Sprout 大作業二 - iceylemon

作者：iceylemon

我在這題的策略並不是最佳解，這邊主要是提供一些優化的想法給大家思考。非常鼓勵各位同學們自己發揮想像力想出更好的策略！(btw，想知道餐點的步驟可以直接看程式碼)

程式碼說明

MovingPipeline class

上課的時候有講過 `MoveCounterToCounter` 系列函式。這系列的函式雖然好用，但有一個缺點：我們要自己人工判斷要從哪個座標 (Counter) 開始移動，也就是要自己記得上一個座標 (Counter) 在哪裡。

但在這份作業我們會想要建立一個「流水線」，所以自動紀錄上一個座標 (Counter) 的功能應該會蠻實用的，還可以節省人工判斷起點的麻煩。因此我實作了 `MovingPipeline` 這個 class 來做的這些事：

1. 呼叫 `AddToPipeline` 來新增一個座標 (Counter) 到流水線，並把對應的路徑加到 `operations`
2. 採用 `vector` 資料結構來維護要移動的系列座標，這樣印出路徑來 debug 會相對方便
3. 重載 `AddToPipeline` 函式來支援各種不同參數
    - e.g. 給他一個 `vector<pair<int, int>>` 叫他找離起點最近的座標

MovePointToPoint 優化

我在 `MovePointToPoint` 這個函式的實作跟上課講的有兩個不一樣的地方：

1. 走完之後直接面對方向終點 point 的方向
2. 特別決定先走左右還是先走上下

第一點的意思是，通常我們如果想要移動，都會是移動到某個 Counter，所以我在這個函式中額外新增了「面對終點的 Counter」的這個步驟。第二點的原因則是因為，我希望能夠讓角色作盡量少的轉向動作，所以我希望在最後一次移動的時候角色剛好就面對終點 Counter。舉個例子，如果我想要從 $(1, 20)$ 去跟一個在 $(8, 13)$ 的 Counter 互動，最後需要面對下方才能跟他互動。如果先往下走，再往左走，最後還要再額外轉一次向。但如果是先往左走，再往下走，就可以少一次轉向的步驟。

MakeSalad

我把 `MakeSalad` 切成了作「前半」跟「後半」的 `Salad`。分別對應的函式是 `MakeFirstHalfSalad` 跟 `MakeLastHalfSalad`。並且我額外維護了「前半」沙拉是否有被完成的變數 `firstHalfSalad`，跟是否有完整的沙拉完成了被放在某個普通桌子的變數 `isSpareSalad`。

前半的沙拉作的事情是拿盤子 + 切番茄；後半則是切生菜

MakeBurgerGeneral

因為三種漢堡的食譜都差不多，只差在 `CheeseBurger` 要加上 `CheeseSlices`，`MegaBurger` 則要再額外加上一份 `Salad`。所以我實作了這個函式來簡化這三種漢堡的程式碼 (重複利用 code)。實作方法為，讓函式吃兩個 `bool` 參數 `cheese` 跟 `mega`，分別代表需不需要加 `CheeseSlices` 跟是不是 `MegaBurger`。

MakeSaladSCMode

專門設計給 `SaladAndCheeseBurger` 模式的 `MakeSalad` 函式。利用了原本的 `MakeSalad`，再提交餐點後移動到指定位置。

MakeCheeseBurgerSCMode

專門設計給 `SaladAndCheeseBurger` 模式的 `MakeCheeseBurger` 函式。因為這個模式製作餐點的順序比較不同，我認為重複利用 `MakeCheeseBurger` 會讓 code 變得比較難以閱讀，所以我另外寫了一個新的。

Recipe Mode 策略

Salad Mode

- 期望分數：18000
- 標準差：0

Salad Mode 因為只會出現 `Salad` 的訂單，所以遊戲沒有任何隨機性。因此程式碼每次執行的結果都會是一樣的。從 `DeliveryCounter` (座標 $(1, 20)$) 開始，`Salad` 只需要 89 回合即可做完 + 送出。

因此，只需要做完 `Salad` 之後，在 `DeliveryCounter` 前面等，等到 100 的倍數回合提交就可以達到完美 18000 分。

Salad and CheeseBurger Mode

- 期望分數：17500 (理論值)
- 標準差：500
- 最大值：18000
- 最小值：17000

從 `DeliveryCounter` (座標 $(1, 20)$) 開始， `CheeseBurger` 只需要 76 回合即可做完 + 送出。

因為 `Salad` 跟 `CheeseBurger` 都可以在 100 回合內做完，所以只要每次都在拿到 Order 的時候馬上開始作就可以拿到 17000。

如果想要拿到 18000 的話，其中一個難點在於我們不知道最後一筆訂單到底會是 `Salad` 還是 `CheeseBurger`，而我們必須在最後一回合 (1800) 的當下提交訂單，所以我們不可能確保我們可以拿到最後一筆訂單。因此，理論上拿到 18000 的機率只有最高 50。

因為前 99 回合我們無法得知下一筆訂單為何，所以我們可以充分利用這段時間來先作一道餐點放在旁邊。最後在第 1800 回合的時候拿這個餐點提交出去。

我這邊選擇先作一個 `CheeseBurger`，並且放在 $(8, 20)$、在下排的 `NormalCounter`。原因在於從 `DeliveryCounter` 出發僅需 17 步即可將其拿起並送出。

但 `Salad` 需要 89 步才能做完，若在得知最後一筆已知訂單是 `Salad` 之後 (在第 1700 回合生成的訂單)，按照原本的方法去作的話會來不及送出最後一個 `CheeseBurger` ($89 + 17 > 100$)。因此，我們需要略微修改 `Salad` 的實作方法。

注意到如果從 `DeliveryCounter` 出發的話，89 步的優化空間應該不多，所以一個直觀的想法是：「不要從 `DeliveryCounter` 出發」。但如果不從 `DeliveryCounter` 出發，`CheeseBurger` 的製作又會亂掉，或是甚至超過 100 回合導致拿不到滿分。所以我們亦需要調整 `CheeseBurger` 的實作方法。

我想到的方法是：「先拿好一個盤子，然後移動到 $(2, 13)$，接下來從 $(2, 13)$ 出發開始作餐點」。原因在於 $(2, 13)$ 往上跟往下走剛好都是會用到的 `NormalCounter`。這樣在得知下一個訂單是 `Salad` 還是 `CheeseBurger` 之後就可以節省一點步驟。

用以上的策略，再進行一些實作上的微調 (詳細請看 `MakeSaladSCMode()` 跟 `MakeCheeseBurgerSCMode()` 兩個函式)，就可以成功把兩者的剩餘回合都壓到 17 以內，最後一次就可以成功提交出 `CheeseBurger`，達到期望值 17500 的分數。

All Recipe Mode

以下結果為進行 8 次實驗後的結果，訂單生成 seed 如公布的 spreadsheet。

- 平均分數：14187.5
- 標準差：2590.89
- 最大值：17000
- 最小值：9600

`Burger` 的製作不會超過 `CheeseBurger` 次數，大約 70 次。`MegaBurger` 就比較麻煩，不管怎樣都無法壓到 100 回合以內 (可以壓到 150 以內，也就是連續來兩個 `MegaBurger` 可以拿到 1400 分，但後面的菜單就要自己努力了)。所以我們勢必會需要利用空閒的時間來準備一些 `MegaBurger` 會需要用到的材料。

我的作法是，如果有空的話就去作 `Salad`。因為 `MegaBurger` 可以想像成是先作一個 `Salad` 再作一個 `CheeseBurger` 疊在一起。所以可以利用之前製作 `Salad` 的實作方法。

這樣的好處在於，如果要作 `MegaBurger`，可以很快的先做好 `Salad`，然後再把剩餘的材料加進去就好。而且如果訂單是 `Salad`，那麼就可以直接提交出去，然後再作一個新的把原本提交出去的補回來。經過計算，若已經有做好一個 `Salad` 放在 $(8, 13)$ 的 `NormalCounter`，那麼從 `DeliveryCounter` 出發，可以在 100 回合內做完一個 `MegaBurger`。

但直接作 `Salad` 會有一個缺點，就是 `Salad` 需要的步數太多了。做完一個 `Salad` 會 delay 到後面的餐點的製作。因此，我這邊將 `Salad` 的步驟分為「前半」跟「後半」。前半是先切番茄，後半則是再把生菜補上。如此一來，如果做完一筆訂單還有剩餘的時間，就可以先作「前半」的 `Salad`，或是如果前半已經做好的話可以作「後半」。經過一些計算，可以找出一個適當的步數，讓角色在空閒步數超過某個數字之後就開始先作餐點，這樣就可以大幅利用等待訂單的時間。

以上的討論大多是侷限在如何利用等候訂單的時間加速 `MegaBurger`，但我們討論的都是從「最早出現的訂單開始作」。但如果我們發現訂單大塞車了 (很多訂單卡住了，例如連續出現 3 個 `MegaBurger`)，這個時候不見得從最早出現的訂單開始作會拿到最高分。因此，我設定了如果當下卡了 3 個訂單，就優先從 700 分以上，並且不是 `MegaBurger` 的訂單開始作。考量在於，如果訂單已經變成 350 分，那麼最慘也只會再少 150 分。那還不如先去作 700 分的訂單。

關於訂單選擇的問題我沒有著墨太多，此方法應該還有加強的空間，例如搭配餐點製作步數加上分數去作枚舉最佳解等等。若同學們有興趣可以自己思考。