Group 44

Guardians of Tetris

1. Top Level Design Specs

1.1 Program Structure

- Python is chosen as the language of choice for providing the background framework for our Tetris engine.
- Python will be used to create a lexical analyzer as well as a parser for interpreting our game programming language.
- There will main game file is written in our own custom game programming language which will be mapped to our python scripts which do the heavy lifting of maintaining our game engine state.
- Our game engine framework will be based on a python external library called pygame.
- There will be a main driver python script that will import the necessary custom made modules for lexers and parsers.

1.2 Offered Primitives

- Our Language offers primitives for the features of the different aspects related to the Tetris game as well the various primitives offered in a general-purpose programming language. The offered primitives are we have constructed so far are[subject to change];

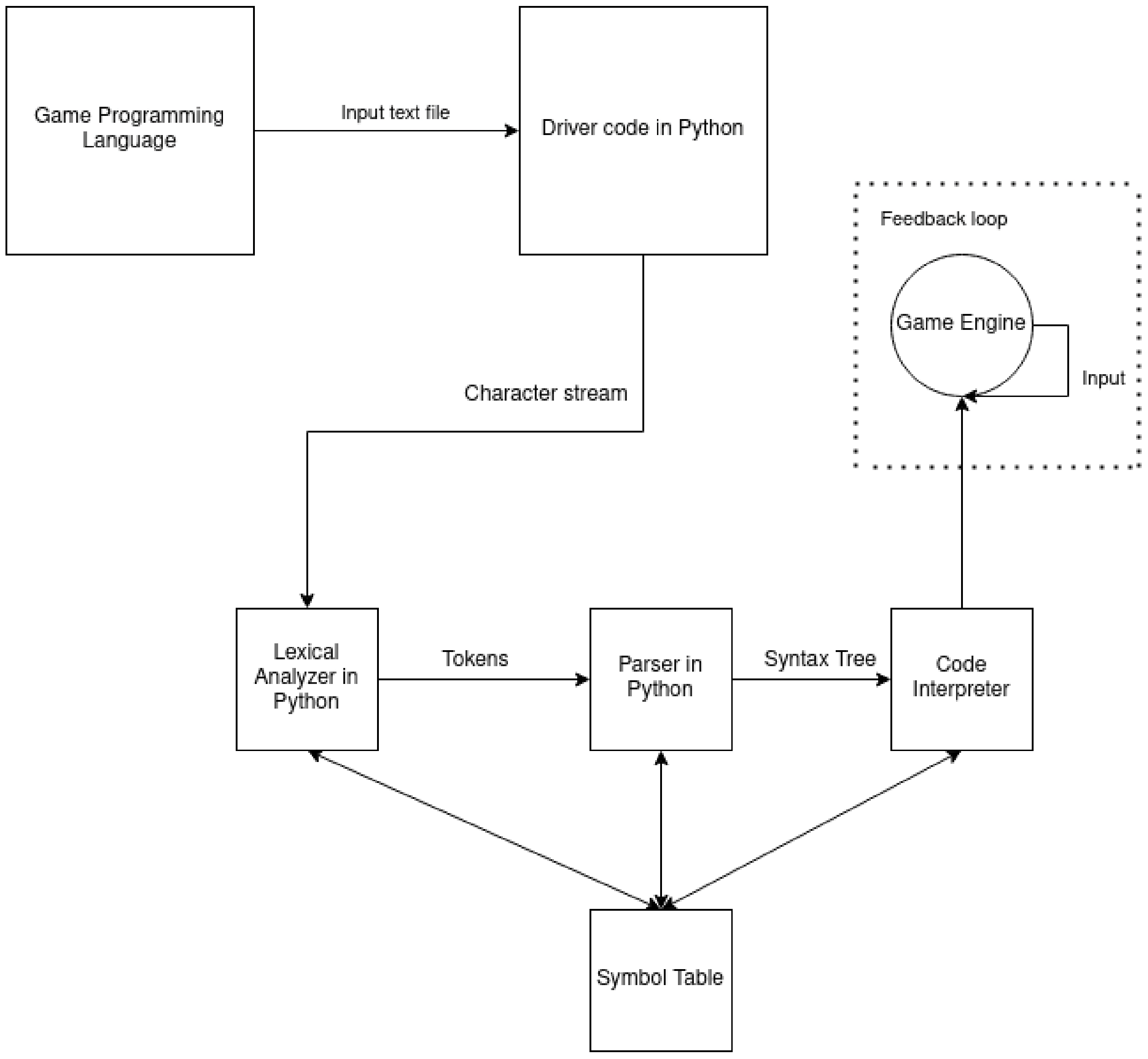
   a. Grid
   b. Variation
   c. Level
   d. Block_list
   e. Block
   f. Control
   g. Drop
   h. Score
   i. Engine
   j. Hold_mode
   k. Ghost_mode
   l. Lock_down

-

1.3 Programmable Features

- We have included a while loop in our construct of language to simulate an iterative loop of engine states over time
- The game programming language will be object-oriented as we will be making objects of required components for the game with associated methods and attributes. The method implementations will however be hidden away in python code.

1.4 Pipeline Schema



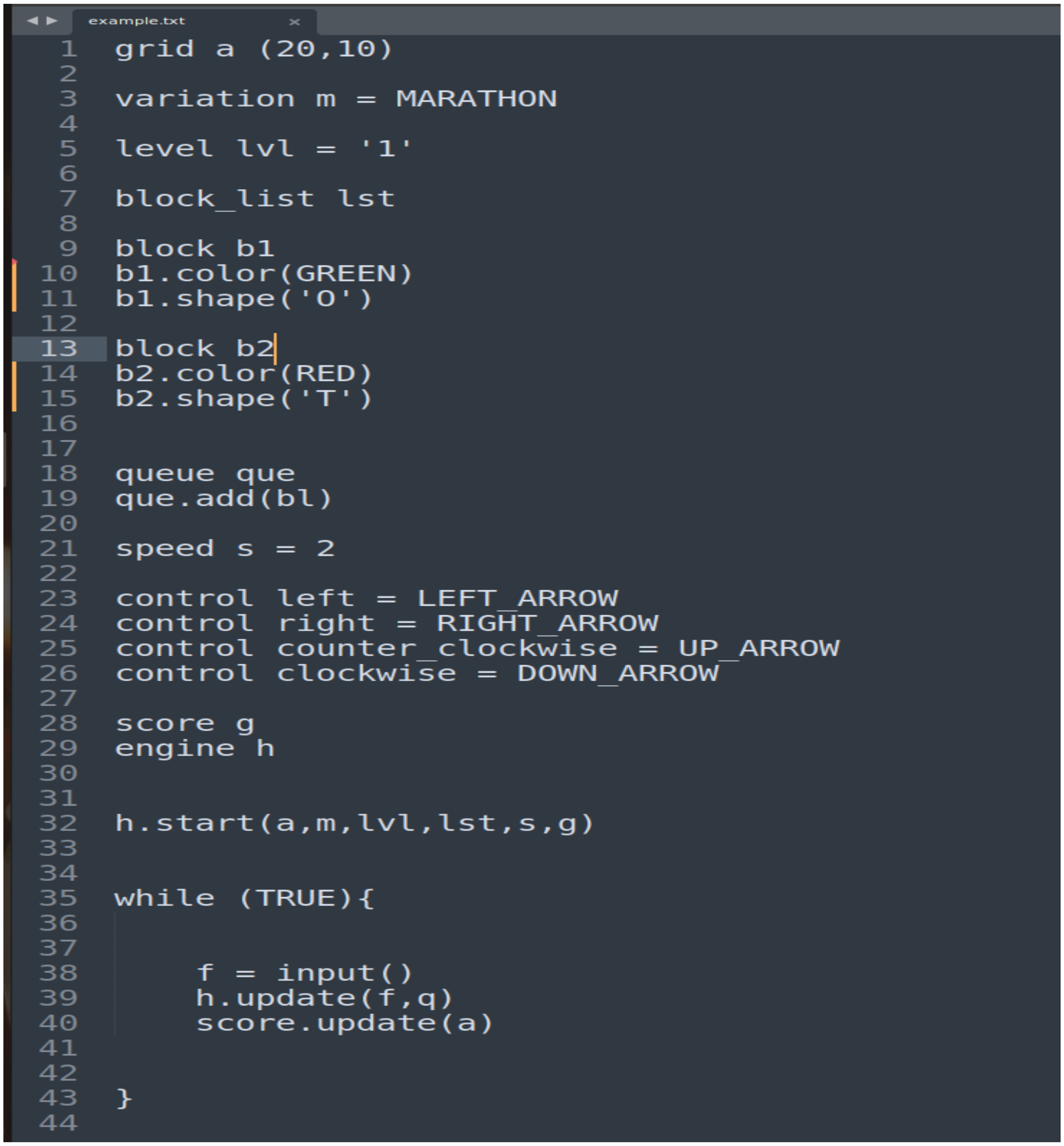
2. Scanner Design

- Our lexicon is small and in the order of ’50s and subject to change depending on design modifications. Below is the list of tokens prepared so far.

1. Token list
    1.1. <id, pointer to symbol table>
    1.2. <assign_op>
    1.3. <number,integer value>
    1.4. <grid>
    1.5. <variation>
    1.6. <control>
    1.7. <queue>
    1.8. <block>
    1.9. <block_list>
    1.10. <lock_down>
    1.11. <score>
    1.12. <engine>
    1.13. <delim>
    1.14. <left_par>
    1.15. <right_par>
    1.16. <dot>
    1.17. <comma>
    1.18. <apo>
    1.19. <left_curly>
    1.20. <right_curly>
    1.21. <orientation>
    1.22. <color>
    1.23. <input>
    1.24. <constant,value in symbol table>
    1.25. <while>
    1.26. <bool>
    1.27. <start>
    1.28. <update>
    1.29. <whitespace>
    1.30. <ghost>

    - The scanner will be implemented using the sly package of python
    - Tokens will be generated by reading our game programming text file and generating tokens by matching with regular expressions.

Example



```
grid a (20,10)

variation m = MARATHON

level lvl = '1'

block_list lst

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

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


queue que
que.add(bl)

speed s = 2

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

score g
engine h


h.start(a,m,lvl,lst,s,g)


while (TRUE){


    f = input()
    h.update(f,q)
    score.update(a)


}

```

The following file will generate a grid of (20,10) pixels, the variation of the game generated will be MARATHON, at level 1. A blocklist has been initialized which consists of only two blocks of the shapes ‘O’ and ‘T’ as given Tetris manual. We have provided them with some custom colours too.
A queue variable has been added which will always display the next blocks to be dropped into the running game. controls have been initialized which specifies the controls available to the player. The rest of the code sets up a game engine and runs an iterative while loop which will keep getting updating the state of our game engine in the background.

Custom made Scanner code has been provided for the example text file, which will list all the tokens. The scanner code will be refined more in future with time for more detail & additions.

- In order to avoid matching lexemes between <identifier> lexeme tokens and custom designed lexemes such as that for <block_list>,<block> etc, the scanner is first is designed in such a way to first identify custom lexemes before identifier lexemes

3. Division of Labour between Parser and Scanner

- The parser will take the token stream generated by the scanner and generate parse trees for interpreting the meaning.
- The responsibility of error handling for invalid tokens will be given to the parser
- The parser will also have the responsibility of mapping the meaning of commands extracted out from our game programming language to pygame commands.

4. Division of Responsibility

Scannner -Abhinav,Ishan
Parser - Rahi, Vaibhav
Mapping of Parser extracted information from with pygame - Aryan,Jay

Objective1: Implement a scanner design in sly for processing of tokens

Objective2: Implement a parser to generate syntax trees to interpret the meaning out of the game language.
Objective 3: Mapping the pygame commands to our game language