자판기 프로젝트 최종 보고서

파이썬프로그래밍

2025.06.16

게임공학과

2022184015

김해님

1. 프로젝트 접근 방식 변경점

실물 자판기의 구조와 동작 방식을 최대한 반영하기 위해 우리 학교 자판기와 유사한 모델을 인터넷에서 조사했다. 학교 자판기와 유사한 실제 자판기의 최대 재고량은 약 500개 내외였으나, 코드의 구조와 계산의 편의를 위해 전체 최대 재고는 30종류 × 20개 = 600개로 설정했다. 각 음료는 최대 20개까지 재고를 가질 수 있도록 구현했다.

* GUI

GUI는 Python의 Tkinter와 Pillow(PIL) 라이브러리를 사용하여 구현했다. 메인 윈도우는 1024x720 크기로 고정되어 있으며, ttk.Notebook을 이용해 소비자 모드와 관리자 모드 두 개의 탭으로 구성되어 있다. 각 모드는 별도의 페이지(SalesPage, AdminPage)로 구현했다. 각 페이지는 실제 자판기와 유사한 형태의 GUI로 구현했다. 자판기처럼 구현하기 위해서 인터넷에서 이미지를 찾아 Images 폴더에 저장했다. 이미지는 PIL.Image.open()으로 불러온 뒤, resize((70, 70))로 크기를 통일하고, ImageTk.PhotoImage로 변환해 Tkinter 위젯에 적용했다. 이미지는 각 슬롯(프레임) 내에 Label 위젯으로 표시된다.

1. 소비자 모드(SalesPage) 인터페이스

* 음료 배치

3행 10열(총 30개)로 음료 슬롯을 배치했다. 각 슬롯에는 음료 이미지, 가격이 표시된 구매 버튼이 들어있다.

* 금액 투입 및 결제

하단에는 1000원, 500원, 100원, 50원 버튼이 있어 클릭 시 금액이 추가된다. "카드" 버튼을 누르면 카드 결제 모드로 전환되어 재고가 있는 모든 상품의 구매 버튼이 활성화된다.

* 구매 버튼 활성화/비활성화

투입 금액과 재고 상황에 따라 각 음료의 구매 버튼이 활성화(초록색) 또는 비활성화(빨간색)됩니다.

* 잔돈 반환

"잔돈 반환" 버튼을 누르면, 투입된 금액이 1000/500/100/50원 단위로 계산되어 반환되고, 관련 메시지가 표시된다. 자판기 잔돈 반환에는 그리디 알고리즘을 사용했다. 이 알고리즘은 반환해야 할 금액에서 가장 큰 단위(1000원, 500원, 100원, 50원)부터 차례로 최대한 많이 사용해서 잔돈을 나누는 방식입니다.

* 상태 표시

현재 투입 금액, 카드 결제 여부, 구매 성공/실패, 잔돈 반환 등 각종 상태 메시지가 화면 우측에 표시된다.

1. 관리자 모드(AdminPage) 인터페이스

* 음료 관리

소비자 모드와 동일하게 3행 10열로 음료가 배치되며, 각 슬롯에는 이미지, 가격 입력창, 가격 저장 버튼, +/− 재고 조정 버튼이 있다. 여기서 수정한 변경점은 바로 소비자 모드에 적용된다.

* 가격 변경

가격 입력창에 값을 입력 후 "저장" 버튼을 누르면 해당 음료의 가격이 변경된다.

* 재고 조정

버튼을 누르면 재고가 1 증가(최대 재고 초과 불가)하고, 버튼을 누르면 1 감소(0 미만 불가)한다.

* 상태 메시지

가격 변경, 재고 조정 등 관리자 작업의 결과가 중앙 메시지로 출력된다.

1. Flow Chart

기존의 플로우차트에서 아이디 입력 부분만 삭제했다.

1. 사용 모듈 및 함수 설명

* VendingMachineUI 클래스

\_\_init\_\_(self, root) 함수는 Tkinter의 루트 윈도우를 받아, 윈도우의 타이틀을 "자판기"로 설정하고 크기를 1024x720 픽셀로 지정하며, 크기 조절이 불가능하도록 한다. 자판기 내부 로직을 담당하는 Machine 객체를 생성한 뒤, 탭 인터페이스를 제공하는 ttk.Notebook 위젯을 생성한다. 소비자 모드와 관리자 모드를 각각 SalesPage와 AdminPage 인스턴스로 만들어 notebook에 추가하고, 각 탭에는 "소비자 모드", "관리자 모드"라는 이름을 붙인다. 마지막으로 notebook을 윈도우 전체에 꽉 차게 배치하여, 사용자가 프로그램을 실행하면 곧바로 두 가지 모드 중 하나를 선택해 사용할 수 있도록 한다. 이 클래스는 자판기 프로그램의 진입점이자, 소비자/관리자 페이지 간의 데이터 및 상태 동기화를 위한 기반을 제공한다.

* SalesPage 클래스 (소비자 모드)

\_\_init\_\_(self, master, machine) 함수는 소비자 모드 페이지의 프레임을 초기화하는 역할을 한다. 부모 프레임을 생성하고, 자판기 로직을 담당하는 Machine 객체를 연결하며, UI에 필요한 각종 변수와 리스트를 세팅한다. 이후 init\_image()로 음료 이미지를 미리 불러오고, render\_tab()을 통해 전체 UI(음료 슬롯, 버튼, 로그 등)를 배치한다. 마지막으로 update\_ui()를 호출해 각 버튼의 활성화 상태와 금액 표시 등이 현재 상태에 맞게 갱신된다.

init\_image(self) 함수는 음료 이미지를 Tkinter에서 사용할 수 있도록 미리 준비하는 역할을 한다. 30개의 음료 이미지 파일을 Images 폴더에서 차례로 불러와 70x70 픽셀로 크기를 통일하고, Pillow 라이브러리의 ImageTk.PhotoImage로 변환하여 self.images 리스트에 저장한다. 이 이미지는 각 음료 슬롯에 시각적으로 표시된다.

render\_tab(self) 함수는 자판기 판매 모드의 전체 UI를 화면에 그리는 함수이다. 기존 프레임과 버튼을 모두 제거한 뒤, 3행 10열로 음료 슬롯 프레임을 배치하고 각 슬롯에 이미지와 구매 버튼을 추가한다. 하단에는 1000/500/100/50원 금액 투입 버튼, 카드 결제 버튼, 잔돈 반환 버튼을 배치하며, 현재 금액과 로그 메시지를 표시하는 라벨도 포함된다. 각 구매 버튼에는 해당 음료의 가격이 표시되고, 클릭 시 buy() 함수가 호출된다.

addMoney(self, num) 함수는 사용자가 금액 투입 버튼을 누를 때 호출된다. Machine 객체의 insert\_money(num) 메서드를 호출해 금액을 누적시키고, 이후 update\_ui()를 호출하여 투입 금액과 구매 가능 상품의 버튼 상태를 즉시 갱신한다.

card(self) 함수는 카드 결제 버튼이 눌렸을 때 호출된다. 투입 금액이 0원일 때만 동작하며, Machine 객체의 money 값을 -1로 설정해 카드 결제 모드임을 표시한다. 카드 결제 모드로 전환되면 update\_ui()를 호출해 재고가 있는 모든 상품의 구매 버튼이 활성화된다.

buy(self, idx) 함수는 사용자가 음료 구매 버튼을 클릭할 때 호출된다. 카드 결제 모드일 경우 Machine의 purchase\_card()를 실행해 구매를 처리하고, 카드 결제는 1회만 허용하므로 money를 0으로 초기화한다. 현금 결제 모드라면 purchase()를 실행해 구매를 처리하고, 투입 금액을 상품 가격만큼 차감하며, 구매 성공 메시지를 로그에 남긴다. 이후 update\_ui()를 호출해 버튼 상태와 금액 표시를 갱신한다.

return\_money(self) 함수는 잔돈 반환 버튼이 클릭될 때 호출된다. Machine 객체의 money 값을 가져와 0으로 초기화한 뒤, 반환할 금액이 없으면 안내 메시지를 띄우고, 반환할 금액이 있으면 1000원, 500원, 100원, 50원 순서로 동전/지폐 개수를 계산하는 그리디 알고리즘을 적용해 각 단위별 개수를 메시지로 출력한다. 마지막으로 UI를 갱신한다.

update\_ui(self) 함수는 현재 투입 금액, 카드 결제 여부, 각 음료별 구매 가능/불가 상태를 실시간으로 반영한다. 카드 결제 모드일 경우 재고가 있는 모든 상품의 구매 버튼을 활성화하고, 금액 표시를 "카드 결제 가능!"으로 변경한다. 현금 결제 모드일 경우 각 음료별로 재고와 투입 금액을 확인해 구매 가능하면 버튼을 활성화(녹색), 불가하면 비활성화(빨간색)로 표시하며, 현재 투입 금액을 화면에 표시한다.

* AdminPage 클래스 (관리자 모드)

init(self, master, machine, sales\_page) 함수는 관리자 모드 프레임을 초기화하고, 자판기 로직(Machine 객체)을 self.Machine에, 소비자 모드(SalesPage 객체)를 self.sales\_page에 연결한다. self.root에는 부모 프레임을 저장하고, self.width와 self.height에는 창의 크기를 설정한다. self.images 리스트는 각 음료의 이미지를 저장하며, self.entries 리스트는 각 음료별 (이름, 가격 입력창 엔트리) 튜플을 저장한다. self.slot\_width, self.slot\_height, self.gap, self.num\_cols, self.num\_rows는 각각 음료 슬롯의 크기, 슬롯 간격, 열과 행의 개수를 지정한다. self.frames 리스트는 각 음료 슬롯 프레임을 저장한다. init\_image() 함수로 음료 이미지를 불러오고, render\_tab() 함수로 전체 UI를 배치한다.

render\_tab(self) 함수는 관리자 모드의 전체 UI를 구성한다. self.frames에 저장된 기존 프레임을 모두 제거하고, self.frames 리스트를 비운다. 가운데 정렬을 위해 total\_width와 left\_margin을 계산한다. self.cur\_log는 현재 작업 결과나 상태 메시지를 저장하는 StringVar 객체이며, 중앙에 cur\_log 라벨을 배치한다. 각 행과 열을 순회하며, x, y 위치에 tk.Frame으로 음료 슬롯을 만들고, self.frames에 추가한다. 각 슬롯에는 이미지(img\_label), 가격 입력창(entry), 가격 저장 버튼(save\_btn), 재고 증가 버튼(plus\_btn), 재고 감소 버튼(minus\_btn)을 배치한다. entry에는 현재 음료 가격이 기본값으로 들어가며, self.entries 리스트에 (음료명, entry) 형태로 저장한다. save\_btn의 command에는 save\_price(), plus\_btn에는 increase\_stock(), minus\_btn에는 decrease\_stock() 함수가 각각 연결된다.

increase\_stock(self, name) 함수는 선택한 음료의 재고(self.Machine.items[name].stock)를 1개 증가시킨다. 재고가 최대 재고(self.Machine.items[name].maxStock)보다 작을 때만 동작하며, 성공 시 self.cur\_log에 "재고 +1 → 현재: n" 메시지를, 최대치일 경우 "최대 재고입니다!" 메시지를 출력한다.

decrease\_stock(self, name) 함수는 선택한 음료의 재고(self.Machine.items[name].stock)를 1개 감소시킨다. 재고가 1개 이상일 때만 동작하며, 성공 시 self.cur\_log에 "재고 -1 → 현재: n" 메시지를, 재고가 0일 경우 "재고는 이미 0입니다." 메시지를 출력한다.

save\_price(self, name, entry\_widget) 함수는 가격 입력창(entry\_widget)에 입력된 값을 해당 음료의 가격(self.Machine.items[name].price)으로 저장한다. 입력값이 0 미만이거나 숫자가 아니면 self.cur\_log에 오류 메시지를 출력하고, 정상적으로 입력되면 가격을 변경한다. 변경 후 self.cur\_log에 "가격이 n원으로 변경되었습니다." 메시지를 출력하고, 판매 페이지의 UI(self.sales\_page.update\_ui())와 관리자 페이지의 가격 입력창(self.update\_ui())을 모두 갱신한다.

update\_ui(self) 함수는 self.entries 리스트를 순회하며, 각 음료의 가격 입력창(entry)에 현재 가격(self.Machine.items[name].price)을 반영한다. 가격이 변경되었거나 외부에서 데이터가 변경된 경우에도 입력창의 값을 최신 상태로 유지한다.

init\_image(self) 함수는 30개 음료의 이미지를 load\_files 리스트에서 불러와 70x70 픽셀로 리사이즈하고, ImageTk.PhotoImage로 변환하여 self.images 리스트에 저장한다. 이 이미지는 각 음료 슬롯에 시각적으로 표시된다.

* Machine 클래스(자판기의 로직 담당)

\_\_init\_\_(self) 함수는 자판기 객체를 초기화하며, self.money에 현재 투입된 금액을 0으로 설정한다. self.items는 자판기에서 판매하는 모든 음료 정보를 담는 딕셔너리로, 키는 음료 이름(문자열), 값은 Item 객체이다. 예를 들어 "아이시스": Item(800, 40)처럼 가격과 초기 재고를 지정하며, 일부 음료는 기본 재고(20개), 일부는 40개로 설정한다.

insert\_money(self, amount) 함수는 amount(투입 금액)가 0보다 클 때만 self.money에 더하고 True를 반환하며, 그렇지 않으면 False를 반환한다. 이 함수에서 amount는 사용자가 투입한 지폐 또는 동전의 금액을 의미한다.

add\_item(self, name, obj) 함수는 새로운 음료를 자판기에 추가하거나 기존 음료 정보를 변경할 때 사용한다. name은 음료 이름(문자열), obj는 Item 객체로, self.items[name]에 할당한다.

get\_items(self) 함수는 현재 자판기에 등록된 모든 음료 정보를 담은 self.items 딕셔너리를 반환한다.

purchase(self, name) 함수는 현금 결제로 음료를 구매할 때 호출된다. name은 음료 이름이며, self.items[name]으로 해당 Item 객체를 가져온다. item.purchase(self.money)를 호출하여 구매 가능 여부와 재고 차감을 처리하고, 구매가 성공하면 self.money에서 해당 음료의 가격(item.price)을 차감한다.

purchase\_card(self, name) 함수는 카드 결제 시 음료를 구매하는 함수이다. self.items[name]으로 해당 Item 객체를 가져오고, item.purchase(self.money)를 호출해 재고를 차감한다. 카드 결제의 경우 self.money를 변경하지 않으며, 구매 성공 여부를 로그에 출력한다.

is\_available(self, name) 함수는 현재 투입 금액(self.money) 또는 카드 결제 상태에서 name에 해당하는 음료를 구매할 수 있는지 확인한다. self.items[name].is\_available(self.money)를 호출하여 재고와 금액 조건을 동시에 체크한다.

refund(self) 함수는 투입된 금액(self.money)을 반환하고, self.money를 0으로 초기화한다. 반환값은 환불할 금액(refund\_amount)이다.

* Item 클래스

\_\_init\_\_(self, price, stock=20) 함수는 Item 객체를 초기화하며, self.price에는 해당 음료의 가격을, self.stock에는 현재 재고를, self.maxStock에는 최대 재고를 저장한다. stock 파라미터의 기본값은 20이며, 생성 시 별도로 지정하면 그 값을 최대 재고로도 사용한다.

is\_available(self, money) 함수는 해당 음료가 구매 가능한지 판단한다. self.stock이 1개 이상이어야 하며, money(투입 금액)가 self.price(음료 가격) 이상이거나 money가 -1(카드 결제 신호)일 때 True를 반환한다. 즉, 재고가 있고 현금 또는 카드 결제로 결제 조건이 충족되면 구매가 가능하다.

purchase(self, money) 함수는 실제 구매가 가능한지 is\_available()로 확인한 뒤, 조건이 충족되면 self.stock을 1 감소시키고 True를 반환한다. 구매 조건이 불충분하면 아무 동작 없이 False를 반환한다.

1. 실행 화면 캡쳐

* 소비자 모드

스크린샷, 텍스트, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.텍스트, 스크린샷이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

* 관리자 모드

텍스트, 스크린샷, 소프트웨어, 컴퓨터 아이콘이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.텍스트, 스크린샷, 컴퓨터 아이콘, 소프트웨어이(가) 표시된 사진

AI 생성 콘텐츠는 정확하지 않을 수 있습니다.

* 실행 방법

Main.py를 실행한다.

1. 느낀 점

최근에 게임엔진1 수업에서 디자인 패턴에 대해 배웠다. MVP 패턴을 적용하여 프로그램을 구현하려고 노력했는데 제대로 됐는지 모르겠다.