Dokumentacja języka “Shaper”

Instalacja i uruchamianie

Instalacja

Wymagania systemowe: system operacyjny Windows

Kompilator wraz z maszyną wirtualną można zainstalować korzystając z instalatora dostępnego na repozytorium w folderze *Release*.

Istnieje również możliwość uruchamiania programu z plików źródłowych. Należy wtedy zapewnić potrzebne biblioteki:

Kompilator (Python)
- antlr4-python3-runtime 4.9.3

Maszyna stanowa (C++)
- biblioteka SFML

W przypadku maszyny stanowej, należy pamiętać aby jej plik wykonawczy znajdował się w tym samym folderze co pliki *.dll*, są generowane przez instalator lub dostępne wśród plików źródłowych maszyny stanowej

Uruchamiane

Pliki z kodem napisanym w języku **Shaper** kompilowane są do kodu pośredniego. W zależności od sposobu instalacji, bytecode można wygenerować za pomocą:

Instalator:
pliku wykonawczego *Shaper.exe* znajdującego się wewnątrz folderu *Compiler.*
Kod źródłowy:
skryptu python’owego *Shaper.py* wewnątrz folderu *Compiler*

W obu przypadkach pod flagą *--help* można znaleźć dostępne opcje uruchomienia kompilatora

Po skompilowaniu do bytecode użytkownik w celu rozpoczęcia działania programu musi uruchomić maszynę wirtualną języka shaper (SVM: *Shaper Virtual Machine*) przekazując skompilowany plik programu jako parametr wywołania..
Przykładowe najprostsze uruchomienie programu zapisanego w języku shaper
>SVM example.hae

Dodatkowo SVM posiada opcjonalne argumenty przekazywane podczas uruchomienia programu

-d uruchamia maszynę w trybie debugowania, jest wyświetlany aktualny stan stosu maszyny
-fps ustawia limit klatek na sekundę utworzonego okna (domyślna wartość 240, przekazanie 0 powoduje usunięcie limitu klatek na sekundę)
-h ustawia wysokość okna w pikselach (domyślna wartość 400 pikseli)
-m ustawia rozmiar pamięci SVM (domyślna wartość 300 podstawowych jednostek pamięci)
-s ustawia maksymalny rozmiar stosu SVM (domyślna wartość 100 podstawowych jednostek pamięci)
-w ustawia szerokość okna w pikselach (domyślna wartość 400 pikseli)

przykładowo wywołanie SVM z podanymi argumentami
>SVM example.hae -w 800 -fps 120
uruchomi program example z oknem o szerokości 800 pikseli wysokości 400 pikseli i z limitem klatek wynoszącym 120 klatek na sekundę

Założenia wstępne

W tej sekcji można znaleźć nasze założenia wstępne do języka, jak powinien działać oraz jakie opcje dostarczać użytkownikowi.

Shaper miał być językiem przeznaczonym do szybkiego i wygodnego tworzenia grafiki 2D statycznej jak i dynamicznej z wykorzystaniem figur. Użytkownik może dostosować parametry figur (pozycja, obrót, długość, itd.), a także tworzyć kształty bardziej zaawansowane za pomocą figur podstawowych.

Typy podstawowe

- **bool:** zmienna przechowująca wartości logiczne True, False
- **int:** zmienna przechowująca wartości całkowite w zakresie od -2,147,483,647 do 2,147,483,647
- **long:** zmienna przechowująca wartości całkowite w zakresie od -9,223,372,036,854,775,807 do 9,223,372,036,854,775,807
- **char:** zmienna przechowująca pojedynczy znak zapisany w formacie ascii o wartościach od 0 do 255
- **float:** zmienna przechowująca wartość zmiennoprzecinkową zapisana przy pomocy 4 bajtów
- **double:** zmienna przechowująca wartość zmiennoprzecinkową zapisaną przy pomocy 8 bajtów
- **color:** zmienna przechowująca wartość koloru zapisanego w formacie RGBA (*red*,*green*,*blue*,*alpha*) zapisanego przy pomocy 4 bajtów po jednym bajcie na kanał

Stałe

- **QUOTER_PI:** stała przechowująca wartość równą ćwiartce π z dokładnością do 7 cyfr po przecinku
- **HALF_PI:** stała przechowująca wartość równą połowie π z dokładnością do 7 cyfr po przecinku
- **PI:** stała przechowująca wartość równą π z dokładnością do 7 cyfr po przecinku
- **TWO_PI:** stała przechowująca wartość równą dwukrotności π z dokładnością do 7 cyfr po przecinku
- **EXP:** stała przechowująca wartość równą stałej Eulera Ɛ z dokładnością do 7 cyfr po przecinku
- 16 zdefiniowanych stałych kolorystycznych m.in **BLACK**, **RED**, **PURPLE**

Typy złożone

- **array:** tablica o stałej liczbie elementów pozwalająca na dynamiczną alokację pamięci, posiada informację o liczbie elementów
- **list:** typ pozwalający na dynamiczne dodawanie i usuwanie elementów, posiada informację o liczbie elementów
- **struct:** pozwala na agregację danych wszystkich typów

Operatory arytmetyczne

Operator	Nazwa operatora	Operator	Nazwa operatora
-	unarny minus	*****	mnożenie
+	dodawanie	**/**	dzielenie
-	odejmowanie	**%**	modulo
++	inkrementacja	−−	dekrementacja

Operatory przypisania

Operator	Nazwa operatora	Operator	Nazwa operatora
=	przypisanie	***=**	przypisanie z mnożeniem
+=	przypisanie z dodawaniem	**/=**	przypisanie z dzieleniem
-=	przypisanie z	**%=**	przypisanie z działaniem

	mnożeniem		modulo

Operatory logiczne i porównania

Operator	Nazwa operatora	Operator	Nazwa operatora
!	logiczne NOT	**<**	mniejsze niż
&	logiczne AND	**>**	większe niż
|	logiczne OR	**<=**	mniejsze lub równe
==	równe	**=>**	większe lub równe
!=	nierówne		

Słowa kluczowe

- **for**
- **while**
- **const**
- **return**
- **if**
- **elif**
- **else**
- **switch**
- **case**
- **break**
- **continue**
- **void**
- **global**

Funkcje rysowania

- **line(x1, y1, x2, y2, color):** rysuje linię pomiędzy dwoma punktami
  - **x1:** pozioma współrzędna punktu pierwszego
  - **y1:** pionowa współrzędna punktu pierwszego
  - **x2:** pozioma współrzędna punktu drugiego
  - **y2:** pionowa współrzędna punktu drugiego

- **triangle(x1, y1, x2, y2, x3, y3, color):** rysuje trójkąt pomiędzy punktami
    - **x1:** pozioma współrzędna punktu pierwszego
    - **y1:** pionowa współrzędna punktu pierwszego
    - **x2:** pozioma współrzędna punktu drugiego
    - **y2:** pionowa współrzędna punktu drugiego
    - **x3:** pozioma współrzędna punktu trzeciego
    - **y3:** pionowa współrzędna punktu trzeciego

- **rectangle(x, y, width, height, color):** rysuje prostokąt o podanych parametrach
    - **x:** pozioma współrzędna lewego dolnego wierzchołka figury
    - **y:** pozioma współrzędna lewego dolnego wierzchołka figury
    - **width:** szerokość prostokąta
    - **height:** wysokość prostokąta
    - **color:** kolor wypełnienia

- **square(x, y, size, color):** rysuje kwadrat o podanych parametrach
    - **x:** pozioma współrzędna lewego dolnego wierzchołka figury
    - **y:** pozioma współrzędna lewego dolnego wiezchołka figury
    - **size:** rozmiar kwadratu
    - **color:** kolor wypełnienia

- **circle(x, y, radius, color):** rysuje koło o podanych parametrach
    - **x:** pozioma współrzędna środka figury
    - **y:** pionowa współrzędna środka figury
    - **color:** kolor wypełnienia

- **ellipse(x, y, width, height, color):** rysuje elipsę o podanych parametrach
    - **x:** pozioma współrzędna środka figury
    - **y:** pionowa współrzędna środka figury
    - **width:** promień elipsy wzdłuż osi X
    - **height:** promień elipsy wzdłuż osi Y
    - **color:** kolor wypełnienia

Funkcje konfiguracyjne

- **canvas(int width, int height):** ustawia rozmiar okna
    - **width:** szerokość okna
    - **height:** wysokość okna

- **frameRateLimit(int limit):** ustawia odgórny limit klatek na sekundę
    - **limit:** limit klatek na sekundę, wartość dodatnia

- **noLoop():** zatrzymuje zapętlanie się funkcji *draw()*, okno programu zostaje otwarte z widoczną ostatnio narysowaną klatką, w przypadku uruchomienia tej funkcji w *setup()* obraz zostanie narysowany tylko raz

- **background(color c):** ustawia kolor tła
    - **c:** kolor tła

- **scale(float s):** ustawia współczynnik skalowania wzdłuż osi
    - **s:** współczynnik skalowania

- **scale(float x,float y):** ustawia współczynniki skalowania wzdłuż odpowiednich osi
    - **x:** współczynnik skalowania wzdłuż osi X
    - **y:** współczynnik skalowania wzdłuż osi Y

- **rotate(float angle):** ustawia kąt obrotu w lewą stronę o podaną wartość
    - **angle:** kąt obrotu podany w radianach

- **translate(int x,int y):** przesuwa punkt 0,0 na wskazane wartości
    - **x:** przesunięcie wzdłuż osi X
    - **y:** przesunięcie wzdłuż osi Y

Szkielet programu

Każdy program musi składać się z dwóch funkcji o nazwach *setup* i *draw*. Funkcja *setup()* musi poprzedzać w kodzie funkcję *draw()*.
Pierwsza z tych funkcji uruchamia się przed rozpoczęciem rysowania okna (funkcja *draw()*). Należy w niej zamieścić pierwotne ustawienia, dostosowujące parametry rysowania obrazu. Funkcja *draw()* wykonuje się w pętli i odpowiada za rysowanie obiektów poprzez funkcje rysowania, a także można wywoływać funkcje konfiguracyjne. Aby zakończyć rysowanie należy:

- wpisać słowo kluczowe **return**, co spowoduje zakończenie działania całego programu wraz z zamknięciem rysowanego okna
- zamknąć rysowane okno, co również zakończy działanie programu
- użyć funkcji **noLoop()** mrożąc rysowany obraz na ostatnio widoczną klatkę, nie kończąc działania programu

Każda instrukcja musi być zakończona znakiem '**;**'.

Definiowanie własnych funkcji

ogólna postać definiowania funkcji wygląda następująco

```
zwracany_typ nazwa_funkcji ([lista_argumentów])
{
  ciało_funkcji;
}
```

zwracany_typ: typ danych zwracany przez funkcję w przypadku kiedy funkcja nic nie zwraca należy użyć słowa kluczowego *void*
nazwa_funkcji: unikatowa nazwa pozwalająca na jednoznaczne wywołanie funkcji
lista_argumentów: lista zawierająca argumenty przekazywane do funkcji. Każdy argument musi posiadać typ danych oraz unikatową nazwę w obrębie funkcji

ciało_funkcji: zawiera listę instrukcji opisujących co dana funkcja robi

Deklaracja i zasięg zmiennych

typ_zmiennej nazwa_zmiennej [= wartość];

typ_zmiennej: typ deklarowanej zmiennej
nazwa_zmiennej: unikatowa nazwa pozwalająca na jednoznaczne odwołanie się do zmiennej
wartość: opcjonalna wartość przypisana do zmiennej podczas jej tworzenia

Zmienna jest widoczna jedynie w bloku, w którym została zadeklarowana i w blokach zagnieżdżonych w tym bloku. Blokiem nazywamy obszar ograniczony parą nawiasów klamrowych, rozpoczyna się on nawiasem '**{**', a kończy '**}**'.

Aby utworzyć zmienną globalną, należy zadeklarować tę zmienną powyżej obowiązkowej funkcji *setup().*
Aby uzyskać dostęp do zmiennej globalnych należy wykorzystać słowo kluczowe **global::** nazwa_zmiennej

Co się jeszcze nie udało zrobić

Z powodu ograniczeń czasowych i zasobów ludzkich nie udało się wykonać wszystkich zaplanowanych funkcjonalności języka ***Shaper***.

Do najważniejszych niewykonanych założeń należą kontenery wbudowane takie jak *lista* czy *struktura*, dodatkowo brak funkcjonalności słów kluczowych *switch, break, continue*.

Dodatkowo język Shaper nie wspiera metod związanych z transformowaniem figur (funkcje *scale, rotate, translate*). Użytkownik również nie jest w stanie kontrolować rozmiaru okna z poziomu kodu, ale może ustawić je podczas uruchamiania wirtualnej maszyny (patrz sekcja *Instalacja i Uruchamianie*).

Jak to działa

Kompilator generujący kod pośredni działa w kilku fazach:

1. Sprawdzana jest poprawność składniowa kodu. W przypadku błędów, informacja o nich zostaje wyświetlona i program przestaje działać
2. Jeśli składnia jest poprawna, następuje sprawdzanie kodu poprzez *Visitor* pod względem semantycznym. Dodatkowo w trakcie tego kroku zbierane są informacje o

zdefiniowanych przez użytkownika funkcjach. W przypadku błędów, informacja o nich zostaje wyświetlona oraz program przestaje działać nie generując żadnego artefaktu

3. Jeśli nie zostały odnalezione żadne błędy składniowe i semantyczne, kod jest ponownie odwiedzany przez inny *Visitor* w celu wygenerowania kodu pośredniego. Można tu rozpoznać kilka kroków:
    - Generowanie kodu deklarującego zmienne globalne
    - Generowanie kodu wywołania funkcji językowych (*setup* i *draw)*
    - Generowanie kodu będącego ciałem zdefiniowanych funkcji
    - Wypełnienie informacji o docelowych miejscach skoku w wyrażeniach warunkowych i pętlach, a także pozycje w kodzie wywoływanych funkcji
4. Zapisywanie wygenerowanych poleceń do pliku z kodem pośrednim przechowywanym w postaci bytecode.

Pliki języka (***.spr***) są kompilowane do postaci bytecode które bezpośrednio działają na SVM (*Shaper Virtual Machine*). Dodatkowo cechą rozróżniającą język Shaper jest sposób interpretowania współrzędnych, bazuje ono na matematycznej formie określania współrzędnych. Lewy dolny róg posiada zerową współrzędną poziomą i pionową, które rosną w kierunku prawego górnego wierzchołka. Zmiana ta powoduje, że ustawianie pozycji figur jest naturalniejsze, co z kolei zapobiega wielu prostym błędom. Samo działanie SVM bazuje na działaniu maszyny wirtualnej opartej o stos gdzie podstawową jednostką zapisu jest int 32 bitowy.