

# DSAP期末專題報告

隊名：會計好難 組員：B10705005 陳思如、B10705033張禾家

## 一、方法

主要沿用沿用StraightFowardController的想法，大部分的函式內容架構都是相同的。其中AngleToSymbol()有多增加一個轉彎次數的參數，因為我們的Snake.Direction()經過不同的轉彎角度後，有時會有負的情況，要將目前角度加上 $360^{\circ}$ (轉彎次數)，在判定行進方向時才會以正的角度判定，如此一來才不會造成錯誤或誤判。

我們CustomController中 NextDirection() 則將不同關卡類別和情況分開來決定，共分為五種：

```
class CustomController : public ISnakeController {
public:
    DirectionType NextDirection(const Game&, size_t) override; //蛇要走的下個方向
private:
    enum class DirectionSymbol { //方向
        RIGHT, DOWN, LEFT, UP, NONE
    };
    DirectionType _nextDirection; //下個方向
    DirectionType _currentDirection; //現在方向
    float _final_angle; //轉彎後角度
    DirectionSymbol _dirSymbol; //此刻蛇頭方向
    DirectionSymbol AngleToSymbol(float, long long); //把角度轉換成方向

    float turn_radius = 3 * 180 / 3.1415926 + 30;
    float GetCollisionDistance(Position, DirectionSymbol, const Game&, size_t); //障礙物距離
    float GetFrontCollisionDistance(Position, float, DirectionSymbol, Position, float); //與其他蛇相距最小距離
    float FrontWallDistance(Position, DirectionSymbol, float, float); //與牆壁相距最小距離
};
```

### 1. 關卡A

場上只有自己一條蛇，並且所有食物和蛇的初始的位置都是由rand()函數決定，那我們相信是絕對沒有極端分布情況的，所以決定採用盡可能走遍整張地圖的方法來吃完食物得分。而走遍整張地圖有很多方法，包括撞到牆迴轉或是以逐漸越繞越小圈來走遍。經過這兩種方法的實驗，我們發現如果採用迴轉的方法，中間會因為迴轉半徑而浪費許多食物。所以決定採用越轉越小圈的方法。其中我們設定一個變數counforturn來紀錄蛇總共轉了幾次，根據counforturn，我們能夠在完全轉完一圈，也就是counforturn是4的倍數後，需要增加他要轉彎的判定距離，完成越轉越小圈的目的。最後經過一些實驗和數字的調整，發現因為蛇自己身體本身的半徑也要納入考量，因此將轉彎距離設定為turn\_radius +  $120^{\circ}(\text{counforturn}/4) + 75$ ，其中120代表他每轉一圈會縮減的距離，而75則是第一次轉彎必須先具備的最佳距離參數。

### 2. 關卡B1、關卡C1

因為已經知道亂數種以及蛇的數量位置等等詳細訊息，所以我們採用客製化的方式來操控，先以id為1的那條蛇出發點位置來判斷是否為公開測資，再來以蛇的數量來判斷為B1或者是C1。

#### 2.1. B1(初始為6條蛇)

記錄下轉彎的次數稱作為cntB，當我們操控的蛇與前方不管是其他蛇或者是牆壁的距離小於turn\_radius+ $120^{\circ}((\text{cntB}\%10)/4)+30$ 時，我們會讓他轉彎。因為是客製化極端測資，我們在所有嘗試中發現第1、7、8次因為左邊空間比較大所以選擇左轉90度，第16次也採取左轉則是因為剛好可以淘汰右上角的蛇，其他次則是右轉90度，然後當碰撞超過20時上面兩隻蛇皆已被殺掉，就可以放心以S型的方式把中上半部的食物吃光，採取轉彎次數若奇數次轉左邊180度

角，偶數次則轉右邊180度角，當第27次時我們則已經走遍上半部的地圖，所以改成換另一個方向的S型來走遍上半部的地圖，補足因迴轉半徑而沒吃到的食物。

### 2.2. C1(初始為5條蛇)

記錄下轉彎的次數稱作為 **countforturnC**。我們會進行轉彎，當我們操控的蛇與前方障礙物的距離小於 **turn\_radius + 15\*(countforturnC/4)**。而經過嘗試發現，在需要轉彎的情況下，第3次進行左轉90度角，其他次進行右轉90度角，最後可以把其他蛇殺死。當場上剩下最後一條蛇時，就回到類似A關卡的模式，可以盡情把場中食物吃完，所以回到A關卡的移動方法。

## 3. 其他關卡

除了已知測資的B1、C1關卡外，我們利用B、C不同的特性設計出分別的移動方法，而兩兩對戰因為蛇的位置不固定所以我們也將他歸類於C關卡。

### 3.1. B類型

這類關卡共有五條蛇，分別分布在四個角落和正中間，所以我們就同樣以五條蛇來判定為B類關卡。因為其他蛇的位置是固定的，所以我們也打算盡可能的走遍整張地圖，但因為有其他蛇作為障礙物，所以地圖不會是一個矩形，在規畫一個越走越小的路徑會變得較為困難。因此我們決定採用上下迴轉的方法來遍歷整張地圖，只有在第一次和走到角落時採取轉90度，其他皆為轉180度。同時也因為採用迴轉的方法，我們需要增加判定原本行走的方向來決定迴轉的方向，否則蛇將會一直在同個路徑上重複移動。在每次轉彎時也會確認將要轉的方向是否有邊界或蛇在他的迴轉半徑內，如果有的話，我們就會轉到另一個方向並且只轉90度，避免蛇重新回到已經走過的路徑。

### 3.2. C類型以及其他

因為這類型的關卡所有物品都是一開始無法隨機的，所以我們決定規劃一個較為通用的方法來避免特化的移動行為。我們主要參考 **StraightFowardController** 的想法，當遇到前方有障礙物時就決定要轉彎，那轉彎的方向我們目前規劃以同一個方向為主，除非將要轉彎的方向在轉彎半徑內有任何會相撞的物品，不論是邊界或其他蛇。因為 **GetCollisionDistance** 判定的距離的只能使用上下左右四個固定的方向，無法滿足我們想要求得轉彎半徑內的圓圈是否有其他物品，所以借用 **game.cpp** 中判定死亡的方法，先透過 **IsCollideWithCircle**，以轉彎半徑作為半徑判定，來看是否這個轉彎半徑內的圓圈有障礙物，有的話我們則再進一步依據行進的方向來看這個在圓圈範圍內的物品時否會在轉彎的路徑上，依此決定下一步轉彎的方向。最後，若幸運地將場上其他的蛇殺光的話，我們則將回到A類關卡的方法，在剩餘時間內遍歷整張地圖，盡可能吃光食物。

## 二、心得

張禾家：

這是今年第二次使用SFML寫小遊戲，雖然因為不熟悉，還不是很會操作關於如何編譯等等的問題，一開始也遭遇很多難關，但透過和同學討論以及上網找資料還有助教以及教授在討論區上的幫助後，成功編譯了這個專案。再加上因為時間在期末的關係，除了這個專案也有許多其他科的考試要準備，所以一開始非常擔心時間不夠，但意外的是開始跟組員一起討論想法後，這件事變得不再困難，反而很有趣。看著那條蛇在畫面中照著自己寫的步驟爬行，覺得很好玩。

陳思如：

我覺得這次的專案是一個很特別的經驗，透過一個已經設定好的環境架構，從中改變一些行為和決定的參數來影響最終的結果。雖然專案進行中有遇到蠻多困難，像是不知道怎麼獲取資料或是蛇的移動路徑不如預期等等，但最後也因為和組員一起討論後，找到可行的方法，而且透過討論來增加更多蛇的可能性。雖然我們這次還沒找到一個很普遍的走法來讓每種關卡都有平均的表現，這可以是我們之後努力的目標，同時我也很好奇其他同學的解法，因為同個問題感覺每個人都會有不同的想法，綜合其他人的想法感覺會讓這個遊戲更有趣！