Dokumentacja końcowa
TKOM
Wiktor Michalski

Język do opisu gier planszowych

Opis funkcjonalny:

Język umożliwia opisanie zasad prostych gier planszowych opartych na losowości i wyborach. Program działa w konsoli - na wejściu przyjmuje napisany przez użytkownika skrypt, po czym zwraca do konsoli (lub pliku) analizę danych wejściowych oraz odpowiednie wyjście wynikające z kodu.

Cechy języka:

- dynamicznie typowany, obsługujący wartości zmiennoprzecinkowe, całkowitoliczbowe oraz tekstowe
- instrukcja pętli **while**, z obsługą **break**
- instrukcja warunkowa **if .. else**
- możliwość definiowania własnych funkcji, które obsługują rekurencję
- funkcja wbudowana **roll(n)** zwracająca losową wartość z zakresu 0..n-1
- funkcja wbudowana **print** wypisująca tekst do konsoli
- funkcja wbudowana **forEachPlayer** wykonująca się dla każdego obiektu typu Player, z obsługą **break**, umożliwiająca odwołanie się do indywidualnych zmiennych każdego obiektu za pomocą przedrostka “**player.**”
- funkcja wbudowana **choice** umożliwiająca wybór akcji oraz gracza, do którego zmiennych można się odwoływać za pomocą przedrostka “**chosen.**”
- obsługa wyrażeń matematycznych i logicznych z uwzględnieniem priorytetu operatorów
- możliwość ustawienia zmiennych globalnych za pomocą przedrostka “**game**.”
- możliwość ustalenia ilości graczy **PLAYERS**
- możliwość ustalenia warunku **WIN**, który pozwala na wypisanie i wskazanie zwycięzców po zakończonej grze

EBNF:

letter = "A" | "B" | "C" | "D" | "E" | "F" | "G” | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p"| "q" | "r" | "s" | "t" | "u" | "v" | "w"| "x" | "y" | "z" ;
digit = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
nonZeroDigit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
naturalNumber = nonZeroDigit , { digit } ;
realNumber = [sign] , { digit }- , "." , { digit }- ;
name = letter , { letter | digit } ;
variableName = name | (name, "." , name)

comparisonOperator = ">" | "<" | "==" | "!="| ">=" | "<=" ;
LogicalOr= LogicalAnd , { “or” , LogicalAnd } ;
LogicalAnd = LogicalCompare , { “and” , LogicalCompare } ;
LogicalCompare = LogicalNegation , [comparisonOperator , LogicalNegation] ;
LogicalNegation = [“!”] , FullMathExpression ;
FullMathExpression = Multiplication , { (“+” | “-”) , Multiplication} ;

Multiplication = PartMathExpression , { (“*” | “/”) , PartMathExpression} ;
PartMathExpression = [“-”] , (Roll | Text | Value | BracketedExpression | FunctionCall) ;

Roll = “roll” , “(“ , LogicalOr , “)” ;
Text = ? any characters between “ and “ ?
Value = (realNumber | naturalNumber | variableName) ;
BracketedExpression = “(“ , LogicalOr, “)” ;
FunctionCall = Identifier , “(“ , { identifier} , “)” ;

Return = “return” , LogicalOr ;
FuncDefinition = “def” , variableName , “(“ , { “variableName” } , “)” ;

initVariable = "var" , variableName , ["=" , LogicalOr] , “;” ;
variableAssignment = variableName , "=" , LogicalOr , “;” ;
break = “break;” , ;
Print = “print” , “(“ , text , “)” ;
whileStatement = "while" , "(" , LogicalOr , ")" , “{“ . blockOfCode , “}” ;
ifStatement = "if" , "(" , LogicalOr ")" , “{“ , blockOfCode , “}” , ["else" , “{“ , blockOfCode , “}”] ;
choiceStatement = "choice" , "(" , LogicalOr , ")" , "{" , { choice } , "}" ;
choice = "[" , LogicalOr , "]" , “{“ , blockOfCode , “}” ;
forEachStatement = "forEachPlayer" , "(" , LogicalOr , ")" , blockOfCode ;
blockOfCode = { (ifStatement | whileStatement | initVariable | variableAssignment | Return | choiceStatement | break | PlayersHeader | functionCall | Roll | Print) }

playersInfo = “PLAYERS” , LogicalOr ;
winCondition = “WIN:” , “[“ , LogicalOr , “]”, ;
program = [playersInfo] , blockOfCode, [winCondition] ;

Opis realizacji:

Program został napisany w języku **Python**, z użyciem pomocniczych biblotek – **argparse** do zarządzania uruchamianiem programu oraz **contextlib** i **unittest** do testowania.

Program składa się z modułów:

Lexer, gdzie tekst wejściowy jest dzielony na odpowiednie tokeny, ze zwracaniem uwagi na poprawność liczb (jest sprawdzana przy dzieleniu ich na rzeczywiste/naturalne), długość zmiennych (ograniczenie do 100 znaków), poprawność identifierów (czy przedrostki się zgadzają, o ile istnieją). Pomijane są znaki białe, z wyjątkiem tych, które są częścią stringów.

Parser, gdzie tokeny są poddawane analizie i na ich podstawie powstaje drzewo składniowe z obiektów typu Node, gdzie ich dzieci to inne Node (dla bardziej złożonych struktur) lub Token (dla pojedynczych znaków, które określają operacje). **Interpretacja** kodu, tj. analiza syntaktyczna, odbywa się rekurencyjnie poprzez wywoływanie funkcji execute() na obiektach potomnych dla każdego Node, począwszy od ProgramNode.

W zależności od zadeklarowanej liczby graczy, do listy `players = []` zostaje dodana odpowiednia liczba obiektów typu Player.

Każda zadeklarowana funkcja tworzy odpowiedni obiekt typu Function, który jest przechowywany w `functions = []`.

Zmienne są przechowywane w słowniku localVarDict (który zawiera albo zmienne głównego kodu programu, albo zmienne lokalne funkcji), słowniku gameVarDict (gdzie są

zmienne globalne z przedrostkiem “game.”) oraz w słownikach variables, indywidualnych dla każdego gracza.

Aby w funkcjach choice() i forEachPlayer() była możliwość używania zmiennych o przedrostkach chosen i player, referencje do odpowiednich obiektów są przechowywane w zmiennych: analysedPlayer i chosenPlayer.

Działanie rekurencji jest oparte o podobny mechanizm – przechowywane są instancje wywołania funkcji. Przy wywołaniu funkcji zapisujemy sobie obecny słownik i obecne wskazanie na funkcję, po czym wywołujemy funkcję zagnieżdżoną.

```
def callFunction(funcObject, args):
    global localVarDict
    global analysedFunc

    lastFunc = analysedFunc
    lastVarDict = localVarDict

    analysedFunc = deepcopy(funcObject)
    localVarDict = analysedFunc.getVarDict(args)
    retValue = analysedFunc.execute()

    analysedFunc = lastFunc
    localVarDict = lastVarDict

    return retValue
```

Testy są oparte o moduł unittest do Pythona, z pomocą których sprawdzam, czy tokeny wyjściowe z Leksera, drzewo składniowe z Parsera i output (w postaci, np. printów) zgadza się z oczekiwaniami. Ponadto testuję przypadki błędne, aby się upewnić, że za każdym razem program odpowiednio zareaguje i poinformuje o błędzie użytkownika.

Przykłady języka w działaniu:

Definicja zmiennej (z inicjalizacją lub bez):

```
var a = 1;
var a;
var a = “text”;
```

Przypisanie wartości do zmiennej:

```
a = 2;
a = “text2”;
a = funkcja(a);
```

Pętla z warunkiem:

```
while(cond) { … }
```

Z obsługą break.

Blok warunkowy:

```
if(cond) { … } else { … }
```

Pętla dla każdego gracza (zarówno w cond, jak i w body można odwoływać się do indywidualnych wartości zmiennych każdego gracza):

```
forEachPlayer(cond) { body }
forEachPlayer(player.inPrison) { players.sad = 1;}
```

Z obsługą break.

Funkcja wbudowana roll:

`roll(6)` – zwraca liczbę od 0 do 5

Blok wyboru choice:

```
choice(6)
{
[player.money > 10] { player.money = player.money - game.tax;}
[player.money < 0] { player.money = player.money + rolled;
chosen.money = chosen.money - rolled;}
}
```

Blok choice jest inicjalizowany liczbą, a następnie do zmiennej rolled przypisana jest losowa wartość z zakresu 0..liczba-1 (symulacja rzutu kostką, ale nie ma konieczności używania potem zmiennej rolled w kodzie). W nawiasach kwadratowych są warunki wykonania danej akcji (może się zdarzyć, że gracz będzie mógł wybrać jedną z kilku, lub jedyną dostępną, lub nie będzie miał żadnej do wyboru). W ramach bloku choice następuje również wybór gracza, wobec którego będzie przeprowadzana akcja – do którego można się odwołać za pomocą przedrostka “chosen.”.

Funkcja wbudowana print wypisująca liczby lub tekst:

```
print(a);
print(“tekst”);
print(111);
```

Definicja funkcji o dowolnej liczbie parametrów:

```
def add(a, b)
{
 return a+b;
}
```

Określenie liczby graczy:

```
PLAYERS 4;
```

Określenie warunku zwycięstwa (wypisuje statystyki graczy i przy zwycięzach pojawia się adnotacja WINNER):

```
WIN:
[player.tries == minTries]
```

I fragment odpowiadającego wyjścia z konsoli:

```
Player 8
   player.inPrison -- 0
   player.tries -- 2

Player 9 - WINNER
   player.inPrison -- 0
   player.tries - 1
```