TKOM – Dokumentacja końcowa

# Język opisu scen graficznych

Mateusz Plesiński

## Opis

Celem projektu jest stworzenie prostego języka programowania umożliwiającego przedstawianie scen graficznych. Język zapewnia procedury rysujące figury geometryczne, wyrażenia warunkowe, pętle (typu for, for_each) , wykonywanie funkcji co pewien określony czas itp.

## Uruchamianie

Do budowy programu używane jest narzędzie cmake. Aby wygenerować pliki do kompilacji w katalogu głownym programy uruchamiamy polecenie cmake a następnie make (Linux) lub otwieramy projekt Visual Studio (Windows).

Program jako argument przyjmuje ścierzkę do pliku z kodem programu. Następnie interpretuje go i tworzy okno sceny graficznej za pomoca biblioteki OpenGL.

## Funkcjonalność:

- wbudowane procedury rysujące figury - prymitywy (box, sphere)
- wbudowane procedury umożliwiające manipulacje figurami (move, scale, chngcol)
- możliwość definiowania i używania własnych funkcji
- operator okresowy („ ~ ”) zapewniający możliwość powtarzania funkcji co pewien zadeklarowany czas
- podstawowe mechanizmy programowania tj. :
    - instrukcja warunkowa
    - pętle for i for_each
    - operatory matematyczne
    - operatory logiczne
    - operatory przyrównania
    - nawiasowanie

## Wymagania funkcjonalne:

- Kontrola poprawności podawanych danych oraz zgłaszanie wykrytych błędów
- Poprawne odczytywanie, parsowanie i analiza skryptów z plików tekstowych
- Generacja scen 3D ze skryptów

## Wymagania niefunkcjonalne:

- Komunikaty o błędach podczas procesu analizy pliku powinny być zrozumiałe dla użytkownika i jasno wskazywać popełnione błędy

## Specyfikacja techniczna

Projekt będzie wykonany w języku C++ (standard C++11). Przewiduję wykorzystanie zewnętrznych bibliotek do testów jednostkowych Catch2, konfiguracji procesu budowania zostanie wykorzystany program cmake.

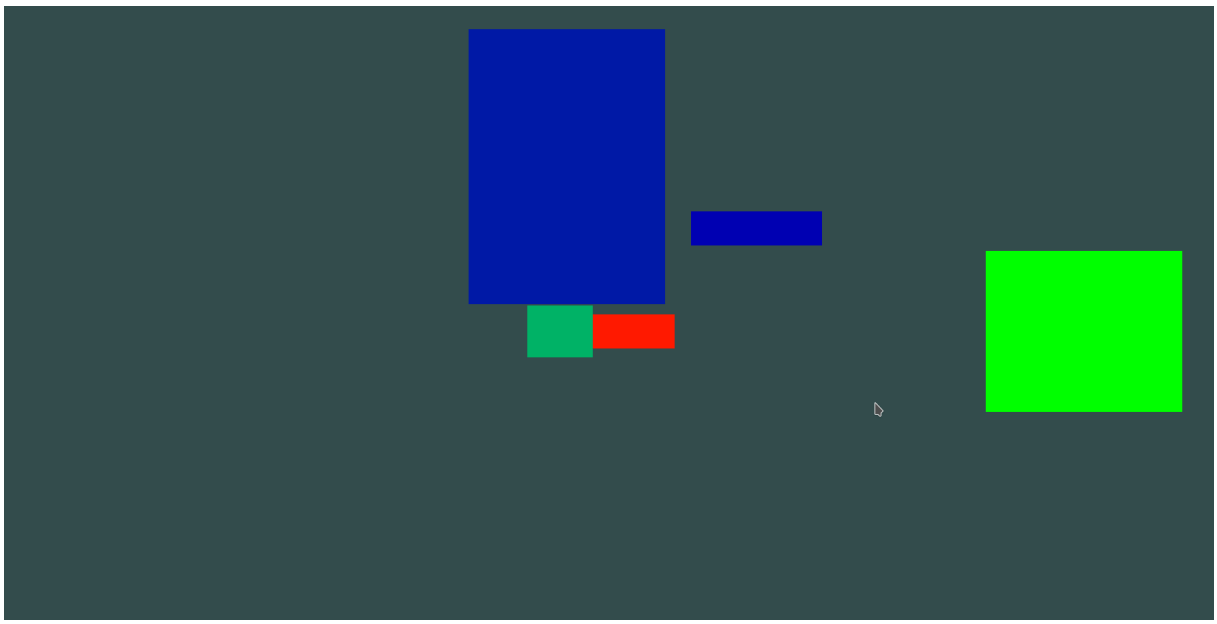
## Przykłady

```
func tkom(a) {
    return a+5;
}
func scene() {
    a=2;
    tkom(a);
    boks draw box pos(0,2,60) col(0,70,40) dim(10,15,17);
    boks1 draw box pos(80,2,60) col(0,100,0) dim(30,47,100);
    boks2 draw box pos(1,50,20) col(0,10,65) dim(30,80,100);
    boks3 draw box pos(10,2,60) col(100,10,0) dim(15,10,10);
    boks4 draw box pos(30,32,60) col(0,0,70) dim(20,10,1);
    cyl draw cylinder pos(50,0,1) col(1,1,0) dim(123,43,15);
    boks move (50 ,50,1);
    boks scale(5,1,1);
    return a;
}
```

```
1. func_moja_funkcja(figura) {
2.     figura move (10, 0, 0);
3. }
4. func_scene (){
5.     szescian  draw box pos(100, 100, 100) col(100, 0, 0) dim(10, 10, 10);
6.     moja_funkcja(szescian) ~(10);
7.     i = 0;
8.     szesc_kolek[10] draw box pos(50, 50, 25) col(0, 0, 0) dim(10, 10, 10);
9.     j;
10.    for (j = 0; j < 10; j = j + 1) {
11.        if (j < 5) {
12.            szesc_kolek[j]move (0, 20, 0);
13.        } else {
14.            szesc_kolek[j]move (0, 80, 0);
15.        }
16.    }
17. }
```

```
1. func_skaluj(figura) {
2.     figura scale (10, 10, 10);
3. }
4. func_scene {
5.     cyl draw cylinder pos(50, 50, 0) col(100, 0, 0) dim(100);
6.     skaluj(cyl);
7.     prostokat draw box pos(10, 10, 0) col(125, 50, 0) dim(10, 10, 0);
8.     for (j = 0; j < 100; j = j + 1) {
9.         if (j < 50) {
10.            prostokat scale (0, 2, 0);
11.        } else {
12.            prostokąt scale (0, 5, 0);
13.        }
14.    }
15. }
```

## Przykład uruchomienia:



## Gramatyka

**Program** = {  FuctionDeffinition } .
**FunctionDeffinition** = ‘func’ , Identifier , ‘(’ , [ Parameters ]  , ‘)’ , Block .
**Block** = “{” , { Instruction | Block } , “}” .
**Instruction** = { ConditionalStatment , “;” | LoopStatment , “;”|
InitializationStatment , “;” |    AssignStatment , “;” | FunctionCall , “;” |
GraphicFunction , “;”| ReturnStatment , “;” } .

**GraphicFunction** = Identifier , [Index]  , Operation , [FigureType] ,
(FigureAttributes | NewVector).
**Operation** = “draw” | “move” | “chngcol” | “scale” .
**NewVector** = “(“ , Assignable  ,  “,” , Assignable , “,’ , Assignable , “)” .
**FigureType** = “box” | “cylinder” .
**FigureAttributes** = Position , Color , Dimensions .
**Position** = “pos” , “(“ , Assignable  ,  “,” , Assignable , “,’ , Assignable , “)” .
**Color** = “col” , “(“ , Assignable  ,  “,” , Assignable , “,’ , Assignable , “)” .
**Dimensions** = “dim” , “(“ , Assignable  ,  “,” , Assignable , “,’ , Assignable , “)” .

**ReturnStatment** = “return” , Assignable
**ConditionalStatment** = “if ” , “(” , Condition , “)” , Block,  [ “else” , Block] .
**LoopStatment** = ForLoop | ForEachLoop .
**ForLoop** = “for” , “(” , (AssignStatment ) , “;” ,
Condition , “;” , AssignStatment,   “)” Block .
**ForEachLoop** = “for_each” , Identifier , “:” , Identifier , Block .
**InitializationStatment** = Identifier , [Index] ,  [ “=” Assignable ] .
**AssignStatment** = Identifier , [Index] , “=” , Assignable .

**FunctionCall** = SimpleFunctionCall | PeriodicFUnctionCall .
**SimpleFunctionCall** = Identifier “(” Parameters “)” .

**PeriodicFunctionCall** = SimpleFunctionCall [ PeriodicOperator “(“ Assignable “)” ] .
**Condition** = AndCndition , { OrOperator , AndContition } .
**AndCond** = EqualityCOndition , {AndOperator , EqualityCondition}
**EqualityCondition** = RelationalCondition , [ EqualOperator , RelationalCondition ]
**RelationalCondition** = SimpleCondition , [ RelativeOperator ,SimpleCondition ]
**SimpleCondition** = [NegOperator] , ( Assignable )

**Parameters** = { Identifier } .
**Assignable** = Expression | FunctionCall .
**Expression** = MultipExpresion { AddtivOperator , MultipExpresion} .
**MultipExpresion** = SimpleExpresion . { MultipOpperator , SimpleExpresion} .
**SimpleExpresion** = Variable | Number | ParentExpresion .
**ParentExpresion** = “(” , Expresion , “)” .
**Variable** = Identifier , [ Index ] .
**Index** = “[“ , Assignable , “]” .

**NegOperator** = “!” .
**OrOperator** = “or” .
**AndOperator** = “and” .
**EqualOperator** = “==” | “!=” .
**RelativeOperator** = “<” | “<=” | “>” | “>=” .
**LogicalOperator** = “or” | “and” .
**PeriodicOperator** = “~” .
**AddtivOperator** = “+” | “-“
**MultipOpperator** = “*” | “/” .
**Number** = [ “-“ ] , Digit , { Digit } .
**Digit** = “0” ..”9” .
**Identifier** = Letter {Digit | Letter} .

**Letter** = “a”..”z” | “A”..”Z” | ”_” | ”-”.

## Lista zdefiniowanych tokenów

‘.’ , ‘,’ , ‘func’ , ‘(‘ , ‘(‘ , ‘{’ , ‘}’ , ‘[‘ , ‘]’ , ‘;’ , ‘:’ , ‘||’ , ‘&&’ , ‘!’ , ‘=’ , ‘==’, ‘!=’ , ‘~’ , ‘<’ , ‘>’ , ‘>=’ , ‘<=’ , ‘+’ , ‘-‘ , ‘/’ , ‘*’ , ‘if’ , ‘else’ , ‘return’ , ‘for’ , ‘draw’ , ‘scale’ , ‘move’, ‘chngcol’, ‘box’, ‘cylinder’, ‘pos’, ‘dim’, ‘col’