Projekt: Gilded Rose

Kacper Laskowski(246853)

Kod poczatkowy:
https://github.com/emilybache/GildedRose-Refactoring-Kata/tree/master/python

Kod zrefraktyzowany:
https://github.com/kaaspaa/zjp

1. **Wymagania aplikacji:**
    - Gilded Rose posiada przedmioty, z wartościami “sell_in”, oraz “quality”
    - “sell_in” posiada infromacje ile dni zostalo na sprzedanie przedmiotu
    - “quality” oznacza jakość danego przedmiotu.
    - Gilded Rose na koniec dnia zmniejsza “sell_in” i “quality” o 1.
    - Jakość przedmiotu nie może przekraczać 50
    - Jakość przedmiotu nie może być ujemna. Przedmioty, które nie sprzedały się w podanym czasie psują się 2 razy szybciej
    - Przedmioty “Conjured”(Zaklęte) psuja sie 2 razy szybciej niz przedmioty normalne.
    - Przedmiot “Sulfuras” nie jest do sprzedania i nie zmniejsza sie jego jakość
    - Jakość przedmiotu “Aged Brie” wzrasta o 1 na koniec dnia
    - Jakość przedmiotu “Backstage passes” wzrasta im blizej jest dnia koncertu(o 3 - jesli jest 5 lub mniej dni, o 2 - jesli jest 10 lub mniej dni, o 1 - w innym przypadku), oraz spada do 0 po koncercie.
2. **Moja refaktoryzacja:**

Na samym początku dodalem opcje sprawdzania czy w item.name znajduje sie slowo “Conjured”. Jeśli tak, to ponownie zmniejszałem item.quality.

Natepnym krokiem bylo umieszczenie sprawdzenia czy item.name nie jest “Sulfuras, Hand of Ragnaros” na początku “fora”, przez którego przechodził każdy przedmiot. Ta czynność zmniejszyła ilość “ifów” z 3 na 1 dotyczace ten jeden przypadek. Następnie dołożyłem dwa sprawdzenia:

- Czy item.name jest “Aged Brie”
- Jeśli nie to czy nie jest “Backstage passes”

Jeśli nie jest i tym, to znaczy, że przedmiot jest albo normalny, albo “Conjured”.

Następnym krokiem było wrzucenie wszystkich zwiększeń/zmniejszeń “quality” i “sell_in” do metod w innym pliku nazwanym “functions.py”. Dodałem również sprawdzenie czy “sell_in” jest już mniejsze od 0, gdyż w

takim przypadku nie ma sensu ponowne zmniejszenie tej wartości, a może to spowodować przykre skutki w przyszłości.

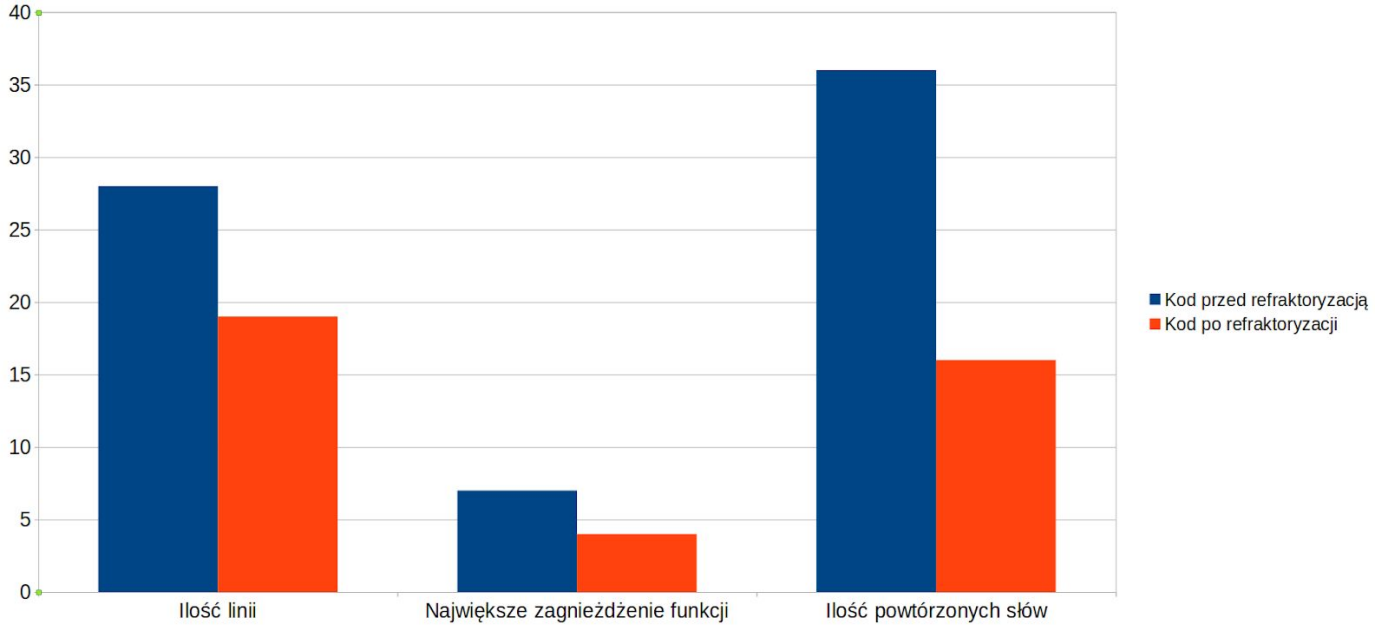
W tym momencie kod już wyglądał względnie zgrabnie jednak, przeszkadzały duże bloki “ifów” w sprawdzeniu ile dni zostało do końca koncertu TAFKAL80ETC. Dlatego sprawdzenie tego przenioslem do functions.py do nowej metody nazwanej “increase_quality_in_concert_passes”.

Dla większej przejrzystości kodu zapytanie o to czy przedmiot jest zaklęty zostało przeniesione do metody “decrase_quality”, co spowodowało kolejne zmniejszenie się bloku kodu w “update_quality” o 2 sprawdzenia.

Najwazniejszym ruchem, bylo stworzenie pliku “item.py” zawierajacego klase Item, oraz metode “update_item”, do której został przeniesiony prawie w całości kod z “update_quality”. W tym momencie “gilded_rose” zostało wychudzone o całą klasę “Item” a metoda “update quality” przechodzi po wszystkich przedmiotach, odpalając przy tym metodę aktualizująca ten przedmiot. Te dzialanie mialo za zadanie oddzielic dwie klasy z pliku “gilded_rose.py” sprawiając przy tym, że każda z nich zawiera funkcje przypisane specjalnie dla niej.

Ostatnim krokiem, było oddzielenie dwoma folderami testy, od kodu działającego. Pomaga to w uporządkowaniu struktury projektu, jak i w znalezieniu poszukiwanych informacji w kodzie.

3. Porownanie:



Jak pokazuje wykres, kod po mojej refaktoryzacji jest znacznie prostszy w zrozumieniu, jak i krótszy w działaniu. Dla malej ilosci danych nie ma to znaczenia, jednak wraz ze wzrostem ich ilości długość działania programu moze sie wydluzyc.

Kacper Laskowski