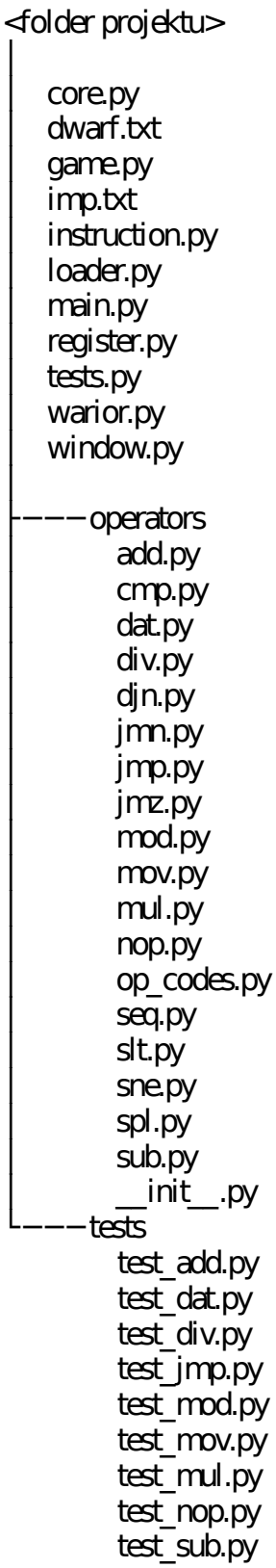
pyMARS – symulator Core Wars

dokumentacja projektowa

- main.py – Główny plik programu, w nim tworzony jest obiekt Game.
- game.py – Zawiera klasę Game, tworzy obiekt GUI, Loader oraz Core. Zawiera również funkcję loop, która jest wywoływana cyklicznie przez GUI. W __init__ w Game tworzeni są wojownicy.
- loader.py – Zawiera klasę Loader. Ładuje kod wojownika z pliku.
- window.py – Zawiera klasę GUI. Tworzy okienko i płótno. Wywołuje przy każdym odświeżeniu funkcję loop z game.py. Rysuje wszystkie bloki rdzenia na płótnie oraz zawiera funkcje do aktualizacji tych bloków. Do okienek użyłem biblioteki tkinter.
- core.py – Zawiera rdzeń, tablicę wojowników oraz rejestry instrukcji, źródła i celu. Początkowo jest on wypełniany instrukcjami DAT.F $0, $0. Zawiera funkcję execute_current_ins, która wykonuje bieżącą instrukcję, oraz wiele funkcji związanych z manipulacją wojownikami oraz samym rdzeniem. Funkcja decode_field przetwarza tryby operandów (np. #, $, *).
- register.py – klasa rejestru, po niej dziedziczą rejestry instrukcji, źródła oraz celu
- instruction.py – Zawiera listę dozwolonych operatorów oraz modyfikatorów. Funkcja represents_int sprawdza, czy string zawiera jedynie liczbę całkowitą. Klasa Instruction zawiera wszystkie elementy instrukcji, czyli operator, modyfikator, operandy i ich tryby. Ponadto jest tam również zmienna last_warior, do której przypisany jest wojownik, który jako ostatni modyfikował instrukcję. Funkcja __eq__ służy do porównywania wszystkich pól klasy, oprócz last_warior. Funkcja from_line służy do przetwarzania linii np. z pliku na obiekt Instruction.
- warior.py – Zawiera atrybuty wojownika, listę procesów oraz funkcje pozwalającą na manipulowanie procesami i instrukcjami.
- op_codes.py – Plik zawiera abstrakcyjną klasę Op_code, po której dziedziczą wszystkie operatory. Zawiera również funkcje wspólne dla wielu operatorów, takie jak np. terminate_process oraz jump_adr. Funkcja execute jest wywoływana przez jądro, oraz wywołuje klasy odpowiednich instrukcji. Ciekawą rzeczą jest użycie __import__ oraz getattr w celu dynamicznego wywoływania klas o nazwie zawartej w stringu. Tak, nazwy funkcji są najpierw walidowane, aby zapobiec wywoływaniu arbitralnych funkcji, co mogłoby być niebezpieczne.
- __init__.py – Plik wymagany przez Pythona.
- pozostałe pliki w operators – Zawierają implementację operatorów dla każdego modyfikatora.



- tests.py – Plik, który trzeba uruchomić, aby wykonać testy jednostkowe. Musiałem zrobić do tego ten plik, aby móc umieścić wszystkie testy w module tests.
- moduł test – zawiera testy jednostkowe (pytest)
- imp.txt – program wojownika znanego jako imp
- dwarf.txt – program wojownika znanego jako dwarf

W projekcie jest wiele rzeczy do poprawy oraz ulepszenia, jak np. implementacja wyjątków, wyświetlanie na ekranie, a nie w konsoli kto wygrał oraz proszenie użytkownika o podanie nazwy pliku, wymiarów okna i wielkości jądra. Dobrym pomysłem byłoby dodanie dynamicznego skalowania okna (co było w pierwszej wersji projektu) oraz statystyk wyświetlanych obok płótna. Nie będę wymieniał wszystkiego, ponieważ można to po prostu podsumować przez „Trzeba było zacząć projekt wcześniej, niż tydzień przed terminem.” Prawdopodobnie są jakieś ostrzeżenia pep8, ale nie zainstalowałem go i już dzisiaj nie zdążę, a nie chcę kolejnych -2 pkt.