COS 214 SOFTWARE MODELLING

Project Report

Kitchen of secrets

Ashely Kapaso
Ayanda Juqu
Chenoa Perkett
Eugene Mpande
Joshua Wereley
Thato Kalagobe

Logo Name

Kitchen of secrets

Executive summary

This project was initiated with the objective of developing a restaurant simulator. We deliberated on the choice between creating a pure simulation or a "restaurant tycoon" game and decided on the latter. A restaurant is a complex system with a multitude of interconnected processes, all working in harmony to produce and serve food to customers. These processes range from simple tasks like seating guests to intricate operations like food preparation at various workstations, coordinated by multiple chefs. Throughout the project implementation, we encountered numerous challenges that we successfully addressed by applying design patterns acquired during the course of the semester.

Background research

Brigade de Cuisine

In our quest to find a system in the restaurant industry to represent the kitchen in a video game we are creating, we came across the Brigade de Cuisine. The Brigade de Cuisine, also known as the kitchen hierarchy, is a hierarchical system found in restaurants and hotels employing extensive staff. This system was developed by Georges Auguste Escoffier. The Brigade de Cuisine ensures that kitchen operations run smoothly. The size and structure of the Brigade de Cuisine vary depending on the size and style of the restaurant.

Here are the key positions in the Brigade de Cuisine:

1. Executive Chef: Sits at the top of the kitchen hierarchy. Their role is primarily managerial. Executive chefs tend to manage kitchens at multiple outlets and are not usually directly responsible for cooking.

2. Chef de Cuisine (Head Chef): Focuses on managerial duties relating to the whole kitchen. For example, they supervise and manage staff, control

costs and make purchases, and liaise with the restaurant manager and suppliers to create new menus.

3. Sous Chef (Deputy Chef): Shares many of the same responsibilities as the head chef, but they are much more involved in the day-to-day operations in the kitchen. The sous chef also fills in for the head chef when they are not present.

4. Chef de Partie (Station Chef): This role is a vital part of the brigade system, but it's split into many different roles. There is more than one chef de partie, and each one is responsible for a different section of the kitchen.

5. Specific chef de partie roles include:
    - Sauté Chef/Saucier (Sauce Chef): Responsible for sautéing foods and creating sauces and gravies that accompany other dishes.
    - Boucher (Butcher Chef): Prepares meat and poultry before they are delivered to their respective stations.
    - Poissonnier (Fish Chef): Prepares fish and seafood.
    - Rotisseur (Roast Chef): Responsible for roast meats and appropriate sauces.
    - Friturier (Fry Chef): Prepares, and specializes in, fried food items.
    - Grillardin (Grill Chef): Prepares grilled foods.

Each position holds an important role in the overall function of the kitchen. Despite this, it's still important that you're aware of the kitchen hierarchy and know how your position operates within this.

Research Usage

Integration of Brigade de Cuisine Concepts

With the Brigade de Cuisine concept as our inspiration, we designed a system to represent the restaurant in our game. Instead of strictly adhering to the traditional Brigade de Cuisine hierarchy, we amalgamated the roles of the head chef and executive chef into one, referred to as the Head Chef in our system. While the core station chef roles remained unchanged, we introduced a few new additions, such as the baker and cook, alongside the grill and fry chef.

Our research played a pivotal role in breaking down the restaurant's functionality into distinct subsystems, each assigned to a team member for implementation. This approach ensured a well-structured and efficient development process, aligning with the principles of the Brigade de Cuisine's organized kitchen system.

Implementation

In the development of our game, we adopted a systematic approach by breaking down the project into well-defined subsystems, each entrusted to a dedicated team member. These subsystems played critical roles in shaping the overall functionality of the game. The key systems included:

- Kitchen System: This system governed the intricacies of food preparation at various workstations within the virtual restaurant, ensuring the culinary experience was realistic and engaging.

- Management System: The management system focused on overseeing the restaurant's day-to-day operations, including staff management, resource allocation, and overall efficiency.

- Reservation System: We implemented a reservation system to manage and schedule customer bookings, enhancing the realism and challenge of the game.

- Customer Care System: This system was designed to simulate customer interactions, their needs, and satisfaction levels, adding depth to the gameplay experience.

- Ordering System: The ordering system facilitated the seamless flow of customer orders from placement to fulfillment, integrating with the cooking system to drive the gameplay forward.

- Accounting System: To complete the restaurant management experience, we incorporated an accounting system to track financial transactions, expenses, and revenue, contributing to the game's economic aspects.

This modular approach to implementation allowed for efficient development, ensuring that each component of the game was managed effectively and contributed to the overall success of the project.

Development

In our pursuit of creating this project, we implemented a structured and collaborative development process to ensure the efficiency and quality of our work. Key aspects of our development approach included:

- Version Control with GitHub: We leveraged GitHub as our primary version control platform, fostering seamless collaboration among team members. This choice allowed us to effectively manage the project's codebase, track changes, and facilitate teamwork.

- Subsystem Branches: To mitigate merge conflicts and maintain code clarity, we adopted a strategy of using subsystem-specific branches. Each team member worked on their designated branch, allowing for independent development while reducing the risk of code conflicts.

- Code Consistency with C++ Linter: We enforced coding standards and consistency by integrating a C++ linter into our development workflow. This tool ensured that the code adhered to predefined coding guidelines, enhancing readability and maintainability.

- Automated Testing with Google Tests: To guarantee the reliability and functionality of our code, we integrated Google Tests into our development process. These tests were automatically triggered on every pull request and push to the repository, verifying that only working and validated code was merged into the remote repository.

- GitHub Actions for Linting: We harnessed the power of GitHub Actions to automate the linting process. By integrating linter checks into our workflow, we maintained code quality and consistency with each code change, minimizing the chances of coding errors.

Our development approach emphasized collaboration, code quality, and automated testing to ensure that the project codebase remained robust and efficient throughout the development lifecycle.

Subsystems

Kitchen

Purpose:

The kitchen is the hub of activity where the entire order fulfillment process takes place. It plays a crucial role in ensuring smooth operations in a restaurant. Here's a breakdown of the kitchen's purpose and workflow:

1. An order is initiated by a waiter and sent to the kitchen.
2. Orders are handled as they come in and assigned to different chefs based on the type of meal.
3. Various chefs work collaboratively on an order, making different components of the order.
4. The order may move between different chefs for various preparations before reaching the head chef.
5. The deputy head chef ensures the final assembly and quality check of the order.
6. Once the order is complete, the kitchen notifies the waiter for order pickup.
7. The head chef is in charge on doing customer service rounds when a customer gets angry and leaves.

System Components

Key components and modules of the Kitchen system:

1. Order
2. Meal
3. Chef
4. Kitchen

Challenges Faced

During the implementation of the kitchen system, several challenges emerged that required innovative solutions. The primary issues revolved around three key areas:

1. Ingredient Calculation: We needed a mechanism to calculate the required ingredients for each order to manage inventory effectively.

2. Order Distribution: Ensuring that the order object was efficiently passed to all the chefs responsible for its preparation.

3. Order Notification: Determining how to notify the waiter promptly when an order was finished or canceled.

Solutions

To tackle these challenges, we applied the following design patterns:

1. Composite Pattern
    - Use: The Composite Pattern is employed to calculate the total price of an order and the necessary ingredients. In this pattern, the "Order" class acts as the composite, providing methods like "calculatePrice" and "calculateIngredients" to facilitate these calculations. This approach simplifies the management of complex meal orders.

2. Chain of Responsibility Pattern
    - Use: The Chain of Responsibility Pattern streamlines the meal preparation process. Each chef is responsible for a specific part of an order. When an order is received, it starts with the Head Chef, who handles administrative duties and then delegates tasks to the appropriate chef. This process continues until the order is ready for serving. For instance, if an order includes a combo meal with a burger, chips, and a drink, the Head Chef manages administration, burger preparation, and drink preparation before passing it to the fry chef for the chips. Finally, it returns to the Head Chef for the final serving, ensuring a smooth and organized kitchen workflow.
3. Observer Pattern:
    - Use: To manage order notifications, we employed the Observer pattern. Management was registered as the observers, and chefs, upon completing an order or canceling it, triggered notifications. This pattern allowed for real-time communication between chefs and management, ensuring timely updates on the status of orders.

Accounting

Purpose:

The accounting system plays a pivotal role in overseeing various accounting-related functions, ensuring the following key objectives are met:

1. Efficient tab management.
2. Diverse billing options for customers.
3. Accurate inventory tracking.

System Components:

The accounting system comprises three distinct sub-systems, each with its specific functions:

1. Tab System:
   - Responsible for the meticulous management of tabs, ensuring they remain current and up-to-date.

2. Billing System:
   - Tasked with invoicing customers using a range of payment methods, providing flexibility and convenience.

3. Inventory System:
   - Focused on monitoring and maintaining an accurate inventory record, guaranteeing the availability of stock.

Challenges Faced:

One of the primary challenges encountered was devising a solution for handling various payment types and managing tabbed payments effectively.

Solutions:

To overcome these challenges, the accounting system incorporates the following design patterns:

1. State Pattern:
   - Applied within the Tab System.
   - Employs three distinct states for tabs: open, closed, and overdue.
   - Tailors tab behavior based on their respective states, optimizing management and customer service.

2. Strategy Pattern:
   - Utilized in the Billing System.
   - Facilitates flexible billing by offering various payment methods, including card, cash, and multiple payment options.
   - Enables streamlined billing for multiple clients sharing a single table with different payment preferences.

Management

Purpose:

The management system's purpose is to employ a strategic approach to oversee interactions among the different game components; ensuring that each component's requests are harmoniously coordinated.

System Components

Game Engine System: This subsystem serves as the focal point for managing interactions among a group of objects; encapsulating the way they communicate with each other.
Player Interaction System: This subsystem is responsible for encapsulating requests into objects, providing a means for parameterized clients to generate requests, manage request queues, and facilitate the reversal of changes.

Challenges Faced and Solutions

The management system will use two design patterns; one for each subsystem:

Mediator Pattern:

- Objective: To define an object that encapsulates interactions among a group of objects, fostering loose coupling by avoiding direct references between objects and enabling independent variations in their interactions.
- Rationale for Use: The Mediator Pattern facilitates interaction between game components by notifying each component of changes and how these changes may impact other components.
- Illustrative Use Case: For instance, the Accounting System could communicate with the Cooking System to check if there are sufficient ingredients for a particular dish.

Facade Pattern:

- Objective: To provide a unified interface to a set of interfaces in a subsystem. It defines a higher-level interface that makes the subsystem easier to use.
- Rationale for Use: The Facade Pattern simplifies the complex interactions and operations within the Player Interaction System, providing a single, simplified interface for clients to interact with the subsystem.
- Illustrative Use Case: When a player interacts with the game, they use a single, easy-to-use interface provided by the Facade to make requests, manage queues, and reverse changes without needing to understand the internal complexities.

Ordering

Purpose:

The purpose of my system is to

1. Facilitate the Customer and Waiter interaction
2. Assemble Orders to send to the Kitchen (via manager)
3. Generate MenuItems tha will be used in the menu

System Components

1. Waiter-Customer Interaction
2. "Build-Your-Own-System"

Problem:

We wanted to simulate the "Build-Your-Own" structure efficiently allowing customers to add various available meals to their order without having to send each meal to the kitchen.

Solution:

The Builder pattern allows customers to compose complex Order objects efficiently by adding various meals, beverages, and customizations step by step. It abstracts the order-building process, making it easy for customers to create customized orders without having to send each meal individually to the kitchen.

This results in a more organized and streamlined ordering process while providing flexibility and customization options to customers.

The Factory pattern allows you to add new types of menu items without modifying existing code. You can introduce new menu items and extend the factory to create these items, making the system more flexible and extensible.

Reservation

System Purpose:

The system's primary objective is to facilitate the efficient management of customer reservations and the allocation of tables across various sections of the restaurant's kitchen floor.

System Components:

1. Table
2. Section
3. Host
4. Receptionist
5. Reservation

Challenges Faced:

The main challenge encountered during the development of the system was finding an effective method to handle the creation of multiple reservations for each customer visiting the restaurant.

Solutions:

To address this challenge, a strategic approach was adopted. The "Factory Method" design pattern was implemented within the Receptionist component, serving as a user-friendly interface for the creation of Reservation instances for each customer. This design pattern offers a flexible and organized way to generate multiple reservations, ensuring smooth customer management within the restaurant.

Customer Care

Purpose:

The Customer Care system is designed to ensure utmost customer satisfaction within the restaurant simulator. It encompasses the management of customer experiences, rounds, and table visits by the chefs, waiters, and managers, all aimed at delivering exceptional service and making customers happy.

System Components:

1. Satisfaction State
2. Customer

Challenges Faced:

The system faced challenges related to efficiently calculating bills based on customer mood and effectively communicating state changes to the user without resorting to complex and error-prone conditional statements.

Solutions:

To overcome these challenges, the Customer Care system leverages the following design patterns:

1. State Pattern:

- Utilize the State pattern to model various states of customer satisfaction. Customer satisfaction can transition between states like "happy," "neutral," and "unhappy" based on their interactions with the restaurant staff and the service they receive. This allows for dynamic responses to customer moods without cumbersome if statements.

2. Observer Pattern:

- Implement the Observer pattern to manage customer satisfaction. Customers serve as subjects, while restaurant staff (waiters, chefs, and managers) act as observers. Staff members can notify the customer about service improvements, such as addressing complaints or providing swift service. These notifications can influence the customer's satisfaction state and provide a feedback mechanism for enhancing the customer experience.

Patterns

Factory
Chain of responsibility
Mediator
Composite
Prototype
State
Strategy
Façade
Builder
Observer