# Text Adventure Game

Projekt PIPR – Lato 2021 – Prowadzący: Oleszkiewicz Witold

## 1. Cel projektu

Celem projektu było stworzenie gry komputerowej pozwalającej graczowi na poruszanie się po opisanej tekstem mapie, walkę z przeciwnikami, podnoszenie przedmiotów, dialog z bohaterami gry, zdobywanie pieniędzy, kupowanie przedmiotów, itd.

Dodatkowo program powinien umożliwiać wczytywanie z pliku danych konfiguracyjnych, gdzie możliwa byłaby pełna konfiguracja bohatera, świata, przeciwników, przedmiotów oraz misji do wykonania.

## 2. Opis projektu

Gra została wykonana w środowisku programistycznym Python w programie Visual Studio Code. Komunikacja z użytkownikiem odbywa się poprzez terminal, w którym są wyświetlane informacje o stanie gry, oraz za pomocą którego gracz wprowadza tekstowo komendy dotyczące zachowania bohatera w grze. Implementacja znajduje się na gitlabie wydziałowym:

https://gitlab-stud.elka.pw.edu.pl/jjedrzej/21l-pipr-jedrzejewski-projekt

## 3. Podział programu na pliki

Oprogramowanie zostało podzielone na pliki i podfoldery, aby jego struktura była bardziej przejrzysta.

W głównym folderze znajduje się plik main.py który odpowiada za uruchomienie gry oraz za wykonywanie głównej pętli programu.

W folderze utilities zamieszczone zostały pliki:

- start_game.py – odpowiadający za początkowe menu w którym gracz wybiera, czy zaczyna nową grę, czy wczytuje już wcześniej zapisaną, a także stworzenie obiektów wczytanych na podstawie wybranego pliku konfiguracyjnego;

- interpret.py – odpowiada za interpretowanie komend wprowadzonych przez gracza;

- commands.py - zawiera funkcje specyficzne dla każdej komendy wywoływane w funkcji interpret po wybraniu przez użytkownika danej komendy;

- additional_func.py – zawiera dodatkowe funkcje wykorzystywane przez funkcje komend, takie jak generowanie zdań i usuwanie ukończonych misji;

- file_data.py – przechowuje tytuł gry oraz ścieżki do wczytania, zapisu i załadowania gry;

- errors.py - zawiera wszystkie klasy wyjątków wykorzystywane w projekci;

W folderze classes zamieszczone zostały pliki odpowiadającej każdej klasie. W jednym pliku world.py są klasy World i Map, a w pliku planet.py klasy Planet i Store.

Do klas zostały napisane testy jednostkowe w odpowiednich plikach
test_statClasses.py dla klas Stats, Player, Item, Character, Npc i Enemy
test_worldClasses.py dla klas World, Map, Planet, Mission i Store

W głównym folderze zamieszczone zostały pliki z których można wczytać grę:

- practise.json – plik krótkiej gry, w którym zapisane są wszystkie konieczne informacje do wygenerowania odpowiednich klas i przetestowania funkcjonalności;

- game.json – przykład rozmiaru pełnej gry - fabuła nie jest dokończona;

## 4. Podział programu na klasy

Klasa Stats odpowiada za przechowywanie, modyfikacje oraz dostęp do statystyk, wykorzystana jest przy tworzeniu klasy Player oraz Items.

Klasa Player dziedziczy po klasie Stats, przechowuje informacje o ekwipunku gracza czyli przedmiotach, które gracz posiada oraz o lokalizacji, w której gracz się znajduje. Jest to kluczowe, ponieważ na podstawie tej lokalizacji jesteśmy w stanie stwierdzić, jakie interakcje ze światem może wykonać użytkownik.

Klasa Items odpowiada za przedmioty w grze, które można podnosić lub kupować. Każdy przedmiot posiada atrybuty statystyk, nazwę, identyfikator id, może posiadać informacje o cenie i czy ma się dodać do ekwipunku gracza po podniesieniu/kupieniu.

Klasa Characters obsługuje podstawowe atrybuty bohaterów w grze, imię i id.

Klasa Enemy dziedziczy po Characters, dodatkowo posiada atrybuty związane ze zdrowiem, siłą, tarczą oraz nagrodą za pokonanie przeciwnika.

Klasa Npc dziedziczy po klasie Characters, dodatkowo obsługuje dialogi bohaterów.

Klasa World przechowuje wszystkie informacje o świecie, mapę, listy obiektów planet i misji oraz gracza. Przy generowaniu klasy World tworzone są obiekty planet, misji i gracza, a przy ich generowaniu tworzone są się sklepy, przedmioty i bohaterowie.

Klasa Map przechowuje listy nazw planet i misji, które generuje wczytując z pliku. Odpowiada za wskazanie możliwych ruchów po mapie na podstawie obecnej lokalizacji gracza.

Klasa Planet tworzy obiekt planety, który posiada opis, listę misji na tej planecie do wykonania oraz informacje o ewentualnym sklepie na planecie. Wskazuje misję, którą gracz teraz wykonuje. Kolejność wykonywania misji zależy od kolejności ustawienia ich na liście.

Klasa Mission odpowiada za świat otaczający gracza, misje są przypisane do konkretnej planety. Jest traktowana jako pewien stan planety. Klasa posiada listy przedmiotów, wrogów i przyjaciół, na których gracz może wykonywać komendy. Przedmioty, wrogowie i przyjaciele są traktowane jako rzeczy do wykonania - zadania. W momencie użycia odpowiedniej komendy (odbycia interakcji z przedmiotem / bohaterem) są usuwane z listy. Klasa posiada

listę wymaganych przedmiotów w ekwipunku gracza. One również są usuwane z listy po wykonaniu zadania. W momencie gdy cztery listy (przedmioty, wrogowie, przyjaciele, wymagane przedmioty) są puste uznajemy misję za wykonaną i przechodzimy do następnej, o ile jest następna. Po wykonaniu zadania gracz dostaje nagrodę (reward) przewidzianą za misję.

Klasa Store obsługuje podstawową funkcjonalność sklepu. Wypisuje opis sklepu oraz przedmioty dostępne do kupienia. Posiada listę obiektów przedmiotów w sklepie, oraz listę identyfikatorów przedmiotów.

## 5. Instrukcja uruchomienia gry

Aby zagrać w grę należy uruchomić plik main.py. Następnie postępować zgodnie z podpowiedziami wyświetlanymi na ekranie.

Domyślnie wybrana jest gra z pliku practise.json. Przejście tej gry umożliwi przetestowanie i nauczenie się wszystkich komend. (ok 7 minut)

Aby udało się przejść tą grę należy zrobić wszystkie możliwe zadania na Ziemi, następnie polecieć na Marsa. Wykonać wszystkie możliwe zadania. Wrócić na Ziemię, odwiedzić sklep, kupić przedmioty. Polecieć na Jowisza i wykonać wszystkie misje. Przez Ziemię i Marsa polecieć na Saturn, wykonać wszystkie zadania i wrócić na Ziemię.

## 6. Opis konfiguracyjnych plików

Konfigurowanie gry odbywa się poprzez zmienianie / dodawanie zawartości w pliku z którego generujemy grę practise.json / game.json.

Można zmienić statystyki początkowe gracza w zakładce ‘’player’’.

Dodać nowe lub zmienić położenie istniejących lokalizacji w zakładce ‘’map’’.

Dodawanie lokalizacji wiąże się z uwzględnieniem nowej lokalizacji w zakładce ‘’planets”, gdzie można też edytować istniejące planety.

Dodawanie misji i sklepów w planetach wiąże się z dodaniem odpowiednich instancji w zakładkach ‘’missions’’ i ’’stores’’.

Analogicznie jest z dodawaniem nowych postaci i przedmiotów.

Każdy nowo powstały obiekt powinien mieć identyczną strukturę do tych zawartych w oryginalnym pliku game.json. Zalecane jest modyfikowanie pliku game.json w nowym pliku, aby nie zniszczyć istniejącego świata.

W pliku file_data.py można zmienić ścieżkę do pliku nowej gry (DEFAULT_GAME), jeżeli np. stworzymy swój nowy świat w nowym pliku, wybrać ścieżkę do zapisu gry (SAVE) oraz do wczytania istniejącej gry (LOAD).

Każda gra zapisuje się domyślnie do pliku SAVE, oznacza to, że jeżeli zapiszemy kolejną grę bez zmiany wartości SAVE w file_data.py to plik się nadpisze i będzie dostępna tylko ostatnia zapisana gra.

## 7. Podsumowanie

Podczas projektu zostało opracowanie oprogramowanie gry spełniające wszystkie cele i założenia. Kod jest podzielony na pliki, co ułatwia poruszanie się po nim.

W trakcie opracowywania gry zaimplementowano wiele testów sprawdzających poprawność działania klas i wybranych funkcji. W pozostałych przypadkach przeprowadzono gruntowne testy ogólne sprawdzające poprawność funkcjonowania oprogramowania jako całości.

Grę przetestowano również przez osoby trzecie, które znalazły pewne błędy zapisu pliku i wyświetlania informacji. Dodatkowo zwróciły uwagę na detale przyjemne dla użytkownika, takie jak wyświetlanie pieniędzy w sklepie i paliwa po przeleceniu na nową planetę.

Podział na klasy ewoluował wraz z pisaniem kodu. Np. na początku nie przewidywano klasy Missions, która połączyła stany różnych planet oraz zadania do wykonania. Podobnie wprowadzono klasę Store, która okazała się bardzo pomocna i ułatwiająca wykonywanie zadań sklepu.

Dodatkowo opracowano w ramach projektu generator zdań powodujący, że komunikaty wypisywane na ekran są bardziej atrakcyjne dla gracza, co powoduje wrażenie głębszej fabuły gry.

Obecny stan projektu pozwala na rozbudowanie go w przyszłości o bardziej skomplikowane funkcje. Np. komenda attack w przyszłości mogłaby być zastąpiona bardziej rozbudowaną symulacją walki. Rozmawianie z bohaterami mogłoby zyskać na atrakcyjności gdyby można było odpowiadać na wiadomości.

## 8. Wnioski

W prototypie projektu korzystałem z informacji przechowywanych w pliku typu json. W głównej wersji wszystko jest obsługiwane na obiektach generowanych z pliku. Do danych jest stosunkowo łatwy dostęp poprzez metody dostępowe.

Przygotowana dokumentacja w kodzie bardzo pomaga przy tworzeniu bardziej rozbudowanych funkcji czerpiących z mniejszych funkcji. Ułatwiło to np. rozróżnianie czy funkcja zwraca listę nazw czy listę obiektów itp.

Opracowane wyjątki pomogły rozstrzygnąć w którym miejscu w pliku lub w kodzie jest błąd.

**9. Oświadczenie**

Oświadczam, że wszystkie oddawane przeze mnie prace stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu PIPR zostały wykonane przeze mnie samodzielnie. Oświadczenie dotyczy prac z semestru letniego roku akademickiego 2020/2021.

Imię i nazwisko: Jan Jędrzejewski

Nr albumu 310713

Data 16.06.2021