1.목표

order, food_truck, food_zone, start_simulation의 코드를 완성하여 푸드 트럭 시뮬레이터가 실행되게 함

2.과제를 해결하는 방법

메인 클래스에서 사용될 order, food_truck, food_zone를 완성하여 주문 개수, 가격, 요리 시간, 푸드 트럭, 손님 큐 등 정보들을 저장할 수 있게 한다.

3.과제를 해결한 방법

각각의 클래스를 완성할 때 해당 클래스의 테스트 클래스를 보면서 작성하였다.

1) order.py

(1) __init__(self, order_time)

test 클래스의 init에서 timestamp와 ordertime이 들어간 것을 보고 초기 값으로 order_time과 order_time을 설정하였다. order_time은 지역 변수를 통해 입력하도록 하였다. 또한 1 이상 5 이하인 qty를 보고 randomint를 통해 1부터 5사이의 값을 저장하는 qty를 생성하였다.

(2) get_order_time(self)

__init__에서 저장된 order_time을 반환하도록 하였다.

(3) get_qty(self)

__init__에서 저장된 qty를 반환하도록 하였다.

(4) wait_time(self, current_time)

현재 시간을 입력받아 현재 시간에서 order_time을 뺀 시간을 반환하도록 하였다.

2) food_truck.py

(1) __init__(self, price, making_time)

test 클래스의 init에서 FoodTruck에 price와 making_time을 입력한 것을 보고 지역 변수로 price와 making_time을 입력받도록 하였다. 또한, current_order와 time_remaining을 None과 0으로 비교하는 것을 보고 각각의 초기값을 None과 0으로 설정하였다.

(2) tick(self)

9부터 -1까지 내려가면서 남아있는 주문이 있다면 남은 시간을 비교하는 것을 보고 주문이 있다면 남은 시간을 -1씩 하면서 0이 될 때까지 비교하고 0 밑으로 가면 남은 주문이 없다는 것을 나타내기 위해 None이라 표시하였다.

(3) is_busy(self)

주문이 없고, 남은 시간도 0이라면 작업 중이지 않은 상태라 판단하여 False라 반환하고, 나

머지 상황에 대해서는 작업 상태라 판단하여 True를 반환하였다.

(4) start_next_food(self, next_order)
next_order를 통해 다음 주문을 받아서 current_order에 저장하고, time_remaining에 makint_time과 다음 주문의 수량을 곱하여 저장한다.

3) food_zone.py
(1) __init__(self, order_probability)
test코드에서 주문 가능성인 order_probability를 입력받는 것을 보고 지역 변수로 넣었다. 또한 assertEqual에서 food_trucks, order_queue, waiting_times, income을 각각 set, list와 비교하는 것을 보고 각각을 set과 list로 초기화했다. 마지막으로 self의 order_probability에 지역 변수 order_probability를 저장했다.

(2) add_truck(self, var_truck)
set인 food_trucks에 set함수인 add를 이용해서 지역 변수로 받은 var_truck set에 추가한다.

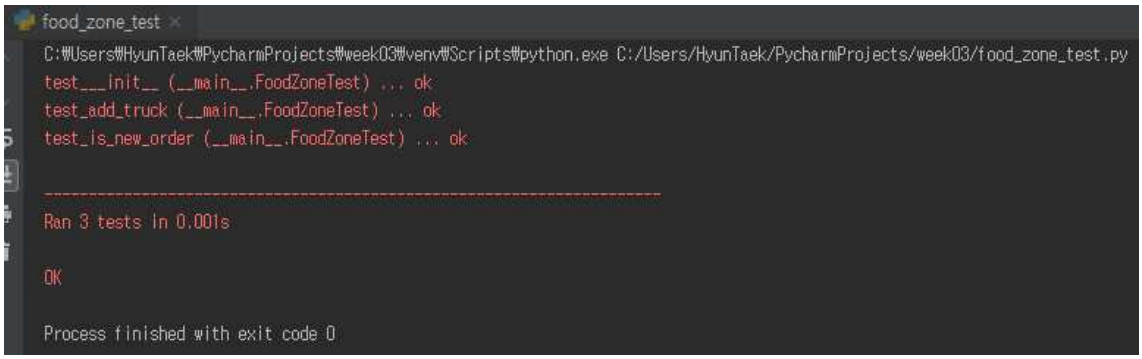
(3) is_new_order(self, picked_num = 0)
0으로 초기화 된 지역 변수 picked_num을 입력하고 1부터 지역 변수인 order_probability까지의 정수 중 무작위로 한 숫자를 뽑아서 picked_num에 저장한다. 저장한 숫자가 order_probability와 같다면 True를 반환하고 그렇지 않다면 False를 반환한다.

4) start_simulation(zone, test_second)
printer_simulator와 유사하다 생각하여 비교하면서 하였다. 우선 for 반복문을 통하여 입력된 시간 내에서 무작위로 주문을 추가하여 푸드 존의 주문 리스트인 order_queue에 추가한다. 그 후 다시 for 반복문을 통하여 푸드 존에 속한 푸드 트럭 내에서 일하는 상태가 아니고, 주문이 남아있을 때, 현재 주문 중 첫 번째 주문을 pop을 통해 꺼내어 new_order에 저장한다. 또한 새 주문의 준비 시간만큼 푸드 존의 대기 시간에 추가한다. 다음으로 start_next_food를 통하여 새 주문을 시작한다. 그리고 새 주문의 가격과 양을 각각 price와 qty에 저장하고 두 개를 곱한 값을 푸드 존의 수입에 추가한다. 마지막으로 tick을 이용하여 푸드 트럭에서 작업을 시작한다.

4.결과화면
1) order_test

2) food_truck_test

3) food_zone_test

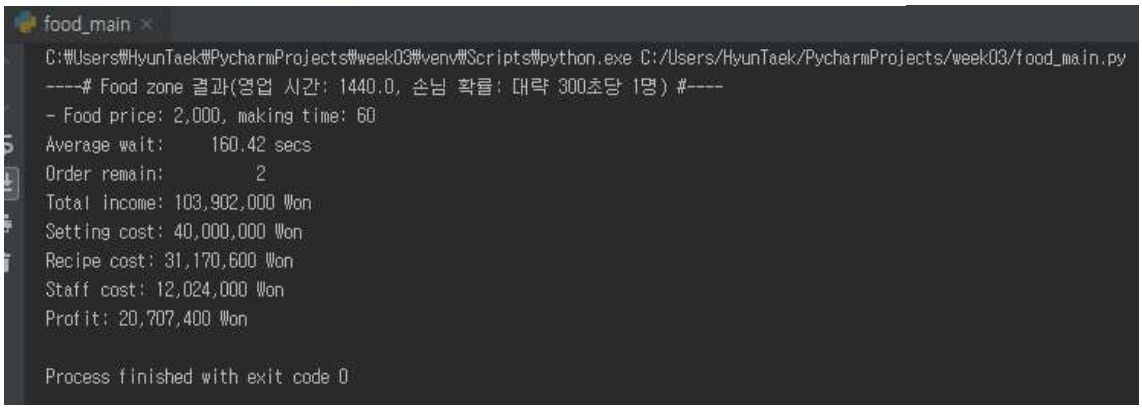


4) food_main

Task class는 3개의 멤버 함수(메소드)를 가지고 있으며 2개의 멤버 변수(필드)를 생성한다. 멤버 변수 중 pages는 1-21사이 무작위 정수로 초기화된다. 멤버 함수 중 `wait_time()`을 통해 Task가 얼마나 오래 기다렸는지 계산할 수 있다.

2.2 Printer Class

Printer class는 주어진 Task를 수행한다. Task를 수행할 때 분당 처리할 수 있는 양에 따라서 시간이 필요하다. 이 기능을 위하여 시간을 보내는 방법과 새로운 업무를 시작하는 방법을 구현해야 한다.



2 Printer simulator

Queue 활용 예제로 사무실에 설치된 'Printer simulator'를 구현한다. Printer class와 Task class를 구현하고 이를 활용하여 simulator를 작성한다. Printer는 한번에 하나의 출력만 할 수 있기 때문에 출력이 이루어지는 동안에 다른 Task는 대기하여야 한다. 수행 결과로 Task가 평균 얼마나 대기하였는지 출력하고, 지정된 시간동안 마치지 못 한 Task가 몇 개인지도 출력한다.

2.1 Task Class

Task class는 Printer가 수행해야할 업무 정보를 담고 있다. 출력해야할 Page 수와 Task가 생성된 시간을 가지고 있다. 모든 Page가 출력되면 Task가 생성된 시간과 현재 시간을 비교하여 수행에 걸린 시간을 계산할 수 있을 것이다.

Printer class는 3개의 멤버 함수(메소드)와 3개의 멤버 변수(필드)를 가지고 있다. 이중 `tick()`은 수행하는 업무에 시간을 투자하는 것으로 이해할 수 있다. 예를 들어 수행해야할 업무가 100이라는 시간이 필요할 때 `start_next()`를 통해 Printer는 업무를 시작하게 된다. 이후 `tick()`을 통하여 100이라는 시간이 1씩 줄어들게 되며 0이 되면 업무가 끝나는 것으로 본다. Simulator는 이 2가지 클래스를 이용하여 구현할 수 있다.

2.3 Simulator Function

시뮬레이션 프로그램 제작을 위한 클래스가 모두 준비되었다. 이제 Simulator를 구현할 차례다. Simulator에서는 '언제' 새로운 Task가 생성될지 무작위로 계산하여야 한다. 새로운 Task를 생성한 후 Queue에 넣어 Printer에게 할당할 준비를 한다. Printer가 아무런 업무를 진행하고 있지 않을 때 Simulator는 Queue에서 Task를 하나 꺼내 업무를 진행하도록 지시한다.

Setting cost: 40,000,000 Won
Recipe cost: 25,914,600 Won
Staff cost: 12,024,000 Won
Profit: 8,443,400 Won

예를 들어 2000원짜리 음식을 만드는 시간을 120초에서 60초로 단축시키면 손님들이 거의 기다리지 않는 것을 알 수 있다. Queue를 활용해 시뮬레이터를 구현하여 사용하면 이와 같은 것을 예측할 수 있다.

Listing 10: Food Truck Simulator Better Result

----# Food zone 결과(영업 시간 : 1440.0, 손님 확률 : 대략 300 초당 1명) #----
– Food price: 2,000, making time: 60
Average wait: 164.91 secs
Order remain: 0
Total income: 103,168,000 Won
Setting cost: 40,000,000 Won
Recipe cost: 30,950,400 Won
Staff cost: 12,024,000 Won
Profit: 20,193,600 Won

과제를 완료한 후 실행시키면 다음과 같은 출력이 나온다. 2,000원짜리 음식을 팔 때 개당 120초가 걸리면 300초마다 손님이 1명씩 온다고 가정했을 때 지나치게 대기 시간이 길어지게 된다(공급이 수요를 못 따라간다.). 이를 어떻게 수정해야하는지 실험해보자. 시뮬레이터는 미리 실험해보고 우리가 어떻게 행동해야하는지 알 수 있도록 한다.

Listing 9: Food Truck Simulator Result

----# Food zone 결과(영업 시간 : 1440.0, 손님 확률 : 대략 300 초당 1명) #----
– Food price: 2,000, making time: 120
Average wait: 444837.50 secs
Order remain: 2767
Total income: 86,382,000 Won