Ng Chi Hon 1155116317 csci3100 HW4

1.(1) Initialize_interface : the game area and answer area appear
Select_Difficulty: Select Difficulty and change the block fall rate
Fall_Block: Block fall every second and number of block falling is according to the difficulty
Choose_question: the question appear in the answer area
Check_Answer : check if answer are correct, return true
Dissolve_block : Dissolve the block and add score to scoreboard
Check_Gameover: Check Gameover condition

(2)

Refinement 1:

Initialize the game interface with Game area and answer area, and allow player to choose difficulty

```
Initialize_interface
ask player to select difficulty
Select_Difficulty
```

Refinement 2:

```
set score as 0
if score database is NULL
    set best as 0
if score database is not NULL
    set best as score database
set Gameover as False

repeat Check_Gameover

until Gameover is true
return score
```

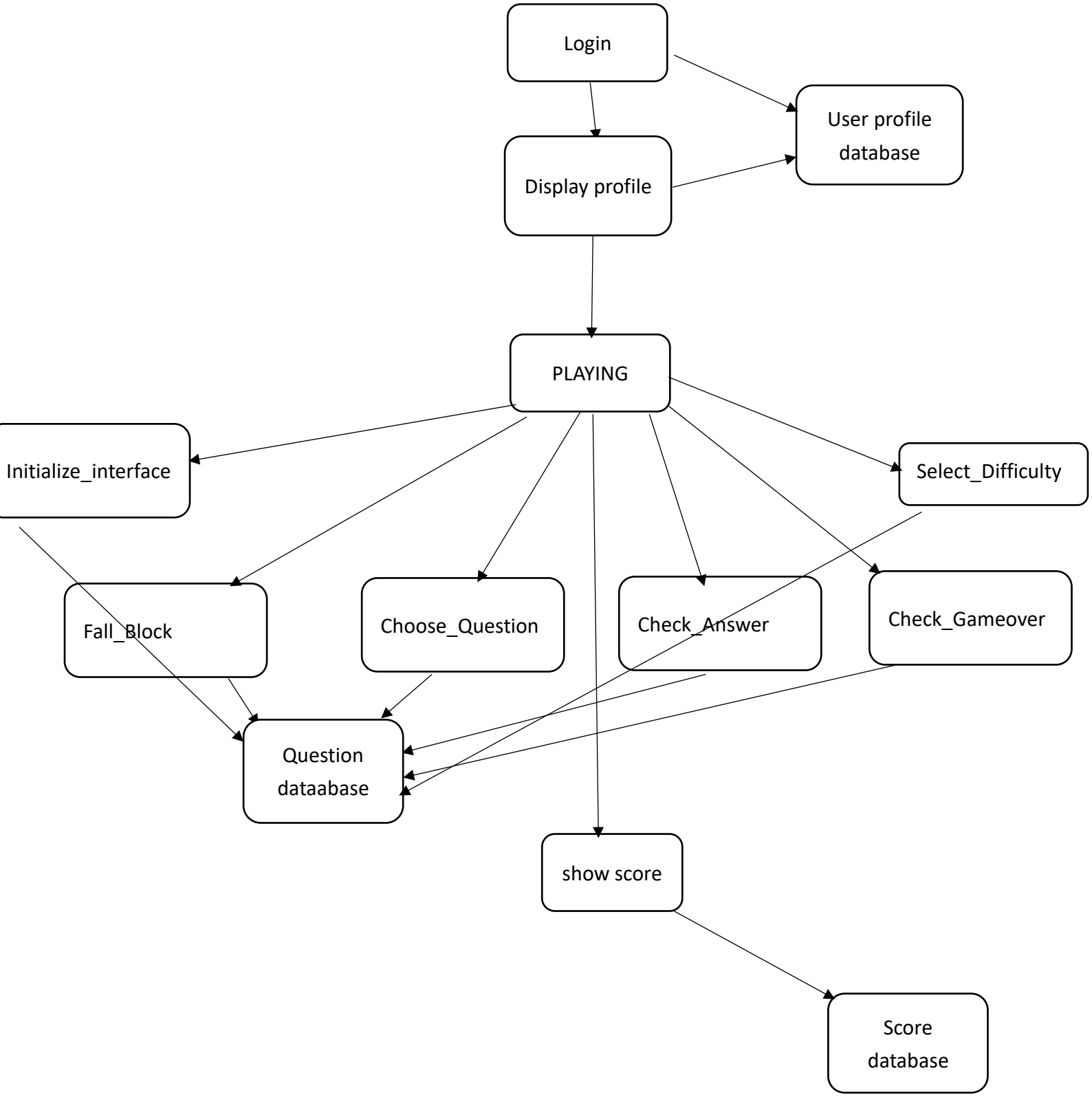
Refinement 3:

```
(Refinemnet 3 is in loop of refinement 2)
repeat Fall_Block every second
    if player click any block of the question
        Choose_question
        set answer as false
```

```
                if player type the answer
                    Check_Answer
                    if Answer is correct
                        Dissolve block
                        score add 100
                    end
                    if Answer is not correct
                        answer turn red colour
                    end
                end
            end
        until Check Gameover as true
```

```
Module BLOCK_HANDLER

export BLOCK : ?;

    function CLICK _STATE (S: in out BLOCK) : Boolean
    function COLOUR    (S: in BLOCK) : string_of_char
    function value (S: in BLOCK) : INTEGER

end BLOCK_HANDLER

Module Question_HANDLER

export Question : ?;

    function questionvalue (S: in out Question) : array (1..4) of integer
    function Fallstate (S: in Question): Boolean

end Question_HANDLER
```

Q2(1)


Login
User profile database
Display profile
PLAYING
Initialize_interface
Select_Difficulty
Fall_Block
Choose_Question
Check_Answer
Check_Gameover
Question dataabase
show score
Score database

Q2(2)

Module Draw_Question

uses getQuestion, Question_HANDLER

exports
    type questionvalue: array (1..4) of integer
    var Drawseed : array (1..4) of integer

    procedure Draw_Question (database: in DB_address ; Drawseed; in out Drawseed; in out questionvalue)
implementation
    is composed of drawBlock
end Draw_Question

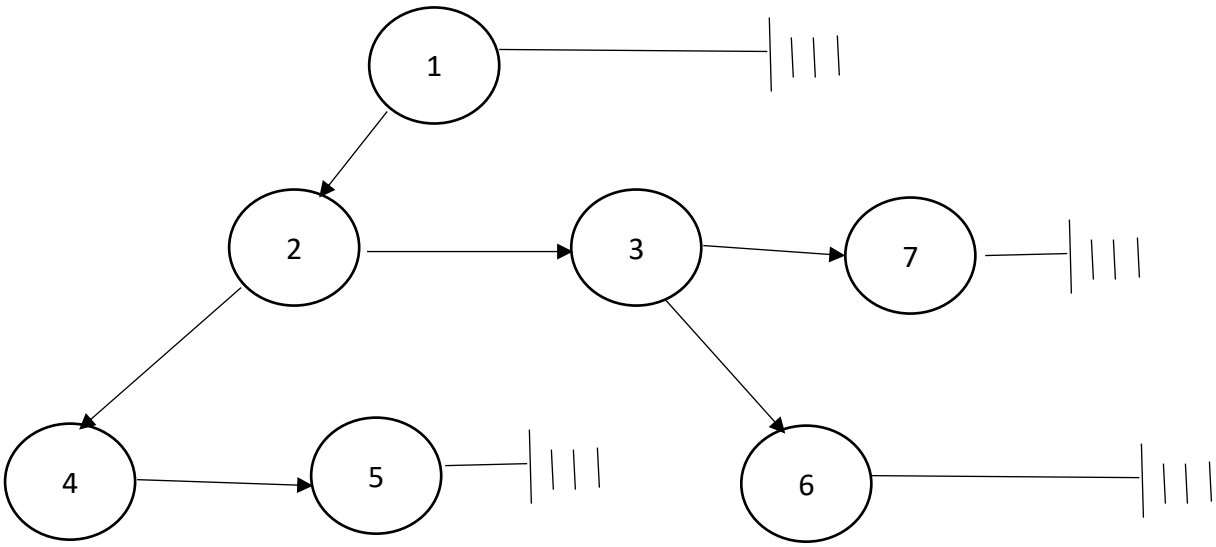
Q3(1)

Time complexity is O(n)

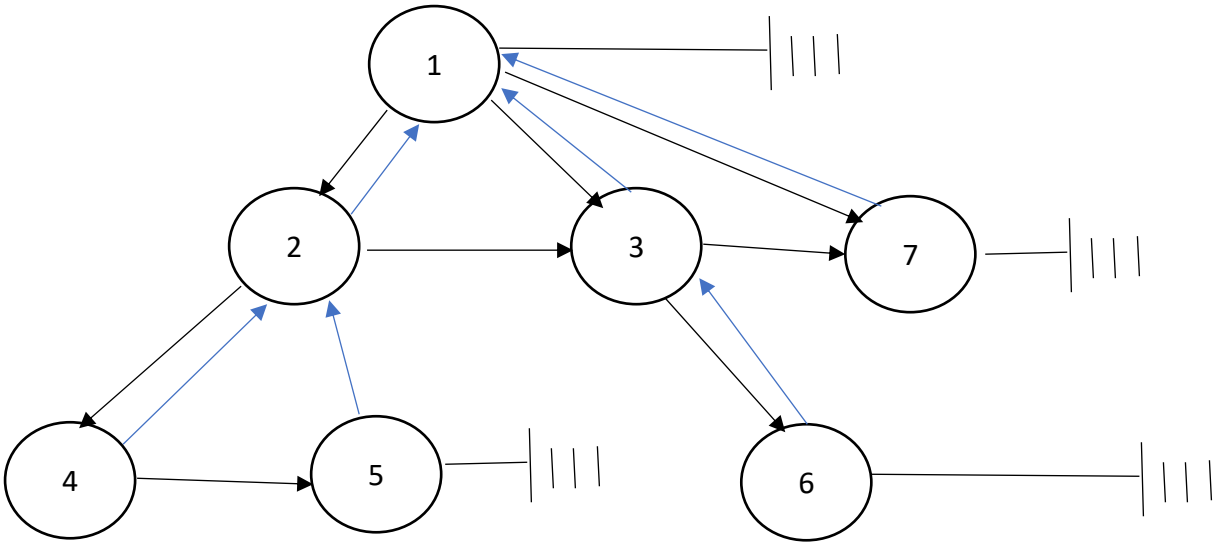
```
Int findLCA
{
    If (root == NULL) return NULL;

    If (root.key == n1 || root.key == n2) return root;

    Node *left_lca    = findLCA(root->left, n1, n2);

    Node *right_lca = findLCA(root->right, n1, n2);
if (left_lca && right_lca)    return root;
return (left_lca != NULL)? left_lca: right_lca;
}
```

Q3) (2) (a)

The original tree can represent in tree and parent pointer tree in both slide 5-14 and 5 -15. For slide 5-14 tree, the two pointer not point to its left right child. One pointer will point to left most child and other point will point to the next sibling.



For slide 5-15 tree, not just two pointer as slide 5-14 tree, we add one more pointer to point to its parents.



The difference of slide 5-14 and 5 -15 tree is that the one parent pointer allow 5 -15 tree to do work more efficiency in traverse because when we visit to low level node, we don't need to trace back it parent with many step code. We can directly trace back node parent immediately.

Q3) (2) (b)

```
public void DFS(Node root) {
        if (node == NULL)
        return;
        DFS (node->right_ sibling);
        DFS (node->leftmost_child);
    }
```

Q3) (2) (c)

```
public void DFS(Node root) {
        if (node == NULL)
        return;
        DFS (node->right_ sibling);
        DFS (node->leftmost_child);
    }
```

Q4 (1) (1): upper_bound (2) lower_bound (3) lower_bound (4) root.val-1
(5) root.val+1 (6) upper_bound (7)root

Q4 (2) (1) NULL (2)current -> val (3) current -val (4) previous->right (5) current

(6) current -> val (7) current -> val (8)`current->right`

Q4 (3) Recursion is that when function invoke itself and it has to allocate memory on the stack for argument, variable for each time recusion. If the recursion time is too large, the memory may not enough, use up or over the recursion depth and make it crash.

To solve it, we should set personal recursion depth and if it overrun, save the data as file and release the memory.