目錄

一、 簡介 ……………………………………………1

1.動機 ……………………………………………1

二、 遊戲介紹 ……………………………………………2

1. 遊戲說明 ……………………………………………2

2. 遊戲圖形 ……………………………………………5

3. 遊戲音效 ……………………………………………6

三、 程式設計 ……………………………………………7

1.程式架構 ……………………………………………7

2.程式類別 ……………………………………………7

3.程式技術 ……………………………………………8

四、 結語 ……………………………………………9

1. 問題及解決方法 ……………………………………………9

2. 時間表 ……………………………………………11

3. 貢獻比例 ……………………………………………12

4. 自我檢核表 ……………………………………………12

5. 收穫 ……………………………………………12

6. 心得感想 ……………………………………………13

附錄 ………………………………………………14
Command ………………………………………………14
Flatland ………………………………………………30
Fog ………………………………………………31
GeographicSystem ………………………………………………34
HydrometeorologicSystem ………………………………………………42
Lightning ………………………………………………48
MainForm ………………………………………………50
Mountain ………………………………………………67
Reef ………………………………………………68
Shell ………………………………………………70
Team ………………………………………………76
Torpedo ………………………………………………78
Typhoon ………………………………………………86
VesselAH ………………………………………………88
VesselBB ………………………………………………93
VesselCG ………………………………………………95
VesselCV ………………………………………………99
VesselDD ………………………………………………101
VesselLV ………………………………………………105
VesselObject ………………………………………………116

一、簡介

1. 動機:

因為換題目的時候知道時間已經沒很多了，因此覺得應該找一個遊戲是不用再花很多時間去做素材方面，把時間著重在寫程式上，這時候就想到之前曾經玩過一些遊戲，像是:世界帝國(一個地圖上分成兩隊，每隊都可以建立城堡，跟攻打敵對的建築物)、麥塊(畫面的顯示都是用不同顏色的方格，同樣也是可以建立跟攻打敵人的資產，只是敵人是由電腦所操作的)，但是他們相同的地方是都可以透過輸入指令來讓遊戲開外掛，雖然這樣在玩遊戲的時候會覺得還要打字輸入，但或許因為我每次玩遊戲都不太容易上手跟都一直輸，所以基乎每個我會玩很久的遊戲，我都知道密技指令，而且對我來說不用一直按按鈕，經過很多步驟，有時候還會不知道要找的東西，被放在哪裡，所以只需要看著小抄輸入就能達到相同效果，對我來說真的感覺比較方便。

二、遊戲介紹

1. 遊戲說明:

遊玩方式:首先可以看到右上角每個功能鍵的第一行左半邊為此按鈕的功能，右邊為須在地圖上點擊所代表的變數，第二行則是需要自行輸入的參數，像是新的船艦名稱跟船艦種類，若有第三行則代表只有這些船艦種類可以執行此一按鈕。因此第一步驟是先去看想要執行的功能鍵上面寫需要輸入什麼參數跟需要在地圖上點擊什麼資訊，接著把這些資訊打在相對應的組別，並點擊座標或是船艦，最後按下功能鍵即可執行。而主要會導致指令失敗的原因有: 當建立船艦時，輸入的船艦名稱已存在於同隊、當防禦時，要抵擋的子彈不存在、當移動船艦時，輸入的速度大於船艦本身可移動的最大速度、當修復補血時，同組在地圖上的船艦不存在、當重新命名船艦時，同組在地圖上的船艦已存在。

遊戲規則:傳統模式是一個雙人一起玩的遊戲，左邊的為 A 隊，右邊的為 B 隊，地圖大小為 20*20 平方單位，(0,0)在左上，並往右下遞增，並且可以透過座標提示欄來方便玩家瞭解當下的座標位置。各組都可以建立船艦跟發射三種子彈:砲彈、魚雷、雷射與防護，而砲彈所造成的傷害範圍為半徑 1.5 單位長度，到達指

定位置後爆炸，但若有船艦在該砲彈行進的路徑上使用防守，則可消滅該砲彈。若想要輸入指令，則需按下 stop 按鈕。相反地，當開始計時的時候，不可輸入戰鬥指令，因為此時會執行戰鬥指令，而戰鬥指令存在順序性，會先執行 A 艦隊再執行 B 艦隊，而 A 艦隊在地圖上的船艦圖形跟輸入指令都為紅色，B 艦隊則為藍色，以方便看出區別。

以下用表格方式列舉全部船艦的屬性(本表使用的時間為遊戲時間):

船艦種類	CV	BB	CG	DD	LV	AH
圖形表示	■	●	◆	▲	⬟	✚
船艦血量	5	4	3	2	1	5
移動最高航速 (單位長度/分鐘)	1	1	2	3	4	1
攻擊最遠距離	25	20	15	10	全部	
FIRE 砲彈傷害值	3	3	2	1		
FIRE 砲彈冷卻時間	15	30	30	60		
LAUNCH 魚雷傷害值			3	3		
LAUNCH 魚雷冷卻時間			60	30		
LASER 雷射傷害值					6	
LASER 雷射冷卻時間					90	
防禦最遠距離	5	10	15	20		
防禦冷卻時間	15	30	30	60		
修復一次的補血量						1
修復最遠距離						3
修復冷卻時間						50

彈幕模式是敵人會自動攻擊你，並且每一關會發射出不同變化的子彈路徑，而玩家只要負責閃躲攻擊，當全部關卡結束後，如果自己的船艦還有剩下血，那代表玩家贏了，相反地，如果玩家的船艦沒血了，則輸了。遊戲結束後會跳出視窗顯示最後得分，初始值為 100 分，並搭配相對應的輸贏背景音樂。

特殊功能:可以按下+1s 按鈕，讓時間快轉 1 秒，看下 1 秒子彈的發射路徑，從中躲過攻擊，又或者是瞭解船艦移動的方向防止撞到障礙地形。

以下用表格方式列舉可以在地圖上增加三種障礙地形:

障礙地形名稱	山	平原	暗礁
呈現圖形	🟦	🟩	◼
船艦移動	X	X	X，傷害值 1
FIRE 的砲彈	X	O	O
LAUNCH 的魚雷	X	X	X
LASER 的雷射	O	O	O

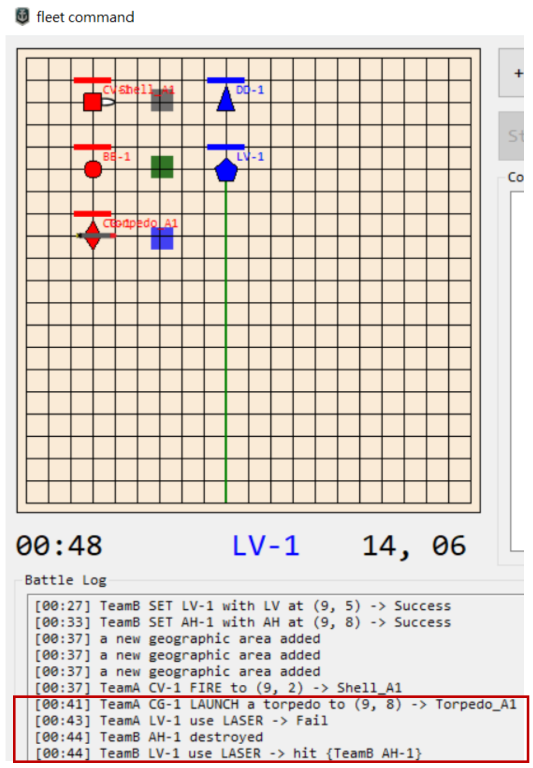
舉例來說:當船艦在移動的過程碰到山，則船艦會停下，並在記錄欄顯示碰到山，不能移動。若碰到暗礁，則也會停下，但是船艦會造成額外的傷害。當 FIRE 的子彈在發射過程中碰到平原，此時可以穿過障礙地形，不會被阻擋下來。

建立須符合以下規則:長寬任意一邊皆不可為 0、任兩地形皆不可重疊

氣象	霧	閃電	颱風
呈現圖形	⬭	⚡	🌀
呈現時間	100	5	移動路徑(直到離開地圖)
傷害時間	▨	一次	持續造成(經過船艦的時間)
傷害值	▨	6	0.2

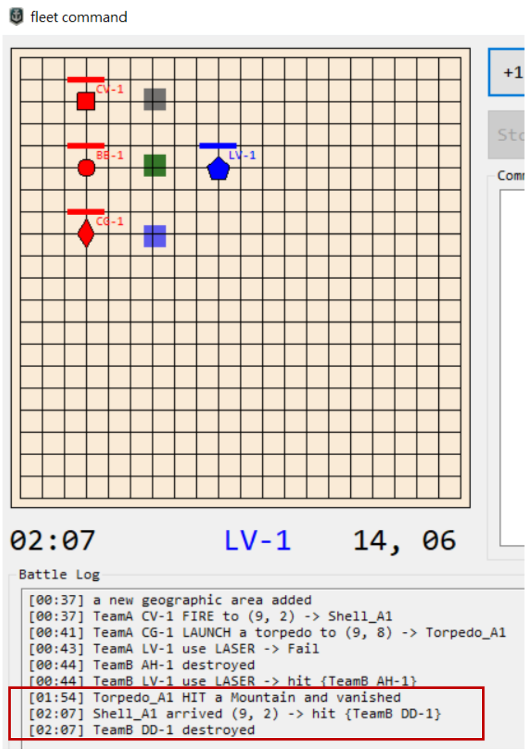
2. 遊戲圖形：

3 種子彈的圖形呈現



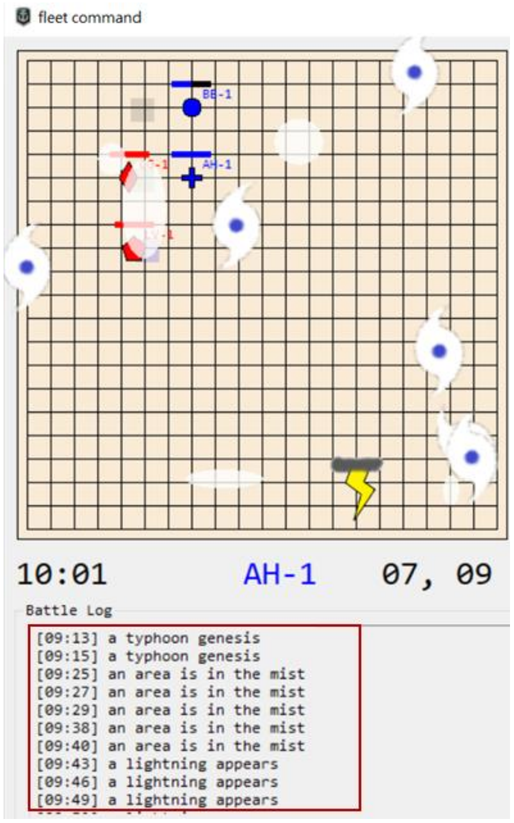
```
[00:27] TeamB SET LV-1 with LV at (9, 5) -> Success
[00:33] TeamB SET AH-1 with AH at (9, 8) -> Success
[00:37] a new geographic area added
[00:37] a new geographic area added
[00:37] a new geographic area added
[00:37] TeamA CV-1 FIRE to (9, 2) -> Shell_A1
[00:41] TeamA CG-1 LAUNCH a torpedo to (9, 8) -> Torpedo_A1
[00:43] TeamA LV-1 use LASER -> Fail
[00:44] TeamB AH-1 destroyed
[00:44] TeamB LV-1 use LASER -> hit {TeamB AH-1}
```

3 種子彈經過障礙地形的實測結果



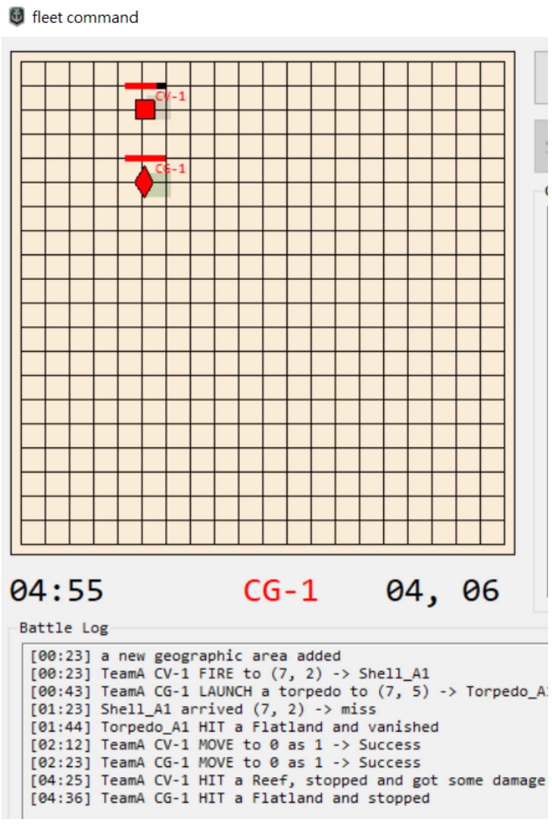
```
[00:37] a new geographic area added
[00:37] TeamA CV-1 FIRE to (9, 2) -> Shell_A1
[00:41] TeamA CG-1 LAUNCH a torpedo to (9, 8) -> Torpedo_A1
[00:43] TeamA LV-1 use LASER -> Fail
[00:44] TeamB AH-1 destroyed
[00:44] TeamB LV-1 use LASER -> hit {TeamB AH-1}
[01:54] Torpedo_A1 HIT a Mountain and vanished
[02:07] Shell_A1 arrived (9, 2) -> hit {TeamB DD-1}
[02:07] TeamB DD-1 destroyed
```

新增 3 種氣象系統(霧、閃電、颱風)



```
[09:13] a typhoon genesis
[09:15] a typhoon genesis
[09:25] an area is in the mist
[09:27] an area is in the mist
[09:29] an area is in the mist
[09:38] an area is in the mist
[09:40] an area is in the mist
[09:43] a lightning appears
[09:46] a lightning appears
[09:49] a lightning appears
```

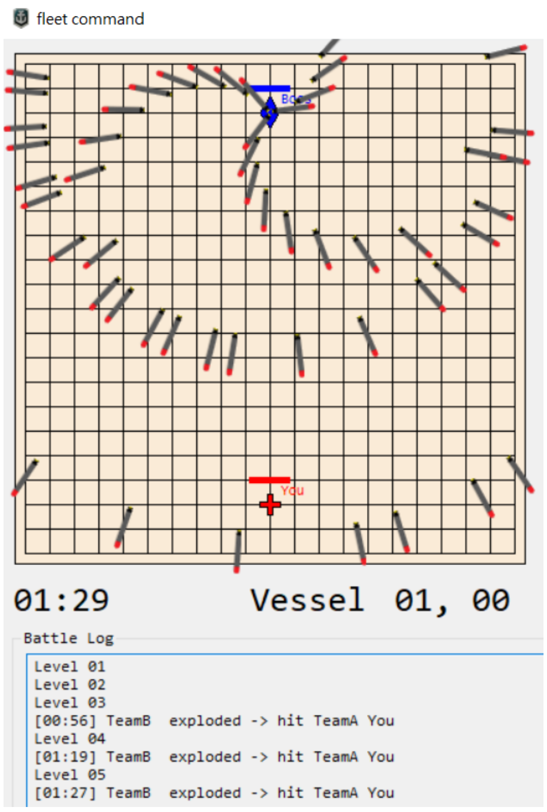
船艦碰到障礙地形產生不同結果實測



```
[00:23] a new geographic area added
[00:23] TeamA CV-1 FIRE to (7, 2) -> Shell_A1
[00:43] TeamA CG-1 LAUNCH a torpedo to (7, 5) -> Torpedo_A
[01:23] Shell_A1 arrived (7, 2) -> miss
[01:44] Torpedo_A1 HIT a Flatland and vanished
[02:12] TeamA CV-1 MOVE to 0 as 1 -> Success
[02:23] TeamA CG-1 MOVE to 0 as 1 -> Success
[04:25] TeamA CV-1 HIT a Reef, stopped and got some damage
[04:36] TeamA CG-1 HIT a Flatland and stopped
```

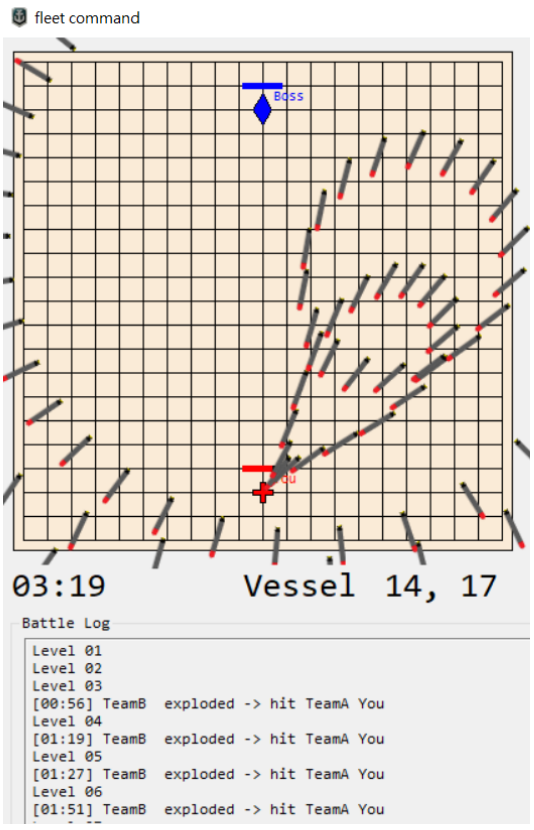
Battle Log 上面的那行最左邊為遊戲裡的時間，中間為現在所點擊的船艦名稱，若此船艦為 A 隊則顯示紅字，B 隊則藍字，最右邊為所點擊的座標位置或是障礙地形的種類。Battle Log 會顯示在遊戲的哪個時間，使用者的操作紀錄。

旋轉並同時顯示多個發射點



```
01:29        Vessel  01, 00
Battle Log
Level 01
Level 02
Level 03
[00:56] TeamB  exploded -> hit TeamA You
Level 04
[01:19] TeamB  exploded -> hit TeamA You
Level 05
[01:27] TeamB  exploded -> hit TeamA You
```

子彈會跟著現在的船艦位置瞄準發射



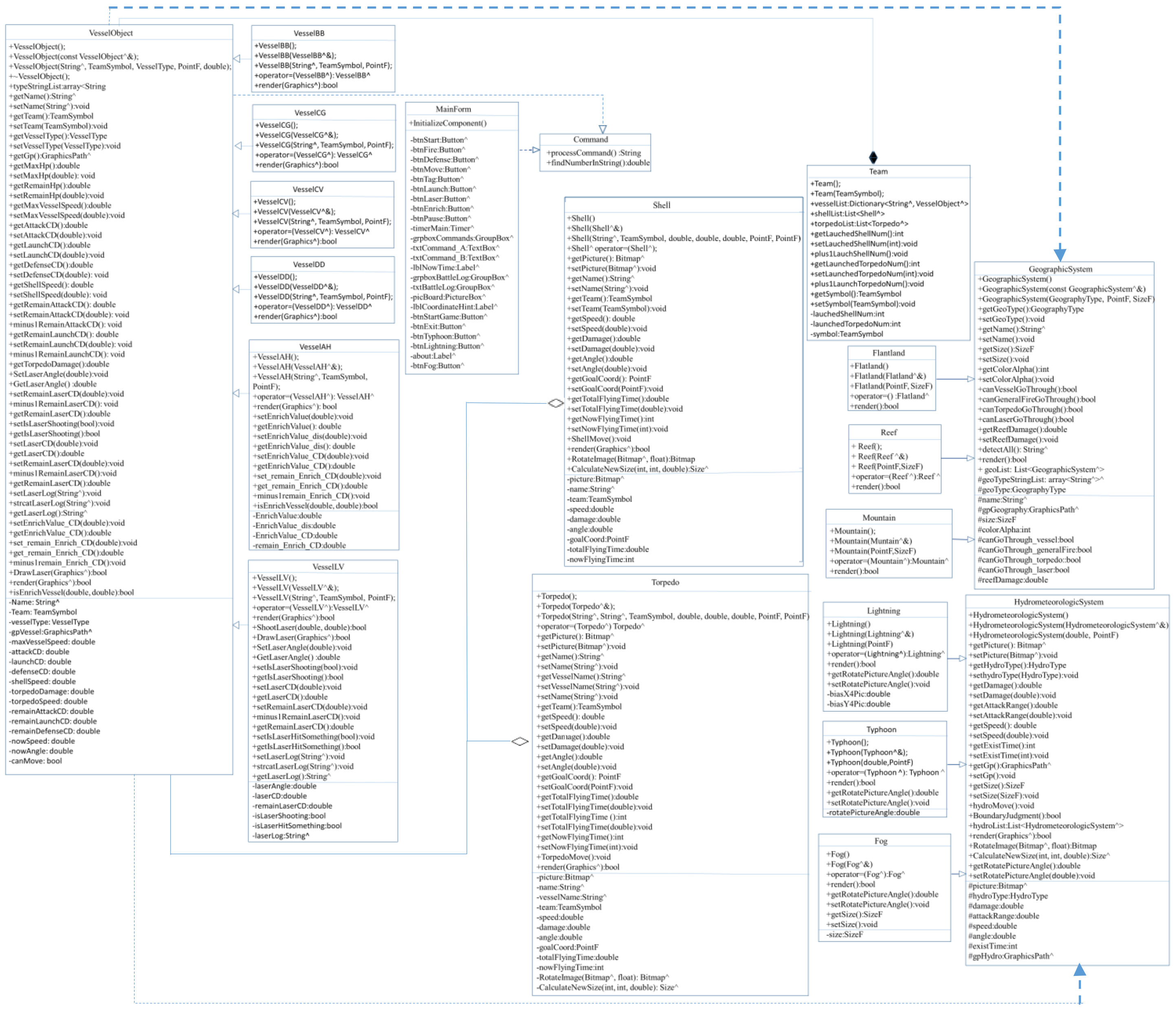
```
03:19        Vessel  14, 17
Battle Log
Level 01
Level 02
Level 03
[00:56] TeamB  exploded -> hit TeamA You
Level 04
[01:19] TeamB  exploded -> hit TeamA You
Level 05
[01:27] TeamB  exploded -> hit TeamA You
Level 06
[01:51] TeamB  exploded -> hit TeamA You
```

3. 遊戲音效：

音效名稱	聲音	觸發按鈕
ding.wav	叮	+1s(下一秒)
fire.wav	連發子彈	FIRE
launch.wav	炮炸	LAUNCH
laser.wav	電擊	LASER
win.wav	驚叫歡呼	彈幕模式遊戲結束時
lose.wav	敲鑼、敲鈸	彈幕模式遊戲結束時

三、程式設計

1. 程式架構:



2. 程式類別:

類別名稱	.h 檔行數	.cpp 檔行數	說明
Command	47	677	處理輸入指令跟地圖被點擊位置，並判斷字串是否有效印出成功或失敗
Flatland	13	55	暗礁的圖形、基本屬性、碰撞設定

Fog	31	76	霧的隨機大小、圖形呈現跟基本屬性
GeographicSystem	112	259	障礙地形碰撞傷害、設定種類跟基本屬性
HydrometeorologicSystem	84	228	氣象的碰撞傷害、種類、移動路徑、持續時間
Lightning	21	56	圖片呈現設定、基本屬性設定
MainForm	356	12	起始畫面的呈現跟點擊觸發功能
Mountain	12	58	高山的圖形、基本屬性、碰撞設定
Reef	13	63	暗礁的圖形、基本屬性、碰撞設定
Shell	76	203	砲彈的發射路徑、圖形、基本屬性
Team	41	42	設定船艦子彈的所屬戰隊、基本特性
Torpedo	84	238	魚雷的發射路徑、圖形、基本屬性
Typhoon	21	71	隨著移動路徑旋轉颱風圖片
VesselAH	58	179	船艦 AH 圖形、船艦屬性、功能值相關設定
VesselBB	24	86	船艦 BB 圖形、船艦屬性、功能值相關設定
VesselCG	30	106	船艦 CG 圖形、船艦屬性、功能值相關設定
VesselCV	25	86	船艦 CV 圖形、船艦屬性、功能值相關設定
VesselDD	27	117	船艦 DD 圖形、船艦屬性、功能值相關設定
VesselLV	88	389	船艦 LV 圖形、船艦屬性、功能值相關設定
VesselObject	88	401	各種船艦的所擁有的功能、共同屬性設定
總行數	1251	3402	

3. 程式技術:

讓船艦同時擁有了繼承跟多型的特性:為了讓程式更加簡化，因此在繼承方面，將每個船艦共同擁有的特性或物件都先設置在雛型船艦內,之後每個新增的船艦都會繼承這個雛形，此外，多型方面則是運用更廣泛，在船艦、子彈、氣象系統都有運用到，像是某些船可以發射特定的子彈類型，某些船則無法、可以判斷每

次會產生隨機大小的霧裡有哪些船艦，並且處於霧裡的船艦就不能移動，其他的船艦則不會有這個限制。

另外程式內也有使用到陣列跟 Linked list，來存取船艦準備要發射的各種子彈類性，因為每次存取的子彈是可以動態增加的，所以搭配 Linked list 可以讓子彈的儲存更有效率。

對打模式裡的颱風氣象系統，使用了旋轉圖片並搭配曲線路徑來離開地圖，旋轉的方式是用圖片的中心點配合三角函數，來自動繪成下一張圖片，並用數學方程式將原本圖片的座標加上固定位移量來達到圖片呈現曲線前進的效果。

彈幕模式則用了不同的旋轉的方式來灑子彈，是配合自己寫的三角函數方程式計算每一次的偏移量，達到下一次的旋轉的起始點會不同

四、結語

1. 問題及解決方法:

在剛開始拉遊戲的初始介面的時候，原本覺得會很簡單，但是寫的過程發現，不管我怎麼設定按鈕的位置，他都顯示不出指定的文字與想要放在的地方，所以一開始我嘗試把我的按鈕調大，按鈕裡的文字也調小，但它還是沒有出現任何字，只有左上角出現一個空白的小框框，我找了好久才發現原來我忘記在一開始初始化的地方新增按鈕出來。這個雖然聽起來像是剛接觸程式才會遇到的問題，但是因為它一開始還是有顯示出按鈕圖形，只是沒有出現我寫入的文字跟不在我指定的地方，所以我那時候就一直沒猜想到是這一方面的問題，因此我真的看了很久才總算解決了。

為了讓船艦的雷射攻擊方式可以更多種變化，所以我設定可以指定發射的角度，因此一開始我在判斷同個角度的所有船艦時，本來想透過發射出來的雷射圖形當跟船艦有碰撞時來判斷為同個角度，但是因為船艦跟障礙地形本身都有一定的大小，或者是當船艦中心移動後沒有停在每個座標的中心位置時，也會產生誤差，因此最後我想到可以利用每艘已建立船艦的座標位置跟發射雷射的船艦用計算

斜率的方式來得到更準確的角度，當相同時船艦才會被消滅。

因為一般大家看到氣象局展示颱風的圖片，他都會呈 360 度旋轉並往前移動，因此一開始我就想那這樣不是要 360 張之後才能載入跟取用，而為了更加方便，所以在上一個神槍手遊戲，我是先用另一個比較擅長的語言把每個角度的圖片旋轉後，再存到資料夾裡之後調用，但是經過幾個禮拜的學習熟悉與跟同學討論後，我又再次挑戰了另一種方法來達到。我先記錄當下的圖片角度、想要旋轉的角度，還有圖片的旋轉中心點，透過這個中心點再加上旋轉角度，來繪畫出一個新的旋轉圖片，最後再用一個方程式讀入來記錄颱風圖片移動的軌跡。

我在寫遊戲裡的第二種彈幕模式，因為想要模仿真實導彈的功能，所以設計船艦要能根據滑鼠的移動方向來做相同方向移動，但是我後來遇到船艦可以跟著滑鼠左右移動了，可是炮彈卻不會自動依照船艦的最新位置去瞄準，因此我最後將砲彈的發射目的地設定到船艦的座標位置，這樣一來在地圖上就能看到砲彈漸漸更改發射方向的變化，還能更有效率的讓發射的發彈維持本來的雨滴發射形狀，因為雨滴的設計是讓本來擴散的砲彈像同一個中心點往前動，所以也寫了很久才讓他最後正常的運作。

在更改我的第一種雙人對打模式的時候，本來想說既然我已經可以判斷到滑鼠在圖片的現在位置，那如果改成點擊指定的座標也能完成每個功能，應該也不會太難，沒想到當我新增 mouseclick 後在做測試的時候，不管我怎麼點擊圖片，座標顯示的地方卻還是顯示錯誤的值，因為我為了方便測試，所以我設定當座標被點擊時，顯示會是一個固定的數值，並且該數值為超出棋盤大小的座標，所以不可能是剛好我按到相同座標，但是怎麼看我還是找不出哪裡寫錯了，因此最後我就想說那換 mousehover 試試，使用 mousehover 主要是想測試滑鼠的鼠標在這個圖片上是不是有被偵測到正確的狀態。而為了跟 mouseclick 的變化作區別，這次我將顯示的座標文字顏色更改，然後竟然發現有幾次的數值是對的了，但大部分的時候卻還是錯的，可是因為我還是不知道問題出在哪，所以只好上網查一下資料，

又想說那試試 mouseup，因為之前的實驗課我曾經聽過有人說當我們滑鼠點擊時是被分成很多個小動作去看，所以我就想既然都一直找不出問題的原因，那就把每一種可能的方法都嘗試看看好了，結果總算這次成功了，但我認為這次也不是因為我程式剛好寫對，因為在起初也還是失敗，只是在這次的測試過程中我發現到它其實是有變化的，只是數值跟顏色閃的太快有時候甚至是看不出來他閃了一下，然後緊接著又因為滑鼠的位置變動，所以數值就被覆改過去了，導致測試的時候一直測不出來。

2. 時間表:

週次	葉子瑄(小時)	說明
1	4	練習使用 Game framework、Git、尋找遊戲題目
2	9	框架重構、建立地圖父類別、製作相關素材
3	19	建立主角、主角拋物線軌跡設計
4	12	主角可以射擊子彈、起始畫面製作
5	12	建立敵人、碰撞判定、輸贏檢測
6	25	加入所有音效、創建初步商城系統、建立會飛的敵人
7	15	飛行敵人可以移動、射擊油桶可以爆炸、建立第 2 張地圖
8	14	飛行敵人可以射擊、射中敵人後可以往後彈跳
9	40	製作選單畫面、建立船艦、攻擊、防禦與移動功能
10	40	建立修復船艦、更換船艦名稱的功能、建立 3 種子彈類型、增加小抄、更換不同船艦的圖片
11	14	建立氣象系統、增加障礙地形系統
12	14	增加颱風系統、完整障礙地形系統的其他功能
13	15	增加冷卻系統、建立第二種關卡模式
14	14	完成第二種模式、完整全部氣象的系統
15	13	更改操作介面跟處理第二關遊戲細節

16	5	新增音效、加入介紹畫面、把細節全部修改完
17	1	展示作品
合計	266	

3. 貢獻比例:

葉子瑄：100%

4. 自我檢核表:

	項目	完成否	無法完成的原因
1	解決 Memory leak	☑已完成 □未完成	
2	自定遊戲 Icon	☑已完成 □未完成	
3	全螢幕啟動	☑已完成 □未完成	
4	有 About 畫面	☑已完成 □未完成	
5	初始畫面說明按鍵及滑鼠之用法與密技	☑已完成 □未完成	
6	上傳 setup/apk/source 檔	☑已完成 □未完成	
7	setup 檔可正確執行	☑已完成 □未完成	
8	報告字型、點數、對齊、行距、頁碼等格式正確	☑已完成 □未完成	
9	報告封面、側邊格式正確	☑已完成 □未完成	
10	報告附錄程式格式正確	☑已完成 □未完成	

5. 收穫:

技巧:學習到如何用一張圖片，利用三角函數來自動生成出 360 度的旋轉照片，再配合數學方程式讓圖片移動時能呈現出曲線的路徑,達到一邊旋轉一邊曲線移動的效果。

多型跟繼承在這次的遊戲裡也運用到很多地方，像是我是先有一個基本的船艦，並先設定好他所擁有的物件，因此當我新增 6 種船艦時，都可以用繼承本來那個最簡陋的船艦。每個船艦所用有的子彈類別則是利用多型的方法，另外還有船艦碰到障礙地形後會不會產生傷害也是用到多型的特性。

氣象系統方面則同時運用了繼承與多型的方面，像是颱風繼承統整全部氣象系統的旋轉功能，而霧則可以看出多型的功能，因為只有在霧裡的船艦沒辦法移動。

除錯方法:我一開始的除錯方法是每一個小功能寫完，會先立馬測試看有沒有成功，這樣就不會因為新增太多東西，不知道該從哪裡開始找，像是射擊所發出的 360 度紅外線就是先畫一條線在一個方便測試的位置看他的角度會不會跟主角的選轉角相同。可是後來發現這個方法還是有點像大海撈針，有時候怎麼看都還是看不出來錯誤，所以後來我會先看如果已經有建立類似的功能，例如本來是主角打到物品後，物品會消失，但是現在錯誤的是敵人打到後，物品不會有反應，則可以先把原本成功的主角程式碼移到會失敗的敵人裡面測試，之後看看原本是主角的那個還是不是可以做到相同功能。如果一切都正常的話，那就再進一步用相同的方法去確認是敵人的槍有沒有碰撞成功，不行的話就知道要再往更初始的地方回推，同樣地也用相同道裡的方法去看敵人的在一開始建立的時候是不是有少寫。而我覺得這個方法挺方便的，像我前面提及的按鈕顯示問題，後來就是用這種方法把失敗的新按鈕跟舊的之間互相對換，最後才找到的。

我覺得找一個地方把有用到的變數印出來也是很有效率的方法，這樣可以確認每個變數值是真的跟你想的一樣，像是我更改完我的遊戲操作按鈕在測試的時候，不管我怎麼輸入，下面的紀錄都顯示失敗，然後我也是看了很久是不是我在字串的處理更改錯了，多讀到一格或是少讀，最後我是把獨到的字串印出來一看，發現果然是我少輸入一個空白了。之後在更改可以點擊地圖的那個方法，我也是先把被點擊到的座標印出來，確認值是對的，再寫到程式裡去做後續處理。

6. 心得、感想:

在這次的製作遊戲專題中，我覺得在幾個方面有很大的成就感，像是上面所提到的那幾個除錯方法，後來我也在其他門要寫專案的課使用這個方法，我覺得真的找錯誤變快很多，之前一開始我都是從輸出錯誤第一行開始看，因此常常除錯的時間都比寫程式還久，也曾經因為找太久作業已經要截止了，所以請同學幫忙，但是現在因為是每組的專案都不一樣了，所以不太能說每個禮拜都一直麻煩同學，因此在寫專案的同時能找到這幾個除錯方法，覺得很實用。或是，雖然一開始花了一會兒去瞭解遊戲的基本用法，但是當自己實際使用這些函式後，做成專屬自己的遊戲時，也是感覺哇自己真強，當然有時候也會因為其他專業課程需要花時間去研究，所以當課程進行到中間時，曾經覺得有點累，似乎應付不太過來，再加上到期中了才突然決定換題目，距離下次展示也只剩幾週，所以那時候突然就感覺不知道該如何是好，壓力也隨之增劇，但我後來想想班上的同學也是一樣啊會累會辛苦，可是也都還在堅持，而且也已經努力到一半了，怎麼能隨便說放棄呢？所以最後我心裡就想說那跟大家一起撐下去吧，遇到難關就跨過去吧，在這個過程也很多老師同學助教給予我一些建議跟幫忙，並一起加油，所以在最後完成的那一刻，真的是很感動，也很慶幸自己當初有趕緊站穩腳步去趕上跟大家落下的差距跟感謝幫助過我的每一位。

附錄：

Command.h:

```
#pragma once
#include "VesselObject.h"
#include "VesselCV.h"
#include "VesselBB.h"Flatland
#include "VesselCG.h"
#include "VesselDD.h"
#include "VesselLV.h"
#include "VesselAH.h"
#include "Team.h"
#include "Shell.h"
#include "Torpedo.h"
```

```
#include "GeographicSystem.h"
#include "Reef.h"
#include "Flatland.h"
#include "Mountain.h"
#include "HydrometeorologicSystem.h"
#include "Lightning.h"
#include "Typhoon.h"
#include "Fog.h"
#define DEBUG
#ifdef DEBUG
using namespace System::Diagnostics;
#endif
using namespace System;
namespace CommandAnalyze {
	enum CommandsType {
		SET_SUCCESS, SET_FAIL,			// set新增
		FIRE_SUCCESS, FIRE_FAIL,		// fire攻擊
		DEFENSE_SUCCESS, DEFENSE_FAIL,	// defense防禦
		TAG_SUCCESS, TAG_FAIL,			// tag更名
		MOVE_SUCCESS, MOVE_FAIL,		// move移動
		LASER_SUCCESS, LASER_FAIL,		// laser雷射
		ENRICH_SUCCESS, ENRICH_FAIL,	// enrich救濟
		LAUNCH_SUCCESS, LAUNCH_FAIL,	// launch魚雷
		UNKNOWN_CMD_TYPE
	};
	String^ processCommand(int, int, String^, Team^, Team^);
	double findNumberInString(String^);
}
```

Command.cpp:

```
#include "Command.h"
namespace CommandAnalyze {
	String^ processCommand(int minute, int second, String^ inpString, Team^ team, Team^ anotherTeam) {
		String^ cmdType = "";
		String^ vesselName = "";
		array<String^>^ elementOfCmd = inpString->Split(); // split by space
		int k, first;
		CommandsType cmdRes;
		String^ resString = "";
```

```
// get command type and vessel name
for (k = 0, first = 1; k < elementOfCmd->Length; k++) {
        if (elementOfCmd[k] != "") {
                if (first == 1) {
                        cmdType = elementOfCmd[k];
                        first = 0;
                }else {
                        vesselName = elementOfCmd[k];
                        break;
                }
        }
}
// now k is at the index of [Vessel_Name]
#pragma region coordinate format process
if (cmdType == "SET" || cmdType == "FIRE" || cmdType == "LAUNCH") {
        int lastIdx = elementOfCmd->Length - 1;
        for (; lastIdx >= 0; lastIdx--) {
                if (elementOfCmd[lastIdx] != "")
                        break;
        }
        for (int u = elementOfCmd[lastIdx]->Length - 1; u >= 0; u--) {
                if (elementOfCmd[lastIdx][u] == ',') {
                        for (int v = inpString->Length - 1; v >= 0; v--) {
                                if (inpString[v] == ',') {
                                        inpString = inpString->Substring(0, v) + " " + inpString->Substring(v);
                                        break;
                                }
                        }
                        delete[] elementOfCmd;
                        elementOfCmd = inpString->Split();
                        // get command type and vessel name
                        for (k = 0, first = 1; k < elementOfCmd->Length; k++) {
                                if (elementOfCmd[k] != "") {
                                        if (first == 1) {
                                                cmdType =elementOfCmd[k];
                                                first = 0;
                                        }
                                        else {
```

```
                        vesselName = elementOfCmd[k];
                        break;
                    }
                }
            }
            // now k is at the index of [Vessel_Name]
            break;
        }
    }
}
#pragma endregion
// SET [Vessel_Name] [Type] [2DCoordinate]
if (cmdType == "SET") {
    String^ vesselType = "";
    double newX = -10.0, newY = -10.0;
    cmdRes = SET_SUCCESS;
    // get vessel type and coordinate
    for (k++, first = 1; k < elementOfCmd->Length; k++) {
        if (elementOfCmd[k] != "") {
            if (first == 1) {
                vesselType = elementOfCmd[k];
                first = 0;
            }
            else if (first == 0) {
                newX = findNumberInString(elementOfCmd[k]);
                first = -1;
            }
            else {
                newY = findNumberInString(elementOfCmd[k]);
                break;
            }
        }
    }
    // Fail: already exist a vessel which name is the same and at the same team
    if (team->vesselList.ContainsKey(vesselName)) {
        cmdRes = SET_FAIL;
    }
    // Search if there is another vessel which is at the new coordinate
```

```
            for each (KeyValuePair<String^, VesselObject^> vessel in team->vesselList) {
                double dis2 = ((vessel.Value->getCoord().X - newX) * (vessel.Value->getCoord().X - newX)) +
((vessel.Value->getCoord().Y - newY) * (vessel.Value->getCoord().Y - newY));
                // Fail: there is another vessel in the new coordinate
                if (dis2 <= COLLAPSE_DISTANCE * COLLAPSE_DISTANCE) {
                    cmdRes = SET_FAIL;
                    break;
                }
            }
            for each (KeyValuePair<String^, VesselObject^> vessel in anotherTeam->vesselList) {
                double dis2 = ((vessel.Value->getCoord().X - newX) * (vessel.Value->getCoord().X - newX)) +
((vessel.Value->getCoord().Y - newY) * (vessel.Value->getCoord().Y - newY));
                // Fail: there is another vessel in the new coordinate
                if (dis2 <= COLLAPSE_DISTANCE * COLLAPSE_DISTANCE) {
                    cmdRes = SET_FAIL;
                    break;
                }
            }
            // Search if there the location is at a geography area
            for each (GeographicSystem^ geo in GeographicSystem::geoList) {
                // Fail: you cannot set a vessel at a geography area
                if (geo->getGp()->IsVisible(COORD_TO_DRAWING_COORD(newX, newY))) {
                    cmdRes = SET_FAIL;
                    break;
                }
            }
            #pragma region create any types of vessels
            if (vesselType == "CV" && cmdRes == SET_SUCCESS) {
                VesselCV^ newCV = gcnew VesselCV(vesselName, team->getSymbol(), PointF(newX, newY));
                team->vesselList.Add(vesselName, newCV);
            }
            else if (vesselType == "BB" && cmdRes == SET_SUCCESS) {
                VesselBB^ newBB = gcnew VesselBB(vesselName, team->getSymbol(), PointF(newX, newY));
                team->vesselList.Add(vesselName, newBB);
            }
            else if (vesselType == "CG" && cmdRes == SET_SUCCESS) {
                VesselCG^ newCG = gcnew VesselCG(vesselName, team->getSymbol(), PointF(newX, newY));
                team->vesselList.Add(vesselName, newCG);
```

```
			}
			else if (vesselType == "DD" && cmdRes == SET_SUCCESS) {
				VesselDD^ newDD = gcnew VesselDD(vesselName, team->getSymbol(), PointF(newX, newY));
				team->vesselList.Add(vesselName, newDD);
			}
			else if (vesselType == "LV" && cmdRes == SET_SUCCESS) {
				VesselLV^ newLV = gcnew VesselLV(vesselName, team->getSymbol(), PointF(newX, newY));
				team->vesselList.Add(vesselName, newLV);
			}
			else if (vesselType == "AH" && cmdRes == SET_SUCCESS) {
				VesselAH^ newAH = gcnew VesselAH(vesselName, team->getSymbol(), PointF(newX, newY));
				team->vesselList.Add(vesselName, newAH);
			}
			#pragma endregion
			// Fail: unknown vessel type
			else {
				cmdRes = SET_FAIL;
			}
			if(vesselName!="")
				resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (wchar_t)(team-
>getSymbol() + 65) + " SET " + vesselName + " with " + vesselType + " at (" + newX.ToString() + ", " + newY.ToString() + ") ->
" + ((cmdRes == SET_SUCCESS) ? "Success\n" : "Fail\n");
			return resString;
		}
		// FIRE [Vessel_Name] [2DCoordinate]
		else if (cmdType == "FIRE") {
			double newX = -10.0, newY = -10.0;
			double nowX = -10.0, nowY = -10.0, dis, dis2; // dis2 means the square of dis, that is, (dis * dis)
			String^ newShellName;
			VesselObject^ thisVessel;
			cmdRes = FIRE_SUCCESS;
			// get the coordinate
			for (k++, first = 1; k < elementOfCmd->Length; k++) {
				if (elementOfCmd[k] != "") {
					if (first == 1) {
						newX = findNumberInString(elementOfCmd[k]);
						first = 0;
					}
```

```
                else {
                        newY = findNumberInString(elementOfCmd[k]);
                        break;
                }
        }
}
// set the name of new shell
newShellName = "Shell_" + ((team->getSymbol() == A) ? "A" : "B") + (team->getLauchedShellNum() +
1).ToString();
// find the vessel from lists
if (team->vesselList.ContainsKey(vesselName))
        thisVessel = team->vesselList[vesselName];
// Fail: Vessel not found
else
        cmdRes = FIRE_FAIL;
// Fail: LV and AH cannot FIRE
if (cmdRes == FIRE_SUCCESS && (thisVessel->getVesselType() == LV || thisVessel->getVesselType() ==
AH))
        cmdRes = FIRE_FAIL;
// Fail: the location of goal is out of range
if (newX < 0.0 || newX > 20.0 || newY < 0.0 || newY > 20.0)
        cmdRes = FIRE_FAIL;
if (cmdRes == FIRE_SUCCESS) {
        // calculate the coordinate of vessel and attack distance
        nowX = thisVessel->getCoord().X; nowY = thisVessel->getCoord().Y;
        dis2 = ((nowX - newX) * (nowX - newX)) + ((nowY - newY) * (nowY - newY));
        dis = Math::Sqrt(dis2);
        // Fail: the attack distance is over than the max attack distance
        if (dis2 > thisVessel->getMaxAttackDistance() * thisVessel->getMaxAttackDistance())
                cmdRes = FIRE_FAIL;
        // Fail: still in CD time
        if (thisVessel->getRemainAttackCD() > 0.0) {
                cmdRes = FIRE_FAIL;
        }
}
if (cmdRes == FIRE_SUCCESS) {
        team->plus1LauchShellNum();
        double angle;
```

```
                        if (nowY == newY && nowX == newX)
                                angle = 0.0, newY += 0.01, newX += 0.01;
                        else
                                angle = (180.0 / Math::PI) * Math::Atan2(nowY - newY, nowX - newX);
                        Shell^ newShell = gcnew Shell(newShellName, team->getSymbol(), thisVessel->getShellSpeed(),
thisVessel->getDamage(), angle, thisVessel->getCoord(), PointF(newX, newY));
                        team->shellList.Add(newShell);
                }
                // enter attack CD time
                if (thisVessel != nullptr && thisVessel->getRemainAttackCD() == 0.0) {
                        thisVessel->setRemainAttackCD(thisVessel->getAttackCD());
                }
                if (vesselName != "")
                        resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (wchar_t)(team-
>getSymbol() + 65) + " " + vesselName + " FIRE to " + "(" + newX.ToString() + ", " + newY.ToString() + ") -> " + ((cmdRes ==
FIRE_SUCCESS) ? (newShellName + "\r\n") : "Fail\r\n");
                return resString;
        }
        // DEFENSE [Vessel_Name] [Shell_Name]
        else if (cmdType == "DEFENSE") {
                String^ shellName;
                VesselObject^ thisVessel;
                double dis2;
                bool hasHitShell = false;
                bool hasFoundShell = false;
                cmdRes = DEFENSE_SUCCESS;
                // get the shell name
                for (k++; k < elementOfCmd->Length; k++) {
                        if (elementOfCmd[k] != "") {
                                shellName = elementOfCmd[k];
                                break;
                        }
                }
                // find the vessel from lists
                if (team->vesselList.ContainsKey(vesselName))
                        thisVessel = team->vesselList[vesselName];
                // Fail: Vessel not found
                else
```

```
                    cmdRes = DEFENSE_FAIL;
                // Fail: LV and AH cannot DEFENSE
                if (thisVessel != nullptr && cmdRes == DEFENSE_SUCCESS && (thisVessel->getVesselType() == LV ||
thisVessel->getVesselType() == AH))
                    cmdRes = DEFENSE_FAIL;
                // Fail: still in CD time
                if (cmdRes == DEFENSE_SUCCESS && thisVessel->getRemainDefenseCD() > 0.0) {
                    cmdRes = FIRE_FAIL;
                }
                if (cmdRes == DEFENSE_SUCCESS) {
                    for (int i = 0; i < team->shellList.Count && !hasFoundShell; i++) {
                        Shell^ thisShell = team->shellList[i];
                        // found the shell
                        if (thisShell->getName() == shellName) {
                            hasFoundShell = true;
                            dis2 = ((thisShell->getNowCoord().X - thisVessel->getCoord().X) * (thisShell-
>getNowCoord().X - thisVessel->getCoord().X)) + ((thisShell->getNowCoord().Y - thisVessel->getCoord().Y) * (thisShell-
>getNowCoord().Y - thisVessel->getCoord().Y));
                            // hit
                            if(dis2<=thisVessel->getMaxDefenseDistance()*thisVessel-
>getMaxDefenseDistance()) {
                                hasHitShell = true;
                                team->shellList.RemoveAt(i);
                                break;
                            }
                        }
                    }
                    for (int i = 0; i < anotherTeam->shellList.Count && !hasFoundShell; i++) {
                        Shell^ thisShell = anotherTeam->shellList[i];
                        // found the shell
                        if (thisShell->getName() == shellName) {
                            dis2 = ((thisShell->getNowCoord().X - thisVessel->getCoord().X) * (thisShell-
>getNowCoord().X - thisVessel->getCoord().X)) + ((thisShell->getNowCoord().Y - thisVessel->getCoord().Y) * (thisShell-
>getNowCoord().Y - thisVessel->getCoord().Y));
                            // hit
                            if (dis2 <= thisVessel->getMaxDefenseDistance() * thisVessel-
>getMaxDefenseDistance()) {
                                hasHitShell = true;
```

```
                                                anotherTeam->shellList.RemoveAt(i);
                                                break;
                                        }
                                }
                        }
                        if (!hasHitShell)
                                cmdRes = DEFENSE_FAIL;
                }
                // enter defense CD time
                if (thisVessel != nullptr && thisVessel->getRemainDefenseCD() == 0.0) {
                        thisVessel->setRemainDefenseCD(thisVessel->getDefenseCD());
                }
                if (vesselName != "")
                        resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] " + vesselName + "
DEFENSE " + shellName + " -> " + (cmdRes == DEFENSE_SUCCESS ? "Hit\r\n" : "Fail\r\n");
                return resString;
        }
        // TAG [Vessel_Name] [New_Name]
        else if (cmdType == "TAG") {
                String^ newName = "";
                VesselObject^ newVessel = gcnew VesselObject;
                cmdRes = TAG_SUCCESS;
                for (k++; k < elementOfCmd->Length; k++) {
                        if (elementOfCmd[k] != "") {
                                newName = elementOfCmd[k];
                                break;
                        }
                }
                if (newName != "" && team->vesselList.ContainsKey(vesselName) && !team-
>vesselList.ContainsKey(newName)) {
                        newVessel = team->vesselList[vesselName];
                        newVessel->setName(newName);
                        team->vesselList.Add(newName, newVessel);
                        team->vesselList.Remove(vesselName);
                }
                else {
                        cmdRes = TAG_FAIL;
                }
```

```
			if (vesselName != "")
				resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (wchar_t)(team->getSymbol() + 65) + " RENAME " + vesselName + " to " + newName + " -> " + ((cmdRes == TAG_SUCCESS) ? "Success\r\n" : "Fail\r\n");
			return resString;
		}
		// MOVE [Vessel_Name] [Speed] [Angle]
		else if (cmdType == "MOVE") {
			VesselObject^ thisVessel;
			double moveSpeed = -10.0, moveAngle = -10.0;
			cmdRes = MOVE_SUCCESS;
			// get the speed and the angle
			for (k++, first = 1; k < elementOfCmd->Length; k++) {
				if (elementOfCmd[k] != "") {
					if (first == 1) {
						moveSpeed = findNumberInString(elementOfCmd[k]);
						first = 0;
					}
					else {
						moveAngle = findNumberInString(elementOfCmd[k]);
						break;
					}
				}
			}
			if (moveSpeed == -10.0) {
				cmdRes = MOVE_FAIL;
			}
			// find the vessel from lists
			if (team->vesselList.ContainsKey(vesselName))
				thisVessel = team->vesselList[vesselName];
			// Fail: Vessel not found
			else
				cmdRes = MOVE_FAIL;
			if (cmdRes == MOVE_SUCCESS) {
				// Fail: move speed is over the max vessel speed
				if (Math::Abs(moveSpeed) > (thisVessel->getMaxVesselSpeed()) * 60.0)
					cmdRes = MOVE_FAIL;
				// Fail: angle is negative value or is over 359
				if (moveAngle < 0.0 || moveAngle > 359.0)
					cmdRes = MOVE_FAIL;
				// Fail: in fog area, cannot moving
				if (thisVessel->getCanMove() == false) {
					cmdRes = MOVE_FAIL;
					Debug::WriteLine("cmdRes, getMove false");
				}
			}
			//Debug::WriteLine(moveSpeed.ToString() + ", " + moveAngle.ToString());
			if (cmdRes == MOVE_SUCCESS) {
				thisVessel->setNowSpeed(moveSpeed / 60.0);
				thisVessel->setNowAngle(moveAngle);
			}
			else {
				if (thisVessel != nullptr) {
					thisVessel->setNowSpeed(0.0);
					thisVessel->setNowAngle(0.0);
				}
			}
			if (vesselName != "")
				resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (wchar_t)(team->getSymbol() + 65) + " " + vesselName + " MOVE to " + moveAngle + " as " + moveSpeed + " -> " + ((cmdRes == MOVE_SUCCESS) ? "Success\r\n" : "Fail\r\n");
			return resString;
		}
		// LASER [Vessel_Name] [Angle]
		else if (cmdType == "LASER") {
			VesselObject^ thisVessel;
			double laserAngle = -10.0;
			cmdRes = LASER_SUCCESS;
			// get the angle
			for (k++; k < elementOfCmd->Length; k++) {
				if (elementOfCmd[k] != "") {
					laserAngle = findNumberInString(elementOfCmd[k]);
					break;
				}
			}
			// find the vessel from lists
```

```
            if (team->vesselList.ContainsKey(vesselName))
                thisVessel = team->vesselList[vesselName];
            // Fail: Vessel not found
            else
                cmdRes = LASER_FAIL;
            // Fail: still remain laser CD
            if (cmdRes == LASER_SUCCESS && thisVessel->getRemainLaserCD() > 0.0)
                cmdRes = LASER_FAIL;
            // Fail: angle is negative value or is over 359
            if (laserAngle < 0.0 || laserAngle > 359.0)
                cmdRes = LASER_FAIL;
            // Fail: only LV can use laser attack
            if (thisVessel != nullptr && thisVessel->getVesselType() != LV)
                cmdRes = LASER_FAIL;
            if (cmdRes == LASER_SUCCESS) {
                thisVessel->SetLaserAngle(laserAngle);
                thisVessel->setIsLaserShooting(true);
                thisVessel->setRemainLaserCD(thisVessel->getLaserCD());
            }
            if (vesselName != "")
                resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (wchar_t)(team->getSymbol() + 65) + " " + vesselName + " use LASER -> " + ((cmdRes == LASER_SUCCESS) ? "Success\r\n" : "Fail\r\n");
            return (cmdRes == LASER_FAIL) ? resString : "";
        }
        // ENRICH [Vessel_Name] [Vessel_Name_whichNeedEnrichment]
        else if (cmdType == "ENRICH") {
            String^ enrichedVesselName = "";
            VesselObject^ enrichedVessel = gcnew VesselObject;
            VesselObject^ thisVessel = gcnew VesselObject;
            cmdRes = ENRICH_SUCCESS;
            // get the name of vessel which need enrichment
            for (k++; k < elementOfCmd->Length; k++) {
                if (elementOfCmd[k] != "") {
                    enrichedVesselName = elementOfCmd[k];
                    break;
                }
            }
            // Fail: the AH vessel is NOT in the list
```

```
                    if (!team->vesselList.ContainsKey(vesselName))
                        cmdRes = ENRICH_FAIL;
                    else
                        thisVessel = team->vesselList[vesselName];
                    // Fail: the vessel which need enrichment is NOT in the same-team-list
                    if (!team->vesselList.ContainsKey(enrichedVesselName))
                        cmdRes = ENRICH_FAIL;
                    else
                        enrichedVessel = team->vesselList[enrichedVesselName];
                    // Fail: only AH can do enrichment
                    if (cmdRes == ENRICH_SUCCESS && thisVessel->getVesselType() != AH)
                        cmdRes = ENRICH_FAIL;
                    // Fail: the location of the vessel which need enrichment is out of the enrich range of AH vessel
                    if (cmdRes == ENRICH_SUCCESS && !thisVessel->isEnrichVessel(enrichedVessel->getCoord().X,
enrichedVessel->getCoord().Y))
                        cmdRes = ENRICH_FAIL;
                    // Fail: Still in CD time
                    if (cmdRes == ENRICH_SUCCESS && thisVessel->get_remain_Enrich_CD() > 0.0)
                        cmdRes = ENRICH_FAIL;
                    if (cmdRes == ENRICH_SUCCESS)
                        enrichedVessel->setRemainHp(enrichedVessel->getRemainHp() + thisVessel->getEnrichValue());
                    if (thisVessel != nullptr && thisVessel->get_remain_Enrich_CD() == 0.0) {
                        thisVessel->set_remain_Enrich_CD(thisVessel->getEnrichValue_CD());
                    }
                    if (vesselName != "")
                        resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (wchar_t)(team-
>getSymbol() + 65) + " " + vesselName    + " ENRICH " + enrichedVesselName + " -> " + ((cmdRes == ENRICH_SUCCESS) ?
"Success\r\n" : "Fail\r\n");
                    return resString;
                }
                // LAUNCH [Vessel_Name] [2DCoordinate]
                else if (cmdType == "LAUNCH") {
                    double newX = -10.0, newY = -10.0;
                    double nowX = -10.0, nowY = -10.0, dis, dis2; // dis2 means the square of dis, that is, (dis * dis)
                    String^ newTorpedoName;
                    VesselObject^ thisVessel;
                    cmdRes = LAUNCH_SUCCESS;
                    // Get the coordinate
```

```
			for (k++, first = 1; k < elementOfCmd->Length; k++) {
				if (elementOfCmd[k] != "") {
					if (first == 1) {
						newX = findNumberInString(elementOfCmd[k]);
						first = 0;
					}
					else {
						newY = findNumberInString(elementOfCmd[k]);
						break;
					}
				}
			}
			// Set the name of new torpedo
			newTorpedoName = "Torpedo_" + ((team->getSymbol() == A) ? "A" : "B") + (team->getLaunchedTorpedoNum() + 1).ToString();
			// Find the vessel from lists
			if (team->vesselList.ContainsKey(vesselName))
				thisVessel = team->vesselList[vesselName];
			// Fail: Vessel not found
			else
				cmdRes = LAUNCH_FAIL;
			// Fail: LV & CV & BB & AH cannot LAUNCH torpedo, only CG and DD can
			if (cmdRes == LAUNCH_SUCCESS && !(thisVessel->getVesselType() == CG || thisVessel->getVesselType() == DD))
				cmdRes = LAUNCH_FAIL;
			// Fail: the location of goal is out of range
			if (newX < 0.0 || newX > 20.0 || newY < 0.0 || newY > 20.0)
				cmdRes = LAUNCH_FAIL;
			if (cmdRes == LAUNCH_SUCCESS) {
				// calculate the coordinate of vessel and attack distance
				nowX = thisVessel->getCoord().X; nowY = thisVessel->getCoord().Y;
				dis2 = ((nowX - newX) * (nowX - newX)) + ((nowY - newY) * (nowY - newY));
				dis = Math::Sqrt(dis2);
				// Fail: the attack distance is over than the max attack distance
				if (dis2 > thisVessel->getMaxAttackDistance() * thisVessel->getMaxAttackDistance())
					cmdRes = LAUNCH_FAIL;
				// Fail: still in CD time
				if (thisVessel->getRemainLaunchCD() > 0.0) {
```

```
                        cmdRes = LAUNCH_FAIL;
                    }
                }
                if (cmdRes == LAUNCH_SUCCESS) {
                    team->plus1LauchShellNum();
                    double angle;
                    if (nowY == newY && nowX == newX)
                        angle = 0.0, newY += 0.01, newX += 0.01;
                    else
                        angle = (180.0 / Math::PI) * Math::Atan2(nowY - newY, nowX - newX);
                    Torpedo^ newTorpedo = gcnew Torpedo(newTorpedoName, thisVessel->getName(), team->getSymbol(), thisVessel->getTorpedoSpeed(), thisVessel->getTorpedoDamage(), angle, thisVessel->getCoord(), PointF(newX, newY));
                    team->torpedoList.Add(newTorpedo);
                }
                // Enter launch CD time
                if (thisVessel != nullptr && thisVessel->getRemainLaunchCD() == 0.0) {
                    thisVessel->setRemainLaunchCD(thisVessel->getLaunchCD());
                }
                if (vesselName != "")
                    resString = "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (wchar_t)(team->getSymbol() + 65) + " " + vesselName + " LAUNCH a torpedo to " + "(" + newX.ToString() + ", " + newY.ToString() + ") -> " + ((cmdRes == LAUNCH_SUCCESS) ? (newTorpedoName + "\r\n") : "Fail\r\n");
                return resString;
            }
            // Unknown Command
            else {
                return "Unknown command\r\n";
            }
        }
        // find a number in a string, ex: "(,,12|3.2,"    -> 123.2
        double findNumberInString(String^ str) {
            double num = -10.0;
            double fracDigit = 0.1;
            bool isNeg = false;
            for (int i = 0, isIntegerPart = 1; i < str->Length; i++) {
                if (str[i] >= '0' && str[i] <= '9') {
                    if (num == -10.0)
```

```
					num = 0.0;
				if (isIntegerPart == 1)
					num = (num * 10.0) + (double)((int)str[i] - 48);
				else
					num += (double)((int)str[i] - 48) * fracDigit, fracDigit *= 0.1;
			}
			else if (str[i] == '.')
				isIntegerPart = 0;
			else if (str[i] == '-')
				isNeg = true;
		}
		if (num == -10.0)
			return num;
		return (isNeg ? (-num) : num);
	}
}
```

Flatland.h:

```
#pragma once
#include "GeographicSystem.h"
ref class Flatland : public GeographicSystem {
public:
	Flatland();
	Flatland(Flatland^&);
	Flatland(PointF, SizeF);
	Flatland^ operator=(Flatland^);
	virtual bool render(Graphics^) override;
};
```

Flatland.cpp:

```
#include "Flatland.h"
Flatland::Flatland() : GeographicSystem() {}
Flatland::Flatland(Flatland^& rSide) {
	geoType = rSide->geoType;
	name = rSide->name;
	gpGeography = rSide->gpGeography;
	mouseIn = false;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	colorAlpha = rSide->colorAlpha;
	reefDamage = rSide->reefDamage;
	canGoThrough_vessel = rSide->canGoThrough_vessel;
	canGoThrough_generalFire = rSide->canGoThrough_generalFire;
	canGoThrough_torpedo = rSide->canGoThrough_torpedo;
	canGoThrough_laser = rSide->canGoThrough_laser;
}
Flatland::Flatland(PointF newCoord, SizeF newSize) : GeographicSystem(FLATLAND, newCoord, newSize) {
	canGoThrough_vessel = false;
	canGoThrough_generalFire = true;
	canGoThrough_torpedo = false;
	canGoThrough_laser = true;
}
Flatland^ Flatland::operator=(Flatland^ rSide) {
	geoType = rSide->geoType;
	name = rSide->name;
	gpGeography = rSide->gpGeography;
	mouseIn = false;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	colorAlpha = rSide->colorAlpha;
	reefDamage = rSide->reefDamage;
	canGoThrough_vessel = rSide->canGoThrough_vessel;
	canGoThrough_generalFire = rSide->canGoThrough_generalFire;
	canGoThrough_torpedo = rSide->canGoThrough_torpedo;
	canGoThrough_laser = rSide->canGoThrough_laser;
	return this;
}
bool Flatland::render(Graphics^ g) {
	Brush^ bruGeo = gcnew SolidBrush(Color::FromArgb(colorAlpha, 23, 100, 14));
	g->FillPath(bruGeo, gpGeography);
	GeographicSystem::render(g);
	return true;
}
```

Fog.h:

```
#pragma once
#include "HydrometeorologicSystem.h"
#define SPEED                   0.0
#define DAMAGE                  0.0
#define ATTACK_RANGE   1.0
#define MAX_FOG_SIZE     15.0
ref class Fog : public HydrometeorologicSystem {
public:
	Fog();
	Fog(Fog^&);
	Fog^ operator=(Fog^);
	virtual bool render(Graphics^) override;
	#pragma region getters and setters
	virtual double getRotatePictureAngle() override;
	virtual SizeF getSize() override;
	virtual void setSize(SizeF) override;
	virtual SizeF getSize4Draw() override;
	virtual void setSize4Draw(SizeF) override;
	#pragma endregion
private:
	SizeF size;
	SizeF size4Draw;
};
```

Fog.cpp:

```
#include "Fog.h"
Fog::Fog() : HydrometeorologicSystem() {
	Random^ r = gcnew Random(DateTime::Now.Millisecond);
	hydroType = FOG;
	PointF centerNowCoord = PointF(r->NextDouble() * BOARD_SIZE, r->NextDouble() * BOARD_SIZE);
	SizeF centerSize = SizeF(r->NextDouble() * Math::Min((double)centerNowCoord.X, (double)BOARD_SIZE - (double)centerNowCoord.X), r->NextDouble() * Math::Min((double)centerNowCoord.Y, (double)BOARD_SIZE - (double)centerNowCoord.Y));
	nowCoord = PointF(centerNowCoord.X - (centerSize.Width / 2.0), centerNowCoord.Y - (centerSize.Height / 2.0));
	nowCoord4Draw = COORD_TO_DRAWING_COORD(nowCoord.X, nowCoord.Y);
	size = centerSize;
	size4Draw = SIZE_TO_DRAWING_SIZE(size.Width, size.Height);
	gpHydro = gcnew GraphicsPath();
	gpHydro->AddEllipse(nowCoord4Draw.X, nowCoord4Draw.Y, size4Draw.Width, size4Draw.Height);
```

```
}
Fog::Fog(Fog^& rSide) {
	picture = rSide->picture;
	hydroType = rSide->hydroType;
	damage = rSide->damage;
	attackRange = rSide->attackRange;
	speed = rSide->speed;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	existTime = rSide->existTime;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	gpHydro = rSide->gpHydro;
}
Fog^ Fog::operator=(Fog^ rSide) {
	picture = rSide->picture;
	hydroType = rSide->hydroType;
	damage = rSide->damage;
	attackRange = rSide->attackRange;
	speed = rSide->speed;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	existTime = rSide->existTime;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	gpHydro = rSide->gpHydro;
	return this;
}
double Fog::getRotatePictureAngle() {
	return 0.0;
}
SizeF Fog::getSize() {
	return size;
}
void Fog::setSize(SizeF newSize) {
	size = newSize;
```

```
}
SizeF Fog::getSize4Draw() {
	return size4Draw;
}
void Fog::setSize4Draw(SizeF newSize4Draw) {
	size4Draw = newSize4Draw;
}
//draw fog
bool Fog::render(Graphics^ g) {
	g->FillPath(gcnew SolidBrush(Color::FromArgb(199, 255, 255, 255)), gpHydro);
	return true;
}
GeographicSystem.h:
#include "RenderParameter.h"
#include "Team.h"
#define VESSEL_AND_GEO_COLLAPSION_DIS ((VESSEL_ICON_LENGTH / 2.0) / BLOCK_LENGTH)
#define SHELL_AND_GEO_COLLAPSION_DIS ((SHELL_ICON_LENGTH / 2.0) / BLOCK_LENGTH)
#define TORPEDO_AND_GEO_COLLAPSION_DIS ((SHELL_ICON_LENGTH / 2.0) / BLOCK_LENGTH)
#define ALPHA_UP_BOUND 255
#define ALPHA_LOW_BOUND 55
#define VIRTUAL_GEO_ALPHA 144
#define REEF_DAMAGE 1.0
using namespace System;
using namespace System::Windows;
using namespace System::Drawing;
using namespace System::Drawing::Drawing2D;
using namespace System::Collections::Generic;
using namespace System::Diagnostics;
enum GeographyType { REEF, FLATLAND, MOUNTAIN, UNKNOWN_GEO_TYPE };
/*
	Reef    Vessel-> F + Damage      Shell-> T        Torpedo -> F        Laser-> T
	Flatland    Vessel-> F      Shell-> T        Torpedo -> F      Laser-> T
	Mountain    Vessel-> F        Shell-> F        Torpedo -> F        Laser  -> T
*/
ref class GeographicSystem {
public:
	GeographicSystem();
	GeographicSystem(const GeographicSystem^&);
	GeographicSystem(GeographyType, PointF, SizeF);
	GeographyType getGeoType();
	void setGeoType(GeographyType);
	String^ getName();
	void setName(String^);
	PointF getCoord();
	void setCoord(PointF);
	PointF getCoord4Draw();
	void setCoord4Draw(PointF);
	SizeF getSize();
	void setSize(SizeF);
	SizeF getSize4Draw();
	void setSize4Draw(SizeF);
	int getColorAlpha();
	void setColorAlpha(int);
	bool canVesselGoThrough();
	bool canGeneralFireGoThrough();
	bool canTorpedoGoThrough();
	bool canLaserGoThrough();
	double getReefDamage();
	void setReefDamage(double);
	GraphicsPath^ getGp();
	String^ detectAll(Team^, int, int, int&);
	virtual bool render(Graphics^) override;
	// Geography list
	static List<GeographicSystem^> geoList;
	bool mouseIn;
protected:
	static array<String^>^ geoTypeStringList = gcnew array<String^>(12) { "Reef", "Flatland", "Mountain", "Unknown
Geography Type" };
	GeographyType geoType;
	String^ name;
	GraphicsPath^ gpGeography;
	PointF coord;
	PointF coord4Draw;
	SizeF size;
	SizeF size4Draw;
	int colorAlpha;
	bool canGoThrough_vessel;
	bool canGoThrough_generalFire;
	bool canGoThrough_torpedo;
	bool canGoThrough_laser;
	double reefDamage;
	// 點到線段距離
	double getDistanceBetweenPointAndSegment(PointF, PointF, PointF);
};
GeographicSystem.cpp:
#include "GeographicSystem.h"
GeographicSystem::GeographicSystem() {
	geoType = UNKNOWN_GEO_TYPE;
	name = "Unknown Geography Type";
	gpGeography = gcnew GraphicsPath();
	mouseIn = false;
	coord = coord4Draw = PointF();
	size = size4Draw = SizeF();
	colorAlpha = ALPHA_UP_BOUND;
	reefDamage = REEF_DAMAGE;
	canGoThrough_vessel = canGoThrough_generalFire = canGoThrough_torpedo = canGoThrough_laser = true;
}
GeographicSystem::GeographicSystem(const GeographicSystem^& rSide) {
	geoType = rSide->geoType;
	name = rSide->name;
	gpGeography = rSide->gpGeography;
	mouseIn = false;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	colorAlpha = rSide->colorAlpha;
	reefDamage = rSide->reefDamage;
	canGoThrough_vessel = rSide->canGoThrough_vessel;
	canGoThrough_generalFire = rSide->canGoThrough_generalFire;
	canGoThrough_torpedo = rSide->canGoThrough_torpedo;
	canGoThrough_laser = rSide->canGoThrough_laser;
}
GeographicSystem::GeographicSystem(GeographyType newGeoType, PointF newCoord, SizeF newSize) {
	coord = newCoord;
	coord4Draw = COORD_TO_DRAWING_COORD(coord.X, coord.Y);
	size = newSize;
	size4Draw = SIZE_TO_DRAWING_SIZE(size.Width, size.Height);
	colorAlpha = ALPHA_UP_BOUND;
	geoType = newGeoType;
	name = geoTypeStringList[geoType];
	gpGeography = gcnew GraphicsPath();
	gpGeography->AddRectangle(RectangleF(coord4Draw, size4Draw));
	mouseIn = false;
	reefDamage = REEF_DAMAGE;
	canGoThrough_vessel = canGoThrough_generalFire = canGoThrough_torpedo = canGoThrough_laser = true;
}
GeographyType GeographicSystem::getGeoType() {
	return geoType;
}
void GeographicSystem::setGeoType(GeographyType newGeoType) {
	geoType = newGeoType;
	name = geoTypeStringList[geoType];
}
String^ GeographicSystem::getName() {
	return name;
}
void GeographicSystem::setName(String^ newName) {
	name = newName;
}
PointF GeographicSystem::getCoord() {
	return coord;
}
void GeographicSystem::setCoord(PointF newCoord) {
	coord = newCoord;
}
PointF GeographicSystem::getCoord4Draw() {
	return coord4Draw;
}
void GeographicSystem::setCoord4Draw(PointF newCoord4Draw) {
	coord4Draw = newCoord4Draw;
}
```

```
SizeF GeographicSystem::getSize() {
	return size;
}
void GeographicSystem::setSize(SizeF newSize) {
	size = newSize;
}
SizeF GeographicSystem::getSize4Draw() {
	return size4Draw;
}
void GeographicSystem::setSize4Draw(SizeF newSize4Draw) {
	size4Draw = newSize4Draw;
}
int GeographicSystem::getColorAlpha() {
	return colorAlpha;
}
void GeographicSystem::setColorAlpha(int newAlpha) {
	if (newAlpha > ALPHA_UP_BOUND)
		newAlpha = ALPHA_UP_BOUND;
	if (newAlpha < ALPHA_LOW_BOUND)
		newAlpha = ALPHA_LOW_BOUND;
	colorAlpha = newAlpha;
}
bool GeographicSystem::canVesselGoThrough() {
	return canGoThrough_vessel;
}
bool GeographicSystem::canGeneralFireGoThrough() {
	return canGoThrough_generalFire;
}
bool GeographicSystem::canTorpedoGoThrough() {
	return canGoThrough_torpedo;
}
bool GeographicSystem::canLaserGoThrough() {
	return canGoThrough_laser;
}
double GeographicSystem::getReefDamage() {
	return reefDamage;
}
void GeographicSystem::setReefDamage(double newReefDamage) {
		reefDamage = newReefDamage;
}
GraphicsPath^ GeographicSystem::getGp() {
		return gpGeography;
}
String^ GeographicSystem::detectAll(Team^ team, int minute, int second, int& newColorAlpha) {
		double minDis = (double)BOARD_SIZE;
		double dis1 = 0.0, dis2 = 0.0, dis3 = 0.0, dis4 = 0.0;
		double dis1_move = 0.0, dis2_move = 0.0, dis3_move = 0.0, dis4_move = 0.0;
		String^ newLog = "";
		List<Shell^> destroyedShellList;
		List<Torpedo^> destroyedTorpedoList;
		destroyedShellList.Clear();
		destroyedTorpedoList.Clear();
		//Vessel detection
		for each (KeyValuePair<String^, VesselObject^>^ vessel in team->vesselList) {
				if (gpGeography->IsVisible(vessel->Value->getCoord4Draw()) && canVesselGoThrough() == false) {
						vessel->Value->DaoTuiLu();  vessel->Value->DaoTuiLu();  vessel->Value->DaoTuiLu();  vessel->Value-
>DaoTuiLu(); vessel->Value->DaoTuiLu();
						// Special case: vessel hit REEF, it will receive damage from REEF
						if (geoType == REEF) {
								vessel->Value->setNowSpeed(0.0);
								vessel->Value->setNowAngle(0.0);
								vessel->Value->setRemainHp(vessel->Value->getRemainHp() - reefDamage);
								newLog += "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (vessel->Value-
>getTeam() == A ? "A " : "B ") + vessel->Value->getName() + " HIT a " + this->getName() + ", stopped and got some damage\r\n";
						}
						// General case
						else {
								vessel->Value->setNowSpeed(0.0);
								vessel->Value->setNowAngle(0.0);
								newLog += "[" + minute.ToString("00") + ":" + second.ToString("00") + "] Team" + (vessel->Value-
>getTeam() == A ? "A " : "B ") + vessel->Value->getName() + " HIT a " + this->getName() + " and stopped\r\n";
						}
				}
		}
		//Shell detection
		for each (Shell^ shell in team->shellList) {
        dis1 = getDistanceBetweenPointAndSegment(shell->getNowCoord(), coord, PointF(coord.X, coord.Y + size.Height));
        dis2 = getDistanceBetweenPointAndSegment(shell->getNowCoord(), coord, PointF(coord.X + size.Width, coord.Y));
        dis3 = getDistanceBetweenPointAndSegment(shell->getNowCoord(), PointF(coord.X + size.Width, coord.Y), PointF(coord.X + size.Width, coord.Y + size.Height));
        dis4 = getDistanceBetweenPointAndSegment(shell->getNowCoord(), PointF(coord.X, coord.Y + size.Height), PointF(coord.X + size.Width, coord.Y + size.Height));
        if (dis1 <= SHELL_AND_GEO_COLLAPSION_DIS || dis2 <= SHELL_AND_GEO_COLLAPSION_DIS || dis3 <= SHELL_AND_GEO_COLLAPSION_DIS || dis4 <= SHELL_AND_GEO_COLLAPSION_DIS) {
            // shell can not go through
            if (canGeneralFireGoThrough() == false) {
                shell->setSpeed(0.0);
                destroyedShellList.Add(shell);
                newLog += "[" + minute.ToString("00") + ":" + second.ToString("00") + "] " + shell->getName() + " HIT a " + this->getName() + " and vanished\r\n";
            }
        }
        if (dis1 < minDis) minDis = dis1;
        if (dis2 < minDis) minDis = dis2;
        if (dis3 < minDis) minDis = dis3;
        if (dis4 < minDis) minDis = dis4;
    }
    for each (Shell^ shell in destroyedShellList)
        team->shellList.Remove(shell);
    //Torpedo detection
    for each (Torpedo^ torpedo in team->torpedoList) {
        dis1 = getDistanceBetweenPointAndSegment(torpedo->getNowCoord(), coord, PointF(coord.X, coord.Y + size.Height));
        dis2 = getDistanceBetweenPointAndSegment(torpedo->getNowCoord(), coord, PointF(coord.X + size.Width, coord.Y));
        dis3 = getDistanceBetweenPointAndSegment(torpedo->getNowCoord(), PointF(coord.X + size.Width, coord.Y), PointF(coord.X + size.Width, coord.Y + size.Height));
        dis4 = getDistanceBetweenPointAndSegment(torpedo->getNowCoord(), PointF(coord.X, coord.Y + size.Height), PointF(coord.X + size.Width, coord.Y + size.Height));
        if (dis1 <= TORPEDO_AND_GEO_COLLAPSION_DIS || dis2 <= TORPEDO_AND_GEO_COLLAPSION_DIS || dis3 <= TORPEDO_AND_GEO_COLLAPSION_DIS || dis4 <= TORPEDO_AND_GEO_COLLAPSION_DIS) {
            // torpedo can not go through
```

```
                        if (canTorpedoGoThrough() == false) {
                                torpedo->setSpeed(0.0);
                                destroyedTorpedoList.Add(torpedo);

                                newLog += "[" + minute.ToString("00") + ":" + second.ToString("00") + "] " + torpedo->getName()
+ " HIT a " + this->getName() + " and vanished\r\n";
                        }
                }
                if (dis1 < minDis) minDis = dis1;
                if (dis2 < minDis) minDis = dis2;
                if (dis3 < minDis) minDis = dis3;
                if (dis4 < minDis) minDis = dis4;
        }
        for each (Torpedo^ torpedo in destroyedTorpedoList)
                team->torpedoList.Remove(torpedo);
        newColorAlpha = (int)((double)ALPHA_UP_BOUND - (minDis * ((double)(ALPHA_UP_BOUND -
ALPHA_LOW_BOUND) / (double)BOARD_SIZE)));
        return newLog;
}
bool GeographicSystem::render(Graphics^ g) {
        if (mouseIn) {
                g->DrawString(name, gcnew Font("Consolas", 12), gcnew SolidBrush(Color::Azure), PointF(coord4Draw.X,
coord4Draw.Y));
        }
        return true;
}

// 點到線段距離
double GeographicSystem::getDistanceBetweenPointAndSegment(PointF point, PointF nowLoc, PointF endLoc) {
        double x = point.X, y = point.Y;
        double x1 = nowLoc.X, y1 = nowLoc.Y;
        double x2 = endLoc.X, y2 = endLoc.Y;
        double cross = (x2 - x1) * (x - x1) + (y2 - y1) * (y - y1);
        if (cross <= 0)
                return Math::Sqrt((x - x1) * (x - x1) + (y - y1) * (y - y1));
        double d2 = (x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1);
        if (cross >= d2)
                return Math::Sqrt((x - x2) * (x - x2) + (y - y2) * (y - y2));
```

```
	double r = cross / d2;

	double px = x1 + (x2 - x1) * r;

	double py = y1 + (y2 - y1) * r;

	return Math::Sqrt((x - px) * (x - px) + (py - y) * (py - y));

}

HyrdrometeorologicSystem.h:

#pragma once

#include "RenderParameter.h"

#include "Team.h"

#define PICTURE_LIGHTNING	dynamic_cast<Bitmap^>(Bitmap::FromFile("picture/lightning.png"))

#define PICTURE_TYPHOON		dynamic_cast<Bitmap^>(Bitmap::FromFile("picture/typhoon.png"))

#define ROTATE_ANGLE	3.0

#define TOTAL_FOG_TIME100.0

enum HydroType{ TYPHOON, LIGHTNING, FOG, UNKNOWN_HYDRO_TYPE };

ref class HydrometeorologicSystem {

public:

	HydrometeorologicSystem();

	HydrometeorologicSystem(HydrometeorologicSystem^&);

	HydrometeorologicSystem(double, PointF);

	Bitmap^ getPicture();

	void setPicture(Bitmap^);

	HydroType getHydroType();

	void sethydroType(HydroType);

	double getDamage();

	void setDamage(double);

	double getAttackRange();

	void setAttackRange(double);

	double getSpeed();

	void setSpeed(double);

	PointF getNowCoord();

	void setNowCoord(PointF);

	PointF getNowCoord4Draw();

	void setNowCoord4Draw(PointF);

	int getExistTime();

	void setExistTime(int);

	GraphicsPath^ getGp();

	void setGp(GraphicsPath^);

	bool getMouseIn();
	void setMouseIn(bool);
	virtual SizeF getSize() override;
	virtual void setSize(SizeF) override;
	virtual SizeF getSize4Draw() override;
	virtual void setSize4Draw(SizeF) override;
	void hydroMove();
	bool BoundaryJudgment(double, double);
	// Hydrometeorology list
	static List<HydrometeorologicSystem^> hydroList;
	virtual bool render(Graphics^) override;
	// rotate the image
	Bitmap^ RotateImage(Bitmap^, float);
	// helper function: calculate new size
	Size^ CalculateNewSize(int, int, double);
	virtual double getRotatePictureAngle() override;
	virtual void setRotatePictureAngle(double) override;
protected:
	Bitmap^ picture;
	HydroType hydroType;
	double damage;
	double attackRange;
	double speed;
	double angle;
	PointF nowCoord;
	PointF nowCoord4Draw;
	int existTime;
	GraphicsPath^ gpHydro;
	bool mouseIn;
};
HyrdrometeorologicSystem.cpp:
#include "HydrometeorologicSystem.h"
HydrometeorologicSystem::HydrometeorologicSystem() {
	hydroType = UNKNOWN_HYDRO_TYPE;
	damage = speed = angle = 0;
	attackRange = 0;
	nowCoord = nowCoord4Draw = PointF();
	existTime = 0;
}
```

```
HydrometeorologicSystem::HydrometeorologicSystem(HydrometeorologicSystem^& rSide) {
	picture = rSide->picture;
	hydroType = rSide->hydroType;
	damage = rSide->damage;
	attackRange = rSide->attackRange;
	speed = rSide->speed;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	existTime = rSide->existTime;
	gpHydro = rSide->gpHydro;
}
HydrometeorologicSystem::HydrometeorologicSystem(double newAngle, PointF newNowCoord) {
	angle = newAngle;
	nowCoord = newNowCoord;
	nowCoord4Draw = COORD_TO_DRAWING_COORD(newNowCoord.X, newNowCoord.Y);
	existTime = 0;
}
Bitmap^ HydrometeorologicSystem::getPicture() {
	return picture;
}
void HydrometeorologicSystem::setPicture(Bitmap^ newPicture) {
	picture = newPicture;
}
HydroType HydrometeorologicSystem::getHydroType() {
	return hydroType;
}
void HydrometeorologicSystem::sethydroType(HydroType newHydroType) {
	hydroType = newHydroType;
}
double HydrometeorologicSystem::getDamage() {
	return damage;
}
void HydrometeorologicSystem::setDamage(double newDamage) {
	damage = newDamage;
}
double HydrometeorologicSystem::getAttackRange() {
	return attackRange;
```

```
}
void HydrometeorologicSystem::setAttackRange(double newAttackRange) {
	attackRange = newAttackRange;
}
double HydrometeorologicSystem::getSpeed() {
	return speed;
}
void HydrometeorologicSystem::setSpeed(double newSpeed) {
	speed = newSpeed;
}
PointF HydrometeorologicSystem::getNowCoord() {
	return nowCoord;
}
void HydrometeorologicSystem::setNowCoord(PointF newNowCoord) {
	nowCoord = newNowCoord;
}
PointF HydrometeorologicSystem::getNowCoord4Draw() {
	return nowCoord4Draw;
}
void HydrometeorologicSystem::setNowCoord4Draw(PointF newNowCoord4Draw) {
	nowCoord4Draw = newNowCoord4Draw;
}
double HydrometeorologicSystem::getRotatePictureAngle() {
	return 0.0;
}
void HydrometeorologicSystem::setRotatePictureAngle(double newRotatePictureAngle) {
	return;
}
int HydrometeorologicSystem::getExistTime() {
	return existTime;
}
void HydrometeorologicSystem::setExistTime(int newExistTime) {
	existTime = newExistTime;
}
GraphicsPath^ HydrometeorologicSystem::getGp() {
	return gpHydro;
}
void HydrometeorologicSystem::setGp(GraphicsPath^ newGpHydro) {
	gpHydro = newGpHydro;
}
bool HydrometeorologicSystem::getMouseIn() {
	return mouseIn;
}
void HydrometeorologicSystem::setMouseIn(bool isMouseIn) {
	mouseIn = isMouseIn;
}
SizeF HydrometeorologicSystem::getSize() {
	return SizeF();
}
void HydrometeorologicSystem::setSize(SizeF newSize) {
	return;
}
SizeF HydrometeorologicSystem::getSize4Draw() {
	return SizeF();
}
void HydrometeorologicSystem::setSize4Draw(SizeF newSize4Draw) {
	return;
}
void HydrometeorologicSystem::hydroMove() {
	if (speed == 0.0)
		return;
	if (speed < 0) {
		speed = speed * (-1);
		angle += 180;
	}
	double dx, dy;
	if (BoundaryJudgment(getNowCoord().X, getNowCoord().Y)) {
		dx = Math::Cos(ANGLE_TO_RADIUS(angle));
		dy = -(Math::Sin(ANGLE_TO_RADIUS(angle)));
		double dis = Math::Sqrt((dx * dx) + (dy * dy));
		setNowCoord(PointF(speed * dx + getNowCoord().X, speed * dy + getNowCoord().Y));
		setNowCoord4Draw(COORD_TO_DRAWING_COORD(getNowCoord().X, getNowCoord().Y));
	}
	else {
		speed = 0.0;
		angle = 0.0;
```

```
            if (nowCoord.X < 0.0)
                nowCoord.X = 0.0;
            if (nowCoord.Y < 0.0)
                nowCoord.Y = 0.0;
            if (nowCoord.X > 20.0)
                nowCoord.X = 20.0;
            if (nowCoord.Y > 20.0)
                nowCoord.Y = 20.0;
            setNowCoord4Draw(COORD_TO_DRAWING_COORD(getNowCoord().X, getNowCoord().Y));
        }
        setRotatePictureAngle(getRotatePictureAngle() + ROTATE_ANGLE);
        return;
}
bool HydrometeorologicSystem::BoundaryJudgment(double _X, double _Y) {
        double end0 = -10.0;
        double end20 = 30.0;
        if (_X < end0 || _Y < end0 || _X > end20 || _Y > end20)
            return false;
        else
            return true;
}
bool HydrometeorologicSystem::render(Graphics^ g) {
        Pen^ penHydro = gcnew Pen(Color::Black);
        Brush^ bruHydro = gcnew SolidBrush(Color::FromArgb(100, 200, 200, 200));
        if (mouseIn && hydroType == FOG) {
            g->DrawString("Fog", gcnew Font("Consolas", 12), gcnew SolidBrush(Color::Black), PointF(nowCoord4Draw.X,
nowCoord4Draw.Y));
        }
        return true;
}
Size^ HydrometeorologicSystem::CalculateNewSize(int width, int height, double RotateAngle) {
        double r = Math::Sqrt(Math::Pow((double)width / 2.0, 2.0) + Math::Pow((double)height / 2.0, 2.0)); //半徑L
        double OriginalAngle = Math::Acos((width / 2.0) / r) / Math::PI * 180.0;   //對角線和X軸的角度θ
        double minW = 0.0, maxW = 0.0, minH = 0.0, maxH = 0.0; //最大和最小的 X、Y座標
        double drawPoint[4];
        drawPoint[0] = (-OriginalAngle + RotateAngle) * Math::PI / 180.0;
        drawPoint[1] = (OriginalAngle + RotateAngle) * Math::PI / 180.0;
        drawPoint[2] = (180.0 - OriginalAngle + RotateAngle) * Math::PI / 180.0;
        drawPoint[3] = (180.0 + OriginalAngle + RotateAngle) * Math::PI / 180.0;
        for (int i = 0; i < 4; i++) //由四個角的點算出X、Y的最大值及最小值
        {
                double x = r * Math::Cos(drawPoint[i]);
                double y = r * Math::Sin(drawPoint[i]);
                if (x < minW)
                        minW = x;
                if (x > maxW)
                        maxW = x;
                if (y < minH)
                        minH = y;
                if (y > maxH)
                        maxH = y;
        }
        return gcnew Size((int)(maxW - minW), (int)(maxH - minH));
}
Bitmap^ HydrometeorologicSystem::RotateImage(Bitmap^ image, float RotateAngle) {
        Size^ newSize = CalculateNewSize(image->Width, image->Height, RotateAngle);
        Bitmap^ rotatedBmp = gcnew Bitmap(newSize->Width, newSize->Height);
        PointF^ centerPoint = gcnew PointF((float)rotatedBmp->Width / 2.0, (float)rotatedBmp->Height / 2.0);
        Graphics^ g = Graphics::FromImage(rotatedBmp);
        g->CompositingQuality = CompositingQuality::HighQuality;
        g->SmoothingMode = SmoothingMode::HighQuality;
        g->InterpolationMode = InterpolationMode::HighQualityBicubic;
        g->TranslateTransform(centerPoint->X, centerPoint->Y);
        g->RotateTransform(RotateAngle);
        g->TranslateTransform(-centerPoint->X, -centerPoint->Y);
        g->DrawImage(image, (float)(newSize->Width - image->Width) / 2.0, (float)(newSize->Height - image->Height) / 2.0,
image->Width, image->Height);
        return rotatedBmp;
}
Lightning.h:
#pragma once
#include "HydrometeorologicSystem.h"
#define SPEED                   0.0
#define DAMAGE                  6.0
#define ATTACK_RANGE    1.0
ref class Lightning : public HydrometeorologicSystem {
```

```
public:
	Lightning();
	Lightning(Lightning^&);
	Lightning(PointF);
	Lightning^ operator=(Lightning^);
	virtual bool render(Graphics^) override;
	virtual double getRotatePictureAngle() override;
	virtual void setRotatePictureAngle(double) override;
private:
	double biasX4Pic = 2.5;
	double biasY4Pic = -(BLOCK_LENGTH + 5.5);
};
Lightning.cpp:
#include "Lightning.h"
Lightning::Lightning(): HydrometeorologicSystem() {
	picture = PICTURE_LIGHTNING;
	gpHydro = gcnew GraphicsPath();
}
Lightning::Lightning(Lightning^& rSide) {
	picture = rSide->picture;
	hydroType = rSide->hydroType;
	damage = rSide->damage;
	attackRange = rSide->attackRange;
	speed = rSide->speed;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	existTime = rSide->existTime;
	gpHydro = rSide->gpHydro;
}
Lightning::Lightning(PointF newNowCoord): HydrometeorologicSystem(0.0, newNowCoord) {
	hydroType = LIGHTNING;
	picture = PICTURE_LIGHTNING;
	damage = DAMAGE;
	attackRange = ATTACK_RANGE;
	speed = SPEED;
	gpHydro = gcnew GraphicsPath();
	gpHydro->AddRectangle(RectangleF(getNowCoord4Draw().X - picture->Width / 2.0 + biasX4Pic,
getNowCoord4Draw().Y - picture->Height / 2.0 + biasY4Pic, picture->Width, picture->Height));
}
Lightning^ Lightning::operator=(Lightning^ rSide) {
	picture = rSide->picture;
	hydroType = rSide->hydroType;
	damage = rSide->damage;
	attackRange = rSide->attackRange;
	speed = rSide->speed;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	existTime = rSide->existTime;
	gpHydro = rSide->gpHydro;
	return this;
}
double Lightning::getRotatePictureAngle() {
	return 0.0;
}
void Lightning::setRotatePictureAngle(double newRotatePictureAngle) {
	return;
}
bool Lightning::render(Graphics^ g) {
	Point ulCorner = Point(getNowCoord4Draw().X - picture->Width / 2.0 + biasX4Pic, getNowCoord4Draw().Y - picture-
>Height / 2.0 + biasY4Pic);
	g->DrawImage(picture, ulCorner);
	HydrometeorologicSystem::render(g);
	return true;
}
MainForm.h:
#include "VesselObject.h"
#include "Team.h"
#include "Command.h"
#include "Shell.h"
#include "Torpedo.h"
#include "GeographicSystem.h"
#include "HydrometeorologicSystem.h"
using namespace CommandAnalyze;
namespace Game {
```

```
using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Collections::Generic;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;
using namespace System::Media;
using namespace System::Text;
using namespace System::Threading::Tasks;
using namespace System::Windows::Forms;
public ref class MainForm : public System::Windows::Forms::Form
{
        System::Media::SoundPlayer^   player = gcnew System::Media::SoundPlayer();
public:
        MainForm(void){
                InitializeComponent();
        }
protected:
        ///  清除任何使用中的資源。
        ~MainForm(){
                if (components)
                        delete components;
        }
private: System::Windows::Forms::Button^   btnStart;
private: System::Windows::Forms::Button^   btnFire;
private: System::Windows::Forms::Button^   btnDefense;
private: System::Windows::Forms::Button^   btnMove;
private: System::Windows::Forms::Button^   btnTag;
private: System::Windows::Forms::Button^   btnLaunch;
private: System::Windows::Forms::Button^   btnLaser;
private: System::Windows::Forms::Button^   btnEnrich;
private: System::Windows::Forms::Button^   btnPause;
private: System::Windows::Forms::Timer^   timerMain;
private: System::Windows::Forms::GroupBox^   grpboxCommands;
private: System::Windows::Forms::TextBox^   txtCommand_A;
private: System::Windows::Forms::TextBox^   txtCommand_B;
private: System::Windows::Forms::Label^   lblNowTime;
```

```
	private: System::Windows::Forms::GroupBox^    grpboxBattleLog;
	private: System::Windows::Forms::TextBox^    txtBattleLog;
	private: System::Windows::Forms::Panel^    pnlGamePage;
	private: System::Windows::Forms::PictureBox^    picBoard;
	private: System::Windows::Forms::Label^    lblCoordinateHint;
	private: System::Windows::Forms::Label^    lblCoordinateHintVS;
	private: System::Windows::Forms::Panel^    pnlMenuPage;
	private: System::Windows::Forms::Button^    btnStartGame;
	private: System::Windows::Forms::Button^    btnExit;
	private: System::Windows::Forms::Button^    btnTyphoon;
	private: System::Windows::Forms::Button^    btnLightning;
	private: System::Windows::Forms::Button^    btnNewGeo;
	private: System::Windows::Forms::Label^    about;
	private: System::Windows::Forms::Button^    btnFog;
	void InitializeComponent(void) {
			this->components = (gcnew System::ComponentModel::Container());
			System::ComponentModel::ComponentResourceManager^    resources    =    (gcnew
System::ComponentModel::ComponentResourceManager(MainForm::typeid));
			this->btnStart = (gcnew System::Windows::Forms::Button());
			this->btnFire = (gcnew System::Windows::Forms::Button());
			this->btnDefense = (gcnew System::Windows::Forms::Button());
			this->btnMove = (gcnew System::Windows::Forms::Button());
			this->btnTag = (gcnew System::Windows::Forms::Button());
			this->btnLaunch = (gcnew System::Windows::Forms::Button());
			this->btnLaser = (gcnew System::Windows::Forms::Button());
			this->btnEnrich = (gcnew System::Windows::Forms::Button());
			this->btnPause = (gcnew System::Windows::Forms::Button());
			this->timerMain = (gcnew System::Windows::Forms::Timer(this->components));
			this->grpboxCommands = (gcnew System::Windows::Forms::GroupBox());
			this->txtCommand_B = (gcnew System::Windows::Forms::TextBox());
			this->txtCommand_A = (gcnew System::Windows::Forms::TextBox());
			this->lblNowTime = (gcnew System::Windows::Forms::Label());
			this->grpboxBattleLog = (gcnew System::Windows::Forms::GroupBox());
			this->txtBattleLog = (gcnew System::Windows::Forms::TextBox());
			// btnStart
			this->btnStart->Font    =    (gcnew    System::Drawing::Font(L"Consolas",    8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnStart->Location = System::Drawing::Point(640, 8);
			this->btnStart->Name = L"btnStart";
			this->btnStart->Size = System::Drawing::Size(205, 65);
			this->btnStart->TabIndex = 0;
			this->btnStart->Text = L"SET	[2D]	(Start)\n[Vessel_Name][Type]\n(CV BB CG DD LV AH)";
			this->btnStart->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnStart->UseVisualStyleBackColor = true;
			this->btnStart->Click += gcnew System::EventHandler(this, &MainForm::btnStart_Click);
			// btnLaser
			this->btnLaser->Font = (gcnew System::Drawing::Font(L"Consolas", 8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnLaser->Location = System::Drawing::Point(640, 78);
			this->btnLaser->Name = L"btnLaser";
			this->btnLaser->Size = System::Drawing::Size(205, 65);
			this->btnLaser->TabIndex = 0;
			this->btnLaser->Text = L"LASER	[Vessel_Name]\n[Angle]\n(LV)";
			this->btnLaser->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnLaser->UseVisualStyleBackColor = true;
			this->btnLaser->Click += gcnew System::EventHandler(this, &MainForm::btnLaser_Click);
			// btnFire
			this->btnFire->Font = (gcnew System::Drawing::Font(L"Consolas", 8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnFire->Location = System::Drawing::Point(850, 8);
			this->btnFire->Name = L"btnFire";
			this->btnFire->Size = System::Drawing::Size(160, 65);
			this->btnFire->TabIndex = 0;
			this->btnFire->Text = L"FIRE	[Vessel]\n[2D]";
			this->btnFire->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnFire->UseVisualStyleBackColor = true;
			this->btnFire->Click += gcnew System::EventHandler(this, &MainForm::btnFire_Click);
			// btnLaunch
			this->btnLaunch->Font = (gcnew System::Drawing::Font(L"Consolas", 8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnLaunch->Location = System::Drawing::Point(850, 78);
			this->btnLaunch->Name = L"btnLaunch";
			this->btnLaunch->Size = System::Drawing::Size(160, 65);
			this->btnLaunch->TabIndex = 0;
			this->btnLaunch->Text = L"LAUNCH      [Vessel]\n[2D]\n(CG and DD)";
			this->btnLaunch->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnLaunch->UseVisualStyleBackColor = true;
			this->btnLaunch->Click += gcnew System::EventHandler(this, &MainForm::btnLaunch_Click);
			// btnDefense
			this->btnDefense->Font = (gcnew System::Drawing::Font(L"Consolas", 8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnDefense->Location = System::Drawing::Point(1015, 8);
			this->btnDefense->Name = L"btnDefense";
			this->btnDefense->Size = System::Drawing::Size(160, 65);
			this->btnDefense->TabIndex = 0;
			this->btnDefense->Text = L"DEFENSE    [Vessel]\n[Shell_Name]";
			this->btnDefense->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnDefense->UseVisualStyleBackColor = true;
			this->btnDefense->Click += gcnew System::EventHandler(this, &MainForm::btnDefense_Click);
			// btnTag
			this->btnTag->Font = (gcnew System::Drawing::Font(L"Consolas", 8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnTag->Location = System::Drawing::Point(1015, 78);
			this->btnTag->Name = L"btnTag";
			this->btnTag->Size = System::Drawing::Size(160, 65);
			this->btnTag->TabIndex = 0;
			this->btnTag->Text = L"TAG         [Vessel]\n[New_Name]";
			this->btnTag->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnTag->UseVisualStyleBackColor = true;
			this->btnTag->Click += gcnew System::EventHandler(this, &MainForm::btnTag_Click);
			// btnMove
			this->btnMove->Font = (gcnew System::Drawing::Font(L"Consolas", 8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnMove->Location = System::Drawing::Point(1180, 8);
			this->btnMove->Name = L"btnMove";
			this->btnMove->Size = System::Drawing::Size(160, 65);
			this->btnMove->TabIndex = 0;
			this->btnMove->Text = L"MOVE	[Vessel]\n[Speed][Angle]";
			this->btnMove->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnMove->UseVisualStyleBackColor = true;
			this->btnMove->Click += gcnew System::EventHandler(this, &MainForm::btnMove_Click);
			// btnEnrich
			this->btnEnrich->Font = (gcnew System::Drawing::Font(L"Consolas", 8.2,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnEnrich->Location = System::Drawing::Point(1180, 78);
			this->btnEnrich->Name = L"btnEnrich";
			this->btnEnrich->Size = System::Drawing::Size(160, 65);
			this->btnEnrich->TabIndex = 0;
			this->btnEnrich->Text = L"ENRICH	[helper]\n[Vessel_Need]\n(AH)";
			this->btnEnrich->TextAlign = System::Drawing::ContentAlignment::TopLeft;
			this->btnEnrich->UseVisualStyleBackColor = true;
			this->btnEnrich->Click += gcnew System::EventHandler(this, &MainForm::btnEnrich_Click);
			// btnPause
			this->btnPause->Enabled = false;
			this->btnPause->Font = (gcnew System::Drawing::Font(L"Consolas", 12,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->btnPause->Location = System::Drawing::Point(563, 80);
			this->btnPause->Name = L"btnPause";
			this->btnPause->Size = System::Drawing::Size(75, 55);
			this->btnPause->TabIndex = 1;
			this->btnPause->Text = L"Stop";
			this->btnPause->UseVisualStyleBackColor = true;
			this->btnPause->Click += gcnew System::EventHandler(this, &MainForm::btnPause_Click);
			// timerMain
			this->timerMain->Interval = 66;
			this->timerMain->Tick += gcnew System::EventHandler(this, &MainForm::timerMain_Tick);
			// grpboxCommands
			this->grpboxCommands->Controls->Add(this->txtCommand_B);
			this->grpboxCommands->Controls->Add(this->txtCommand_A);
			this->grpboxCommands->Font = (gcnew System::Drawing::Font(L"Consolas", 9,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->grpboxCommands->Location = System::Drawing::Point(563, 142);
			this->grpboxCommands->Name = L"grpboxCommands";
			this->grpboxCommands->Size = System::Drawing::Size(817, 425);
			this->grpboxCommands->TabIndex = 2;
			this->grpboxCommands->TabStop = false;
			this->grpboxCommands->Text = L"Commands";
			// txtCommand_B
			this->txtCommand_B->ForeColor = System::Drawing::Color::Blue;
			this->txtCommand_B->Location = System::Drawing::Point(415, 24);
			this->txtCommand_B->Multiline = true;
			this->txtCommand_B->Name = L"txtCommand_B";
			this->txtCommand_B->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;
			this->txtCommand_B->Size = System::Drawing::Size(385, 384);
			this->txtCommand_B->TabIndex = 1;
			// txtCommand_A
			this->txtCommand_A->ForeColor = System::Drawing::Color::Red;
			this->txtCommand_A->Location = System::Drawing::Point(15, 24);
			this->txtCommand_A->Multiline = true;
			this->txtCommand_A->Name = L"txtCommand_A";
			this->txtCommand_A->ScrollBars = System::Windows::Forms::ScrollBars::Vertical;
			this->txtCommand_A->Size = System::Drawing::Size(385, 384);
			this->txtCommand_A->TabIndex = 0;
			// lblNowTime
			this->lblNowTime->AutoSize = true;
			this->lblNowTime->Font = (gcnew System::Drawing::Font(L"Consolas", 19.8F,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,
				static_cast<System::Byte>(0)));
			this->lblNowTime->Location = System::Drawing::Point(5, 524);
			this->lblNowTime->Name = L"lblNowTime";
			this->lblNowTime->Size = System::Drawing::Size(107, 38);
			this->lblNowTime->TabIndex = 3;
			this->lblNowTime->Text = L"MM:SS";
			// grpboxBattleLog
			this->grpboxBattleLog->Controls->Add(this->txtBattleLog);
			this->grpboxBattleLog->Controls->Add(this->lblMsg4AddingNewGeo);
			this->grpboxBattleLog->Font = (gcnew System::Drawing::Font(L"Consolas", 9,
System::Drawing::FontStyle::Regular, System::Drawing::GraphicsUnit::Point,static_cast<System::Byte>(0)));
			this->grpboxBattleLog->Location = System::Drawing::Point(12, 573);
			this->grpboxBattleLog->Name = L"grpboxBattleLog";
```

```
this->grpboxBattleLog->Size = System::Drawing::Size(1368, 257);
this->grpboxBattleLog->TabIndex = 4;
this->grpboxBattleLog->TabStop = false;
this->grpboxBattleLog->Text = L"Battle Log";
// btnStart : click
System::Void btnStart_Click(System::Object^   sender, System::EventArgs^   e) {
try {
        txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
        btnStart->Enabled = false; btnFire->Enabled = false; btnDefense->Enabled = false; btnMove-
>Enabled = false;
        btnTag->Enabled = false; btnLaunch->Enabled = false; btnLaser->Enabled = false; btnEnrich-
>Enabled = false;
        btnPause->Enabled = true; btnNextSecond->Enabled = false;
        btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
        btnNewGeo->Enabled = false;
        array<String^>^ cmdList;
        // analyze commands of team A
        cmdList = txtCommand_A->Text->Split('\n');
        for each (String^ cmdString in cmdList) {
                if (cmdString != "") {
                        cmdString = "SET " + cmdString+ " " + lblCoordinateHint->Text;
                        if (cmdString == "")
                                continue;
                txtBattleLog->Text += processCommand(minute, second, cmdString, team_A, team_B);
                }

        }
        // analyze commands of team B
        cmdList = txtCommand_B->Text->Split('\n');

        for each (String^ cmdString in cmdList) {
                if (cmdString != "") {
                        cmdString = "SET " + cmdString+ " " + lblCoordinateHint->Text;


                        if (cmdString == "")
                                continue;


                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_B,
team_A);
                    }

                }
                txtCommand_A->Text = ""; txtCommand_B->Text = "";
                timerMain->Enabled = true;
            }
            catch (...) {}
        }
        // btnTag : click
        System::Void btnTag_Click(System::Object^   sender, System::EventArgs^   e) {
            try {
                txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
                btnStart->Enabled = false; btnFire->Enabled = false; btnDefense->Enabled = false; btnMove-
>Enabled = false;
                btnTag->Enabled = false; btnLaunch->Enabled = false; btnLaser->Enabled = false; btnEnrich-
>Enabled = false;
                btnPause->Enabled = true; btnNextSecond->Enabled = false;
                btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
                btnNewGeo->Enabled = false;
                array<String^>^ cmdList;
                // analyze commands of team A
                cmdList = txtCommand_A->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "TAG " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_A,
team_B);
                    }
                }
                // analyze commands of team B
                cmdList = txtCommand_B->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "TAG " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_B,
team_A);
                    }
                }
                txtCommand_A->Text = ""; txtCommand_B->Text = "";
                timerMain->Enabled = true;
            }
            catch (...) {}
        }
        // btnLaunch : click
        System::Void btnLaunch_Click(System::Object^   sender, System::EventArgs^   e) {
            try {
                txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
                btnStart->Enabled = false; btnFire->Enabled = false; btnDefense->Enabled = false; btnMove-
>Enabled = false;
                btnTag->Enabled = false; btnLaunch->Enabled = false; btnLaser->Enabled = false; btnEnrich-
>Enabled = false;
                btnPause->Enabled = true; btnNextSecond->Enabled = false;
                btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
                btnNewGeo->Enabled = false;
                array<String^>^ cmdList;
                // analyze commands of team A
                cmdList = txtCommand_A->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "LAUNCH " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_A,
team_B);
                    }
                }
                // analyze commands of team B
                cmdList = txtCommand_B->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "LAUNCH " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_B,
team_A);
                    }
                }
                player->SoundLocation = "launch.wav";
                player->LoadAsync();
                player->PlaySync();
                txtCommand_A->Text = ""; txtCommand_B->Text = "";
                timerMain->Enabled = true;
            }
            catch (...) {}
        }
        // btnLaser : click
        System::Void btnLaser_Click(System::Object^  sender, System::EventArgs^  e) {
            try {
                txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
                btnStart->Enabled = false; btnFire->Enabled = false; btnDefense->Enabled = false; btnMove-
>Enabled = false;
                btnTag->Enabled = false; btnLaunch->Enabled = false; btnLaser->Enabled = false; btnEnrich-
>Enabled = false;
                btnPause->Enabled = true; btnNextSecond->Enabled = false;
                btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
                btnNewGeo->Enabled = false;
                array<String^>^ cmdList;
                // analyze commands of team A
                cmdList = txtCommand_A->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "LASER " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_A,
team_B);
                    }
                }
                // analyze commands of team B
```

```
                cmdList = txtCommand_B->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "LASER " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text  +=  processCommand(minute,  second,  cmdString,  team_B,
team_A);
                    }
                }
                player->SoundLocation = "laser.wav";
                player->LoadAsync();
                player->PlaySync();
                txtCommand_A->Text = ""; txtCommand_B->Text = "";
                timerMain->Enabled = true;
            }
            catch (...) {}
        }
        // btnEnrich : click
        System::Void btnEnrich_Click(System::Object^    sender, System::EventArgs^    e) {
            try {
                txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
                btnStart->Enabled  =  false;  btnFire->Enabled  =  false;  btnDefense->Enabled  =  false;  btnMove-
>Enabled = false;
                btnTag->Enabled  =  false;  btnLaunch->Enabled  =  false;  btnLaser->Enabled  =  false;  btnEnrich-
>Enabled = false;
                btnPause->Enabled = true; btnNextSecond->Enabled = false;
                btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
                btnNewGeo->Enabled = false;
                array<String^>^ cmdList;
                // analyze commands of team A
                cmdList = txtCommand_A->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "ENRICH " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text  +=  processCommand(minute,  second,  cmdString,  team_A,
team_B);
                    }
                }
                // analyze commands of team B
                cmdList = txtCommand_B->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "ENRICH " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_B,
team_A);
                    }
                }
                txtCommand_A->Text = ""; txtCommand_B->Text = "";
                timerMain->Enabled = true;
            }
            catch (...) {}
        }
        // btnFire : click
        System::Void btnFire_Click(System::Object^  sender, System::EventArgs^  e) {
            try {
                txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
                btnStart->Enabled = false; btnFire->Enabled = false; btnDefense->Enabled = false; btnMove-
>Enabled = false;
                btnTag->Enabled = false; btnLaunch->Enabled = false; btnLaser->Enabled = false; btnEnrich-
>Enabled = false;
                btnPause->Enabled = true; btnNextSecond->Enabled = false;
                btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
                btnNewGeo->Enabled = false;
                array<String^>^ cmdList;
                // analyze commands of team A
                cmdList = txtCommand_A->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "FIRE " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_A,
team_B);
                    }
                }
                // analyze commands of team B
                cmdList = txtCommand_B->Text->Split('\n');
                for each (String^ cmdString in cmdList) {
                    if (cmdString != "") {
                        cmdString = "FIRE " + lblCoordinateHintVS->Text + " " + cmdString;
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_B,
team_A);
                    }

                }
                player->SoundLocation = "fire.wav";
                player->LoadAsync();
                player->PlaySync();
                txtCommand_A->Text = ""; txtCommand_B->Text = "";
                timerMain->Enabled = true;
            }
            catch (...) {}
        }
        // btnDefense : click
        System::Void btnDefense_Click(System::Object^    sender, System::EventArgs^    e) {
            try {
                txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
                btnStart->Enabled = false; btnFire->Enabled = false; btnDefense->Enabled = false; btnMove-
>Enabled = false;
                btnTag->Enabled = false; btnLaunch->Enabled = false; btnLaser->Enabled = false; btnEnrich-
>Enabled = false;
                btnPause->Enabled = true; btnNextSecond->Enabled = false;
                btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
                btnNewGeo->Enabled = false;
                array<String^>^ cmdList;
                // analyze commands of team A
                cmdList = txtCommand_A->Text->Split('\n');
```

```
                        for each (String^ cmdString in cmdList) {
                            if (cmdString != "") {
                                cmdString = "DEFENSE " + lblCoordinateHintVS->Text + " " + cmdString;
                                if (cmdString == "")
                                    continue;
                                txtBattleLog->Text  +=  processCommand(minute,  second,  cmdString,  team_A,
team_B);
                            }
                        }
                        // analyze commands of team B
                        cmdList = txtCommand_B->Text->Split('\n');
                        for each (String^ cmdString in cmdList) {
                            if (cmdString != "") {
                                cmdString = "DEFENSE " + lblCoordinateHintVS->Text + " " + cmdString;
                                if (cmdString == "")
                                    continue;
                                txtBattleLog->Text  +=  processCommand(minute,  second,  cmdString,  team_B,
team_A);
                            }
                            txtCommand_A->Text = ""; txtCommand_B->Text = "";
                        }

                        timerMain->Enabled = true;
                    }
                    catch (...) {}
                }
                // btnMove : click
                System::Void btnMove_Click(System::Object^    sender, System::EventArgs^    e) {
                    try {
                        txtCommand_A->Enabled = false; txtCommand_B->Enabled = false;
                        btnStart->Enabled  =  false;  btnFire->Enabled  =  false;  btnDefense->Enabled  =  false;  btnMove-
>Enabled = false;
                        btnTag->Enabled  =  false;  btnLaunch->Enabled  =  false;  btnLaser->Enabled  =  false;  btnEnrich-
>Enabled = false;
                        btnPause->Enabled = true; btnNextSecond->Enabled = false;
                        btnTyphoon->Enabled = true; btnLightning->Enabled = true; btnFog->Enabled = true;
                        btnNewGeo->Enabled = false;
                        array<String^>^ cmdList;
                        // analyze commands of team A
                        cmdList = txtCommand_A->Text->Split('\n');
                        for each (String^ cmdString in cmdList) {
                            if (cmdString != "") {
                                cmdString = "MOVE " + lblCoordinateHintVS->Text + " " + cmdString;
                                if (cmdString == "")
                                    continue;
                                txtBattleLog->Text += processCommand(minute, second, cmdString, team_A,
team_B);
                            }
                        }
                        // analyze commands of team B
                        cmdList = txtCommand_B->Text->Split('\n');
                        for each (String^ cmdString in cmdList) {
                            if (cmdString != "") {
                                cmdString = "MOVE " + lblCoordinateHintVS->Text + " " + cmdString;
                                if (cmdString == "")
                                    continue;
                                txtBattleLog->Text += processCommand(minute, second, cmdString, team_B,
team_A);
                            }
                        }
                        txtCommand_A->Text = ""; txtCommand_B->Text = "";
                        timerMain->Enabled = true;
                    }
                    catch (...) {}
                }
                // btnPause : click
                System::Void btnPause_Click(System::Object^    sender, System::EventArgs^    e) {
                    try {
                        txtCommand_A->Enabled = true; txtCommand_B->Enabled = true; btnNextSecond->Enabled =
true;
                        btnStart->Enabled = true; btnFire->Enabled = true; btnDefense->Enabled = true; btnMove->Enabled
= true;
                        btnTag->Enabled = true; btnLaunch->Enabled = true; btnLaser->Enabled = true; btnEnrich-
>Enabled = true;
                        btnPause->Enabled = false;
                        timerMain->Enabled = false;
```

```
                        btnTyphoon->Enabled = false; btnLightning->Enabled = false; btnFog->Enabled = false;
                        btnNewGeo->Enabled = true;
                    }
                    catch (...) {}
                }
                // btnNextSecond : click
                System::Void btnNextSecond_Click(System::Object^    sender, System::EventArgs^    e) {
                    array<String^>^ cmdList;
                    // analyze commands of team A
                    cmdList = txtCommand_A->Text->Split('\n');
                    for each (String^ cmdString in cmdList) {
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_A, team_B);
                    }
                    // analyze commands of team B
                    cmdList = txtCommand_B->Text->Split('\n');
                    for each (String^ cmdString in cmdList) {
                        if (cmdString == "")
                            continue;
                        txtBattleLog->Text += processCommand(minute, second, cmdString, team_B, team_A);
                    }
                    player->SoundLocation = "ding.wav";
                    player->LoadAsync();
                    player->PlaySync();
                    txtCommand_A->Text = ""; txtCommand_B->Text = "";
                    timerMain_Tick(gcnew Object(), gcnew EventArgs());
                }
```

MainForm.cpp:

```
#include "MainForm.h"
using namespace System;
using namespace System::Windows::Forms;
[STAThreadAttribute]
void Main(array<String^>^ args) {
    Application::EnableVisualStyles();
    Application::SetCompatibleTextRenderingDefault(false);
    Game::MainForm form;
    Application::Run(%form);
```

```
}
Mountain.h:
#pragma once
#include "GeographicSystem.h"
ref class Mountain : public GeographicSystem {
public:
	Mountain();
	Mountain(Mountain^&);
	Mountain(PointF, SizeF);
	Mountain^ operator=(Mountain^);
	virtual bool render(Graphics^) override;
};
Mountain.cpp:
#include "Mountain.h"
Mountain::Mountain() : GeographicSystem() {}
Mountain::Mountain(Mountain^& rSide) {
	geoType = rSide->geoType;
	name = rSide->name;
	gpGeography = rSide->gpGeography;
	mouseIn = false;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	colorAlpha = rSide->colorAlpha;
	reefDamage = rSide->reefDamage;
	canGoThrough_vessel = rSide->canGoThrough_vessel;
	canGoThrough_generalFire = rSide->canGoThrough_generalFire;
	canGoThrough_torpedo = rSide->canGoThrough_torpedo;
	canGoThrough_laser = rSide->canGoThrough_laser;
}
Mountain::Mountain(PointF newCoord, SizeF newSize) : GeographicSystem(MOUNTAIN, newCoord, newSize) {
	canGoThrough_vessel = false;
	canGoThrough_generalFire = false;
	canGoThrough_torpedo = false;
	canGoThrough_laser = true;
}
Mountain^ Mountain::operator=(Mountain^ rSide) {
	geoType = rSide->geoType;
	name = rSide->name;
	gpGeography = rSide->gpGeography;
	mouseIn = false;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	colorAlpha = rSide->colorAlpha;
	reefDamage = rSide->reefDamage;
	canGoThrough_vessel = rSide->canGoThrough_vessel;
	canGoThrough_generalFire = rSide->canGoThrough_generalFire;
	canGoThrough_torpedo = rSide->canGoThrough_torpedo;
	canGoThrough_laser = rSide->canGoThrough_laser;
	return this;
}
bool Mountain::render(Graphics^ g) {
	Brush^ bruGeo = gcnew SolidBrush(Color::FromArgb(colorAlpha, 45, 45, 245));
	g->FillPath(bruGeo, gpGeography);
	GeographicSystem::render(g);
	return true;
}
Reef.h:
#pragma once
#include "GeographicSystem.h"
ref class Reef : public GeographicSystem {
public:
	Reef();
	Reef(Reef^&);
	Reef(PointF, SizeF);
	Reef^ operator=(Reef^);
	virtual bool render(Graphics^) override;
};
Reef.cpp:
#include "Reef.h"
Reef::Reef() : GeographicSystem() {
	reefDamage = REEF_DAMAGE;
}
```

```
Reef::Reef(Reef^& rSide) {
	geoType = rSide->geoType;
	name = rSide->name;
	gpGeography = rSide->gpGeography;
	mouseIn = false;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	colorAlpha = rSide->colorAlpha;
	reefDamage = rSide->reefDamage;
	canGoThrough_vessel = rSide->canGoThrough_vessel;
	canGoThrough_generalFire = rSide->canGoThrough_generalFire;
	canGoThrough_torpedo = rSide->canGoThrough_torpedo;
	canGoThrough_laser = rSide->canGoThrough_laser;
}
Reef::Reef(PointF newCoord, SizeF newSize) : GeographicSystem(REEF, newCoord, newSize) {
	reefDamage = REEF_DAMAGE;
	canGoThrough_vessel = false;
	canGoThrough_generalFire = true;
	canGoThrough_torpedo = false;
	canGoThrough_laser = true;
}
Reef^ Reef::operator=(Reef^ rSide) {
	geoType = rSide->geoType;
	name = rSide->name;
	gpGeography = rSide->gpGeography;
	mouseIn = false;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	size = rSide->size;
	size4Draw = rSide->size4Draw;
	colorAlpha = rSide->colorAlpha;
	reefDamage = rSide->reefDamage;
	canGoThrough_vessel = rSide->canGoThrough_vessel;
	canGoThrough_generalFire = rSide->canGoThrough_generalFire;
	canGoThrough_torpedo = rSide->canGoThrough_torpedo;
	canGoThrough_laser = rSide->canGoThrough_laser;
	return this;
}
//draw reef
bool Reef::render(Graphics^ g) {
	Brush^ bruGeo = gcnew SolidBrush(Color::FromArgb(colorAlpha, 100, 100, 100));
	g->FillPath(bruGeo, gpGeography);
	GeographicSystem::render(g);
	return true;
}
Shell.h:
#pragma once
#include "RenderParameter.h"
#define EXPLOSION_RADIUS 1.5
#define PICTURE dynamic_cast<Bitmap^>(Bitmap::FromFile("picture/shell.png"))
#define INF 2147483647.0
using namespace System;
using namespace System::Windows;
using namespace System::Drawing;
using namespace System::Drawing::Drawing2D;
using namespace System::Collections::Generic;
using namespace System::Diagnostics;
ref class Shell {
public:
	Shell();
	Shell(Shell^&);
	Shell(String^, TeamSymbol, double, double, double, PointF, PointF);
	Shell^ operator=(Shell^);
	Bitmap^ getPicture();
	void setPicture(Bitmap^);
	String^ getName();
	void setName(String^);
	TeamSymbol getTeam();
	void setTeam(TeamSymbol);
	double getSpeed();
	void setSpeed(double);
	double getDamage();
	void setDamage(double);
	double getAngle();
	void setAngle(double);
	PointF getNowCoord();
	void setNowCoord(PointF);
	PointF getNowCoord4Draw();
	void setNowCoord4Draw(PointF);
	PointF getGoalCoord();
	void setGoalCoord(PointF);
	double getTotalFlyingTime();
	void setTotalFlyingTime(double);
	int getNowFlyingTime();
	void setNowFlyingTime(int);
	void ShellMove();
	bool render(Graphics^);
	// rotate the image
	Bitmap^ RotateImage(Bitmap^, float);
	// helper function: calculate new size
	Size^ CalculateNewSize(int, int, double);
private:
	Bitmap^ picture;
	String^ name;
	TeamSymbol team;
	double speed;
	double damage;
	double angle;
	PointF nowCoord;
	PointF nowCoord4Draw;
	PointF goalCoord;
	double totalFlyingTime;
	int nowFlyingTime;
};
Shell.cpp:
#include "Shell.h"
Shell::Shell() {
	picture = PICTURE;
	name = "Unknown";
	team = NONE;
	speed = damage = angle = 0.0;
	nowCoord = nowCoord4Draw = goalCoord = PointF();
	totalFlyingTime = 0.0;
	nowFlyingTime = 0;
}
Shell::Shell(Shell^& rSide) {
	picture = rSide->picture;
	name = rSide->name;
	team = rSide->team;
	speed = rSide->speed;
	damage = rSide->damage;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	goalCoord = rSide->goalCoord;
	totalFlyingTime = rSide->totalFlyingTime;
	nowFlyingTime = rSide->nowFlyingTime;
}
Shell::Shell(String^ newName, TeamSymbol whichTeam, double newSpeed, double newDamage, double newAngle, PointF 
beginCoord, PointF endCoord) {
	picture = PICTURE;
	name = newName;
	team = whichTeam;
	speed = newSpeed;
	damage = newDamage;
	angle = newAngle;
	nowCoord = beginCoord;
	nowCoord4Draw = COORD_TO_DRAWING_COORD(nowCoord.X, nowCoord.Y);
	goalCoord = endCoord;
	if (speed != 0.0)
		totalFlyingTime = Math::Sqrt( Math::Abs(((nowCoord.X - goalCoord.X) * (nowCoord.X - goalCoord.X)) + 
((nowCoord.Y - goalCoord.Y) * (nowCoord.Y - goalCoord.Y))) ) / speed;
	else
		totalFlyingTime = INF;
	nowFlyingTime = 0;
}
Shell^ Shell::operator=(Shell^ rSide) {
	picture = rSide->picture;
	name = rSide->name;
	team = rSide->team;
	speed = rSide->speed;
	damage = rSide->damage;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	goalCoord = rSide->goalCoord;
	totalFlyingTime = rSide->totalFlyingTime;
	nowFlyingTime = rSide->nowFlyingTime;
	return this;
}
Bitmap^ Shell::getPicture() {
	return picture;
}
void Shell::setPicture(Bitmap^ newPicture) {
	picture = newPicture;
}
String^ Shell::getName() {
	return name;
}
void Shell::setName(String^ newName) {
	name = newName;
}
TeamSymbol Shell::getTeam() {
	return team;
}
void Shell::setTeam(TeamSymbol whichTeam) {
	team = whichTeam;
}
double Shell::getSpeed() {
	return speed;
}
void Shell::setSpeed(double newSpeed) {
	speed = newSpeed;
}
double Shell::getDamage() {
	return damage;
}
void Shell::setDamage(double newDamage) {
	damage = newDamage;
}
double Shell::getAngle() {
	return angle;
}
void Shell::setAngle(double newAngle) {
	angle = newAngle;
}
PointF Shell::getNowCoord() {
	return nowCoord;
}
void Shell::setNowCoord(PointF newNowCoord) {
	nowCoord = newNowCoord;
}
PointF Shell::getNowCoord4Draw() {
	return nowCoord4Draw;
}
void Shell::setNowCoord4Draw(PointF newNowCoord4Draw) {
	nowCoord4Draw = newNowCoord4Draw;
}
PointF Shell::getGoalCoord() {
	return goalCoord;
}
void Shell::setGoalCoord(PointF newGoalCoord) {
	goalCoord = newGoalCoord;
}
double Shell::getTotalFlyingTime() {
	return totalFlyingTime;
}
void Shell::setTotalFlyingTime(double newTotalFlyingTime) {
	totalFlyingTime = newTotalFlyingTime;
}
int Shell::getNowFlyingTime() {
	return nowFlyingTime;
}
void Shell::setNowFlyingTime(int newNowFlyingTime) {
	nowFlyingTime = newNowFlyingTime;
}
```

```
void Shell::ShellMove() {
	double dx = (getGoalCoord().X - getNowCoord().X);
	double dy = (getGoalCoord().Y - getNowCoord().Y);
	double dis = Math::Sqrt((dx * dx) + (dy * dy));
	setNowCoord(PointF(getNowCoord().X + getSpeed() * (dx / dis), getNowCoord().Y + getSpeed() * (dy / dis)));
	setNowCoord4Draw(COORD_TO_DRAWING_COORD(getNowCoord().X, getNowCoord().Y));
	setNowFlyingTime(getNowFlyingTime() + 1);
}
bool Shell::render(Graphics^ g) {
	Bitmap^ bitmapRotate;
	bitmapRotate = RotateImage(picture, angle + 180);
	// Create Point of center.
	Point ulCorner = Point(getNowCoord4Draw().X - bitmapRotate->Width / 2.0, getNowCoord4Draw().Y - bitmapRotate-
>Height / 2.0);
	// Draw image to screen.
	g->DrawImage(bitmapRotate, ulCorner);
	//draw name
	g->DrawString(name, gcnew Font("Consolas", 8), gcnew SolidBrush(team == A ? Color::Red : Color::Blue),
PointF(nowCoord4Draw.X + BIAS_X_4_DRAW_NAME, nowCoord4Draw.Y - BIAS_Y_4_DRAW_NAME));
	return true;
}
Size^ Shell::CalculateNewSize(int width, int height, double RotateAngle) {
	double r = Math::Sqrt(Math::Pow((double)width / 2.0, 2.0) + Math::Pow((double)height / 2.0, 2.0)); //半徑L
	double OriginalAngle = Math::Acos((width / 2.0) / r) / Math::PI * 180.0;   //對角線和X軸的角度θ
	double minW = 0.0, maxW = 0.0, minH = 0.0, maxH = 0.0; //最大和最小的 X、Y座標
	double drawPoint[4];
	drawPoint[0] = (-OriginalAngle + RotateAngle) * Math::PI / 180.0;
	drawPoint[1] = (OriginalAngle + RotateAngle) * Math::PI / 180.0;
	drawPoint[2] = (180.0 - OriginalAngle + RotateAngle) * Math::PI / 180.0;
	drawPoint[3] = (180.0 + OriginalAngle + RotateAngle) * Math::PI / 180.0;
	for (int i = 0; i < 4; i++){ //由四個角的點算出X、Y的最大值及最小值
		double x = r * Math::Cos(drawPoint[i]);
		double y = r * Math::Sin(drawPoint[i]);
		if (x < minW)
			minW = x;
		if (x > maxW)
			maxW = x;
		if (y < minH)
```

```
				minH = y;
			if (y > maxH)
				maxH = y;
		}
		return gcnew Size((int)(maxW - minW), (int)(maxH - minH));
}
Bitmap^ Shell::RotateImage(Bitmap^ image, float RotateAngle) {
		Size^ newSize = CalculateNewSize(image->Width, image->Height, RotateAngle);
		Bitmap^ rotatedBmp = gcnew Bitmap(newSize->Width, newSize->Height);
		PointF^ centerPoint = gcnew PointF((float)rotatedBmp->Width / 2.0, (float)rotatedBmp->Height / 2.0);
		Graphics^ g = Graphics::FromImage(rotatedBmp);
		g->CompositingQuality = CompositingQuality::HighQuality;
		g->SmoothingMode = SmoothingMode::HighQuality;
		g->InterpolationMode = InterpolationMode::HighQualityBicubic;
		g->TranslateTransform(centerPoint->X, centerPoint->Y);
		g->RotateTransform(RotateAngle);
		g->TranslateTransform(-centerPoint->X, -centerPoint->Y);
		g->DrawImage(image, (float)(newSize->Width - image->Width) / 2.0, (float)(newSize->Height - image->Height) / 2.0,
image->Width, image->Height);
		return rotatedBmp;
}
Team.h:
#pragma once
#include "VesselObject.h"
#include "Shell.h"
#include "Torpedo.h"
using namespace System;
using namespace System::Windows;
using namespace System::Drawing;
using namespace System::Collections::Generic;
using namespace System::Diagnostics;
ref class Team {
public:
		Team();
		Team(TeamSymbol);
		// vessel list
		Dictionary<String^, VesselObject^> vesselList;
		// shell list
```

```
	List<Shell^> shellList;
	// Torpedo list
	List<Torpedo^> torpedoList;
	#pragma region getters and setters
	// shell
	int getLauchedShellNum();
	void setLauchedShellNum(int);
	void plus1LauchShellNum();
	// torpedo
	int getLaunchedTorpedoNum();
	void setLaunchedTorpedoNum(int);
	void plus1LaunchTorpedoNum();
	TeamSymbol getSymbol();
	void setSymbol(TeamSymbol);
	#pragma endregion
private:
	int lauchedShellNum;
	int launchedTorpedoNum;
	TeamSymbol symbol;
};
Team.cpp:
#include "Team.h"
Team::Team() {
	vesselList.Clear();
	shellList.Clear();
	torpedoList.Clear();
	lauchedShellNum = 0;
	launchedTorpedoNum = 0;
	symbol = NONE;
}
Team::Team(TeamSymbol newSybol): Team() {
	symbol = newSybol;
}
int Team::getLauchedShellNum() {
	return lauchedShellNum;
}
void Team::setLauchedShellNum(int newLauchedShellNum) {
	lauchedShellNum = newLauchedShellNum;
```

```
}
void Team::plus1LauchShellNum() {
	lauchedShellNum++;
}
int Team::getLaunchedTorpedoNum() {
	return launchedTorpedoNum;
}
void Team::setLaunchedTorpedoNum(int newLaunchedTorpedoNum) {
	launchedTorpedoNum = newLaunchedTorpedoNum;
}
void Team::plus1LaunchTorpedoNum() {
	launchedTorpedoNum++;
}
TeamSymbol Team::getSymbol() {
	return symbol;
}
void Team::setSymbol(TeamSymbol newTeamSymbol) {
	symbol = newTeamSymbol;
}
```

Torpedo.h:

```
#pragma once
#include "RenderParameter.h"
using namespace System;
using namespace System::Windows;
using namespace System::Drawing;
using namespace System::Drawing::Drawing2D;
using namespace System::Collections::Generic;
using namespace System::Diagnostics;
#define TORPEDO_EXPLODE_RANGE 0.3
#define PICTURE dynamic_cast<Bitmap^>(Bitmap::FromFile("picture/torpedo.png"))
#define INF 2147483647.0
ref class Torpedo {
public:
	Torpedo();
	Torpedo(Torpedo^&);
	Torpedo(String^, String^, TeamSymbol, double, double, double, PointF, PointF);
	Torpedo^ operator=(Torpedo^);
	Bitmap^ getPicture();
```

```
        void setPicture(Bitmap^);
        String^ getName();
        void setName(String^);
        String^ getVesselName();
        void setVesselName(String^);
        TeamSymbol getTeam();
        void setTeam(TeamSymbol);
        double getSpeed();
        void setSpeed(double);
        double getDamage();
        void setDamage(double);
        double getAngle();
        void setAngle(double);
        PointF getNowCoord();
        void setNowCoord(PointF);
        PointF getNowCoord4Draw();
        void setNowCoord4Draw(PointF);
        PointF getGoalCoord();
        void setGoalCoord(PointF);
        double getTotalFlyingTime();
        void setTotalFlyingTime(double);
        int getNowFlyingTime();
        void setNowFlyingTime(int);
        void TorpedoMove();
        // Torpedo's Icon
        bool render(Graphics^);
private:
        Bitmap^ picture;
        String^ name;
        String^ vesselName;
        TeamSymbol team;
        double speed;
        double damage;
        double angle;
        PointF nowCoord;
        PointF nowCoord4Draw;
        PointF goalCoord;
        double totalFlyingTime;
```

```
	int nowFlyingTime;
	// Helper function of render
	// Rotate to icon
	Bitmap^ RotateImage(Bitmap^, float);
	// Calculate the new size
	Size^ CalculateNewSize(int, int, double);
};
Torpedo.cpp
#include "Torpedo.h"
// Default constructor
Torpedo::Torpedo() {
	picture = PICTURE;
	name = vesselName = "Unknown";
	team = NONE;
	speed = damage = angle = 0.0;
	nowCoord = nowCoord4Draw = goalCoord = PointF();
	totalFlyingTime = 0.0;
	nowFlyingTime = 0;
}
// Copy constructor
Torpedo::Torpedo(Torpedo^& rSide) {
	picture = rSide->picture;
	name = rSide->name;
	vesselName = rSide->vesselName;
	team = rSide->team;
	speed = rSide->speed;
	damage = rSide->damage;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	goalCoord = rSide->goalCoord;
	totalFlyingTime = rSide->totalFlyingTime;
	nowFlyingTime = rSide->nowFlyingTime;
}
// Constructor
Torpedo::Torpedo(String^ newName, String^ newVesselName, TeamSymbol whichTeam, double newSpeed, double newDamage,
double newAngle, PointF beginCoord, PointF endCoord) {
	picture = PICTURE;
	name = newName;
	vesselName = newVesselName;
	team = whichTeam;
	speed = newSpeed;
	damage = newDamage;
	angle = newAngle;
	nowCoord = beginCoord;
	nowCoord4Draw = COORD_TO_DRAWING_COORD(nowCoord.X, nowCoord.Y);
	goalCoord = endCoord;
	if (speed != 0.0)
		totalFlyingTime = Math::Sqrt(Math::Abs(((nowCoord.X - goalCoord.X) * (nowCoord.X - goalCoord.X)) +
((nowCoord.Y - goalCoord.Y) * (nowCoord.Y - goalCoord.Y)))) / speed;
	else
		totalFlyingTime = INF;
	nowFlyingTime = 0;
}
// Overload assign operator
Torpedo^ Torpedo::operator=(Torpedo^ rSide) {
	picture = rSide->picture;
	name = rSide->name;
	vesselName = rSide->vesselName;
	team = rSide->team;
	speed = rSide->speed;
	damage = rSide->damage;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	goalCoord = rSide->goalCoord;
	totalFlyingTime = rSide->totalFlyingTime;
	nowFlyingTime = rSide->nowFlyingTime;
	return this;
}
#pragma region getters and setters
Bitmap^ Torpedo::getPicture() {
	return picture;
}
void Torpedo::setPicture(Bitmap^ newPicture) {
	picture = newPicture;
```

```
}
// Get torpedo's name
String^ Torpedo::getName() {
        return name;
}
// Set torpedo's name
void Torpedo::setName(String^ newName) {
        name = newName;
}
String^ Torpedo::getVesselName() {
        return vesselName;
}
void Torpedo::setVesselName(String^ newVesselName) {
        vesselName = newVesselName;
}
// Get torpedo's team
TeamSymbol Torpedo::getTeam() {
        return team;
}
// Set torpedo's team
void Torpedo::setTeam(TeamSymbol whichTeam) {
        team = whichTeam;
}
// Get torpedo's speed
double Torpedo::getSpeed() {
        return speed;
}
// Set torpedo's speed
void Torpedo::setSpeed(double newSpeed) {
        speed = newSpeed;
}
// Get torpedo's damage
double Torpedo::getDamage() {
        return damage;
}
// Set torpedo's damage
void Torpedo::setDamage(double newDamage) {
        damage = newDamage;
```

```
}
// Get torpedo's angle
double Torpedo::getAngle() {
	return angle;
}
// Set torpedo's angle
void Torpedo::setAngle(double newAngle) {
	angle = newAngle;
}
// Get coordinate now
PointF Torpedo::getNowCoord() {
	return nowCoord;
}
// Set coordinate now
void Torpedo::setNowCoord(PointF newNowCoord) {
	nowCoord = newNowCoord;
}
// Get coordinate on the graphic now
PointF Torpedo::getNowCoord4Draw() {
	return nowCoord4Draw;
}
// Set coordinate on the graphic now
void Torpedo::setNowCoord4Draw(PointF newNowCoord4Draw) {
	nowCoord4Draw = newNowCoord4Draw;
}
// Get goal Coordinate
PointF Torpedo::getGoalCoord() {
	return goalCoord;
}
// Set goal Coordinate
void Torpedo::setGoalCoord(PointF newgoalCoord) {
	goalCoord = newgoalCoord;
}
double Torpedo::getTotalFlyingTime() {
	return totalFlyingTime;
}
void Torpedo::setTotalFlyingTime(double newTotalFlyingTime) {
	totalFlyingTime = newTotalFlyingTime;
```

```
}
int Torpedo::getNowFlyingTime() {
	return nowFlyingTime;
}
void Torpedo::setNowFlyingTime(int newNowFlyingTime) {
	nowFlyingTime = newNowFlyingTime;
}
#pragma endregion
// Move function of torpedo
void Torpedo::TorpedoMove() {
	// Laplace x
	double dx = (getGoalCoord().X - getNowCoord().X);
	// Laplace y
	double dy = (getGoalCoord().Y - getNowCoord().Y);
	// Total difference
	double dis = Math::Sqrt((dx * dx) + (dy * dy));
	// Set new Coordinate
	setNowCoord(PointF(getNowCoord().X + getSpeed() * (dx / dis), getNowCoord().Y + getSpeed() * (dy / dis)));
	// Set Coordinate on the graphic
	setNowCoord4Draw(COORD_TO_DRAWING_COORD(getNowCoord().X, getNowCoord().Y));
}
// Icon of torpedo
bool Torpedo::render(Graphics^ g) {
	// draw picture
	// Image after rotate
	Bitmap^ bitmapRotate;
	bitmapRotate = RotateImage(picture, angle + 180);
	// Create Point of center.
	Point ulCorner = Point(getNowCoord4Draw().X - bitmapRotate->Width / 2.0, getNowCoord4Draw().Y - bitmapRotate-
>Height / 2.0);
	// Draw image to screen.
	g->DrawImage(bitmapRotate, ulCorner);
	#pragma region draw name
	g->DrawString(name, gcnew Font("Consolas", 8), gcnew SolidBrush(team == A ? Color::Red : Color::Blue),
PointF(nowCoord4Draw.X + BIAS_X_4_DRAW_NAME, nowCoord4Draw.Y - BIAS_Y_4_DRAW_NAME));
	#pragma endreigon
	return true;
}
```

```
//Helper functions of render
// Rotate to icon
Bitmap^ Torpedo::RotateImage(Bitmap^ image, float angle) {
	Size^ newSize = CalculateNewSize(image->Width, image->Height, angle);
	Bitmap^ rotatedBmp = gcnew Bitmap(newSize->Width, newSize->Height);
	PointF^ centerPoint = gcnew PointF((float)rotatedBmp->Width / 2.0, (float)rotatedBmp->Height / 2.0);
	Graphics^ g = Graphics::FromImage(rotatedBmp);
	g->CompositingQuality = CompositingQuality::HighQuality;
	g->SmoothingMode = SmoothingMode::HighQuality;
	g->InterpolationMode = InterpolationMode::HighQualityBicubic;
	g->TranslateTransform(centerPoint->X, centerPoint->Y);
	g->RotateTransform(angle);
	g->TranslateTransform(-centerPoint->X, -centerPoint->Y);
	g->DrawImage(image, (float)(newSize->Width - image->Width) / 2.0, (float)(newSize->Height - image->Height) / 2.0,
image->Width, image->Height);
	//g->Dispose();
	return rotatedBmp;
}
// Calculate the new size
Size^ Torpedo::CalculateNewSize(int width, int height, double angle) {
	double r = Math::Sqrt(Math::Pow((double)width / 2.0, 2.0) + Math::Pow((double)height / 2.0, 2.0)); //半徑L
	double OriginalAngle = Math::Acos((width / 2.0) / r) / Math::PI * 180.0;	//對角線和X軸的角度θ
	double minW = 0.0, maxW = 0.0, minH = 0.0, maxH = 0.0; //最大和最小的 X、Y座標
	double drawPoint[4];
	drawPoint[0] = (-OriginalAngle + angle) * Math::PI / 180.0;
	drawPoint[1] = (OriginalAngle + angle) * Math::PI / 180.0;
	drawPoint[2] = (180.0 - OriginalAngle + angle) * Math::PI / 180.0;
	drawPoint[3] = (180.0 + OriginalAngle + angle) * Math::PI / 180.0;
	// 由四個角的點算出 X、Y 的最大值及最小值
	for (int i = 0; i < 4; i++){
		double x = r * Math::Cos(drawPoint[i]);
		double y = r * Math::Sin(drawPoint[i]);
		if (x < minW)
			minW = x;
		if (x > maxW)
			maxW = x;
		if (y < minH)
			minH = y;
		if (y > maxH)
			maxH = y;
	}
	return gcnew Size((int)(maxW - minW), (int)(maxH - minH));
}
```

Typhoon.h:

```
#pragma once
#include "HydrometeorologicSystem.h"
#define SPEED			(4.0 / 60.0)
#define DAMAGE			0.05
#define ATTACK_RANGE	2.0
ref class Typhoon : public HydrometeorologicSystem {
public:
	Typhoon();
	Typhoon(Typhoon^&);
	Typhoon(double, PointF);
	Typhoon^ operator=(Typhoon^);
	virtual double getRotatePictureAngle() override;
	virtual void setRotatePictureAngle(double) override;
	virtual bool render(Graphics^) override;
private:
	double rotatePictureAngle;
};
```

Typhoon.cpp:

```
#include "Typhoon.h"
Typhoon::Typhoon(): HydrometeorologicSystem() {
	picture = PICTURE_TYPHOON;
	rotatePictureAngle = 0.0;
	gpHydro = gcnew GraphicsPath();
}
Typhoon::Typhoon(Typhoon^& rSide) {
	picture = rSide->picture;
	hydroType = rSide->hydroType;
	damage = rSide->damage;
	attackRange = rSide->attackRange;
	speed = rSide->speed;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	rotatePictureAngle = rSide->rotatePictureAngle;
	existTime = rSide->existTime;
	gpHydro = rSide->gpHydro;
}
Typhoon::Typhoon(double newAngle, PointF newNowCoord): HydrometeorologicSystem(newAngle, newNowCoord) {
	hydroType = TYPHOON;
	picture = PICTURE_TYPHOON;
	damage = DAMAGE;
	attackRange = ATTACK_RANGE;
	speed = SPEED;
	rotatePictureAngle = 0.0;
	gpHydro = gcnew GraphicsPath();
	gpHydro->AddRectangle(RectangleF(getNowCoord4Draw().X - picture->Width / 2.0, getNowCoord4Draw().Y - picture-
>Height / 2.0, picture->Width, picture->Height));
}
Typhoon^ Typhoon::operator=(Typhoon^ rSide) {
	picture = rSide->picture;
	hydroType = rSide->hydroType;
	damage = rSide->damage;
	attackRange = rSide->attackRange;
	speed = rSide->speed;
	angle = rSide->angle;
	nowCoord = rSide->nowCoord;
	nowCoord4Draw = rSide->nowCoord4Draw;
	rotatePictureAngle = rSide->rotatePictureAngle;
	existTime = rSide->existTime;
	gpHydro = rSide->gpHydro;
	return this;
}
double Typhoon::getRotatePictureAngle() {
	return rotatePictureAngle;
}
void Typhoon::setRotatePictureAngle(double newRotatePictureAngle) {
	rotatePictureAngle = newRotatePictureAngle;
	if (rotatePictureAngle == 360.0)
		rotatePictureAngle = 0.0;
}
```

```
bool Typhoon::render(Graphics^ g) {
	#pragma region draw picture
	// Image after rotate
	Bitmap^ bitmapRotate;
	bitmapRotate = RotateImage(picture, rotatePictureAngle + 180);
	// Create Point of center.
	Point ulCorner = Point(getNowCoord4Draw().X - bitmapRotate->Width / 2.0, getNowCoord4Draw().Y - bitmapRotate-
>Height / 2.0);
	// Draw image to screen.
	g->DrawImage(bitmapRotate, ulCorner);
	#pragma endregion
	HydrometeorologicSystem::render(g);
	return true;
}
VesselAH.h:
#pragma once
// Mercy
#include "VesselObject.h"
#pragma region data table
#define NAME                  ("CV")
#define HP                    5.0
#define MAX_VESSEL_SPEED   (1.0 / 60.0)
#define MAX_ATTACK_DIS        25.0
#define ATTACK_CD             60.0
#define MAX_DEFENSE_DIS       5.0
#define DEFENSE_CD            15.0
#define DAMAGE                3.0
#define SHELL_SPEED           (4.0 / 60.0)
#define MEnrichValue    1.0
#define MEnrichValue_dis   3.0
#define MEnrichValue_CD    50.0
#define l1ong 6.0;
#define halfl1ong 3.0;
#define w1ong 4.0;
#define halfw1long 2.0;
#pragma endregion
ref class VesselAH : public VesselObject {
public:
	VesselAH();
	VesselAH(VesselAH^&);
	VesselAH(String^, TeamSymbol, PointF);
	VesselAH^ operator=(VesselAH^);
	virtual bool render(Graphics^) override;
	virtual void setEnrichValue(double _EnrichValue) override;
	virtual double getEnrichValue() override;
	virtual void setEnrichValue_dis(double _EnrichValue_dis) override;
	virtual double getEnrichValue_dis() override;
	virtual void setEnrichValue_CD(double _EnrichValue_CD) override;
	virtual double getEnrichValue_CD() override;
	virtual void set_remain_Enrich_CD(double _remain_Enrich) override;
	virtual double get_remain_Enrich_CD() override;
	virtual void minus1remain_Enrich_CD() override;
	virtual bool isEnrichVessel(double, double) override;
private:
	double EnrichValue;
	double EnrichValue_dis;
	double EnrichValue_CD;
	double remain_Enrich_CD;
};
VesselAH.cpp:
// Mercy
#include "VesselAH.h"
// Default constructor
VesselAH::VesselAH() :
	VesselObject(NAME, NONE, AH, PointF(), HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
	EnrichValue = MEnrichValue;
	EnrichValue_CD = MEnrichValue_CD;
	EnrichValue_dis = MEnrichValue_dis;
	remain_Enrich_CD = 0;
}
// Copy constructor
VesselAH::VesselAH(VesselAH^& rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	this->EnrichValue = rSide->EnrichValue;
	this->EnrichValue_CD = rSide->EnrichValue_CD;
	this->EnrichValue_dis = rSide->EnrichValue_dis;
	this->remain_Enrich_CD = rSide->remain_Enrich_CD;
	canMove = rSide->canMove;
}
// Constructor
VesselAH::VesselAH(String^ _name, TeamSymbol _team, PointF _coord) :
	VesselObject(_name, _team, AH, _coord, HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
	/*darling*/
	EnrichValue = MEnrichValue;
	EnrichValue_CD = MEnrichValue_CD;
	EnrichValue_dis = MEnrichValue_dis;
	remain_Enrich_CD = 0;
}
// Overload assignment operator
VesselAH^ VesselAH::operator=(VesselAH^ rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	this->EnrichValue = rSide->EnrichValue;
	this->EnrichValue_CD = rSide->EnrichValue_CD;
	this->EnrichValue_dis = rSide->EnrichValue_dis;
	this->remain_Enrich_CD = rSide->remain_Enrich_CD;
	canMove = rSide->canMove;
	return this;
}
// Icon of CV
bool VesselAH::render(Graphics^ g) {
	Brush^ bruVessel = gcnew SolidBrush((getTeam() == A) ? Color::Red : Color::Blue);
	Pen^ penVessel = gcnew Pen(Color::Black);
	array<PointF>^ point_Regular_cross = gcnew array<PointF>(12);
	point_Regular_cross[0].X = getCoord4Draw().X + halfw1long; point_Regular_cross[0].Y = getCoord4Draw().Y - halfw1long;
	point_Regular_cross[1].X = point_Regular_cross[0].X; point_Regular_cross[1].Y = point_Regular_cross[0].Y - l1ong;
	point_Regular_cross[2].X = getCoord4Draw().X - halfw1long; point_Regular_cross[2].Y = point_Regular_cross[1].Y;
	point_Regular_cross[3].X = point_Regular_cross[2].X; point_Regular_cross[3].Y = point_Regular_cross[0].Y;
	point_Regular_cross[4].X = point_Regular_cross[3].X - l1ong; point_Regular_cross[4].Y = point_Regular_cross[3].Y;
	point_Regular_cross[5].X = point_Regular_cross[4].X; point_Regular_cross[5].Y = getCoord4Draw().Y + halfw1long;
	point_Regular_cross[6].X = point_Regular_cross[3].X; point_Regular_cross[6].Y = point_Regular_cross[5].Y;
	point_Regular_cross[7].X = point_Regular_cross[6].X; point_Regular_cross[7].Y = point_Regular_cross[6].Y + l1ong;
	point_Regular_cross[8].X = point_Regular_cross[0].X; point_Regular_cross[8].Y = point_Regular_cross[7].Y;
	point_Regular_cross[9].X = point_Regular_cross[0].X; point_Regular_cross[9].Y = point_Regular_cross[6].Y;
	point_Regular_cross[10].X = point_Regular_cross[9].X + l1ong; point_Regular_cross[10].Y = point_Regular_cross[9].Y;
	point_Regular_cross[11].X = point_Regular_cross[10].X; point_Regular_cross[11].Y = point_Regular_cross[0].Y;
	gpVessel->Reset();
	gpVessel->AddPolygon(point_Regular_cross);
	g->FillPath(bruVessel, gpVessel);
	g->DrawPath(penVessel, gpVessel);
	VesselObject::render(g);
	return true;
}
void VesselAH::setEnrichValue(double _EnrichValue) {
	this->EnrichValue = _EnrichValue;
}
double VesselAH::getEnrichValue() {
	return this->EnrichValue;
}
void VesselAH::setEnrichValue_CD(double _EnrichValueCD) {
	this->EnrichValue_CD = _EnrichValueCD;
	if (EnrichValue_CD< 0.0)
		EnrichValue_CD = 0.0;
}
double VesselAH::getEnrichValue_CD() {
	return this->EnrichValue_CD;
}
void VesselAH::setEnrichValue_dis(double _EnrichValue_dis) {
	this->EnrichValue_dis = _EnrichValue_dis;
	if (EnrichValue_dis< 0.0)
		EnrichValue_dis = 0.0;
}
double VesselAH::getEnrichValue_dis() {
	return this->EnrichValue_dis;
}
void VesselAH::set_remain_Enrich_CD(double _remain_Enrich) {
	this->remain_Enrich_CD = _remain_Enrich;
	if (remain_Enrich_CD< 0.0)
		remain_Enrich_CD = 0.0;
}
```

```
double VesselAH::get_remain_Enrich_CD() {
	return this->remain_Enrich_CD;
}
void VesselAH::minus1remain_Enrich_CD() {
	this->remain_Enrich_CD -= 1.0;
	if (remain_Enrich_CD< 0.0)
		remain_Enrich_CD = 0.0;
}
bool VesselAH::isEnrichVessel(double _x, double _y) {
	double fx = (getCoord().X - _x)*(getCoord().X - _x);
	double fy = (getCoord().Y - _y)*(getCoord().Y - _y);
	if (fx + fy <= MEnrichValue_dis * MEnrichValue_dis)
		return true;
	else
		return false;
}
VesselBB.h:
#pragma once
// Battle Ship
#include "VesselObject.h"
#pragma region data table
#define NAME				("BB")
#define HP				4.0
#define MAX_VESSEL_SPEED	(1.0 / 60.0)
#define MAX_ATTACK_DIS			20.0
#define ATTACK_CD			30.0
#define MAX_DEFENSE_DIS			10.0
#define DEFENSE_CD			30.0
#define DAMAGE				3.0
#define SHELL_SPEED			(2.0 / 60.0)
#pragma endregion
ref class VesselBB : public VesselObject {
public:
	VesselBB();
	VesselBB(VesselBB^&);
	VesselBB(String^, TeamSymbol, PointF);
	VesselBB^ operator=(VesselBB^);
	virtual bool render(Graphics^) override;
};
VesselBB.cpp:
// Battle Ship
#include "VesselBB.h"
// Default constructor
VesselBB::VesselBB() :
	VesselObject(NAME, NONE, UNKNOWN_VESSEL_TYPE, PointF(), HP, MAX_VESSEL_SPEED, 
MAX_ATTACK_DIS, ATTACK_CD, MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
}
// Constructor
VesselBB::VesselBB(String^ _name, TeamSymbol _team, PointF _coord) :
	VesselObject(_name, _team, BB, _coord, HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD, 
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
}
// Copy constructor
VesselBB::VesselBB(VesselBB^& rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
}
```

```
// Overload assignment operator
VesselBB^ VesselBB::operator=(VesselBB^ rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
	return this;
}
// Icon of BB
bool VesselBB::render(Graphics^ g) {
	Brush^ bruVessel = gcnew SolidBrush((getTeam() == A) ? Color::Red : Color::Blue);
	Pen^ penVessel = gcnew Pen(Color::Black);
	gpVessel->Reset();
	gpVessel->AddEllipse(getCoord4Draw().X - (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y - 
(VESSEL_ICON_LENGTH / 2.0), VESSEL_ICON_LENGTH, VESSEL_ICON_LENGTH);
	g->FillPath(bruVessel, gpVessel);
	g->DrawPath(penVessel, gpVessel);
	VesselObject::render(g);
	return true;
}
```
VesselCG.h:

```
#pragma once
// Battle Cruiser
#include "VesselObject.h"
#pragma region data table
#define NAME                    ("CG")
#define HP                      3.0
#define MAX_VESSEL_SPEED    (2.0 / 60.0)
#define MAX_ATTACK_DIS          15.0
#define ATTACK_CD               30.0
#define MAX_DEFENSE_DIS         15.0
#define DEFENSE_CD              30.0
#define DAMAGE                  2.0
#define SHELL_SPEED             (3.0 / 60.0)
#define LAUNCH_CD               60.0
#define TORPEDO_DAMAGE          3.0
#define TORPEDO_SPEED       (2.0 / 60.0)
#pragma endregion
ref class VesselCG : public VesselObject {
public:
	VesselCG();
	VesselCG(VesselCG^&);
	VesselCG(String^, TeamSymbol, PointF);
	VesselCG^ operator=(VesselCG^);
	virtual bool render(Graphics^) override;
};
```

VesselCG.cpp:

```
// Battle Cruiser
#include "VesselCG.h"
// Default constructor
VesselCG::VesselCG() :
	VesselObject(NAME, NONE, CG, PointF(), HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
}
// Copy constructor
VesselCG::VesselCG(VesselCG^& rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	maxDefenseDistance = rSide->maxDefenseDistance;
	attackCD = rSide->attackCD;
	launchCD = rSide->launchCD;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	torpedoDamage = rSide->torpedoDamage;
	torpedoSpeed = rSide->torpedoSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainLaunchCD = rSide->remainLaunchCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
}
// Constructor
VesselCG::VesselCG(String^ _name, TeamSymbol _team, PointF _coord) :
	VesselObject(_name, _team, CG, _coord, HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
	launchCD = LAUNCH_CD;
	remainLaunchCD = 0;
	torpedoDamage = TORPEDO_DAMAGE;
	torpedoSpeed = TORPEDO_SPEED;
}
// Overload assignment operator
VesselCG^ VesselCG::operator=(VesselCG^ rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	maxDefenseDistance = rSide->maxDefenseDistance;
	attackCD = rSide->attackCD;
	launchCD = rSide->launchCD;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	torpedoDamage = rSide->torpedoDamage;
	torpedoSpeed = rSide->torpedoSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainLaunchCD = rSide->remainLaunchCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
	return this;
}
// Icon of CG
bool VesselCG::render(Graphics^ g) {
	Brush^ bruVessel = gcnew SolidBrush((getTeam() == A) ? Color::Red : Color::Blue);
	Pen^ penVessel = gcnew Pen(Color::Black);
	// Create array of points that define lines to draw.
	array<Point>^ points = gcnew array<Point>(4);
	points[0] = Point(getCoord4Draw().X, getCoord4Draw().Y + (VESSEL_ICON_LENGTH / 2.0) * 1.732);
	points[1] = Point(getCoord4Draw().X + (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y);
	points[2] = Point(getCoord4Draw().X, getCoord4Draw().Y - (VESSEL_ICON_LENGTH / 2.0) * 1.732);
	points[3] = Point(getCoord4Draw().X - (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y);
	gpVessel->Reset();
	gpVessel->AddPolygon(points);
	g->FillPath(bruVessel, gpVessel);
	g->DrawPath(penVessel, gpVessel);
	VesselObject::render(g);
	return true;
}
VesselCV.h:
#pragma once
// Aircraft Carrier
#include "VesselObject.h"
#pragma region data table
#define NAME				("CV")
#define HP					5.0
#define MAX_VESSEL_SPEED	(1.0 / 60.0)
#define MAX_ATTACK_DIS		25.0
#define ATTACK_CD			60.0
#define MAX_DEFENSE_DIS		5.0
#define DEFENSE_CD			15.0
#define DAMAGE				3.0
#define SHELL_SPEED			(4.0 / 60.0)
#pragma endregion
ref class VesselCV : public VesselObject {
public:
	VesselCV();
	VesselCV(VesselCV^&);
	VesselCV(String^, TeamSymbol, PointF);
	VesselCV^ operator=(VesselCV^);
	virtual bool render(Graphics^) override;
};
VesselCV.cpp:
// Aircraft Carrier
#include "VesselCV.h"
// Default constructor
VesselCV::VesselCV():
	VesselObject(NAME, NONE, CV, PointF(), HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
}
// Copy constructor
VesselCV::VesselCV(VesselCV^& rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
}
// Constructor
VesselCV::VesselCV(String^ _name, TeamSymbol _team, PointF _coord):
	VesselObject(_name, _team, CV, _coord, HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
}
// Overload assignment operator
VesselCV^ VesselCV::operator=(VesselCV^ rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
	return this;
}
// Icon of CV
bool VesselCV::render(Graphics^ g) {
	Brush^ bruVessel = gcnew SolidBrush((getTeam() == A) ? Color::Red : Color::Blue);
	Pen^ penVessel = gcnew Pen(Color::Black);
	gpVessel->Reset();
	gpVessel->AddRectangle(RectangleF(getCoord4Draw().X - (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y - (VESSEL_ICON_LENGTH / 2.0), VESSEL_ICON_LENGTH, VESSEL_ICON_LENGTH));
	g->FillPath(bruVessel, gpVessel);
	g->DrawPath(penVessel, gpVessel);
	VesselObject::render(g);
	return true;
}
VesselDD.h:
#pragma once
// Destroyer
#include "VesselObject.h"
#pragma region data table
#define NAME				("DD")
#define HP				2.0
#define MAX_VESSEL_SPEED	(3.0 / 60.0)
#define MAX_ATTACK_DIS		10.0
#define ATTACK_CD			15.0
#define MAX_DEFENSE_DIS		20.0
#define DEFENSE_CD			60.0
#define DAMAGE			1.0
#define SHELL_SPEED			(3.0 / 60.0)
#define LAUNCH_CD			30.0
```

```
#define TORPEDO_DAMAGE			3.0
#define TORPEDO_SPEED		(2.0 / 60.0)
#pragma endregion
ref class VesselDD : public VesselObject {
public:
	VesselDD();
	VesselDD(VesselDD^&);
	VesselDD(String^, TeamSymbol, PointF);
	VesselDD^ operator=(VesselDD^);
	virtual bool render(Graphics^) override;
};
VesselDD.cpp:
// Destroyer
#include "VesselDD.h"
// Default constructor
VesselDD::VesselDD() :
	VesselObject(NAME, NONE, DD, PointF(), HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
}
// Copy constructor
VesselDD::VesselDD(VesselDD^& rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	maxDefenseDistance = rSide->maxDefenseDistance;
	attackCD = rSide->attackCD;
	launchCD = rSide->launchCD;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	torpedoDamage = rSide->torpedoDamage;
	torpedoSpeed = rSide->torpedoSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainLaunchCD = rSide->remainLaunchCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
}
// Constructor
VesselDD::VesselDD(String^ _name, TeamSymbol _team, PointF _coord) :
	VesselObject(_name, _team, DD, _coord, HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {

	launchCD = LAUNCH_CD;
	remainLaunchCD = 0;
	torpedoDamage = TORPEDO_DAMAGE;
	torpedoSpeed = TORPEDO_SPEED;
}
// Overload assignment operator
VesselDD^ VesselDD::operator=(VesselDD^ rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	maxDefenseDistance = rSide->maxDefenseDistance;
	attackCD = rSide->attackCD;
	launchCD = rSide->launchCD;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	torpedoDamage = rSide->torpedoDamage;
	torpedoSpeed = rSide->torpedoSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainLaunchCD = rSide->remainLaunchCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
	return this;
}

// Icon of DD
bool VesselDD::render(Graphics^ g) {
	Brush^ bruVessel = gcnew SolidBrush((getTeam() == A) ? Color::Red : Color::Blue);
	Pen^ penVessel = gcnew Pen(Color::Black);
	// Create array of points that define lines to draw.
	array<Point>^ points = gcnew array<Point>(3);
	if (getTeam() == A) {
		points[0] = Point(getCoord4Draw().X, getCoord4Draw().Y + (VESSEL_ICON_LENGTH / 2.0) * 2);
		points[1] = Point(getCoord4Draw().X + (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y - (VESSEL_ICON_LENGTH / 2.0));
		points[2] = Point(getCoord4Draw().X - (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y - (VESSEL_ICON_LENGTH / 2.0));
	}
	else {
		points[0] = Point(getCoord4Draw().X, getCoord4Draw().Y - (VESSEL_ICON_LENGTH / 2.0) * 2);
		points[1] = Point(getCoord4Draw().X - (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y + (VESSEL_ICON_LENGTH / 2.0));
		points[2] = Point(getCoord4Draw().X + (VESSEL_ICON_LENGTH / 2.0), getCoord4Draw().Y + (VESSEL_ICON_LENGTH / 2.0));
	}

	gpVessel->Reset();
	gpVessel->AddPolygon(points);
	g->FillPath(bruVessel, gpVessel);
	g->DrawPath(penVessel, gpVessel);
	VesselObject::render(g);
	return true;
```

```
}
VesselLV.h:
#pragma once
// Laser Vessel
#include "VesselObject.h"
#define LASER_ATTACK_TIME    9.0
#define LASER_ATTACK_RADIUS          0.6
#pragma region data table
#define NAME                         ("LV")
#define HP                           1.0
#define MAX_VESSEL_SPEED      (4.0 / 60.0)
#define MAX_ATTACK_DIS               25.0
#define ATTACK_CD                    90.0
#define MAX_DEFENSE_DIS              0.0
#define DEFENSE_CD                   0.0
#define DAMAGE                       6.0
#define SHELL_SPEED                  (0.0 / 60.0)
#define LASER_CD              90.0
#pragma endregion
ref class VesselLV : public VesselObject {
public:
	VesselLV();
	VesselLV(VesselLV^&);
	VesselLV(String^, TeamSymbol, PointF);
	VesselLV^ operator=(VesselLV^);
	virtual bool render(Graphics^) override;
	//雷射判斷函數
	bool ShootLaser(double _xx, double _yy) {
		double X_Long = 0;
		if (LaserAngle >= 0 && LaserAngle <= 90 || LaserAngle <= 360 && LaserAngle >= 270) {
			X_Long = 20.0 - getCoord().X;
		}
		else {
			X_Long = getCoord().X - 0.0;
		}
		double slope_long = 0;
		if (Math::Cos(ANGLE_TO_RADIUS(LaserAngle)) == 0) { slope_long == 0; }
		else { slope_long = X_Long / Math::Cos(ANGLE_TO_RADIUS(LaserAngle)); }
```

```
        double Y_long;
        Y_long = slope_long * Math::Sin(ANGLE_TO_RADIUS(LaserAngle))*(-1);
        ////////////////
        if (X_Long != 0) {
            double slopeofLaser = Y_long / X_Long;
            double dis = Math::Abs(slopeofLaser*_xx + (-1)*_yy + (-1)*(slopeofLaser*getCoord().X +
getCoord().Y*(-1))) / Math::Sqrt(slopeofLaser*slopeofLaser + 1.0);
            if (dis <= 0.5) return true;
            else return false;
        }
        else {
            double dis = Math::Abs(_xx - getCoord().X);
            if (dis <= 0.5) return true;
            else return false;
        }
    }
    virtual bool DrawLaser(Graphics^) override;
    #pragma region getters and setters
    virtual void SetLaserAngle(double) override;
    virtual double GetLaserAngle() override;
    virtual void setIsLaserShooting(bool) override;
    virtual bool getIsLaserShooting() override;
    virtual void setLaserCD(double) override;
    virtual double getLaserCD() override;
    virtual void setRemainLaserCD(double) override;
    virtual void minus1RemainLaserCD() override;
    virtual double getRemainLaserCD() override;
    virtual void setIsLaserHitSomething(bool) override;
    virtual bool getIsLaserHitSomething() override;
    virtual void setLaserLog(String^) override;
    virtual void strcatLaserLog(String^) override;
    virtual String^ getLaserLog() override;
    #pragma endregion
private:
    double LaserAngle;
    double laserCD;
    double remainLaserCD;
    bool isLaserShooting;
```

```
	bool isLaserHitSomething;
	String^ laserLog;
};
VesselLV.cpp:
// Laser Vessel
#include "VesselLV.h"
// Default constructor
VesselLV::VesselLV() :
	VesselObject(NAME, NONE, LV, PointF(), HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
	laserCD = LASER_CD;
	isLaserHitSomething = false;
	laserLog = "";
}
// Copy constructor
VesselLV::VesselLV(VesselLV^& rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	laserCD = rSide->laserCD;
	isLaserHitSomething = rSide->isLaserHitSomething;
	laserLog = rSide->laserLog;
	canMove = rSide->canMove;
}
// Constructor
VesselLV::VesselLV(String^ _name, TeamSymbol _team, PointF _coord) :
	VesselObject(_name, _team, LV, _coord, HP, MAX_VESSEL_SPEED, MAX_ATTACK_DIS, ATTACK_CD,
MAX_DEFENSE_DIS, DEFENSE_CD, DAMAGE, SHELL_SPEED) {
	laserCD = LASER_CD;
	isLaserHitSomething = false;
	laserLog = "";
}
// Overload assignment operator
VesselLV^ VesselLV::operator=(VesselLV^ rSide) {
	mouseIn = rSide->mouseIn;
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	attackCD = rSide->attackCD;
	maxDefenseDistance = rSide->maxDefenseDistance;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	laserCD = rSide->laserCD;
	isLaserHitSomething = rSide->isLaserHitSomething;
	laserLog = rSide->laserLog;
	canMove = rSide->canMove;
	return this;
```

```
}
// Icon of LV
bool VesselLV::render(Graphics^ g) {
	Brush^ bruVessel = gcnew SolidBrush((getTeam() == A) ? Color::Red : Color::Blue);
	Pen^ penVessel = gcnew Pen(Color::Black);
	array<PointF>^ point_Regular_pentagon = gcnew array<PointF>(5);
	point_Regular_pentagon[0].X = getCoord4Draw().X; point_Regular_pentagon[0].Y = getCoord4Draw().Y - 10;
	point_Regular_pentagon[3].X = getCoord4Draw().X - 10 * Math::Cos(ANGLE_TO_RADIUS(54));
point_Regular_pentagon[3].Y = getCoord4Draw().Y + 10;
	point_Regular_pentagon[2].X = getCoord4Draw().X + 10 * Math::Cos(ANGLE_TO_RADIUS(54));
point_Regular_pentagon[2].Y = getCoord4Draw().Y + 10;
	double side = Math::Abs(10 * Math::Cos(ANGLE_TO_RADIUS(54)) * 2);
	point_Regular_pentagon[1].X = point_Regular_pentagon[2].X + (Math::Cos(ANGLE_TO_RADIUS(72))*side);
	point_Regular_pentagon[4].X = point_Regular_pentagon[3].X - (Math::Cos(ANGLE_TO_RADIUS(72))*side);
	point_Regular_pentagon[1].Y = point_Regular_pentagon[2].Y - (Math::Sin(ANGLE_TO_RADIUS(72))*side);
	point_Regular_pentagon[4].Y = point_Regular_pentagon[2].Y - (Math::Sin(ANGLE_TO_RADIUS(72))*side);
	gpVessel->Reset();
	gpVessel->AddPolygon(point_Regular_pentagon);
	g->FillPath(bruVessel, gpVessel);
	g->DrawPath(penVessel, gpVessel);
	VesselObject::render(g);
	return true;
}
#pragma region setters and getters
void VesselLV::SetLaserAngle(double _angle) {
	LaserAngle = _angle;
}
double VesselLV::GetLaserAngle() {
	return LaserAngle;
}
void VesselLV::setIsLaserShooting(bool newLaserShooting) {
	isLaserShooting = newLaserShooting;
}
bool VesselLV::getIsLaserShooting() {
	return isLaserShooting;
}
void VesselLV::setLaserCD(double newLaserCD) {
	laserCD = newLaserCD;
```

```
}
double VesselLV::getLaserCD() {
	return laserCD;
}
void VesselLV::setRemainLaserCD(double newRemainLaserCD) {
	remainLaserCD = newRemainLaserCD;
	if (remainLaserCD < 0.0)
		remainLaserCD = 0.0;
}
void VesselLV::minus1RemainLaserCD() {
	remainLaserCD -= 1.0;
	if (remainLaserCD < 0.0)
		remainLaserCD = 0.0;
}
double VesselLV::getRemainLaserCD() {
	return remainLaserCD;
}
void VesselLV::setIsLaserHitSomething(bool newIsLaserHitSomething) {
	isLaserHitSomething = newIsLaserHitSomething;
}
bool VesselLV::getIsLaserHitSomething() {
	return isLaserHitSomething;
}
void VesselLV::setLaserLog(String^ newLaserLog) {
	laserLog = newLaserLog;
}
void VesselLV::strcatLaserLog(String^ newLaserLog) {
	laserLog += newLaserLog;
}
String^ VesselLV::getLaserLog() {
	return laserLog;
}
#pragma endregion
bool VesselLV::DrawLaser(Graphics ^ g) {
	Pen^ penVessel = gcnew Pen(Color::Green, 2);
	double X_Long = 0;
	double Y_long = 0;
	double slope_long = 0;
```

```
if (LaserAngle > 360) LaserAngle -= 360;
if (LaserAngle < 0)LaserAngle += 360;
if (LaserAngle >0 && LaserAngle <90 || LaserAngle < 360 && LaserAngle >270) {
        if (LaserAngle >= 0 && LaserAngle <= 45 || LaserAngle >= 315 && LaserAngle < 360) {
                X_Long = BOARD_LENGTH - getCoord4Draw().X + BIAS_X;
                slope_long = X_Long / Math::Cos(ANGLE_TO_RADIUS(LaserAngle));
                Y_long = Math::Abs(slope_long) * Math::Sin(ANGLE_TO_RADIUS(LaserAngle))*(1);
                double fx = (getCoord4Draw().X + X_Long);
                double fy = getCoord4Draw().Y - Y_long;

                double slope = (getCoord4Draw().Y - fy) / (getCoord4Draw().X - fx);
                if (fx > (BOARD_LENGTH + BLOCK_LENGTH)) {
                        double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                        fx = BOARD_LENGTH + BLOCK_LENGTH;
                        fy = (c + slope * fx);
                }
                else    if (fx < BIAS_X) {
                        double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                        fx = BIAS_X;
                        fy = (c + slope * fx);
                }
                else    if (fy >(BOARD_LENGTH + BLOCK_LENGTH)) {
                        double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                        fy = BOARD_LENGTH + BLOCK_LENGTH;
                        fx = (fy + (-1)*c) / slope;
                }
                else if (fy < BIAS_Y) {
                        double c = (-1)*slope*getCoord4Draw().X + getCoord4Draw().Y;
                        fy = BIAS_Y;
                        fx = (fy + (-1)*c) / slope;
                }
                g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, fx, fy);
        }
        else if (LaserAngle >45 && LaserAngle <= 90) {
                Y_long = getCoord4Draw().Y - BIAS_Y;
                slope_long = Y_long / Math::Sin(ANGLE_TO_RADIUS(LaserAngle));
                X_Long = Math::Abs(slope_long)*Math::Cos(ANGLE_TO_RADIUS(LaserAngle));
```

```
			double fx = (getCoord4Draw().X + X_Long);
			double fy = getCoord4Draw().Y - Y_long;


			double slope = (getCoord4Draw().Y - fy) / (getCoord4Draw().X - fx);
			if (fx > BOARD_LENGTH + BLOCK_LENGTH) {
				double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
				fx = BOARD_LENGTH + BLOCK_LENGTH;
				fy = (c + slope * fx);
			}
			else	if (fx < BIAS_X) {
				double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
				fx = BIAS_X;
				fy = (c + slope * fx);
			}
			else	if (fy >(BOARD_LENGTH + BLOCK_LENGTH)) {
				double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
				fy = BOARD_LENGTH + BLOCK_LENGTH;
				fx = (fy + (-1)*c) / slope;
			}
			else	if (fy < BIAS_Y) {
				double c = (-1)*slope*getCoord4Draw().X + getCoord4Draw().Y;
				fy = BIAS_Y;
				fx = (fy + (-1)*c) / slope;
			}
			g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, fx, fy);
		}
		else {
			Y_long = BOARD_LENGTH - getCoord4Draw().Y + BIAS_Y;
			slope_long = Y_long / Math::Sin(ANGLE_TO_RADIUS(LaserAngle));
			X_Long = Math::Abs(slope_long)*Math::Cos(ANGLE_TO_RADIUS(LaserAngle));
			/*	if	((getCoord4Draw().X	+	X_Long)	>(BOARD_LENGTH	+	BLOCK_LENGTH)	||
(getCoord4Draw().X + X_Long) < BIAS_X && (getCoord4Draw().Y + Y_long) > (BOARD_LENGTH + BLOCK_LENGTH) ||
(getCoord4Draw().Y + Y_long < BIAS_Y)) {}
			else { g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, getCoord4Draw().X + X_Long,
getCoord4Draw().Y + Y_long); }
			*/
			double fx = (getCoord4Draw().X + X_Long);
			double fy = getCoord4Draw().Y + Y_long;
```

```
                double slope = (getCoord4Draw().Y - fy) / (getCoord4Draw().X - fx);
                if (fx > (BOARD_LENGTH + BLOCK_LENGTH)) {
                        double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                        fx = BOARD_LENGTH + BLOCK_LENGTH;
                        fy = (c + slope * fx);
                }
                else    if (fx < BIAS_X) {
                        double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                        fx = BIAS_X;
                        fy = (c + slope * fx);
                }
                else    if (fy >(BOARD_LENGTH + BLOCK_LENGTH)) {
                        double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                        fy = BOARD_LENGTH + BLOCK_LENGTH;
                        fx = (fy + (-1)*c) / slope;
                }
                else    if (fy < BIAS_Y) {
                        double c = (-1)*slope*getCoord4Draw().X + getCoord4Draw().Y;
                        fy = BIAS_Y;
                        fx = (fy + (-1)*c) / slope;
                }
                g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, fx, fy);
        }
}
else if (LaserAngle == 180) {
        X_Long = getCoord4Draw().X - BIAS_X;
        g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, (double)(BIAS_X), getCoord4Draw().Y);
}
else if (LaserAngle == 0) {
        X_Long = BOARD_LENGTH - getCoord4Draw().X;
        g->DrawLine(penVessel,   getCoord4Draw().X,   getCoord4Draw().Y,   (double)(BOARD_LENGTH   +
BLOCK_LENGTH), getCoord4Draw().Y);
}
else if (LaserAngle == 90) {
        Y_long = getCoord4Draw().Y - BIAS_Y;
        g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, getCoord4Draw().X, (double)(BIAS_Y));
}
else if (LaserAngle == 270) {
			Y_long = BOARD_LENGTH - getCoord4Draw().Y + BLOCK_LENGTH;
			g->DrawLine(penVessel,	getCoord4Draw().X,	getCoord4Draw().Y,	getCoord4Draw().X,
(double)(BOARD_LENGTH + BLOCK_LENGTH));
		}
		else {
			if (LaserAngle >= 135 && LaserAngle <= 225) {
				X_Long = getCoord4Draw().X - BIAS_X;
				slope_long = X_Long / Math::Cos(ANGLE_TO_RADIUS(LaserAngle));
				Y_long = Math::Abs(slope_long) * Math::Sin(ANGLE_TO_RADIUS(LaserAngle));

				double fx = (getCoord4Draw().X - X_Long);
				double fy = getCoord4Draw().Y - Y_long;
				double slope = (getCoord4Draw().Y - fy) / (getCoord4Draw().X - fx);
				if (fx > (BOARD_LENGTH + BLOCK_LENGTH)) {
					double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
					fx = BOARD_LENGTH + BLOCK_LENGTH;
					fy = (c + slope * fx);
				}
				else	if (fx < BIAS_X) {
					double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
					fx = BIAS_X;
					fy = (c + slope * fx);
				}
				else	if (fy >(BOARD_LENGTH + BLOCK_LENGTH)) {
					double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
					fy = BOARD_LENGTH + BLOCK_LENGTH;
					fx = (fy + (-1)*c) / slope;
				}
				else if (fy < BIAS_Y) {
					double c = (-1)*slope*getCoord4Draw().X + getCoord4Draw().Y;
					fy = BIAS_Y;
					fx = (fy + (-1)*c) / slope;
				}
				g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, fx, fy);
			}
			else if (LaserAngle >90 && LaserAngle <135) {
				Y_long = getCoord4Draw().Y - BIAS_Y;
				slope_long = Y_long / Math::Sin(ANGLE_TO_RADIUS(LaserAngle));
		X_Long = Math::Abs(slope_long)*Math::Cos(ANGLE_TO_RADIUS(LaserAngle));

		double fx = (getCoord4Draw().X + X_Long);
		double fy = getCoord4Draw().Y - Y_long;
		double slope = (getCoord4Draw().Y - fy) / (getCoord4Draw().X - fx);
		if (fx > (BOARD_LENGTH + BLOCK_LENGTH)) {
			double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
			fx = BOARD_LENGTH + BLOCK_LENGTH;
			fy = (c + slope * fx);
		}
		else if (fx < BIAS_X) {
			double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
			fx = BIAS_X;
			fy = (c + slope * fx);
		}
		else	if (fy >(BOARD_LENGTH + BLOCK_LENGTH)) {
			double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
			fy = BOARD_LENGTH + BLOCK_LENGTH;
			fx = (fy + (-1)*c) / slope;
		}
		else	if (fy < BIAS_Y) {
			double c = (-1)*slope*getCoord4Draw().X + getCoord4Draw().Y;
			fy = BIAS_Y;
			fx = (fy + (-1)*c) / slope;
		}
		g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, fx, fy);
	}
	else {
		Y_long = BOARD_LENGTH - getCoord4Draw().Y + BIAS_Y;
		slope_long = Y_long / Math::Sin(ANGLE_TO_RADIUS(LaserAngle));
		X_Long = Math::Abs(slope_long)*Math::Cos(ANGLE_TO_RADIUS(LaserAngle));
		double fx = (getCoord4Draw().X + X_Long);
		double fy = getCoord4Draw().Y + Y_long;
		double slope = (getCoord4Draw().Y - fy) / (getCoord4Draw().X - fx);
		if (fx > (BOARD_LENGTH + BLOCK_LENGTH)) {
			double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
			fx = BOARD_LENGTH + BLOCK_LENGTH;
			fy = (c + slope * fx);
```

```
                        }
                        else     if (fx < BIAS_X) {
                                double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                                fx = BIAS_X;
                                fy = (c + slope * fx);
                        }
                        else if (fy >(BOARD_LENGTH + BLOCK_LENGTH)) {
                                double c = (-1)*(slope*getCoord4Draw().X - getCoord4Draw().Y);
                                fy = BOARD_LENGTH + BLOCK_LENGTH;
                                fx = (fy + (-1)*c) / slope;
                        }
                        else if (fy < BIAS_Y) {
                                double c = (-1)*slope*getCoord4Draw().X + getCoord4Draw().Y;
                                fy = BIAS_Y;
                                fx = (fy + (-1)*c) / slope;
                        }
                        g->DrawLine(penVessel, getCoord4Draw().X, getCoord4Draw().Y, fx, fy);
                }
        }
        return true;
}
VesselObject.h:
#pragma once
#include "RenderParameter.h"
#include "Shell.h"
#include "Torpedo.h"
#define COLLAPSION_DAMAGE 1.0
#define TORPEDO_DAMAGE_RADIUS 1.2
using namespace System;
using namespace System::Windows;
using namespace System::Drawing;
using namespace System::Collections::Generic;
using namespace System::Diagnostics;
enum VesselType {CV = 0, BB, CG, DD, LV, AH, UNKNOWN_VESSEL_TYPE};
ref class VesselObject {
public:
        VesselObject();
        VesselObject(const VesselObject^&);
```

```
	VesselObject(String^, TeamSymbol, VesselType, PointF, double, double, double, double, double, double, double, double);
	virtual ~VesselObject();
	static array<String^>^ typeStringList = gcnew array<String^> {"CV", "BB", "CG", "DD", "LV", "AH",
"UNKNOWN_VESSEL_TYPE"};
	#pragma region getters and setters
	String^ getName();
	void setName(String^);
	TeamSymbol getTeam();
	void setTeam(TeamSymbol);
	VesselType getVesselType();
	void setVesselType(VesselType);
	PointF getCoord();
	void setCoord(PointF);
	PointF getCoord4Draw();
	void setCoord4Draw(PointF);
	GraphicsPath^ getGp();
	double getMaxHp();
	void setMaxHp(double);
	double getRemainHp();
	void setRemainHp(double);
	double getMaxVesselSpeed();
	void setMaxVesselSpeed(double);
	double getMaxAttackDistance();
	void setMaxAttackDistance(double);
	double getAttackCD();
	void setAttackCD(double);
	// Torpedo CD
	double getLaunchCD();
	void setLaunchCD(double);
	double getMaxDefenseDistance();
	void setMaxDefenseDistance(double);
	double getDefenseCD();
	void setDefenseCD(double);
	double getDamage();
	void setDamage(double);
	double getShellSpeed();
	void setShellSpeed(double);
	double getRemainAttackCD();
```

```
void setRemainAttackCD(double);
void minus1RemainAttackCD();
// Torpedo only
double getRemainLaunchCD();
void setRemainLaunchCD(double);
void minus1RemainLaunchCD();
// Torpedo only
double getTorpedoDamage();
void setTorpedoDamage(double);
double getTorpedoSpeed();
void setTorpedoSpeed(double);
double getRemainDefenseCD();
void setRemainDefenseCD(double);
void minus1RemainDefenseCD();
double getNowSpeed();
void setNowSpeed(double);
double getNowAngle();
void setNowAngle(double);
void setCanMove(bool);
bool getCanMove();
void initializeMove();
#pragma region LV only
virtual void SetLaserAngle(double _angle) override;
virtual double GetLaserAngle() override;
virtual void setIsLaserShooting(bool _angle) override;
virtual bool getIsLaserShooting() override;
virtual void setLaserCD(double) override;
virtual double getLaserCD() override;
virtual void setRemainLaserCD(double) override;
virtual void minus1RemainLaserCD() override;
virtual double getRemainLaserCD() override;
virtual void setIsLaserHitSomething(bool) override;
virtual bool getIsLaserHitSomething() override;
virtual void setLaserLog(String^) override;
virtual void strcatLaserLog(String^) override;
virtual String^ getLaserLog() override;
#pragma endregion
#pragma region AH only
```

```
        virtual void setEnrichValue(double _EnrichValue) override;
        virtual double getEnrichValue() override;
        virtual void setEnrichValue_dis(double _EnrichValue_dis) override;
        virtual double getEnrichValue_dis() override;
        virtual void setEnrichValue_CD(double _EnrichValue_CD) override;
        virtual double getEnrichValue_CD() override;
        virtual void set_remain_Enrich_CD(double _remain_Enrich) override;
        virtual double get_remain_Enrich_CD() override;
        virtual void minus1remain_Enrich_CD() override;
        #pragma endregion
        #pragma endregion
        void VesselMove();
        void DaoTuiLu();
        bool BoundaryJudgment(double, double);
        virtual bool DrawLaser(Graphics^) override;
        virtual bool render(Graphics^) override;
        virtual bool isEnrichVessel(double, double) override;
        bool mouseIn;
protected:
        String^ name;
        TeamSymbol team;
        VesselType vesselType;
        PointF coord;
        PointF coord4Draw;
        GraphicsPath^ gpVessel;
        double maxHp;
        double remainHp;
        double maxVesselSpeed;
        double maxAttackDistance;
        double maxDefenseDistance;
        double attackCD;
        double launchCD;
        double defenseCD;
        double damage;
        double shellSpeed;
        double torpedoDamage;
        double torpedoSpeed;
        double remainAttackCD;
```

```
	double remainLaunchCD;
	double remainDefenseCD;
	double nowSpeed;
	double nowAngle;
	bool canMove;
};
VesselObject.cpp:
#include "VesselObject.h"
#pragma region constructors
VesselObject::VesselObject() {
	name = "Unknown";
	team = NONE;
	vesselType = UNKNOWN_VESSEL_TYPE;
	coord = PointF(0.0, 0.0);
	coord4Draw = coord;
	gpVessel = gcnew GraphicsPath();
	maxHp = 0;
	remainHp = 0;
	maxVesselSpeed = 0;
	maxAttackDistance = 0;
	maxDefenseDistance = 0;
	attackCD = 0;
	launchCD = 0;
	defenseCD = 0;
	damage = 0;
	shellSpeed = 0;
	torpedoDamage = 0;
	torpedoSpeed = 0;
	remainAttackCD = 0;
	remainLaunchCD = 0;
	remainDefenseCD = 0;
	nowSpeed = 0;
	nowAngle = 0;
	initializeMove();
}
VesselObject::VesselObject(const VesselObject^& rSide) {
	name = rSide->name;
	team = rSide->team;
	vesselType = rSide->vesselType;
	coord = rSide->coord;
	coord4Draw = rSide->coord4Draw;
	gpVessel = rSide->gpVessel;
	maxHp = rSide->maxHp;
	remainHp = rSide->remainHp;
	maxVesselSpeed = rSide->maxVesselSpeed;
	maxAttackDistance = rSide->maxAttackDistance;
	maxDefenseDistance = rSide->maxDefenseDistance;
	attackCD = rSide->attackCD;
	launchCD = rSide->launchCD;
	defenseCD = rSide->defenseCD;
	damage = rSide->damage;
	shellSpeed = rSide->shellSpeed;
	torpedoDamage = rSide->torpedoDamage;
	torpedoSpeed = rSide->torpedoSpeed;
	remainAttackCD = rSide->remainAttackCD;
	remainLaunchCD = rSide->remainLaunchCD;
	remainDefenseCD = rSide->remainDefenseCD;
	nowSpeed = rSide->nowSpeed;
	nowAngle = rSide->nowAngle;
	canMove = rSide->canMove;
	mouseIn = rSide->mouseIn;
}
VesselObject::VesselObject(String^ newName, TeamSymbol whichTeam, VesselType newVesselType, PointF newCoord, double newHp, double newMaxVesselSpeed, double newMaxAttackDistance, double newAttackCD, double newMaxDefenseDistance, double newDefenseCD, double newDamage, double newShellSpeed) {
	name = newName;
	team = whichTeam;
	vesselType = newVesselType;
	coord = newCoord;
	coord4Draw = COORD_TO_DRAWING_COORD(coord.X, coord.Y);
	gpVessel = gcnew GraphicsPath();
	maxHp = newHp;
	remainHp = newHp;
	maxVesselSpeed = newMaxVesselSpeed;;
	maxAttackDistance = newMaxAttackDistance;
	attackCD = newAttackCD;
	maxDefenseDistance = newMaxDefenseDistance;
	defenseCD = newDefenseCD;
	damage = newDamage;
	shellSpeed = newShellSpeed;
	remainAttackCD = 0;
	remainDefenseCD = 0;
	nowSpeed = 0;
	nowAngle = 0;
	initializeMove();
}
#pragma endregion
#pragma region deconstructor
VesselObject::~VesselObject() {
	//delete gpVessel;
	Debug::WriteLine("Deconstructor of VesselObject has been called!");
}
#pragma endregion
#pragma region getters and setters
String^ VesselObject::getName() {
	return name;
}
void VesselObject::setName(String^ newName) {
	name = newName;
}
TeamSymbol VesselObject::getTeam() {
	return team;
}
void VesselObject::setTeam(TeamSymbol newTeam) {
	team = newTeam;
}
VesselType VesselObject::getVesselType() {
	return vesselType;
}
void VesselObject::setVesselType(VesselType newVesselType) {
	vesselType = newVesselType;
}
PointF VesselObject::getCoord() {
	return coord;
```

```
}
void VesselObject::setCoord(PointF newCoord) {
        coord = newCoord;
}
PointF VesselObject::getCoord4Draw() {
        return coord4Draw;
}
void VesselObject::setCoord4Draw(PointF newCoord4Draw) {
        coord4Draw = newCoord4Draw;
}
GraphicsPath^ VesselObject::getGp() {
        return gpVessel;
}
double VesselObject::getMaxHp() {
        return maxHp;
}
void VesselObject::setMaxHp(double newMaxHp) {
        maxHp = newMaxHp;
}
double VesselObject::getRemainHp() {
        return remainHp;
}
void VesselObject::setRemainHp(double newHp) {
        remainHp = newHp;
        if (remainHp < 0.0)
                remainHp = 0.0;
        if (remainHp > maxHp)
                remainHp = maxHp;
}
double VesselObject::getMaxVesselSpeed() {
        return maxVesselSpeed;
}
void VesselObject::setMaxVesselSpeed(double newMaxVesselSpeed) {
        maxVesselSpeed = newMaxVesselSpeed;
}
double VesselObject::getMaxAttackDistance() {
        return maxAttackDistance;
}
```

```
void VesselObject::setMaxAttackDistance(double newMaxAttackDistance) {
	maxAttackDistance = newMaxAttackDistance;
}
double VesselObject::getAttackCD() {
	return attackCD;
}
void VesselObject::setAttackCD(double newAttackCD) {
	attackCD = newAttackCD;
}
double VesselObject::getLaunchCD() {
	return launchCD;
}
void VesselObject::setLaunchCD(double newLaunchCD) {
	launchCD = newLaunchCD;
}
double VesselObject::getMaxDefenseDistance() {
	return maxDefenseDistance;
}
void VesselObject::setMaxDefenseDistance(double newMaxDefenseDistance) {
	maxDefenseDistance = newMaxDefenseDistance;
}
double VesselObject::getDefenseCD() {
	return defenseCD;
}
void VesselObject::setDefenseCD(double newDefenseCD) {
	defenseCD = newDefenseCD;
}
double VesselObject::getDamage() {
	return damage;
}
void VesselObject::setDamage(double newDamage) {
	damage = newDamage;
}
double VesselObject::getShellSpeed() {
	return shellSpeed;
}
void VesselObject::setShellSpeed(double newShellSpeed) {
	shellSpeed = newShellSpeed;
```

```
}
double VesselObject::getTorpedoDamage() {
        return torpedoDamage;
}
void VesselObject::setTorpedoDamage(double newTorpedoDamage) {
        torpedoDamage = newTorpedoDamage;
}
double VesselObject::getTorpedoSpeed() {
        return torpedoSpeed;
}
void VesselObject::setTorpedoSpeed(double newTorpedoSpeed) {
        torpedoSpeed = newTorpedoSpeed;
}
double VesselObject::getRemainAttackCD() {
        return remainAttackCD;
}
void VesselObject::setRemainAttackCD(double newRemainAttackCD) {
        remainAttackCD = newRemainAttackCD;
        if (remainAttackCD < 0.0)
                remainAttackCD = 0.0;
}
void VesselObject::minus1RemainAttackCD() {
        remainAttackCD--;
        if (remainAttackCD < 0.0)
                remainAttackCD = 0.0;
}
double VesselObject::getRemainLaunchCD() {
        return remainLaunchCD;
}
void VesselObject::setRemainLaunchCD(double newRemainLaunchCD) {
        remainLaunchCD = newRemainLaunchCD;
        if (remainLaunchCD < 0.0)
                remainLaunchCD = 0.0;
}
void VesselObject::minus1RemainLaunchCD() {
        remainLaunchCD--;
        if (remainLaunchCD < 0.0)
                remainLaunchCD = 0.0;
```

```
}
double VesselObject::getRemainDefenseCD() {
	return remainDefenseCD;
}
void VesselObject::setRemainDefenseCD(double newRemainDefenseCD) {
	remainDefenseCD = newRemainDefenseCD;
	if (remainDefenseCD < 0.0)
		remainDefenseCD = 0.0;
}
void VesselObject::minus1RemainDefenseCD() {
	remainDefenseCD--;
	if (remainDefenseCD < 0.0)
		remainDefenseCD = 0.0;
}
double VesselObject::getNowSpeed() {
	return nowSpeed;
}
void VesselObject::setNowSpeed(double newNowSpeed) {
	nowSpeed = newNowSpeed;
}
double VesselObject::getNowAngle() {
	return nowAngle;
}
void VesselObject::setNowAngle(double newNowAngle) {
	nowAngle = newNowAngle;
}
void VesselObject::setCanMove(bool newCanMove) {
	canMove = newCanMove;
}
bool VesselObject::getCanMove() {
	return canMove;
}
void VesselObject::initializeMove() {
	canMove = true;
	mouseIn = false;
}
void VesselObject::SetLaserAngle(double _angle) {
	return;
```

```
}
double VesselObject::GetLaserAngle() {
	return 0.0;
}
void VesselObject::setIsLaserShooting(bool newLaserShooting) {
	return;
}
bool VesselObject::getIsLaserShooting() {
	return false;
}
void VesselObject::setLaserCD(double newLaserCD) {
	return;
}
double VesselObject::getLaserCD() {
	return 0.0;
}
void VesselObject::setRemainLaserCD(double newRemainLaserCD) {
	return;
}
void VesselObject::minus1RemainLaserCD() {
	return;
}
double VesselObject::getRemainLaserCD() {
	return 0.0;
}
void VesselObject::setEnrichValue(double _EnrichValue) {
	return;
}
double VesselObject::getEnrichValue() {
	return 0;
}
void VesselObject::setEnrichValue_CD(double _EnrichValueCD) {
	return;
}
double VesselObject::getEnrichValue_CD() {
	return 0;
}
void VesselObject::setEnrichValue_dis(double _EnrichValue_dis) {
        return;
}
double VesselObject::getEnrichValue_dis() {
        return 0;
}
void VesselObject::set_remain_Enrich_CD(double _remain_Enrich) {
        return;
}
double VesselObject::get_remain_Enrich_CD() {
        return 0;
}
void VesselObject::minus1remain_Enrich_CD() {
        return;
}
void VesselObject::setIsLaserHitSomething(bool newIsLaserHitSomething) {
        return;
}
bool VesselObject::getIsLaserHitSomething() {
        return false;
}
void VesselObject::setLaserLog(String^ newLaserLog) {
        return;
}
void VesselObject::strcatLaserLog(String^ newLaserLog) {
        return;
}
String^ VesselObject::getLaserLog() {
        return "";
}
#pragma endregion
bool VesselObject::BoundaryJudgment(double _X, double _Y) {
        if (_X < 0.0 || _Y < 0.0 || _X > 20.0 || _Y > 20.0)
                return false;
        else
                return true;
}
void VesselObject::VesselMove() {
        if (nowSpeed == 0.0 || canMove == false)
        return;
    if (nowSpeed < 0) {
        nowSpeed = nowSpeed * (-1);
        nowAngle += 180;
    }
    double dx;
    double dy;
    if (BoundaryJudgment(getCoord().X, getCoord().Y)) {
        dx = Math::Cos(ANGLE_TO_RADIUS(nowAngle));
        dy = -(Math::Sin(ANGLE_TO_RADIUS(nowAngle)));
        double dis = Math::Sqrt((dx * dx) + (dy * dy));
        setCoord(PointF(nowSpeed * dx + getCoord().X, nowSpeed * dy + getCoord().Y));
        setCoord4Draw(COORD_TO_DRAWING_COORD(getCoord().X, getCoord().Y));
    }else {
        nowSpeed = 0.0;
        nowAngle = 0.0;
        if (coord.X < 0.0)
            coord.X = 0.01;
        if (coord.Y < 0.0)
            coord.Y = 0.01;
        if (coord.X > 20.0)
            coord.X = 20.0;
        if (coord.Y > 20.0)
            coord.Y = 20.0;
        setCoord4Draw(COORD_TO_DRAWING_COORD(getCoord().X, getCoord().Y));
    }
}
void VesselObject::DaoTuiLu() {
    nowSpeed = -nowSpeed;
    VesselMove();
    nowSpeed = -nowSpeed;
}
bool VesselObject::render(Graphics^ g) {
    // draw name
    g->DrawString(name, gcnew Font("Consolas", 8), gcnew SolidBrush(team == A ? Color::Red : Color::Blue),
PointF(coord4Draw.X + BIAS_X_4_DRAW_NAME, coord4Draw.Y - BIAS_Y_4_DRAW_NAME));
    #pragma region draw HP strip
    const double BIAS_X_4_HPSTRIP = BLOCK_LENGTH - 3.0;
```

```
	const double BIAS_Y_4_HPSTRIP = 18.5;
	double stripBaseLen = BIAS_X_4_HPSTRIP + BIAS_X_4_HPSTRIP;
	double stripValueLen = stripBaseLen * (remainHp / maxHp);
	Pen^ penHpStrip_base = gcnew Pen(Color::Black, 5.0);
	Pen^ penHpStrip_value = gcnew Pen((team == A ? Color::Red : Color::Blue), 5.0);
	g->DrawLine(penHpStrip_base, PointF(coord4Draw.X - BIAS_X_4_HPSTRIP, coord4Draw.Y - BIAS_Y_4_HPSTRIP), PointF(coord4Draw.X + BIAS_X_4_HPSTRIP, coord4Draw.Y - BIAS_Y_4_HPSTRIP));
	g->DrawLine(penHpStrip_value, PointF(coord4Draw.X - BIAS_X_4_HPSTRIP, coord4Draw.Y - BIAS_Y_4_HPSTRIP), PointF(coord4Draw.X - BIAS_X_4_HPSTRIP + stripValueLen, coord4Draw.Y - BIAS_Y_4_HPSTRIP));
	#pragma endregion
	if (mouseIn) {
		Pen^ penAttackRange = gcnew Pen(Color::CornflowerBlue, 2.0);
		Pen^ penDefenseRange = gcnew Pen(Color::Brown, 2.0);
		Brush^ bruAttackRange = gcnew SolidBrush(Color::FromArgb(50, 20, 30, 40));
		Brush^ bruDefenseRange = gcnew SolidBrush(Color::FromArgb(50, 240, 10, 40));
		double atkDis = COORD_TO_DRAWING_COORD(maxAttackDistance, 0.0).X;
		double defDis = COORD_TO_DRAWING_COORD(maxDefenseDistance, 0.0).X;
		g->FillEllipse(bruAttackRange, (float)(coord4Draw.X - atkDis), (float)(coord4Draw.Y - atkDis), (float)(atkDis * 2.0), (float)(atkDis * 2.0));
		g->FillEllipse(bruDefenseRange, (float)(coord4Draw.X - defDis), (float)(coord4Draw.Y - defDis), (float)(defDis * 2.0), (float)(defDis * 2.0));
		g->DrawEllipse(penAttackRange, (float)(coord4Draw.X - atkDis), (float)(coord4Draw.Y - atkDis), (float)(atkDis * 2.0), (float)(atkDis * 2.0));
		g->DrawEllipse(penDefenseRange, (float)(coord4Draw.X - defDis), (float)(coord4Draw.Y - defDis), (float)(defDis * 2.0), (float)(defDis * 2.0));
	}
	return true;
}
// LV only
bool VesselObject::DrawLaser(Graphics^ g) {
	return false;
}
// AH only
bool VesselObject::isEnrichVessel(double _x, double _y) {
	return false;
}
```