

|  |
| --- |
| G6 – Team 06  Clarence Ngoh Peng Yu  Teo Pei Sheng Nicholas  Yan Han |



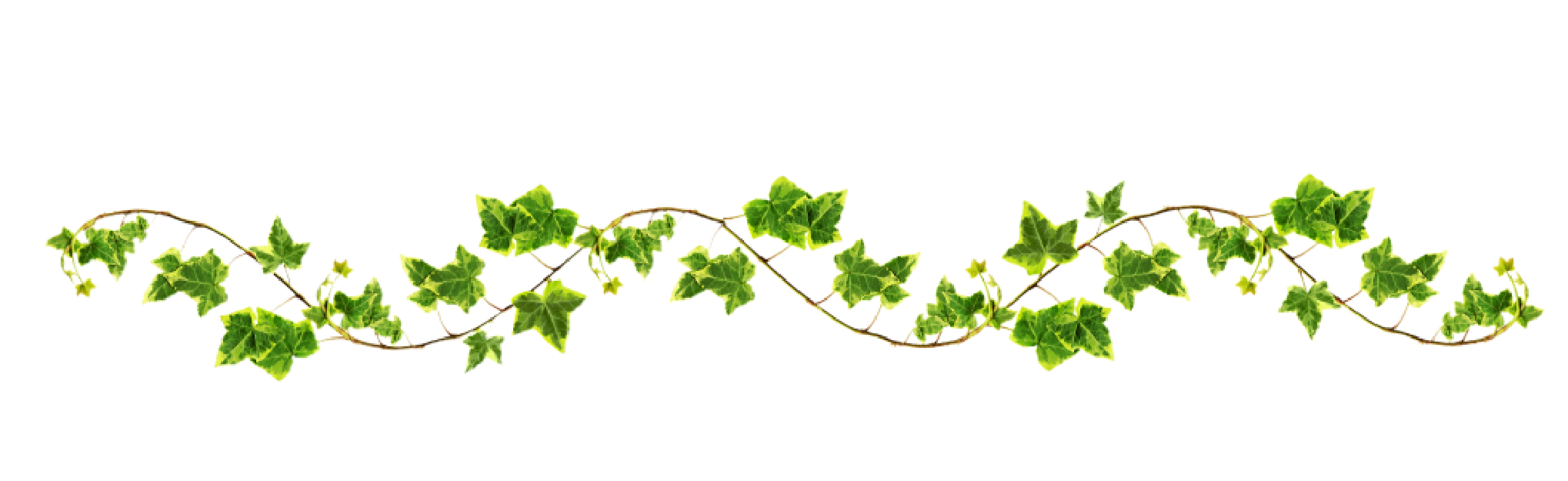


Table of Contents

[1. Use Case Diagram 4](#_Toc416206786)

[1.1 Use Case Description 4](#_Toc416206787)

[2. Domain Diagram 5](#_Toc416206788)

[3. Use Case Specifications 5](#_Toc416206789)

[3.1 Register 5](#_Toc416206790)

[3.2 Login 6](#_Toc416206791)

[3.3 Logout 6](#_Toc416206792)

[3.4 Send Friend Request to Player 7](#_Toc416206793)

[3.5 Unfriend Player 8](#_Toc416206794)

[3.6 Accept or Reject Friend Request 8](#_Toc416206795)

[3.7 Clear Plot 9](#_Toc416206796)

[3.8 Harvest Plot 9](#_Toc416206797)

[3.9 Plant Crop 10](#_Toc416206798)

[3.10 Buy Crop 10](#_Toc416206799)

[3.11 Gift Crop 11](#_Toc416206800)

[4. System Sequence Diagram 12](#_Toc416206801)

[4.1 Register 12](#_Toc416206802)

[4.2 Login 13](#_Toc416206803)

[4.2 Logout 14](#_Toc416206804)

[4.3 Send Friend request to Player 14](#_Toc416206805)

[4.4 Unfriend Player 15](#_Toc416206806)

[4.5 Accept Friend request 15](#_Toc416206807)

[4.6 Reject Friend request 16](#_Toc416206808)

[4.8 Clear Plot 16](#_Toc416206809)

[4.9 Harvest Plot 17](#_Toc416206810)

[4.10 Plant Crop 18](#_Toc416206811)

[4.11 Buy Crop 19](#_Toc416206812)

[4.12 Gift Crop 19](#_Toc416206813)

[5. Sequence Diagram 20](#_Toc416206814)

[5.1 Register 20](#_Toc416206815)

[5.2 Login 20](#_Toc416206816)

[5.3 Logout 21](#_Toc416206817)

[5.4 Send Friend Request to Player 21](#_Toc416206818)

[5.5 Unfriend Player 22](#_Toc416206819)

[5.6 Accept Friend Request 22](#_Toc416206820)

[5.7 Reject Friend Request 22](#_Toc416206821)

[5.8 Clear Plot 23](#_Toc416206822)

[5.9 Harvest Plot 24](#_Toc416206823)

[5.10 Plant Crop 25](#_Toc416206824)

[5.11 Buy Crop 26](#_Toc416206825)

[5.12 Gift Crop 26](#_Toc416206826)

[6. Class Diagram 27](#_Toc416206827)

[7. Object-Oriented Design Considerations 29](#_Toc416206828)

[8. Miscellaneous 32](#_Toc416206829)

[8.1 Lessons learnt 32](#_Toc416206830)

9. Game Manual ………………………………………………………………………………………………………………………….33

# Use Case Diagram



## Use Case Description

|  |  |
| --- | --- |
| Use Case | Description |
| Register | This use case allows an unregistered user to register a new account. |
| Login | This use case allows player to login. |
| Logout | This use case allows player to logout. |
| Send friend request to Player | This use case allows player to send friend request to another player. |
| Accept friend request | This use case allows player to accept friend request from another player. |
| Reject friend request | This use case allows player to reject friend request from another player. |
| Unfriend Player | This use case allows player to unfriend another player. |
| Clear Plot | This use case allows player to clear plot. |
| Harvest Plot | This use case allows player to harvest plot. |
| Plant Crop | This use case allows player to plant crop. |
| Buy Crop | This use case allows player to buy crop. |
| Gift Crop | This use case allows player to gift crop to another player. |

# Domain Diagram



# Use Case Specifications

## Register

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Register |
| Actor: | Player |
| Preconditions: | 1. The player has selected register option on the login menu. |
| Main Flow of Events: | 1. The system prompts the player for a username, and the player enters his preferred username. 2. The system checks if there is an identical username in the player.csv file. 3. The system prompts the player for a password, and the player enters his preferred password. 4. The system stores the player’s new username and password into the player.csv file and displays a message for the successful registration. 5. The player is brought back to the login menu page. 6. The end. |
| Alternate Flow of Events: | 1. If the player has entered an invalid username, the system displays an error message to prompt him that the username has been taken. 2. The system returns back to the login menu. |

## Login

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Login |
| Actor: | Player |
| Preconditions: | 1. The player has selected login option on the login menu. |
| Main Flow of Events: | 1. The system prompts the player for the username and password. 2. The player enters the username and password, and the system checks if the username has been registered before and the password is correct. 3. The player is brought to the main menu page, where the system will display a welcome message with the following information:  * Name/Username * Rank * XP * Gold  1. The end. |
| Alternate Flow of Events: | 1. If the player has entered an invalid username, the system displays an error message to prompt him to register first. 2. The system returns back to the login menu. 3. If the player has entered an invalid password, the system displays an error message indicating that the password is wrong. 4. The system returns to the login menu. |

## Logout

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Logout |
| Actor: | Player |
| Preconditions: | 1. The player has to be login and selected logout on the main menu. |
| Main Flow of Events: | 1. The player will be logout. |

## 3.4 Send Friend Request to Player

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Send Request |
| Actor: | Player |
| Preconditions: | 1. The player is in the friend menu, which displays the lists of friends and friend requests. 2. The player has selected the option to send request. |
| Main Flow of Events: | 1. The player inputs the name of the username of the player he wishes to befriend. 2. The system checks if the input matches any of the username of current friends or the player himself. 3. The system retrieves the player and checks if there is already a request between the two players. 4. The system displays a message of the request being successfully sent with the name of the player receiving the request. 5. The end. |
| Alternate Flow of Events: | 1. If the player inputs a username of his current friends. 2. The player is notified that this player is already a friend. 3. The system returns to the friend menu. 4. If the player inputs his own username. 5. The player is notified that he cannot be friends with himself. 6. The system returns to the friend menu. 7. If the player inputs an invalid username that does not belong to any player. 8. The player is notified that the player does not exist. 9. The system returns to the friend menu. 10. If the player inputs a username of a player that already has a pending request. 11. The player is notified that a request is pending between the two players. 12. The system returns to the friend menu. |

## 3.5 Unfriend Player

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Unfriend |
| Actor: | Player |
| Preconditions: | 1. The player is in the friend menu, which displays the lists of friends and friend requests. 2. The player has selected the option to unfriend a player. |
| Main Flow of Events: | 1. The player inputs the friend number displayed on the friend menu corresponding to the player that he wishes to unfriend. 2. The system retrieves the player and removes the player from the list of friends. 3. The system displays a message indicating that the player is no longer a friend. 4. The end. |
| Alternate Flow of Events: | 1. If the player inputs a number that is not a valid request number. 2. The player is notified of the invalid input. 3. The system returns to the friend menu. |

## 3.6 Accept or Reject Friend Request

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Accept/Reject Request |
| Actor: | Player |
| Preconditions: | 1. The player is in the friend menu, which displays the lists of friends and friend requests. 2. The player has selected the option to accept or reject request. |
| Main Flow of Events: | 1. The player inputs the request number displayed on the friend menu corresponding to the request that he wishes to accept or reject. 2. The system retrieves the request and changes the status of the request to be accepted or rejected. 3. The system displays a message of the request being successfully accepted or rejected with the name of the request’s sender. 4. The end. |
| Alternate Flow of Events: | 1. If the player inputs a number that is not a valid request number. 2. The player is notified of the invalid input. 3. The system returns to the friend menu. |

## 3.7 Clear Plot

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Clear Plot |
| Actor: | Player |
| Preconditions: | 1. The player is in the farm menu, which displays the plots of land. 2. The player has selected the option to clear plot. |
| Main Flow of Events: | 1. The player inputs the plot number of the plot he wishes to clear. 2. The system gets the plot and checks if the plot has wilted. 3. If the plot is has wilted, the system clears the plot and deducts 5 gold from the player. 4. The system displays a message of the plot being cleared. 5. The end. |
| Alternate Flow of Events: | 1. If the plot which the player chooses does not have a wilted plant. 2. The player is notified that the plot does not have a wilted plant to clear. 3. The system returns to the farm menu. |

## Harvest Plot

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Harvest Crop |
| Actor: | Player |
| Preconditions: | 1. The player is in the farm menu, which displays the plots of land. 2. The player has selected the option to harvest crop. |
| Main Flow of Events: | 1. The player inputs the plot number of the plot he wishes to harvest. 2. The system gets the plot and checks if the plot is fully grown. 3. If the plot is fully grown, the system gets the type of crop which is being planted on the plot. 4. The system gets the EXP and gold of the crop, and then adds both EXP and gold to the player. 5. The plot is cleared to be empty again. 6. The end. |
| Alternate Flow of Events: | 1. If the plot which the player chooses is not fully grown, the harvest is unsuccessful. 2. The player is notified of the unsuccessful attempt to harvest. 3. The system returns to the farm menu. |

## Plant Crop

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Plant Crop |
| Actor: | Player |
| Preconditions: | 1. The player is in the farm menu, which displays the plots of land. 2. The player has selected the option to plant crop. |
| Main Flow of Events: | 1. The player inputs the plot which he wishes to plant the crop. 2. The system gets the plot and checks if the plot is empty. If plot is empty, the system then gets and displays the list of available crops in the inventory. 3. The player inputs the crop number of the crop he wishes to plant. 4. The system retrieves the crop from the inventory, and then plants the crop on the plot. 5. The end. |
| Alternate Flow of Events: | 1. If the plot which the player chooses is not empty, the planting is unsuccessful. 2. The player is notified of the unsuccessful attempt to plant, and is advised to select another plot. 3. The system returns to the farm menu. |

## Buy Crop

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Buy Crop |
| Actor: | Player |
| Preconditions: | 1. The player is in the inventory menu, which displays the player’s rank, gold and crops. 2. The player has selected the option to buy crops. |
| Main Flow of Events: | 1. The system displays the list of crops available for sale, their cost, period of harvest and XP worth. 2. The player inputs the number corresponding to the number of the crop he wishes to purchase. 3. The system decrements the player’s gold and adds the crop to the player’s inventory. 4. The end. |
| Alternate Flow of Events: | 1. If the player inputs a number that is not a valid crop number. 2. The player is notified of the invalid choice, and is asked to input another number. 3. If the player inputs a quantity and he does not have enough gold to purchase that amount. 4. The player is notified that he has insufficient gold 5. The system returns to the inventory menu. |

## 3.11 Gift Crop

|  |  |
| --- | --- |
| Use Case Scenario | |
| Reference: | Gift Crop |
| Actor: | Player |
| Preconditions: | 1. The player is in the inventory menu, which displays the player’s rank, gold and crops. 2. The player has selected the option to gift crops. |
| Main Flow of Events: | 1. The system retrieves the player’s crops and displays the list of crops available for gifting. 2. The player inputs the number corresponding to the number of crop he wishes to gift. 3. The system prompts the player for the username of the friend he wishes to gift the crop to. 4. The end. |
| Alternate Flow of Events: | 1. If the player inputs a number that is not a valid crop number. 2. The player is notified of the invalid choice, and is asked to input another number. 3. If the player inputs a player’s username that does not exist. 4. The player is notified that the player does not exist. 5. The system returns to the inventory menu. 6. If the player inputs the username of a player whom he has gifted before previously in the same day. 7. The system displays a failure message indicating that a gift has been sent to the player on that day. |

# System Sequence Diagram

## Register



## 4.2 Login



## Logout



## Send Friend request to Player



## Unfriend Player



## Accept Friend request



## Reject Friend request



## 4.8 Clear Plot



## Harvest Plot



## 4.10 Plant Crop



## 4.11 Buy Crop



## Gift Crop



# Sequence Diagram

## Register



## Login



## Logout



## Send Friend Request to Player



## Unfriend Player



## Accept Friend Request



## Reject Friend Request



## Clear Plot



## Harvest Plot





## Plant Crop



## Buy Crop



## Gift Crop



# Class Diagram





# Object-Oriented Design Considerations

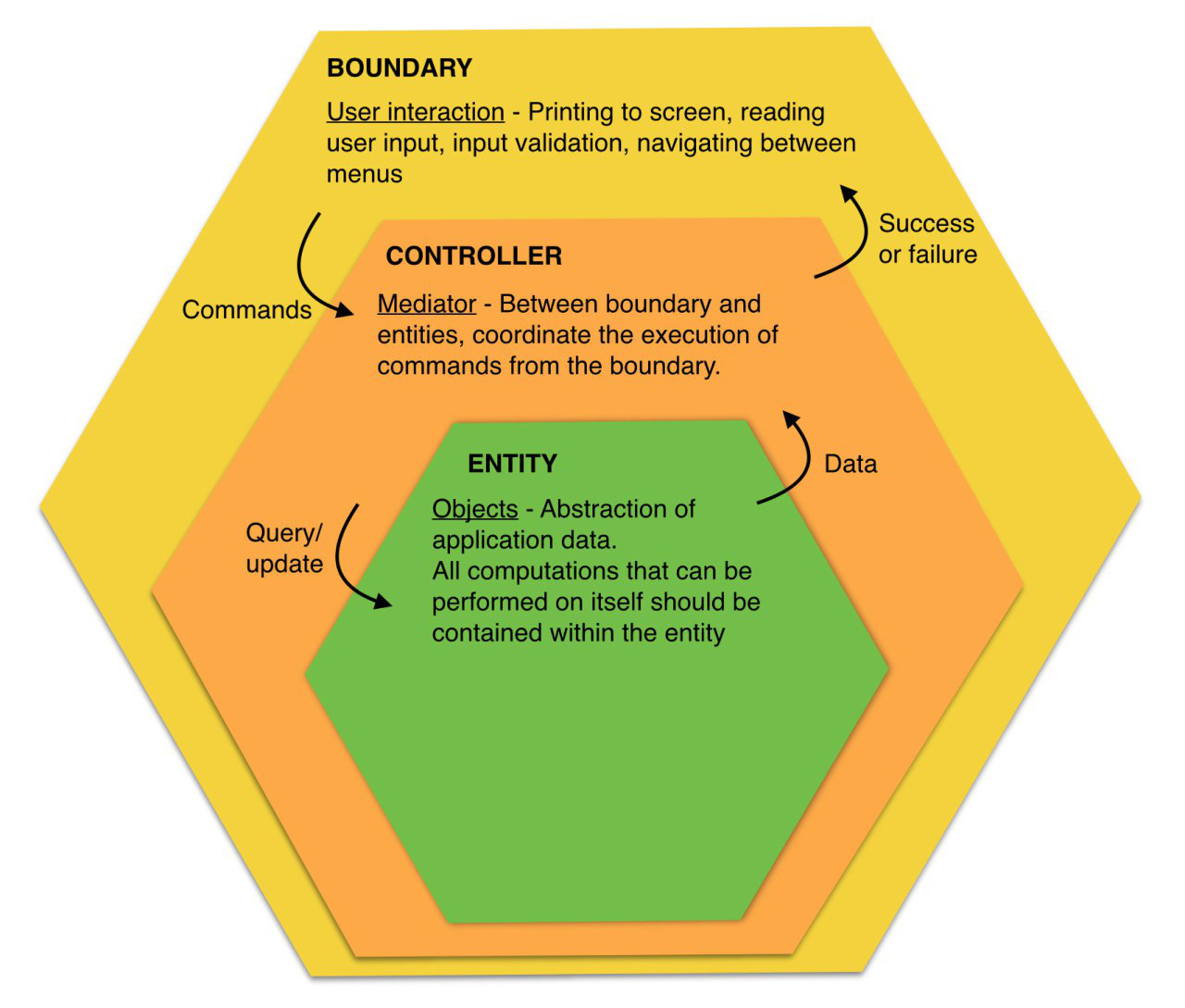
**Single Responsibility Principle**

In order to make our code easy to understand and maintain, one of the key design goals was to adhere to the SRP. What this means is that each class should have only one reason to change. Therefore, when we need to modify our application, our changes can be targeted at only a few classes without breaking or rewriting the whole app.

For example, as we use CSV files for data persistence, each CSV file is read by a separate DAO (PlayerDAO reads players.csv and CropDAO reads crops.csv). When we need to modify what we store into the Player object, we will only need to modify PlayerDAO.

**Boundary Controller Entity Pattern**

In practice, we have organised our classes into packages using the boundary - controller - entity pattern to strengthen SRP. This separation into layers has made it easy for us to conceptualise the flow of data within our application as follows.



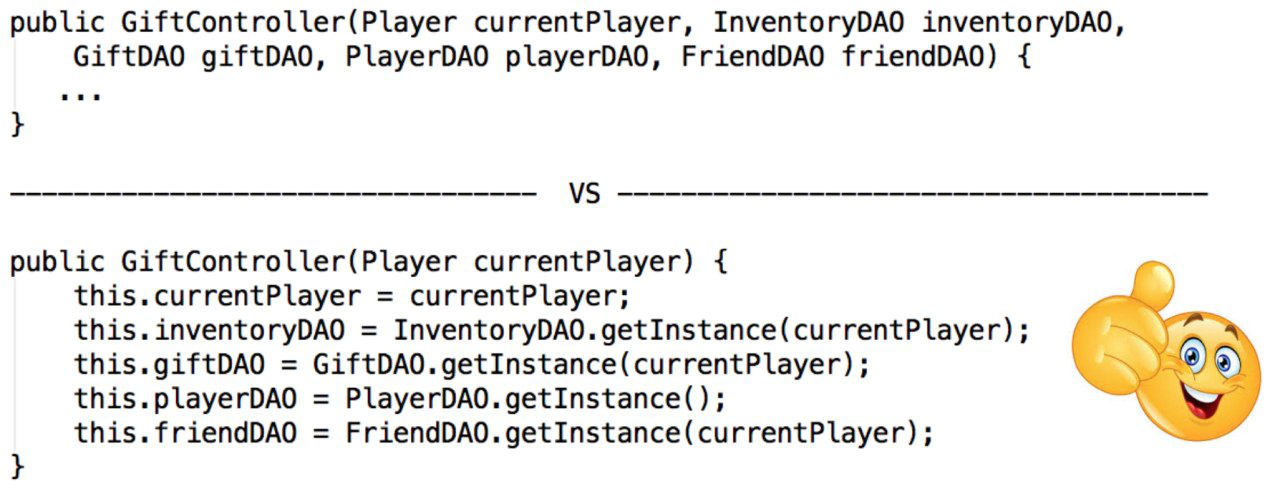
This gives us a a good guideline on where to do certain actions, for example, we will only print output and read user input in the boundary classes, not in anywhere else. This makes our code flexible, as we can one day migrate from a console app to a GUI by simply swapping out the boundary classes, with no need to change the rest of the code.

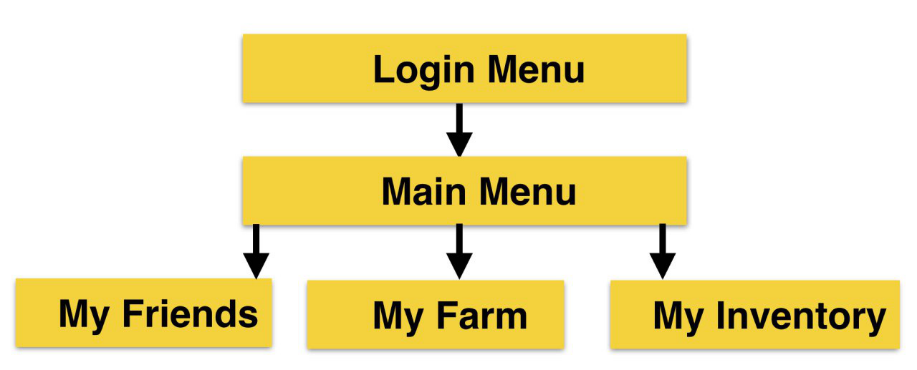
**Singleton Design Pattern**

Considering that the DAO classes are used very frequently throughout the code, sometimes in other DAOs, we needed to figure out a way to ensure that each class that uses the DAO is actually pointing to the same instance of it, for consistency of data. To do so, we designed each DAO as a singleton.

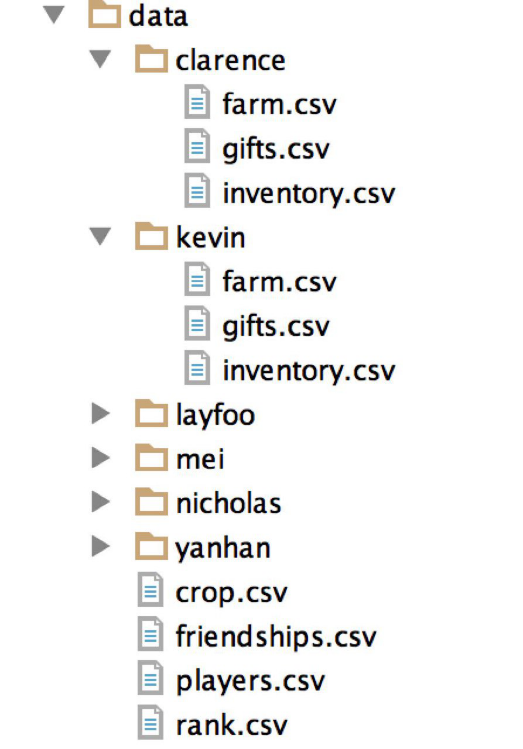
“Singletons are used for centralized management of internal or external resources and they provide a global point of access to themselves.” - www.oodesign.com/singleton-pattern

By ensuring that all our DAOs can only have one instance, and passing the reference to this instance through a static method, all classes that uses the DAOs can now conveniently access them without needing to pass in the references through extremely long constructors (which are hard to keep track of too!)



**Hierarchy of Menu classes**

We designed our menu classes according to the actual levels of menus within the application. Each instance of a Menu object is created by a Menu object from the previous level. This makes it easy for us to navigate to the previous menu by ending the method call.

**Data folder organisation**

In our data file, we chose to create separate directories for every user, each containing data belonging to the player. Data that is shared between players is stored in the main data folder.

The reason we did so is because this will minimise the amount of data that is read from file when a player logs in, and also the data that is written to the file when we save, as compared to saving all the data together.

This could also potentially allow multiple players to play together without overwriting each other!

**Configuration File**

We use a configuration file to store attributes that are commonly changed, instead of hardcoding it in our code. This makes it easy for us to make changes to game behaviour without recompilation. Implementation wise, we are using the Properties object to read the configuration file and a static wrapper class to make these ‘environment variables’ accessible anywhere within our code.

# http://www.fcpl.org/sites/default/files/images/hero/Farm_City_hero.jpgMiscellaneous

## Lessons learnt

In this OOAD project, I learn that other than knowing how to write codes, it is important to be able to understand my team mates’ codes and vice versa. Thus highlighting the importance of having a good practice of commenting our codes regularly. In addition, I have also learnt the importance of having patience and perseverance as sometimes it can be quite annoying trying to debug our codes.

* Nicholas

It was really an enjoyable and fruitful journey because I was able to learn a lot from my team mates. I learn the beauty of Object Oriented concepts, how it makes the application more readable and flexible. Other than that, I have also learn how to communicate with my team mates to make sure that everyone is on the same page throughout the project. Furthermore, it is important to ensure consistency throughout the project so as to uphold a certain standard for our codes.

* Clarence

Through this project, I learn a lot about how to do an entire project, especially on the designing aspect of an application. At the same time, I have also learnt how to work and communicate with different people and be receptive to other people’s idea. It was tough at first as we did not knew each other before this. However, overall I have enjoyed the process of the project and interacting with my team mates.

* Yan Han