UNIVERSITÉ DE LIÈGE

INFO2051

ADTrees App

Authors:
BERNARD Aurélien & OZDEMIR Kenan & SAFADI Tasnim

December 16, 2019

Contents

1 Introduction

1.1 Description

For our project we decided to create a caring game about trees. Trees can be planted in a world map which is divided into zones.

A zone can contain only one tree. In order to plant a tree, the user must first purchase the desired zone. Zones have distinguished characteristics depending on their location. Once planted, the tree health deterioration will vary depending on the actual tree type (eg: a cactus will need less hydration than a pine tree).

This brings us to our next element; trees. There are four tree types. The user may purchase any of them given that s/he has enough coins. Any tree can be named and planted in any zone. Once a tree is planted, the user may maintain it. There are three elements that the player has to consider in order to keep the tree healthy; hydration, nourishment, and pollution.

To keep a tree healthy, various items exist. For the hydration and nourishment, the user may purchase items to water and nourish the tree. As for the pollution, the user will be redirected towards a polluted zone screen. From there, the player must drag the garbage to the corresponding bin; fruits have to be put in the compost bin whereas bottle and cans must be put in the recycle bin.

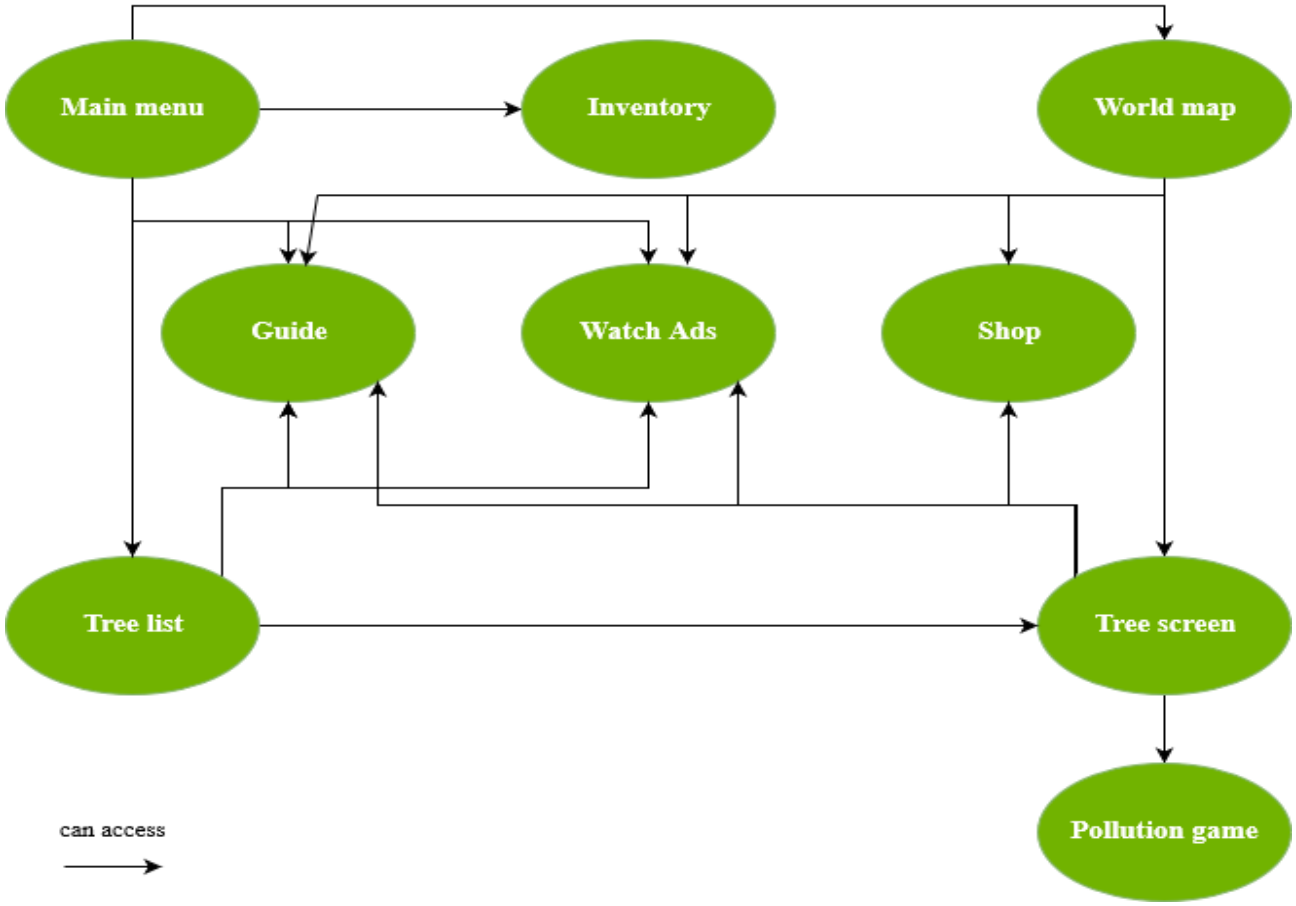
None of this would be possible without a currency. Therefore, the user starts with a certain amount of coins. Earning coins can be done in two ways.

- The first one is by watching ads, this will increase the user's available coins by 3.
- The second one is by cleaning the tree pollution. The user will earn the number of garbage removed only if there are none left. If the user leaves the game before removing all garbage, s/he will not earn any coins and the pollution of the tree will not be changed.

1.2 Structure

The final version of our game has nine different screens.

1. **Main menu:** This is the screen the user will be first presented with.
2. **Inventory:** On this page, the player may view the quantity of each item s/he possesses and their description.
3. **Shop:** All items will be purchased from this page.
4. **World map:** From this page, the user can purchase zones as well as plant trees.The zone characteristics will be displayed on this screen as well.
5. **Guide:** This page is designed to answer potential questions the player might have on the game.
6. **Ads:** Ads will be displayed via this page.
7. **Tree screen:** Any interaction with a planted tree will be done from this page.
8. **Tree list:** A list of all the planted trees. Any planted tree can be accessed from this page.
9. **Pollution game:** A polluted zone containing garbage to be cleaned.



2 Object Oriented programming

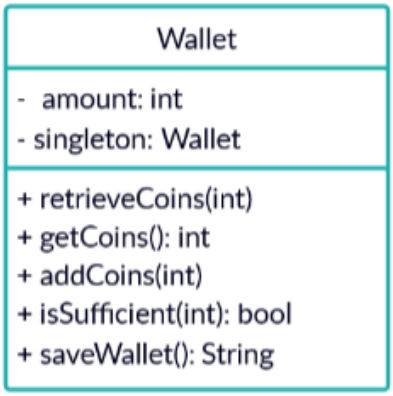
Aside from StateFulWidget, StateLessWidget, and, State classes, we created a few of our own in order to organize our program's structure.

2.1 Wallet

2.1.1 Description

This class represents the user's "wallet"; that is the number of available coins to the user. It will handle any coin spending or earning. This class is a singleton. The reason we used this design pattern is because there should only be one instance of it. In addition, we needed to easily access the number of coins available from any part of our code as well as updating that number.

2.1.2 UML

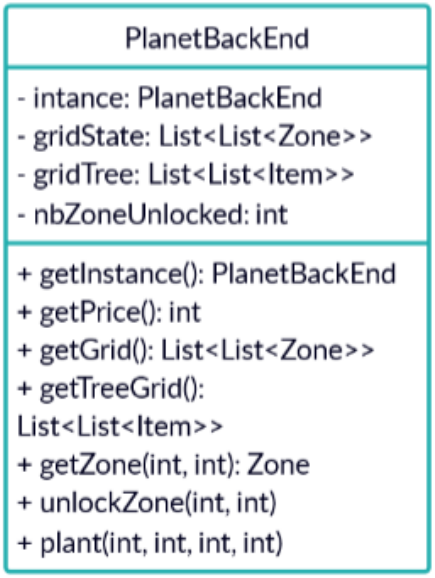


2.2 PlanetBackEnd

2.2.1 Description

The world map is represented via this class. Therefore, PlanetBackEnd contains all information regarding the ”state” of all the different zones. We decided to use the singleton design pattern for this class as well. As the evolution of the world map is unique within the game, having a single instance made sense.

2.2.2 UML

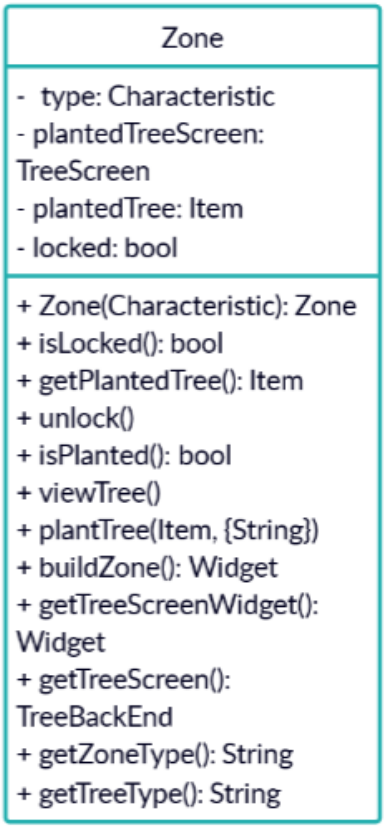


2.3 Zone

2.3.1 Description

Our world map is a grid; each square within this grid represents a zone. In order to represent a zone, we created a zone class. This class groups the characteristics of a zone, the planted tree info (if there is one and the zone is unlocked).

2.3.2 UML

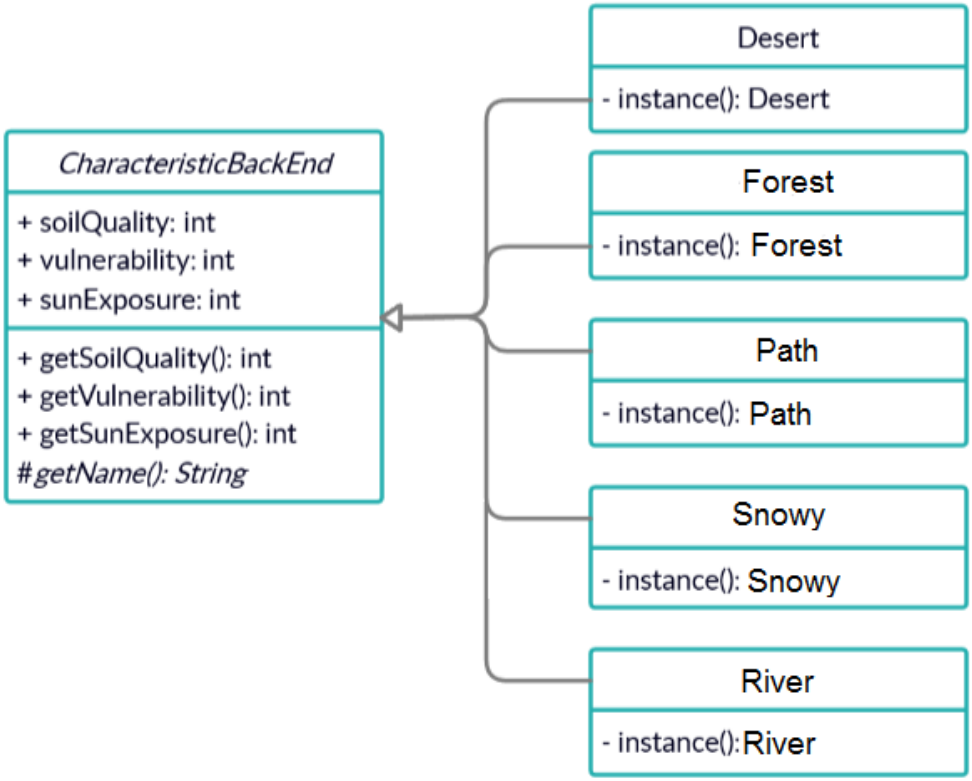


2.4 CharacteristicBackEnd

2.4.1 Description

As mentioned before, each zone has different characteristics. There are only five different types of zones; River, Path, Desert, Snowy, and, Forest. We decided to represent the characteristic as an abstract class and for each type of zone we created a singleton sub class of characteristic. Using singletons for the different types removed the need of instantiating new characteristic for each zone.

2.4.2 UML

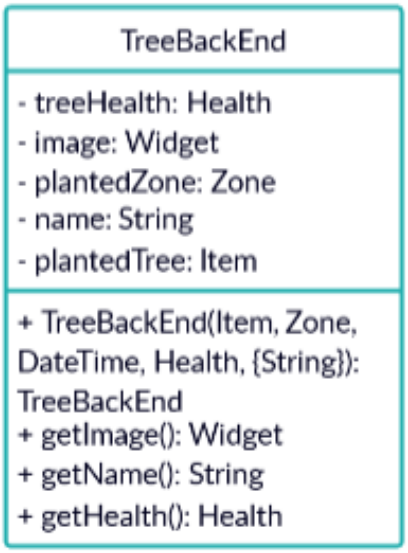


2.5 TreeBackEnd

2.5.1 Description

Everytime a new tree is planted, a new instance of this class is created. All the information regarding the planted tree can be found in this class. In particular, one can find the type of tree, its health, its name, the zone it was planted in and its image.

2.5.2 UML

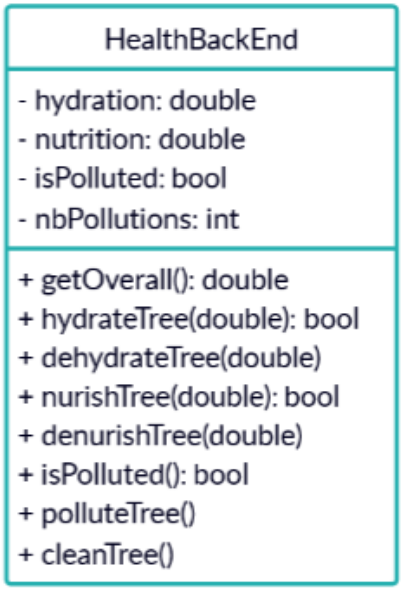


2.6 HealthBackEnd

2.6.1 Description

Each tree has a health bar. This class is used in order to manage the details of the health. The current health of a tree is determined by summing its hydration, nutrition and nbPollution. The implementation of the functions that (de)nourish, (de)hydrate, clean, and pollute the tree are defined within this class.

2.6.2 UML



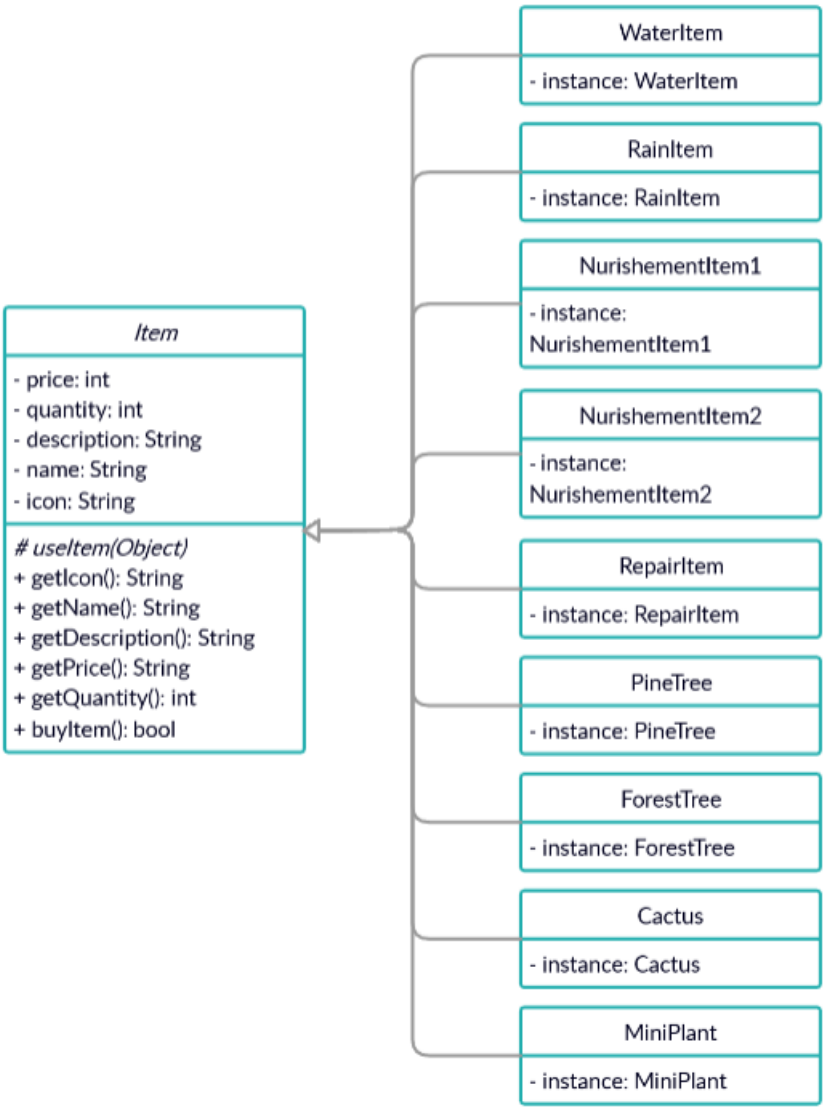
2.7 Item

2.7.1 Description

Our game requires nine items in order to have all the functionalities mentioned before. Although every item is different from the others, they still all share common elements. In fact, not only do they share common attributes but they also share the same methods. Therefore, Item is an abstract class with some methods and

some abstract methods. Every item extends Item and is a singleton. Items are represented as singletons in order to avoid multiple instantiation.

2.7.2 UML



2.8 Pollution

2.8.1 Description

In order to make the pollution game, we made a Pollution class. This class is a singleton. We simply update the game state to the current tree whenever the game is called. The garbage is generated at random positions in pollutionList.

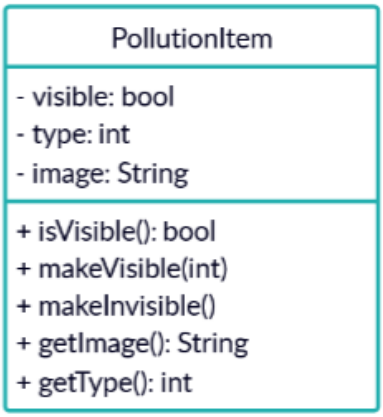
2.8.2 UML

2.9 PollutionItem

2.9.1 Description

The garbage present in the polluted zone is represented in the PollutionItem class. This class only has two attributes; its visibility and type. There are four kinds of garbage and any of these can either be visible on not.

2.9.2 UML

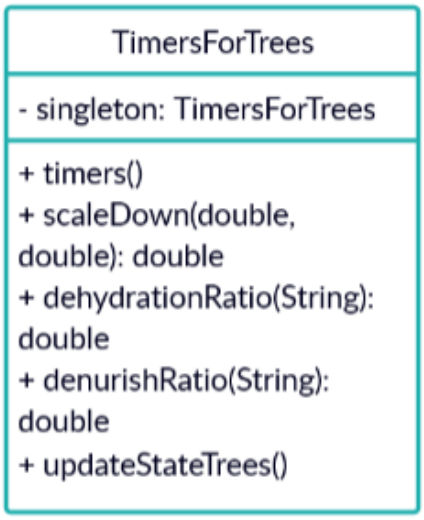


2.10 Timer

2.10.1 Description

We created another singleton class; Timer. This class updates the state of the planted trees periodically. The use of a singleton prevents multiple instantiation of this class.

2.10.2 UML

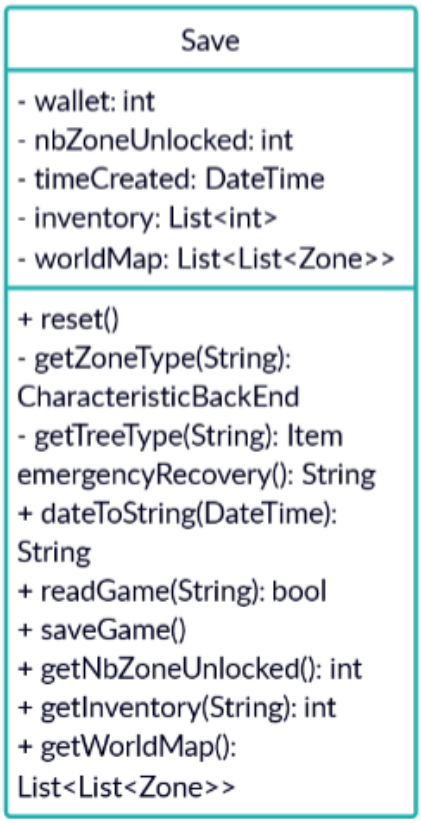


2.11 Save

2.11.1 Description

Last but not least, in order to save our game we created a singleton class Save. This class is called throughout our code, whenever a significant change is made in the state of the game; an important action of the user or the timer updates the trees. For each save, the data is encrypted and saved in a file. And when the user opens the application the game state is loaded from the encrypted file.

2.11.2 UML



3 Stateful and stateless Widgets

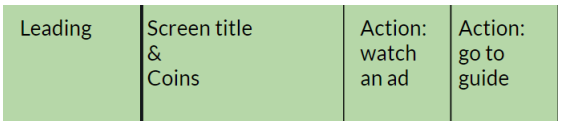
In this section we will briefly describe the widgets within our screens as well as the app bar.

3.1 AppBar

3.1.1 Description

As the appbar is an element that is present on every page, it was decided to always build it with the same structure. As it can be seen on the diagram on the right there are 3 distinct set of elements on the app bar. The leftmost element is a leading button. This button is a back arrow that pops the current context. It is only displayed if it is not the context at the bottom of the stack.The second element from the left is a rich auto size text. This text is set to be displayed on at most 2 lines. It displays the screen title and the users amount of coins in yellow. The two remaining buttons are respectively from left to right, the ”watch an ad” button for a reward and the ”go to guide” button. The latter will open the guide screen only if the user is not already on the guide screen. The color of the app bar is set to transparent to be able to customize its background on each screen individually.

3.1.2 widget



3.2 Main menu

3.2.1 Description

The main menu is compose primarily of a scaffold stacked on a decorated container to give the app bar a background. This scaffold’s body consists of container which is the parent to a list view. In this list view there is a tile for each option of the main menu. Taping on one of the tiles will push the screen indicated by the tile on top of the navigator stack.

3.2.2 widget

3.3 World map

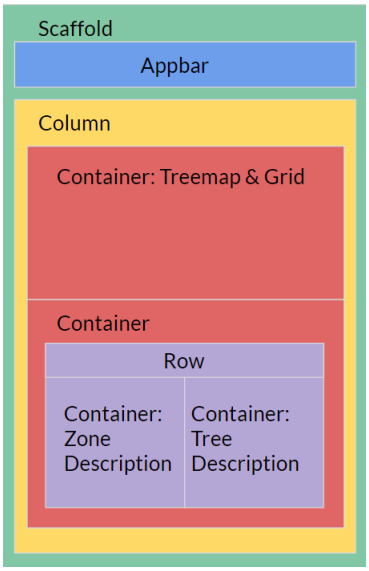
3.3.1 Description

The world map consists of a scaffold. The body of the scaffold contains a column of 2 containers. One of the container displays the map of the game by piling up an image and a grid. Whereas the other container depends on the user's activity and has a row with 2 containers: Zone description and Tree description.

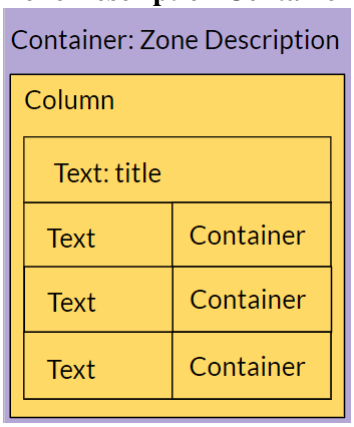
Initially, this container indicates to select a zone of the map. Once a zone is selected, its characteristics are displayed.

For this container, there are 3 possible scenarios. If the zone is locked, the user sees an unlock button. If the zone is unlocked but no trees are planted, a grid of all the different trees is displayed. If the zone is unlocked and a tree is planted, the health of the planted tree is displayed. When the user tries to plant a tree, a popup appears either asking for the name of the new tree or alerting the user that s/he has to buy plants. Another popup also appears when the user's currency is not sufficient to unlock a zone.

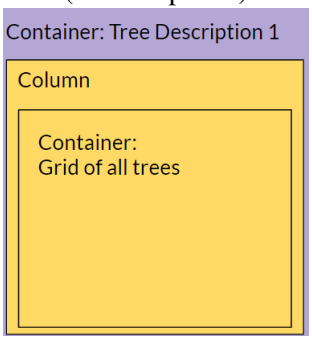
3.3.2 widget



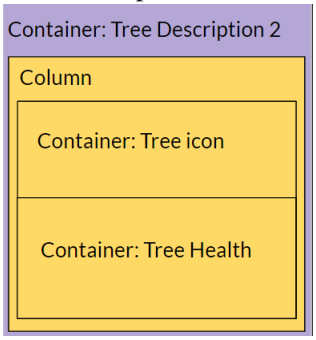
Zone Description Container



Tree Description Container
(with no plants)



Tree Description Container
(with a plant)

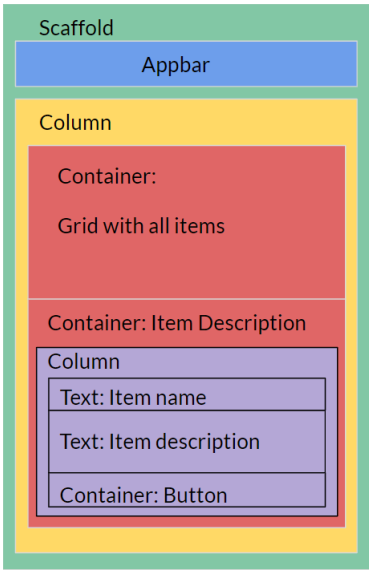


3.4 Inventory & shop

3.4.1 Description

The inventory and the shop are very similar. The body of the scaffold has a column with 2 containers. One of the container displays a grid with all items. In the case of the shop, the cost of the item is displayed whereas in the case of the inventory, the possessed quantity of the item is displayed. In the other container, there is a column with a few children: the name of the selected item, its description, and a ”Buy’ button for the shop. When the user tries to buy an item, a popup indicates if the item has been successfully bought or not.

3.4.2 widget

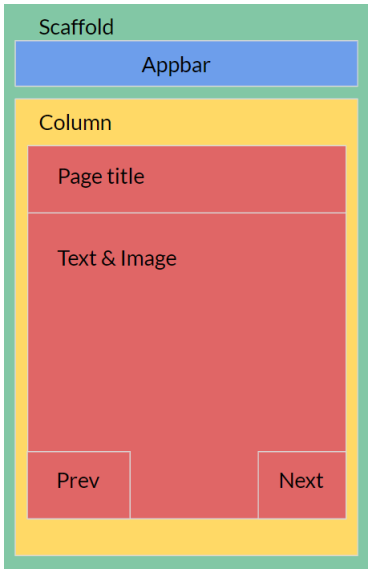


3.5 Guide

3.5.1 Description

The guide consists of a scaffold. The body of the scaffold depends on the page of the guide the user is on, it is therefore a stateful widget. The body of the widget consists of a column with 3 children. The first one holds the title of the current page. The second one is a scrollable widget holding both, the text and images of the current guide’s page. The last child of the column is a row holding two buttons to navigate between the next and previous page of the guide.

3.5.2 widget

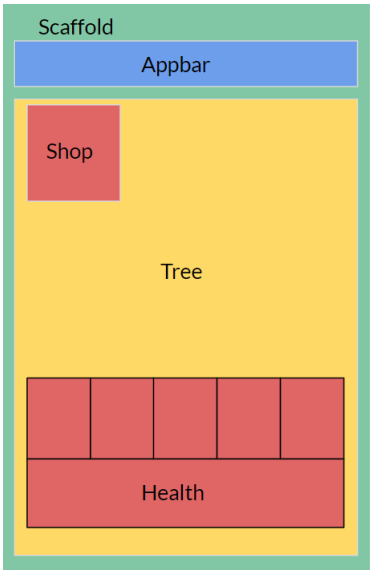


3.6 Tree screen

3.6.1 Description

The tree screen consists of a scaffold. The body of this scaffold is a column with 3 children. The first child is a container holding the tree's image. The last child is the container stacked over the tree section. This container holds a tapable image that brings the user to the shop screen. The second widget of the column is the bottom part of the screen, containing the action buttons and the health bar. The action buttons consist of row of containers encapsulated within gesture detectors. Each button performs a different action on tap. They either affect the plant's health, open a mini game, or displays a popup. Lastly the health section is container with a health bar built from two stacked containers.

3.6.2 widget



3.7 Tree list

3.7.1 Description

The tree list body consists of a column with 3 elements. Respectively, the radio buttons, the titles the elements in the list, and lastly the tree list encapsulated in a padding widget for aesthetic purposes. The radio button are built with the radio button widget followed by a container holding the button's title. Upon change of the selected sorting option the tree list is sorted and the screen is refreshed in order to reflect the change. The titles of the list elements are two containers in a row that are also affected by the selected sorting option. The last element of this screen is the tree list. This list is dynamically generated using a list view builder that builds one container for each element in the planted trees list. These elements are rows with the name of the tree and information about the tree sorting option. The elements of the list are encapsulated in gesture detectors that bring the user to the tree screen of the planted tree if tapped.

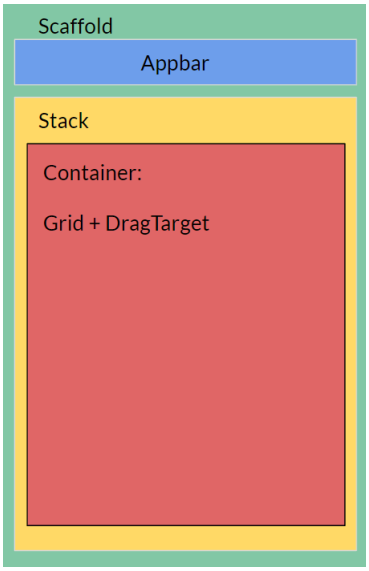
3.7.2 widget

3.8 Pollution game

3.8.1 Description

The mini-game is a scaffold with a column containing a grid and a row with 2 containers. The grid has all the polluted items. The row contains the two drag targets. The garbage is generated at random positions in the grid every time the game is opened. The tree will only be cleaned if the user gets rid of all the trash.

3.8.2 widget



4 Challenges

During the development of this application a wide variety of challenges were faced, both technical and managerial.

On the technical side most of the issues faced were due to inexperience. It was the first time that we tackled a project of such a considerable size. It was also the first time that we developed a mobile application, using a new language and its UI toolkit (dart and flutter). Building an intuitive user interface with a pleasing design was also a major challenge as none of us is an artist. All of these issues had to be solved by self learning and trial and error. Not knowing what we were doing made us implement poorly some functionalities. These faulty implementations eventually became time consuming to correct further along development. Although we attempted to plan in the best of our capabilities the various design patterns and interactions in the application, our lack of experience made us unable to consider all the difficulties and changes that occurred during development. A clear example of this is the real time refreshing of the heath bar in the tree screen. We didn't think of this issue until the application was tested, and simple modifications that solved the problem resulted in memory leaks. Due to time constraints, we therefore decided to not refresh automatically the health bar image and leave it as it is.

On the organizational side, the main issues were time constraints and labour division. During this project we tried to always work together, in a "sprint" manner. This was done to be able to trace a plan and set ourselves deadlines. However, it was difficult to allocate time that suited every member as we all have responsibilities outside of university and other projects to work on. Furthermore, although every member at the beginning of the project was tasked with the implementation of different features, we all eventually worked on every part of the application. This led to misunderstandings on who modified what and how they did so, which made us loose a great deal of time.

Finally, other challenges that were more related to misfortunes than our performance were also encountered. At the end of the development we faced issues with version control. Our version control tool was wrongly used and therefore, failed at correctly pushing and merging files, which lead in some cases to lose hours of work and hundreds of lines of code at a time. Another problem that we faced was poorly documented packages to perform functionalities that we intended to have in the application. The main example

of this being notifications and advertisement packages. The package that we intended to use in order to have notifications was not working. After hours of attempting to solve the issues, this idea had to be abandoned.

5 Potential updates

Sadly, there were a few more functionalities we wanted to add to the game but didn't have time to implement. We will briefly list a few.

In our presentations, we mentioned our plan to have settings page. However, after discussing it all together, we thought a more interesting addition would have just been a language selection. We might plan on adding this feature later on. This would make the game more comprehensive for any person with English difficulties. We would have added a language class that would contain the texts in different languages and would handle returning the text corresponding with the current language of the game.

Another feature we wanted to add was the possibility of selling already purchased items and removing a tree that has already been planted. Given the structure of our code, this could have been easily done by simply adding a few extra functions.

As for the backend, initially, the characteristics of a zone were supposed to influence the progression of the tree's health. Sadly, this is currently not the case. We do however, plan on adding this feature later on. Again, thanks to our code's structure, this can be easily done with the addition of a couple functions.

Last but not least, this isn't directly related to the actual game but rather the coding style; we could have made things "cleaner" by adding a "util" file. This file would contain all the functions with similarities (eg: creating a popup function). This would have reduced the overall number of functions we had to implement.

This section could likely be endless; there are plenty of ideas we could have implemented to make the game more entertaining. And we plan on making slowly but surely our game more consistent.

6 Conclusion

This project allowed us to experiment for the very first time such a high-level programming language. Despite not knowing the implementation of the libraries, it was most of the time very intuitive to use them thanks to their good documentation. Any of our ideas could be implementable through the huge amount of libraries that were available. Even tho we were a bit concerned about the optimality of the functions that we used, the performance of the app was always smooth and responsive. Additionally, running the app was a satisfaction as the fruit of our latest efforts could be directly visualised and appreciated. The app seemed to be endlessly improvable with new supplementary features. The latter reasons led to many adaptations of the functionalities' exact definition. So, many discussions were necessary to determine the boundaries of the app. Fortunately, we have always been working together for this app and there was no lack of communication. Overall, we estimate to have spent around 60 hours each of us for this project.