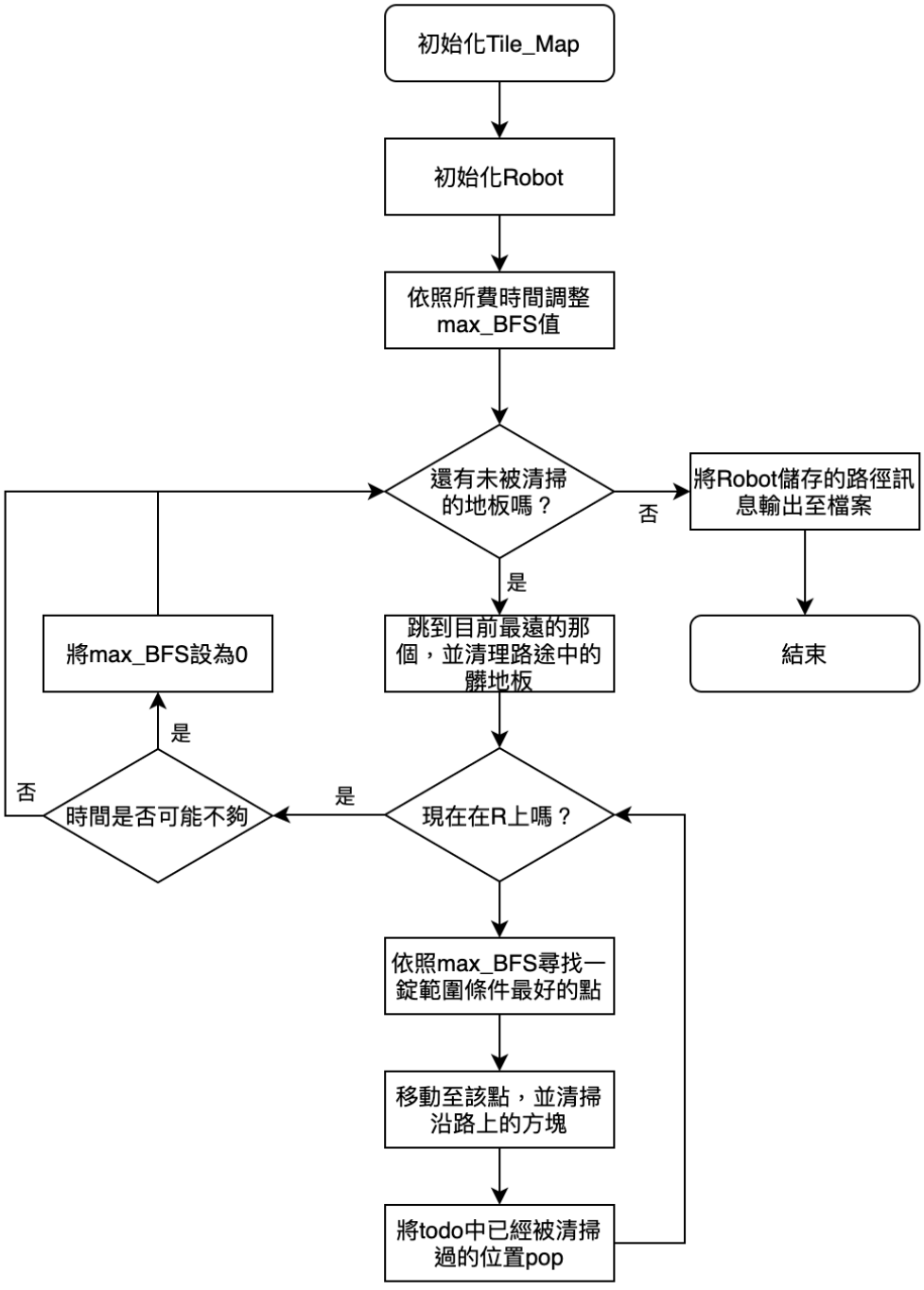
壹、 Project Description

一、 Program Flow Chart



二、 Detailed Description

1. 概述

本程式主要由兩大部分組成，包括記載地圖資訊的 tile、tule_map 類別，以及負責走路打掃的 Robot 類別，接下來將由各個類別的成員變數和函式為切入點來介紹。

2. class position

(1) 概述

用於描述二維整數坐標這個在程式中會反覆使用到的概念，很小巧的類別。

```
14  struct position{
15      int row;
16      int col;
17
18      position(int r=-1, int c=-1):row(r), col(c){}
19      bool operator!=(position &other){
20          return (row != other.row) || (col != other.col);
21      }
22      bool operator==(position &other){
23          return (row == other.row) && (col == other.col);
24      }
25      //0=右，1=左，2=上，3=下
26      position stepwise_move(int dir){
27          return position(row+direction[dir][0],col+direction[dir][1]);
28      }
29  };
```

(2) 成員變數

變數	型別	簡述
row	int	表示其列座標
col	int	表示其行座標

(3) 成員函式

函式	型別	簡述
position	-	建構子，設定其行列值
operator==	bool	回傳另個 position 值是否與自己相同
operator!=	bool	回傳另個 position 值是否與自己不相同
stepwise_move	position	依照傳入值，回傳當下位置上、下、左、右相鄰位置之中一個的值

3. class trio

(1) 概述

用於 BFS，作為被塞進 queue 的型別，包含位置、步數、路徑紀錄

```
31  struct trio{
32      position pos;
33      int step;
34      vector<position> related;
35      trio(position p = position(), int s=-1, vector<position> v = vector<position>());
36  };
```

(2) 成員變數

變數	型別	簡述
pos	position	位置
step	int	步數
related	vector<position>	目前為止經過的位置，依照順序紀錄（不包括起點）

4. class tile

(1) 概述

代表一格地板，紀錄與單格地板相關的資料。

```
38  class tile{
39  public:
40      int minstep;
41      int type;
42      bool cleaned;
43      int search_visited;
44      position pos;
45      vector<position> related;
46  public:
47      tile(){}
48      void set(int t, int row, int col);
49      void clean();
50  };
```

(2) 成員變數

變數	型別	簡述
minstep	int	與充電點的最短距離
type	int	該格的類型，包括 floor、wall、recharge 三種
cleaned	bool	代表該格是否被清掃過了
search_visited	int	在 tile_map 的 find_uncleaned 中，紀錄一個 search id，用於分辨是否在該輪的 BFS 中拜訪過了
pos	position	代表該格的位置
related	vector <position>	由充電站走的最短走法，會經過的所有位置（包括充電站的位置，但充電站本身例外，其 related 為空）

(3) 成員函式

函式	型別	簡述
tile	-	default constructor
set	void	初始化各項變數，並設定 pos、type 值
clean	void	使該格被清掃，將 cleaned 改為 true

5. class tile_map

(1) 概述

以 tile 類別為基礎，組織完整的地板地圖，並提供一些功能協助 robot 運作。

```
53  class tile_map{
54  public:
55      tile **map;
56      position Rpos;
57      stack<position> todo;
58      int cols, rows;
59      int B;
60      int walkable_num;
61      int search_id;
62  public:
63      tile_map(ifstream infile);
64      bool is_walkable(position pos);
65      void calculate_minstep();
66      tile& get_tile(position p, const char* origin);
67      void print_out(int t);
68      trio find_uncleaned(position init, int battery);
69  };
```

(2) 成員變數

變數	型別	簡述
map	tile**	代表地圖的二維陣列
Rpos	position	充電點的位置
todo	stack <position>	儲存「尚未清理」地板的 Stack，距離充電點越遠越上方
rows	int	map 的總列數
cols	int	map 的總欄數
B	int	roobot 的最大電池容量，方便處理輸入用
walkable_num	int	可走的總格數
search_id	int	在 find_uncleaned 中使用，代表特定某輪的 BFS

(3) 成員函式

函式	型別	簡述
tile_map	-	建構子，從.data 檔案中讀取地圖資料
calculate_minstep	void	以 BFS 計算地圖上所有 tile 的 minstep、related 值
find_uncleaned	trio	回傳該位置附近一定範圍內最好 tile 的位置、距離、路徑
is_walkable	bool	回傳該位置的 tile 是否可以走（非 wall 且不超出邊界）
get_tile	tile&	取得該位置的 tile 資料，參數包括一個標示來源的字串，方便 Debug
print_out	void	將 map 印出來，Debug 用

(4) 精選函式詳述

- void calculate_minstep()

以 BFS 的方式計算地圖上所有可以走的 tile 的 minstep 值和 related 資料，會在建構子中呼叫。先將充電點的位置塞入 trio 的 queue 中，其 trio 值位置為充電點位置、step 為 0，related 為空向量。

進入迴圈後，先取出 queue 最前的 trio，若該位置的 tile 未被拜訪過(minstep 為-1)，則將該位置的 tile 之 minstep 設為 trio.step，related 為 trio.related（不包括自己）。接下來往四個方向擴散，若該格可以走而且還沒被走過，就將其位置、當下 step+1，以及 related 加上當下位置，做成 trio，丟進 queue 中。

然後將 trio.pos 加入 todo 中，如此一來待會開始清掃移動時，stack 就會由最遠到最近取出可以走的 position。

最後將 queue 的最前 pop 掉，若 queue 已經空了，就跳出迴圈，反之則繼續執行。

最後的最後，將 walkable_num 設為 todo 的大小。

```
104  void tile_map::calculate_minstep(){
105      queue<trio> Q;
106      trio tp;
107      vector<position> vp;
108      while(!Q.empty()) Q.pop();
109      Q.push(trio(Rpos,0));
110      get_tile(Rpos,"root clean").cleaned = true;
111      while(!Q.empty()){
112          tp = Q.front();
113          vp = tp.related;
114          vp.push_back(tp.pos);
115          if(get_tile(tp.pos,"calculate_minstep3").minstep == -1){
116              get_tile(tp.pos,"calculate_minstep1").minstep = tp.step;
117              get_tile(tp.pos,"calculate_minstep2").related = vp;
118              for(int i=0;i<4;i++){
119                  if(is_walkable(tp.pos.stepwise_move(i)) && get_tile(tp.pos.stepwise_move(i),"calculate_minstep4").minstep==-1){
120                      Q.push(trio(tp.pos.stepwise_move(i),tp.step+1,vp));
121                  }
122              }
123              todo.push(tp.pos);
124          }
125          Q.pop();
126      }
127      walkable_num = todo.size();
128  }
```

- trio find_uncleaned(position , int)

與 calculate_minstep 的概念類似，但是是在行走過程中，在任意位置上尋找附近最值得前往的 tile，而一旦找到一個較近的未清潔地板，就會停止向外搜索，只與同距離的比較。

首先會先使 search_id 遞增，作為這一輪搜尋的代號。將起始位置旁邊的四個鄰居做成 trio 加入 queue。進入迴圈後，若發現當下的位置的 minstep 比當下的 best 還大，就跳離迴圈，然後檢查是否電池狀況不允許前往該位置，或是否在這輪已經拜訪過了（tile 的 search_visited 是否跟 search_id 相同），若是則直接前往下一輪。

接下來要看這個位置是否被清潔過，若被清過了，他就只需要往下傳播而已，若沒被清過，就不需要往下傳播，而跟當前最佳解比較，看看是否成為最佳解。接下來，兩種狀況都會將 tile 的 search_visited 設為 seach_id，最後 pop 掉。

若 queue 為空或是超出了 max_BFS 限制的迴圈次數，就結束並將最佳解回傳，否則繼續下一輪。

```cpp
trio tile_map::find_uncleaned(position init, int battery){
    queue<trio> Q;
    trio tp;
    trio best;
    vector<position> vp;
    tile tt;
    int count = 0;
    search_id++;
    while(!Q.empty()) Q.pop();
    for(int i=0;i<4;i++) if(is_walkable(init.stepwise_move(i))) Q.push(trio(init.stepwise_move(i),1));
    // Q.push(trio(init,0));
    get_tile(Rpos,"root clean").search_visited = search_id;
    while(!Q.empty() && count++<max_BFS){
        tp = Q.front();
        if(best.step != -1 && best.step < tp.step) break;
        vp = tp.related;
        vp.push_back(tp.pos);
        tt = get_tile(tp.pos,"find_uncleaned1");
        if(tt.search_visited != search_id && tp.step + tt.minstep < battery-1){
            if(tt.cleaned){
                for(int i=0;i<4;i++){
                    if(is_walkable(tp.pos.stepwise_move(i))){
                        Q.push(trio(tp.pos.stepwise_move(i),tp.step+1,vp));
                    }
                }
            }
            if(best.step == -1 || tile_compare(tt,get_tile(best.pos,"find_uncleaned2"))){
                best = tp;
                best.related.push_back(tp.pos);
            }
        }
        tt.search_visited = search_id;
        Q.pop();
    }
```

6. class robot

(1) 概述

本類別代表實際上在地圖上移動的掃地機器人，對地圖進行清潔並記錄自己的行動軌跡。

```cpp
class robot{
public:
    int maxbattery;
    int battery;
    position Rpos;
    position pos;
    vector<position> footprint;
public:
    robot();
    void walk();
    void jump();
    void hop();
    bool is_on_recharge();
    void print_out(ofstream& ofs);
};
```

(2) 成員變數

變數	型別	簡述
maxbattery	int	代表其電池滿電量
battery	int	電池當前電量
Rpos	position	充電點的位置
pos	position	當下位置
footprint	vector <position>	從開始到結束所經過的所有位置

(3) 成員函式

函式	型別	簡述
robot	-	建構子
walk	void	走，在上、下、左、右鄰居中比較，移動一格
jump	void	大跳，從 Recharge 的位置跳躍到 todo 最上方的位置
hop	void	小跳，運用 find_uncleaned 找出範圍內最值得移動的格子
is_on_recharge	bool	判斷是否在充電點上
print_out	void	將 footprint 的東西輸出成檔案

(4) 精選函式詳述

- void walk()

本函式代表「走」的行動，比較上下左右四個方向之後，選擇最適合者異動過去。

首先就是找到適合的方向，該方向必須能走，其到充電站的距離也不能超過當下電量減 1，接下來，若是第一個符合者就直接先設為最佳解，若非第一個，則要與既有的最佳解用 tile_compare 函式進行比較，若較好則成為最佳解。

找好方向之後就移動過去，將移動後的位置加入 foot_print，然後判斷是否要回復或減少電量。

```
210 void robot::walk(){
211     tile_map &tm = *TileMap;
212     position best, tpos;
213     //找到適合的移動方向
214     for(int i=0;i<4;i++){
215         tpos = position(pos.row+direction[i][0], pos.col+direction[i][1]);
216         if(tm.is_walkable(tpos) && tm.get_tile(tpos,"robot walk1").minstep <= battery-1){
217             if(best.row == -1 || tile_compare(tm.get_tile(tpos,"robot walk2"), tm.get_tile(best,"robot walk3"))){
218                 best = tpos;
219             }
220         }
221     }
222     if(best.row == -1){
223         printf("Error: robot cannot move, battery=%d\n", battery);
224         exit(4);
225     }
226     //移動至最好的位置
227     pos = best;
228     if(best != Rpos) footprint.push_back(best);
229     if(!tm.get_tile(best,"robot walk4").cleaned) tm.get_tile(best,"robot walk4.5").clean();
230     //減少電量
231     if(is_on_recharge()) battery = maxbattery;
232     else battery--;
233     if(battery <= 0){
234         printf("Error: robot battery exhausted\n");
235         exit(5);
236     }
237 }
```

- void jump()

本函式代表「大跳」的動作，從充電點跳至 tile_map 中 todo 最上方的位置，總是在充電點上方時執行，如此可以保證每個 tile 都被清掃過。

首先取出並 pop 掉 todo 最上方的位置，將該 tile 的 related 從頭到尾跑過一遍，clean 並加入 footprint。接著移動到該位置並減少相當於 related.size()-1 的電量，就結束了。

```
195  void robot::jump(){
196      tile_map &tm = *TileMap;
197      if(tm.todo.empty()){
198          printf("Error: jump when todo stack empty\n");
199          exit(8);
200      }
201      tile &target = tm.get_tile(tm.todo.top(),"robot jump2");
202      tm.todo.pop();
203      for(position p:target.related){
204          footprint.push_back(p);
205          if(!tm.get_tile(p,"robot jump3").cleaned) tm.get_tile(p,"robot jump3").clean();
206      }
207      pos = target.pos;
208      battery -= (target.related.size() - 1);
209  }
```

- void hop()

本函式代表「小跳」的動作，透過 tile_map 中的 find_uncleaned 函式，找出附近最值得去的位置，然後跳過去。

首先就是執行 find_uncleaned，找出目標位置，若 find_uncleaned 沒能找到合適的，就會將工作交由 walk 函式執行。

若有找到合適的位置，則執行類似於 jump 的作法，不過其要加入 footprint 的，是回傳的 trio 資料中所記載的路徑，而且因為有可能碰到充電點，電量的增減也要考慮充電的情況，而因為 find_uncleaned 的路徑中，除了目標以外，不會經過未清掃過的 tile，因此不需要逐個清掃。

```
238  void robot::hop(){
239      tile_map& tm = *TileMap;
240      trio best;
241      best = tm.find_uncleaned(pos,battery);
242      if(best.step == -1) {
243          walk();
244      }
245      else{
246          if(!tm.get_tile(best.pos,"robot hop3").cleaned) tm.get_tile(best.pos,"robot hop2").clean();
247          for(int i=0;i<best.related.size();i++){
248              footprint.push_back(best.related[i]);
249              if(best.related[i] == Rpos) battery = maxbattery;
250              else battery--;
251          }
252          pos = best.pos;
253      }
254  }
```

7. 公用精選

(1) 變數精選

變數	型別	簡述
TileMap	tile_map*	本程式中唯一的 tile_map 物件，為了使 robot 型別方便取得其資訊，放在 global 中
max_BFS	int	用於限制 find_uncleaned 中迴圈循環的次數，根據 TileMap 建構的時間設定

start	clock_t	紀錄程式開始時的 clock()值
last	clock_t	用於其他需要紀錄 clock()值時

(2) 函式精選

函式	**型別**	**簡述**
tile_compare	bool	判斷兩個 tile 適合移動過去的程度，未清潔過的較優，若相同則距離 todo top 的 minstep 層級較近者較優，若皆相同則回傳 false，通常代表不替換
set_maxBFS	void	根據 TileMap 建構的時間設定 max_BFS 的大小

8. main 函式

如文件開頭的流程圖，首先將 TileMap 和 Robot 初始化，接著調整 BFS 值，接著就是反覆清掃至掃完為止。

在清掃進行的 while 迴圈中，每次都是始於充電點，終於充電點，一開始先用 jump 跳到很遠的地方，然後再以一個小的 while 迴圈多次執行 hop 直到回到充電點，walk 只在 find_uncleaned 失敗時於 hop 內部呼叫執行。每次 hop 完都會嘗試清理一次 todo，用 while 迴圈 pop 直到頂端的位置是未清掃的。

而每一輪，都會查看時間是否有不夠的可能，若時間緊迫，會將 max_BFS 設為 0，也就是每次都執行 walk，雖然路徑較長，但速度快很多。

全部清掃完畢之後，就輸出檔案，並結束。

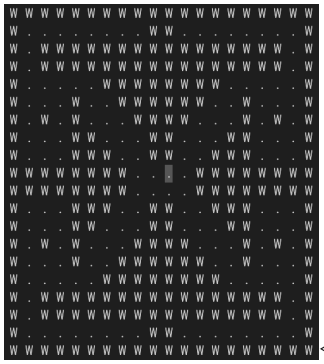
有些步驟未寫在流程圖中的，大多是除錯用的，而有關時間的紀錄也會在 main 中進行。

貳、 Testcase Design

一、 Detailed Description of the Test case

本測資設計的較小巧，主要是希望測試在有只有最短距離來回才能清掃到的點時，掃地機器人是否能正確找到路徑，並且在電量不耗盡的情況下成功清掃完成。

參考資源回收符號的設計，以對稱為原則。設置了四個在邊緣的單行道，限制掃地機器人的行動。



以較清楚的方式呈現，牆為「W」，可走的地為「.」，反白處為 R