[TKOM] LOGO

Adamski Maciej

Kwiecień 2021

1 Wstęp/Opis

Celem projektu jest napisanie aplikacji okienkowej, która będzie podobna do znanego programu *Logomocja*. Na ekranie będą poruszać się żółwie wykonujące określone przez użytkownika w formie kodu zadania. Względem tradycyjnego *Logo*, ta wersja będzie działała na konstrukcjach obiektowych oraz równoległych.

2 Funkcjonalność

2.1 Tworzenie nowego żółwia

Tworzenie obiektu nowego żółwia, do którego później można się odwołać, a także modyfikować parametry. Obiekt złożony mający swoje metody, które można wywoływać na danym żółwiu.

```
1    Turtle zolw1;
2    Turtle zolw2;
```

2.2 Usuwanie żółwia

Usuwa obiekt żółwia z programu oraz z przestrzeni do rysowania.

```
1    delete zolw1;
2    delete zolw2;
```

2.3 Idź do przodu

Mówi żółwiowi, że ma poruszyć się do przodu o określoną ilość pikseli oraz zależnie od ustawień pędzla rysować pod sobą kreskę lub nie. Wymaga podania ilości kroków jako parametru.

```
zolw1.go(20); // 20 kroków do przodu
zolw2.go(500); // 500 kroków do przodu
```

2.4 Pozycja żółwia

Możliwość pobrania współrzędnych x, y aktualnej pozycji żółwia.

```
zolw1.pos.x; // zwraca współrzędną X
zolw1.pos.y; // zwraca współrzędną Y
```

2.5 Obrót

Mówi żółwiowi, że ma obrócić się o zadany kąt w lewo lub w prawo. Nie zmienia on swojej pozycji, a jedynie kierunek, w którym będzie się poruszał. Wymaga podania kąta, o który ma się obrócić żółw jako parametr.

```
zolw1.right(90); // obrót w prawo o 90 stoni
zolw2.right(-90); // obrót w prawo o -90 stopni, czyli w lewo o 90
zolw2.left(90); // obrót w lewo o 90 stopni
```

2.6 Pobranie kąta o jaki obrócony jest żółw

Możliwość pobrania wartości o jaki obrócony jest żółw względem początkowego ustawienia.

```
zolw1.direction; // zwraca kąt
```

2.7 Pętla

Konstruckja pozwalająca na wykonanie szeregu instrukcji wyznaczoną ilość razy. Należy podać ile razy pętla ma się wykonać, a także w bloku zdefiniować instrukcje, które mają się wykonać.

```
repeat(10) { // powtórz 10 razy kod w nawiasach klamrowych
    zolw1.go(10);
    zolw1.right(90);
}
```

2.8 Czyszczenie całego ekranu

Funkcja czyści cały ekran - usuwa wszystkie narysowane linie. Nie zmienia pozycji żółwia

```
clean();
```

2.9 Pędzel jako składowa żółwia

Pędzel będzie osobną klasą, która będzie składową żółwia. Będzie posiadała odpowiednie parametry, do których będzie można dostać się przez operator wyłuskania, pozwalając na pobranie wartości lub ustawienie nowej.

```
zolw1.brush.size; // zwróci grubość pędzla
zolw1.brush.color; // zwróci klasę kolor
zolw1.brush.enable; // zwróci informację czy pędzel jest włączony
    czy wyłączony
```

2.10 Zmiana grubości pędzla

Pozwala zmieniać grubość pędzla żółwia, na podany jako parametr. Grubość pędzla definiuje wielkość linii rysowanej przez żółwia.

```
zolw1.brush.size; // zwraca wielkość pędzla
zolw1.brush.setSize(10); // ustawia grubość pędzla na 10
zolw2.brush.setSize(50); // ustawia grubość pędzla na 50
```

2.11 Włączenie/wyłączenie pędzla

Pozwala włączyć lub wyłączyć pędzel, czyli zmienić czy podczas chodzenia żółw będzie rysował ślad czy nie.

```
zolw1.brush.enable; // zwróci informację czy pędzel jest włączony
zolw1.brush.enable = true; // żółw będzie rysować linię
zolw2.brush.enable = false; // żółw nie będzie rysować linii
```

2.12 Klasa koloru

Zdefiniowana zostanie klasa koloru, która będzie składową klasy pędzla. Będzie ona zawierała trzy elementy - składowe przestrzeni RGB.

```
zolw1.brush.color; // pobranie klasy kolor pędzla danego żółwia
```

2.13 Zmiana koloru pędzla

Pozwala zmienić kolor pędzla danego żółwia na podany jako argument, w postaci stringa i formacie heksadecymalny. Metoda rozłoży ciąg znaków na trzy składowe.

```
zolw1.brush.color.set("#000000"); // ustawia kolor pędzla na czarny
zolw2.brush.color.set("#ffffff"); // ustawia kolor pędzla na biały
```

2.14 Pobranie koloru pędzla

Będzie możliwość pobrania koloru pędzla w postaci stringa w formacie heksadecymalny. Zarówno jako cały kolor, jak i poszczególne składowe.

```
zolw1.brush.color.R; // string zawierający składową R
zolw1.brush.color.getHex(); // zwróci cały kolor w postaci
    heksadecymalnej
```

2.15 Pokazanie/ukrycie żółwia

Pozwala zmieniać widoczność żółwia na planszy. Nie ma to wpływu na to czy żółw maluje linię, czy nie - nawet ukryty będzie rysował przy włączonym pędzlu.

```
zolw1.hidden; // zwróci informację czy żółw jest widoczny
zolw1.hidden = false; // żółw będzie widoczny
zolw2.hidden = true; // żółw nie będzie widoczny
```

2.16 Tworzenie zmiennych zawierających współrzedne x, y

Pozwala zapisać współrzedne [x, y] w jednej zmiennej typu Punkt. Współrzędne mogą być inicjalizowane podczas deklaracji lub potem za pomocą odpowiednich metod.

```
Point punkt1(10, 10); // zmienna x=10, y=10
Point punkt2; // domyślnie x=0, y=0
punkt2.x; // pobierze współrzędną x punktu punkt2
punkt1.setX(20); // teraz x=20, y=20
```

2.17 Przeniesienie żółwia w dane miejsce

Pozwala przenieść wybranego żółwia we wskazane miejsce. Miejsce może być wskazane przez zmienną typu Punkt, dwie zmienne Integer lub dwie wartości liczbowe.

```
zolw1.move(punkt1); // przenosi żółwia w miejsce wskazane przez
    zmienną typu Punkt
zolw2.move(x, y); // przenosi żółwia w miejsce wskazane przez
    zmienne x i y
```

```
3    zolw3.move(20, 20); // przenosi żółwia w miejsce o współrzędnych
         podanych bezpośrednio
```

2.18 Przeniesienie wszystkich żółwi na punkt startowy

Przenosi wszystkie żółwie na środek planszy oraz ustawia domyslny kierunek ich poruszania.

```
1    allToStart();
```

2.19 Instrukcje warunkowe

Typowa konstrukcja warunkowa znana z różnych języków programowania. Pozwala zdefiniować szereg instrukcji, które wykonają się tylko w przypadku wystąpienia określonego warunku. Możliwe też jest dodanie bloku typu *else*.

```
1    if (x < 20) { // jeśli zmienna jest mniejsza od 20 to...
2        zolw1.go(x);
3    } else { // w pozostalych przypadkach
4        zolw.go(x - 20);
5    }
```

2.20 Operacje matematyczne w wyrażeniach

Pozwala na użycie operatorów matematycznych w wyrażeniach czy w wyołaniach funkcji.

```
1    zolw1.go(2 * 2);
2    zolw2.right(90 + 90);
3    repeat(20 - 15) {}
4    if ((x/2) == 2) {}
```

2.21 Tworzenie funkcji

Pozwala definiować własne funkcje z określonymi parametrami i zwracanym typem. Funkcja może przyjmować dowolną ilość parametrów oraz może zwracać wskazany typ lub żadnego. Do zdefiniowanych funkcji można się później odwołać.

```
1    function square(Turtle zolw) // definicja nowej funkcji square z
         parametrem
2    {
3        ...
4    }
```

```
    square(zolw1); // wywołanie nowej funkcji square

    Integer function retInt() // definicja nowej funkcji retInt bez
        parametrów, zwracająca typ Integer
    {
        return 1;
    }

    Integer testInt = retInt(); // testInt ma wartość 1
```

2.22 Zadania cykliczne

Pozwala na dodanie zadania cyklicznego wykonywanego co określoną ilość milisekund. Dodatkowo można ustalić ile razy ma się wykonać - jest to parametr opcjonalny.

```
    repeatTime(1000, 10) { // Zadanie wykona się 10 razy co sekundę
        zolw1.go(20);
    }
    repeatTime(1000) { // To samo zadanie, ale nie mające określonej
        ilości wykonań
        zolw1.go(20);
    }
```

3 Gramatyka

Gramatyka języka używanego w tym projekcie, wyrażona w notacji *EBNF*.

```
    program = { instruction };
    instruction = functionCall | functionDef | if | repeat | repeatTime
        | defAnyVar;



(* konstrukcje *)
    repeat = "repeat", "(", intPlus, ")", block;
    repeatTime = "repeatTime", "(", intPlus, [",", intPlus], ")", block;
    if = "if", "(", condition, ")", block, ["else", block];

    block = ( "{", { instruction }, [ return ]"}" ) | instruction |
        return;
   anyMemberLevel = id, { ".", id };


```

```
(* funkcje *)
    parameters = parameter, { ",", parameter };
    parameter = turtleVar | pointVar | stringVar | intVar;

    arguments = argument, { "," argument };
    argument = id | expression;

    functionCall = anyMemberLevel, "(", [ arguments ], ")",
        endInstruction;
    functionDef = [allTypes], "function", id, "(", [ parameters ], ")",
        block;

    return = "return", ( expression | condition ), endInstruction;

    condition = andCondition, { orOperator, andCondition };
    andCondition = relationCondition, { andOperator, relationCondition };
    relationCondition = [ notOperator ], ( ( expression,
        relationOperator, expression ) | booleanWord | ( "(", condition,
        ")" ) )


    expression = term, { addOperator, term };
    term = factor, { multiOperator, factor };
    factor = [ "-" ], ( ( "(", expression, ")" ) | id | int );



(* operatory *)
    orOperator = "||" | "or";
    andOperator = "&&" | "and";

    equalOperator = "==" | "!=";
    relationOperator = "<" | ">" | "<=" | ">=" | equalOperator ;

    addOperator = "+" | "-";
    multiOperator = "*" | "/";
    mathOperator = addOperator | multiOperator;

    assignOperator = "=";

    notOperator = "!";

    accessOperator = ".";



(* znaki *)
    digitWithoutZero = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" |
        "9";
    digitZero = "0";
```

```
61      digit = digitZero | digitWithoutZero;
62
63      lowerCase = "a"..."b";
64      upperCase = "A"..."B";
65      letter = lowerCase | upperCase;
66
67      symbol = "!" | "@" | "#" | "$" | "%" | "^" | "&" | "*" | "(" | ")" |
            "-" | "=" | "_" | "+" | "{" | "}" | "[" | "]" | ":" | ";" | "'"
            | "," | "." | "/" | "<" | ">" | "?" | "`" | "~";
68
69      alphanumeric = letter | digit | symbol;
70
71      booleanWord = "true" | "false";
72
73
74
75  (* zmienne *)
76      id = letter, {letter | digit | "_"};
77
78      turtleType = "Turtle";
79      turtleVar = turtleType, id;
80      turtleVarDef = turtleVar, endInstruction;
81
82      pointType = "Point";
83      pointVar = pointType, id;
84      pointVarDef = pointVar, "(", int, ",", int, ")", endInstruction;
85
86      string = '"', { alphanumeric | " "}, '"';
87      stringType = "String";
88      stringVar = stringType, id;
89      stringVarDef = stringVar, assignOperator, string, endInstruction;
90
91      int = digitZero | intPlus;
92      intPlus = digitWithoutZero, {digit};
93      intType = "Integer";
94      intVar = intType, id;
95      intVarDef = intVar, assignOperator, int, endInstruction;
96
97      boolType = "Boolean";
98      boolVar = boolType, id;
99      boolVarDef = boolVar, assignOperator, booleanWord, endInstruction;
100
101     defAnyVar = turtleVarDef | pointVarDef | stringVarDef | intVarDef |
            boolVarDef;
102     allTypes = turtleType | pointType | stringType | intType | boolType;
103
104
105
106 (* inne *)
107     comment = "//", {alphanumeric | " "};
```

```
    delete = "delete", id;
    endInstruction = ";";
```

4 Problemy i decyzje z nimi związane

4.1 Brak wyróżnionej funkcji startowej

Program nie będzie posiadał wyróżnionej funkcji startowej pokroju *main* czy *init*. Cały kod będzie wykonywany po kolei.
Wiąże się to z kolejnymi dwoma decyzjami:

- Brak definicji funkcji przed jej wywołaniem (Patrz Punkt 4.2)
- Brak definicji zmiennej przed jej użyciem (Patrz Punkt 4.3)

4.2 Brak definicji funkcji przed jej wywołaniem

Funkcja musi być zdefiniowana zanim może być użyta.
W przypadku próby wywołania funkcji, która nie została jeszcze zdefiniowana zostanie zwrócony odpowiedni błąd.

```
Poprawna konstrukcja:
    function xx() {};
    xx();

Niepoprawna konstrukcja:
    xx();
    function xx() {}; // lub całkowity brak tej linii
```

4.3 Brak definicji zmiennej przed jej użyciem

Nie można się odwołać do zmiennej, która nie została wcześniej zdefiniowana. W innym przypadku zostanie zwrócony błąd.

```
Poprawne konstrukcje:
    Integer xx = 20;
    use(xx);

Niepoprawne konstrukcje:
    use(xx);
    Integer xx = 20; // lub całkowity brak tej linii
```

4.4 Brak rzutowania Integer na Boolean

W programie nie będzie występowało rzutowanie typu **Integer** na typ **Boolean**.

5 Tokeny

W projekcie będą rozróżniane następujące tokeny:

- **repeat**
- **repeatTime**
- **delete**
- **function**
- **if**
- **else**
- **{**, **}**
- (,)
- +, **-**, *****, /
- **,**
- **;**
- ==, **!**=, <, >, <=, >=
- **and**, **&&**, **or**, ||
- **true**, **false**
- **String**, **Integer**, **Turtle**, **Point**, **Boolean**
- **return**

6 Dostępne typy zmiennych

W programie będą dostępne następujące typy zmiennych:

- **Turtle** - klasa reprezentująca żółwia
- **Point** - klasa reprezentująca punkt kartezjański na planszy
- **String** - ciąg znakowy
- **Integer** - liczba całkowita
- **Boolean** - typ logiczny *true/false*

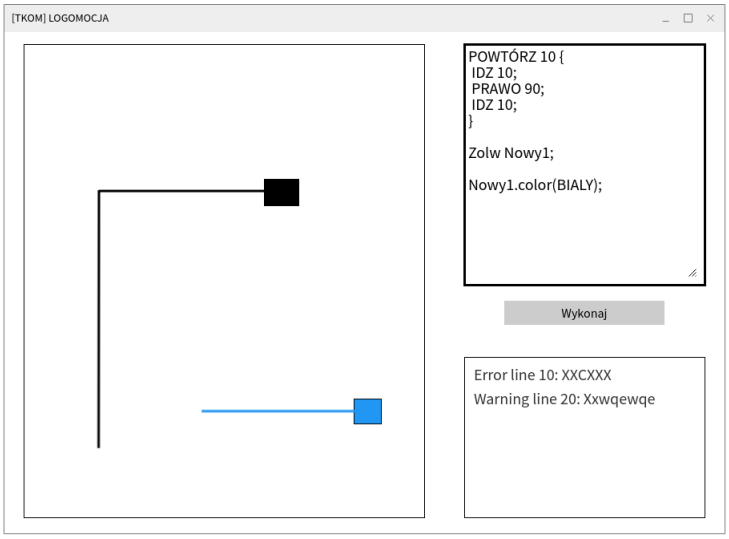
7 Obsługa błędów

Obsługą błędów zajmie się oddelegowany do tego moduł. Moduł ten będzie wykorzystywany na każdym etapie przetwarzania programu.
Komunikat o błędzie zawierał będzie takie informacje jak linia, w której wystąpił oraz prawdopdobną przyczynę.
Błędy będą wyświeltane w odpowiednim polu GUI.

8 Interfejs graficzny

GUI zostanie stworzone z użyciem *QT*. Okno zostanie podzielone na 3 części:

- Okno rysowania, w którym będą wyświetlane żółwie oraz to co narysują
- Pole tekstowe, w którym będziemy wpisywać kod, który ma się wykonać
- Pole tekstowe, w którym będą wyświetlane wyniki przetwarzania kodu



Rysunek 1: Prototyp wyglądu GUI

9 Sposób uruchomienia, wej./wyj.

Aplikacja będzie uruchomiana w sposób dostarczony przez środowisko QT.
Kod będzie mógł być dostarczany w następujący sposób

- Przez pole tekstowe w oknie aplikacji
- Wcztytanie z pliku

Wyjście aplikacji będzie natomiast zrealizowane za pomocą pola tekstowego w oknie aplikacji, na którym program będzie mógł wypisywać informacje.

10 Testowanie

10.1 Testy jednostkowe

Poszczególne konstrukcje i moduły będą sprawdzane za pomocą testów jednostkowych pozwalające przetestować elementarne funkcjonalności.
Testy te zostaną prawdopodobnie zrealizowane w bibliotece *Catch2*, ale może to ulec zmianie.

10.2 Testy integracyjne

Testy integracyjne będą przeprowadzane automatycznie. Zostanie przygotowanych kilka scenariuszy testujących wiele warstw kodu i różnych komponentów celem sprawdzenia ich wspólnego działania.
Sceneriusze będą napisane prawodopodbnie w tej samej bibliotece co testy jednostkowe.

10.3 Testy GUI

Działanie graficznego interfesju użytkownika będzie testowane manualnie.