PROJECT MANAGEMENT SYSTEM

By Jedd Madayag (Tester), Nouar Mahamoud (Secretary) , Gabriel Mano Lasig (Scrum Master) , Jettlance Rivera (Software Developer),

Contents

Introduction

In this group project, we (NoName Systems) have been given the task to create a management system to efficiently manage properties and tenants within real estate context. The system is designed for the user to perform tasks in the console such as adding properties, adding tenants, leasing properties to tenants, repossessing properties, and displaying property and tenant information.

The layout of this report will start of the introduction and then writing about the design which includes a justification of selected data structures and algorithms and an analysis of the algorithms which provide key functionality. After, we are going to write about the testing section where I include a statement of testing approach used and a table of test cases. Finally, to wrap up the group report we are going to write the conclusion which will contain the summary of work, the limitations and critical reflection and what we could have change in this project to avoid repeating mistakes in the future.

Design

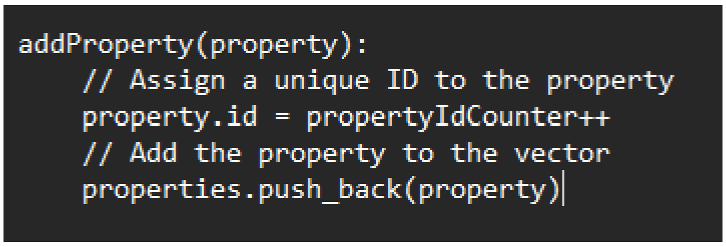
Selected Data Structures

This project would not be possible without the use of vectors as they are incredibly efficient important for storing properties and tenants. This is because vectors dynamically resize themselves if needed. This allows the system to accommodate several properties and tenants without reallocating a fixed amount of memory. Secondly, using vectors also provide efficient sequential access to elements which is optimal for iterating through properties and tenants when displaying it on the console or performing operations.

Algorithms

A. **Adding a property (‘addProperty’).** This algorithm is responsible for adding a new property to the management system. It assigns a unique ID to the property using the ‘propertyIdCounter’ variable then the property along with its assigned ID is added to the properties vector using the ‘push_back’ function.

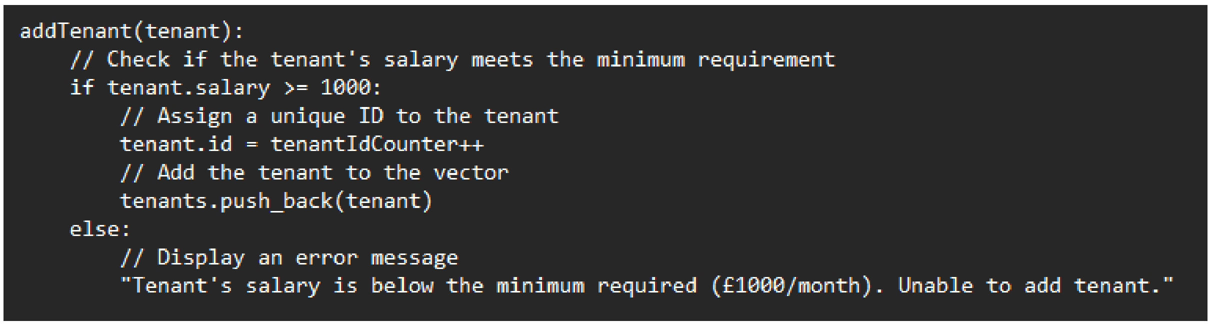
Pseudo Code:



```
addProperty(property):
    // Assign a unique ID to the property
    property.id = propertyIdCounter++
    // Add the property to the vector
    properties.push_back(property)
```

Algorithm Analysis:
Time Complexity: O (1) for appending to the end of the vector.
Space Complexity: O (1) for adding a single property.

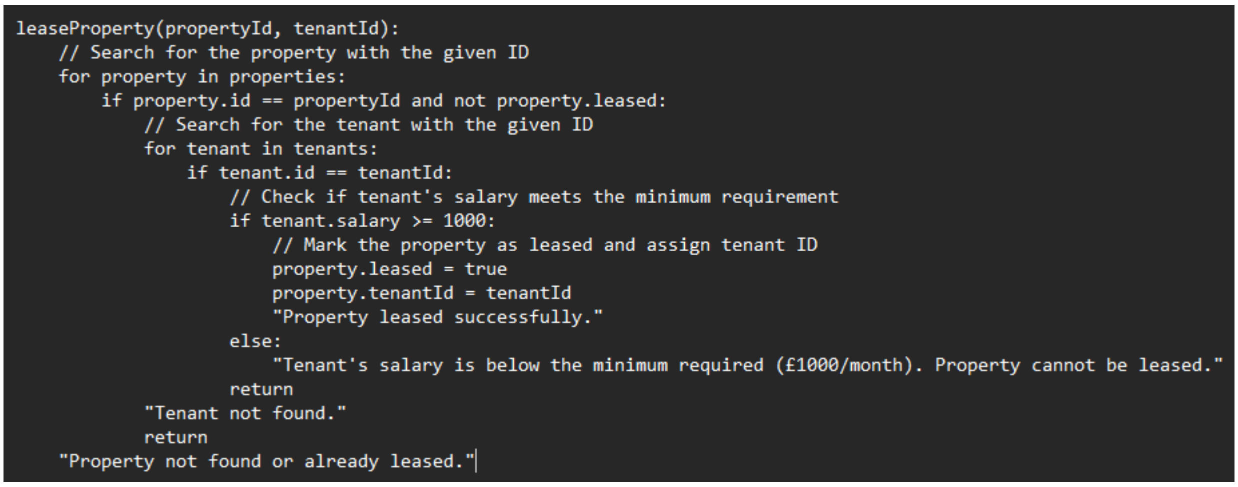
B. **Adding a tenant (‘addTenant’).** This algorithm is responsible for adding a new tenant to the system. Firstly, it checks if the tenant’s salary meets the minimum requirement(£1000/month). If sufficient, the algorithm proceeds. Like adding a property, it assigns a unique ID to the tenant using the ‘tenantIdCounter’ which increments for each new tenant.
Pseudo Code:



```
addTenant(tenant):
    // Check if the tenant's salary meets the minimum requirement
    if tenant.salary >= 1000:
        // Assign a unique ID to the tenant
        tenant.id = tenantIdCounter++
        // Add the tenant to the vector
        tenants.push_back(tenant)
    else:
        // Display an error message
        "Tenant's salary is below the minimum required (£1000/month). Unable to add tenant."
```

Algorithm Analysis:
Time Complexity: O (1) for appending to the end of the vector.
Space Complexity: O (1) for adding a single tenant.

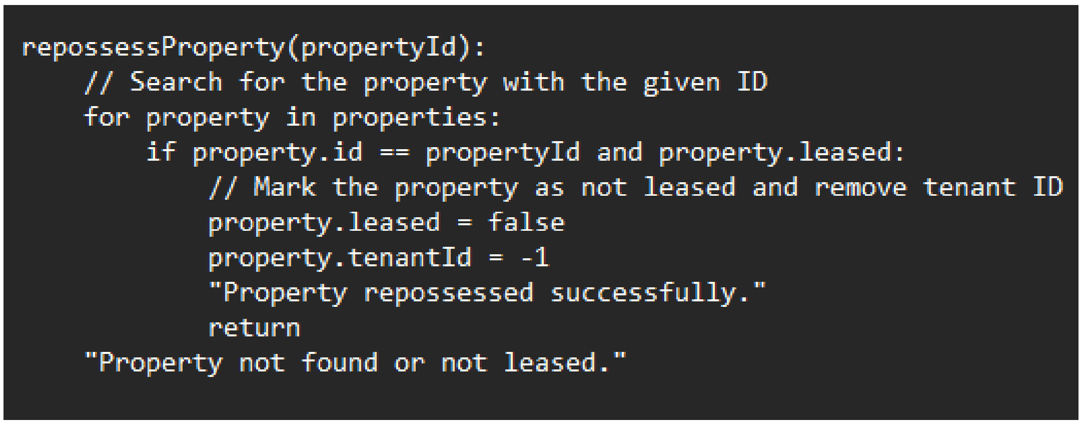
C. **Leasing a property to a tenant.** This algorithm handles the process of leasing a property to a tenant. It takes the IDs of the property and the tenant as input then it searches the property with the given ID in the ‘properties’ vector. If the property is found and not is not leased, then the algorithm continues. It then searches for the tenant with the given ID in the ‘tenants’ vector. If a tenant is found and their salary meets the requirements the algorithm proceeds to lease the property. Finally, the property is assigned as leased, and the tenant ID is assigned to it. If not successful, it will output an error message on the console as displayed in the image.
Pseudo Code:



```
leaseProperty(propertyId, tenantId):
    // Search for the property with the given ID
    for property in properties:
        if property.id == propertyId and not property.leased:
            // Search for the tenant with the given ID
            for tenant in tenants:
                if tenant.id == tenantId:
                    // Check if tenant's salary meets the minimum requirement
                    if tenant.salary >= 1000:
                        // Mark the property as leased and assign tenant ID
                        property.leased = true
                        property.tenantId = tenantId
                        "Property leased successfully."
                    else:
                        "Tenant's salary is below the minimum required (£1000/month). Property cannot be leased."
                    return
            "Tenant not found."
            return
    "Property not found or already leased."
```

Algorithm Analysis:
Time Complexity: O (n) in the worst case, where n is the number of properties, due to the need to search for the property by ID.
Space Complexity: O (1) as it operates on existing data structures.

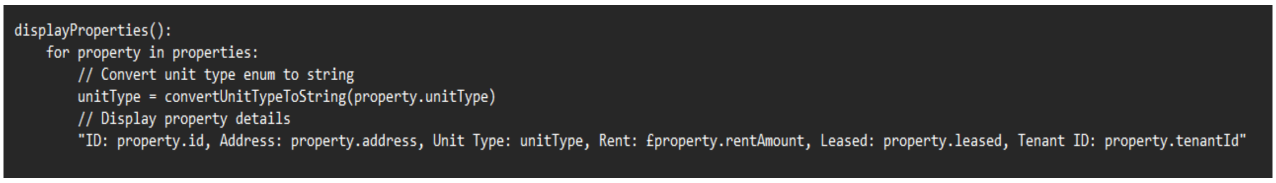
D. **Repossessing a property (‘repossessProperty’).** This algorithm handles the process of repossessing a property. It takes the ID of the property to be repossessed. Then, it searches for the property with the given ID in the ‘properties’ vector. If the property is found and is leased, the algorithm continues. After, the property is marked as not leased, and the tenant ID is set to -1 which indicated that’s its available to be leased again. Upon success, a message will show on the console.
Pseudo Code:



```
repossessProperty(propertyId):
    // Search for the property with the given ID
    for property in properties:
        if property.id == propertyId and property.leased:
            // Mark the property as not leased and remove tenant ID
            property.leased = false
            property.tenantId = -1
            "Property repossessed successfully."
            return
    "Property not found or not leased."
```

Algorithm Analysis:
Time Complexity: O(n) in the worst case, where n is the number of properties, due to the need to search for the property by ID.
Space Complexity: O (1) as it operates on existing data structures.

E. **Displaying Properties (‘displayProperties’).** This algorithm is responsible for displaying all properties stored in the project management system. It starts off my iterating through each property in the ‘properties’ vector. Then it converts the enum value to a string representation. Finally, it displays the ID, address, unit type, rent amount, lease status, and tenant ID (if leased) of each property.



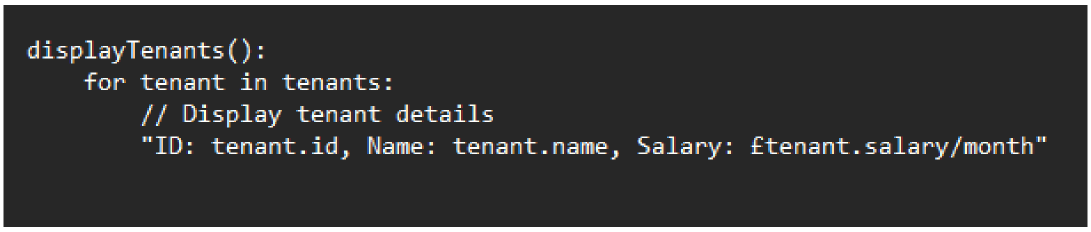
```
displayProperties():
    for property in properties:
        // Convert unit type enum to string
        unitType = convertUnitTypeToString(property.unitType)
        // Display property details
        "ID: property.id, Address: property.address, Unit Type: unitType, Rent: £property.rentAmount, Leased: property.leased, Tenant ID: property.tenantId"
```

Algorithm Analysis:

Time Complexity: O(n), where n is the number of properties, as it iterates through all properties to display their details.

Space Complexity: O (1) as it operates on existing data structures.

F. **Displaying Tenants(‘displayTenants’).** This algorithm is responsible for displaying all tenants stored in the system. Firstly, it iterates through each tenant in the ‘tenants’ vector. Finally, it displays the ID, name and salary of each tenant.



```
displayTenants():
    for tenant in tenants:
        // Display tenant details
        "ID: tenant.id, Name: tenant.name, Salary: £tenant.salary/month"
```

Algorithm Analysis:
Time Complexity: O(m), where m is the number of tenants, as it iterates through all tenants to display their details.
Space Complexity: O (1) as it operates on existing data structures.

Testing

Functionality	Test Case description	Input	Expected Output
void addProperty	Add a property to the system with valid details (property ID, address, unit type and rent amount	“4, Welwyn Garden City, 1, 2000”	The property will be successfully added to the system which can be seen when user chooses option 5 ‘Display Properties’ from the menu. Here it will show that the property ID of 4,

			the address, unit and rent amount has been added to the list of properties.
void addTenant	Add a tenant to the system with valid details (tenant ID, name, monthly salary)	“3, Jedd Madayag, 1400”	The tenant will be successfully added to the system which can be seen when user chooses option 6 ‘Display Tenants’ from the menu. From here, this tenant will be added on to the list of tenants
void leaseProperty	Lease a property to a tenant	“4, 3”	As we’ve added a property ID of 4, it will display the tenant ID for the property ID 4 and that it has been leased. This is possible when the user chooses option 5.
void displayProperties	Displays all added properties in the system	“5”	With the menu clearly stating that choice 5 is to ‘Display properties’, this will display any existing or recently added properties to the system.
void displayTenants	Displays all added tenants in the system	“6”	This will display any existing or recently added tenants to the system
void repossessProperty	Repossesses a property	“4”	As we’ve added the property ID of 4 and have leased it, this will be made possible to repossess. The message will display ‘Property

			repossessed successfully’.

For the testing approach, we tested the system with boundary values for input parameters such as property ID, tenant ID, rent amount and salary. Keeping this in mind, the system was tested to ensure if the system responded to the rent amount or salary of a specific tenant. For example, if the tenant’s salary is below £1000 then it would display a message, “Tenant’s salary is below the minimum required (£1000/month). Unable to add tenant.”

Conclusion

In conclusion, the report outlines the development of a property management system by NoName Systems, focusing on efficient management of properties and tenants in the real estate context. The use of vectors as the primary data structure is highlighted for storing properties and tenants due to their efficiency and dynamic resizing capability. The report discusses key algorithms, including adding properties and tenants, leasing properties to tenants, repossessing properties, and displaying property and tenant information. To demonstrate each algorithm's operation, pseudo code is included. Overall, the report gives a concise description of the functioning and architecture of the system with a focus on how well it manages tenants and properties. One limitation that arose was that there was duplicate IDs, and this occurred simply because the addProperty and addTenant functions only assigned IDs but did not check if the ID already existed. The current structure of the code risks data integrity issues. The lack of checks for unique IDs can lead to inconsistencies and potential errors. To improve reliability, it's essential to implement a mechanism to ensure that each property and tenant receives a truly unique identifier. The way the approach to a similar task in the future would be changed is to add more checks, specifically for the user input like rent amounts and salaries. This will allow for the user to have a better experience with the system as it guides the user, for example “Rent amount must be a positive number”. With the help of this, it ensures that the user provides the correct input, saving time and frustration.