《計算機概論與程式設計期末專題報告》

題目：Crazy Arcade

組員：113511103 吳沛恩、113511216 洪崧祐

一、 前言

這份專題使用 C++實現《瘋狂炸彈人》的遊戲。我們設計的遊戲包含單人模式與雙人模式，並結合了電腦玩家 AI 控制以增進遊戲的挑戰性與趣味性。這份作業花費了我們很多個期末前的日日夜夜，更花了我們無數的心思在其中，並利用物件導向程式設計的技巧進行撰寫，並持續優化遊戲體驗與增加內容。為了此專案我們自學了 OOP 中物件的封裝與繼承技巧，讓程式更好維護與理解、路徑尋徑演算法 (BFS) 以及使用者介面設計等，成功打造出一個完整且可擴展的遊戲系統。

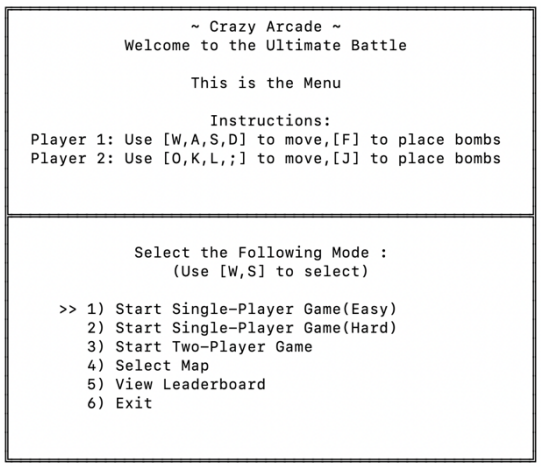
二、 基礎遊戲介紹

（一） 遊戲流程說明

1. 玩家透過放置炸彈來擊敗對手，並在地圖中收集道具來增強自己的能力。
2. 主要遊戲模式：單人模式（AI 對戰）和雙人模式（玩家對戰）。
3. 遊戲初始條件：玩家的初始生命值設定為 3 條。每當玩家被炸彈的爆炸範圍擊中，就會損失一條生命。如果生命值降至 0，玩家將被淘汰，遊戲結束。如果雙方玩家同時失去所有生命，則判定為平局。
4. 遊戲結束條件：任一玩家生命值歸零，或時間耗盡。

（二） 道具介紹

1. 增加玩家分數道具 （綠色星星★）
2. 增加玩家生命值 （紅色愛心❤️）
3. 玩家可同時放置的炸彈數量+ 1 （黑桃♠️）
4. 炸彈範圍+ 1 （黑色齒輪符號❁）



圖一 遊戲目錄介面

（三） 計分方式

分數計算規則與玩家的得分基於以下幾種情況：

1. 擊敗對手：成功利用炸彈擊中對手時，玩家將獲得 100 分的獎勵。
2. 收集道具：
   - 加分道具（綠色星星★）：每個加分道具能增加 50 分。
   - 其他功能性道具（如增加生命、炸彈數量、爆炸範圍）則可獲得 20 分。
3. 存活時間：遊戲結束後，存活時間會作為排行榜中的一個附加因素，用來區分分數相同的玩家。

（四） 地圖設計炸與彈爆炸規則

地圖一共分成三種：簡易版地圖、中等版地圖、困難地圖（隨機生成障礙物），地圖由固定障礙物、可破壞障礙物、道具以及空地組成。

玩家的炸彈初始爆炸範圍為 3 格。炸彈爆炸時，爆炸路徑上的可摧毀障礙物（Destructible Obstacles）都會被破壞，破壞後有機率會隨機生成道具可供玩家搜集。每個範圍增加道具（❁）會讓炸彈的爆炸範圍額外增加 1 格。

三、 核心功能實現

3-1. 專案中的檔案功能介紹

透過物件的封裝與繼承我們將專案的程式碼分成 13 個檔案，包含標頭檔、原始碼以及純文字檔。以下將詳細進行介紹：

```
├── main.cpp          # 程式進入點
├── GameCrazyArcade.h / .cpp  # 遊戲邏輯
├── AIController.h / .cpp     # AI 控制邏輯
├── GameObject.h / .cpp       # 遊戲物件管理
├── Menu.h / .cpp             # 使用者介面
├── Globals.h / .cpp          # 全域變數
├── leaderboard.txt           # 領導榜單
└── animation.txt             # 動畫效果
```

圖二 專案檔案架構

- **遊戲核心邏輯：GameCrazyArcade.h / GameCrazyArcade.cpp**

負責遊戲的初始化、遊戲循環與炸彈管理。控制遊戲的開始、結束條件檢測，以及玩家的分數和生命管理。以下的 Player 為玩家物件。主要功能函數：

1. gameInitialization(Player, Player)：初始化狀態與控制鍵。
2. drawGame(Player, Player GameObject)：繪製遊戲畫面。
3. updateBombs(Player, Player, GameObject)：更新炸彈狀態及爆炸範圍。
4. GameStart(Player, Player, GameObject, Menu)：啟動遊戲循環，檢查勝負條件。

- **AI 玩家控制邏輯：AIController.h / AIController.cpp**

功能：控制電腦玩家（AI）的行為，我們設計的 AI 是以 Finite State Machine 來實現，包含了以下五種狀態：逃避炸彈、等待炸彈爆炸、收集道具、攻擊玩家、閒置巡邏，會根據當下地圖的情況在這五種狀態進行切換。同時，我們也使用了 BFS 廣度優先搜索演算法尋找最短路徑。主要功能函數：

1. updateState(Player, Player)：如同 AI 玩家的大腦，會根據遊戲環境調整不同狀態。
2. handleEscapeState(Player)：尋找安全點並逃離炸彈爆炸的危險區域。
3. handleAttackPlayerState(Player, Player)：追蹤玩家位置並放置炸彈攻擊。
4. handleFetchItemsState(Player)：收集地圖上的道具。
5. bfsFindPath()：使用 BFS 演算法計算路徑。

以上五個為主要負責控制的函數，還有其他 19 個輔助的功能性函數不在此多介紹。

- **遊戲物件管理：GameObject.h / GameObject.cpp**

功能為管理遊戲中的玩家狀態、炸彈狀態與排行榜與基本的輸入輸出功能，例如鍵盤制、玩家操作等。主要功能函數為：

1. loadLeaderboard()：讀取排行榜。
2. updateLeaderboard()：更新排行榜。
3. displayLeaderboard()：顯示排行榜。
4. setConioTerminalMode() / resetTerminalMode()：設定/重置終端機輸入模式。

- **使用者介面：Menu.h / Menu.cpp**

此檔案負責遊戲主選單的控制、地圖選擇與遊戲初始與結束動畫等。提供視覺化的效果，增強遊戲的使用者體驗。主要功能函數：

1. initial_animation()：開場動畫。
2. player1_win_ani() / player2_win_ani() / ai_win_ani()：勝利動畫。
3. DeathEffect(Player &p1, Player &p2, int)：遊戲結束效果。
4. Display_Main_Menu(int)：顯示主選單。
5. Map_Selection(vector<string>, int)：地圖選擇介面。

- **全域變數與設定：Globals.h / Globals.cpp**

功能：儲存所有遊戲的全域變數和設定參數。包括地圖尺寸、遊戲符號、玩家控制鍵等。所有其他的檔案都會使用到這些全域變數。其中主要的參數有：

1. int mapWidth / int mapHeight：地圖寬度與高度。
2. string borderSymbol / string obstacleSymbol：邊界與障礙物符號。
3. char p1_up, p1_down, p1_left, p1_right, p1_bomb：玩家 1 控制鍵。
4. char p2_up, p2_down, p2_left, p2_right, p2_bomb：玩家 2 控制鍵。
5. vector<vector<string>> gameMap：遊戲地圖陣列。

- **純文字檔：leaderboard.txt / animation.txt**

前者用來儲存前十名玩家的姓名、分數和遊戲時間。後者用來儲存遊戲的開場動畫、勝利動畫和結束效果的 ASCII 字元。

3-2. 專案中的物件導向程式設計

設計遊戲的過程中，我發現透過物件封裝與繼承的方式可以更容易去維護各種函數以及增加新的功能。右圖為本專案各物件之間彼此的互動關係，GameObject 為基礎類別（Base Class），用來提供所有遊戲物件的基礎功能，管理遊戲物件的基本資訊

圖三 專案中物件彼此的互動關係

（例如玩家狀態、排行榜、炸彈狀態）、提供排行榜的存取和操作功能以及處理終端機輸入輸出（例如玩家移動控制）。Menu 繼承自 GameObject，GameCrazyArcade 繼承自 AIController，同時 AIController 繼承自 GameObject。透過設定不同變數與函數的權限（Public, Protected, Private），以達成良好的程式權限管理。

此外，檔案與物件彼此之間需要相互依賴才能正常運作，透過 #include 檔案的方式來實現。下圖為個別檔案與依賴對象的關係表。

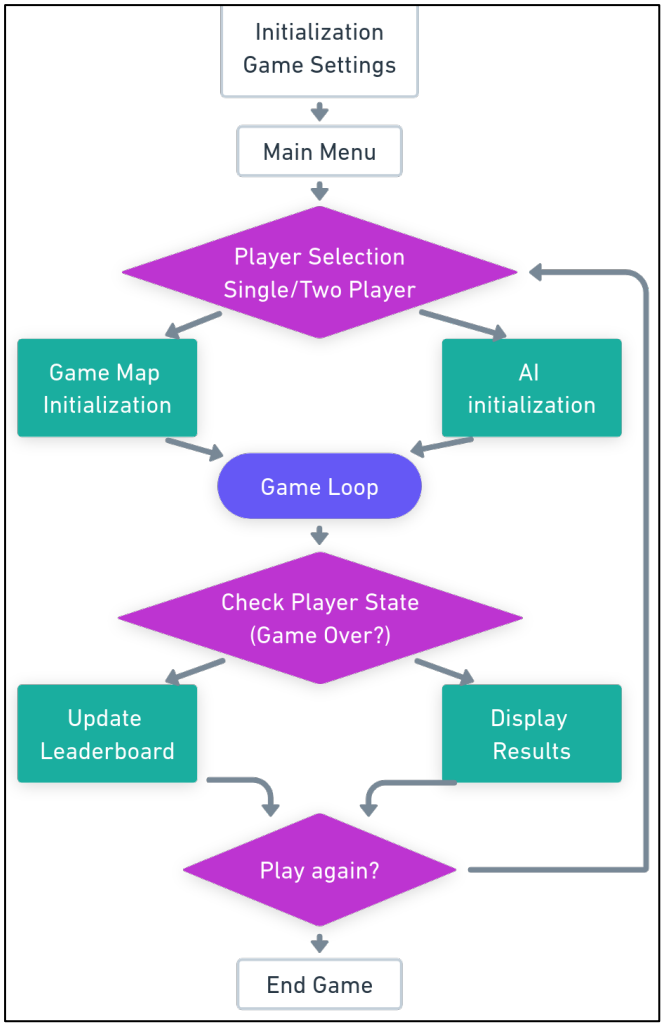
依賴類別/檔案	依賴對象	依賴原因
main.cpp	GameCrazyArcade	管理遊戲流程與邏輯。
	GameObject	載入與儲存排行榜數據。
	Menu	顯示選單和動畫效果。
	Globals	使用全域變數來設定遊戲參數。
GameCrazyArcade	AIController	繼承 AI 行為邏輯。
	GameObject	使用玩家和排行榜功能。
	Menu	顯示遊戲相關動畫與更新。
AIController	GameObject	訪問玩家屬性與遊戲地圖。
GameObject	Globals	訪問全域設定與配置。
Menu	GameObject	使用玩家狀態與遊戲設置。
	Globals	訪問全域符號與地圖數據。

圖四 專案中物件彼此依賴關係

3-3. 程式執行的流程圖

右圖為遊戲的流程圖。首先是初始化的動作，創建一個 GameObject 物件，並同時將排行榜進行更新。接著玩家可以在 Menu 中選擇不同的對戰模式，單人遊戲或是雙人對戰。若是雙人對戰，玩家可選擇不同難度的地圖或是隨機生成的地圖。若是單人與 AI 對戰，玩家可選擇簡易模式或困難模式（AI 移動速度的不同）。也可以選擇查看 LeaderBoard 的分數排名。

選擇玩之後，遊戲進入初始化的動作。創建玩家 Player 物件，地圖初始化設定如隨機生成障礙物與各種道具。進入主遊戲迴圈之後，若玩家選擇雙人模式，遊戲會判斷是否有鍵盤輸入並按照指令移動玩家。若為單人模式，AI 則是會根據當下地圖與人類玩家的位置，在不同狀態進行切換。最後，當有勝負出現時，遊戲會請贏家輸入名字，若為前 10 名則會將分數與使用時間紀錄在遊戲的記憶中。



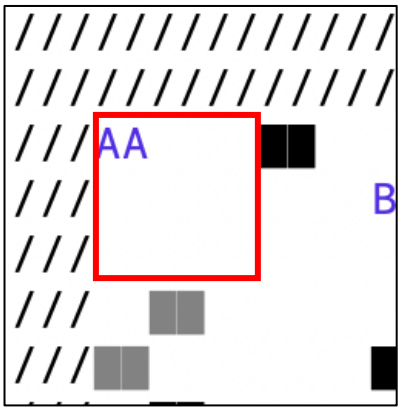
圖五 程式執行流程圖

四、 亮點功能介紹

基礎的加分項目我們皆有完成，我們在此針對幾項特別的功能與我們製作專案時遇到比較大的問題進行介紹。

（一） 隨機生成地圖

我們希望遊戲地圖能夠在每次啟動時隨機生成，但這樣會導致一個問題：玩家初始位置周圍可能被障礙物完全包圍，無法移動。嘗試了許多方法後，我們最後決定在隨機生成地圖時，設置了玩家保護區。即玩家初始位置周圍一定範圍內（3×3）不會生成障礙物或炸彈。同時，我們讓地圖中障礙物的分布遵循一定的機率規則，避免地圖太過空曠或擁擠。



圖六 隨機地圖之保護區

此外，針對道具的部分，可破壞的障礙物中有 80%的機會會有道具在其中，在之中會有 50%是加分道具、10%為增加生命道具、30%為增加炸彈數量道具而 10%增加炸彈範圍道具。機率的部分可以按喜好進行設定。

（二） 使用者互動介面

為了增進遊戲體驗，我們花很多心力進行遊戲介面的設計以及改良（User Interface, Menu System），也加上了動畫效果。主選單提供遊戲模式選擇、排行榜查看等功能。遊戲動畫部分我們手刻了開場動畫、玩家勝利動畫等。在地圖選擇部份，我們也提供不同難度和風格的地圖選項、並設計了不同顏色的道具增強玩家體驗。最後，我們也設計了 Leader Board，讓玩家可以知道自己的分數在歷史的排名，並有機會被記錄在其中。

（三） 廣度優先搜尋演算法（BFS Algorithm）

為了讓電腦 AI 玩家尋找到玩家、道具或是到安全躲避地點的最短路徑，我們利用廣度優先搜尋（BFS）找出路徑，讓 AI 能支援避開危險區域與規劃安全撤離路徑。

運作流程：AI 根據當前狀態選擇目標點（例如安全區域、道具或玩家位置）。使用 BFS 計算到目標點的最短路徑。根據計算結果，逐步移動到目標點。使用 BFS 有以下亮點：

1. 即時計算：在遊戲循環中快速計算路徑，確保 AI 反應迅速。
2. 避開危險：自動避開炸彈爆炸範圍，減少 AI 失誤率。
3. 靈活性高：可根據不同狀態規劃不同的路徑策略。

（四） AI 狀態機設計 (Finite State Machine Design)

原本設計的 AI 玩家是每一步都根據當前的狀態去計算最短路徑，然後決定下一步的移動方式。實踐之後發現，這並不是一個理想的做法，因為地圖的狀態時時刻刻都在變化，而 AI 玩家很常出現逗留在原地或是躲不了炸彈的狀況。因此我們決定採用不一樣的邏輯，以狀態機的方式設計，定義不同狀態，排出在不同環境條件下的各狀態的優先順序。

例如，當遇到附近有炸彈時，最優先的事情就是躲避炸彈，尋找一個合適安全的地點。再來是攻擊玩家，若當下沒有被炸彈炸到的危險，尋找到玩家的最短路徑並在合適的時刻放下炸彈。再來是炸開障礙物去獲得道具。

以下會針對 State Machine 中三個重要的要素：1. 狀態 State：AI 在特定時間下所處的行為狀態；2. 轉換條件 Transition Condition：狀態之間的切換條件；3. 行為 Action：每個狀態中執行的動作，進行介紹。

● 本專案中 AI 的五種狀態（State）與行為

▼表一　本次設計的五種狀態名稱

狀態名稱	狀態行為	切換條件	切換後的狀態
ESCAPE	逃避炸彈爆炸範圍	周圍存在炸彈，且在爆炸範圍內	WAIT_EXPLOSION, FETCH_ITEMS
WAIT_EXPLOSION	等待炸彈爆炸，停留在安全區域	炸彈爆炸倒計時結束	FETCH_ITEMS, ATTACK_PLAYER
FETCH_ITEMS	收集地圖上的道具	附近有道具，或沒有立即的威脅	ATTACK_PLAYER, ESCAPE
ATTACK_PLAYER	追擊並攻擊玩家	玩家在可攻擊範圍內，且有炸彈可放置	ESCAPE, WAIT_EXPLOSION
IDLE	閒置或隨機移動	沒有明顯威脅或可執行的行動	ESCAPE, FETCH_ITEMS

- **轉換條件 (Transition Condition)**

在程式中 updateState 這個函數如同 AI 的大腦，判斷環境在以下狀態中轉換。

1. ESCAPE（逃避炸彈）

行為：AI 會快速逃離炸彈的爆炸範圍。

切換條件：

(1) 進入 ESCAPE 狀態：

- 附近有炸彈，且炸彈即將爆炸。
- 當前位置在炸彈的爆炸範圍內。

(2) 離開 ESCAPE 狀態：

- 成功逃出爆炸範圍 → 進入 IDLE
- 發現附近有道具 → 進入 FETCH_ITEMS
- 玩家在可攻擊範圍→進入 ATTACK_PLAYER

2. WAIT_EXPLOSION（等待爆炸）

行為：放炸彈後 AI 停留在安全區等待爆炸。

切換條件：

(1) 進入 WAIT_EXPLOSION 狀態：

- AI 剛剛放置了一顆炸彈
- AI 判斷自己所在的位置是安全的

(2) 離開 WAIT_EXPLOSION 狀態：

- 爆炸後範圍已安全→進入 FETCH_ITEMS
- 玩家在可攻擊範圍→進入 ATTACK_PLAYER
- 炸彈爆炸後沒有其他明確目標→進入 IDLE

3. FETCH_ITEMS（收集道具）

行為：AI 會移動到道具所在的位置並收集。

切換條件：

(1) 進入 FETCH_ITEMS 狀態：

- 附近有道具。
- 當前沒有炸彈威脅。
- 沒有玩家在可攻擊範圍內。

(2) 離開 FETCH_ITEMS 狀態：

- 道具已經被收集
- 玩家在可攻擊範圍→進入 ATTACK_PLAYER
- 附近有炸彈威脅 → 進入 ESCAPE

4. ATTACK_PLAYER（攻擊玩家）

行為：AI 會接近玩家並放置炸彈。

切換條件：

(1) 進入 ATTACK_PLAYER 狀態：

- 玩家在攻擊範圍內。
- AI 有炸彈可以放置。
- 周圍沒有明顯的炸彈威脅。

(2) 離開 ATTACK_PLAYER 狀態：

- 放炸彈等待爆炸→進入 WAIT_EXPLOSION
- 玩家移出可攻擊範圍 → 進入 IDLE 或 FETCH_ITEMS。
- 附近出現炸彈威脅 → 進入 ESCAPE。

5. IDLE（閒置/巡邏）

行為：在地圖上隨機移動，等待新的狀態觸發

切換條件：

(1) 進入 IDLE 狀態：

- 沒有炸彈威脅
- 沒有道具在附近
- 玩家不在可攻擊範圍內

(2) 離開 IDLE 狀態：

- 發現炸彈威脅 → 進入 ESCAPE
- 發現道具 → 進入 FETCH_ITEMS
- 玩家進入可攻擊範圍 → 進入 ATTACK_PLAYER

五、 專題製作心得

當初在選擇專題題目時，我們想要選擇一個難度較高，而且比較好玩的遊戲，便毅然決然選了 Crazy Arcade，但當我們開始做了才發現這難度比我們所想的要高太多。我覺得 Crazy Arcade 最難的點在於 AI 玩家的設計，從初始版本很笨的 AI 玩家到後來改用 Finite State Machine 的做法，大幅增進了 AI 玩家的智能。我們設計的 AI 玩家在分類上屬於 Rule-Based AI，也就是根據事先規劃好的邏輯規則與狀態機切換行為，達到動態應對環境的效果。

儘管它屬於基於狀態機的有限人工智慧，並無法進行深度學習或策略優化，但在遊戲場景中已經可以展現出環境感知、狀態判斷和動態路徑尋徑的核心特性。未來可以朝向增加更多狀態以及往更細緻的狀態切換方向努力，讓 AI 玩家展現更高的智能。

除了在不斷提升我們遊戲的遊玩性和順暢度外，我們也特別注重遊戲的順暢度、美感與使用者互動介面相關的細節，讓遊戲操作起來更直覺、更美觀、更順暢，希望能讓使用者能有很棒的遊戲體驗。

還有一件在我做這份專題讓我印象深刻的事情，那就是 Debug，真正體會到之前學長說的 Coding 只有兩成在打 Code，剩下八成都在 Debug，尤其在程式寫到 2 千行以上的時候，那個 debug 難度非同小可，因此我們才決定將其分類封裝以及繼承，模組化後分區 Debug，其中經歷了用文字無法描述的 Debug 地獄，體驗到 GPT 也幫不上忙的絕望，到此為止解決了表面能看到的所有 bug。

一開始在做這個專題的時候，僅僅是為了做專題而做專題，但是隨著時間推移，完成度越來越高的時候，越做越投入，做到走火入魔的地步，連出去吃飯都在討論怎麼修改專題，真的是廢寢忘食在做這份專題。至此為止，我們盡全力做到了最好，即使還有一些能改進的地方，我們也覺得無愧於這次的專題，希望助教能給予我們一個滿意的成績。

最後希望助教在看完心得和我們 demo 完遊戲後也可以嘗試遊玩我們的專題結果，如果助教也能享受遊玩這遊戲的過程，那我們也會感到非常榮幸。