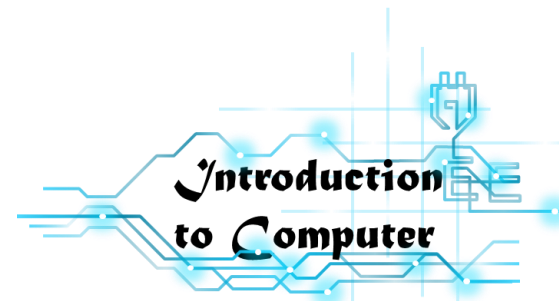


Little Tank

TA: 電子一 謝明倫

yans@media.ee.ntu.edu.tw

2014/05/21



Contents

- Quick Start

- 如果時間不多, 可以看這裡就好

- 內容說明

- 基本的程式介紹

- 詳細說明

- 比較進階的說明

```
@ : wall
. : missile
Tank: > v < ^
```

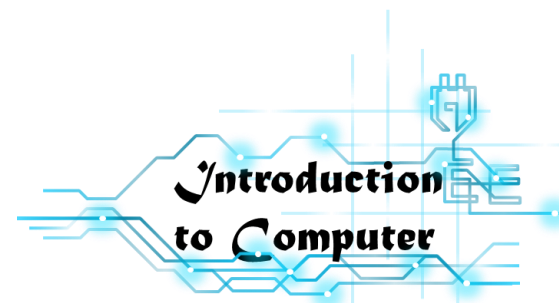
```
Green (you)
Team[0] has 1 tank(s)
Starting life:5
Yellow(opponent)
Team[1] has 1 tank(s)
Starting life:5
Red (3th party)
Team[2] has 3 tank(s)
Starting life:2
Random agent
Time: 1000 -> 0
```

```
@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
@           @ @ @ @ @
@           > . . . @
@           ^           v
@           @           @
@ @ @ @ @           @ @ @
@ @ . @           @ @ @
@ @ > @           @ @ @
@ @           @ @ @
@ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
Time: 892
Tank[0]'s Life:5
Tank[1]'s Life:3
Tank[2]'s Life:2
Tank[3]'s Life:0
Tank[4]'s Life:1
Team[0] has 1 tank(s)
Team[1] has 1 tank(s)
Team[2] has 2 tank(s)
```

Missiles



Quick Start

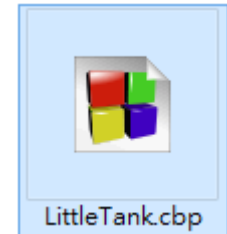


**Introduction
to Computer**

Build & run

- Windows

- 若有Code::Blocks, 直接點LittleTank.cbp編譯專案即可
- 若不想用C::B, 建專案把所有檔案丟進去編譯即可



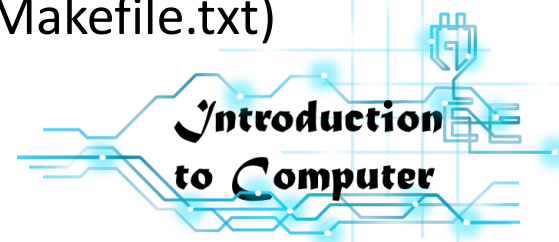
- Mac

- 依網站步驟<https://gist.github.com/cnruby/960344>，安裝ncurses

```
install ncurses on macosx
1 $ curl -O ftp://ftp.gnu.org/gnu/ncurses/ncurses-5.9.tar.gz
2 $ tar -xvzf ncurses-5.9.tar.gz
3 $ cd ./ncurses-5.9
4 $ ./configure --prefix=/usr/local \
5   --without-cxx --without-cxx-binding --without-ada --without-progs --without-curses-h \
6   --with-shared --without-debug \
7   --enable-widec --enable-const --enable-ext-colors --enable-sigwinch --enable-wgetch-events \
8 && make
9 $ sudo make install
```

- Linux

- 同上, 下載NCurses並解壓縮後，
於terminal下指令./configure，然後make。
- 依一般使用3th party library的方式編譯(可參考Makefile.txt)



遊戲畫面

Global Map

```
@@@@@@@@@@@@@@@@@
@      >  @  @  @  @
@      >  .    .  @
@      ^      @      v  @@
@      ^      @      ^  @
@@ @      @      @  @@
@@. @      @      @  @
@@>  <  @      @
@@      @      @  @
@@@@@@@@@@@@@@@@@
```

State Panel

```
Time: 892
Tank[0]'s Life:5
Tank[1]'s Life:3
Tank[2]'s Life:2
Tank[3]'s Life:0
Tank[4]'s Life:1
Team[0] has 1 tank(s)
Team[1] has 1 tank(s)
Team[2] has 2 tank(s)
```

```
Green (you)
Team[0] has 1 tank(s)
Starting life:5
Yellow(opponent)
Team[1] has 1 tank(s)
Starting life:5
Red (3th party)
Team[2] has 3 tank(s)
Starting life:2
Random agent

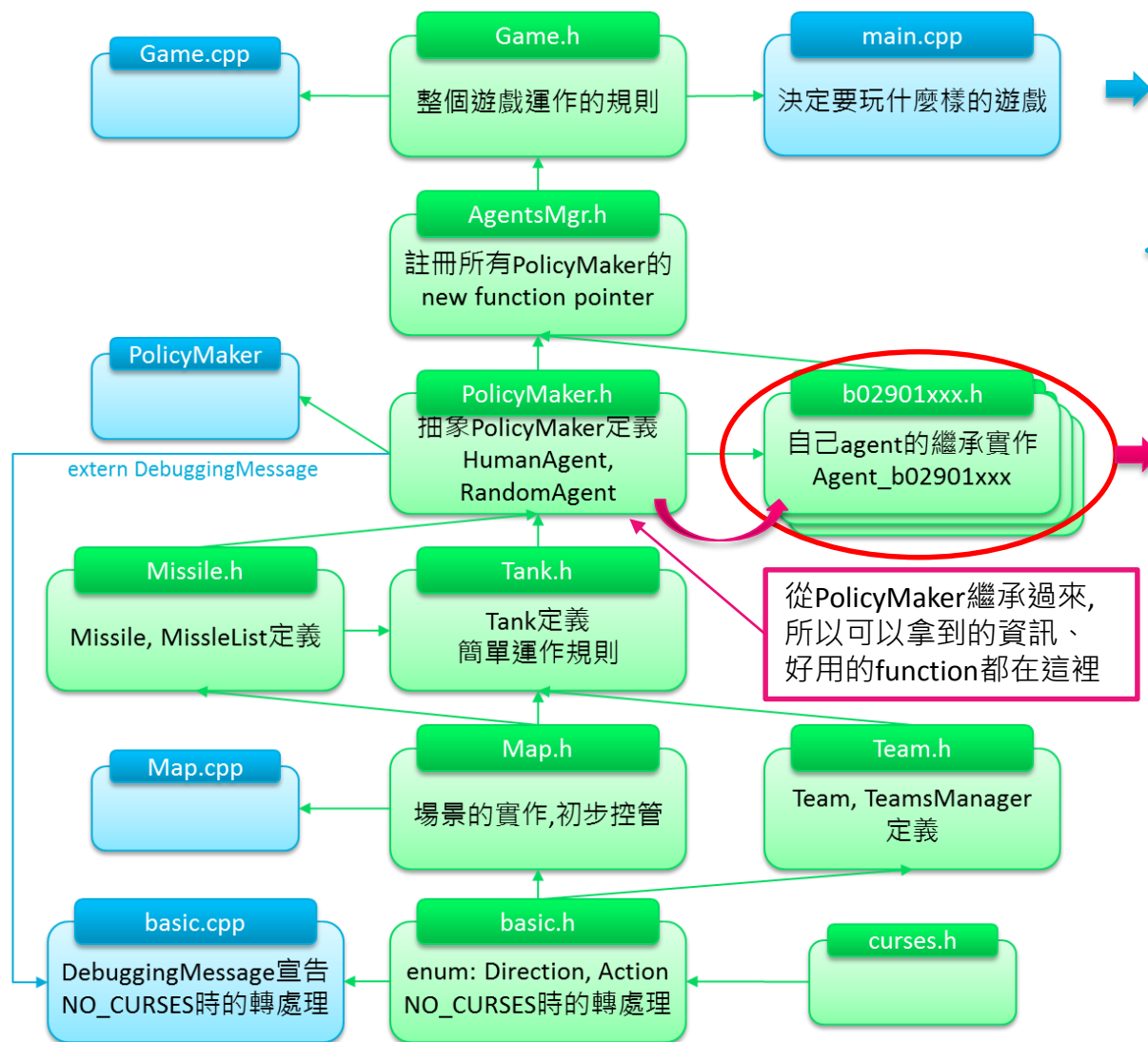
@ : wall
. : missile
Tank: > v < ^
Time: 1000 -> 0
```

一開始在humanGame模式下，以鍵盤上下左右鍵+空白鍵發射子彈，操作綠色坦克車。最主要對手是黃色坦克，另外三個紅色坦克是地圖怪物，由RandomAgent操作(血量也較少)

放紅色坦克目的在於避免有人龜縮不動，也讓躲子彈比較有意義。

The game points is calculating by: (lose: 0, match: 400, win: 900+time remain)

LittleTank程式架構



三種執行遊戲的模式

- humanGame: 人控遊戲, 幫助快速理解遊戲規則
- singleGame: 挑選AI在特定場地上進行單場對戰
- battleAll: 所有的agent交互對戰, TA評分時使用的模式(生成csv報表)

本次作業需要完成的地方, 實作自己的Agent

開始動手 – b02901xxx.h

```
1 #ifndef _b02901xxx_h_
2 #define _b02901xxx_h_
3
4 ////! TODO 1: modify the ifndef/define protection as your ID like "_b02901xxx_h_"
5
6 #include "../PolicyMaker.h"
7
8 ////! TODO 2: rename your agent class name as "Agent_b02901xxx" with your own student ID
9 class Agent_b02901xxx: public PolicyMaker{
10 public:
11 ////! TODO 3: put your student ID for the constructor of PolicyMaker (the base class)
12 // you can have argument(s), but all these argument(s) must have their default value
13 Agent_b02901xxx(): PolicyMaker("b02901xxx"){
14
15 //{ ===== you can add any member functions and datas here =====
16
17 ////! TODO 4: implement your own actionToDo function here
18 virtual Action actionToDo(int arg){
19
20
21
22
23
24
25
26 };
```

所有的b02901xxx、檔名都要改成自己的學號

繼承自PolicyMaker

PolicyMaker的constructor需要name資訊

可以自由添加class內的data或function輔助

必須要實作的地方。
每回合Game都會詢問Agent想做什麼動作，此function須return一個Action。(若是不合法的動作Game會自動無視)

Action有(6種):

noAct(不做動作),
U_Act(相當於按↑鍵), D_Act(相當於按↓鍵),
L_Act(相當於按←鍵), R_Act(相當於按→鍵),
fire(相當於按空白鍵, 發射飛彈)

如何得到場地的資訊，請見PolicyMaker的protected部分。Agent只能看見以自己為中心5x5的範圍。

Introduction
to Computer

註冊Agent – AgentsMgr.h

```
7  ///! TODO 1: put your h/cpp files in "agents" folder
8  ///! TODO 2: include your b02901xxx.h file here
9  #include "agents/b02901xxx.h"
10 #include "agents/b02901000.h"
11
12 // function pointer
13 typedef PolicyMaker* (*pfNewAgent) (void);
14
15 template<class T>
16 PolicyMaker* fNewAgent () {return new T;}
17
18 class AgentsMgr{
19 public:
20     std::vector<pfNewAgent>    pAllNewAgentFunc;
21     std::vector<std::string>    agentName;
22     int** scores;
23
24     AgentsMgr() {
25         ///! TODO 3: add your agent class "Agent_b02901xxx" in a new push_back, so TA can "new" your agent
26         pAllNewAgentFunc.push_back(&fNewAgent<RandomAgent>);
27         pAllNewAgentFunc.push_back(&fNewAgent<Agent_b02901xxx>);
28         pAllNewAgentFunc.push_back(&fNewAgent<Agent_b02901000>);
```

把你的b02901xxx.h include近來

在<>裡面填上你的class即註冊完畢

若有多個Agent可以多push_back幾個

在main裡面需要你的agent ID來進行遊戲，
而你的agent ID即為這裡push_back的順序。
以此處為例: RandomAgent為0, 你的Agent為1,
Agent_b02901000為2

main.cpp設定

```
1  #include<cstdlib>
2  #include<ctime>
3
4  #include "Game.h"
5
6  int main() {
7      Game game;
8      // TA: choose one of the following mode to start the game: humanGame, singleGame, battleAll
9
10     //game.humanGame(4,1);
11     // TA: parameters: int randSeed=4,int aiAgent=0,int viewI=0
12     // TA: suggested random seed: 4,9,80,7352,8632
13
14     game.singleGame(time(0),0,1);
15     // TA: parameters: int randSeed=4,int GreenAgent=0,int YellowAgent=1,bool showGame=true
16
17     //game.battleAll("test.csv",false);
18     // TA: parameters: const char* dumpFileName,bool showGame=false
19
20     #ifndef NO_CURSES
21     printw("Please press q to exit\n");
22     while( getch() != 'q' ) {}
23     #endif
24
25     return 0;
26 }
```

三種執行遊戲的模式 相關參數註解裡有解釋

humanGame在一開始的時候，藉由親手操作來體會整體環境的操作規則。(也確認整個專案可以build)
singleGame是當同學們AI寫好以後可以測試、比較用。
battleAll則是批改用的模式，會把所有的agent都交互比過數次，並生成.csv檔觀察結果。

評分 - battleAll

- 執行battleAll模式後可以得到.csv的分析
 - 可直接使用excel開啟

	A	B	C	D
1		0 RandomAg	b02901xxx	b02901000
2	RandomAg	2760	3372	2338
3	b02901xxx	7492	4774	6989
4	b02901000	5896	3045	4678

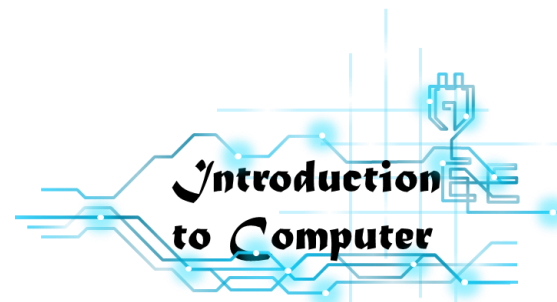
欄位解說:

這一欄代表b02901xxx與RandomAgent進行10場遊戲後,
B02901xxx所得到的積分總和。

10場遊戲為: 5場左上角綠色+同樣5個場地右下角黃色
(黃色綠色互換)

- 基本分: 只要B3-B4 \geq 500即可
 - 代表(你的Agent對戰Random)更勝於(原本的Agent對戰Random)

標準版內容說明



遊戲規則解說

- 坦克動作規則(詳見Tank.h)
 - 如果面相與Action方向不一樣，會先轉向
 - 如果面相與Action方向一樣，會向前一步
 - fire 與 [移動或轉動]都需要回復時間，兩者分開計算
 - Fire成功後要等第5回合才能再次發射Missile
 - [移動或轉動]成功後要等第2回合才能再次[移動或轉動]
 - 可以想像為: 坦克移動速度0.5，飛彈移動速度1



因為PolicyMaker有const Tank*，
你可以取用Tank所有的 public constant function
以取得現在坦克資訊。

```
int getPos() const{return position;}
int getLife() const{return life;}
int getTeamID() const{return m_team.getTeamID();}
Direction getFacing() const{return facing;}
bool isDead() const{return life<=0;}
bool isCleared() const{return position==1;}

int getmRecovering() const{return motionRecovering;}
int getfRecovering() const{return firingRecovering;}
```

Agent可以拿到的資料範圍

- 場地資訊藉由PolicyMaker的protected內容提供
 - 你可以不需要知道怎麼算出來的，知道可以怎麼用就好

```
15 class PolicyMaker{
16 private:
17     Map* theMap;
18     MissileList* theMissiles;
19     Tank** pTanks;
20     int tankNum;
21     char name[32];
22 protected:
23     const Tank* pTank;
24     Map view;
25     std::vector<Missile> MsslinView;
26     std::vector<Tank> TankinView;
27
28 void pos2xy(const int pos,int& x,int& y){
32 void getView(int rad);
33 void getMissileInView(int rad);
34 void getTankInView(int rad);
```

} 指向所有真實的場地資訊，當然不能讓大家直接使用

00 01 02 03 04
05 06 07 08 09
10 11 me 13 14
15 16 17 18 19
20 21 22 23 24

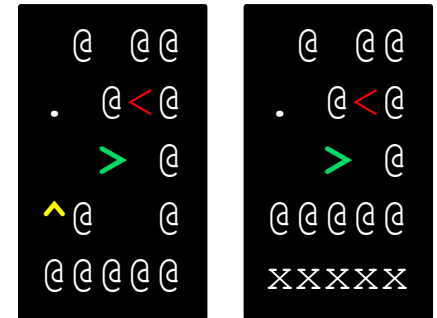
執行這些function後，會更新目前視野範圍內對應的資訊到上面相應的內容裡。

rad為視野半徑，最大可以到2
得到以自己為中心5x5的範圍



view的使用

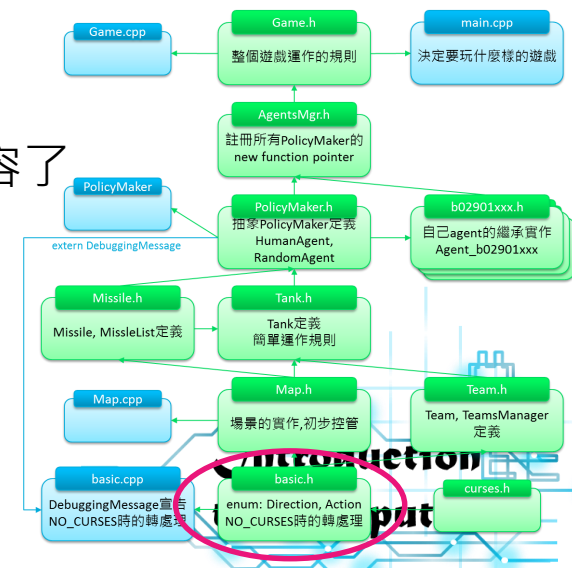
- view亦是一個Map, 相關操作可以在Map.h找到
- 最基本的用法，直接用中括號[]給參數即可
 - 可以把他想像成一個char view[25]
 - 如果在main.cpp給viewL = 2 即為這樣的視野
 - 超出邊界的地方會用x顯示



```
00 01 02 03 04
05 06 07 08 09
10 11 me 13 14
15 16 17 18 19
20 21 22 23 24
```

– 唯一的不同處: 要先用char來接

- ctype其實是int, 前面多放了一些描述顏色的資訊
- 若用char來接資料，就不會有那些無法理解的內容了



來看範例

你也可能這樣用

```
virtual Action actionToDo(int arg){
```

```
    getView(2);
```

```
    // view:
```

```
    // 00 01 02 03 04
```

```
    // 05 06 07 08 09
```

```
    // 10 11 me 13 14
```

```
    // 15 16 17 18 19
```

```
    // 20 21 22 23 24
```

```
    Direction tankInLine = noDir;
```

```
    if (view.isTank( 7)) tankInLine = U_Dir;
```

```
    else if (view.isTank(17)) tankInLine = D_Dir;
```

```
    else if (view.isTank(11)) tankInLine = L_Dir;
```

```
    else if (view.isTank(13)) tankInLine = R_Dir;
```

```
    if (tankInLine!=noDir){
```

```
        if (tankInLine==pTank->getFacing()) return fire;
```

```
    }
```

```
    DebuggingMessage = "Hello~";
```

```
    return randMove();
```

```
}
```

getView之後view才會是最新的

Map內建的功能之一

return 一種Action

你可以取用public constant function的資訊

可能會有一些自己訂的輔助用member function

這是一個Global variable (std::string)，可供debug輔助

```
switch(tankInLine){
```

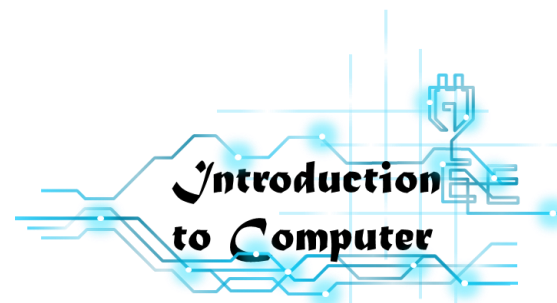
```
case U_Dir: return ( (view[enemyPos]&0xFF)=='v' )?true:false;
```

```
case D_Dir: return ( (view[enemyPos]&0xFF)=='^' )?true:false;
```

```
case L_Dir: return ( (view[enemyPos]&0xFF)=='>' )?true:false;
```

```
case R_Dir: return ( (view[enemyPos]&0xFF)=='<' )?true:false;
```

```
default: return false;
```



vector是什麼？

- 可以把他想像成“加強版的Array”

- 一樣用[]存取內容

- 多了...

- size()來看有多少東西
 - Push_back()新增內容
 - 也是用template來做的

```
std::vector<Missile> MsslinView;  
std::vector<Tank> TankinView;
```

- ...

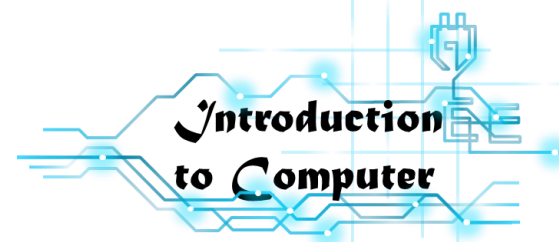
- 是個很常用也很好用的STL資料結構

- <http://www.cplusplus.com/reference/vector/vector/?kw=vector>

```
void avoidMsslpath(){  
    getMissileInView(2);  
    for(int i=0,s=MsslinView.size();i<s;++i){  
        MsslinView[i].getPos()  
        MsslinView[i].getFacing()  
    }  
}
```

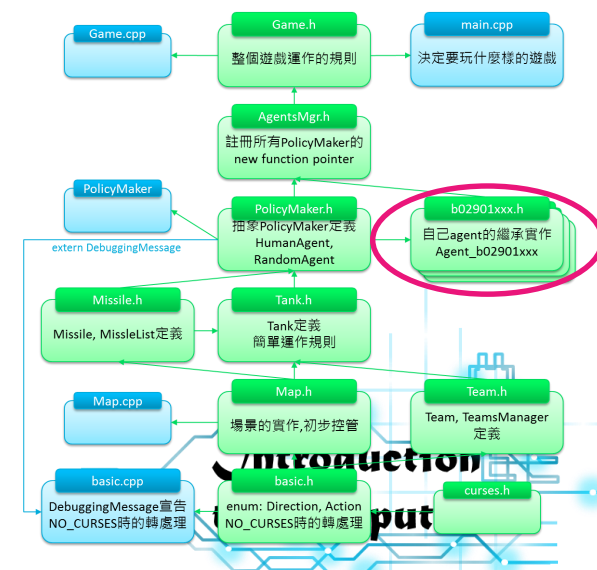
需先
更新, 一樣有視野範圍半徑 ≤ 2 限制

```
void getMissileInView(int rad);  
void getTankInView(int rad);
```



Agent記憶體的使用規範

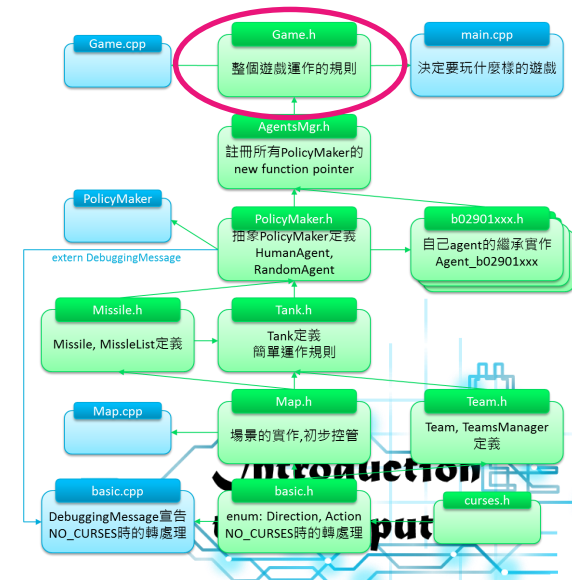
- Class內的各種public/private data/function都可以自己設定
- 但不能使用跨場的記憶
 - 以免因對戰順序而有失公平 (會互換位置比多場)
 - 不能使用static data或global變數, (static const可以)
 - 每一個單場都會new新的instance，結束後delete
- View, MsslinView跟TankinView都是複製過的副本
 - 理論上可以任意使用不怕修改



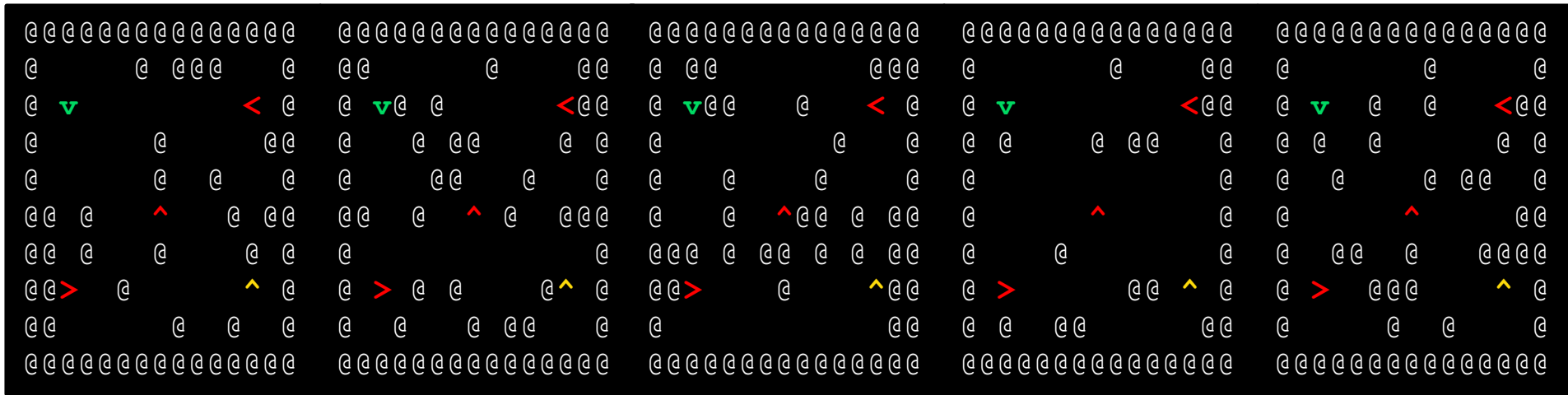
Game裡的random seed

- 地圖是亂數產生的
 - 但每次產生前都有先重設random seed
 - 是以固定的seed會對應到固定的地圖
 - 也使得agent對戰的結果會固定 (rand拿到的次序有關)
 - 不過我們會比多場 & 位置互換來確保

```
void Game::game_standard_init(int GreenAgent, int YellowAgent, int randSeed) {  
    timer = 1000;  
    srand(randSeed);  
    map.clear();  
    map.newMap();  
  
    // team 0: tankNum=1,color: green  
    // team 1: tankNum=1,color: yellow  
    // team 2: tankNum=3,color: red  
}
```



評分用的場地



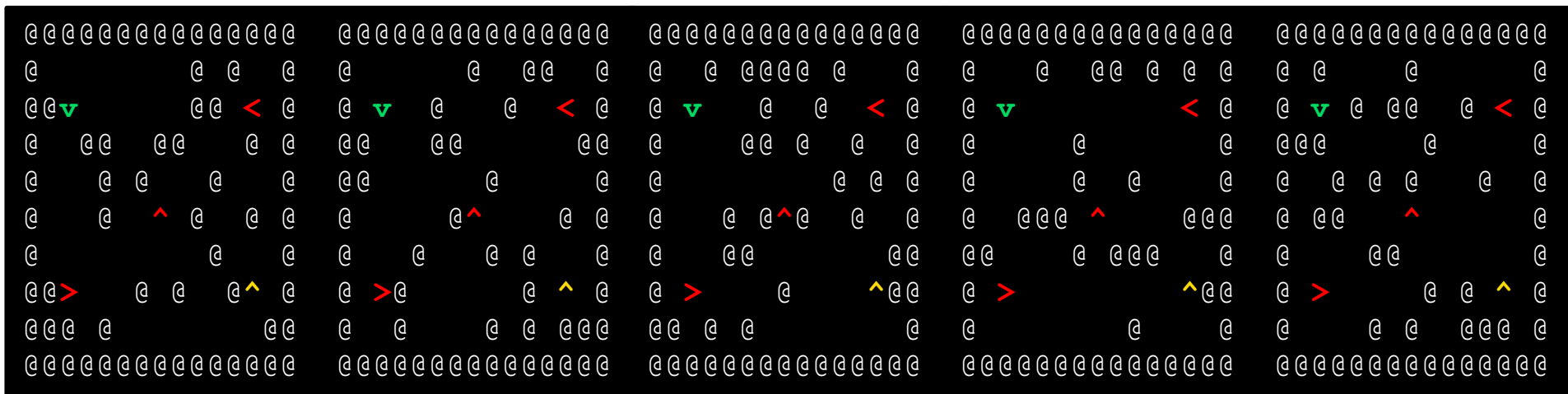
Seed 4

Seed 9

Seed 80

Seed 7352

Seed 8632



單場計分規則

State Panel

```
Time: 405
Tank[0]'s Life:3
Tank[1]'s Life:0
Tank[2]'s Life:0
Tank[3]'s Life:0
Tank[4]'s Life:0
Team[0] has 1 tank(s)
Team[1] has 0 tank(s)
Team[2] has 0 tank(s)
```

```

oooooooooooooooo
e      e  eee   e
e              e
e .      e      ee
e          e  e   e
ee e          e ee
ee<e      e      e e
ee  e          e
ee          e  e  e
oooooooooooooooo
```

```
team 0 wins!!
(team 0: 1305 p;
 team 1:    0 p )
```

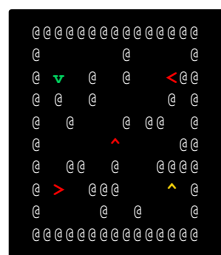
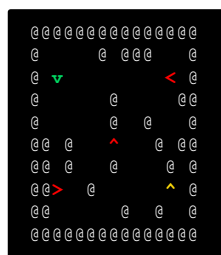
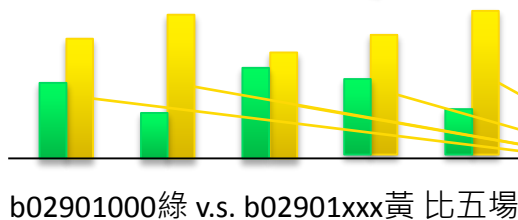
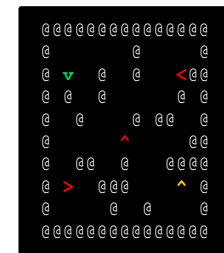
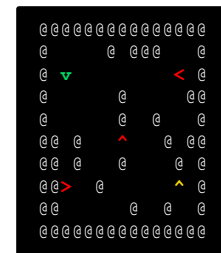
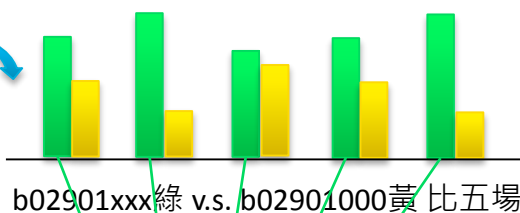
被擊毀=>這場得0分

時間倒數結束，還存活可得400分

倒數結束前擊毀所有其他對手=>得900+剩餘時間做為獎勵

battleAll的對戰表

	A	B	C	D
1		0 RandomAg	b02901xxx	b02901000
2	RandomAg	2760	3372	2338
3	b02901xxx	7492	4774	6989
4	b02901000	5896	3045	4678



b02901xxx十場得分

b02901000十場得分

共6989

共3045

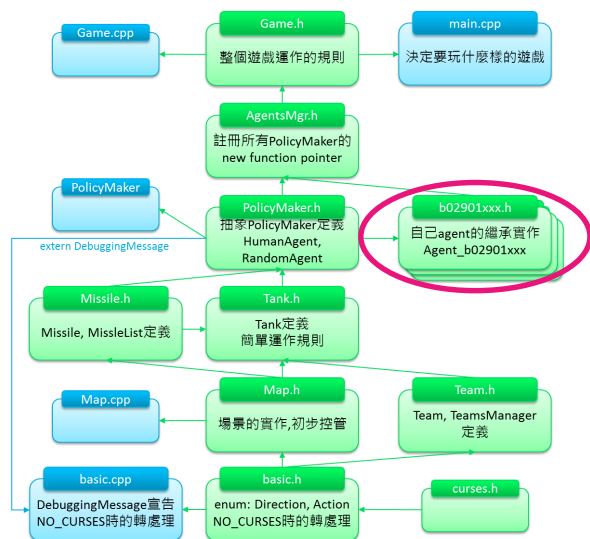
Debug 小建議

- 可以利用DebuggingMessage印出debug資訊
- 原本的程式每個cycle都會印出DebuggingMessage

– 是std::string物件

- =“xxx”來更新內容
- 只要不被更新掉就會保持下來
- 想要印多個資料可以+=“xxx\n”
- 建議可以搭配human來測試不同狀況

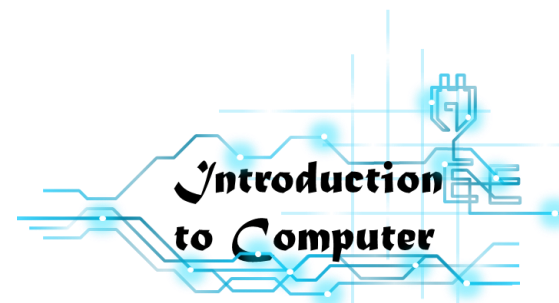
```
...
Time: 892
Tank[0]'s Life:5
Tank[1]'s Life:3
Tank[2]'s Life:2
Tank[3]'s Life:0
Tank[4]'s Life:1
Team[0] has 1 tank(s)
Team[1] has 1 tank(s)
Team[2] has 2 tank(s)
Hello~
```



```
////! TODO 4: implement your own actionToDo function here
virtual Action actionToDo(int arg){
    getView(2);
    Direction tankInLine = noDir;
    if(view.isTank(2)||view.isTank(7)) tankInLine = U_Dir;
    else if(view.isTank(22)||view.isTank(17)) tankInLine = D_Dir;
    else if(view.isTank(10)||view.isTank(11)) tankInLine = L_Dir;
    else if(view.isTank(14)||view.isTank(13)) tankInLine = R_Dir;
    if(tankInLine!=noDir){
        if(tankInLine==pTank->getFacing()) return fire;
        else return Action(tankInLine);
    }
    DebuggingMessage = "Hello~";
    return randMove();
}
```

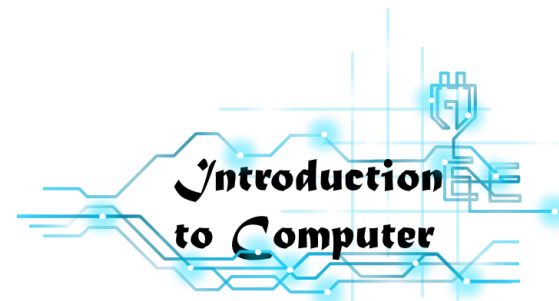
Introduction
to Computer

進階說明



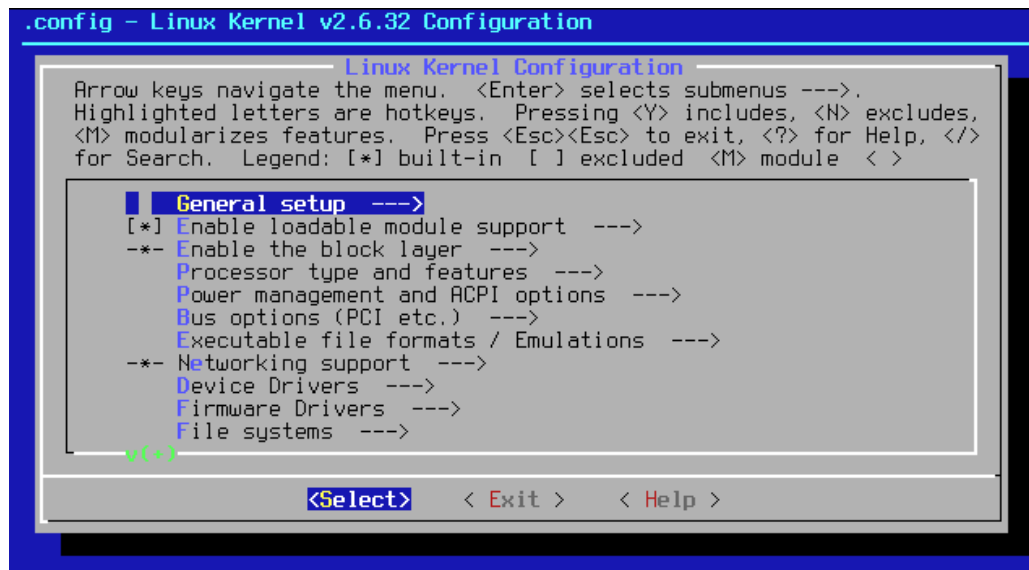
說明

- 這邊會說明一些對大一有幫助的寫程式技巧
 - 不過...那個...有些地方我沒有寫得很好，不要學
 - (尤其演算法上我都沒有優化...)
- 特別聽一些進階的用法、心法就好
- 理論上...如果你是走資派的話，
大四畢業以後程式都會寫得比我好



Curses

- 詳細的安裝說明請見CodeBlocks_3rdprtylib_pdcurese.pdf
- 圖形使用者介面(GUI)興起前，設計程式介面用
 - 稱為文字使用者介面(text user interface, TUI)



```
.config - Linux Kernel v2.6.32 Configuration

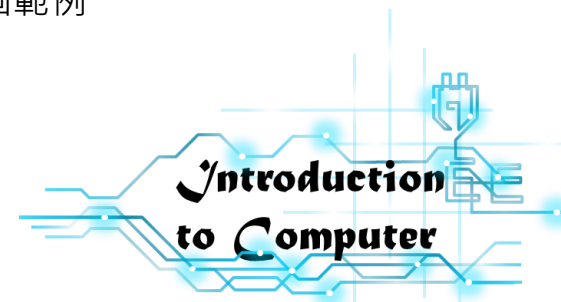
Linux Kernel Configuration
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </>
for Search. Legend: [*] built-in [ ] excluded <M> module <>

[*] General setup --->
  [*] Enable loadable module support --->
  -- Enable the block layer --->
    Processor type and features --->
    Power management and ACPI options --->
    Bus options (PCI etc.) --->
    Executable file formats / Emulations --->
  -- Networking support --->
    Device Drivers --->
    Firmware Drivers --->
    File systems --->

v(*)

<Select> < Exit > < Help >
```

一個範例



Curses function for debug

- Curses的初始化

- 顯示相關: `initscr`, `curs_set`
- 鍵盤控制相關: `keypad`, `nodelay`

```
void Game::curses_init(){  
    #ifndef NO_CURSES  
        initscr();  
        nodelay(stdscr, TRUE);  
        keypad(stdscr, TRUE);  
        curs_set(0);  
    #endif  
}
```

- 輸入輸出

- 輸入、時間控制: `nodelay`, `getch`, `napms`
- 輸出: `printw`, `mvaddch`

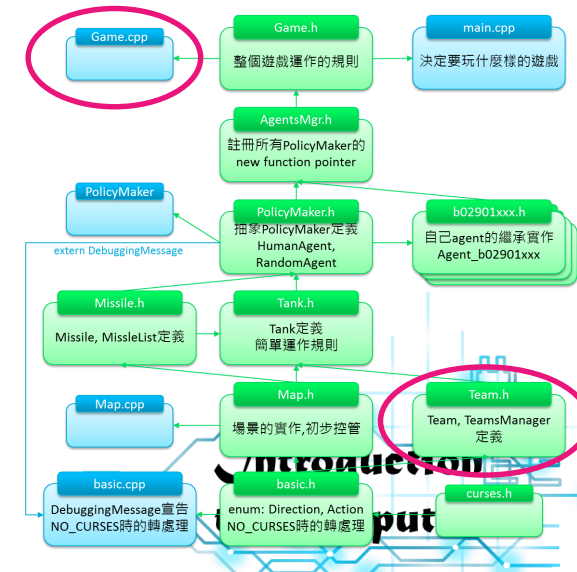
調整這個可改換頁速度

關掉的話`getch`要拿到鍵盤按鍵才會繼續

- 顏色的設定

- 相關函式: `start_color`, `init_pair`

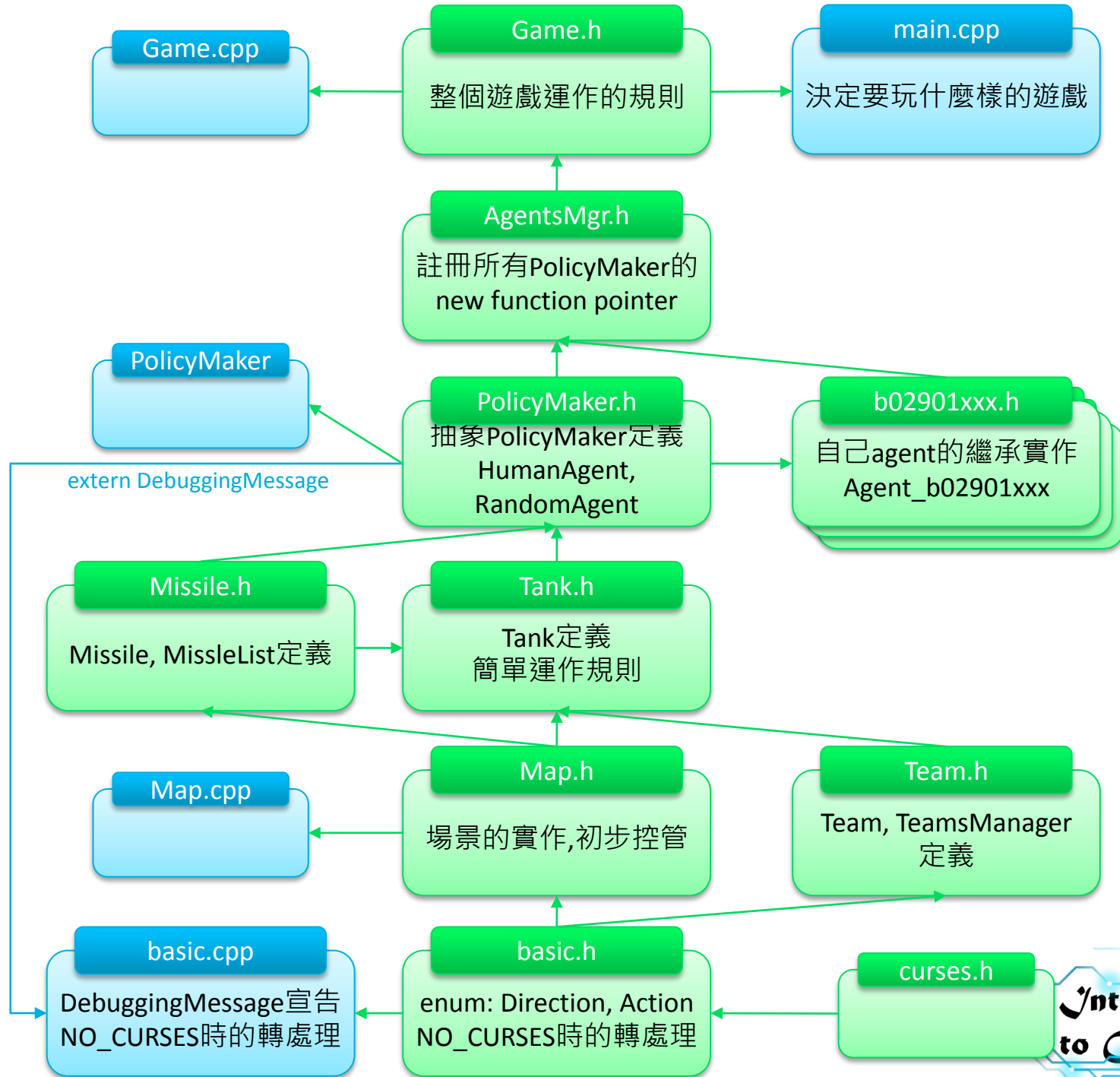
```
void initCURSES_Color(){ // called after Teams added  
    #ifndef NO_CURSES  
        start_color();  
        for(int teamID=0;teamID<teamCount;++teamID){  
            init_pair( teamID+1, Teams[teamID].curses_color, COLOR_BLACK );  
        }  
    #endif  
}  
  
int colorChar(char rawChar) const{  
    #ifndef NO_CURSES  
        return ( rawChar | COLOR_PAIR(teamID+1) );  
    #endif  
}
```



NO_CURSES

- 如果你真的裝不起來PDCurses或Ncurses
可以在basic.h解開註解，`#define NO_CURSES`
- 我有把重要的幾個curses函式用ANSI C++做個陽春版取代
 - 缺點
 - 換頁很醜...印換行來洗畫面
 - 沒有human agent可以用
 - 沒有上色功能
- 複習`#ifdef`, `#ifndef`做功能切換、debug的用途
- 要熟悉printf的用法 (curses的printw用法跟printf一樣)
 - <http://www.cplusplus.com/reference/cstdio/printf/?kw=printf>





OOP思考

- 基本上這個程式是用比較物件導向思考寫成的
 - (TA OS: 我已經回不去了><“)
- 不同功能由不同class控管
 - Class的封裝、getter, setter的安排、const很重要
 - 某些觀念的抽象化 => 繼承與多型
 - PolicyMaker的virtual function由後人實現

```
int getPos() const{return position;}
int getLife() const{return life;}
int getTeamID() const{return m_team.getTeamID();}
Direction getFacing() const{return facing;}
bool isDead() const{return life<=0;}
bool isCleared() const{return position==-1;}

int getmRecovering() const{return motionRecovering;}
int getfRecovering() const{return firingRecovering;}
```

AgentsMgr註冊原理

- 核心思考: 每一場都要new不同的agent
 - 最好可以用類似array [i]方式取用
 - 要回傳一個PolicyMaker*讓多型使用
- 用template來產生不同的function來new

```
template<class T>
PolicyMaker* fNewAgent () {return new T;}
```

- 利用function pointer array (後續可以用 [i])
 - 利用typedef方便撰寫&增加可讀性
- (若有興趣請自行google C++ function pointer)

```
// function pointer
typedef PolicyMaker* (*pfNewAgent) (void);
```



AgentsMgr註冊原理

- 把func. Template產生的func.位置存到func. pointer array中

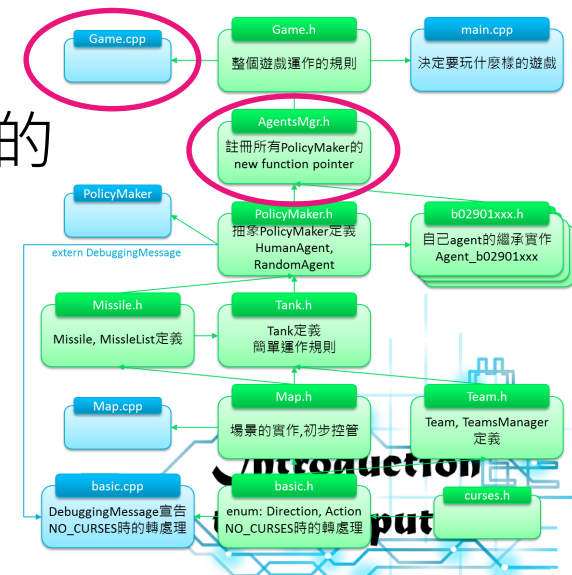
```
std::vector<pfNewAgent>    pAllNewAgentFunc;  
  
////! TODO 3: add your agent class "Agent_b02901xxx" in a new push_back, so TA can "new" your agent  
pAllNewAgentFunc.push_back(&fNewAgent<RandomAgent>);  
pAllNewAgentFunc.push_back(&fNewAgent<Agent_b98901177>);
```

- 使用時把該位置的func. pointer取出來new

```
pAgents = new PolicyMaker*[tankNum];  
pAgents[0] = (* agntsMgr.pAllNewAgentFunc[GreenAgent ] )();  
pAgents[1] = (* agntsMgr.pAllNewAgentFunc[YellowAgent] )();  
pAgents[2] = new RandomAgent;  
pAgents[3] = new RandomAgent;  
pAgents[4] = new RandomAgent;
```

- PolicyMaker的destructor記得也要是virtual的
— 才能

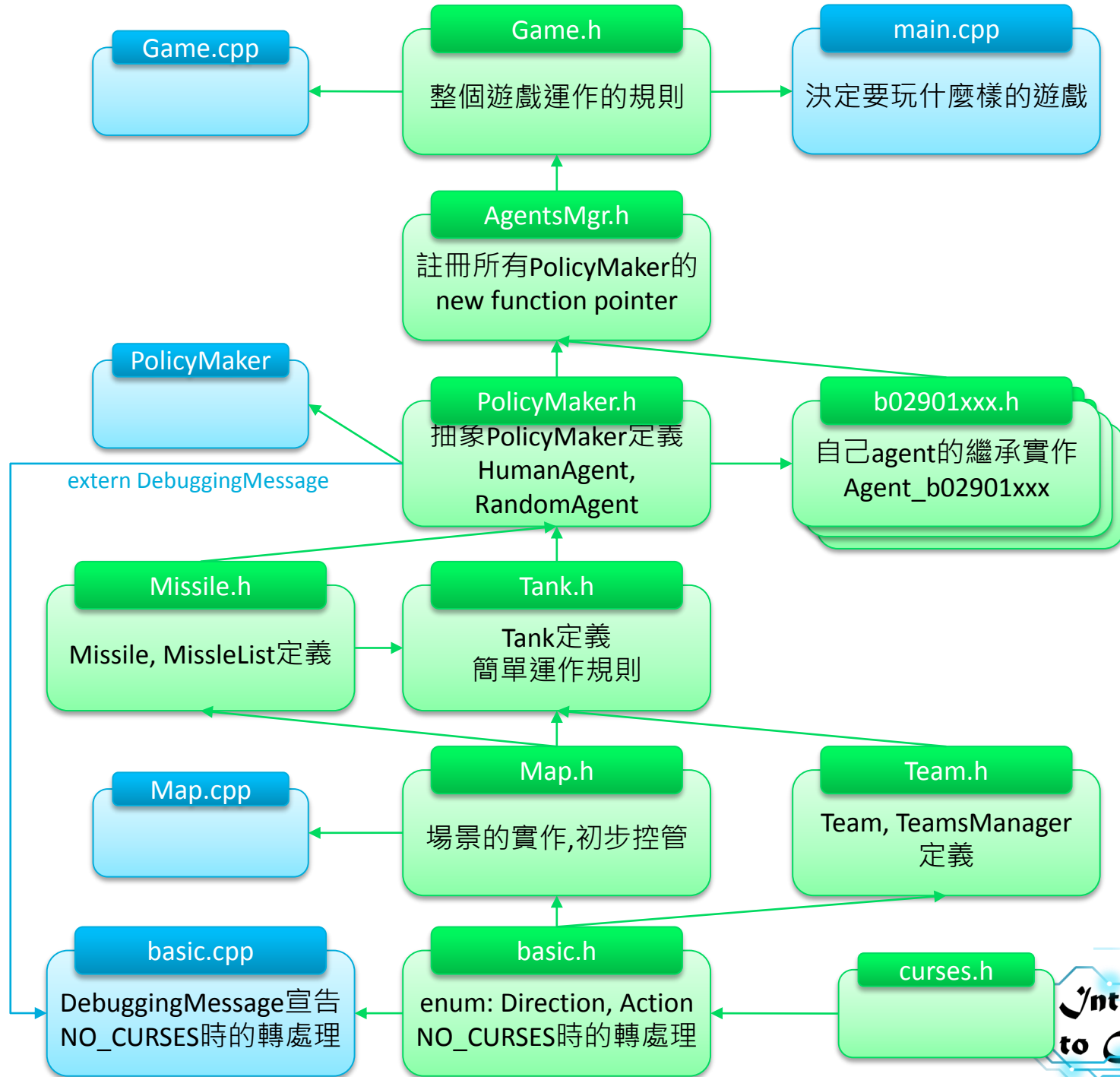
```
void Game::game_delete(){  
    teamMgr.clear();  
    for(int i=0;i<tankNum;++i){  
        delete pTanks[i];  
        delete pAgents[i];  
    }  
    delete [] pTanks;  
    delete [] pAgents;  
    tankNum = 0;  
}
```



enum使用

- 寫程式的時候常會需要一些抽象的描述/性質
 - 在以前你可能會用int 1,2,3...來定上下左右之類的
 - 高級一點你可能會用一個class包一個int
- enum就是這樣的功能
 - 可以有類似class的效果
 - Return一個Action
 - 實際上是用數字實作
 - 可以增加程式可讀性、OOP感

```
enum Direction{  
    noDir = 0,  
    U_Dir,  
    D_Dir,  
    L_Dir,  
    R_Dir  
};  
  
enum Action{  
    noAct = 0,  
    U_Act=U_Dir,  
    D_Act=D_Dir,  
    L_Act=L_Dir,  
    R_Act=R_Dir,  
    fire  
};
```

AI開發測試

- 強烈建議寫reflex agent或heuristic看一層找最好就好
 - 如果茶餘飯後有興趣(?)當然可以玩玩看寫好一點的AI
- 如果想寫search的:
 - 把地圖都看完以後背下來，遇到敵人剛好也在畫面中...
 - 甚至可以alpha-beta...不過我覺得會算超久...
- 如果想寫neurons:
 - 我覺得這個問題好像蠻適合的(?)
 - 25個input neuron接view用三層結構
 - 不過不一定會好就是
- 學習模式/使用模式 => 調參數/拿好用的參數來用

