**基本資料**

|  |  |  |  |
| --- | --- | --- | --- |
| 使用者 | 高子翔 | 選擇使用的類別庫 | 中華小當家 |
| 使用前成績 | Bronze/490名 | 使用後成績 | Bronze/498名 |
|  | | | |

**量化指標**

|  |  |  |  |
| --- | --- | --- | --- |
| 大項指標 | 細項指標 | 同意打 √ | 總評  (1-5) |
| 易讀性 | 類別拆解方式（有哪些類別）符合直覺。 | √ | 4 |
| 類別、變數與函式名稱清楚，符合直覺。 | √ |
| 單個函式內的程式碼量適中、功能單一明確。 |  |
| 避免使用難推測其意義的字面常數。 | √ |
| 避免使用特別炫技且難以理解的程式碼。 | √ |
| 程式碼使用一致的風格與模式。 |  |
| 程式碼結構儘量扁平。 | √ |
| 在有需要的地方提供適量清楚的註解。 |  |
| 使用性 | 提供足量、適切且容易選取的（成員）函式以達任務需求。 |  | 4 |
| 函式參數個數適中、易於分辨其意義。 | √ |
| 容易除錯：避免使用全域變數。 | √ |
| 容易除錯：善用 const/constexpr。 | √ |
| 修改擴充性 | 適時使用迴圈與陣列，擴充資料量時不需修改太多程式碼。 |  | 5 |
| 善用具名常數，易於調整數值。 | √ |
| （成員）函式：實作與介面切割清楚，修改時不會牽動過多其它程式碼。 | √ |
| （成員）函式：功能單一明確，修改時不會牽動過多其它程式碼。 | √ |

**質性意見**

感覺使用說明可以更加詳細，雖然已經很多東西，但是如class Table內的Finditem()就沒有特別說明會改動到item\_position，也可能是我太菜看到referance沒有馬上反應過來，但感覺可以加個說明。

如果有更多的範例程式，使用起來也會更加快速，因為大部分都在看source code想這個函式在做什麼，雖然manual有簡略說明，但實際能拿來幹嘛都是不太確定的。後來也是詢問了才知道FindTable()很好用。

使用時有遇到幾個bug，或是說需要使用者額外去考慮的點：

1. 原先Chef::PutOnTable()如果周圍剛好都不是空桌子的話，會沒有動作，後來應急直接隨便給了個地方讓角色移動就解決了，如果真的要解決的話應該要想辦法去尋找最近的空桌子放，可是我好懶。

|  |
| --- |
| void **PutOnTable**() const {  int dir[8][2] = {{0, -1}, {0, 1}, {1, 0}, {-1, 0},  {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};  for(int d = 0; d < 8; ++d) {  int new\_x = x\_ + dir[d][0], new\_y = y\_ + dir[d][1];  if (kitchen\_.**IsTable**(new\_x, new\_y) && table\_.**IsEmpty**(new\_x, new\_y)) {  **Use**(new\_x, new\_y);  return;  }  }  } |
| void **PutOnTable**() const {  int dir[8][2] = {{0, -1}, {0, 1}, {1, 0}, {-1, 0},  {1, 1}, {1, -1}, {-1, 1}, {-1, -1}};  for(int d = 0; d < 8; ++d) {  int new\_x = x\_ + dir[d][0], new\_y = y\_ + dir[d][1];  if (kitchen\_.**IsTable**(new\_x, new\_y) && table\_.**IsEmpty**(new\_x, new\_y)) {  **Use**(new\_x, new\_y);  return;  }  }  **Move**(WINDOW);  Return;  } |

1. 在Chef::GetIngredients()拿取原本廚房不含，而是有一定加工的材料時，會因在Kitchen::GetCoordinate()內回傳的(-1, -1)造成距離上的誤判。如角色位於(1, 1)，要拿取位於(7, 2)的RAW\_TART時會先因為其原本不存在於廚房，得到回傳值(-1, -1)，後來跟桌上物品比對距離還是(-1, -1)比較近，於是便輸出Move -1 -1而造成錯誤。

後來在距離判定前加上如果原本廚房不存在，則直接看桌子上該物品的位置。

|  |
| --- |
| std::pair<int, int> **GetCoordinate**(const std::string &item) const {  if (item\_coordinate\_.**count**(item)) {  return item\_coordinate\_.**at**(item);  }  else {  return std::make\_pair(-1, -1);  }  }  std::pair<int, int> **GetIngredients**(std::string ingredient) const {  auto pos = kitchen\_.**GetCoordinate**(ingredient);  std::pair<int, int> p;  if (table\_.**FindItem**(ingredient, p)) {  if (**Distance**(p) < **Distance**(pos)) {  pos = p;  }  }  return pos;  } |
| std::pair<int, int> **GetCoordinate**(const std::string &item) const {  if (item\_coordinate\_.**count**(item)) {  return item\_coordinate\_.**at**(item);  }  else {  return std::make\_pair(-1, -1);  }  }  std::pair<int, int> **GetIngredients**(std::string ingredient) const {  auto pos = kitchen\_.**GetCoordinate**(ingredient);  std::pair<int, int> p;  if (table\_.**FindItem**(ingredient, p)) {  if (pos.first == -1 || **Distance**(p) < **Distance**(pos)) {  pos = p;  }  }  return pos;  } |