Zadanie 1.

Skrypt o nazwie lotto.py zawiera prostą implementacja maszyny losującej, w której użytkownik może wybrać 6 liczb sam, albo skorzystać z opcji chybił trafił, w której program sam wybierze mu liczby. Następnie program losuje 6 innych liczb i jak w Lotto wypisuje ile liczb udało się użytkownikowi trafić. Napisz testy do lotto.py (z wykorzystaniem monkeypatch).

Zadanie 2.

Zaimplementuj dodatkowe funkcjonalności oraz testy do gry opracowywanej w ramach materiałów wideo (pliki: classes_base.py, game_base.py, test_classes_base.py). Proponuję, by każda osoba zaimplementowała co najmniej jedną funkcjonalność. Sugeruję, by osoby o nazwisku rozpoczynającym się na A-D zaczęli od funkcjonalności nr 1, E-J nr 2, K nr 3, L-Ł nr 4, M-Sobi nr 5, Sobo-Z nr 6.

F1 Atakowanie gracza przez przeciwników

Zmodyfikować program w taki sposób, aby w każdej rundzie, w której gracz zaatakuje przeciwnika, ten sam przeciwnik zadawał obrażenia graczowi. W tym celu należy rozbudować klasę gracza o możliwość przyjmowania obrażeń (metoda take_damage()), a klasę przeciwnika o możliwość atakowania (metoda attack()).

Odnośniki do kodu: klasa Character(wydzielona wspólna funkcjonalność klas Player i Enemy), modyfikacja metody play() z klasy Game

F2 Wypisywanie stanu gry

Zmodyfikować program w taki sposób, aby na początku każdej rundy wypisywał dane dotyczące zarówno gracza, jak również wszystkich przeciwników będących jeszcze w grze (można skorzystać z metody str()). Dodatkowo, na koniec gry wypisać, czy gracz wygrał daną rozgrywkę, czy przegrał (zakładamy, że wygrana następuje wtedy gdy graczowi udało się pokonać wszystkich przeciwników)

Odnośniki do kodu: metoda print_game_status() z klasy Game i modyfikacja metod play() z klasy Game, oraz main()

F3 System punktów doświadczenia

Zmodyfikować program w taki sposób, żeby gracz za zabicie przeciwnika otrzymywał określoną liczbę punktów doświadczenia. W uproszczonej wersji, liczba punktów doświadczenia za każdego przeciwnika może być taka sama. Po zdobyciu określonej liczby punktów, gracz "przechodzi poziom", jego moc się odnawia i jednocześnie zwiększa się liczba wymaganych punktów do przejścia kolejnego poziomu.

Odnośniki do kodu: atrybut (wraz z getterem i setterem) _exp oraz metoda level_up() w klasie Player, modyfikacja metody play() z klasy Game

F4 Generator losowych przeciwników

Zmodyfikować program w taki sposób, żeby była możliwość wygenerowania listy losowych przeciwników na podstawie takich informacji jak: liczba przeciwników w danej rozgrywce, minimalne i maksymalne zdrowie każdego przeciwnika, możliwe rodzaje przeciwników.

Odnośniki do kodu: metoda randomize_enemies() w klasie Game oraz potencjalna modyfikacja w metodzie main()

F5 Przeciwnicy z wieloma głowami nie umierają od razu

Zmodyfikować program w taki sposób, żeby przeciwnicy, którzy mają wiele głów (w dotychczasowym kodzie jest to Hydra i DragonHydra), gdy ich punkty życia spadną do 0 ginęli tylko, gdy pozostała im jedna głowa, w przeciwnym przypadku tracą głowę, a ich punkty życia wracają do wartości bazowej.

Odnośniki do kodu: modyfikacja metody take_damage() w klasie Hydra

F6 Nakładanie efektów na gracza

Zmodyfikować program w taki sposób, żeby niektórzy przeciwnicy mogli nakładać na gracza efekty, które przez określoną liczbę rund co rundę będą zmniejszać (warto się też zastanowić nad tym, żeby implementacja zakładała również istnienie efektów pozytywnych np. leczenia) punkty życia gracza. Przykładowa klasa Effect mogła by mieć takie atrybuty jak nazwa, skutek, oraz czas trwania, a każdy gracz miałby listę takich efektów.

Odnośniki do kodu: cała klasa Effect oraz PoisonousHydra, metody add_effect(), remove_effect() oraz apply_effects() w klasie Player, modyfikacja metody play() z klasy Game