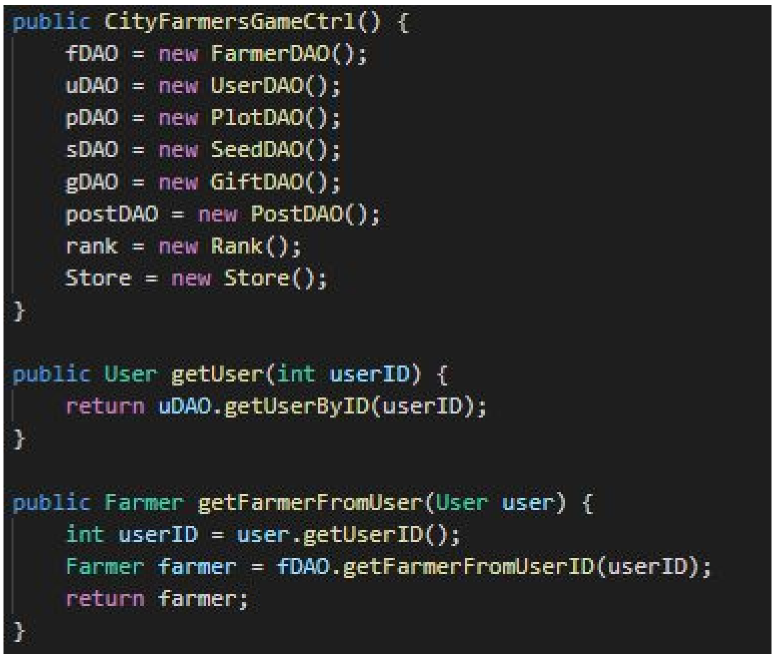
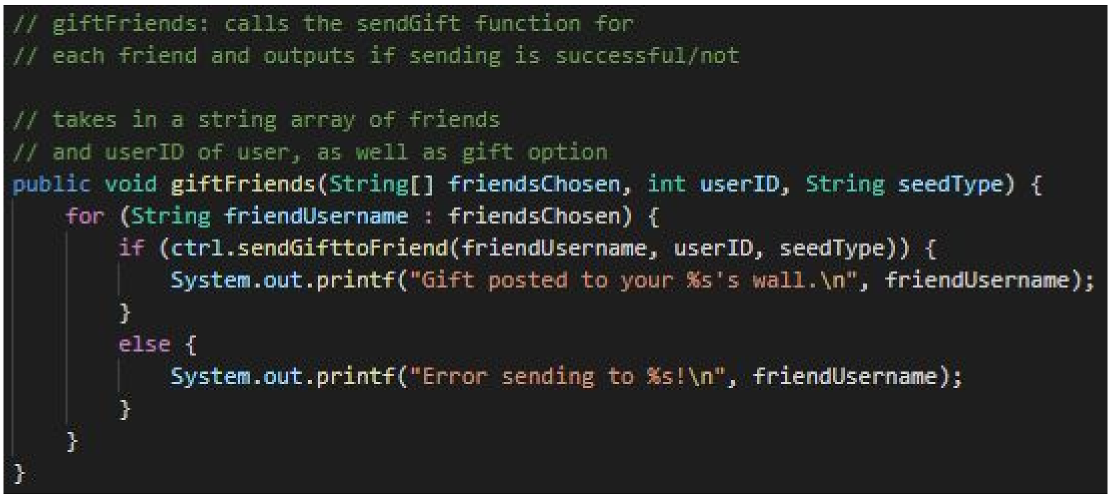
Object Oriented Design - SOLID Principles

1. **The Single Responsibility Principle — Classes should have a single responsibility and thus only a single reason to change**.
   a. CityFarmersCtrl.java: Interacting with all the DAO classes for updating the database
      An example: the getUser and getFarmerFromUser methods in CityFarmersCtrl



```java
public CityFarmersGameCtrl() {
    fDAO = new FarmerDAO();
    uDAO = new UserDAO();
    pDAO = new PlotDAO();
    sDAO = new SeedDAO();
    gDAO = new GiftDAO();
    postDAO = new PostDAO();
    rank = new Rank();
    Store = new Store();
}

public User getUser(int userID) {
    return uDAO.getUserByID(userID);
}

public Farmer getFarmerFromUser(User user) {
    int userID = user.getUserID();
    Farmer farmer = fDAO.getFarmerFromUserID(userID);
    return farmer;
}
```

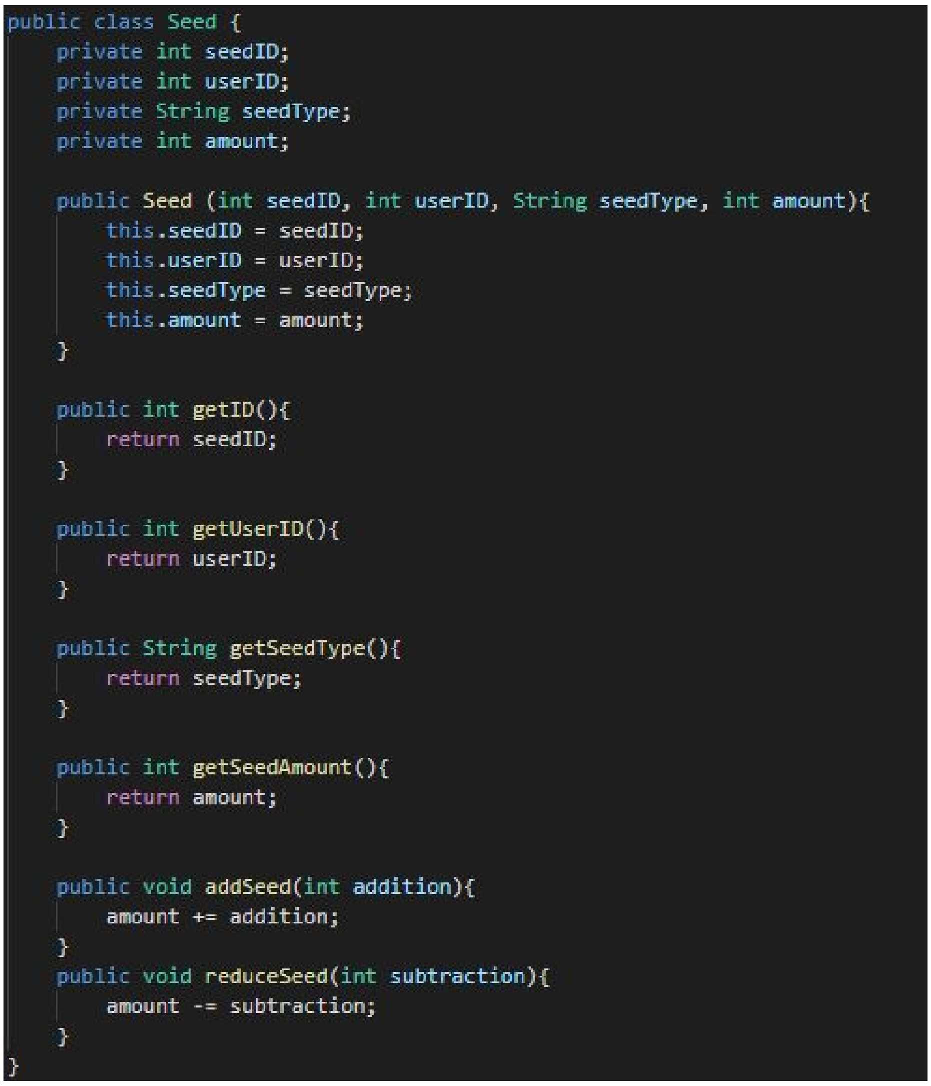
   b. CityFarmersMenu.java: for handling and processing user input and most print statements for users.
      An example is the giftFriends method in CityFarmersMenu, which calls the method in CityFarmersCtrl (initialised as ctrl here). CityFarmersCtrl does the processing and logic of the method, and CityFarmersMenu is responsible for printing out the relevant statements depending on the success or failure of the method.



```java
// giftFriends: calls the sendGift function for
// each friend and outputs if sending is successful/not

// takes in a string array of friends
// and userID of user, as well as gift option
public void giftFriends(String[] friendsChosen, int userID, String seedType) {
    for (String friendUsername : friendsChosen) {
        if (ctrl.sendGifttoFriend(friendUsername, userID, seedType)) {
            System.out.printf("Gift posted to your %s's wall.\n", friendUsername);
        }
        else {
            System.out.printf("Error sending to %s!\n", friendUsername);
        }
    }
}
```

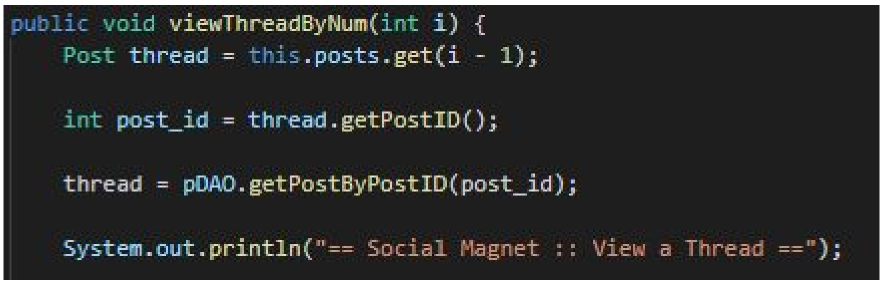
   c. DAO Classes: only handles connection with DB

d. Each class (e.g Farmer, Plot classes) have functions specific to its own object. (Encapsulation of object-specific functions within the classes itself) An example would be Seed class, which has addSeed and reduceSeed function specific to the Seed object.

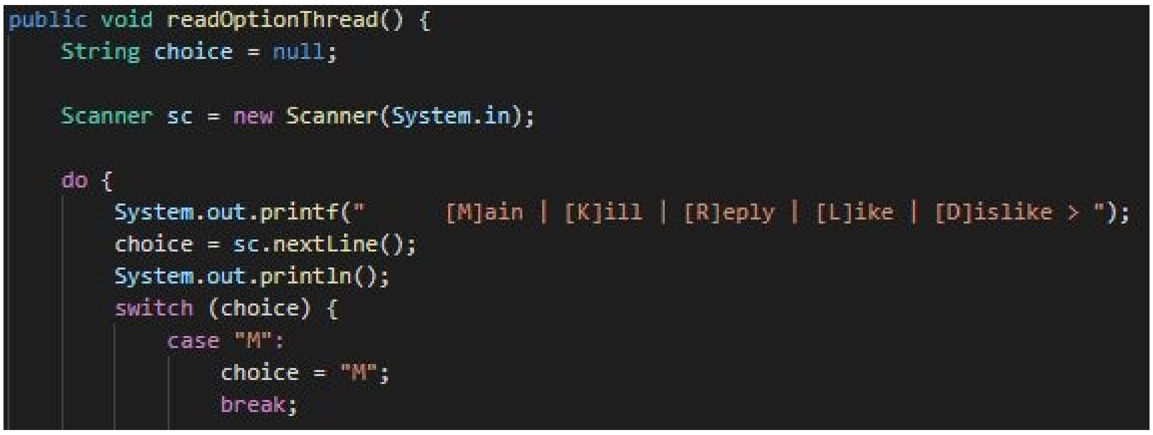


```java
public class Seed {
    private int seedID;
    private int userID;
    private String seedType;
    private int amount;

    public Seed (int seedID, int userID, String seedType, int amount){
        this.seedID = seedID;
        this.userID = userID;
        this.seedType = seedType;
        this.amount = amount;
    }

    public int getID(){
        return seedID;
    }

    public int getUserID(){
        return userID;
    }

    public String getSeedType(){
        return seedType;
    }

    public int getSeedAmount(){
        return amount;
    }

    public void addSeed(int addition){
        amount += addition;
    }
    public void reduceSeed(int subtraction){
        amount -= subtraction;
    }
}
```

e. Wall.java: Responsible for handling functions related to the wall
Example: viewThreadByNum and readOptionThread methods



```java
public void viewThreadByNum(int i) {
    Post thread = this.posts.get(i - 1);

    int post_id = thread.getPostID();

    thread = pDAO.getPostByPostID(post_id);

    System.out.println("== Social Magnet :: View a Thread ==");
```



```
public void readOptionThread() {
    String choice = null;

    Scanner sc = new Scanner(System.in);

    do {
        System.out.printf("      [M]ain | [K]ill | [R]eply | [L]ike | [D]islike > ");
        choice = sc.nextLine();
        System.out.println();
        switch (choice) {
            case "M":
                choice = "M";
                break;
```

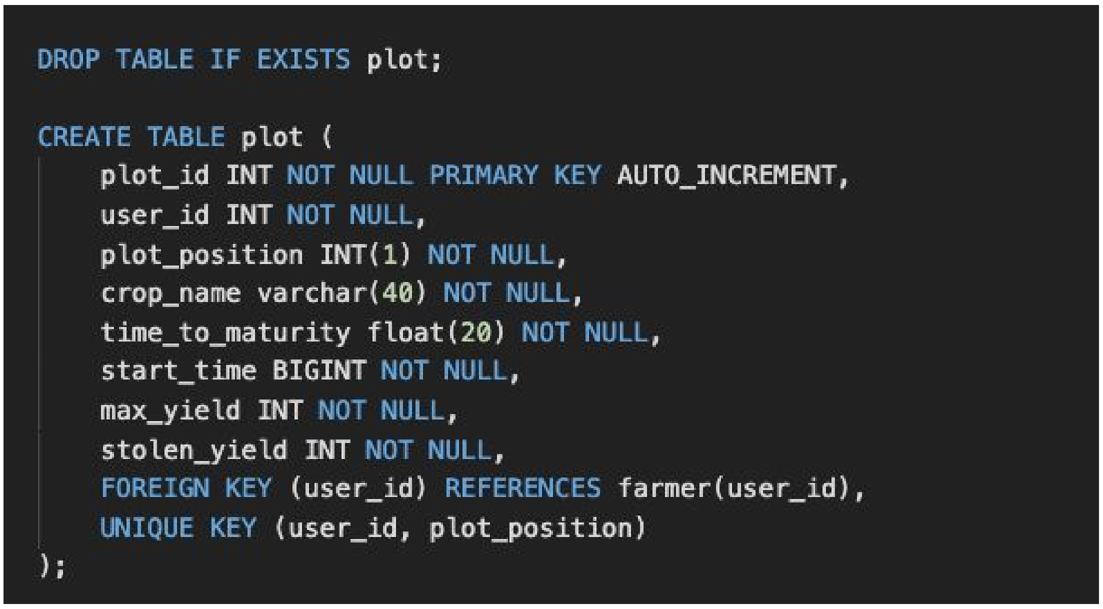
2. **The Open/Closed Principle — Classes and other entities should be open for extension but closed for modification.**
    a. CityFarmersGame: no classes extend other classes as each class / object has its own use. Any extension to the original functionality with regards to the object can be done in its own class.
    Example: Farmer.java, which doesn't extend other classes as there are no other classes which will share similar functions with Farmer.

3. **The Liskov Substitution Principle, The Interface Segregation Principle, and The Dependency Inversion Principle**
    a. Even though a Farmer "is a" User, farmer doesn't inherit user as we feel that the Farmer object has its own unique functions and attributes that a User doesn't have. Hence, a Farmer object becomes an attribute within User even though they share the "is a" relationship.
    b. As we did not see the need for any class to inherit a particular class, no parent classes were created for our project.

Other Design considerations:

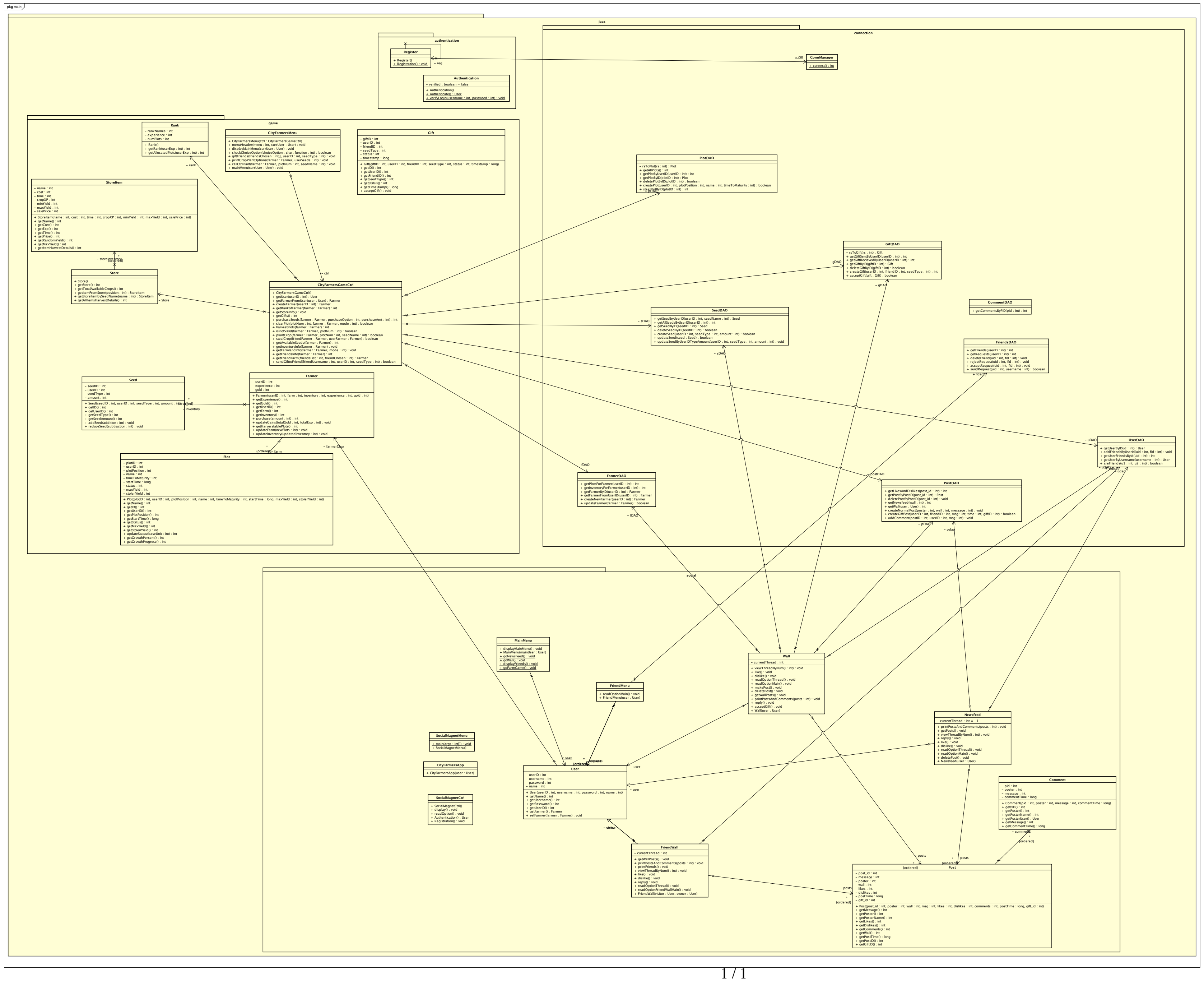
1. <u>Implementation of MVC (Model-View-Controller) structure for our application</u>
Implementing our application by splitting it into M-V-C allowed us to organise our classes depending on the functionalities of the classes and objects, achieving separation of concerns.
Our group feels that this method of separation allowed us to better classify certain functions into its respective categories, and hence made the tracing of the code more efficient and simpler for ourselves.
In addition, this ensured that our processing is separated from our display functions, allowing us to work concurrently on different aspects of the project.

2. <u>Implementation of surrogate keys across all SQL tables</u>
Surrogate keys are keys that do not have any contextual or business meaning while natural keys possess contextual or business meaning.


```sql
DROP TABLE IF EXISTS plot;

CREATE TABLE plot (
    plot_id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    plot_position INT(1) NOT NULL,
    crop_name varchar(40) NOT NULL,
    time_to_maturity float(20) NOT NULL,
    start_time BIGINT NOT NULL,
    max_yield INT NOT NULL,
    stolen_yield INT NOT NULL,
    FOREIGN KEY (user_id) REFERENCES farmer(user_id),
    UNIQUE KEY (user_id, plot_position)
);
```

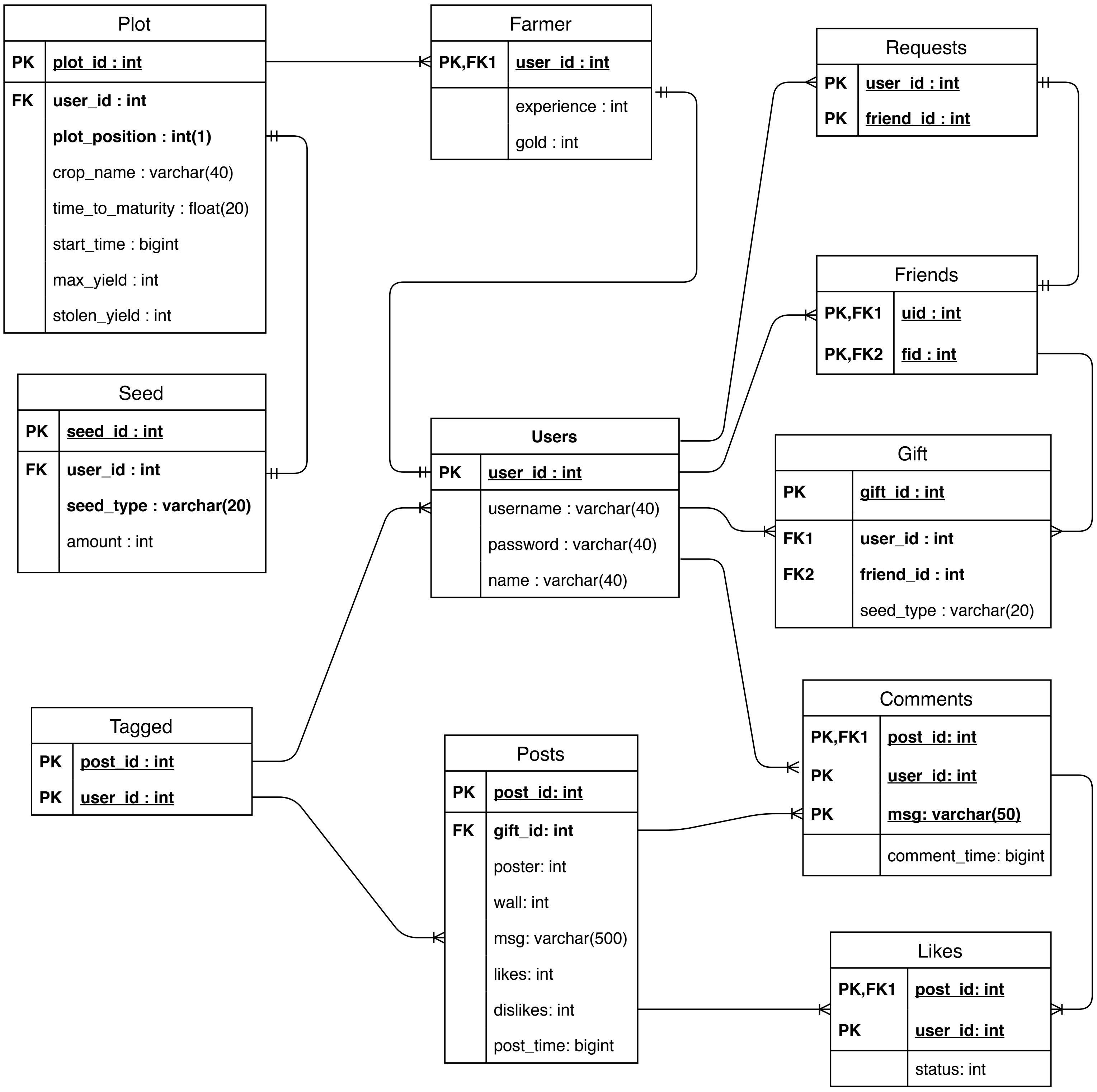
For example, the primary key for plot is a surrogate key which does not possess any contextual meaning, in the game. A plot is uniquely defined by the combination of user_id and plot_position.

Our group chose to use surrogate primary keys to allow for business continuity when changes need to be made to data type or attributes. Furthermore, it is impossible to generate duplicate surrogate keys, since these keys are system-generated. By using DAOs as a data controller, we were able to enforce logical checks before any CRUD operation is done on the MySQL database, further enhancing data integrity.


pkg main
java
authentication
Register
+ Register()
+ Registration() : void
- reg
Authentication
- verified : boolean = false
+ Authentication()
+ Authenticate() : User
+ verifyLogin(username : int, password : int) : void
connection
ConnManager
+ connect() : int
~ cm
game
Rank
- rankNames : int
- experience : int
- numPlots : int
+ Rank()
+ getRank(userExp : int) : int
+ getAllocatedPlots(userExp : int) : int
- rank
CityFarmersMenu
+ CityFarmersMenu(ctrl : CityFarmersGameCtrl)
+ menuHeader(menu : int, currUser : User) : void
+ displayMainMenu(currUser : User) : void
+ checkChoiceOption(choiceOption : char, function : int) : boolean
+ giftFriends(friendsChosen : int[], userID : int, seedType : int) : void
+ printCropPlantOptions(farmer : Farmer, userSeeds : int) : void
+ callCtrlPlant(farmer : Farmer, plotNum : int, seedName : int) : void
+ mainMenu(currUser : User) : void
- ctrl
Gift
- giftID : int
- userID : int
- friendID : int
- seedType : int
- status : int
- timestamp : long
+ Gift(giftID : int, userID : int, friendID : int, seedType : int, status : int, timestamp : long)
+ getID() : int
+ getUserID() : int
+ getFriendID() : int
+ getSeedType() : int
+ getStatus() : int
+ getTimeStamp() : long
+ acceptGift() : void
StoreItem
- name : int
- cost : int
- time : int
- cropXP : int
- minYield : int
- maxYield : int
- salePrice : int
+ StoreItem(name : int, cost : int, time : int, cropXP : int, minYield : int, maxYield : int, salePrice : int)
+ getName() : int
+ getCost() : int
+ getExp() : int
+ getTime() : int
+ getPrice() : int
+ getRandomYield() : int
+ getMaxYield() : int
+ getItemHarvestDetails() : int
- storeInventory
*
{ordered}
Store
+ Store()
+ getStore() : int
+ getTotalAvailableCrops() : int
+ getItemFromStore(position : int) : StoreItem
+ getStoreItembySeedName(name : int) : StoreItem
+ getAllItemsHarvestDetails() : int
- Store
CityFarmersGameCtrl
+ CityFarmersGameCtrl()
+ getUser(userID : int) : User
+ getFarmerFromUser(user : User) : Farmer
+ createFarmer(userID : int) : Farmer
+ getRankofFarmer(farmer : Farmer) : int
+ getStoreInfo() : void
+ getGifts() : int
+ purchaseSeeds(farmer : Farmer, purchaseOption : int, purchaseAmt : int) : int
+ clearPlot(plotNum : int, farmer : Farmer, mode : int) : boolean
+ harvestPlots(farmer : Farmer) : int
+ isPlotValid(farmer : Farmer, plotNum : int) : boolean
+ plantCrop(farmer : Farmer, plotNum : int, seedName : int) : boolean
+ stealCrop(friendFarmer : Farmer, userFarmer : Farmer) : boolean
+ getAvailableSeeds(farmer : Farmer) : int
+ getInventoryInfo(farmer : Farmer) : void
+ getFarmlandInfo(farmer : Farmer, mode : int) : void
+ getFriendsInfo(farmer : Farmer) : int
+ getFriendFarm(friendsList : int, friendChosen : int) : Farmer
+ sendGifttoFriend(friendUsername : int, userID : int, seedType : int) : boolean
Seed
- seedID : int
- userID : int
- seedType : int
- amount : int
+ Seed(seedID : int, userID : int, seedType : int, amount : int)
+ getID() : int
+ getUserID() : int
+ getSeedType() : int
+ getSeedAmount() : int
+ addSeed(addition : int) : void
+ reduceSeed(subtraction : int) : void
*
{ordered}
- inventory
Farmer
- userID : int
- experience : int
- gold : int
+ Farmer(userID : int, farm : int, inventory : int, experience : int, gold : int)
+ getExperience() : int
+ getGold() : int
+ getUserID() : int
+ getFarm() : int
+ getInventory() : int
+ purchase(amount : int) : int
+ updateGains(totalGold : int, totalExp : int) : void
+ getHarverstablePlots() : int
+ updateFarm(newPlots : int) : void
+ updateInventory(updatedInventory : int) : void
*
{ordered}
- farm
- farmerChar
Plot
- plotID : int
- userID : int
- plotPosition : int
- name : int
- timeToMaturity : int
- startTime : long
- status : int
- maxYield : int
- stolenYield : int
+ Plot(plotID : int, userID : int, plotPosition : int, name : int, timeToMaturity : int, startTime : long, maxYield : int, stolenYield : int)
+ getName() : int
+ getID() : int
+ getUserID() : int
+ getPlotPosition() : int
+ getStartTime() : long
+ getStatus() : int
+ getMaxYield() : int
+ getStolenYield() : int
+ updateStatus(baseUnit : int) : int
+ getGrowthPercent() : int
+ getGrowthProgress() : int
PlotDAO
- rsToPlot(rs : int) : Plot
+ getAllPlots() : int
+ getPlotByUserID(userID : int) : int
+ getPlotByID(plotID : int) : Plot
+ deletePlotByID(plotID : int) : boolean
+ createPlot(userID : int, plotPosition : int, name : int, timeToMaturity : int) : boolean
+ stealPlotByID(plotID : int) : int
- pDAO
GiftDAO
- rsToGift(rs : int) : Gift
+ getGiftSentByUserID(userID : int) : int
+ getGiftRecievedByUserID(userID : int) : int
+ getGiftByID(giftID : int) : Gift
+ deleteGiftByID(giftID : int) : boolean
+ createGift(userID : int, friendID : int, seedType : int) : int
+ acceptGift(gift : Gift) : boolean
- gDAO
- gDAO
SeedDAO
+ getSeedbyUserID(userID : int, seedName : int) : Seed
+ getAllSeedsByUserID(userID : int) : int
+ getSeedByID(seedID : int) : Seed
+ deleteSeedByID(seedID : int) : boolean
+ createSeed(userID : int, seedType : int, amount : int) : boolean
+ updateSeed(seed : Seed) : boolean
+ updateSeedByUserIDTypeAmount(userID : int, seedType : int, amount : int) : void
- sDAO
- sDAO
CommentDAO
+ getCommentsByPID(pid : int) : int
FriendsDAO
+ getFriends(userID : int) : int
+ getRequests(userID : int) : int
+ deleteFriend(uid : int, fid : int) : void
+ rejectRequest(uid : int, fid : int) : void
+ acceptRequest(uid : int, fid : int) : void
+ sendRequest(uid : int, username : int) : boolean
+ fdao
UserDAO
+ getUserByID(id : int) : User
+ addFriendsByUserId(uid : int, fid : int) : void
+ getUserFriendsById(uid : int) : int
+ getUserByUsername(username : int) : User
+ areFriends(u1 : int, u2 : int) : boolean
- uDAO
- uDAO
- udao
FarmerDAO
+ getPlotsForFarmer(userID : int) : int
+ getInventoryForFarmer(userID : int) : int
+ getFarmerByID(userID : int) : Farmer
+ getFarmerFromUserID(userID : int) : Farmer
+ createNewFarmer(userID : int) : Farmer
+ updateFarmer(farmer : Farmer) : boolean
- fDAO
- fDAO
PostDAO
+ getLikesAndDislikes(post_id : int) : int
+ getPostByPostID(post_id : int) : Post
+ deletePostByPostID(post_id : int) : void
+ getNewsfeed(wall : int) : int
+ getWall(user : User) : int
+ createNormalPost(poster : int, wall : int, message : int) : void
+ createGiftPost(userID : int, friendID : int, msg : int, time : int, giftID : int) : boolean
+ addComment(postID : int, userID : int, msg : int) : void
- postDAO
- pDAO
- pdao
social
MainMenu
+ displayMainMenu() : void
+ MainMenu(mainUser : User)
+ goNewsFeed() : void
+ goWall() : void
+ displayFriends() : void
+ goFarmGame() : void
FriendMenu
+ readOptionMain() : void
+ FriendMenu(user : User)
Wall
- currentThread : int
+ viewThreadByNum(i : int) : void
+ like() : void
+ dislike() : void
+ readOptionThread() : void
+ readOptionMain() : void
+ makePost() : void
+ deletePost() : void
+ getWallPosts() : void
+ printPostsAndComments(posts : int) : void
+ reply() : void
+ acceptGift() : void
+ Wall(user : User)
SocialMagnetMenu
+ main(args : int[]) : void
+ SocialMagnetMenu()
CityFarmersApp
+ CityFarmersApp(user : User)
SocialMagnetCtrl
+ SocialMagnetCtrl()
+ display() : void
+ readOption() : void
+ Authentication() : User
+ Registration() : void
- user
*
{ordered}
- friends
- requests
- user
- user
User
- userID : int
- username : int
- password : int
- name : int
+ User(userID : int, username : int, password : int, name : int)
+ getName() : int
+ getUsername() : int
+ getPassword() : int
+ getUserID() : int
+ getFarmer() : Farmer
+ setFarmer(farmer : Farmer) : void
- visitor
- owner
FriendWall
- currentThread : int
+ getWallPosts() : void
+ printPostsAndComments(posts : int) : void
+ printFriends() : void
+ viewThreadByNum(i : int) : void
+ like() : void
+ dislike() : void
+ reply() : void
+ readOptionThread() : void
+ readOptionFriendWallMain() : void
+ FriendWall(visitor : User, owner : User)
Newsfeed
- currentThread : int = -1
+ printPostsAndComments(posts : int) : void
+ getPosts() : void
+ viewThreadByNum(i : int) : void
+ reply() : void
+ like() : void
+ dislike() : void
+ readOptionThread() : void
+ readOptionMain() : void
+ deletePost() : void
+ Newsfeed(user : User)
Comment
- pid : int
- poster : int
- message : int
- commentTime : long
+ Comment(pid : int, poster : int, message : int, commentTime : long)
+ getPID() : int
+ getPoster() : int
+ getPosterName() : int
+ getPosterUser() : User
+ getMessage() : int
+ getCommentTime() : long
- comments
*
{ordered}
- posts
*
{ordered}
- posts
*
{ordered}
- posts
*
{ordered}
Post
- post_id : int
- message : int
- poster : int
- wall : int
- likes : int
- dislikes : int
- postTime : long
- gift_id : int
+ Post(post_id : int, poster : int, wall : int, msg : int, likes : int, dislikes : int, comments : int, postTime : long, gift_id : int)
+ getMessage() : int
+ getPoster() : int
+ getPosterName() : int
+ getLikes() : int
+ getDislikes() : int
+ getComments() : int
+ getWall() : int
+ getPostTime() : long
+ getPostID() : int
+ getGiftID() : int

Social Magnet

Entity Relationship Diagram with Physical Data Type



Plot
PK plot_id : int
FK user_id : int
plot_position : int(1)
crop_name : varchar(40)
time_to_maturity : float(20)
start_time : bigint
max_yield : int
stolen_yield : int
Farmer
PK,FK1 user_id : int
experience : int
gold : int
Requests
PK user_id : int
PK friend_id : int
Friends
PK,FK1 uid : int
PK,FK2 fid : int
Seed
PK seed_id : int
FK user_id : int
seed_type : varchar(20)
amount : int
Users
PK user_id : int
username : varchar(40)
password : varchar(40)
name : varchar(40)
Gift
PK gift_id : int
FK1 user_id : int
FK2 friend_id : int
seed_type : varchar(20)
Comments
PK,FK1 post_id: int
PK user_id: int
PK msg: varchar(50)
comment_time: bigint
Tagged
PK post_id : int
PK user_id : int
Posts
PK post_id: int
FK gift_id: int
poster: int
wall: int
msg: varchar(500)
likes: int
dislikes: int
post_time: bigint
Likes
PK,FK1 post_id: int
PK user_id: int
status: int