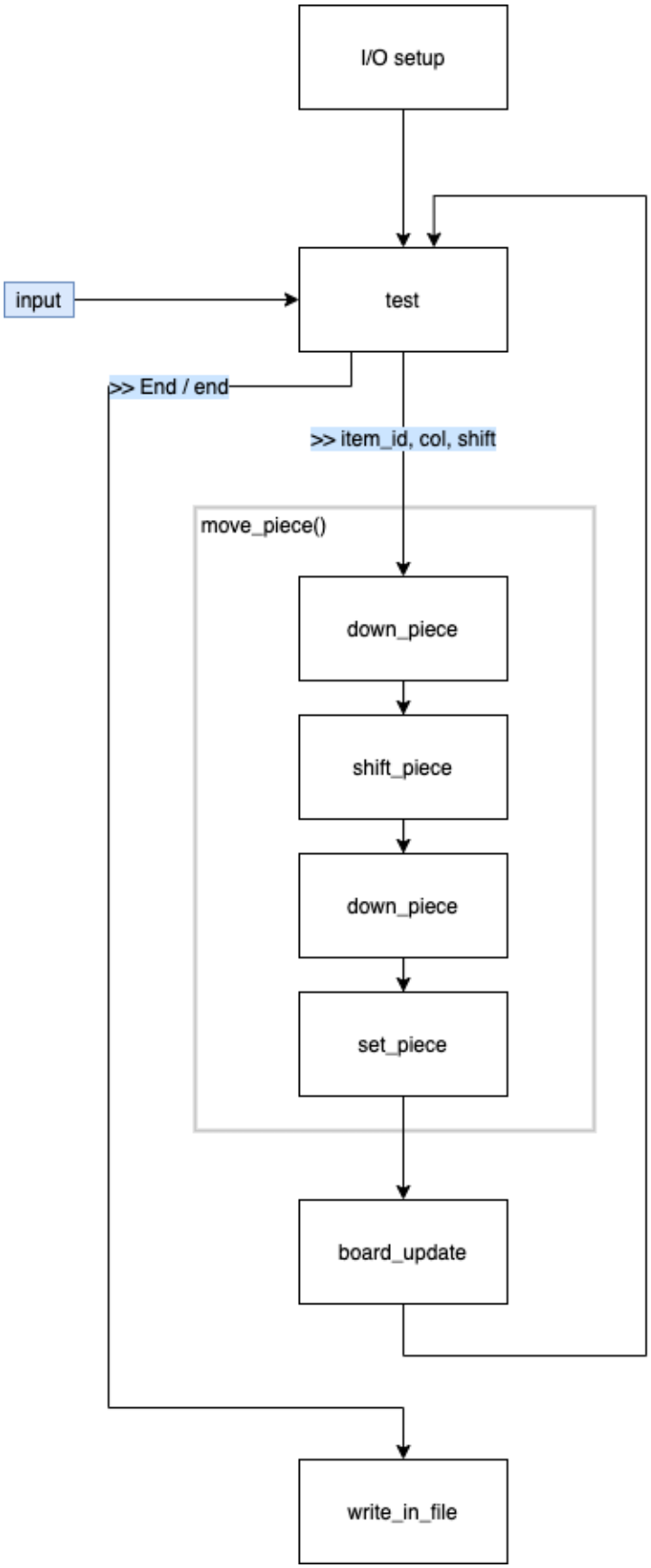
108062308_proj1

學號：108062308

姓名：黃俊瑋

Github：https://github.com/tangerine1202/NTHU-EECS204002-Project1-Tetris

Project Description


I/O setup
input
test
>> End / end
>> item_id, col, shift
move_piece()
down_piece
shift_piece
down_piece
set_piece
board_update
write_in_file

Flow Chart

設計說明

名詞定義：

- Board：儲存整個 board 的資訊，大小為 m * n。
- Piece：儲存一塊落下方塊的資訊。

設計概念

整體程式可以分成兩大部分，第一，移動 Piece 並在確認後將資訊儲存到 Board 中；第二，計算 Board 的下落與消除。

在最 trivial 的解法中，以下操作需要大量運算：

1. 比對 Piece 是否與 Board 上存在的格子重疊（暴力比對 O(4*m)）
2. 更新 Board 下落與消除後的結果（更新 O(m*m*n)）

因此在設計時針對這兩大部分優化。

加速重疊比對

如果使用 Array 儲存，每次確認重疊都需要 O(4) 的時間。因為最大 Board 寬度僅 40，因此我們可以使用 bitset 的方式儲存資料，並使用 and 操作比對是否有重疊。在 B 作法上，此作法改進微乎其微，但在 A 作法的部分，因為上方落下的列可以落進下方的列，此時原作法 O(n)，但使用 bitset 做是 O(1)，因此此方法在 A 作法時可以有效改善時間複雜度

更新 Board 下落與消除後的結果

如果使用 2D Array 儲存 Board，每次消除時都讓上方列下落，更新時間複雜度 O(m*m*n)，時間複雜度極差。此處引入 stack 的概念，每次將未消除的列的 index 放入 stack 中，列的完整資訊則保留在原本 2D Array 上，如此確認最後剩餘列的時間 O(m)，更新版面 O(m*n)，大幅改進時間複雜度。在 A 作法上，因為會將可下落的列寫入下方列內，因此退化成 (m*n) 更新列資訊及確認最後剩餘列，(m*n) 更新版面。

功能實作方法說明

Piece 運作

主要步驟為：

1. 建立 Piece（ `Piece constructor` ）
2. 落下至停止（ `Board::down_piece` ）
3. 橫移至停止（ `Board::shift_piece` ）
4. 落下至停止（ `Board::down_piece` ）
5. 將資訊寫入 Board 中（ `Board::set_piece` ）

其中 `down_piece` 與 `shifht_piece` 有做重疊與左右邊界檢查。

重疊檢查

對應 function 為 `Board::isNotOverlapping`

主要利用 bitset 的優勢，使用 and O(1) 內即可知道是否有重疊發生。

左右邊界檢查

對應 function 為 `Board::isInHorizontalBound`

雖然實際上不用做，不過為提高程式功能的完整性，仍然選擇實作出來。

此處是 bitset 的主要缺點，因為使用 shift 移動時，如果超出邊界 bit 會直接消失，難以檢查。我選擇將 bitset 大小開為 Board 寬的兩倍，同樣利用超出邊界 bit 會直接消失的特性，如果需檢查右移是否出界，則比對「直接右移完，再移回原位」及「先左移一個 Board 的寬度，再右移，再移回原位」的結果，如果會超出右界，則前者會有 bit 消失，結果將與後者不同。

下落與消除機制

對應的 function 為 `Board::update()`。

此機制重點在於使用 stack 儲存可以維持目前最上方的列，簡化 A 作法之運算，也讓其可以兼容兩種作法。實際邏輯請參考下方 pseudo code。

```
# pseudo code
l = current line idx (from bottom to top)
s = stack

for l from bottom to top:

    if l is full or l is empty
        continue
    else if s is empty
        s.push(l)
        continue
```

```
    # A 作法 start
    else if l can fit into s.top
        board.fit_into(l, s.top)
        if s.top is full
            s.pop
        continue
    # A 作法 end

    s.push(l)
```

其他輔助功能說明

Hyper Parameter

此份檔案支援討論區所提到之 A, B 作法，及 info, debug 相關的 output。

在檔案開頭的 `#define` 中可以開關上述功能，以下簡單說明：

- `#define FINAL_SUBMIT`

  確認為最後上傳的檔案，會將所有開發用輸出關閉，只留下最後題目要求的輸出，並將其導入到 108062308_proj1.final 檔案。

- `#define A_SOLUTION`

  程式預設為 B 作法，即題目原先要求之作法。開啟此選項可以改為 A 作法。

- `#define VERBOSE`

  輸出程式運作相關（info, warning, error）資訊 到 terminal。

- `#define DEBUG_ENABLE`

  輸出 debug 使用的資訊到 terminal。

I/O 設定

當 commend 不帶任何參數值，使用 terminal input 作為程式 input，並輸出到 output.final。

當 commend 只帶一個參數值，使用此參數作為 input.file，並輸出到 {input_file_name}.final。

當 `FINAL_SUBMIT` 開啟時，強制使用題目規定之 I/O。

Special Input

加入 `end`, `show`, `show_more`, `show_all` commend，方便 debug。

Utility Function

`info`, `warning`, `error`, `debug` 四項會將傳入的 string 加上有顏色的前綴後輸出到 terminal，顯示整體運作過程，並加速 debug 的流程。

Test case Design

因為這份 code 兼容 A, B 作法，因此設計了一份 A, B 作法輸出不同的 test case。

此為消除前的結果（右方 `|` 為邊界）

```
1      1 |
111  111 |
11111111.|
   11   .|
11111111.|
111  111.|
```

之後在 `.` 插入一直槓。

B 作法輸出應為：

```
         |
         |
1      1 |
111  111 |
   11   1|
111  1111|
```

A 作法輸出應為：

```
         |
         |
         |
         |
1      1 |
111  1111|
```