PROIEKT - PIZZERIA

1. Imiona i nazwiska autorów

Paweł Wysocki 325248
Jabłoński Piotr 325163

2. Założenia

- Każdy klient zamawia przystawkę, pizzę oraz napój
- Zamówienia klientów są przypisywane do stolika, przy którym siedzą, wspólny rachunek
- Pizze różnią się rozmiarem, rodzajem, ceną i czasem przygotowania
- Wszystkie produkty jednej kategorii (np. pizze) do jednego stolika są wydawane jednocześnie (np. jedna pizza została przygotowana szybciej, nie zostanie wydana dopóki pozostałe pizze do danego stolika nie zostaną zrobione)

3. Hierarchia klas

Simulation:

- *size_t* time
- *const* ***Menu*** &menu
- unsigned seed
- *size_t* client_index
- *size_t* group_index
- *size_t* table_inde
-
- *size_t* staff_index
- *size_t* total_earned
- *ofstream* logs
- *vector*<***Table***> active_tables
- *vector*<***Table***> all_tables
- *vector*<***Waiter***> waiters
- *vector*<*enum class Event*> event_history
- *vector*<*enum class Event*> next_events
- *enum class Event* current_event

Food:

- *bool* ready
- *unsigned short* preparation_time
- *unsigned int* base_price
- *string* name

  Pochodne:
  - **Appetizer**
  - **Pizza:**
    - enum class Size
  - **Drink:**
    - enum class Volume

Human:

- *unsigned int* id

  Pochodne:

  - **Client**
  - **Waiter:**
    - *bool* occupied
    - *vector<unsigned int>* assigned_tables

FoodList:

- *vector<**Pizza**>* pizzas
- *vector<**Drink**>* drinks
- *vector<**Appetizer**>* appetizers

  Pochodne:

  - **Menu**
  - **Order:**
    - *bool* drinks_ready_to_serve
    - *bool* appetizers_ready_to_save
    - *bool* pizzas_ready_to_save

Table:

- *bool* ready
- *unsigned int* table_id
- *unsigned int* earned
- *enum class TableSize* size
- *enum class Status* status
- ***Group*** group
- ***Order*** order
- *const **Menu*** &menu

Group:

- *unsigned int* group_id
- *unsigned int* group_size
- *bool* group_complete
- *vector<Client>* clients
- *vector<unsigned int>* awaiting_ids

Read:

- ***Menu*** menu

RandomNumber:

- *static mt19937* engine

4. Podział obowiązków

Paweł Wysocki - przeważnie obiekty
Piotr Jabłoński - przeważnie symulacja

5. Opis działania symulacji i jej uruchomienia

A) Opis działania Symulacji:

1. Szkielet symulacji

Symulacja ma prostą strukturę:

- Wylosuj nowe wydarzenie (od tej pory również nazywane **Event**)
- Wykonaj odpowiednie instrukcje zależne od tego wydarzenia
- Powtórz dopóki aktualna tura będzie równa turze zadanej

Event to *scoped enum*, którego warianty są losowane wagowo przez funkcję losującą. (**update_event)**
Nie jest ona natomiast w pełni losowa. Domyślnie rozpiska procentowa wygląda tak:

- 55% na wysłanie kelnerów do stolików (**ModTable**)
- 40% na przyjście nowych klientów (**NewClients)**
- 3% na to że nowo przybyli klienci wyjdą (**ClientsExit**)
- 2% na wypadek w kuchni (**KitchenAccident**)

Funkcja ta ma za zadanie upewnić się żeby przebieg wydarzeń w proi_pizzerii był zbliżony do funkcjonalności w prawdziwej pizzerii.

Jeżeli wszystkie stoliki są zajęte, *update_event* będzie zwracał tylko *ModTable (96%)* lub *KitchenAccident* (4%),

Jeżeli wszystkie stoliki są puste, funkcja gwarantuje żeby grupka ludzi skierowała się do środka proi_pizzerii,

Można również manipulować *update_event* dodając *event* do wektora *next_events*, który funkcja losująca priorytetyzuje. W ten sposób losowany jest *NewClients* na samym początku uruchomienia programu.

2. Handle_event

Handle_event to jest funkcja, która wywołuje funkcje pomocnicze zależnie od aktualnego wydarzenia. Jest ich 4 - tyle ile eventów:

- handle_mod_table() - większość logiki znajduje się w tej funkcji. Sprawdza ona czy kelner przypisany do stolika jest wolny, czy grupa jest kompletna lub gotowa do zamówienia oraz czy już zapłaciła i jest gotowa do wyjścia (W ostatnim przypadku dodane do *next_events* jest *ClientsExit*
- handle_new_table() - z wektora *all_tables* przepisana jest pierwsza pozycja i dodana do *active_tables*. Takie działanie jest konieczne, aby po zwolnieniu się stołu, można było przypisać ten stół do kolejnej grupy. Przypisuje ona również kelnerów.
- handle_clients_exit() - spontaniczna sytuacja w której klienci wychodzą z pizzerii podczas czekania na menu
- handle_kit_acc() - najprostsza ze wszystkich funkcji pomocniczych, symuluje problemu w kuchni, co oznacza że w tej turze pizzeria znajduje się w stanie chaosu i ruch stoi w miejscu

B) Pliki wejściowe:

Plik wejściowy powinien zawierać dane dotyczące pozycji w menu w następującym formacie:

- przystawka: APPETIZER,{nazwa przystawki},{cena},{czas przygotowania}
- napój: DRINK,{nazwa napoju},{cena}

*przyjęte jest, że czas przygotowania napojów jest równy 1 jednostkę czasu.

- pizza: PIZZA,{nazwa pizzy},{cena},{czas przygotowania}

W przypadku napojów i pizz zostaną automatycznie wygenerowane obiekty w każdym możliwym rozmiarze.

C) Format argumentów wejściowych:

./{nazwa pliku wykonawczego} {nazwa pliku źródłowego z zawartością menu} {liczba jednostek czasu - ile ma trwać symulacja} {liczba małych stołów} {liczba średnich stołów} {liczba dużych stołów} {liczba kelnerów}

Przykładowo dla ./proi_23L_201_pizzeria menu.txt 100 1 2 3 5:

- menu pobierane z pliku “menu.txt”
- 100 tur
- 1 mały stolik (dwuosobowy)
- 2 standardowe stoły (czteroosobowe)
- 3 duże stoły (ośmioosobowe)
- 5 kelnerów

6. Wykorzystane elementy biblioteki STL

W projekcie wykorzystano:

- std::array - do przechowywania kolekcji o statycznym rozmiarze w sposób elegancki
- std::pair - do ujednolicenia dwóch typów jako jednego - użyte razem z std::array żeby efektywnie dane dwuczłonowe
- std::vector - do przechowywania kolekcji obiektów o nieokreślonym rozmiarze
- std::vector::iterator - do wykonywania operacji na wektorach i ich elementach
- std::find - do znajdowania konkretnych elementów w wektorach

7. Opis zidentyfikowanych sytuacji wyjątkowych i ich obsługi

Rozpoznawane sytuacje wyjątkowe:
- podano zbyt mało argumentów
- nie znaleziono pliku pod podaną ścieżką
- brakuje danych w pliku źródłowym (linijka niekompletna)
- podanie 0 lub wartości niebędącej liczbą dla argumentów liczbowych

W przypadku wystąpienia któregokolwiek z tych błędów program poinformuje o tym użytkownika i zakończy działanie.

8. Opis przeprowadzonych testów

Funkcjonalność każdego z obiektów wykorzystywanych w projekcie została dokładnie sprawdzona za pomocą biblioteki catch2. Przetestowano 11 obiektów, przeprowadzając łącznie 441 asercji.