CORE WARS

Dokumentacja projektu indywidualnego

PAWEŁ MÜLLER

1. CZYM SĄ TE „WOJNY RDZENIOWE"? KTO W OGÓLE CHCIAŁBY W TO GRAĆ?

Wojny Rdzeniowe (ang. *Core War*) to ~~zmora pierwszorocznych studentów~~ gra komputerowa. Polega ona na pisaniu programów-wojowników w języku mającym asemblerowe naleciałości, zwanym Redcode. Wyżej wymienionych wojowników umieszcza się ich w cyklicznym rdzeniu, który staje się polem bitwy oraz naprzemiennie wykonuje ich instrukcje.

Brzmi prosto? Nic bardziej mylnego. Programy te mogą przemieszczać się po rdzeniu, stawać się wieloprocesowymi, żądnymi krwi wytworami wyobraźni programistów (bo kto inny, w dobie gier 3D i serwisów z multimediami na żądanie, chciałby się tym zajmować), zmusić inny program do wykonania narzuconego mu kodu i wiele, wiele więcej. Gra toczy się tak długo, aż jeden z wojowników… umrze (kto by pomyślał?). Po bardziej szczegółowe wyjaśnienie przebiegu rozgrywki oraz obowiązujących zasad - kliknij tutaj.

2. JAK URUCHOMIĆ PROGRAM? – KRÓTKA INSTRUKCJA OBSŁUGI.

Przed uruchomieniem programu należy upewnić się, czy na naszym komputerze zainstalowana jest najnowsza wersja Pythona. Dodatkowym elementem wymaganym przez grę jest biblioteka termcolor. Jeśli posiadasz już te elementy na swojej stacji roboczej możesz zacząć czytanie tego rozdziału od punktu 3.

1. Instalacja Pythona.

Jest to trywialny proces, z którym poradzi sobie **każdy**. Wystarczy pobrać odpowiedni plik z podanej strony (kliknij tutaj!), a następnie odpowiedzialnie przejść przez standardowy proces instalacji. I już!

2. Instalacja biblioteki termcolor.

Ten krok nie jest tak prosty jak poprzedni, ponieważ wymaga od użytkownika użycia klawiatury, ale to nam nie straszne. Otwórz na swoim komputerze **Terminal** (lub **Wiersz polecenia**, jeśli używasz systemu Windows). Zacznijmy hackować, niczym w Hollywoodzkich filmach - w okno, które pojawiło się na twoim ekranie wpisz:

```
pip install termcolor
```

Teraz cierpliwie przeczekaj proces instalacji. Gotowe!

3. Pobieranie plików programu na swój komputer.

Wszystkie pliki, potrzebne do uruchomienia programu, można znaleźć w moim repozytorium na platformie GitHub (kliknij tutaj!).

Są to:

Należy umieścić je w jednym, wybranym katalogu w znanym sobie miejscu.

Opcjonalnym krokiem jest pobranie katalogu Warriors, gdzie znajdują się przykładowe pliki wojowników, których można użyć w rozgrywkach i umieszczenie go w tym samym folderze, co reszta pobranych materiałów.

4. Jak powinien wyglądać plik wojownika?

Gotowce są dla słabeuszy, dlatego najlepiej napisać kod własnego wojownika, by zrozumieć zasady gry. Nie będę się tu rozwodził nad tym jak pisać wydajnych wojowników, a nad sposobem zapisu ich instrukcji do pliku. Najlepiej zapisać go z rozszerzeniem `.red`.

Przykładowy plik wojownika z objaśnieniami:

```
; Dwarf       ← Każda linia poprzedzona znakiem „;” jest komentarzem

ADD #4, 3     ; execution begins here  ← Komentarz na końcu instrukcji
MOV.I 2, @2
JMP -2                                 } Instrukcje
DAT #0, #0
```

Instrukcje mają określoną budowę - składają się z trzyliterowego mnemonika (np. `MOV`), opcjonalnego modyfikatora oddzielonego kropką (np. `I`) oraz jednej lub dwóch zmiennych (np. 2), które mogą mieć opcjonalny tryb adresowania (np. @). Jest to formatowanie narzucane z góry przez autorów gry.

5. Uruchomienie programu.

Najważniejsza część tego rozdziału, czyli co, gdzie i jak zrobić, żeby rozpocząć bitwę.

Zacznij od uruchomienia **Terminala** (lub **Wiersza polecenia**, jeśli używasz systemu Windows). Następnie używając komendy **cd** przejdź do katalogu, w którym zachowane zostały przez Ciebie pliki (jeśli nie wiesz, jak to zrobić, wpisz polecenie *help cd*).

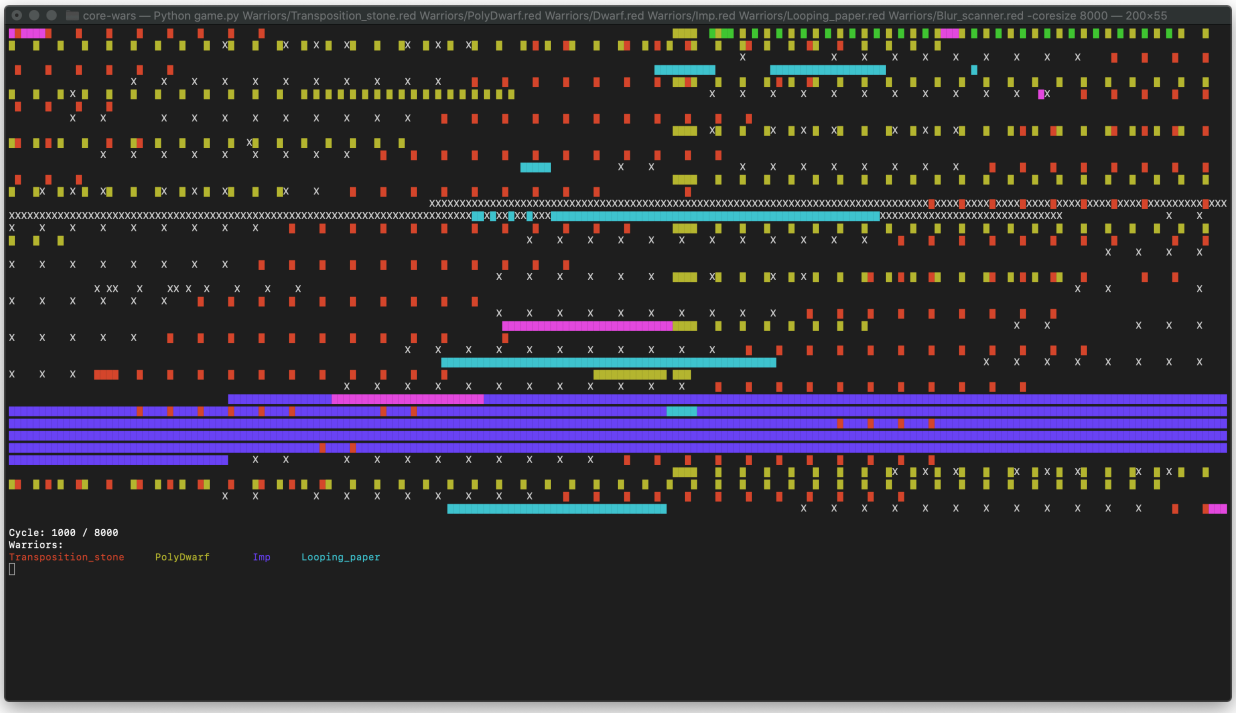
Grę można włączyć przy użyciu następującej komendy:

```
python game.py Warriors/Imp.red -coresize 2000 -cycleslimit 5000
```

Objaśnienia poszczególnych argumenów:

`python`	Uruchamia plik.
`game.py [ścieżka1 ścieżka2 itd…]`	Nazwa głównego pliku, który obsługuje grę. Po nim należy wpisać ścieżki do plików z instrukcjami wojownika.
`-coresize [liczba]`	Możliwość zmiany rozmiaru rdzenia (pola walki) Najlepiej podać liczbę będącą wielokrotnością 100. Domyślna wartość: 4 000.
`-cycleslimit [liczba]`	Możliwość zmiany limitu cykli (jeśli gra nie zostanie w tym czasie rozstrzygnięta, zostanie ogłoszony remis). Domyślna wartość: 10 000.

Pole oznaczone kolorem czerwonym jest **wymagane**. Nawiasów kwadratowych nie zapisujemy. Elementów wymienionych w tabelce możemy używać w dowolnej kolejności.



Jeśli wszystko wykonaliśmy poprawnie, nasza gra po pewnym czasie powinna wyglądać podobnie do tej, którą przedstawiłem na grafice powyżej.

3. STRUKTURA PROGRAMU – KRÓTKA INSTRUKCJA OBSŁUGI.

Swój program podzieliłem na 5 plików i 3 główne klasy:

- `Game.py`:
  - obsługa uruchamiania programu,
  - przetwarzanie argumentów,

- `MARS.py` - klasa `MARS`:
  - obsługuje rdzeń,
  - przeprowadza symulację rundy,
  - umożliwia wizualną reprezentację rdzenia,

- `Warrior.py` - klasa `Warrior`:
  - obsługuje wojowników,
  - odpowiednio wczytuje pliki,

- `Redcode.py` - klasa `Instruction`:
  - obsługuje język Redcode,
  - interpretuje i uruchamia:
    - tryby adresacji,
    - modyfikatory,
    - instrukcje,
  - konwertuje linię tekstu z pliku wojownika na odpowiedni obiekt,

- `Validating_tools.py` - klasy błędów:
  - obsługa błędów i wyjątków.

4. ZAKRES WYKONANYCH PRAC

Pracy było dużo, a ilość problemów wzrastała lawinowo. Poniżej zamieszczam podsumowanie projektu ze strony programisty.

1. Co udało się osiągnąć?

Krótko i zwięźle: wszystkie założenia projektowe.

Mój program obsługuje wymagane rodzaje instrukcji, modyfikatorów oraz trybów adresacji. Potrafi odczytywać pliki wojowników oraz dokonywać ich interpretacji, a także wykrywać błędy w ich strukturze. Sporą część funkcjonalności pokryłem testami jednostkowymi, które pomogły mi kontrolować jakość wprowadzanych z czasem zmian. Zgodnie z wymogami zaimplementowałem możliwość graficznej reprezentacji rdzenia (i nawet obsługuje kolory!). Dodatkowo mój kod (bez modyfikacji ustawień IDE) nie zawiera żadnych błędów lintera flake8, włącznie z błędami zbyt długich linijek. Niemalże każda funkcja została opatrzona opisem w języku angielskim objaśniającym jej działanie. Wszystkie metody odpowiedzialne za obsługę rdzenia zabezpieczyłem przed przypadkowym odwołaniam się poza zakres indeksowania (wielokrotnie powtarzające się dzielenie modulo). A co najważniejsze - nauczyłem się mnóstwa nowych instrukcji i metod Pythona.

2. Trudności napotkane przy tworzeniu projektu.

Najwięcej problemów stanowiło programowanie metod związanych z wykonywaniem instrukcji wojowników. Trudność pojawiła się w momencie, gdy trzeba było obliczać adresy źródłowe i docelowe w rdzeniu przy pomocy trybów adresacji (te zostały opracowane na podstawie asemblera, co, przy jego nieznajomości, stanowiło spore wyzwanie). Dodatkowym utrudnieniem był fakt, że wiele źródeł wiedzy o grze podawały sprzeczne ze sobą informacje, co czasem stwarzało dylematy (w projekcie trzymam się konwencji z instrukcji nadesłanej w poleceniu). Najczęściej powtarzane przeze mnie błędy to: brak nawiasów na końcu wywoływania bezargumentowej funkcji, nadmiarowe nawiasy dopisywane przy zmiennych obiektu oraz brak inkrementacji iteratora pętli (znalezienie tego konkretnego trwało 2 dni…). Warto również wspomnieć, że wyświetlanie tak dużej ilości informacji przy pomocy tekstu nie należy do najbardziej wydajnych, dlatego wiele uwagi poświęciłem szukaniu sensownego sposobu przekazywania tekstu do Terminala / Wiersza polecenia. W tym przypadku zastosowałem *list comprehension*, co kilkukrotnie przyspieszyło wykonywanie programu.

5. BIBLIOGRAFIA

Podczas opracowywania projektu korzystałem z wielu źródeł oraz pomocy:

- The beginners' guide to Redcode - poradnik załączony razem z poleceniem projektowym, to od niego zacząłem poznawać podstawy języka Redcode i zasady działania gry,
- corewar.io Documentation - wspaniała dokumentacja, dzięki niej zrozumiałem i mogłem zaprojektować największą część programu, w szczególności tryby adresacji zmiennych,
- Wojny rdzeniowe Redcode: samouczek - ta strona pozwoliła mi zrozumieć zasady działania instrukcji SPL,
- Python __Underscore__ Methods - jak sama nazwa wskazuje, to tutaj znalazłem spis wszystkich „metod z podkreśleniem”,
- Dokumentacja języka Python - obszerna baza wiedzy na temat ogromnej ilości modułów, klas, metod i zasad działania samego środowiska.

6. PODZIĘKOWANIA

Na koniec chciałbym bardzo podziękować mojemu przewodnikowi po świecie Pythona - Panu Mateuszowi Modrzejewskiemu, za okazaną cierpliwość, życzliwość i wszystkie dobre rady, które zostały mi udzielone.

Należy również wspomnieć o wszystkich tych, którzy swoją radą i opinią kształtowali ostateczną wersję projektu oraz niniejszej dokumentacji (Moniko, Karolino, Grzesiu, Ksawery, Olku, Oskarze, Rudolfie - dzięki!).