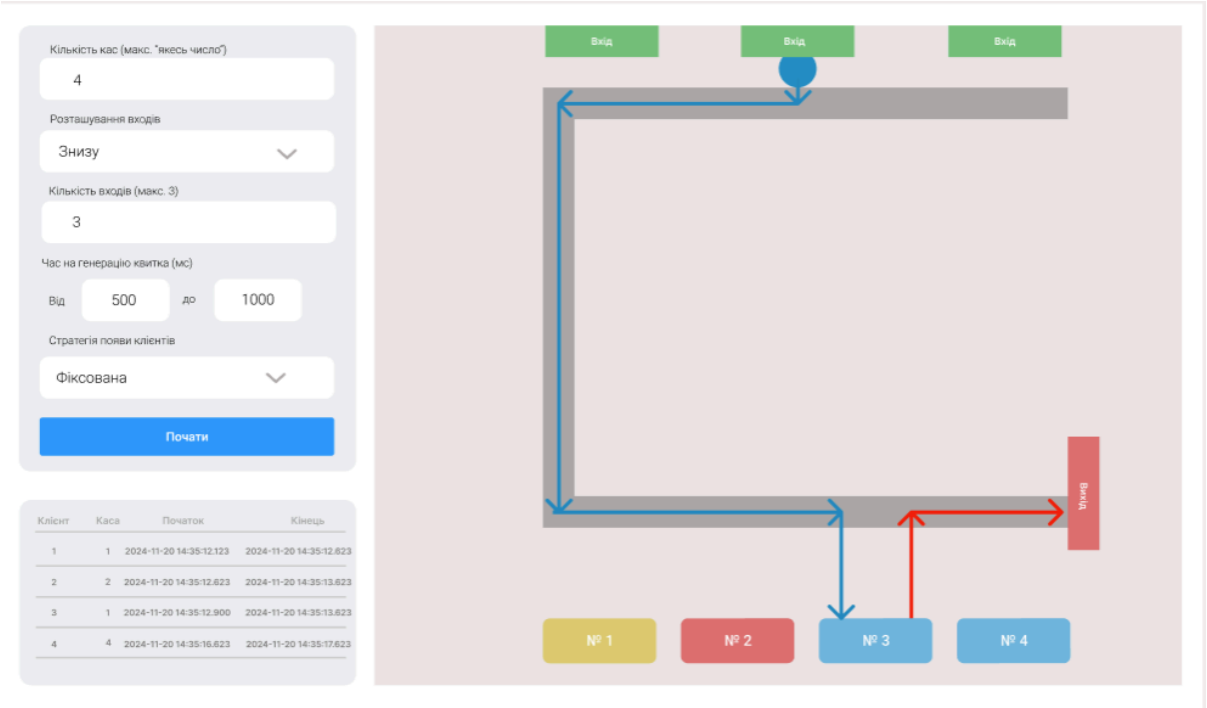
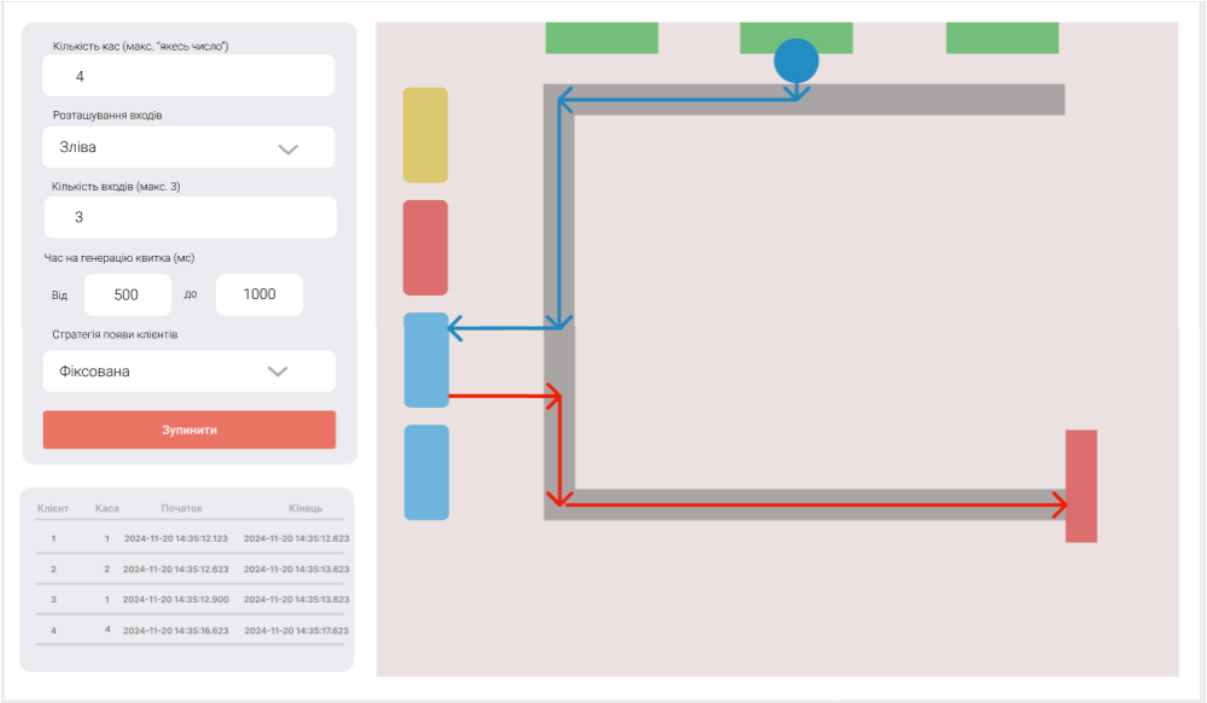
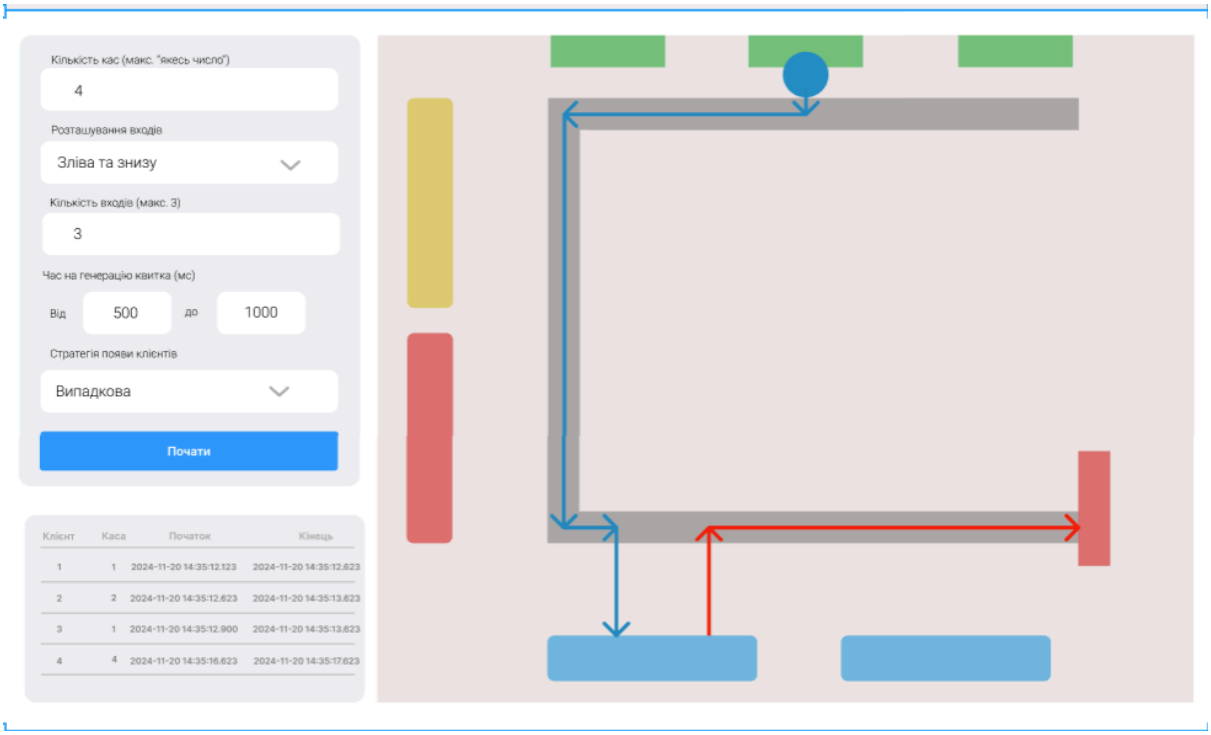
Проект: симулятор роботи залізничних кас
Команда:

Team Lead	Жила Володимир
Architectures	Вихованець Костянтин & Кафльовський Роман
Designers	Герус Тетяна & Галімурка Анастасія
Frontend Devs	Гудзенко Денис & Марчук Роман
Backend Devs	Лев Володимир & Гусар Артур
QAs	Шмигелюк Олег & Шмигелюк Орест

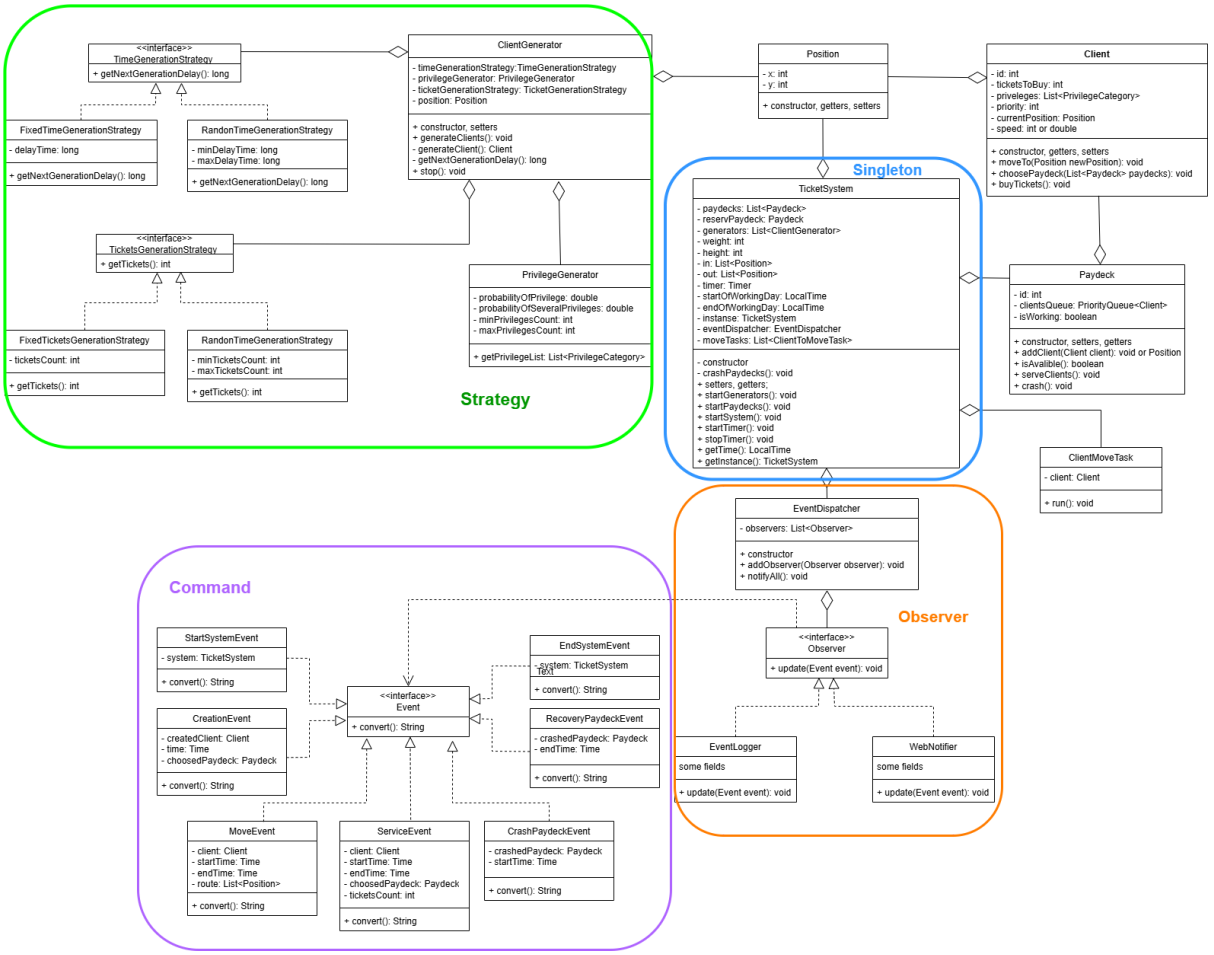
Дизайн:





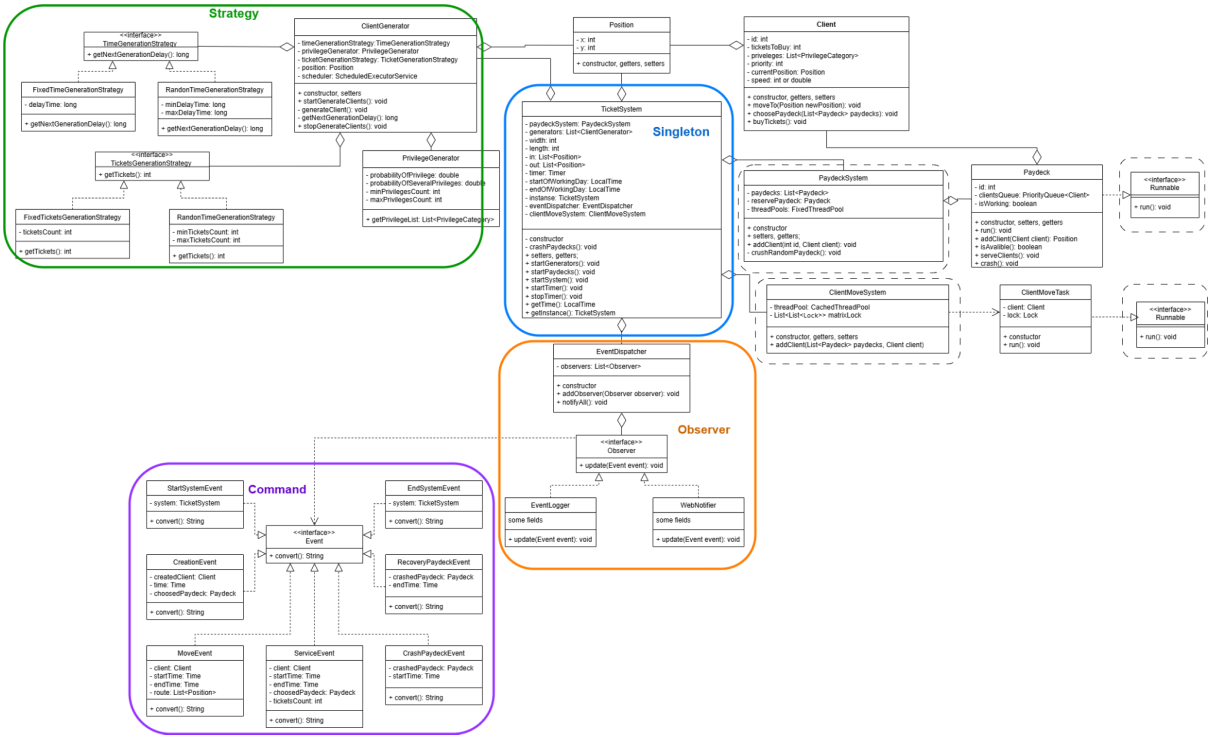


Початкова діаграма класів:



Фінальна (майже) діаграма:

Зміни виділені пунктирем.



Антипатерн:

В нашому випадку клас TicketSystem є God Object, оскільки він зосереджує в собі забагато відповідальностей та функціональності: управління квитками (paydecks), управління чергами та позиціями клієнтів (in/out positions), контроль часу роботи (timer, startOfWorkingDay, endOfWorkingDay), управління генераторами клієнтів (generators), обробку подій (eventDispatcher) і навіть рух клієнтів (moveTasks). Такий дизайн порушує принцип SOLID єдиної відповідальності - клас намагається контролювати всі аспекти системи.

Патерни:

Стратегія:

Strategy застосовується для гнучкого управління генерацією часу обслуговування клієнтів та кількості квитків через класи TimeGenerationStrategy та TicketsGenerationStrategy. Це дозволяє легко змінювати алгоритми генерації (фіксований чи випадковий час очікування, різна кількість квитків) без модифікації основного коду

системи, що особливо корисно для симуляції різних сценаріїв роботи

Спостерігач:

Observer використовується для відстеження та реагування на різні події в системі (StartSystemEvent, EndSystemEvent, CreationEvent, MoveEvent, ServiceEvent, CrashPaydeckEvent) через EventDispatcher. Його головна перевага полягає в тому, що він забезпечує слабку зв'язність між компонентами системи - нові спостерігачі (як EventLogger для логування та WebNotifier для веб-сповіщень) можуть бути додані без змін в основному коді системи. Це дозволяє гнучко розширювати функціональність моніторингу роботи кас, наприклад, додавати нові способи сповіщень про події чи збір статистики, при цьому не змінюючи логіку роботи самої системи продажу квитків.

Одинак:

Singleton використовується для класу TicketSystem, оскільки система залізничних кас повинна бути єдиною в межах всього додатку, а це гарантує, що всі каси працюють з однією й тією ж системою квитків, спільними даними про наявність місць та розкладом, запобігаючи можливим конфліктам при продажу квитків різними касами.

Команда:

Кожен компонент, такий як StartSystemEvent, EndSystemEvent, CreationEvent, MoveEvent, ServiceEvent та CrashPaydeskEvent, інкапсулює конкретну команду або дію, яку він виконує. Це дозволяє в майбутньому розширювати функціональність симуляції, додаючи нові типи подій чи операцій без необхідності перебудовувати всю систему.Паттерн Команда робить систему більш гнучкою та розширюваною. Перевага - нові команди можуть бути легко додані без необхідності змінювати існуючий код.

Кінцева діаграма:

Додано:

PaydeckSystem

Інкапсулює логіку, пов'язану з роботою платіжних терміналів, таку як: управління чергою клієнтів, перевірка доступності платіжного каси, обслуговування клієнтів та їх обробка

ClientMoveSystem
Відповідає за керування переміщенням клієнтів у симуляції. Забезпечує логіку, пов'язану з черговістю клієнтів, їх розподілом між касами та синхронізацією цих процесів.

<Interface> Runnable
Paydesk реалізований як ранбл для того, щоб обслуговувати клієнтів у окремих потоках, не блокуючи інші компоненти, адже в системі передбачено використання багатопоточності.

ClientMoveTask реалізований як Runnable для того, щоб забезпечити більш еефективну координацію та синхронізацію руху клієнтів. Цей ранбл відповідає за переміщення клієнтів між касами.