Opis programu

Interpreter języka skryptowego napisany w języku C#, rozszerzający funkcjonalność silnika Unity. Proces interpretacji generuje tablicę zmiennych, graf przejść oraz listę punktów początkowych dla systemu dialogowego do gier. Udostępnia również metody pozwalające na płynne przechodzenie między kolejnymi węzłami systemu.

Gramatyka

char = ? single input character ? ;

letter = ? character between a-z or A-Z ? ;

digit = "0" | positiveDigit ;

positiveDigit = ? single non 0 digit ? ;

int = (digit) | (positiveDigit (digit)*) ;

float = int | (int"."(digit)*) ;

bool = "true" | "false" ;

plainText = char* ;

string = """plainText"""

varName = letter(letter | digit)* ;

comment = ("//" plainText "\n") | ("/*" plainText "*/") ;

script = (include | varDeclaration | hook)* ;

include = "include" quotedText ;

varDeclaration = "var" varName ;

block = dialogueBlock | optionsBlock | listBlock | modifyBlock ;

dialogueBlock = [condition] formatedText [transitionBranch] [block] ;

hook = "["varName"]" block "[""/"varName"]" ;

option = "->" formatedText "{" block "}" ;

optionsBlock = optionsCond "{" option ("," option)* "}" [block] ;

listBlock = listStyle (("{" block("," block)$^{+}$ "}") | optionsBlock) [block] ;

transitionBranch = "=>" [condition] (varName | "end") ;

listStyle = ["random"] ["looped"] ;

condition = ["if" expr "then" ["once"]] ;

optionsCond = "choose" ["default" int] ;

modifyBlock = "modify" varName eqOp expression [block] ;

expr = simpExpr [relOp simpExpr] ;

simpExpr = term [addOp term] ;

term = factor [mulOp factor] ;

factor = value | ("("expr")") ;

eqOp = "=" ;

relOp = "<" | "<=" | ">" | ">=" | "==" | "!=" ;

addOp = sign | "or" ;

mulOp = "*" | "/" | "and" ;

value = varName | int | float | bool ;

sign = "+" | "-" ;

Tokeny

string, int, bool, float, "var", varName, "end", "include", "["hook"]", "[/"hookEnd"]", "->", "{", "}", ",", "=>", "random", "looped", "if", "then", "once", "choose", "default", "modify", "(", ")", "=", "<", "<=", "==", "!=", ">", ">=", "or", "and", "*", "/", "+", "-"

Konstrukcja programu

DialogueObject – finalna klasa trzymająca system dialogowy w postaci:

- słownika zmiennych w postaci <nazwa, wartość>.
- punktów wejścia, łączących nazwę używaną przy wywołaniu z pierwszym, odpowiadającym jej węzłem.
- Listę węzłów systemu dialogowego

Klasa pozwala na uruchomienie systemu od wybranego miejsca oraz przechodzenie do kolejnych węzłów pozwalające na interakcję (bloki tekstu oraz listy opcji do wyboru).

Token – struktura przechowująca pojedynczy element tworzony przez lekser. Składa się z typu tokenu oraz jego wartości trzymanej w generycznym polu typu string.

TokenType – enumerator określający typ tokenu.

Reader – klasa zwracająca kolejne znaki z zawartości skolejkowanych w niej plików. Pozwala na dodawanie kolejnych plików w trakcie trwania pracy (monitoruje jednocześnie czy dany plik nie znalazł się wcześniej w kolejce). Dodatkowo udostępnia informację o obecnie analizowanym pliku oraz pozycji obecnie zwróconego znaku.

Scanner – klasa łącząca znaki z Readera w tokeny, zwracane na życzenie Parsera. Usuwa komentarze oraz nadmiarowe białe znaki(wszystkie poza pojedynczymi spacjami wewnątrz tekstów dialogowych). Nazwy zmiennych są porównywane ze słownikiem słów kluczowych na podstawie dynamicznie generowanego hasha oraz wartości.

Hash – obiekt realizujący generację hasha w postaci pojedynczej wartości typu int. Umożliwia wygenerowanie jego wartości z kompletnego stringa jak i poprzez iteracyjne dodawanie kolejnych znaków.

Parser – klasa odpowiedzialna za budowę finalnego systemu dialogowego z pliku wejściowego. Korzystając z Scannera buduje kolejne węzły grafu przejścia, inicjalizuje zmienne i punkty wejścia. W wypadku niepowodzenia zwraca na logger treść błędu i miejsce jego wystąpienia w kodzie.

Block – klasa bazowa, realizująca pojedynczy węzeł systemu. Pozwala na wejście do reprezontowanego przez nią węzła albo przejście do następnika.

DialogueBlock – przeciążenie klasy Block. Utrzymuje w sobie tekst do wyświetlenia oraz warunek wejścia do bloku i jego następników. Utrzymuje też stan tego czy blok został odwiedzony. Próba wejścia do węzła kiedy warunek nie jest spełniony kończy pracę. Przejście z węzła do następnika powoduje wybranie węzła połączonego z daną nazwą, o ile warunek przejścia został spełniony i rozgałęzienie zostało zainicjalizowane, w przeciwnym wypadku program przejdzie do następnego bloku.

ModifyBlock – blok trzymający wyrażenie modyfikujące wartość zmiennej. Wejście do bloku wykonuje podaną operację i przechodzi automatycznie do kolejnego węzła.

ListBlock – blok przedstawiający listę bloków lub listę możliwych do wyboru opcji. Lista może być pomieszana w sposób losowy albo zapętlona. Wejście do listy opcji zwraca obecny blok i pozwala na wybór jednej z nich, natomiast wejście do standardowej listy powoduje zwrócenie kolejnego bloku z listy aż do momentu wyczerpania opcji przy sukcesywnych wywołaniach(o ile lista nie jest zapętlona). W wypadku wyczerpania opcji, blok zwraca null. Dodatkowo, lista monitoruje element na którym skończyła pracę albo domyślną opcję.

Expression – wyrażenie matematyczne w postaci drzewa rozbioru. Finalna wartość zostaje zwrócona, w momencie wywołania, poprzez wstępujące wyliczanie wartości kolejnych węzłów drzewa. Aby było to możliwe, wyrażenie potrzebuje do działania słownika z wartościami zmiennych.

Wymagania

- Zapis dialogów w formie zewnętrznego skryptu pozwalającego na zmianę treści(poprawki, zmiana wersji językowej) bez konieczności rekompilacji całego projektu.
- Interpretacja kodu w trakcie realizacji programu.
- Kontrola błędów zintegrowana z wbudowanym loggerem Unity. Powiadamianie o typie błędu, miejscu wystąpienia oraz możliwych sposobach rozwiązania problemu.
- Wykonywanie napisanego kodu w wyniku wywołań z innych modułów programu.
- Błędy przedstawiane w postaci zrozumiałej dla osoby na codzień nie programującej.
- Podział skryptów na mniejsze pliki

Przykład testowy

```
var x //deklaracja zmiennej x


[Test] //początek bloku o nazwie Test
modify x = 5 //zmiana wartości x na 5
if 3*4 == 12 then "Text" //blok o treści "Text" z warunkiem wejścia zwracającym prawdę
        => if true then once Branch //jednokrotne przejście do punktu wejściowego o nazwie Branch
modify x = (5+2)*3+4 //zmiana wartości x na 25
if x == 23 then "test1" //blok z tekstem "test1" i fałszywym warunkiem wejścia
"test2" //blok nieosiągalny z powodu fałszywego warunku w poprzedniej linii
[/Test] //koniec bloku o nazwie Test


Wywołanie bloku Test spowoduje wywołanie operacji przypisania do x wartości 5 i wyświetlenie stringa "Text" a następnie przejście i wykonanie bloku o nazwie Branch. Każde kolejne wywołanie powtórzy tą operację z tą różnicą, że zamiast przejść do Branch, wykonane zostanie przypisanie do x wyniku wyrażenia (5+2)*3+4. W następnym węźle zostanie wykonane porównaie 23 z 25 i program zakończy pracę z powodu zwrócenia fałszu.


[Branch]
"ABC"
looped {random {"D1", "D2" => end}, {"E1", "E2"}}
```

```
choose { -> "A" {"A2"}
        -> "B" {"B2"}
        -> "C" {"C2"}
        }
[/Branch]
```

Wywołanie bloku Brach wyświetli tekst "ABC". Następnie do wyświetleni zostanie wybrany losowy element "D1" albo "D2". Po wyświetleniu "D2" system skończy pracę. Natomiast po wyświetleniu "D1" zostanie przedstawiona lista 3 opcji do wyboru.

Kolejne wywołania zamiast "D1" lub "D2" będą wyświetlać "E1", "D1" lub "D2", "E2" itd.

Obsługa błędów

Błędy blokują budowe wewnętrznego automatu stanów i przerywają dalszą analizę skryptu. Użytkownik zostaje powiadomiony o błędzie poprzez komunikat zawierający nazwę pliku, wiersz i kolumnę w tekście, przyczynę błędu. Jeśli jest to możliwe, komunikat zasugeruje też możliwy sposób naprawy błędu.

Błędy:

- Niepoprawny sposób zapisu typu int i float
- Niepoprawne nazwy plików (np. "1%$@#R"), zmiennych (np. 2Odpowiedz) i bloków
- Literówki w nazwach zmiennych i bloków
- Brak zamknięcia bloku nazwangeo
- Niepoprawna kolejność zamykania bloków
- Niepoprawne sparowanie nawiasów
- Nieoczekiwany symbol
- Ponowna deklaracja zmiennej lub punktu początkowego o danej nazwie
- Brak zmiennej lub punktu początkowego o podanej nazwie
- Brak bloku w liście lub punkcie poocząkowym systemu
- Błędne wyrażenie matematyczne
- Nieoczekiwane zakończenie pliku wejściowego