Mateusz Wasiak I.ISI 277354

TKOM – Język do wizualizacji

Dokumentacja Końcowa

1. Opis

Celem projektu jest stworzenie języka, pozwalającego na wizualizację obiektów. Język ten ma mieć możliwość defniowania własnych fgur przy pomocy funkcji oraz rysowania podstawowych kształtów

2. Funkcjonalność

2.1 Defniowanie funkcji
2.2 Instrukcje warunkowe
2.3 Pętle
2.4 Rysowanie podstawowych fgur geometrycznych (odcinek, kwadrat, koło, punkt)

3. Specyfkacja

3.1 Środowisko programistyczne

Projekt zostanie wykonany w języku C++. Wykorzystana zostanie biblioteka Qt do rysowania obiektów. Biblioteka cxxopts może zostać wykorzystana do pobierania argumentów z linii poleceń. Biblioteka catch2 może zostać wykorzystana do pisania testów jednostkowych. Aby skompilować program trzeba będzie użyć polecenia qmake, a następnie polecenian make.

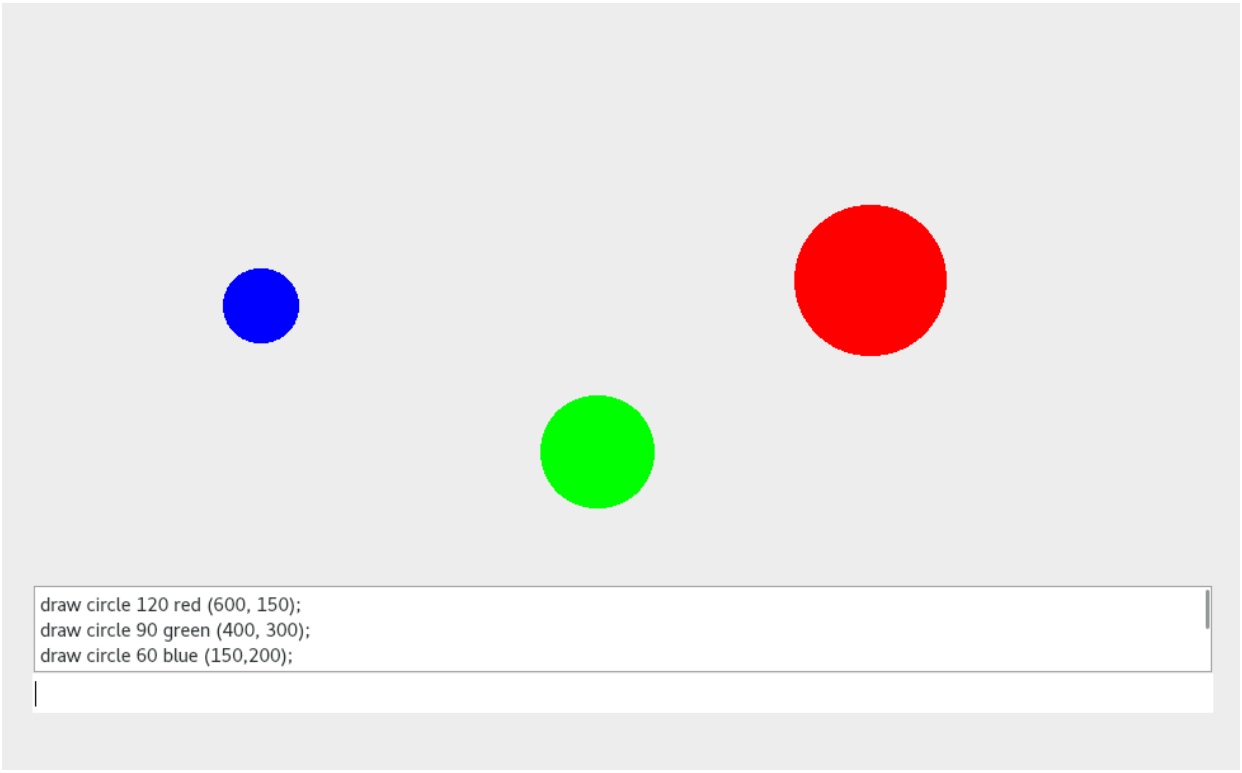
3.2 Interpreter

Interaktywny z nieskończoną pętlą, tzw. Read Evaluate Print Loop. Wczytane zostanie polecenie ze standardowego wejścia, wykonane, a wynik zostanie umieszczony na rysunku.

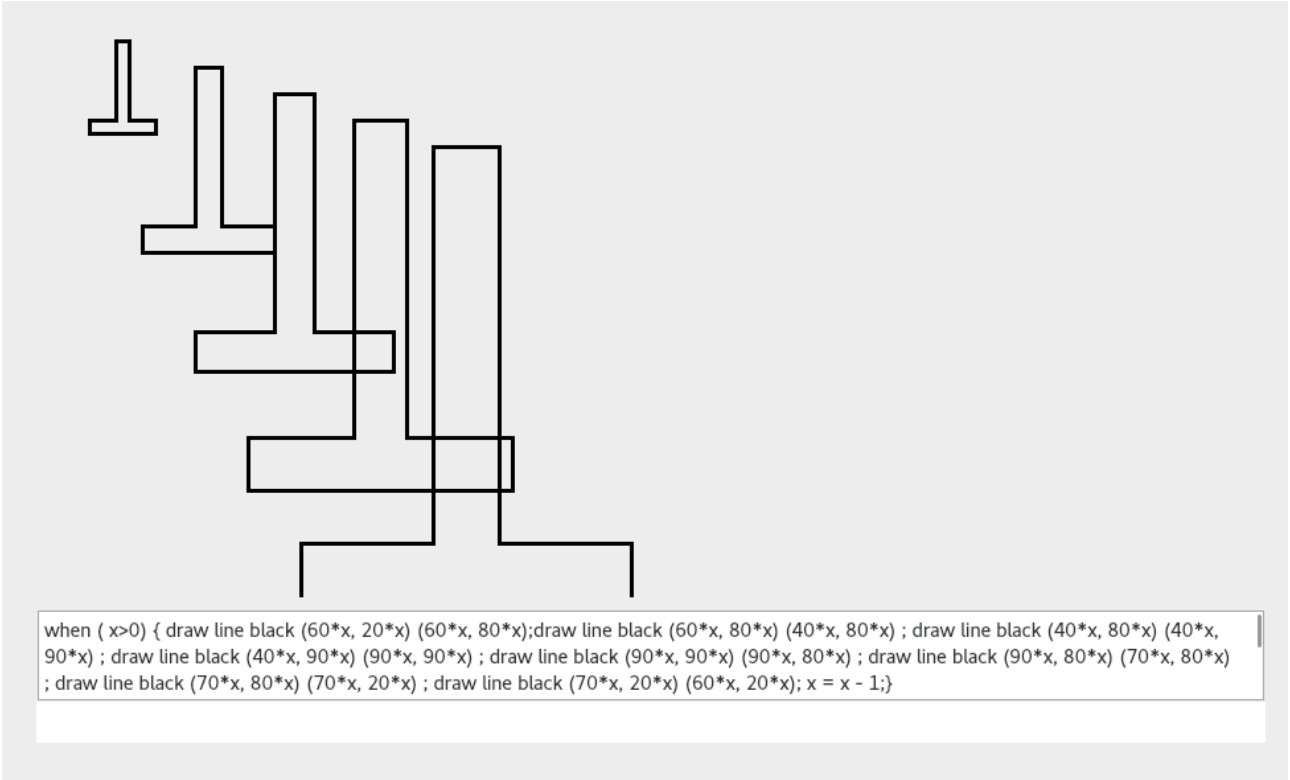
3.3 Typy

3.3.1 Integer
3.3.2 func
3.3.3 void

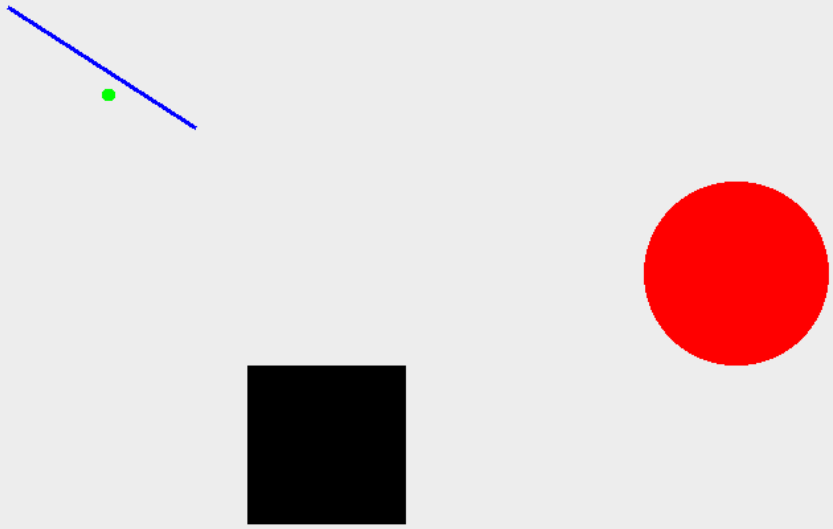
4. Przykłady:



```
draw circle 120 red (600, 150);
draw circle 90 green (400, 300);
draw circle 60 blue (150,200);
```



```
when ( x>0) { draw line black (60*x, 20*x) (60*x, 80*x);draw line black (60*x, 80*x) (40*x, 80*x) ; draw line black (40*x, 80*x) (40*x, 90*x) ; draw line black (40*x, 90*x) (90*x, 90*x) ; draw line black (90*x, 90*x) (90*x, 80*x) ; draw line black (90*x, 80*x) (70*x, 80*x) ; draw line black (70*x, 80*x) (70*x, 20*x) ; draw line black (70*x, 20*x) (60*x, 20*x); x = x - 1;}
```



```
draw point green (90, 90);
draw line blue (20,30) (160, 120);
draw circle 140 red (500, 160);
```

5. Gramatyka:

Opis gramatyki w notacji EBNF. Symbol startowy to input text.

statement = (assign_statement | loop_statement | if_statement | functon_literal | functon_apply | draw | clear_statement | exit_statement) ;
assign_statement = identifer, ‘=’, add_expr, ‘;’ ;
if_statement = ‘if’, conditon, operation_block ;
loop_statement = ‘when’, conditon, operation_block ;
functon_literal = ‘func’, identier, ‘(‘, [identifier_list], ‘)’ block ;
functon apply = identier, ’(‘ [parameter_list] ‘)’, ‘;’ ;
clear_statement = ‘clear’, ‘;’ ;
exit_statement = ‘exit’, ‘;’ ;
return_statement = ‘return’, expr, ‘;’ ;
block = ‘{’, {statement}, [return statement] ‘}’ ;
operation_block = ‘{’, {statement}, ‘}’ ;
alpha = ? A-Z or a-z ? ;
non-zero digit = ? 1-9 ? ;
digit = ‘0’ | non-zero digit ;
alphanum = alpha | digit ;
natural number = non-zero digit, {digit} ;
integer = ‘0’ | [‘-’], natural number ;
identier = (alpha | ‘_’), {alphanum | ‘_’} ;
mul_operator = ‘*’ | ‘/’ ;
add_operator = ‘+’ | ‘-’ ;
relation_operator = ‘<’ | ‘<=’ | ‘>’ | ‘>=’ ;
eq_operator = ‘==’ | ‘!=’ ;
and_operator = ‘&&’;
or_operator = ‘||’ ;
conditon = ‘(‘, or_expr ‘)’ ;
positon = ‘(‘, add_expr, ‘,’, add_expr, ‘)’ ;
atom = integer | func apply | func literal | identier ;
predefined_fgure = quadrangle | point | circle | line ;
circle = ‘circle’, size, color, position ;
quadrangle = ‘quadrangle’, size, color, position ;
point = ‘point’, color, position
line = ‘line’, color, position, position ;
draw = ‘draw’, predefined fgure , ‘;’ ;
size = integer ;
color = ‘black’ | ‘red’ | ‘blue’ | ‘green’ ;
expr = (add_expr | or_expr) ;
add_expr = mul_expr {add_operator, mul_expr} ;
mul_expr = atom {mul_operator, atom} ;
or_expr = and_expr {or_operator and_expr} ;
and_expr = eq_expr {and_operator eq_expr} ;
eq_expr = rel_expr {eq_operator rel_expr} ;
rel_expr = atom {rel_operator atom} ;
parameter_list = expr {‘,’ expr} ;
identifier_list = identifier {‘,’ identifier} ;