POLITECHNIKA KRAKOWSKA

Interpreter tekstu do kodu w języku Python

Projekt z przedmiotu Automaty, Języki i Obliczenia

Ustawa o ochronie danych osobowych

Projekt realizujący aplikację wspomagającą rozmieszczenie gotowych elementów sterujących na scenie w programie Blender. Informacje o rozmieszczeniu i opcjach wprowadzane są jako tekst w języku naturalnym.

Spis treści

Wstęp

TWORZONA PRACA INŻYNIERSKA „Zestaw interaktywnych animacji 3D, opisujących wybrane zagadnienia z kursu grafiki komputerowej" wymaga prostych w obsłudze i zrozumiałych elementów. Używany w trakcie jej realizacji program Bledner najczęściej zapewnia w tym celu przystępne środowisko graficzne. Tworzenie interaktywnej animacji wymaga nie tylko utworzenia właściwej animacji ilustrującej dany problem lub zjawisko, ale także pewnych elementów sterujących pozwalających na ingerencję w przebieg animacji. Elementami sterującymi są między innymi: menu, suwak, pole wprowadzania danych. Dodatkowo można wprowadzić etykiety opisujące dany element sterujący lub całą animację. Tworzenie etykiet i elementów sterujących dla każdej animacji osobno byłoby czasochłonne, dlatego warto zaimportować gotowe elementy z pliku ustawiając odpowiednio ich parametry. Niestety import wszystkich obiektów sterujących i innych wymaganych przez nie danych jest skomplikowany. Aby uniknąć obciążania użytkownika zapamiętywaniem sposobu importu elementów sterujących można wygenerować skrypt który zaimportuje i ustawi obiekty automatycznie.

Cel projektu

Celem projektu jest zrealizowanie aplikacji zamieniającej informacje o rozmieszczeniu elementów sterujących zapisanych w języku naturalnym na skrypt w języku Python, który dokona tego rozmieszczenia w programie Blender. W ten sposób użytkownik nie musi pamiętać szczegółów dotyczących importu elementów sterujących.

Technologia

Ze względu na wieloplatformowość oraz graficzny interfejs zastosowano język Java.

Projekt systemu

Organizacja działania:

- Zdania są wprowadzane do aplikacji w postaci pliku tekstowego.
- Rozdzielenie na zdania i wyrazy
- Sprawdzenie poprawności zdań

- Tworzenie skryptu na podstawie prawidłowych zdań
- Zapis skryptu wynikowego

Plik wejściowy

Plikiem wejściowym powinien być plik tekstowy z kodowaniem znaków UTF-8. Zdania rozdzielane są kropką lub kropką i spacją. W zdaniu ~~na~~ powinna znajdować się informacja o elemencie jaki chcemy zaimportować następnie wymagane przez niego opcje lub słowa kluczowe oraz informacje dla modułu rozmieszczającego elementy. Przykłady prawidłowych zdań:

mała etykieta moc zlokalizowana po lewej u góry.
duże menu posiada opcje 1,2 zlokalizowane na środku na dole.

W dołączonym projekcie możliwymi elementami są etykieta i menu. Po słowie kluczowym etykieta następuje jej treść oraz położenie. Jeśli chcemy utworzyć menu, musimy użyć słowa kluczowego opcje aby znajdujące się za nim – oddzielone przecinkami – fragmenty stały się osobnymi opcjami.

Moduł

Wiele złożonych elementów projektu można połączyć w moduły tak aby aplikacja była łatwiejsza w rozbudowie. Każdy moduł można podzielić na dwie główne części:

- Sprawdzenie poprawności fragmentu zdania pod kątem danego modułu
- Generowanie fragmentu skryptu na podstawie prawidłowego zdania

Możliwość używania wskaźników do funkcji pozwoliłaby na użycie obiektów jednej klasy moduł wraz z dedykowaną dla tego obiektu funkcją generującą wpisy. Jednak Java nie pozwala na takie użycie dlatego każdy moduł definiowany jest jako osobna klasa.

Informacje wymagane przez moduł:

- Deterministyczny automat skończony
  - Tablica przejść automatu
  - Tablica stanów akceptujących
  - Tabela wartości dla pojawiających się na wejściu wyrazów
- Klasa reprezentująca moduł
  - Funkcja dokonująca wpisów do skryptu na podstawie podanych argumentów

Moduł location:

Odpowiada za pobranie informacji o położeniu elementu. Reaguje na słowa typu góra/dół lewo/środek/prawo. Środek sceny umyślnie nie może być położeniem żadnego elementu aby nie zasłonić właściwej animacji. Rysunek przedstawia możliwe położenia na scenie

Moduł Main:

Odpowiada za pobranie informacji o obiekcie jaki będzie importowany. Dodatkowo moduł jest potrzebny przy ustalaniu modułów używanych przez dane zdanie. W trakcie generacji skryptu moduł odpowiada za zaimportowanie obiektu z podanego w kodzie pliku.

Moduł Menu:

Usuwa z argumentu wejściowego słowo kluczowe „wpisy" oraz „opcje". Dla każdego wpisu (wpisy oddzielone są przecinkiem) generuje własność (property) i dodaje ją do wskazanego w wywołaniu obiektu)

Moduł Etykieta:

Ustawia własność *title* obiektu będącego argumentem wywołania na wartość będącą kolejnym argumentem. Dodaje kod ustawiający tą własność do skryptu wynikowego.

Integracja modułów

Aby zintegrować moduły należy je umieścić w tablicy wskazując element startowy. W przypadku tego projektu modułem startowy jest moduł importujący obiekty do sceny programu Bledner oraz wstawiający kolejny moduł do tablicy. (w zależności od tego jaki moduł pojawił się na jego wejściu) Jeśli wszystkie moduły znajdą się w stanie akceptującym, takie zdanie zostaje uznane za poprawne i wysłane do funkcji w klasie odpowiedzialnej za przydział odpowiednich fragmentów tekstu do funkcji w odpowiedniej klasie. Funkcję rozdzielającą pełni klasa dispacher – należy w niej „zarejestrować" moduł. W tym celu należy dodać odpowiedni wpis do tabeli oraz wywołanie funkcji dla wybranych fragmentów zdania w klasie modułu.

Plik wynikowy:

Wynikiem działania programu jest skrypt podobny do przykładowego:

```
#CZĘŚĆ WSPÓLNA DLA WSZYSTKICH ELEMENTÓW STERUJĄCYCH
    #IMPORT WYMAGANYCH BIBLIOTEK
    IMPORT BLENDER
    FROM BLENDER IMPORT SCENE, OBJECT, CAMERA
    IMPORT BPY

    #UTWORZENIE KAMERY I USTAWIENIE JEJ PARAMETRÓW
    CAMDATA = CAMERA.NEW('PERSP')
    CAMDATA.NAME = 'HUDCAM'
    CAMDATA.LENS = 16.0


    SCENE = SCENE.NEW('HUD')                    # UTWORZENIE NOWEJ SCENY
    OB =SCENE.OBJECTS.NEW(CAMDATA, 'CAM')       # DODANIE KAMERY
    OB.SETLOCATION(0.0, 0.0, 4.0)               # USTAWIENIE POŁOŻENIA KAMERY
```

```
SCENE.MAKECURRENT()                    # USTAWIENIE POWYŻSZEJ SCENY JAKO
BIEŻĄCEJ

#CZĘŚĆ WYKONYWANA DLA KAŻDEGO ELEMENTU STERUJĄCECO
    LIB = BPY.LIBRARIES.LOAD('//FILE.BLEND')    # ZAŁADOWANIE PLIKU Z ELEMENTAMI
    PSEUDOOB = LIB.OBJECTS.APPEND('KAC')        # ZAŁADOWANIE DO OBIEKTU JEDNEGO
    ELEMENTU
    OB = SCENE.OBJECTS.LINK(PSEUDOOB)           # DODANIE DO SCENY ELEMENTU
    STERUJĄCEGO

    OB.GETPROPERTY("MASS").SETDATA(100)         # NADANIE WARTOŚCI DLA
                                                WŁASNOŚCI(PROPERTY) OBIEKTU
```

Tokenizacja

Aby prawidłowo odczytać plik wejściowy jest on w całości wczytywany do struktury StringBuilder. Jest to wymagane gdyż opcje elementu sterującego mogą mieścić się w więcej niż jednej linii. Dlatego założono że separatorem zdań będzie znak kropki (ze spacją lub bez. Jego użycie nie jest dozwolone w opcjach obiektu.

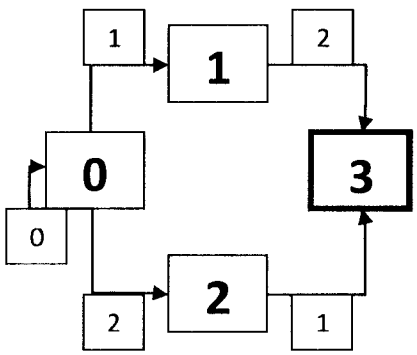
DFA

DFA - **deterministic finite automaton** czyli deterministyczny automat skończony, używany jest w programie do sprawdzenia czy podane zdanie zawiera wszelkie potrzebne informacje. Użycie jednego automatu utrudniłoby znacznie rozbudowę aplikacji dlatego dokonano podziału mechanizmu na moduły. Każdy moduł posiada własny mniejszy deterministyczny automat skończony. W trakcie pracy nad zdaniem definiowane są moduły użyte w tym zdaniu. Aby zdanie zostało zaakceptowane wszystkie użyte w nim moduły muszą się znaleźć w stanie akceptującym. Aby odseparować proces sprawdzania poprawności zdania od wykonywania wpisów w pliku (aby uniknąć problemu gdy błąd pojawia się na końcu zdania) w trakcie działania deterministycznego automatu skończonego do zdania wprowadzany jest znak „^" oznaczający zmianę stanu automatu. Pozwoli on później na odpowiednie rozłożenie zdania tak aby do danego modułu trafiły tylko potrzebne mu dane.

SŁOWA KLUCZOWE	WARTOŚĆ
GÓRY, GÓRZE, DOLE, DOŁU	1
LEWO, LEWEJ, PRAWO, PRAWEJ	2
POZOSTAŁE	0

STAN\WEJŚCIE	0	1	2
0	0	1	1
1	1	2	3
2	2	3	2

Tabele wykorzystywane przez deterministyczny automat skończony.



Schemat automatu dla modułu odpowiedzialnego za położenie elementu.

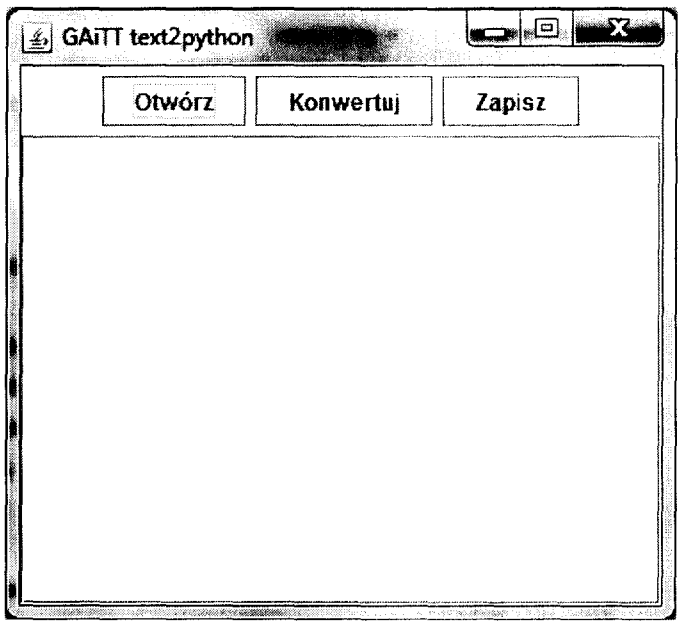
Realizacja Aplikacji

Obsługa programu

Główną cechą aplikacji miała być intuicyjna obsługa. GUI aplikacji składa się z trzech przycisków umożliwiających kolejno:

- Wczytanie pliku z instrukcjami dla programu
- Dokonanie translacji z języka naturalnego na skrypt pytona
- Zapis skryptu do pliku

Dodatkowo program wyświetla komunikaty informujące o tym czy zdanie przeszło analizę leksykalną.



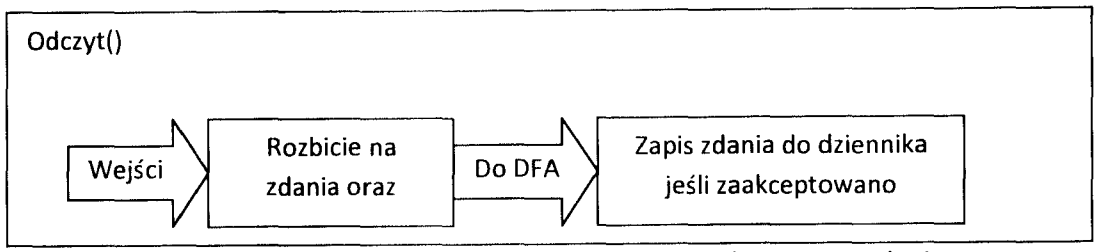
Założono że nie zawsze wynik translacji ma zostać zapisany. W szczególności przy wystąpieniu błędów, użytkownik ma możliwość poprawy pliku i uruchomienie translacji bez zamykania aplikacji.

Schemat budowy aplikacji

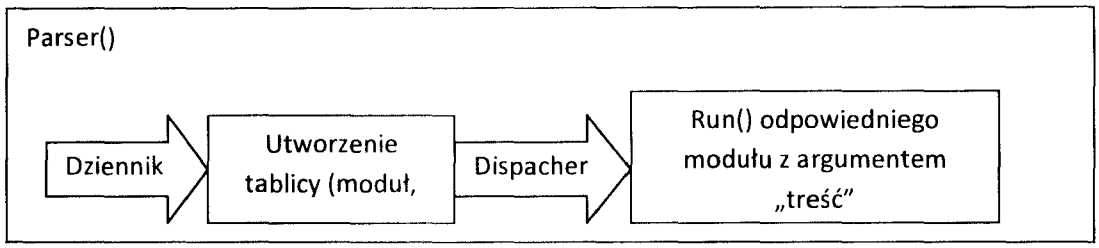
W trakcie realizacji główny nacisk kładziono na jej uniwersalność. Niestety uniwersalność często pociąga za sobą trudności w konfiguracji. Mimo użycia budowy modułowej konfiguracja tej aplikacji do własnych potrzeb nie należy do prostych.

Poniżej przedstawiono ogólny schemat budowy aplikacji.

Główna część aplikacji to funkcje Odczyt(), Parser() i Write() – ich dokładniejszy opis znajduje się poniżej. Pominięto funkcje dotyczące budowy GUI



Funkcja odczyt() odpowiada za pobranie zawartości pliku i rozbicie jej na zdania oraz wyrazy. Każdy wyraz ze zdania trafia do bieżącego automatu – początkowo jest to automat startowy ale w trakcie pracy jest od zmieniany na właściwy automat modułu. Zmiana automatu odbywa się w oparciu o słowa kluczowe (np. pojawienie się słowa „zlokalizowane" powoduje przełączenie bieżącego automatu na ten należący do modułu location) W trakcie przetwarzania zdania wypełniania jest tablica automatów używanych w tym procesie. Gdy całe zdanie zostanie przetworzone następuje sprawdzenie czy wszystkie automaty w tablicy znajdują się w stanie akceptującym. Jeśli tak- zdanie trafia do dziennika wraz z dodatkowymi symbolami zaznaczającymi kiedy nastąpiła zmiana stanu automatu oraz nazwami modułów odpowiedzialnymi za dany fragment.



Funkcja parser() pobiera dane z dziennika. Dla każdego zdania tworzy (za pomocą wyrażeń regularnych) tablicę zawierającą nazwę modułu oraz fragment zdania którego ten moduł potrzebuje. Następnie klasa dispacher na podstawie swojej informacji o modułach oraz fragmentach które są przez moduł wymagane, uruchamia funkcję run() odpowiedniego modułu z fragmentem zdania jako argumentem. Funkcja run() każdego modułu odpowiada za prawidłową zamianę fragmentu zdania na kod w języku pyton. Kod uzyskany z wszystkich użytych funkcji run() zostanie dopisany do skryptu.

Funkcja write() odpowiada za zapis wcześniej uzyskanego kodu oraz fragmentów wspólnych dla wszystkich modułów (np. ustawienie kamery i wybór pliku z którego nastąpi import). Jeśli nie będzie żadnych informacji z modułów (wszystkie zdania nieprawidłowe lub nie wczytano pliku) funkcja zapisze do pliku część wspólną.

Użycie aplikacji

Aby użyć aplikacji należy ją umieścić w tym samym folderze co plik file.blend (zawierający elementy sterujące) Następnie wczytać plik z instrukcjami, dokonać konwersji i zapisać wynik. Skopiować zawartość wynikowego pliku tekstowego i wkleić do pustego pliku w oknie „Text Editor" programu Blender. Wybrać z menu text opcję „Run Python Script". W oknie 3dView powinna pojawić się nowa scena z elementami. Aby zaobserwować rozmieszczenie elementów należy wybrać widok z kamery (domyślnie num0).

Sposób rozbudowy projektu

Aby dodać własny moduł należy do źródła projektu dodać własną klasę. Własna klasa powinna posiadać co najmniej atrybut name typu String określający nazwę modułu oraz funkcję której argumentami będzie nazwa obiektu i argumenty których wymaga ten moduł. Argumenty można przekazać jako String lub tablicę typu String – w zależności od potrzeb. Następnie uzupełnić tablicę w pliku test.java. Kolejnym etapem jest wprowadzenie informacji w klasie dispacher tak aby odpowiednie fragmenty zdania były przekazywane do zadanej funkcji w nowej klasie modułu.

Propozycje rozbudowy projektu

Dostarczony projekt można rozbudować o nowe moduły oraz łatwo zmodyfikować obecne.

- Projekt nie w pełni obsługuje wszystkie formy gramatyczne i odmiany słów powodując nadmierne restrykcje nałożone na plik wejściowy. Rozwiązaniem tego problemu jest dopisanie wyrazów do list modułów oraz wprowadzenie lematyzacji.
- Wygodną funkcją byłoby kopiowanie wynikowego skryptu bezpośrednio do schowka
- Aplikacja może być bardziej elastyczna jeśli wprowadzi się wybór pliku z elementami sterującymi (obecnie plik file.blend musi znajdować się w katalogu w którym jest plik .blend interaktywnej animacji)
- Aplikację można w łatwy sposób umieścić jako usługę na stronie internetowej wraz z plikiem zawierającym elementy, tak aby mniej doświadczeni użytkownicy Blendera nie byli zniechęcani zawiłością importu.

Literatura:

"Python w 24 godziny" - Ivan van Leningham

„Mastering Blender" – Tony Mullen

www.blender.org – działy dotyczące skryptów w języku Python oraz API języka Python dla blendera.