Stage -2 Report

Syntax Directed Translation Scheme

Context-Free Grammar

Rule 0 S' -> start
Rule 1 start -> ID
Rule 2 start -> parameter
Rule 3 parameter -> engine
Rule 4 parameter -> score
Rule 5 parameter -> control
Rule 6 parameter -> speed
Rule 7 parameter -> block
Rule 8 parameter -> level
Rule 9 parameter -> grid
Rule 10 parameter -> variation
Rule 11 grid -> GRID ID ASSIGN LPAREN NUMBER COMMA NUMBER RPAREN
Rule 12 variation -> VARIATION ID ASSIGN MODE
Rule 13 level -> LEVEL ID ASSIGN APO NUMBER APO
Rule 14 block -> ID DOT COLOR LPAREN OPTIONS RPAREN
Rule 15 block -> ID DOT SHAPE LPAREN APO ORIENTATION APO RPAREN
Rule 16 block -> BLOCK ID
Rule 17 speed -> SPEED ID ASSIGN NUMBER
Rule 18 control -> CONTROL ID ASSIGN DIRECTION
Rule 19 score -> SCORE ID
Rule 20 engine -> ENGINE ID

*Rule 0 is added as part of LR automaton grammar
Starting symbol: start
Non-Terminals: start,parameter,engine,score,control,speed,block,level,grid
Terminals/Tokens:
GRID,ID,ASSIGN,LPAREN,NUMBER,ID,COMMA,RPAREN,SCORE,ENGINE,BLOCK,SPEED,
VARIATION,COLOR,SHAPE,APO,OPTIONS,DIRECTIONS,MODE,DOT

LR(1) Automaton

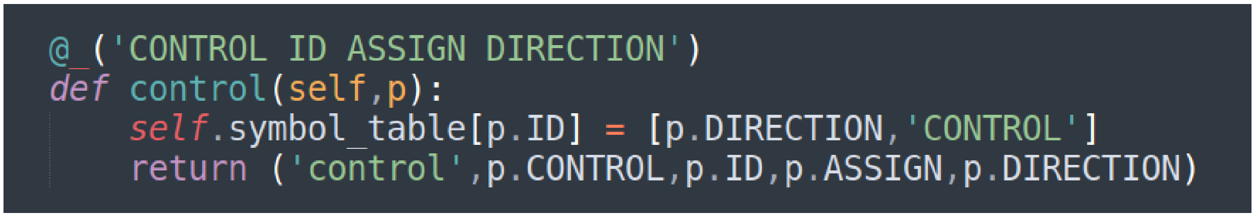
The python SLY library is based on LALR(1) architecture, so the parse tree generated is based on a bottom-up based recursion tree.

The description given at the end of the report specifies the LR automaton specific to our parser. Each state keeps track of the grammar rules that might be in the process of being matched at that point. Within each rule, the “.” character indicates the current location of the parse within that rule. In addition, the actions for each valid input token are listed.

Parser specs

As part of the syntax-directed translation scheme, we deduce the semantic meaning of our programming constructs in a custom data structure which helps us in deciphering the meaning of our game programming code. The custom data structure functions in a way as a symbol table for us. The symbol table contains the name of the identifiers the relevant token it belongs to, the value and other related information. The translation is done at the time of construction of the parse tree itself.

Consider the following example below

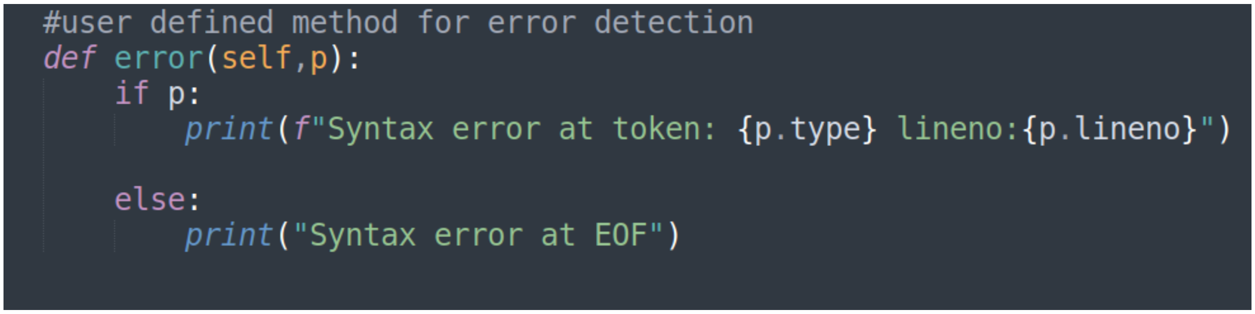


```
@_('CONTROL ID ASSIGN DIRECTION')
def control(self,p):
    self.symbol_table[p.ID] = [p.DIRECTION,'CONTROL']
    return ('control',p.CONTROL,p.ID,p.ASSIGN,p.DIRECTION)
```

The following code corresponds to grammar **rule 18** of our CFG is part of our parser code (parser.py). The parser returns a parse tree construction as ‘control’ as the parents and the rest of the elements of the tuple as its children. The corresponding action on the identification of this rule is to store a record in our representative symbol table with the identifier name, the value it holds and the token it represents

Parser Challenges

We have included a custom error rule in our parser where we can identify in our game programming language where a syntax error and corresponding to which token occurs if any do exist.



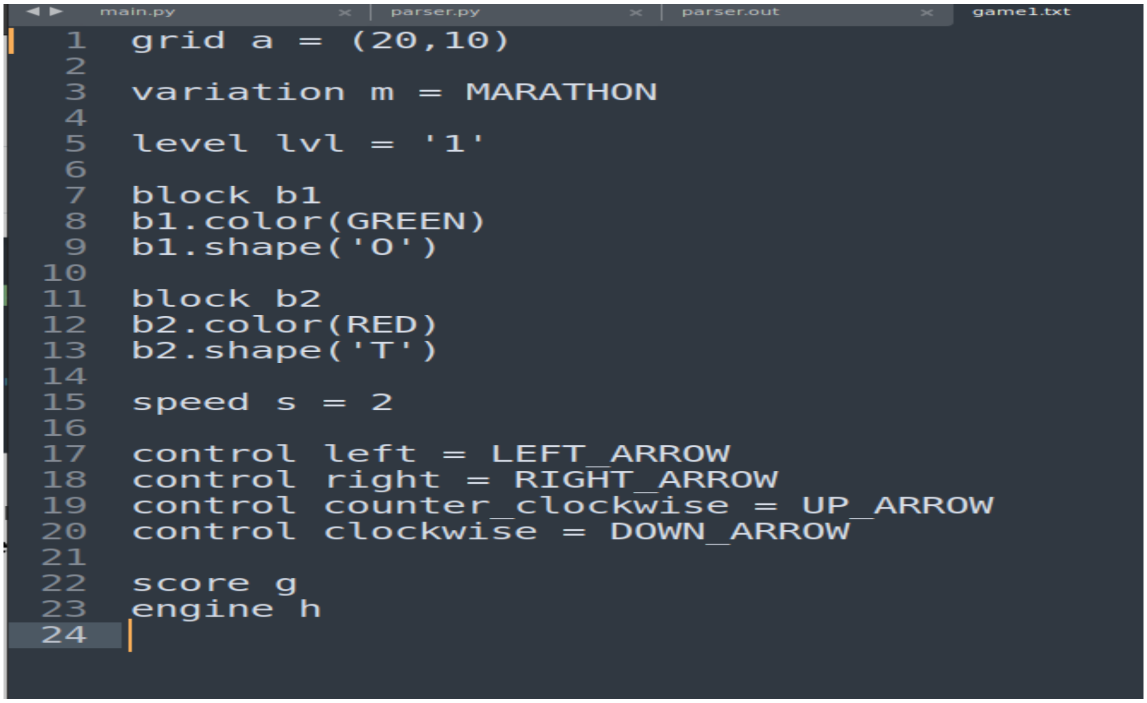
```
#user defined method for error detection
def error(self,p):
    if p:
        print(f"Syntax error at token: {p.type} lineno:{p.lineno}")

    else:
        print("Syntax error at EOF")
```

We did face a challenge where we were getting syntax errors when we are processing the whole of our game programming language at once instead of processing it line by line. We still have to figure out why such an error occurs.

Test cases

We have provided a custom test case file named game1.txt which is what any typical game file will look like.



```
grid a = (20,10)

variation m = MARATHON

level lvl = '1'

block b1
b1.color(GREEN)
b1.shape('O')

block b2
b2.color(RED)
b2.shape('T')

speed s = 2

control left = LEFT_ARROW
control right = RIGHT_ARROW
control counter_clockwise = UP_ARROW
control clockwise = DOWN_ARROW

score g
engine h
```

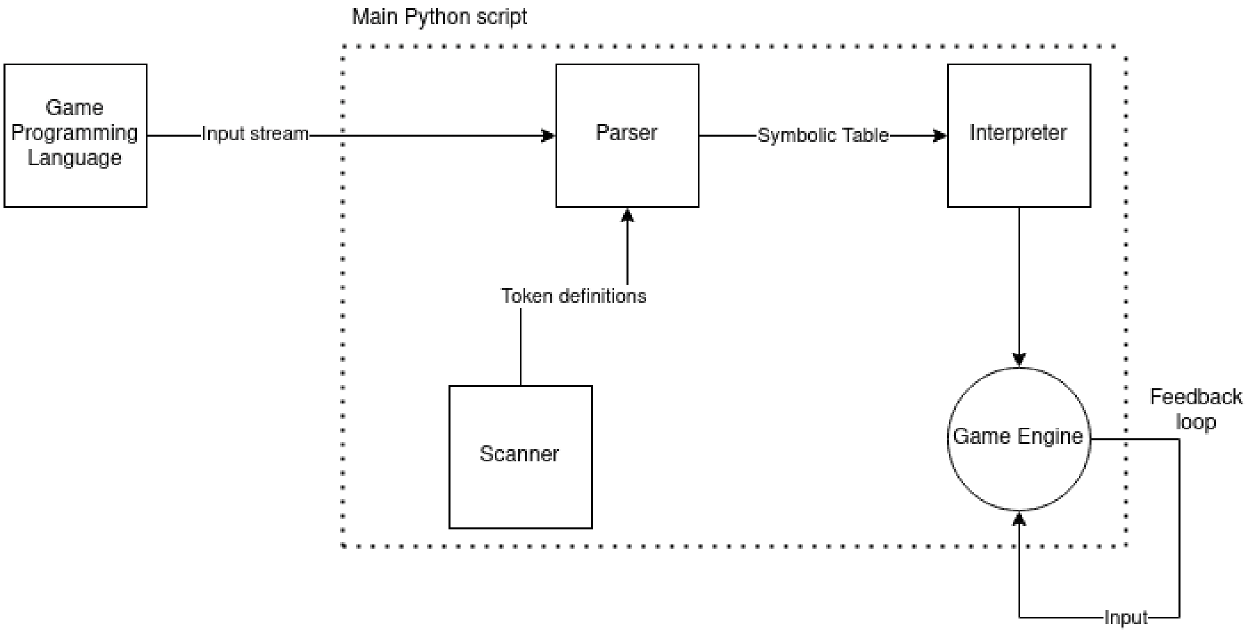
The game file specifies the grid size in a variable a, the variation of the game that will be played is mentioned in the variable m. Similarly the variable lvl stores the starting level of the game. The keyword block identifies which kind Tetris blocks will be available for use in the game engine as well as their color. We also specify the speed of the falling blocks in the variable s.We also specify the control parameters for the blocks using the variables left,right,counter and counter clockwise. The variables h keep track of the game engine states while the variable g keeps track of the score relevant to the user in state of the game engine

End-to-End Toolchain

The scanner specifies the token definition and corresponds lexemes that fit corresponding to each token. These specific token definitions are then fed to the parser to perform the syntax analysis. Syntax analysis rules are paired up with the actions to perform the syntax-directed translation at the same time. The representative table which will be formed during the semantic analysis will be passed to the interpreter who will be responsible for generating the game engine.

Workflow

Main python script imports the scripts for the scanner, parser and interpreter. The main python file accesses the text file of our game programming language and processes it line by line, passing each line to the parser. The parser while internally generating the parse trees also creates the representative symbolic table which will be given to the interpreter for creating the game engine



state 0

```
(0) S' -> . start
(1) start -> . ID
(2) start -> . parameter
(3) parameter -> . engine
(4) parameter -> . score
(5) parameter -> . control
(6) parameter -> . speed
(7) parameter -> . block
(8) parameter -> . level
(9) parameter -> . grid
(10) parameter -> . variation
(20) engine -> . ENGINE ID
(19) score -> . SCORE ID
(18) control -> . CONTROL ID ASSIGN DIRECTION
(17) speed -> . SPEED ID ASSIGN NUMBER
(14) block -> . ID DOT COLOR LPAREN OPTIONS RPAREN
(15) block -> . ID DOT SHAPE LPAREN APO ORIENTATION APO RPAREN
(16) block -> . BLOCK ID
(13) level -> . LEVEL ID ASSIGN APO NUMBER APO
(11) grid -> . GRID ID ASSIGN LPAREN NUMBER COMMA NUMBER RPAREN
(12) variation -> . VARIATION ID ASSIGN MODE
ID             shift and go to state 2
ENGINE         shift and go to state 12
SCORE          shift and go to state 13
CONTROL        shift and go to state 14
SPEED          shift and go to state 15
BLOCK          shift and go to state 16
LEVEL          shift and go to state 17
GRID           shift and go to state 18
VARIATION      shift and go to state 19

start                  shift and go to state 1
parameter              shift and go to state 3
engine                 shift and go to state 4
score                  shift and go to state 5
control                shift and go to state 6
```

```
    speed               shift and go to state 7
    block               shift and go to state 8
    level               shift and go to state 9
    grid                shift and go to state 10
    variation           shift and go to state 11
```

state 1

```
    (0) S' -> start .
```

state 2

```
    (1) start -> ID .
    (14) block -> ID . DOT COLOR LPAREN OPTIONS RPAREN
    (15) block -> ID . DOT SHAPE LPAREN APO ORIENTATION APO RPAREN
    $end            reduce using rule 1 (start -> ID .)
    DOT             shift and go to state 20
```

state 3

```
    (2) start -> parameter .
    $end            reduce using rule 2 (start -> parameter .)
```

state 4

```
    (3) parameter -> engine .
    $end            reduce using rule 3 (parameter -> engine .)
```

state 5

```
    (4) parameter -> score .
    $end            reduce using rule 4 (parameter -> score .)
```

state 6

```
    (5) parameter -> control .
    $end            reduce using rule 5 (parameter -> control .)
```

state 7

```
(6) parameter -> speed .
$end            reduce using rule 6 (parameter -> speed .)
```

state 8

```
(7) parameter -> block .
$end            reduce using rule 7 (parameter -> block .)
```

state 9

```
(8) parameter -> level .
$end            reduce using rule 8 (parameter -> level .)
```

state 10

```
(9) parameter -> grid .
$end            reduce using rule 9 (parameter -> grid .)
```

state 11

```
(10) parameter -> variation .
$end            reduce using rule 10 (parameter -> variation .)
```

state 12

```
(20) engine -> ENGINE . ID
ID              shift and go to state 21
```

state 13

```
(19) score -> SCORE . ID
ID              shift and go to state 22
```

state 14

```
(18) control -> CONTROL . ID ASSIGN DIRECTION
```

```
    ID              shift and go to state 23
```

state 15

```
    (17) speed -> SPEED . ID ASSIGN NUMBER
    ID              shift and go to state 24
```

state 16

```
    (16) block -> BLOCK . ID
    ID              shift and go to state 25
```

state 17

```
    (13) level -> LEVEL . ID ASSIGN APO NUMBER APO
    ID              shift and go to state 26
```

state 18

```
    (11) grid -> GRID . ID ASSIGN LPAREN NUMBER COMMA NUMBER RPAREN
    ID              shift and go to state 27
```

state 19

```
    (12) variation -> VARIATION . ID ASSIGN MODE
    ID              shift and go to state 28
```

state 20

```
    (14) block -> ID DOT . COLOR LPAREN OPTIONS RPAREN
    (15) block -> ID DOT . SHAPE LPAREN APO ORIENTATION APO RPAREN
    COLOR           shift and go to state 29
    SHAPE           shift and go to state 30
```

state 21

```
    (20) engine -> ENGINE ID .
    $end            reduce using rule 20 (engine -> ENGINE ID .)
```

state 22

```
    (19) score -> SCORE ID .
    $end            reduce using rule 19 (score -> SCORE ID .)
```

state 23

```
    (18) control -> CONTROL ID . ASSIGN DIRECTION
    ASSIGN          shift and go to state 31
```

state 24

```
    (17) speed -> SPEED ID . ASSIGN NUMBER
    ASSIGN          shift and go to state 32
```

state 25

```
    (16) block -> BLOCK ID .
    $end            reduce using rule 16 (block -> BLOCK ID .)
```

state 26

```
    (13) level -> LEVEL ID . ASSIGN APO NUMBER APO
    ASSIGN          shift and go to state 33
```

state 27

```
    (11) grid -> GRID ID . ASSIGN LPAREN NUMBER COMMA NUMBER RPAREN
    ASSIGN          shift and go to state 34
```

state 28

```
    (12) variation -> VARIATION ID . ASSIGN MODE
    ASSIGN          shift and go to state 35
```

state 29

```
(14) block -> ID DOT COLOR . LPAREN OPTIONS RPAREN
LPAREN        shift and go to state 36
```

state 30

```
(15) block -> ID DOT SHAPE . LPAREN APO ORIENTATION APO RPAREN
LPAREN        shift and go to state 37
```

state 31

```
(18) control -> CONTROL ID ASSIGN . DIRECTION
DIRECTION     shift and go to state 38
```

state 32

```
(17) speed -> SPEED ID ASSIGN . NUMBER
NUMBER        shift and go to state 39
```

state 33

```
(13) level -> LEVEL ID ASSIGN . APO NUMBER APO
APO           shift and go to state 40
```

state 34

```
(11) grid -> GRID ID ASSIGN . LPAREN NUMBER COMMA NUMBER RPAREN
LPAREN        shift and go to state 41
```

state 35

```
(12) variation -> VARIATION ID ASSIGN . MODE
MODE          shift and go to state 42
```

state 36

```
    (14) block -> ID DOT COLOR LPAREN . OPTIONS RPAREN
    OPTIONS         shift and go to state 43


state 37

    (15) block -> ID DOT SHAPE LPAREN . APO ORIENTATION APO RPAREN
    APO             shift and go to state 44


state 38

    (18) control -> CONTROL ID ASSIGN DIRECTION .
    $end            reduce using rule 18 (control -> CONTROL ID ASSIGN DIRECTION .)


state 39

    (17) speed -> SPEED ID ASSIGN NUMBER .
    $end            reduce using rule 17 (speed -> SPEED ID ASSIGN NUMBER .)


state 40

    (13) level -> LEVEL ID ASSIGN APO . NUMBER APO
    NUMBER          shift and go to state 45


state 41

    (11) grid -> GRID ID ASSIGN LPAREN . NUMBER COMMA NUMBER RPAREN
    NUMBER          shift and go to state 46


state 42

    (12) variation -> VARIATION ID ASSIGN MODE .
    $end            reduce using rule 12 (variation -> VARIATION ID ASSIGN MODE .)


state 43

    (14) block -> ID DOT COLOR LPAREN OPTIONS . RPAREN
    RPAREN          shift and go to state 47


state 44

    (15) block -> ID DOT SHAPE LPAREN APO . ORIENTATION APO RPAREN
    ORIENTATION     shift and go to state 48


state 45

    (13) level -> LEVEL ID ASSIGN APO NUMBER . APO
    APO             shift and go to state 49


state 46

    (11) grid -> GRID ID ASSIGN LPAREN NUMBER . COMMA NUMBER RPAREN
    COMMA           shift and go to state 50


state 47

    (14) block -> ID DOT COLOR LPAREN OPTIONS RPAREN .
    $end            reduce using rule 14 (block -> ID DOT COLOR LPAREN OPTIONS RPAREN .)


state 48

    (15) block -> ID DOT SHAPE LPAREN APO ORIENTATION . APO RPAREN
    APO             shift and go to state 51


state 49

    (13) level -> LEVEL ID ASSIGN APO NUMBER APO .
    $end            reduce using rule 13 (level -> LEVEL ID ASSIGN APO NUMBER APO .)


state 50

    (11) grid -> GRID ID ASSIGN LPAREN NUMBER COMMA . NUMBER RPAREN
    NUMBER          shift and go to state 52
```

state 51

```
    (15) block -> ID DOT SHAPE LPAREN APO ORIENTATION APO . RPAREN
    RPAREN          shift and go to state 53
```

state 52

```
    (11) grid -> GRID ID ASSIGN LPAREN NUMBER COMMA NUMBER . RPAREN
    RPAREN          shift and go to state 54
```

state 53

```
    (15) block -> ID DOT SHAPE LPAREN APO ORIENTATION APO RPAREN .
    $end            reduce using rule 15 (block -> ID DOT SHAPE LPAREN APO ORIENTATION APO RPAREN .)
```

state 54

```
    (11) grid -> GRID ID ASSIGN LPAREN NUMBER COMMA NUMBER RPAREN .
    $end            reduce using rule 11 (grid -> GRID ID ASSIGN LPAREN NUMBER COMMA NUMBER RPAREN .)
```