Artur Tkaczyk nr. Indeksu: 269356

JĘZYK OPISU SCEN GRAFICZNYCH

1. Funkcjonalność

- Język ma wbudowane funkcje tworzące podstawowe figury geometryczne: koło, prostokąt, trójkąt, wielokąt. W funkcjach tych podaje się również kolor wypełnienia figury.
- Utworzone obiekty można poddawać transformacjom geometrycznym: obrót, skalowanie, pochylenie, przeniesienie. Można też urzyć transform() do stworzenia macierzy przekształcenia
- Obiekty przypisuje się do zmiennych rozpoczynających się znakiem $.
- Można tworzyć obiekty złożone, czyli takie, które mogą się składać z wielu innych obiektów (w tym z innych obiektów złożonych).
- Można tworzyć animacje korzystające z utworzonych obiektów. Z każdym obiektem animacji związany jest jeden obiekt, który w określonych momentach w czasie osiąga określone transformacje geometryczne. Obiekt dokonuje transformacji między punktami czasowymi w sposób liniowy.
- Istnieje możliwość zapętlenia animacji.
- Język ignoruje białe znaki.
- Istnieje możliwość pisania komentarzy w kodzie źródłowym
- Żeby wyświetlić obiekt należy użyć polecenia : show $nazwa. Natomiast, jeżeli chcemy rozpocząć animację należy użyć polecenia : animate $nazwa.
- rectangle(lewy górny x, lewy górny y, szerokość, wysokość, kolor)
- circle(promień, kolor)
- triangle(współrzędne x i y wszystkich 3 punktów, kolor)
- polygon(współrzędne x i y wszystkich punktów, kolor)
- move(różnica x, różnica y)
- rotate(kąt)
- scale(skala x, skala y)
- shearx(współczynnik) , sheary(współczynnik)
- transform(6 współczynników do ustalenia w transformacji 2D)

2. Przykłady

- Ten kod utworzy grafikę domu, a następnie wyświetli go na ekranie

```
$sciany= rectangle(200, 200, 400, 400, rgb(0,255,0));
$dach = triangle(0, 300, 300, 200, -300, 200, rgb(255,0,0));
$drzwi = rectangle(-50, 100, 100, 300, rgb(0,0,0));
$okno = rectangle(-175, 100, 100, 100, rgb(0,0,255));
object house { $sciany, $dach, $drzwi, $okno, move(250, 0) * $okno };
show house;
```

- Ten kod stworzy prostą animację

```
object thing {
rectangle(-100, 100, 200, 200, rgb(0,0,255)),
circle(50, rgb(255,0,0))
};
animation thing2 {
loop : yes,
0 : move(-200, -200) * $thing,
1 : move(200, 200) * rotate(90) * move(-200, 200),
2 : move(200, -200) * rotate(90) * move(200, 200),
3 : move(-200, -200) * rotate(90) * move(200, -200),
```

```
4 : move(-200, 200) * rotate(90) * move(-200, -200)
};
animate thing2;
```

3. Formalna specyfikacja języka

Tekst znajduje się w oddzielnym pliku: grammar.txt

4. Uruchomienie

Nazwę pliku tekstowego z kodem źródłowym należy podać jako argument polecenia przy uruchamianiu, w przypadku jego braku, program sam się o niego dopyta.

5. Realizacja

- Program będzie napisany w języku Java.
- Klasa Lekser będzie posiadała funkcję getToken, która będzie zwracała następny token
- Wszystkie tokeny będą obiektami jednej z klas pochodnych od klasy Token
- Występujące w programie tokeny: zmienna, znak przypisania, słowa kluczowe: (transform, move, rotate, shearx, sheary, scale, rectangle, triangle, circle, polygon, object, animation, animate, show), znaki: (“;”, “:”, “,” , “(“ , “)”), liczba
- Klasa Token będzie posiadała podklasy: BasicToken (token, który nie zawiera żadnej dodatkowej informacji, przechowuje jedynie indentyfikator typu tokenu np. OBJECT_TOKEN, SCALE_TOKEN), VarToken (token, który oprócz identyfikatora posiada jeszcze String do przechowania nazwy zmiennej), NumberToken (token, który oprócz identyfikatora posiada jeszcze int do przechowania wartości liczbowej)
- Klasa Parser będzie posiadała funkcję getCommand która będzie zwracała obiekt klasy Command, która reprezentuje odrębne polecenie zakończone średnikiem
- Obiekt klasy Command zawiera drzewo rozbioru, wktórym każdy węzeł i liść to określone działanie np. przypisz, stwórz macierz transformacji
- Klasa Model pobiera z klasy Parser kolejne obiekty Command i wywołuje dla nich funkcję eval(), które wywołuje polecenie
- Klasa Variables która przechowuje wszystkie zadeklarowane zmienne i ich wartości
- Klasa Object przechowuje obiekt do narysowania
- Klasa Animation przechowuje pojedyńczą animację
- Klasa View odpowiada wyświetlanie sceny, jest wywoływana na polecenie klasy Model
- Klasa Start tworzy obiekty Model i View
- Klasa Error, która zawiera informację na temat powstałego błędu. Jest wyrzucany przez klasy: Lekser, Parser lub Model. W przypadku wystąpienia błedu wyświetlana jest informacja o błedzie i szczegółach o nim, a następnie program kończy swoje działanie. Program kończy działanie już po pierwszym wykrytym błędzie.

6. Testowanie

- Wywołam interpreter dla kilku napisanych skryptów zgodnych z gramatyką języka i sprawdzę czy efekt będzie taki sam jak oczekiwany
- Dla kilku skryptów wyświetlających animację sprawdzę, czy wyświetla się ona płynnie
- W stworzonych skryptach wprowadzę celowo różnego rodzaju błędy i sprawdzę czy program je wyłapuje i czy wyświetla odpowiednie komunikaty o błędzie.