- `react-native-swiper`
  Version 1.6.0
- `react-native-touchable-scale`
  Version 2.2.0
- `react-native-vector-icons`
  Version 10.0.2

- `react-navigation`
  Version 5.0.0

4.3 Development dependencies

Babel (Version 7.20.0) is used to compile the source code.

`dependency-cruiser` (Version 15.5.0) generates dependency diagrams for the maintenance guide.

4.4 Programming languages

Bloom is mainly written in JavaScript. React uses HTML to define components and CSS for styling.

5 Source code

This chapter contains documentation for each module and asset in the source code.

How to update

Whenever a module or asset is added, removed or modified, this chapter must be updated to match. When adding new documentation, use the structure guide at the start of the relevant section.

5.1 App

Source File

App.js

Behaviour

The App component serves as the main entry point of the application. It initializes state management and navigation structures. It begins with a loading screen (LoadingScreen), transitioning to the main app interface through the HomeStackNavigator upon completion of loading. It also sets the base app background to black, and defaults the status bar to hidden for a more immersive experience.

Properties

- `SafeAreaProvider` and `SafeAreaView`: Ensures compatibility with device screen edges and notches.
- `StatusBar`: Hidden for an immersive experience.
- `NavigationContainer`: Manages navigation throughout the app.
- `PlayerConfigProvider`, `PlantDataProvider`, `SpeciesProgressProvider`, `CompletedLevelsProvider`: Context providers for managing global state.

Internal variables and functions

- `isLoading` (state): Determines whether the loading screen is displayed.
- `handleFinishLoading` (function): Sets `isLoading` to false, indicating loading completion.

Dependencies

React Native:
`Platform`, `StatusBar`, `View`

react-native-safe-area-context:
`SafeAreaView`, `SafeAreaProvider`

@react-navigation/native:
`NavigationContainer`

Custom Providers and Components:
`HomeStackNavigator`, `PlayerConfigProvider`, `PlantDataProvider`, `CompletedLevelsProvider`, `SpeciesProgressProvider`, `LoadingScreen`

5.2 Assets

Asset files are located in `src/assets`

These include images, fonts and audio.

5.2.1 backgrounds

Background images.

- `1.png`
  Background for the default room.
- `2.png`
  Background for room 2.
- `3.png`
  Background for room 3.
- `4.png`
  Background for room 4.
- `misc/congrats_screen.png`
  Displays when a quiz is completed.
- `misc/level_select.png`
  Displays when the user is selecting a trivia level to attempt.
- `misc/loading_screen1.png`
  Loading screen.
- `misc/loading_screen2.png`
  Alternate loading screen.
- `misc/menu_bg.png`
  Displays in the background of 'menu' screens.
- `misc/quiz_screen.png`
  Displays in the background while the user is attempting a quiz.

5.2.2 fonts

Fonts:

- `DotGothic16`
  Source:
  `assets/fonts/DotGothic16-Regular.ttf`
  This file contains the DotGothic16 Regular font, a modern and versatile typeface suitable for various design needs.
- `PressStart2P`
  Source:

`assets/fonts/PressStart2P-Regular.ttf`
This file comprises the PressStart2P Regular font, characterized by its pixelated, retro gaming-inspired style, evocative of classic video game text.

5.2.3 header

`header.png` and `header_liner.png`
These files contain the Header components used in the application. They provide a top navigation or title bar for the app's interface.
Source: `assets/header`
Usage: Header

5.2.4 icons

Source: `assets/icons/xp`

- `xp`
  `0.png`, `10.png`, `30.png`, `50.png`, `70.png`, `90.png`, `lvl_container.png` are files containing images for different experience levels. They represent functionality related to plant mastery.
  Usage: XPBar
- `archive_icon.png`
  This file contains the image of the 'Archive' icon used in the application. It represents functionalities related to archiving or storing data.
  Usage: Plant
- `back_icon.png`
  This file contains the image of the 'back' icon used in the application to return the game to the main screen.
  Usage: QuizQuitButton, HomeStackNavigator
- `close_icon.png`
  This file contains the image of the 'close' icon used in the application to close the plant menu.
  Usage: Plant
- `coin_icon.png`
  This file contains the image of the 'coin' icon used in the application. It represents functionalities related to in-game rewards.
  Usage:CoinDisplay
- `collection_icon.png`
  This file contains the image of the 'collection' icon used in the application to display the collection screen.
  Usage: CollectionButton
- `delete_icon.png`

This file contains the image of the 'delete' icon used in the application and related to plant functionalities.
Usage: Plant

- `down_icon.png`, `up_icon.png`
  These files contain the images of the 'down' and 'up' icons used in the application and related to the quiz screen and level selection functionality.
  Usage: LevelSelectionScreen

- `fertilize.png`
  This file contains the image of the 'fertilise' icon used in the application and related to the plant component.
  Usage: Plant

- `game_stats_icon.png`
  This file contains the image of the 'game statistics' icon used in the application and related to the plant mastery functionality.
  Usage: GameStatsButton

- `hearts_icon.png`
  This file contains the image of the 'hearts' icon used in the application and related to the plant component.
  Usage: HeartsDisplay

- `learn_icon.png`
  This file contains the image of the 'learn the plant' icon used in the application related to the plant component and quiz functionality.
  Usage: Plant

- `level_icon_.png`
  `level_icon_complete.png`, `level_icon_locked.png`, and `level_icon_unlocked.png` are files containing the images for 'levels status' icons used in the application to the level select screen and quiz functionality.
  Usage: LevelSelectionScreen

- `link_icon.png`
  This file contains the image of the 'link the real plant' icon used in the application related to the plant component and linking a real plant to the app functionality.
  Usage: Plant

- `text_box.png`
  This file contains the image of the 'text box' icon used in the application related to the quiz functionality.
  Usage: QuizScreen

- `water_icon.png`
  This file contains the image of the 'water a plant' icon used in the application related to the plant component and functionality.

Usage: Plant

5.2.5 oracle

Source: `assets/oracle_edit`
`sun_big_smile_1.png`, `sun_big_smile_2.png`, `sun_normal.png`, `sun_sad.png`, `sun_smile_1.png`, `sun_smile_2.png`, `sun_surprise.png` are files used to display the Oracle component.
Usage: Oracle

5.2.6 plants

Source: `assets/plants`
This directory contains images of various plants at different stages of their development. Each plant has multiple images representing its growth from a seedling to full maturity.
Usage: plantsConfig

5.2.7 sounds

Source: `assets/sounds/music`
`8-bit-arcade.mp3`, `feed-the-machine-classic-arcade-game.mp3`, `short-circuits-classic-arcade-game.mp3` contain the background music tracks used in the application. The music is selected to enhance the user experience, providing an ambient and engaging atmosphere.
Usage: backgroundMusic.js

5.2.8 uncategorised assets

- `icon_container.png`
  This file contains a template for all other icons in the application.
- `adaptive_icon.png`, `icon.png`
  These files are variations of the main application icon.

5.3 Components

Component source files are located in `src/components`

Each component source file exports exactly one React `Component`. See https://react.dev/reference/react/Component for more details.

The documentation for each component is structured as follows:

Source file

The file which exports the component. New component files should be named according to the convention `...Component.js`

Behaviour

An overview of what the component does.

Properties

The properties which can be customised for each instance of the component. For more information, see
https://reactnative.dev/docs/props?language=javascript

Internal variables and functions

These are the variables and functions defined in the source code which generates the component. This does not include variables and functions imported from other modules. Note that these are not exported and cannot be accessed/called on the component or instances of the component (although they can be passed as properties to child components).

Dependencies

The list of libraries and modules used by the component.

Usage

The list of modules which use the component.

5.3.1 CoinDisplay

Source file

`CoinComponent.js`

Behaviour

This component renders the coin icon on different screens. `textColor` is set to be `balck` and `shadowColor` is set to be `white` if the number of coins are 0, else the `textColor` is set to be `white` and `shadowColor` is set to be `black`.

Properties

`styles`
Defines the style of the container of the coin icon when it is being used on other screens

Internal variables and functions

`textColor`
Could either be `black` or `white`

`shadowColor`
Could either be `black` or `white`

Dependencies

Reactive Native:
`View`, `StyleSheet`, `Image`

Usage

`GameScreen.js`, `LevelSelectScreen.js`, `QuizScreen.js`

5.3.2 CollectionButton

Source file

`CollectionComponent.js`

Behaviour

This component renders the collection screen icon on the main screen. It is also responsible for navigating the user to `ColectionScreen.js` when the user taps the icon.

Properties

`styles`
Defines the style of the container of the coin icon when it is being used on other screens

Internal variables and functions

`handlePress`
A function that handles the actions that need to be done when the user taps the collection icon.

Dependencies

Reactive Native:
`StyleSheet`, `Image`

Libraries:
`react-native-touchable-scale`

Usage

`GameScreen.js`

5.3.3 EvolveAnimation

Source file

`EvolveAnimation.js`

Behaviour

This component renders the animation of evolving a plant after finishing a quiz. The component will use the two properties `currentImage` and `nextImage` to render the animation.

Properties

`currentImage`
Stores the image of the plant in the current stage.

`nextImage`
Stores the image of the plant in the next stage.

Internal variables and functions

`slidingStyle`
A function that defines the sliding animation of the plant

Dependencies

Reactive Native:
`StyleSheet`, `Image`, `View`, `Animated`

Usage

`QuizScreen.js`

5.3.4 FloatingMenu

A component which defined the style of a circular menu where users could perform actions.

Source file

`CircularMenu.js`

Behaviour

If the `visible` property is set to `true`, the component will map the position and the item content using `menuItem`. If `visible` is set to `false`, the whole menu will not be accessible and visible to the user.

After all the position and item content are mapped, the component will load and display the items and images associated with each button with an expanding animation.

Depending on which item is being pressed by the user, the menu will call specific function/s.

When the close button is pressed `visible` will be set to `false` and the circular menu will be archived with a contracting animation.

Properties

`visible`
`true` if the component is opened, `false` if it is being archived

`menuItem`
Developers could pass variables to the CircularMenu component using format `{ icon: , isImage: true, angle: , id: }`, where `icon` is the image of the menu content, `angle` is the position of the button and `id` is an identifier of the item.

Internal variables and functions

`getTransformStyle`
This internal function renders the position of each item in the menu based on the property `angle`. It also defines the animation which is the path of each icon for the menu to expand.

Dependencies

Reactive Native:
`View`, `StyleSheet`, `Animated`, `Easing`, `Image`, `Modal`, `Dimensions`

Libraries:
`react-native-touchable-scale`,

`react-native-vector-icons/FontAwesome`

Usage

`PlantComponent.js`

5.3.5 GameStatsButton

Source file

`GameStatsComponent.js`

Behaviour

This component renders the Game stats button and handles the action performed when this button is being pressed.

Properties

`styles`
Defines the style of the container of the game stats button when it is being used on other screens.

Internal variables and functions

`handlePress`
A function that would be called when the user presses the button, which will navigate the user to the game stats screen.

Dependencies

Reactive Native:
`Image`, `StyleSheet`

Libraries:
`react-native-touchable-scale`

Usage

`GameScreen.js`

5.3.6 getPlantHitbox

Source file

`PlantHitbox.js`

Behaviour

This component generates the hitboxes of the plants on the main screen as different stages would need different sizes of hitboxes. The hitbox will be calculated using `progress`.

Properties

`progress`
Stores the progress of the plant

Internal variables and functions

`height`
Stores the height of the hitbox

Dependencies

N/A

Usage

`plantComponent.js`

5.3.7 Header

Source file

`HeaderComponent.js`

Behaviour

This component calculates the position of the header of the game screen based on the dimensions of different devices and renders the images needed.

Properties

N/A

Internal variables and functions

`deviceWidth`
Stores the width of the device's screen

`deviceHeight`
Stores the height of the device's screen

Dependencies

Reactive Native:
`View`, `StyleSheet`, `Image`, `Dimensions`

Usage

`GameScreen.js`

5.3.8 HeartsDisplay

Source file

`HeartsComponent.js`

Behaviour

This component renders the heart icon, heart refresh timer and the remaining lives that the player has. If the remaining lives are 0 then `textColor` will be black and `shadowColor` will be white. If it is not 0 then `textColor` will be white and `shadowColor` will be black. If `hearts` is smaller than 5, the timer will start by itself, counting down 1 hour.

Properties

`styles`
Defines the style of the container of the heart when it is being used on other screens.

Internal variables and functions

`textColor`
The color of the number of remaining lives.

`shadowColor`
The color of border of the number of remaining lives.

`renderer`
A function that renders a timer in the format of hh:mm:ss.

Dependencies

Reactive Native:
`View`, `StyleSheet`, `Image`

Libraries:
`react-countdown`

States:
`playerConfigContext.js`: `hearts`

Usage

`GameScreen.js`, `LevelSelectScreen.js`, `QuizScreen.js`

5.3.9 MenuComponent

Source file

`MenuComponent.js`

Behaviour

A component that renders the menu of the game at `GameScreen.js`. When `menuVisisble` is set to `true` the contents in the menu will be rendered and shown to the user using a modal built-in slide animation. When the user clicks outside of the menu `menuVisisble` will be set to `false`, and the menu will collapse. `menuVisisble` will also be set to `false` if the user clicks on one of the items in the menu.

Properties

`menuVisible`
`true` if the component is opened, `false` if it is being archived

`closeMenu`
A function to toggle the state of `menuVisible` (`true`/`false`)

Internal variables and functions

`clearData`
A function to handle clearing the saved data of the game

Dependencies

Reactive Native:
`View`, `StyleSheet`, `Image`, `Modal`, `NativeModules`, `TouchableOpacity`

Libraries:
`@react-native-async-storage/async-storage`
`react-native-touchable-scale`

States:
`plantsDataContext.js`
`speciesProgressContext.js`
`playerConfigContext.js`
`completedLevelsContext.js`

Usage

`GameScreen.js`

5.3.10 Oracle

Source file

`OracleComponent.js`

Behaviour

This component renders the oracle icon, which changes randomly in a random time frame of 2 seconds to 10 seconds. It will also blink naturally between 2 seconds and 7 seconds, with a 10 percent chance to blink twice.

Properties

`styles`
Defines the style of the container of the oracle when it is being used on other screens.

Internal variables and functions

`changeImage`
A function that will be called randomly to change the oracle image to a different form (e.g. from oracleNormal to oracleSmile).

`performBlink`
A function that contains the main logic of rendering the blink images and randomizing the blinks.

Dependencies

Reactive Native:
`Image`

Usage

`GameScreen.js`, `LevelSelectScreen.js`, `QuizScreen.js`, `PlantLinkComponent.js`

5.3.11 Plant

A component which displays either a plant, or a slot in which a plant can be added.

Source file

`PlantComponent.js`

Behaviour

If the `isArchived` property is set to `true`, the component will search the saved plant data for a plant with an `archiveID` that matches the component's id property. If isArchived is set to `false`, it will look for a matching `plantPosition ID` instead.

If a matching ID is found, the component will load and display the image associated with that plant at its current stage of progress. Otherwise, the component will display a + sign, prompting the user to add a new plant.

If the component is tapped while it is still displaying a + sign, it will bring up the SelectPlantModal.

If the component is tapped while it is displaying a plant, it will bring up a circular menu with options to attempt a quiz, link this plant to a real-life plant, water the plant, fertilise the plant, and archive the plant (if it is on the GameScreen) or permanently delete the plant (if it is already in the collection).

Properties

`currentBackgroundID`
`number`, the ID of the room which this plant component is in, if applicable.

`id`
`number`, a unique identifier for the plant component given by the screen in which it appears. If this plant component is in a room, the `id` will correspond to its position in that room. If this plant component is on the CollectionScreen, the `id` will correspond to an `archiveID` of a saved plant.

`isArchived`
`true` if the component is in the Collection Screen, `false` if it is in the Game Screen

`style`
`StyleSheet` with default styling for the plant component.

Internal variables and functions

`formatCountdownTime()`

Function that accesses the remaining time on the watering countdown, and converts it to a format which is readable by the user. Returns a `string`.

`handleAddPlantPress()`
Shows the SelectPlantModal when the user selects the slot to add a new plant.

`handleArchiveButtonPress()`
Called when the user touches the button to send this plant to the collection. Modifies the saved plant data, setting this plant's `plantPositionID` to `null` and giving it a unique `archiveID`, and resets the plant component to an empty slot.

`handleDeleteButtonPress()`
Called when the user selects the option to delete the plant. Asks for confirmation and then modifies the saved plant data to remove this plant.

`handleFertilizeButtonPress()`
Called when the user fertilises the plant. Adds XP, displays an animation and starts a cooldown before the plant can be fertilised again.

`getPlantImagePath()`
Returns the relative path to the image for the plant currently occupying this slot. Matches the plant's species and current stage of growth.

`handleMenuItemPress()`
Called when a FloatingMenu button is tapped. Determines which button was tapped based on its `id` and calls the appropriate function.

`handlePressInPlant()`
Displays the FloatingMenu when the plant is selected by the user.

`handlePressOutPlant()`
Hides the FloatingMenu when the plant is unselected by the user.

`handleRemoveFromArchive()`
Called when the user selects a plant from the collection to occupy this component's slot. Changes the appearance of the plant to match, and modifies the saved plant data, setting the plant's `archiveID` to `null` and its `plantPositionID` to the ID for this slot.

`handleSelectFromArchive()`
Called from SelectPlantModal when the user selects the option to add a plant from the collection. Hides the SelectPlantModal and shows the SelectFromArchiveModal.

`handleSelectPlant()`
Called when the user selects a new plant to put in this component's slot. Changes the appearance of the plant component to match, and adds the new plant to the saved plant data.

`handleToggleRealLifeScreen()`

Function for testing RealLifeScreen. Should not be used for other purposes.

`menuItemsList`
Passed to the FloatingMenu child component. Includes instructions for how each button should be positioned, the icon it should display, and the `id` which is used by `handleMenuItemPress()` to determine which function should be called when the button is tapped.

`handleWaterButtonPress()`
Function called when the user waters the plant. Calls `setWaterButtonPressed` and `startCountdown`, and displays an animation.

`startCountdown()`
Function that resets the timer when the plant is watered.

Dependencies

React:
`React`, `useContext`, `useEffect`, `useState`

React Native:
`Alert`, `Animated`, `Image`, `TouchableOpacity`, `View`

Libraries:
`@react-native-async-storage/async-storage`
`@react-navigation/native`
`react-native-responsive-fontsize`
`react-native-touchable-scale`
`react-native-vector-icons/FontAwesome`

Modules:
FloatingMenu
GameText.js
getPlantHitbox
plantDataContext
PlantStyles.js
playerConfigContext
RealLifeScreen
ScaleAnimation
SelectFromArchiveModal
SelectPlantModal
speciesProgressContext

Assets:
archive_icon.png
close_icon.png
delete_icon.png
fertilize_icon.png
icon_container.png

learn_icon.png
link_icon.png
water_icon.png

Usage

CollectionScreen
TwoDimSpace

5.3.12 RealLifeScreen

Source file

`PlantLinkComponent.js`

Behaviour

If `realLifeScreenVisible` property is set to `true`, the component will search the plant details (species name, water timer, care instructions)for that type of plant. After all the details have been loaded, the link screen will be shown to the user.

The details that are being fetched from `plantConfig` are `name`, `timer`, `careInstructions` and `stageAdvice`.

The user could perform two actions in this component. The first one is to add a photo. `buttonContent` is by default set to `null`, if `buttonContent` is `null` a text of "add photo" will be displayed on the button. If `buttonContent` is not null then, the photo saved in `buttonContent` then add photo button will show the picture instead.

The second action that the user could do is to edit the details of their plant. The nickname that the user entered for their plant would be stored in `nickname`, stage of their plant in real life would be stored in `Stage` and the last time they watered their plant will be stored in `watered`.

The user could save the details entered and the countdown timer would be started once the user presses the "save" button. The `countdown` variable is calculated by `timer` minus `wartered`. The `countdown` variable would be in seconds which would be formatted by `formatCountdownTime`.

Properties

`realLifeScreenVisible`
`true` if the component is opened, `false` if it is being archived

`plantID`
A property to identify the species of the plant to fetch details from `plantDataContext.js`

`countdown`
Stores the countdown time. This property is set up to make `countdown` accessible in `GameScreen.js` and this component

`watered`
Stores last time watered. This property is set up to make `watered` accessible in `GameScreen.js` and this component

`linked`

`false` by default set to `true` if the user has entered details and saved the details of the plant

`buttonContent`
Stores the picture of the plant. This property is set up to make `buttonContent` accessible in `GameScreen.js` and this component

Internal variables and functions

`plantConfig`
Stored the data fetched from `plantDataContext.js`

`nickname`
Stores the nickname of the plant

`Stage`
Stores the current stage of the plant

Dependencies

Reactive Native:
`View`, `StyleSheet`, `Image`, `Modal`, `TextInput`, `TouchableOpacity`, `Dimensions`, `ImageBackground`

Libraries:
`expo-image-picker`
`react-native-touchable-scale`
States:
`plantsDataContext.js`

Usage

`plantComponent.js`

5.3.13 ScaleAnimation

Source file

`ScaleAnimation.js`

Behaviour

This component renders the touch animation of the plant on the main screen.

Properties

`children`
Is the object that the animation is applied to

`isActive`
`true` if the animation is in progress, `false` when the animation ends

Internal variables and functions

`scaleIn`
A function that defines the animation when the plant is tapped, which enlarges slightly

`scaleOut`
A function that defines when the user moves his finger away from the plant, it returns to the normal size

Dependencies

Reactive Native:
`View`, `TouchableOpacity`, `Animated`

Usage

`plantComponent.js`

5.3.14 SelectFromArchiveModal

Source file

`ArchiveSelectionModal.js`

Behaviour

If `visible` is set to be `false` the menu will not be visible to the user. If the user taps on the "collection" button from `PlantSelectionModal.js` `visible` will be set to `true` and this menu will be shown with a modal slide animation. When `visible` is set to be `true` the component will fetch images of archived plants `plantDataContext.js`. The fetched images will be used as the icon of each plant that the user could add. `visible` will be set to `false` again if the user chooses a plant or taps outside of the menu. The menu will then collapse.

Properties

`visible`
`true` if the component is opened, `false` if it is being archived

`handleRemoveFromArchive`
A function passed as a property to handle instances where users choose a plant from their list of archived plants where changes are needed to be made in `plantDataContext.js`

Internal variables and functions

`setStage4ImagePath`
A function that fetches the image paths of all archived plants.

Dependencies

Reactive Native:
`View`, `Image`, `Modal`, `ScrollView`

Libraries:
`react-native-touchable-scale`
States:
`plantsDataContext.js`
Styles:
`PlantStyles.js`

Usage

`PlantSelectionModal.js`
`plantComponent.js`

5.3.15 SelectPlantModal

Source file

`PlantSelectionModal.js`

Behaviour

If `visible` is set to be `false` the menu will not be visible to the user. If the user taps on the "+" button from `plantComponent.js` `visible` will be set to `true` and this menu will be shown with a modal slide animation. When `visible` is set to be `true` the component will fetch images of all available plants `plantDataContext.js`. The fetched images will be used as the icon of each plant that the user could add. `visible` will be set to `false` again if the user chooses a plant, taps on the "collection" or taps outside of the menu. The menu will then collapse.

Properties

`visible`
`true` if the component is opened, `false` if it is being archived

`handleRemoveFromArchive`
The main purpose of setting this property up is to make it accessible in `ArchiveSelectionModal.js`

Internal variables and functions

`setStage4ImagePath`
A function that fetches the image paths of all available plants in the game.

Dependencies

Reactive Native:
`View`, `Image`, `Modal`, `ScrollView`

States:
`plantsDataContext.js`
Styles:
`PlantStyles.js`

Usage

`plantComponent.js`

5.3.16 TwoDimSpace

Source file

`2DSpaceComponent.js`

Behaviour

This component contains the logic for rendering different backgrounds when users unlock multiple backgrounds. It is also responsible for mapping the plant to the correct plant position using `plantPositions` and `roomID`.

Properties

`plantPositions`
This property contains the positions of slots where the user could place a plant

`roomID`
This property contains the identifier of the current room that the user is at

Internal variables and functions

`plantsForCurrentRoom`
This function maps the plant saved in `plantsDataContext.js` to the current room using `roomID`

Dependencies

Reactive Native:
`StyleSheet`, `ImageBackground`

States:
`plantsDataContext.js`

Usage

`GameScreen.js`

5.3.17 QuizQuitButton

Source file

`QuizQuitButton.js`

Behaviour

This component renders the quit button of the quiz screen.

Properties

N/A

Internal variables and functions

`handleBackPress`
This is a function that will be called when the button is being pressed. It will render a confirmation screen and ask if the user wants to return to the main screen.

Dependencies

Reactive Native:
`StyleSheet`, `Image`, `Dimensions`, `Alert`

Libraries:
`react-native-touchable-scale`
`@react-navigation/native`

Usage

`HomeStackNavigator.js`

5.3.18 XPBar

Source file

`XPBarComponent.js`

Behaviour

This component renders the XP bar on the main screen. It fetches the current XP of the user from `playerConfigContext.js` and uses it to show the correct image of XP bar. The component will fetch the `requiredXP` of the current level from `levelUpConfig.js`. `animateLevelUp` is set to be `false` by default, if `animateLevelUp` is true then the component will render the level up animation of the XP bar.

Properties

`animateLevelUp`
This property will only be `true` if the player XP in `playerConfigContext.js` exceeds the `requiredXP`

Internal variables and functions

`getCurrentLevelXP`
Is a function that fetches the XP of the current level

`getNextLevelXP`
Is a function that fetches the XP needed for the next level

`xpPercentage`
Is a variable that stores the progress of the current level

Dependencies

Reactive Native:
`View`, `Image`, `StyleSheet`, `Animated`

Libraries:
`react-native-touchable-scale`
States:
`playerConfigContext.js`

Usage

`GameScreen.js`

5.4 Navigation

Navigation source files are located in `src/navigation`

Each navigation source file exports either a `NavigationContainer` component (see https://reactnavigation.org/docs/navigation-container/) or `Stack.Navigator` component (see https://reactnavigation.org/docs/stack-navigator/).

Under normal circumstances, a React app only requires one `NavigationContainer` component and one `Stack.Navigator` component. If for any reason it becomes necessary to add more, the documentation for these new files should be structured to match the existing files documented here. At a minimum, the documentation for each file should contain the sections below:

Source file

The file which exports the navigation component.

For `Stack.Navigator` components, the source file should follow the naming convention `...StackNavigator.js`

It would be highly inadvisable to create a new `NavigationContainer`, but if this were to be done, the new source file should follow the naming convention `...NavigationContainer.js`

Component

The type of navigation component: either a `Stack.Navigator` component or a `NavigationContainer` component.

Internal variables and functions

Variables and functions defined by the source code which creates the navigation component (not including imported variables and functions, even if they are assigned to a new variable). These are not exported and cannot be accessed by other modules.

Dependencies

The list of libraries and modules used by the navigation component.

Usage

The list of modules which use the navigation component.

5.4.1 HomeStackNavigator

This is the main (and currently the only) navigator. It navigates between all screens which can be reached by the user.

Source file

`stack/HomeStackNavigator.js`

Component

`Stack.Navigator` (see https://reactnavigation.org/docs/stack-navigator/)

Screens

This navigator can navigate to the following screens:

- CollectionScreen
- GameScreen
- GameStatsScreen
- LevelSelectionScreen
- QuizScreen
- SettingsScreen
- ShopScreen

Internal variables and functions

`CustomBackButton`
A `TouchableScale` component (see https://www.npmjs.com/package/react-native-touchable-scale) using the back_icon.png image asset. This component is intended to replace the default 'Back' button on most screens.

`styles`
A `StyleSheet` (see https://reactnative.dev/docs/stylesheet) for the `CustomBack Button` component.

Dependencies

React:
`React`

React Native:
`Dimensions`, `Image`, `StyleSheet`

React Navigation / Native:

`useNavigation`

Libraries:
`react-native-touchable-scale`

Modules:
CollectionScreen
GameScreen
GameStatsScreen
LevelSelectionScreen
QuizQuitButton
QuizScreen
SettingsScreen
ShopScreen
Assets:
back_icon.png

Usage

App

5.4.2 NavigationContainer

Used to change the behaviour of the navigation stack.

Source file

`NavigationContainer.js`

Component

`NavigationContainer` (see
https://reactnavigation.org/docs/navigation-container/)

Internal variables and functions

None

Dependencies

React Navigation / Native:
`NavigationContainer`

Usage

None

5.5 Screens

Screen component source files are located in `src/screens`

Each screen component source file exports exactly one React `Component`. Screen components have been categorised separately because they should only be used as a screen, whereas Components can be used for a variety of purposes.

The documentation for each screen component is structured as follows:

Source file

The file which exports the screen component. New screen component files should be named according to the convention `...Screen.js`

Navigator

The navigator or app function which defines the screen.

Label

The label which is given to the screen by the navigator.

Behaviour

An overview of the screen, including its constituent components and the information it uses.

Properties

The properties which are passed to the screen component by the navigator or the main app function. For more information, see https://reactnative.dev/docs/navigation

Note that while `navigation` and `route` are automatically passed to all screen components, they should only be included in the documentation if they are actually used by that screen component.

Internal variables and functions

These are the variables and functions defined in the source code which generates the screen component. This does not include variables and functions imported from other modules. Note that these are not exported and cannot be accessed/called on the screen component.

Dependencies

The list of libraries and modules used by the screen component.

Usage

The list of modules which navigate to the screen.

5.5.1 CollectionScreen

This screen displays all the plants which have been archived by the user, and allows the user to interact with them (including an option to permanently delete a plant).

Source file

`CollectionScreen.js`

Navigator

HomeStackNavigator

Label

`CollectionScreen`

Behaviour

The screen is comprised of:

- An `ImageBackground` component which displays the standard menu background
- A `View` component which contains the title: `Collection`
- A `PagerView` component which contains a Plant component for each archived plant in plantDataContext

Properties

None

Internal variables and functions

`archivedPlants`
Array of all saved plants from plantDataContext which have a non-`null` `archiveID`, i.e. plants which have been archived by the user.

`styles`
`StyleSheet` containing styling for the screen's components.

Dependencies

React:
`React`

React Native:
`Dimensions`, `ImageBackground`, `StyleSheet`, `View`

Libraries:
`react-native-pager-view`

Modules:
GameText.js
misc/menu_bg.png
Plant
plantDataContext

Usage

CollectionButton

5.5.2 GameScreen

This screen displays the main screen of the game. The user initially access to this screen when opens the game. The screen also contains placeholders for the user to interact with in-game plants.

Source file

`GameScreen.js`

Navigator

HomeStackNavigator

Label

`GameScreen`

Behaviour

The screen is comprised of:

- `menuVisible` and `MenuComponent` to toggle the visibility of the in-game menu. When the user presses on the menu button and is detected by the `menuVisible`, the menu shows up. The menu will be hidden when the `menuVisible` detects the close button is pressed.
- `AsyncStorage`, `levelUpModalVisible` and `shouldShowLevelUpModal` together toggle the visibility of the level-up modal by retrieving level data in an asynchronous storage. When the user gains a certain amount of XP and is retrieved by `levelUpModalVisible`, `shouldShowLevelUpModal` will be triggered and display the level-up animation on the screen.
- `setupPlayer, playRandomBackgroundMusic, pauseBackgroundMusic, stopBackgroundMusic` components for playing or stopping background when mounted or unmounted. Random background music starts playing automatically when the game is opened. The user can turn off the background music in SettingsScreen to unmount the components.
- A `Swiper` component to display different rooms by swiping. When the user reaches certain levels and unlocks new rooms, the user can visit new rooms by swiping the game screen.
- A `Header` component to render various touchable icons (settings, hearts, collection, game stats, coin).
- An `XPBar` to display the game XP and level. The XP bar contains the user's game level number and a progress bar determines the XP progress to reach the next level.

- A `HeartsDisplay` to display the user's current game Hearts configuration and the timer which shows the time for recovering the next free game Hearts.
- A `CoinDisplay` button to display the coin configuration and access to ShopScreen when the button is pressed.
- A `CollectionButton` button to access to CollectionScreen when the button is pressed
- A `GameStatsButton` button to access to GameStatsScreen when the button is pressed

Properties

None

Internal variables and functions

`menuVisible`
State variable for tracking the visibility of the in-game menu.

`levelUpModalVisible`
State variable for tracking the visibility of the level-up modal.

`shouldShowLevelUpModal`
State variable for indicating when to show the level-up modal.

`levelUpModalVisible`
State variable to track the visibility of the level-up modal.

`navigation`
The hook represents the navigation object.

`unlockedRooms`
Result of the `getUnlockedRooms` function from the usePlayerConfig hook, representing the list of unlocked rooms for the current player level.

`checkLevelUp`
An asynchronous function to check if the player has leveled up.

`handleToggleMenu`
The function to toggling the visibility of the in-game menu.

`useFocusEffect`
A hook for performing an effect when the screen is focused. It calls the `checkLevelUp` with a dependency on the level variable whenever the screen is focused.

`useEffect`

A hook for setting up and cleaning up background music. It triggers random music to play in the background when the component mounts. The cleanup function stops the background music when the component is unmounted.

Dependencies

React:
`React`

React Native:
`Modal`, `Image`, `TouchableOpacity`, `View`

Libraries:
`react-native-vector-icons/FontAwesome`
`react-native-touchable-scale`
`@react-native-async-storage/async-storage`
`react-native-swiper`
`@react-navigation/native`
Modules:
CoinDisplay
XPBar
Header
Oracle
HeartsDisplay
CollectionButton
GameStatsButton
FloatingMenu
TwoDimSpace
backgroundsConfig
GameScreenStyles.js

Usage

None

5.5.3 GameStatsScreen

This screen allows the user to manage plant data and display different plants' mastery levels.

Source file

`GameStatsScreen.js`

Navigator

HomeStackNavigator

Label

`GameStatsScreen`

Behaviour

The screen is comprised of:

- `usePlantContext` and `useProgressContext` components to fetch plant configuration data `plantsConfig` and species progress data `speciesProgress`, and display detailed information about a selected plant
- `plantDetailVisible` state and `close_plantdetails` function to control the visibility of the selected plant
- A `renderMasteryItem` function to display mastery levels of different plants with `FlatList`
- A progress bar represents the mastery level progress of each plant, with the width of the bar reflecting the percentage completion with `item.progress`
- A `masteryLevels` state component updates plant configuration and species progress, triggering a re-render

Properties

None

Internal variables and functions

`menuVisible`
State variable for tracking the visibility of the in-game menu.

`plantsConfig`
State variable containing configuration data for different plant species, obtained from plant context.

`speciesProgress`

State variable containing progress data for different plant species, obtained from progress context.

`plantDetailVisible`
State variable for indicating when to show the plant detail modal.

`selectedPlant`
State variable storing information about the currently selected plant.

`handleSelectPlant(plantID)`
The function for setting `selectedPlant` state according to plant ID.

`disableSelectPlant()`
The function for setting `selectedPlant` state to NULL.

`getPlantImagePath()`
The function for retrieving the image path for the selected plant based on its growth stage and progress.

`getProperty(propertyName)`
The function retrieving a property value from `selectedPlant` state based on the provided property name.

`getPlantImagePath()`
The function for toggling the visibility of the plant details modal.

`close_plantdetails()`
The function for closing the plant details modal.

Dependencies

React:
`React`

React Native:
`View`, `Text`, `StyleSheet`, `Modal`, `FlatList`, `TouchableOpacity`, `Image`, `ImageBackground`

Libraries:
`react-native-touchable-scale`
Modules:
GameText.js
Styles
misc/menu_bg.png
plantDataContext
speciesProgressContext

Usage

GameStatsButton

5.5.4 LevelSelectionScreen

This screen displays the level selection of the game. The user accesses this screen when they click on the quiz button in the `floatingMenu`. The screen contains levels of the selected plant either in unlocked or locked form.

Source file

`LevelSelectScreen.js`

Navigator

HomeStackNavigator

Label

`LevelSelectionScreen`

Behaviour

The screen is comprised of:

- An `ImageBackground` component that displays a background image.
- A `View` component, a core component used for grouping and styling other components.
- A `PagerView` component containing a `Plant` component for each archived plant in `usePlantContext`.
- A `Text` component, used for displaying text.
- A `StyleSheet` component, used for creating and managing styles.
- An `Alert` component, for displaying alert dialogs.
- An `Image` component, for displaying images.
- A `Dimensions` component, used for obtaining the device screen dimensions.
- An `Animated` module, for creating animated components.
- A `ViewPager` component, for implementing a swipeable view with pages.
- A `TouchableScale` component, a custom touchable component that scales on press.
- `RFPercentage, RFValue`: Utility functions for responsive font sizing.
- A `GameText` component, a custom text component with specific styles.
- `LevelsConfig`: A module or state that contains configuration data for game levels.
- A `HeartsDisplay` component, a custom component for displaying hearts.

- A `CoinDisplay` component, a custom component for displaying coins.
- `usePlayerConfig`: A custom hook for accessing player configuration data.
- `CompletedLevelsContext, useCompletedLevelsContext`: Context and custom hook for managing completed levels.
- An `Oracle` component, possibly representing an oracle character.

Properties

None

Internal variables and functions

`heartsAndCoinsTop`
A variable to determine the top position of hearts and coins based on the aspect ratio.

`backgroundImage`
A variable holding the path to the background image.

`upIcon, downIcon, levelCompleteIcon, levelLockedIcon, levelUnlocekdIcon`
Variables holding paths to various icons used in the component.

`deviceWidth` and `deviceHeight`
Variables holding the dimensions of the device screen.

`textSizeBig` and `textSizeSmall`
Variables determining the font sizes based on the device height.

`aspectRatio`
A variable calculated based on the device height and width.

`scrollIndicatorTopOpacity` and `scrollIndicatorBottomOpacity`
Animated values representing the opacity of scroll indicators.

`viewPagerRef`
A reference to the ViewPager component.

`swipeTextOpacity`
A state variable to control the opacity of the swipe text.

`startBlinking`
A function to start a blinking animation for the swipe text.

`fadeOutScrollIndicators`
A function to animate the fading out of scroll indicators.

`resetScrollIndicators`
A function to reset the opacity of scroll indicators.

`handlePageScroll`
A function handling the scrolling of pages in the ViewPager.

`renderPage`
A function to render a page for the ViewPager based on the level.

`handlePress`
A function handling the press event on a level container, triggering navigation to the quiz screen if conditions are met.

Dependencies

React:
`React`,`useState`,`useEffect`,`useRef`

React Native:
`Dimensions`, `ImageBackground`, `StyleSheet`,
`View`,`Text`,`Alert`,`Image`,`Animated`

Libraries:
`react-native-pager-view`,`react-native-touchable-scale`,`react-native-responsive-fontsize`

Modules:
GameText.js
levelsConfig
HeartsDisplay
CoinDisplay
playerConfigContext
completedLevelsContext

Usage

Plant

5.5.5 LoadingScreen

LoadingScreen enhances the user experience by providing a smooth transition as the game loads various data and assets.

Source file

`LoadingScreen.js`

Navigator

App

Label

`LoadingScreen`

Behaviour

- A `View` component as the main container.
- An `Animated.View` to handle the fade-in and fade-out animation.
- An `Image` component to display a loading screen image.

Properties

None

Internal variables and functions

`loadingMessage`
State variable to display the current loading status.
`fadeAnim`
State variable for managing fade animation.
`loadPlantData`, `loadFonts`, `loadPlayerState`, `loadCompletedLevels`, `loadSpeciesProgress`, `loadPlantsConfig`
Functions for loading various data

Dependencies

React:
`useState`, `useEffect`, `useContext`
React Native:
`View`, `Dimensions`, `Text`, `Image`, `Animated`, `AsuncStorage`
Modules:
plantDataContext
playerConfigContext
completedLevelsContext

speciesProgressContext
plantsConfig

Usage

None

5.5.6 QuizScreen

The QuizScreen component is responsible for managing the quiz interface in the game.

Source file

`QuizScreen.js`

Navigator

HomeStackNavigator

Label

`QuizScreen`

Behaviour

- A `View` component used for grouping and styling other components.
- A `Modal` component used to create a modal. It is conditionally rendered based on various state variables (showModal, showInstructions, showEvolveModal, showCongratsBackground).
- A `GameText` custom component used for rendering text. It is styled with additional styles, adjusting padding.
- A `ScrollView` component used for scrolling content. It contains `GameText` displaying instructions retrieved from the currentInstructions variable.
- A `EvolveAnimation` ustom component displaying an animation based on current and next images of a plant. It uses images from the currentPlant object.
- A `ImageBackground` component for displaying a background image. It uses the quizBackground image and styles it to cover the entire screen.
- A `Oracle` custom component displayed as an image. It is positioned absolutely on the screen.
- A `TouchableScale` A touchable component positioned absolutely on the screen with a specified left and top.

Properties

None

Internal variables and functions

`showRewardMessage`
Boolean state to determine whether to show a reward message.
`currentQuestionIndex`
Integer state representing the index of the current question in the quiz.
`questions`
Array state to store the quiz questions.
`showModal`
Boolean state to control the visibility of the modal.
`showInstructions`
Boolean state to determine whether to show quiz instructions.
`feedbackMessage`
String state to store feedback messages.
`showGameOverModal`
Boolean state to control the visibility of the game over modal.
`updatedList`
State to store a list that may be updated during the quiz.
`currentInstructions`
String state to store the current quiz instructions.
`currentLevel`
String state representing the current level in the quiz.
`currentPlant`
Object state representing the current plant in the quiz.
`feedbackModalVisible`
Boolean state to control the visibility of the feedback modal.
`showCongratsBackground`
Boolean state to control the visibility of the congratulations background.
`isLevelCompleted`
Boolean state to indicate whether the current level is completed.
`showEvolveModal`
Boolean state to control the visibility of the evolve modal.
`storedGrowthStage`
State to store the initial growth stage of the plant.
`levelUpGrowthStage`
State to store the growth stage at which the level is upgraded.
`correctAnswersCount`
Integer state to track the number of correct answers.
`incorrectAnswersCount`
Integer state to track the number of incorrect answers.
`bonusCoins`
Integer state to store the bonus coins earned during the quiz.
`getGrowthIndex`
Function to calculate the growth index based on the growth stage.
`masteryLevel`
Variable storing the mastery level calculated based on the growth stage.

`handleStartQuiz`
Function to handle the start of the quiz by hiding instructions.
`arrangeData`
Function to arrange data for the progress of a plant.
`handleAnswer`
Function to handle user answers and update states accordingly.
`decreasePlayerHearts`
Function to decrease player hearts and handle game over conditions.
`getCurrentGrowthStage`
Function to get the current growth stage of a plant based on its progress.
`getPreviousGrowthStage`
Function to get the previous growth stage of a plant.
`handleGoHome`
Function to handle navigation back to the home screen.
`completeQuiz`
Function to handle the completion of the quiz, update states, and display rewards.
`updatePlantsProgress`
Function to update the progress of a plant and return arranged data.
`renderItem`
Function to render individual answer items in the quiz.
`renderFeedbackModal`
Function to render the feedback modal.
`renderGameOverModal`
Function to render the game over modal.

Dependencies

React:
`React`, `useState`, `useEffect`, `useContext`
React Native:
`View`, `Dimensions`, `Modal`, `Image`, `Animated`,
`AsyncStorage, \verb`ImageBackground— , `ScrollView`, `FlatList`
Libraries:
`react-native-responsive-fontsize,react-native-touchable-scale`
Modules:
GameText.js
plantsTriviaConfig
levelsConfig
plantDataContext
playerConfigContext
speciesProgressContext
completedLevelsContext
HeartsDisplay
CoinDisplay
EvolveAnimation

misc/quiz_screen.png icons/text_box.png

Usage

LevelSelectionScreen

5.5.7 SettingsScreen

SettingsScreen is a vital component of the game application, designed to offer players a customisable game experience by allowing them to adjust settings such as background music.

Source file

`SettingsScreen.js`

Navigator

HomeStackNavigator

Label

`SettingsScreen`

Behaviour

The SettingsScreen includes the following elements:

- A `View` component as the container.
- A `GameText` component to display the "Settings" title.
- React's `Switch` component to toggle the background music on and off.

Properties

None

Internal variables and functions

`musicEnabled`
State variable to track whether the music is enabled or not.
`toggleMusicSwitch`
Function to toggle the background music and update the state.

Dependencies

React:
`React`, `useState`
React Native:
`View`, `Switch`, `StyleSheet`
Libraries:
None
Modules:
None

Usage

None

5.5.8 ShopScreen

The ShopScreen is the screen for user to view different skins in shop and purchase skins and and Hearts.

Source file

`ShopScreen.js`

Navigator

HomeStackNavigator

Label

`ShopScreen`

Behaviour

The screen is comprised of:

- A `toggleShowOnlyUnowned` function toggles the state variable `showOnlyUnowned`, which controls the filter of showing only un-owned skins or all skins in the purchase menu. User can perform the switching by pressing the `troggleButton`, switching between `buttonNotOwned` and `buttonOwnedNotApplied`.
- A `renderGridItem` renders individual items in the `FlatList` to adjust the style and button functionality based on whether the skin is owned, applied, or neither.
- A `handlePlantPress` function sets the selected plant in the state, influencing the display of user's choice of plant skins in the `FlatList` when a plant button is pressed by the user.
- A `handleShowAll` function sets the selected plant to NULL, and displays all skins in the `FlatList`. The function is triggered when the "All" button is pressed. User can filter the plant skin menu by pressing specific plant-type buttons `plantButton`, or canceling the filtering by pressing the "All" button `allButton`
- A `handleBuyHearts` function calculates the total cost based on the number of hearts to buy and prompts the user to confirm the purchase through a modal when the user attempts to buy hearts with the Hearts purchasing button.
- A `handleSkinAction` function either initiates the purchase of a skin or applies an owned skin when a skin item is selected and pressed by the user.

- A `confirmPurchase` module for finalizing the purchase on either skin or hearts after the user confirms through the modal, then updates the player's coins, hearts, and plant configuration accordingly.

Properties

None

Internal variables and functions

`screenHeight`
State variable for the representation of the height of the device screen.

`screenWidth`
State variable for the representation of the width of the device screen.

`aspectRatio`
State variable for the calculations of the aspect ratio of the device screen.

`itemsTop, itemsPad, itemsRight, itemsGap, textTop, coinsBot`
State variables for dynamic adjust styling with aspect ratio.

`textSize`
Represents the font size based on the screen height.

`buttonFontSize`
Represents the font size for buttons based on the screen height.

`numColumns`
Number of columns in the FlatList.

`windowWidth`
Width of the device window.

`itemWidth`
Width of items in the FlatList.

`totalItemHorizontalMargin`
Total horizontal margin for all items.

`flatListWidth`
Width of the FlatList container.

`toggleShowOnlyUnowned`
The function Toggles the state variable `showOnlyUnowned`.

`renderGridItem`
The function renders individual items in the FlatList.

`handlePlantPress`
The function handles the press event for plant buttons.

`handleShowAll`

The function handles the press event for the "All" button.

`handleBuyHearts`
The function handles the press event for buying hearts.

`handleSkinAction`
The function handles the press event for skin purchases or applies.

`confirmPurchase`
The function for the purchase confirmation and the state updates accordingly.

`ConfirmationModal`
the function renders the confirmation modal for purchases.

`capitalizeFirstLetter`
The function for capitalizes the first letter of a string.

`formatData`
The function handles formatting data for the FlatList.

Dependencies

React:
`React`

React Native:
`View`, `Text`, `StyleSheet`, `FlatList`,`Image`, `ScrollView`, `ImageBackground`, `Dimensions`,`Modal`

Libraries:
`react-native-responsive-fontsize`
`react-native-touchable-scale`
Modules:
playerConfigContext
GameText.js
misc/menu_bg.png
CoinDisplay
plantDataContext

Usage

CoinDisplay

5.6 States

State source files are located in `src/states`

States are used for information which is shared between different modules across the app.

The documentation for each state is structured as follows:

Source file

The file which exports the state. There is no strict naming convention for state files, but the name should reflect the state's object type and intended usage.

Object type

The state's object type. This could be a native JavaScript object such as an `Array` or an `Object`, or it could be an object from the React library.

Internal variables and functions

These are the variables and functions defined in the source code which generates the state. This does not include variables and functions imported from other modules. These are not exported and cannot be accessed/called by other modules.

Properties and methods

These can be accessed by other modules which use the state.

Dependencies

The list of libraries and modules used by the state.

Usage

The list of modules which use the state.

5.6.1 backgroundsConfig

Contains config data for each different room:

- `id`
  `number`, a unique identifier for the room.
- `name`
  `String`, the name of the background.
- `image`
  `Image` (see https:reactnative.dev/docs/image), the background image.
- `levelRequired`
  `number`, the minimum level which the user must reach before they can switch to the room.
- `plantPositions`
  `Array`, the positions in the room where plants can be placed. Each position is an `Object` with the following properties: `id`, a unique identifier; `bottom`, the offset from the bottom of the screen, expressed as a percentage of the screen's height; `left`, the offset from the left side of the screen, expressed as a percentage of the screen's width.

Source file

`backgroundsConfig.js`

Object type

`Object`

Internal variables and functions

None

Properties and methods

The config data for each room is stored as a property, following the naming convention `background1`, `background2`, ...

Dependencies

React:
`React`
React Native:
`AsyncStorage`
Modules:
plantsConfig
plantDataContext

Usage

GameScreen playerConfigContext

5.6.2 completedLevelsContext

Manages and provides the context for completed levels of different plant species in the game.

Source file

`completedLevelsContext.js`

Object type

`Object`

Internal variables and functions

`updateCompletedLevels`
Function updates the completed level for a specific plant species.

Properties and methods

`completedLevels`
Property stores the completed levels for each plant species.
`setCompletedLevels`
Function updates the state of completed levels.
`updateCompletedLevels`
Function to update the completed level for a specific plant species.

Dependencies

React:
`createContext`, `useState`, `useContext`
React Native:
`AsyncStorage`
Modules:
plantsConfig

Usage

GameScreen
GameStatsScreen

5.6.3 levelsConfig

Defines the configuration for the levels in the game, including the total number of levels, experience points (XP) rewards, and tracking of completed levels.

Source file

`levelsConfig.js`

Object type

`Object`

Internal variables and functions

`levelsConfig`
Variable containing the configuration for each game level.

Properties and methods

None

Dependencies

None

Usage

LevelSelectionScreen
QuizScreen

5.6.4 levelUpConfig

Specifies the experience points (XP) required for each level, serving as a key configuration for the game's leveling system.

Source file

`levelUpConfig.js`

Object type

`Object`

Internal variables and functions

`requiredXP`
Object where each key represents a game level, and each value is an object containing the `xpRequired` for that level.

Properties and methods

Each key (level number) contains an object with a single property `xpRequired`, indicating the amount of XP needed to reach that level.

Dependencies

None

Usage

playerConfigContext
XPBar

5.6.5 plantsConfig

Defines detailed configurations for each plant type in the game, including properties like name, care instructions, growth stages, and challenges.

Source file

`plantsConfig.js`

Object type

`Object`

Internal variables and functions

`plants`
Object containing individual plant configurations.

Properties and methods

Each key (plant ID) represents a unique plant, with its properties such as `plantID`, `name`, `iconPath`, `type`, `colours`, `careInstructions`, `challenges`, `skins`, etc.

Dependencies

None

Usage

LoadingScreen
completedLevelsContext
plantDataContext
playerConfigContext
speciesProgressContext

5.6.6 plantDataContext

Manages the state and configuration of plant data within the game, including storing and updating individual plant instances and the overall plant configuration.

Source file

`plantsDataContext.js`

Object type

`Object`

Internal variables and functions

`plantData`
Array to store the state of individual plant instances.
`plantsConfig`
Object to store the configuration of the plants, initialized with `defaultPlantsConfig`.
`updatePlantData`
Function updates the state of individual plant instances and saves it to AsyncStorage.
`savePlantsConfig`
Function saves the updated plants configuration to AsyncStorage and updates state.
`updatePlantsConfig`
Updates the state of plants configuration.

Properties and methods

`plantData`
Contains the current state of all plant instances.
`plantsConfig`
Contains the current configuration of all plants.
`updatePlantData`
Updates and persists plant data.
`savePlantsConfig`
Saves and updates the plants configuration.
`updatePlantsConfig`
Updates the plants configuration state.

Dependencies

React:
`React`
React Native:

`AsyncStorage`
Modules:
plantsConfig

Usage

LoadingScreen
ShopScreen

5.6.7 plantsTriviaConfig

Contains trivia questions and answers related to various plants, structured in levels to test and enhance player knowledge about plant care and characteristics.

Source file

`plantsTriviaConfig.js`

Object type

`Object`

Internal variables and functions

`plantsTriviaConfig`
Object where each key is a plant ID and each value is an object representing different levels of trivia for that plant. Each level contains instructions, a coin reward, and an array of questions with multiple-choice answers.

Properties and methods

Each plant ID contains levels (`level1`, `level2`, etc.), with each level having `instructions`, `coin` (reward), and `questions`. Each question in `questions` array includes `question` text and `answers`, an array of answer objects with `text` and `isCorrect` properties.

Dependencies

None

Usage

QuizScreen

5.6.8 playerConfigContext

Manages the player's state in the game, including hearts, experience points (XP), level, coins, and other player-related data.

Source file

`playerConfigContext.js`

Object type

`Object`

Internal variables and functions

`HEART_INCREASE_INTERVAL`, `MAX_HEARTS`
Constants for heart management.
`defaultPlantProgress`, `defaultPlayerState`
Functions to initialize default states for player progress and overall player state.
`updatePlayerState`
Function to update the player's state and save it to AsyncStorage.
`decreaseHearts`, `addXP`, `addCoins`
Functions to modify specific aspects of the player's state (hearts, experience points, and coins).
`getUnlockedRooms`
Function to determine unlocked backgrounds based on player level.
`resetPlayerConfig`
Resets player configuration to default.

Properties and methods

Properties:
Player state variables like `hearts`, `xp`, `level`, `coins`, `unlockedBackgrounds`, `plantProgress`, `timer`, and `lastUpdated`.
Methods:
`updatePlayerConfig`
Updates player configuration.
`decreaseHearts`
Decreases the number of hearts.
`addXP`
Adds experience points.
`addCoins`
Adds coins to the player's state.
`getUnlockedRooms`
Retrieves unlocked rooms based on the player's level.
`resetPlayerConfig`

Resets the player's state to its default values.

Dependencies

React:
`React`
React Native:
`AsyncStorage`
Modules:
levelUpConfig
backgroundsConfig
plantsConfig

Usage

LoadingScreen

5.6.9 speciesProgressContext

Manages the progress for different species in the app, keeping track of individual species' progress and updating it as necessary.

Source file

`speciesProgressContext.js`

Object type

`Object`

Internal variables and functions

`defaultSpeciesProgress`
Object initialized based on `plantsConfig`, setting initial progress for each species to zero.
`updateSpeciesProgress`
Function to update the progress of a specific species and save it to AsyncStorage.

Properties and methods

`speciesProgress`
Object representing the progress of each species.
`updateSpeciesProgress`
Method used to update the progress for a particular species.

Dependencies

React:
`React`
React-Native:
`AsyncStorage`
Modules:
plantsConfig
plantDataContext

Usage

App
LoadingScreen

5.7 Styles

Style source files are located in `src/styles`

Each style source file exports either a `StyleSheet` (see https://reactnative.dev/docs/stylesheet) or a pre-styled `Component`.

The documentation for each style source file is structured as follows:

Type

The type of object exported by the file.

Dependencies

The list of libraries and modules used by the style source file.

Usage

The list of modules which use the style source file.

5.7.1 CollectionStyles.js

Styling intended for components on the CollectionScreen. Currently unused.

Type

`StyleSheet`

Dependencies

React Native:
`StyleSheet`

Usage

None

5.7.2 GameScreenStyles.js

Styling for components on the GameScreen.

Type

`StyleSheet`

Dependencies

React Native:
`Dimensions`, `StyleSheet`

Libraries:
`react-native-responsive-fontsize`

Usage

GameScreen

5.7.3 GameText.js

Text which displays in a gamified style.

Type

`Text` component

Dependencies

React:
`React`

React Native:
`StyleSheet`, `Text`

Usage

Components:
CoinDisplay
CollectionButton
GameStatsButton
HeartsDisplay
MenuComponent
Plant
RealLifeScreen
SelectFromArchiveModal
SelectPlantModal
XPBar

Screens:
CollectionScreen
GameScreen
GameStatsScreen
LevelSelectionScreen
QuizScreen
SettingsScreen
ShopScreen

5.7.4 PlantStyles.js

Styling for plant images.

Type

`StyleSheet`

Dependencies

React Native:
`StyleSheet`

Usage

Plant
SelectFromArchiveModal
SelectPlantModal

5.8 Utilities

Utility files are located in `src/utilities`

Utilities are objects or functions which have been separated from the modules which use them, for the sake of readability and reusability. Each utility file should export exactly one function or object - if it is necessary to export more than one, they should be bundled together as properties/methods of a single object.

Utilities are documented file-by-file. The documentation for each utility file is structured as follows:

Type

The type of thing that the utility file exports: either an object or a function. If it is an object, its properties and methods should be listed. If it is a function, the type of data it returns should be described, and its parameters should be listed.

Internal variables and functions

These are the variables and functions defined in the utility file (not including variables and functions imported from other modules, even if they are assigned to a new variable). These are not exported and cannot be accessed by other modules.

Dependencies

The list of libraries and modules used by the utility file.

Usage

The list of modules which use the utility file.

5.8.1 backgroundMusic.js

Utility to control the background music, including playing, stopping and starting.

Type

Object with the following methods:

- `pauseBackgroundMusic()`
  Pauses the music.
- `playRandomBackgroundMusic()`
  Chooses a random song which is different from the previous song played. Loads the song to the audio player and sets the volume to 0.2. Sets an event listener to call this method again when the song finishes.
- `setupPlayer()`
  Configures the audio player.
- `stopBackgroundMusic()`
  Unloads the current song from the audio player.

Internal variables and functions

`lastPlayedIndex`
Mutable variable that stores the index of the last played song

Dependencies

Libraries:
`expo-av`

Assets:
music/8-bit-arcade.mp3
music/feed-the-machine-classic-arcade-game.mp3
music/short-circuits-classic-arcade-game.mp3

Usage

GameScreen
SettingsScreen

5.8.2 menuUtilities.js

Contains utilities relating to the main menu.

Type

Currently only contains the `toggleMenu` method.

Internal variables and functions

N/A.

Dependencies

Variable `currentVisibility` is passed.

Usage

GameScreen

6 Structure

The app's structure centers on `App.js` for initialization, with `HomeStackNavigator` for screen navigation, modular components for UI, and context providers for state management, enriched by assets, styles, and utilities.

How to update

Program flow represents, broadly, how the code executes from start to finish. This is a high-level abstraction and does not need to be updated unless major changes are made to the source code, such as adding a new screen or navigator.
Module interaction is a diagram representing the complete circular interactions between each internal module in the application. It is created using the `dependency-cruiser` dev dependency. To update it, you can run the following code in your terminal on **one line**:

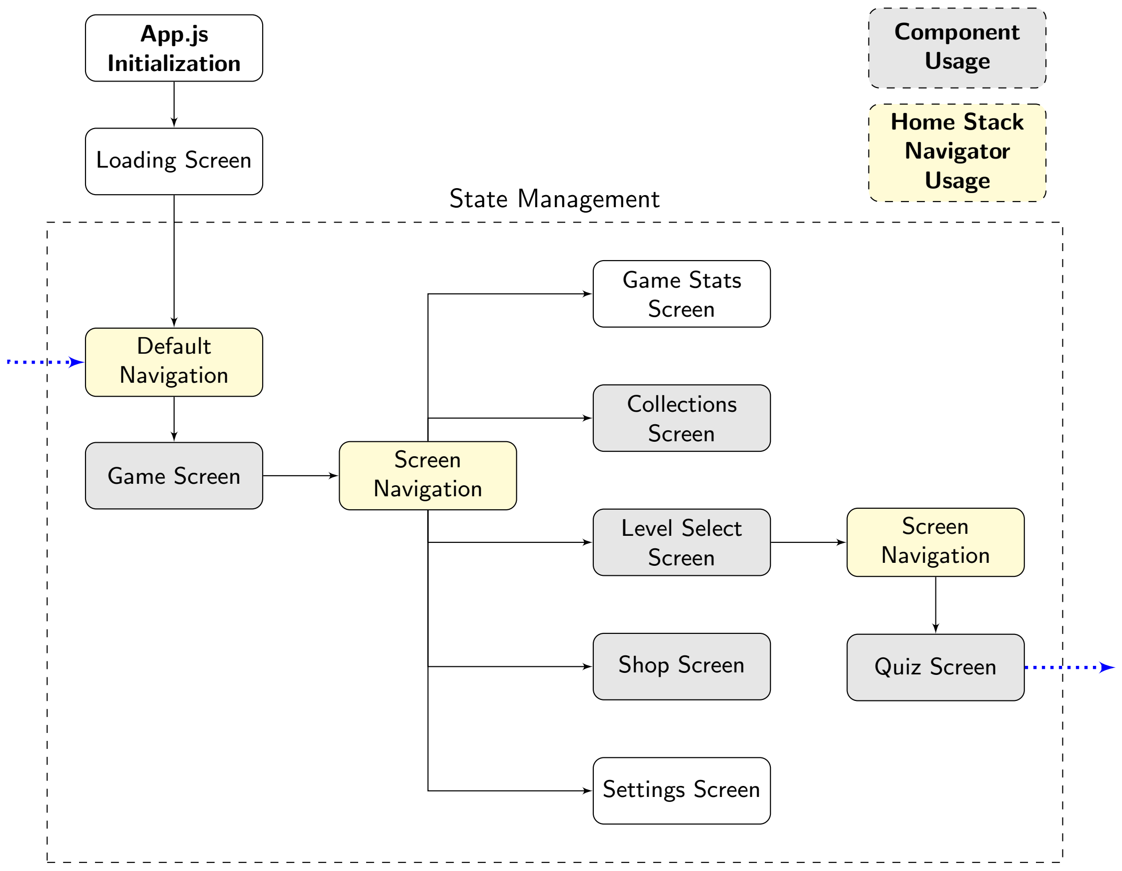
```
npx depcruise src --include-only "^src" --exclude "src/(assets|styles)"
--output-type dot | dot -T png > dependency-graph.png
```

Library interaction is a diagram representing the interactions between each internal components and external modules used within the game. It is created using the `dependency-cruiser` dev dependency. To update it, you can run the following code in your terminal on **one line**:

```
npx depcruise src node_modules
--exclude "src/(assets|styles|navigation|states|utilities)"
--output-type dot | dot -Tpng -o dependency-graph2.png
```

The new dependency graphs will be placed into the root file.

6.1 Program flow



***See Chapter 5 for a breakdown of `Screen -> Component` interactions.**

6.2 Module interaction

A full dependency diagram detailing the interactions between modules may be found on the GitHub repository: dependency-graph.png

6.3 Library interaction

A full dependency diagram detailing the interactions between components and libraries may be found on the GitHub repository: node-module-dependency-graph.png

7 Installation guide for developers

This chapter describes the process of downloading the source code to your computer and running a test build.

How to update

If the installation and test build process changes, these instructions must be updated to match.

If the app is tested (and confirmed to work with no major issues) on a new device, that device should be added to the list of tested devices. By the same token, if a device on the list begins to manifest major issues, it should be removed.

7.1 Requirements

Node.js

Node.js is used to install dependencies and create test builds. It can be downloaded from https://nodejs.org/en/download

Expo Go

Expo Go must be installed on your mobile device in order to access the test build. It is available from the following sources:

- iOS: https://apps.apple.com/us/app/expo-go/id982107779
- Google Play:
  https://play.google.com/store/apps/details?id=host.exp.exponent&referrer=www&pli=1
- GitHub: https://github.com/expo/expo

Please note that Bloom does not support web browsers. You must install Expo Go on a real or virtual mobile device.

The test build has been confirmed to run on the following devices:

- Apple iPhone 6s

- Apple iPhone 12
- Apple iPhone 13 Pro Max
- Apple iPhone 15 Pro Max
- Google Pixel 7
- Google Pixel 7 Pro
- Xiaomi Redmi 11 Prime

If you are using a device not listed above, and you encounter difficulties with the test build, we recommend using a Virtual device instead.

7.2 Optional tools

Git

We recommend using Git to download the source code. Git can be downloaded from https://git-scm.com/downloads

Virtual device

You may find it more convenient to access the test build through a virtual device, rather than using a physical one. Loading times can be substantially lower when using a virtual device on the same computer which is running the test build. However, be aware that the user experience may be very different on a physical device, and be sure to test the app on a physical device before release.

We recommend using the Android Emulator to create virtual devices. Follow these steps to set up an Android virtual device (AVD):

1. Install Android Studio: https://developer.android.com/studio
2. Create an AVD:
   https://developer.android.com/studio/run/managing-avds
   Note that at the **Select hardware** stage, you must choose a device which supports Google Play, or it will be unable to install Expo Go. We recommend choosing Pixel 7.
3. **Optional:** Install Expo Go from Google Play. This is not usually necessary, as it will install automatically when you run the test build.

7.3 Downloading the source code

The source code for Bloom is available from
https://github.com/dfoshidero/Bloom

We recommend using Git to copy the source code, but it is also possible to download the source code as a ZIP file.

Using Git to copy the source code

Begin by following these instructions to create a fork of the GitHub repository: https://docs.github.com/en/get-started/quickstart/fork-a-repo

If you are using Git in the terminal or Git Bash, you can then use these commands to download your fork:

1. `cd <path>`
   where `<path>` is the full path of the folder where you want to put your local repository. Note that the repository's root folder will be **inside** that folder.
2. `git clone <URL>`
   where `<URL>` is the full URL of your forked repository. You can copy and paste the URL directly from your web browser's address bar.

Downloading a ZIP file

Follow these instructions:
https://docs.github.com/en/repositories/working-with-files/using-files/downloading-source-code-archives

7.4 Running the test build

In the terminal or Git Bash, enter the following commands:

1. `cd <path>`
   where `<path>` is the full path of your project root directory.
2. `npm install`
   to install dependencies.
3. `npx expo start`
   if you are using a physical mobile device. A QR code will appear, which you will need to scan with your device.

   OR

   `npm run android`
   if you are using an Android virtual device.

When you run the test build, you may see the following warning:
`Some dependencies are incompatible with the installed expo version`
You may be prompted to fix this issue with the command:
`npx expo install --fix`

Do not use this command.

The reason for this warning is that the app was intentionally designed to use older versions of some libraries, as it relies on features which have been removed in the most recent versions. If you use the fix command, or attempt to update these libraries by any other method, the app will not compile.

8 Build instructions

The application is built into an APK using the Expo and EAS CLI.

How to update

This is a relatively generic process for building an APK and it will probably not need to change. However, if any changes to the process occur (including workarounds for troublesome steps), the instructions must be updated to match.

8.1 Prerequisites

To prepare for the build process:

- Set up & checkout a new git branch for Build, as these changes should not be applied to main.
- Install Expo CLI: `npm install -g expo-cli`.
- Install the latest EAS CLI: `npm install -g eas-cli`.
- Register for an Expo account if necessary, then log in using `eas login`.

8.2 Configure the project

- Run `eas build:configure` to configure the project for EAS Build.
- Configure the project details in ‘app.json‘, including:
  - App name: "`Bloom`"
  - Slug: "`Bloom`"
  - Package name: "`com.teamPlum.bloom`"
  - Version: "`1.x.x`"

  - Other specific settings such as icon paths, splash screen paths, or asset bundle patterns.

- Set the appropriate build settings in ‘`eas.json`‘ for APK generation, including ensuring ‘`android.buildType`‘ set to ‘`apk`‘.

8.3 Run a build

- Identify any potential issues with `npx expo-doctor`.
- Start the build process with `eas build -p android --profile [profile-name]`, substituting `[profile-name]` with your chosen build profile like ‘preview‘ or ‘production‘.
- Monitor build progress and logs via `eas build:list`.

8.4 Deploy the build

- For emulators: After the build, use `eas build:run -p android` to download and install the APK on an Android Emulator.
- For physical devices: Download the APK from the build details page or the provided link, and install it on the device. Alternatively, use ‘adb install path/to/the/file.apk‘ for ADB installations.

8.5 Next steps

- Test the build thoroughly on both emulators and physical devices.
- In case of crashes or other issues, refer to Production build issues for troubleshooting guidance.
- For future app store deployment, consult Expo documentation on using `eas submit`.

9 Troubleshooting

This chapter describes known issues which can occur when building for testing or production.

How to update

Any build issues must be listed in this chapter as they arise, even if there are no known fixes or they cannot be reproduced. Do not remove issues from this chapter unless extensive testing has guaranteed that the issue will not recur.

9.1 Production build issues

9.1.1 Using Logcat in Android Studio for troubleshooting

To troubleshoot issues with the "Bloom" app, especially those causing crashes on startup, follow these steps in Android Studio:

1. **Set Up Logcat for Android Emulator:**
   - Open your project in Android Studio.
   - Start your Android Emulator.
   - Open the Logcat window (usually found at the bottom of the Android Studio window).
2. **Filter for the "Bloom" Package in Logcat:**
   - In the Logcat window, find the search bar.
   - Enter package name, `com.teamPlum.bloom`, to filter logs specific to our app.
3. **Install and Run the App on Emulator:**
   - Deploy your app to the emulator.
   - Run the app. This will begin generating logs in Logcat.

4. **Analyze Logs for Issues:**
    - Thoroughly read through the logs in the Logcat.
    - Look for errors or warnings that may indicate what is causing the app to crash.
5. **Identify and Fix Issues:**
    - Common issue: Missing `"react-native-gesture-handler"`. This is required for the built app, though not for dev tests.
    - Fix the identified issues and rebuild the app.
    - Re-run and re-test to ensure the issues are resolved.

9.2 Test build issues

There are no known test build issues at present.

10 Expansion

This chapter contains suggestions and tips for further development.

Process is an overview of the process for adding a new feature.

Extension points describes some of the most common changes and additions that developers will need to make in order to expand the app.

Feature recommendations contains ideas for features which could be added to the app in future.

How to update

Process should be updated to reflect the current team practices for adding features.

Extension points should be added whenever commonalities are found between new features, i.e. when developers find that certain steps are being repeated for each new feature they add. Each entry should describe in full how to make some kind of addition to the code, and should include cross-references to the relevant sections of the source code documentation.

Feature recommendations should be removed when those features are added. New feature ideas should not be immediately added to this section, but should instead be added to the product backlog. It is only when development is about to be put on hold that this section should be updated with the uncompleted tasks in the backlog. Entries in this section should be at the level of detail of an overarching epic or high-level user story.

10.1 Process

Ideas for new features can be generated in brainstorming sessions. Ideally, the entire team should be present for such a session, so as to affirm their commitment to implementing the features.

Ideas may also arise from customer meetings.

The feature should be broken down into user stories, describing every detail necessary for implementation. These should then be added to the product backlog, assigned story points and prioritised accordingly.

It is the responsibility of the developer implementing the feature to ensure that every aspect of the feature, including all its interactions with the rest of the app, has been fully tested. As much testing as possible should occur before changes are committed and pushed to the GitHub repository, but this should not be a barrier to frequent commits.

10.2 Extension points

10.2.1 Adding new components

Whenever a new feature is implemented, it will usually necessitate the creation of one or more new components. Think of these as the basic building blocks of a React app, which display images and text to the user as well as processing input.

Detailed instructions for creating React components can be found at
https://react.dev/learn/your-first-component#defining-a-component

In order to be visible to the user, your new component will need to be parented to a screen or
other component. A full explanation of parent and child components can be found at
https://react.dev/learn/your-first-component#nesting-and-organizing-components

However, note that while these instructions recommend defining multiple components in the same file, the convention for this project is to create a separate file for each component, and import them as needed. For more information about importing modules, see
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import

After the component has been added, it should be documented using the structure in Components.

10.2.2 Adding new screens

For larger features, it may be necessary to create one or more new screens. A screen is a special type of component which behaves as the name suggests, i.e. it fills the entirety of the display on the user's device. For further details, see
https://reactnavigation.org/docs/screen/

If a screen is intended to be reachable by the user, it should normally be included in the HomeStackNavigator. Some special screens such as the LoadingScreen are called directly from App.

In order for the user to reach your new screen, there will need to be a component which navigates to it. Typically, this will be accomplished by adding a new button to the GameScreen.

After the screen has been added, it should be documented using the structure in Screens.

10.2.3 Adding new states

States can be used to store any information which needs to be shared between multiple components. This is achieved with contextual variables. For an in-depth guide, see https://react.dev/learn/passing-data-deeply-with-context

States can also store config details, allowing for easy modification as needed. In this case, all that is needed is an object with properties that hold the required data.

Whenever a new state is added, it should be documented according to the structure in States.

10.2.4 Saving data

Some information, such as added plants and player progress, needs to persist when the user closes and reopens the app. This should be accomplished with a contextual variable (see Adding new states) that saves all new data to `AsyncStorage` (see https://react-native-async-storage.github.io/async-storage/docs/usage).

10.2.5 Adding new plant species

A relatively easy way to add new content to the app is by adding a new plant species. The app is designed to accommodate new plant species without requiring you to write any new code. Simply follow these steps:

1. Decide how many growth stages the new plant will have. The minimum is 2 (a starting stage and a fully-grown stage) but a typical plant has 4 stages, allowing the user to gradually build up their knowledge across multiple trivia levels.
2. Add image files for the new plant to plants, and make a note of the path for each image - you will need it for the next step. Each growth stage should be represented by a separate image.
3. Add the plant's details to plantsConfig. This includes the relative path to the image files.
4. Add trivia questions and answers to plantsTriviaConfig. Each question should have 1 correct answer and 3 incorrect answers. The number of trivia levels should be 1 less than the number of growth stages.

10.2.6 Adding new plant skins

Unlockable plant skins are a simple way to reward the user for learning about their favourite plant, and giving it a brand new look at the same time, which helps to renew their interest in the game.

Unlockable skins should represent a rare species or subspecies which is closely related to the plant's default species. Ideally, the new species will have a distinctive, vibrant appearance, but it is essential that its care instructions are the same, as the trivia and plant details are shared across skins.

To add a new plant skin, you will need to create a new image asset for each of the plant's growth stages. Find the plant's directory in plants and create a subdirectory with the name of the new species, then put the image files inside.

You will also need to modify the plant's details in plantsConfig. Add a new element to the `skins` array and add the relative paths of the new image files. Finally, set the `unlockCondition`, which is the number of coins the user must spend to unlock the skin.

10.2.7 Adding new plant interaction options

The app's most important mechanics all involve interaction with the plant through its FloatingMenu component. Some mechanics, such as watering or fertilising the plant, are handled entirely within the FloatingMenu and Plant components. For more complex mechanics, the menu button navigates to another screen or displays another component.

In either case, the best way to implement a new plant interaction option is to add a new button to the Plant component's `menuItemsList` array. You will also need to create a handler function and add it to `handleMenuItemPress()`

10.3 Feature recommendations

During the initial development of Bloom, there were various ideas which emerged from brainstorming sessions and customer meetings but could not be implemented before the project deadline. Those ideas are listed here, along with an overview of the steps necessary to implement them, to serve as inspiration for future development.

10.3.1 Achievements

When the user completes certain actions, they would be rewarded with an achievement. When an achievement is unlocked, it should be immediately shown to the user via pop-up notification, alert, or by simply displaying a previously hidden component.

There should also be an achievements screen, accessible from the main GameScreen, which displays all the achievements the user has unlocked so far. It may also be desirable for the user to see some or all of the locked achievements, along with the action needed to unlock them. This would increase the user's engagement and could help to nudge them in the right direction if they aren't sure what to do next.

Achievements could include progression milestones, such as reaching a certain level, spending a certain number of coins, or reaching maximum mastery for a certain number of plants. There could also be humorous achievements, such as forgetting to water your plants for a long period of time. Achievement names should be relevant to the theme of the app: for instance, 'Plant parent' for reaching maximum mastery.

Extension points for this feature include:

- Adding new screens, so the user can see their unlocked achievements.
- Adding new states to keep a record of which achievements have been unlocked.
- Saving data so that the app 'remembers' which achievements the user has unlocked.

It would likely also be necessary to add functions to several different Components that trigger when the user takes certain actions (such as completing a mastery level) and set the achievements as unlocked.

10.3.2 Notifications

The user should receive notifications when their plants need to be watered.

The app could also use notifications as a prompt to promote engagement. For instance, if the user hasn't opened the app in a long time, it could remind them to attempt the next trivia level for their last selected plant.

The main extension point for this feature is Adding new states, and one or more Utilities would likely be required to manage the notifications.

10.3.3 Growth stages for linked plants

When a user links an in-app plant to one of their real-life plants, it should change how that plant displays its growth stages. Instead of basing the growth stage on the mastery level, the plant should grow when the user takes proper care of it.

If the plant is watered and fertilised on time, and the user does it 2 or more times in a row, each successful watering and fertilisation after the first should move the plant to the next growth stage. If the user forgets to water or fertilise the plant, its growth stage will not revert, but it will require 2 successful waterings and fertilisations before the plant grows again.

There are no extension points for this feature, as it involves changing existing behaviour rather than adding new functionality. The main module which will need to be modified is Plant.

10.3.4 Plant sharing

Users should be able to share plants with their friends. When a plant is selected, there should be an interaction option to generate a short code containing that plant's information (its species, growth stage, and real-life link if applicable). The user can then send that code to others.

When you select a slot to add a plant, there should be an option to enter the code, after which the encoded plant will appear in that slot.

Each user should have a unique identifier (i.e. a username or handle) with which their plants are labelled. When a plant sent by another user is selected, this label should be displayed.

The main extension point is Adding new plant interaction options, as this functionality is accessed through the Plant component.