

Color Block Rush

Press space to start or esc to quit

Press c for credits

計算機程式設計
期末專題書面報告

b08201007 林睿庠 數學一

目次

一、 專題說明

- (一) 遊戲說明/文案
- (二) 版面配置
- (三) 功能表
- (四) 程式架構圖&組員分工

二、 程式碼 Detail

- (一) 類別架構與關聯圖/封裝及 friend 設定
- (二) Constructor/Destructor
- (三) Operator Overloading
- (四) Objective C++技巧

一、 專題說明

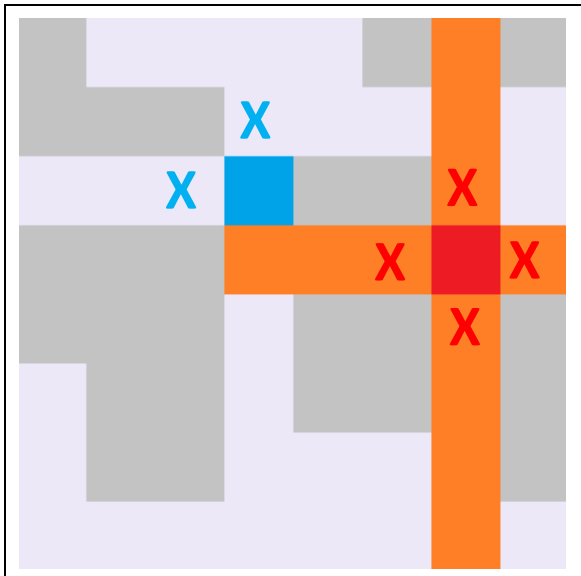
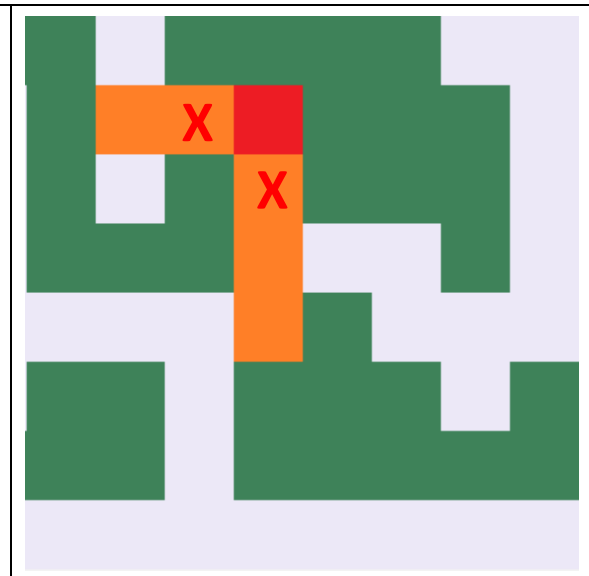
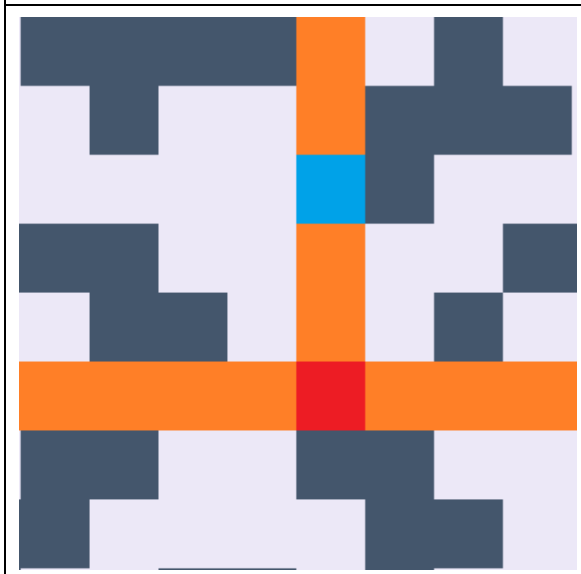
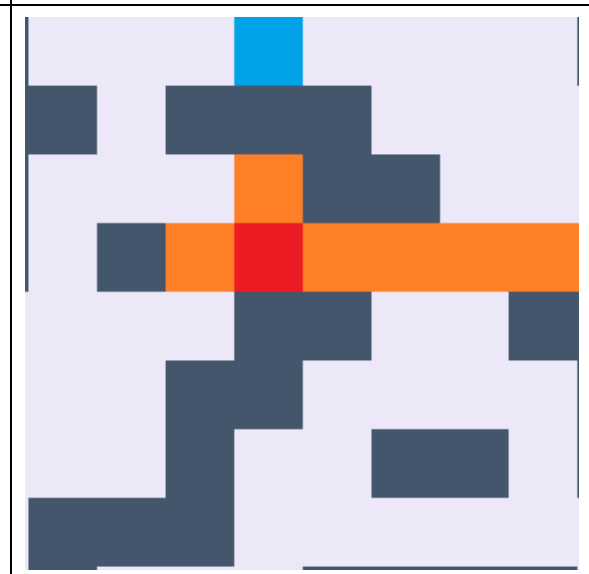
(一) 遊戲說明/文案

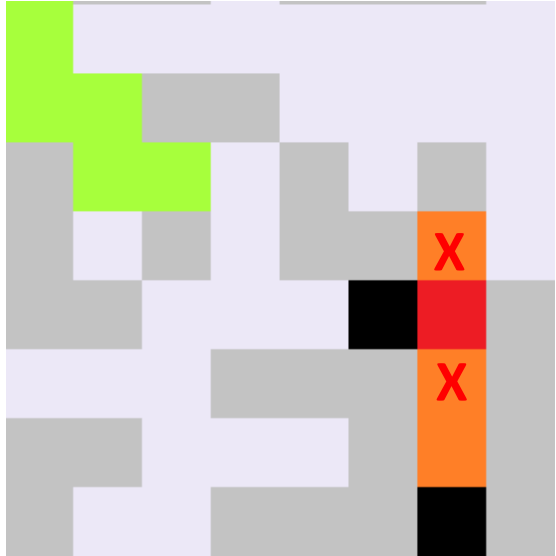
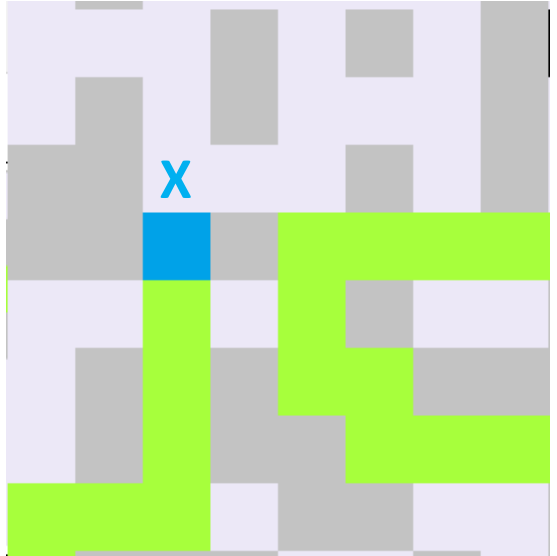
此遊戲為一個雙人遊戲(亦可選擇單邊由電腦控制)，玩家可以任意選擇攻方與守方。攻方及守方大致上可以分別被視為西洋其中的城堡與國王，唯一差別在於國王不能斜走(所以不可能吃掉城堡)，而城堡一次只能走一格(攻擊範圍不變)，雙方每回合都必須移動，並由攻方先走。為了增加遊戲公平與趣味，地圖上(18x18)會設有牆壁以及障礙物(預設六種圖，可以隨機選取)，攻方攻及範圍不可穿牆。攻方目標在於使守方在其攻擊範圍內並於一回合內無法脫離，或是使守方走任何走法都會進入攻擊範圍內，若滿足以上兩條件之一，由攻方獲勝；反之，若在八十回合內攻方未達成獲勝條件，則由守方獲勝。並提供兩種可以在開始畫面中勾選的模式(可複選或不選)

(1) 隨機毒區：隨機生成一個地點(不會包含玩家所在位置)、大小(至少七格)、形狀(必為連通快)隨機的毒區，地圖上一次只會出現三個毒區，任一玩家進入直接結束遊戲並由另一位玩家獲勝

(2) 隨機炸彈：與毒區概念相同，但大小只有一格，場上最多會同時出現五顆炸彈，任一玩家踩到直接結束遊戲並由另一位玩家獲勝

註：若玩家同時開啟炸彈和毒區模式，炸彈必定不在毒區內。

	
<p>攻方為紅色，攻擊範圍為橘線，可移動位置為四周紅叉；守方為藍色，可移動位置為四周藍叉</p>	<p>任意一方碰到牆時，不可穿牆或是移動到牆上，圖中紅叉表示此狀況下紅色可以移動到的位置</p>
	
<p>圖中，守方在攻方攻擊範圍內</p>	<p>守方位置落在攻方攻擊範圍的延伸線上，但因為攻擊範圍不能穿牆，所以守方在攻擊範圍之外</p>

	
<p>黑色為炸彈，可以視為臨時的牆，不可以移動到其上，可以移動的方式以紅叉表示</p>	<p>綠色為毒區，可以視為臨時的牆，不可以移動到其上，可以移動的方式以藍叉表示</p>

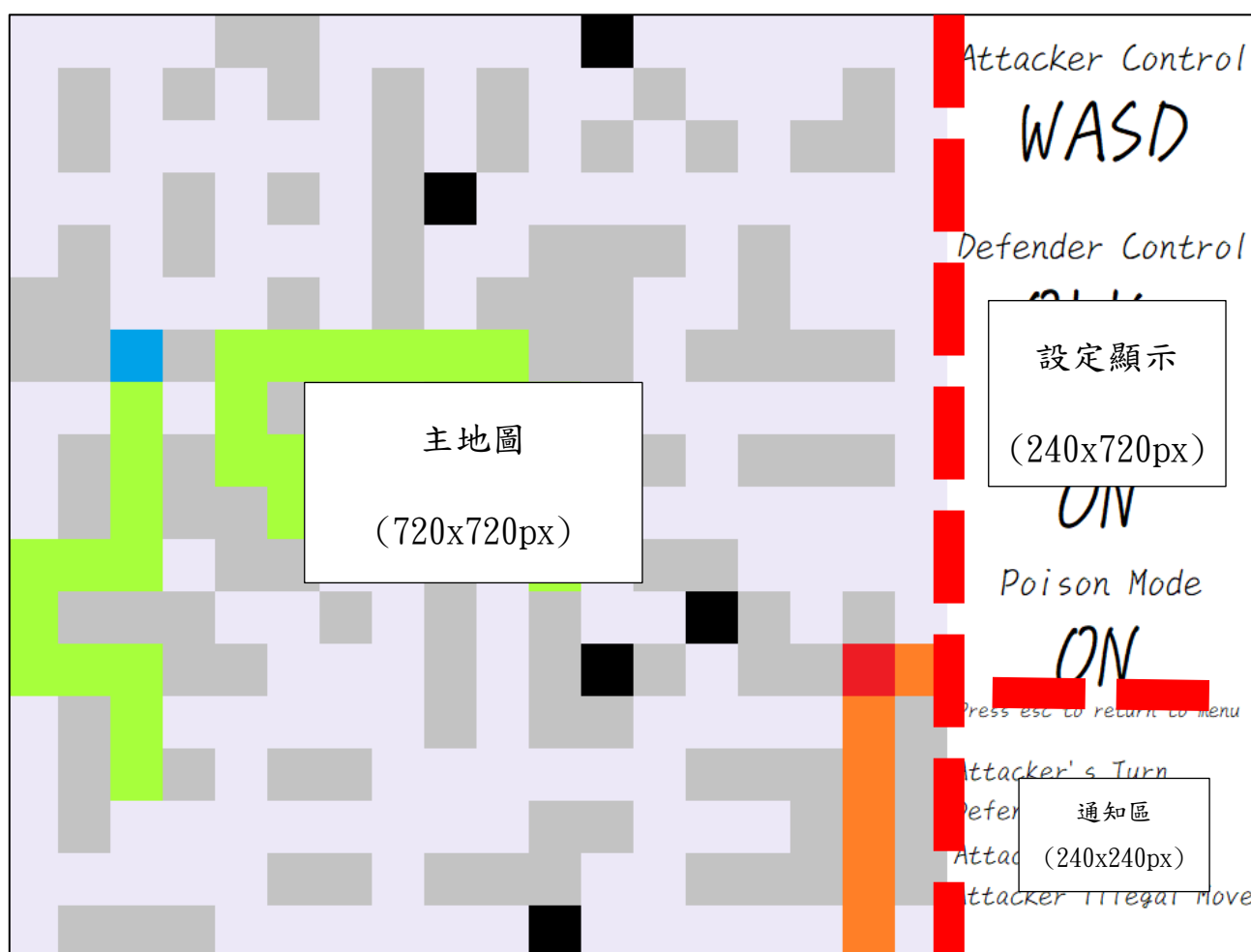
(二)版面配置

1. 開始畫面

主選單、玩家控制教學(詳見功能表)

2. 遊戲畫面

遊戲版面(960x720px):主地圖(720x720px)、通知區(240x240px)、設定顯示(240x720px)



(三) 功能表

開始畫面選單		
地圖選擇	七選一	預設 x6+隨機地圖
延伸模式	勾選 x2(可複選)	毒區、炸彈模式
攻守方控制	攻方是否開啟 AI， 守方是否開啟 AI，	玩家 or 電腦操控

玩家鍵盤配置			
w	攻方向上一格	o	守方向上一格
a	攻方向左一格	k	守方向左一格
s	攻方向下一格	l	守方向下一格
d	攻方向右一格	;	守方向右一格

遊戲版面(960x720px)	
主地圖(720x720px)	主要遊戲畫面，有美工
通知區(240x240px)	顯示通知、該誰
設定顯示(240x720px)	顯示雙方控制、特殊模式狀況

(四)程式架構圖&組員分工

Project Color Block Rush

AI

Class
Attacker

Class
Defender

Class
LTexture

Class
Map

Class
Winner

林睿庠

林睿庠

林睿庠

林睿庠
樊樺

林睿庠
樊樺

李沐恩

二、程式碼 Detail

(一)類別架構與關聯圖/封裝及 friend 設定

Note: 主要實作物件導向的 class 為 map 與 Itexture, ai、attacker、defender 為以 function 為主體的 class, 應題目要求仍然一併列出。

- Class Map (map.h, map.cpp)

```
public int array[18][18]= {{0}}
```

- Array that records objects on the 18x18 tiled map.

```
public void initialize()
```

- Function triggered by constructor that initializes values for class map.

```
public void set_bomb()
```

- sets 5 bombs at random locations on the map

```
public void set_poison()
```

- sets 3 poison areas ranging from 7~12 blocks on the map

```
public bool islegal(int, int)
```

- Checks if the coordinate (int, int) input is legal for attacker to move to.

```
public bool istrapped(int, int)
```

- Checks if the coordinate (int, int) input is legal for defender to move to.

```
public void operator+()
```

- Save player position data to the array on map.

```
public bool check_win()
```

- Check if any player wins.

- *Class LTexture (LTexture.h, LTexture.cpp)*

```
public LTexture()
```

- Constructor that initializes values for mWidth, mHeight, and mTexture.

```
public bool loadFromFile( std::string path )
```

- Loads textures from file.

```
public void render( int x, int y, int width, int height)
```

- Function for rendering.

```
public friend void getwidth(LTexture &l)
```

- Function that sets private member mWidth.

```
public friend void getheight(LTexture &l)
```

- Function that sets private member mHeight.

```
public void operator!()
```

- Outputs failure of loading texture

```
public void operator-()
```

- Destructor, destroys texture.

```
private SDL_Texture* mTexture
```

- Private SDL_Texture pointer.

```
private int mWidth
```

- Private int that stores the width of the texture.

```
private int mHeight
```

- Private int that stores the height of the texture.

- *AI (ai.h, ai.cpp)*

```
int get_distance(int, int, int, int)
```

- A function that calculates the distance for the fastest route between the attacker and defender. Uses a modified version of the BFS Algorithm (first in first out) with time complexity $O(n \log n)$.

- *Class Attacker (attacker.h, attacker.cpp)*

```
public int x, y
```

- Stores the current coordinates of the attacker.

```
public void initialize(void)
```

- Function called by constructor for class attacker, initializes the x, y coordinates of the attacker

```
public void up()
```

- Moves attacker one step upward.

```
public void down()
```

- Moves attacker one step downward.

```
public void left()
```

- Moves attacker one step leftward.

```
public void right()
```

- Moves attacker one step rightward.

```
public void ai_move()
```

- Auto moves attacker one step in a direction.

- *Class Defender (defender.h, defender.cpp)*

```
public int x, y
```

- Stores the current coordinates of the defender.

```
public void initialize(void)
```

- Function called by constructor for class defender, initializes the x, y coordinates of the attacker

```
public void up()
```

- Moves defender one step upward.

```
public void down()
```

- Moves defender one step downward.

```
public void left()
```

- Moves defender one step leftward.

```
public void right()
```

- Moves defender one step rightward.

```
public void ai_move()
```

- Auto moves defender one step in a direction.

- 我接下來會主要著重在 *map*、*LTexture* 兩個 *class* 的運作原理，因為此二 *class* 皆有效實作了較完整物件導向的性質。
- 絕大部分封裝設定為 *public*，原因為其需開放讓外界存取與修改，因為此專案大多數 *class* 間需要有極大的資料共享，所以大都採取此

方法，僅 `LTexture.cpp` 中的 `texture` 大小符合 `private` 的需求，由於本專題採用 `tilled map` 的方式去 `render` 介面，因此 `texture` 的大小、從檔案讀取的圖需要嚴格控管寫入，以免不小心更動。

● Private 成員

<code>LTexture.cpp</code>	
<code>SDL_Texture* mTexture</code>	存取從檔案讀取的圖形資料
從檔案讀取的 <code>texture</code> 圖形資料對於 <code>tilled map</code> 來說極為重要，有必要限制外部 <code>function</code> 與 <code>class</code> 的存取、更動，以免造成顯示上的問題	
<code>int mWidth</code>	<code>texture</code> 的寬
確保 <code>texture</code> 的寬不備輕易更動，以免造成圖形變形	
<code>int mHeight</code>	<code>texture</code> 的高
確保 <code>texture</code> 的高不備輕易更動，以免造成圖形變形	

● Friend 成員

<code>LTexture.cpp</code>	
<code>void getwidth(LTexture &l)</code>	讀取 <code>texture</code> 的寬
return 身為 <code>private</code> 成員的 <code>mWidth</code>	
<code>void getheight(LTexture &l)</code>	讀取 <code>texture</code> 的高
return 身為 <code>private</code> 成員的 <code>mHeight</code>	

(二) Constructor/Destructor

map.h、map.cpp	in class map, constructor
以 <code>map(){initialize();}</code> 這個 constructor 觸發 initialize 函數	
<pre>void map::initialize(void) { std::string str_map_line; std::string map_file_name; map_file_name="./maps/default_"; map_file_name.push_back(map_selection+'0'); map_file_name+="txt"; std::ifstream fin; fin.open(map_file_name.c_str()); int j=0; while(fin>>str_map_line) { for(int i=0; i<18; i++) { m.array[i][j]=str_map_line[i]-'0'; } j++; } fin.close(); }</pre>	
LTexture.h、LTexture.cpp	in class LTexture, constructor
<code>LTexture();</code>	
<pre>LTexture::LTexture() { mTexture = NULL; mWidth = 0; mHeight = 0; }</pre>	

attacker.h、attacker.cpp	in class attacker, constructor
以 <code>attacker(){initialize();};</code> 這個 constructor 觸發 initialize	
<pre>void attacker::initialize(void) { if(m.islegal(17,17)) { x=17; y=17; legal_move=1; } }</pre>	
defender.h、defender.cpp	in class defender, constructor
以 <code>defender(){initialize();};</code> 這個 constructor 觸發 initialize	
<pre>void defender::initialize(void) { if(m.istrapped(0,0)) { x=0; y=0; legal_move=1; } }</pre>	

以上皆為 Constructor，之所以將 initialize 函式獨立寫在 Constructor 外面是因為初始化不一定只有在最剛開始的時候，為了節省計算、記憶體資源，遊戲重新開始時，仍舊重新使用既有的變數，所以會校再次初始化。

(三) Operator Overloading

map.h \ map.cpp	in class map
<pre>void operator+();</pre>	
<pre>void map::operator+() { for(int i=0; i<18; i++) { for(int j=0; j<18; j++) if(array[i][j]==1 array[i][j]==2 array[i][j]==8) { array[i][j] = 0; } } array[_attacker.x][_attacker.y] = 1; array[_defender.x][_defender.y] = 2; int x_up=_attacker.x-1; int y_up=_attacker.y; while(x_up>=0&&(array[x_up][y_up]==0 array[x_up][y_up]==2)) { if(array[x_up][y_up]==0) { array[x_up][y_up]=8; x_up--; } else if(array[x_up][y_up]==2) { x_up--; } else { break; } } int x_down=_attacker.x+1; int y_down=_attacker.y; while(x_down<=17&&(array[x_down][y_down]==0 array[x_down][y_down]==2)) { if(array[x_down][y_down]==0) { array[x_down][y_down]=8; x_down++; } else if(array[x_down][y_down]==2) { x_down++; } else { break; } } int x_left=_attacker.x; int y_left=_attacker.y-1; while(y_left>=0&&(array[x_left][y_left]==0 array[x_left][y_left]==2)) { if(array[x_left][y_left]==0) { array[x_left][y_left]=8; y_left--; } else if(array[x_left][y_left]==2) { y_left--; } else { break; } } }</pre>	

<pre> int x_right=_attacker.x; int y_right=_attacker.y+1; while(y_right<=17&&(array[x_right][y_right]==0 array[x_right][y_right]==2)) { if(array[x_right][y_right]==0) { array[x_right][y_right]=8; y_right++; } else if(array[x_right][y_right]==2) { y_right++; } else { break; } } } </pre>	
LTexture.h、LTexture.cpp	in class LTexture
<pre>void operator! ();</pre>	
<pre> void LTexture::operator! () { printf("Failed to load arrow texture!\n"); } </pre>	
LTexture.h、LTexture.cpp	in class LTexture
<pre>void operator- ();</pre>	
<pre> void LTexture::operator- () { if(mTexture != NULL) { SDL_DestroyTexture(mTexture); mTexture = NULL; mWidth = 0; mHeight = 0; } } </pre>	

以上三個 operation overloading 是勉強擠出來的，因為此遊戲實則沒有太大必要使用之。

(四)Objective C++ 技巧

物件導向技巧	是否達成
多型	無
繼承	無
封裝	有，前面敘述之建構子、私有成員、朋友函式、運算子多載皆屬此類，應該遠遠超過三項了

對於期末專題的一些想法

與他人合作，真的是練習 *coding* 的最佳方式，組員間互相看 *code* 以及對於格式、命名逐漸產生共識，真的是一個非常有趣的過程。老師選擇在學期末讓我們做專題真的是非常棒的想法，也希望之後學弟妹也可以享受這個過程。

然而，我希望可以提一個小小的建議，最後的書面報告是否可以放寬題目的限制?現有的架構下，會造成很多人為了符合題目需求而對 *code* 做出一些沒必要的調整。像是 *operation overloading*，每個人最少要三個，一組三個人就要九個，真的有一點多。

建議允許同學從多一些題目中選擇幾個問題作回答，亦可加入像是是否使用良好、有效降低運算的演算法(嘿嘿，我們這組的 *AI* 很符合， $O(n \log n)$)...等其他問題。

這學期的課程真的非常充實，謝謝老師!!