Schedule Review

W	Date	Lecture	Homework
1	09/11, 09/15	Lec01: Course Information, Environment Introduction, and OOP Concept	Homework 00 (Fri.)
2	09/18, 09/22	Lec01: Course Information, Environment Introduction, and OOP Concept / Lec02: Class Introduction, and Essential STL Introduction	
3	09/25, **09/29**	Lec02: Class Introduction, and Essential STL Introduction / No class due to Moon Festival	Homework 01 (Mon.)
4	10/02, 10/06	Lec02: Class Introduction, and Essential STL Introduction	
5	**10/09**, 10/13	No class due to the bridge holiday of Nation day / Lec03: Encapsulation	Homework 02 (Mon.)
6	10/16, 10/20	Lec03: Encapsulation / Lec04: Inheritance	
7	10/23, 10/27	Lec04: Inheritance	Homework 03 (Mon.)
8	10/30, 11/03	Lec04: Inheritance	
9	**11/06**, **11/10**	**Physical Hand-Written Midterm** / **Physical Computer-based Midterm**	
10	11/13, 11/17	Lec05: Polymorphism	
11	11/20, 11/24	Lec05: Polymorphism	Homework 05 (Mon.)
12	11/27, 12/01	Lec05: Polymorphism	
13	12/04, 12/08	Lec06: Composition & Interface	Homework 06 (Mon.)
14	12/11, 12/15	Lec06: Composition & Interface	
15	12/18, 12/22	Lec07: Efficiency + Dependency Injection	Homework 07 (Mon.)
16	12/25, 12/29	**Physical Hand-Written Final**, Flexible time	
17	**01/01**, 01/05	No class / **Physical Computer-based Final**	
18	01/08, 01/12	**No class here**	

Object Oriented Programming

Lec03: Encapsulation

Sun Chin-Yu（孫勤昱）

cysun@ntut.edu.tw

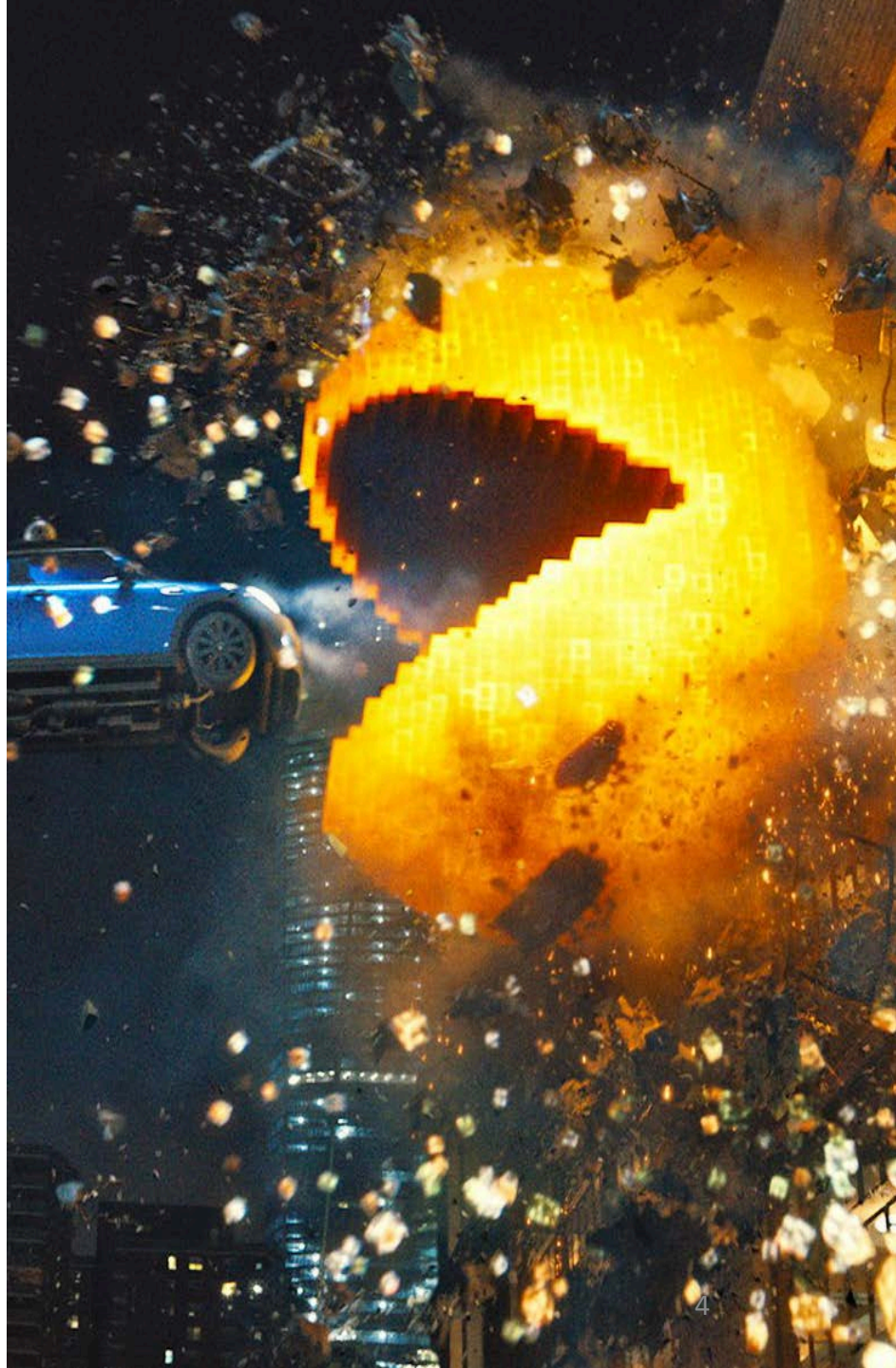
2023/09/25

大綱

- 為什麼我們會需要封裝
- 權限控制（Private、Public）
- Getter/Setter
- 補充資料

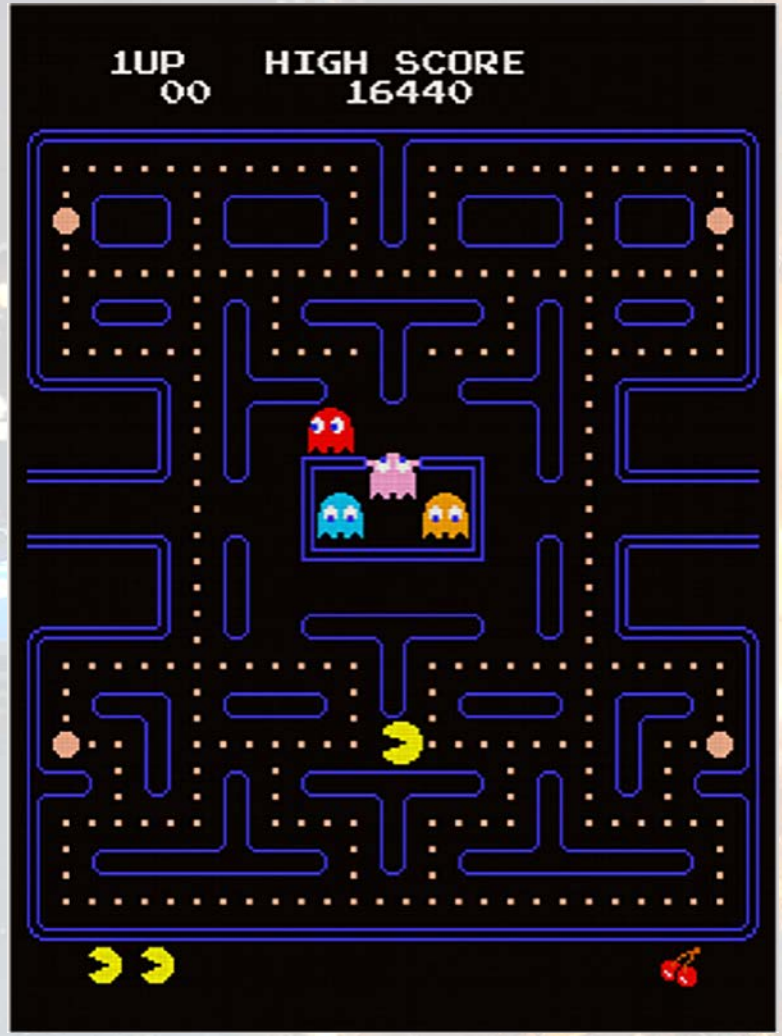
為什麼我們會需要封裝?

以Pac-Man遊戲為例



Pac-Man

- Bandai NAMCO製作，1980年風靡全球
- 遊戲概念：
  - 主角：Pac-Man
  - 四隻幽靈：Blinky、Pinky、Inky、Clyde
  - 豆子、特殊能力的豆子（暫時有能力吃掉幽靈）、加分水果
- 遊戲模式：
  - 在**有限次數**內，控制遊戲的主角吃掉迷宮內所有的豆子
  - 不能被幽靈抓到
  - 分數越高越厲害
- https://g.co/kgs/hoVyAX



https://www.newgrounds.com/art/view/toonpunch01/blinky-pinky-inky-and-clyde

使用類別來實現Pac-Man

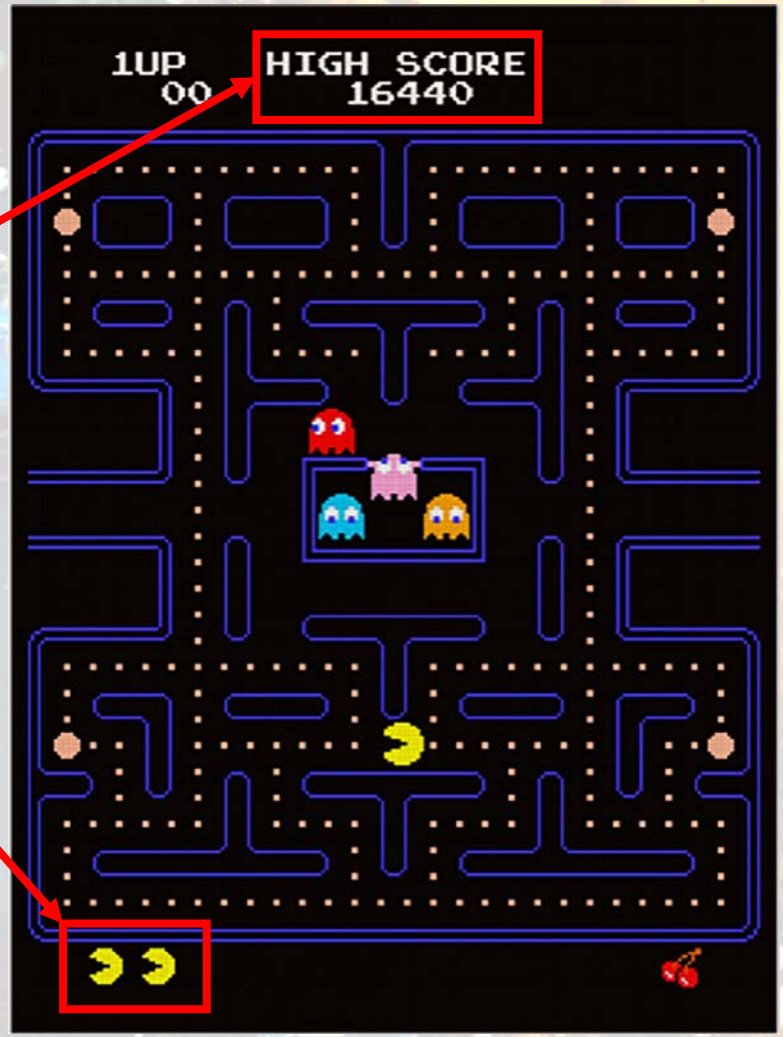
- 簡易範例（尚未導入Private/Public）

Pacman.hpp

```hpp
#ifndef PACMAN_HPP
#define PACMAN_HPP

#include <string>
#include <vector>

class Pacman {
public:
    int live = 3;
    int point = 0;

    int getPoint();
    void setPoint(int point);
    int getLive();
    void setLive(int live);
};

#endif

```

Pacman.cpp

```cpp
#include "../include/pacman.hpp"

int Pacman::getPoint(){
    return point;
}

int Pacman::getLive(){
    return live;
}

void Pacman::setLive(int live){
    this->live = live;
}

void Pacman::setPoint(int point){
    this->point = point;
}
```



Main.cpp內容

- 假設，我們預期這個類別：
- 當被幽靈抓到時：
  - 生命值（live）減少一命
- 當吃到豆子時：
  - 分數（point）增加一百分

- 這個類別滿足了我們的需求
- 但...

Pacman.hpp

```hpp
#ifndef PACMAN_HPP
#define PACMAN_HPP

#include <string>
#include <vector>

class Pacman {
public:
    int live = 3;
    int point = 0;

    int getPoint();
    void setPoint(int point);
    int getLive();
    void setLive(int live);
};

#endif

```

main.cpp

```cpp
#include "../include/pacman.hpp"

int main(){
    Pacman pacman;

    /* 如果被鬼抓到之後 ... */

    /* 第一種方法：使用直接存取成員的方式將生命值 -1 */
    pacman.live -= 1;

    /* 第二種方法：使用函數 */
    pacman.setLive(pacman.getLive() - 1);

    /* ========================================== */

    /* 如果吃到糖豆之後 */

    /* 第一種方法：使用直接存取成員的方式將分數加 100 */
    pacman.point += 100;

    /* 第二種方法：使用函數 */
    pacman.setPoint(pacman.getPoint() + 100);
}
```

設計問題（一）

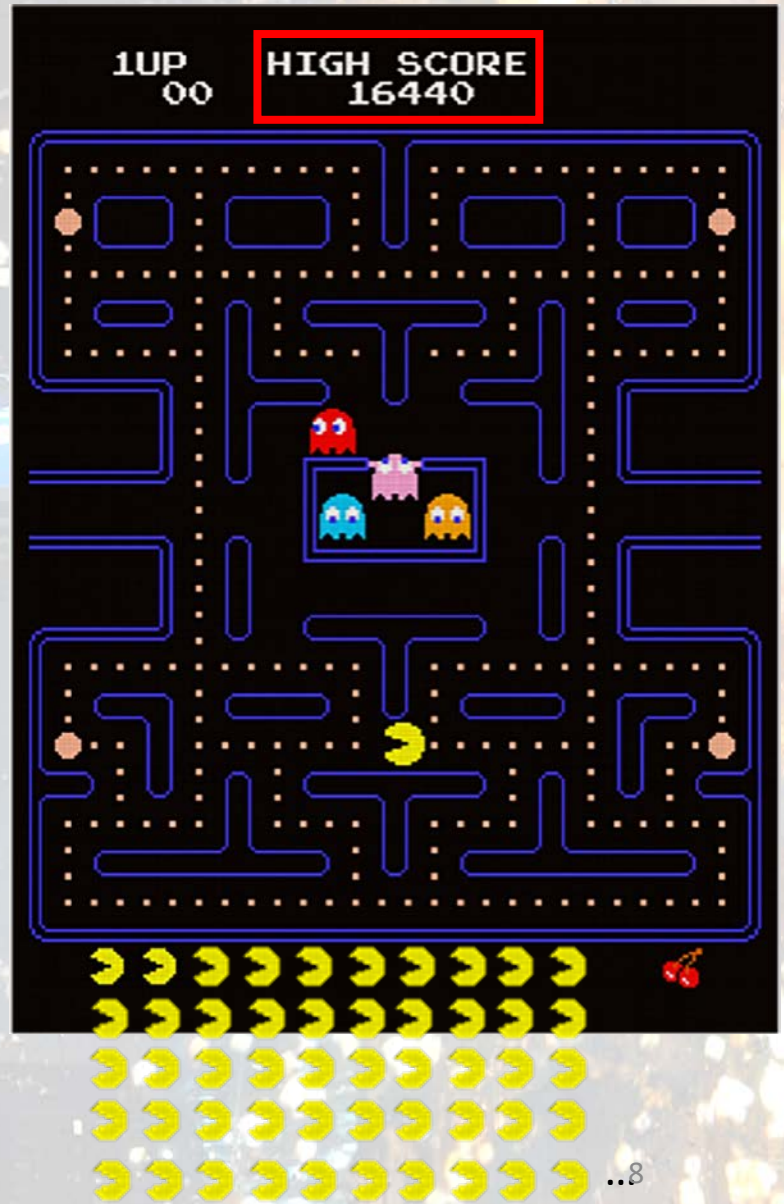
- 破壞遊戲平衡

Pacman.hpp

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  public:
9      int live = 3;
10     int point = 0;
11
12     int getPoint();
13     void setPoint(int point);
14     int getLive();
15     void setLive(int live);
16 };
17
18 #endif
19
```

直接存取成員
`pacman.live = 144`

直接使用沒有限制的Setter
`pacman.setLive(144)`



設計問題（二）

- 資訊洩漏

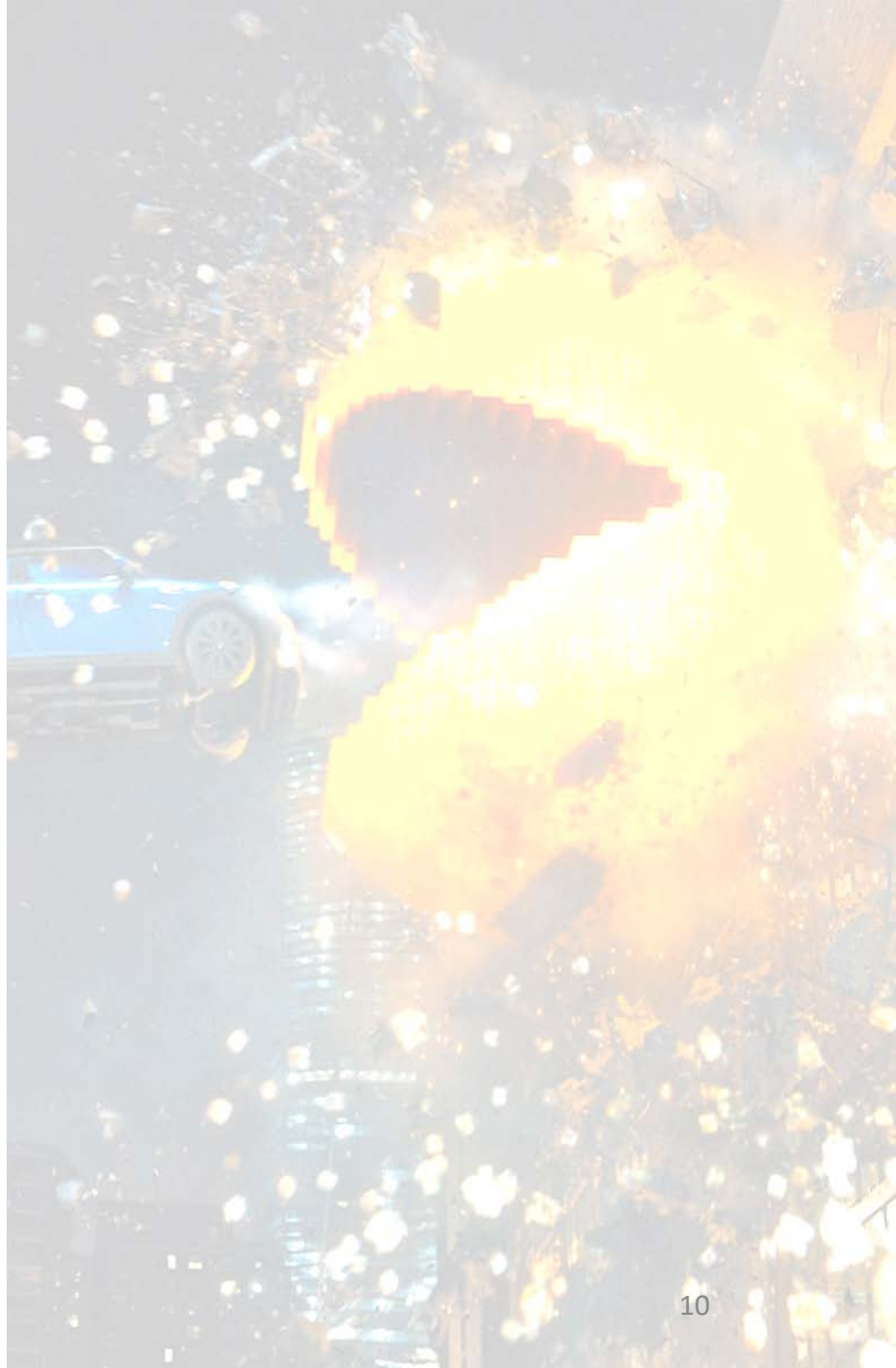
Pacman.hpp

```hpp
#ifndef PACMAN_HPP
#define PACMAN_HPP

#include <string>
#include <vector>

class Pacman {
public:
    int live = 3;
    int point = 0;

    bool specialEventActivated = false; // 特殊活動是否啟動
    std::string specialEventName = "Halloween Surprise"; // 特殊活動的名稱

    int getPoint();
    void setPoint(int value);
    int getLive();
    void setLive(int lives);
};

#endif
```

洩漏過多資訊給外部

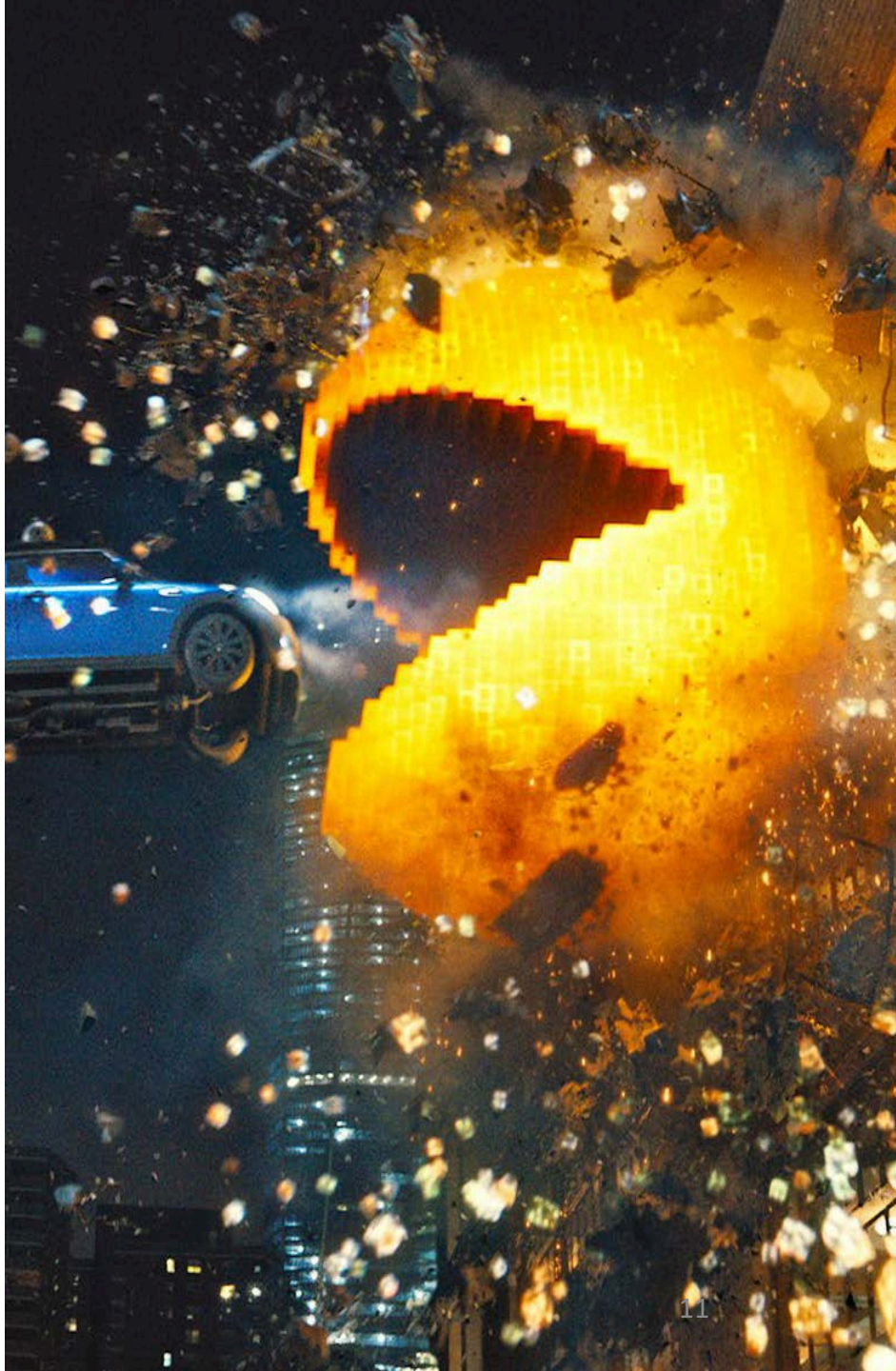
解決方法

- 那麼顯然，我們必須要做一些措施來讓這件事情不會被發生
  - 不能讓成員直接被存取
  - 若要更改值，確認丟進來的值是不是合法的
  - 對值進行特殊處理法，不要讓使用者直接更改值（思考題）



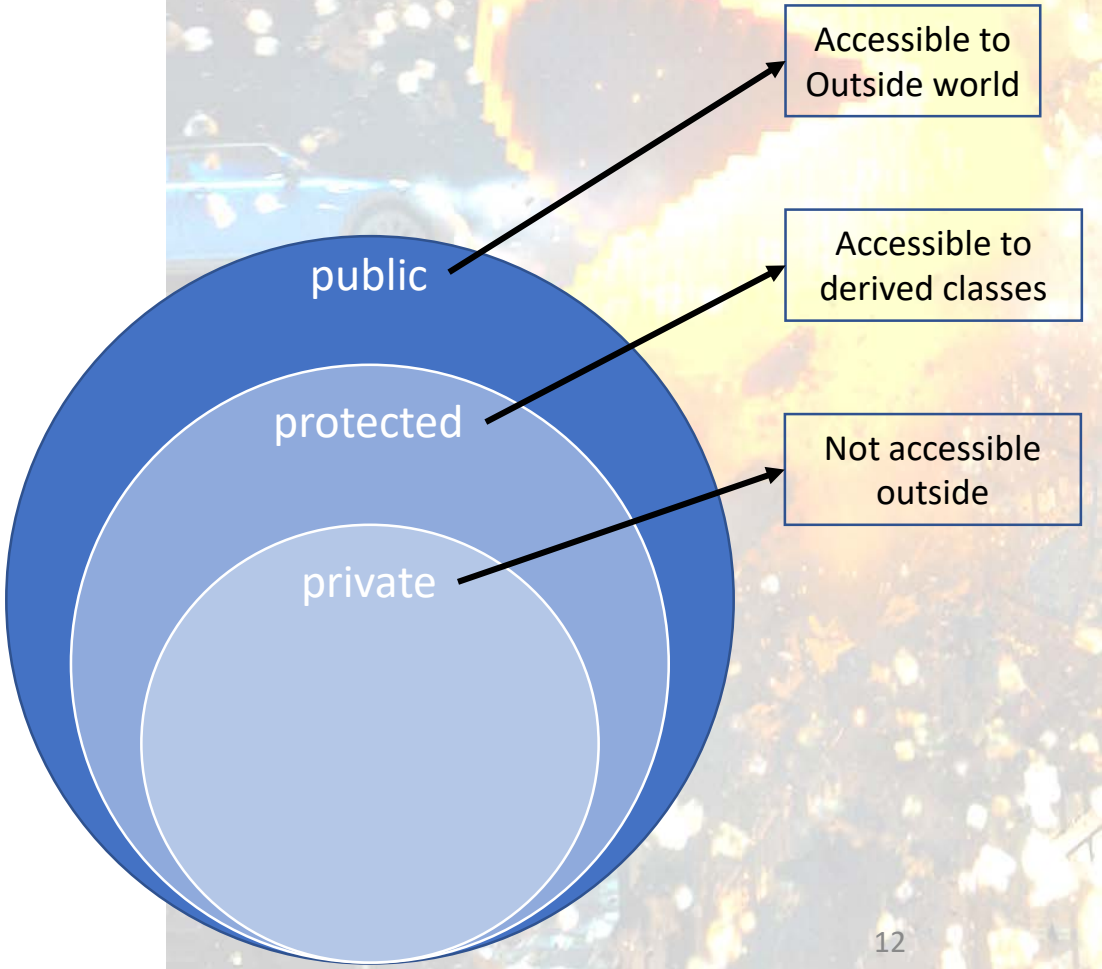
存取修飾字
(Access modifier)

Public & Private



存取修飾字

- 要做到資訊隱藏（Information hiding）可以使用存取修飾字
- 分為：
  - Public：是指對存取權限完全的公開，任何物件都可以存取
  - Private：是限制最嚴格的access modifier，只有在類別本身內部可以存取
  - ~~Protected：存取是受限的，除了類別自己可以使用外，只有其成員函式與類別的朋友（friend）以及子類別可以存取~~
    - 「繼承」的時候介紹



Case1：Public

- 為什麼我們可以直接存取成員?
  - 因為public將所有的成員能夠被外部存取
  - 在public底下的code block都能夠被外部存取

Pacman.hpp

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  public:
9      int live = 3;
10     int point = 0;
11     int getPoint();
12     void addPoints();
13     int getLive();
14     void decreaseLive();
15 };
16
17 #endif
18
```

Case2：Private

- 另一個權限控制稱為private
  - private底下的成員僅能在類別內存取，外部無法存取
- 透過右圖的修改，我們可以確保live與point無法被外部存取
  - pacman.live -> 編譯錯誤
  - pacman.point -> 編譯錯誤
- 若未指定存取修飾字，則預設是private

Pacman.hpp

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  private:
9      int live = 3;
10     int point = 0;
11 public:
12     int getPoint();
13     void addPoints();
14     int getLive();
15     void decreaseLive();
16 };
17
18 #endif
19
```

https://www.onlinegdb.com/

權限控制

- 透過`private`我們可以用來使成員**（或函式）**無法被外部直接存取
- 透過`public`我們可以用來使成員**（或函式）**能夠被外部存取

- 等等，什麼情況我們會讓函式不能被外部直接存取?

Pacman.hpp

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  private:
9      int live = 3;
10     int point = 0;
11 public:
12     int getPoint();
13     void addPoints();
14     int getLive();
15     void decreaseLive();
16 };
17
18 #endif
19
```

遊戲中的彩蛋

- 接下來我們換個例子，在這款遊戲中埋個彩蛋。

- 當有人呼叫這個bonus()的function，他應該傳入一個用於代表情況的code
  - 若code = 1則做閃爍背景
  - 若code = 2則讓Pac-Man變大
  - 若code = 3則讓Pac-Man的速度變快

- 我們先暫時忽略要考慮對應的code執行函數這件事情，實踐!

Pacman.hpp

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  private:
9      int live = 3;
10     int point = 0;
11 public:
12     int getPoint();
13     void addPoints();
14     int getLive();
15     void decreaseLive();
16     void bonus(int code);
17
18     void blinkBackground();
19     void dilatePacman();
20     void speedUpPacman();
21 };
22
23 #endif
24
```

權限控制（Bonus）

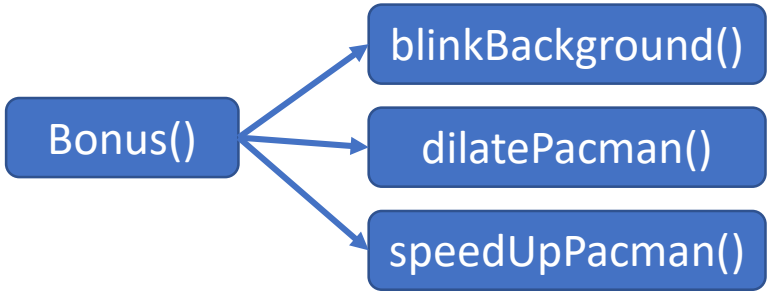
- 想想看，這些bonus function應該要被外部存取嗎?
  - 如果這些函數能夠被外部存取，似乎少了需要用code去判斷的意義?
  - 如果這些函數能夠被外部存取，似乎少了「彩蛋」的驚喜感

Pacman.hpp

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  private:
9      int live = 3;
10     int point = 0;
11 public:
12     int getPoint();
13     void addPoints();
14     int getLive();
15     void decreaseLive();
16     void bonus(int code);
17
18     void blinkBackground();
19     void dilatePacman();
20     void speedUpPacman();
21 };
22
23 #endif
24
```

權限控制（Bonus）

- 我們將這些函式視為 bonus 的子函式，也就是要有這些子函式才能造就 bonus 的函數得以實現



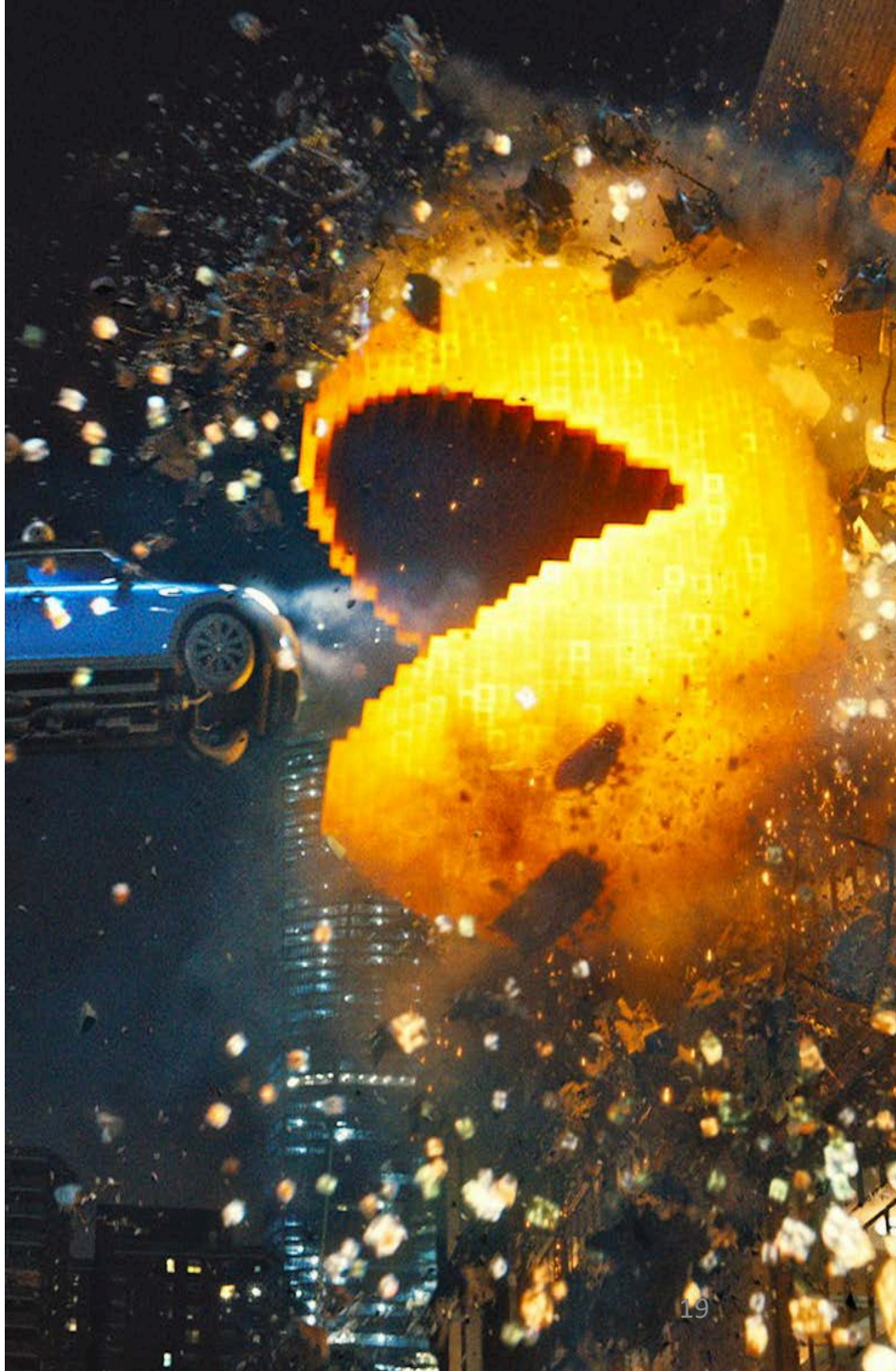
  - 我們應該只有要開放給使用者 bonus 這個函式
  - 不應該 public 這些實踐 bonus 的子函式
  - 挪到 private 來防止這些函式直接被外部存取

- 如果Pacman.hpp提供的公開接口不提到或使用Private中的成員變數與函式，則外部程式碼基本上不會知道它們的存在
  - 上述是不考慮逆向工程的情況下☺

Pacman.hpp

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  private:
9      int live = 3;
10     int point = 0;
11     void blinkBackground();
12     void dilatePacman();
13     void speedUpPacman();
14 public:
15     int getPoint();
16     void addPoints();
17     int getLive();
18     void decreaseLive();
19     void bonus(int code);
20 };
21
22 #endif
23
```

Getter/Setter



Getter/Setter的概念

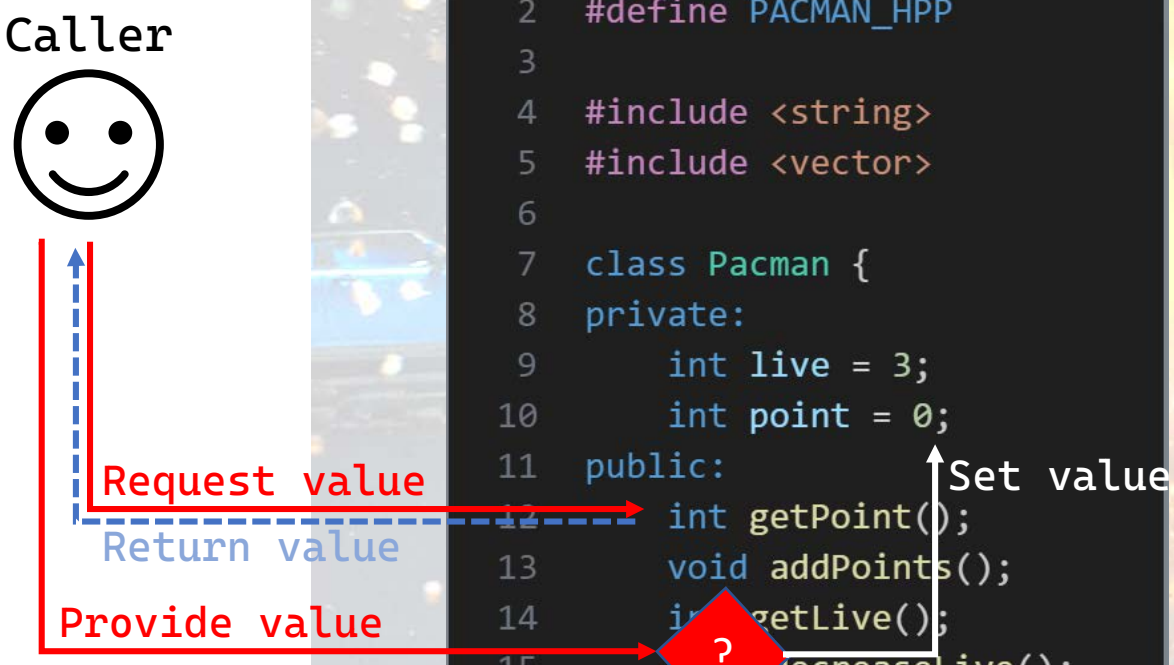
- 如果不允許外部直接存取成員
  - 則需要使用**函數**來存取
  - Private屬性下，成員變數的管家（或溝通橋梁）

- 分成兩類：
  - 從物件取得資料：Getter
    - getPoint()：取得Pacman的點數
    - getLive()：取得Pacman還有幾條命
  - 從物件修改、存取成員、執行事件：Setter
    - addPoints()：新增點數
    - decreaseLive()：減少生命數量

```
1  #ifndef PACMAN_HPP
2  #define PACMAN_HPP
3
4  #include <string>
5  #include <vector>
6
7  class Pacman {
8  private:
9      int live = 3;
10     int point = 0;
11 public:
12     int getPoint();
13     void addPoints();
14     int getLive();
15     void decreaseLive();
16 };
17
18 #endif
19
```

Getter/Setter的職責

- Getter
  - 回傳數值給Caller
- Setter
  - 檢查Caller傳入的數值會不會造成異常
  - 設定數值至成員




```
#ifndef PACMAN_HPP
#define PACMAN_HPP

#include <string>
#include <vector>

class Pacman {
private:
    int live = 3;
    int point = 0;
public:
    int getPoint();
    void addPoints();
    int getLive();
    void decreaseLive();
};

#endif
```

檢查異常情況

- 延續前個例子，在這款遊戲中埋個彩蛋

- 當有人呼叫這個bonus()的function，他應該傳入一個用於代表情況的code
  - 若code = 1則做閃爍背景
  - 若code = 2則讓Pac-Man變大
  - 若code = 3則讓Pac-Man的速度變快
  - **其他code不應該存在**

```
#ifndef PACMAN_HPP
#define PACMAN_HPP

#include <string>
#include <vector>

class Pacman {
private:
    int live = 3;
    int point = 0;
    void blinkBackground();
    void dilatePacman();
    void speedUpPacman();
public:
    int getPoint();
    void addPoints();
    int getLive();
    void decreaseLive();
    void bonus(int code);
};

#endif

```

Setter的防範性

- 延續前個例子，在這款遊戲中埋個彩蛋
- 當有人呼叫這個bonus()的function，他應該傳入一個用於代表情況的code
  - 若code = 1則做閃爍背景
  - 若code = 2則讓Pac-Man變大
  - 若code = 3則讓Pac-Man的速度變快
  - **其他code不應該存在**
- 我們先省略實踐這些子函式，單純實踐看看bonus函式
  - Getter的職責：將值回傳
  - Setter的職責：防範使用者傳入的值、設定值、執行事件
    - Setter防範的部分：收到非預期的code時，跳出例外

```cpp
#include "../include/pacman.hpp"

int Pacman::getPoint(){
    return point;
}

int Pacman::getLive(){
    return live;
}
```

Getter的職責

```cpp
void Pacman::decreaseLive(){
    this->live -= 1;
}

void Pacman::addPoints(){
    this->point += 100;
}

void Pacman::bonus(int code){
    if(code == 1){
        blinkBackground();
    }else if(code == 2){
        dilatePacman();
    }else if(code == 3){
        speedUpPacman();
    }else{
        throw NotImplementCodeException();
    }
}
```

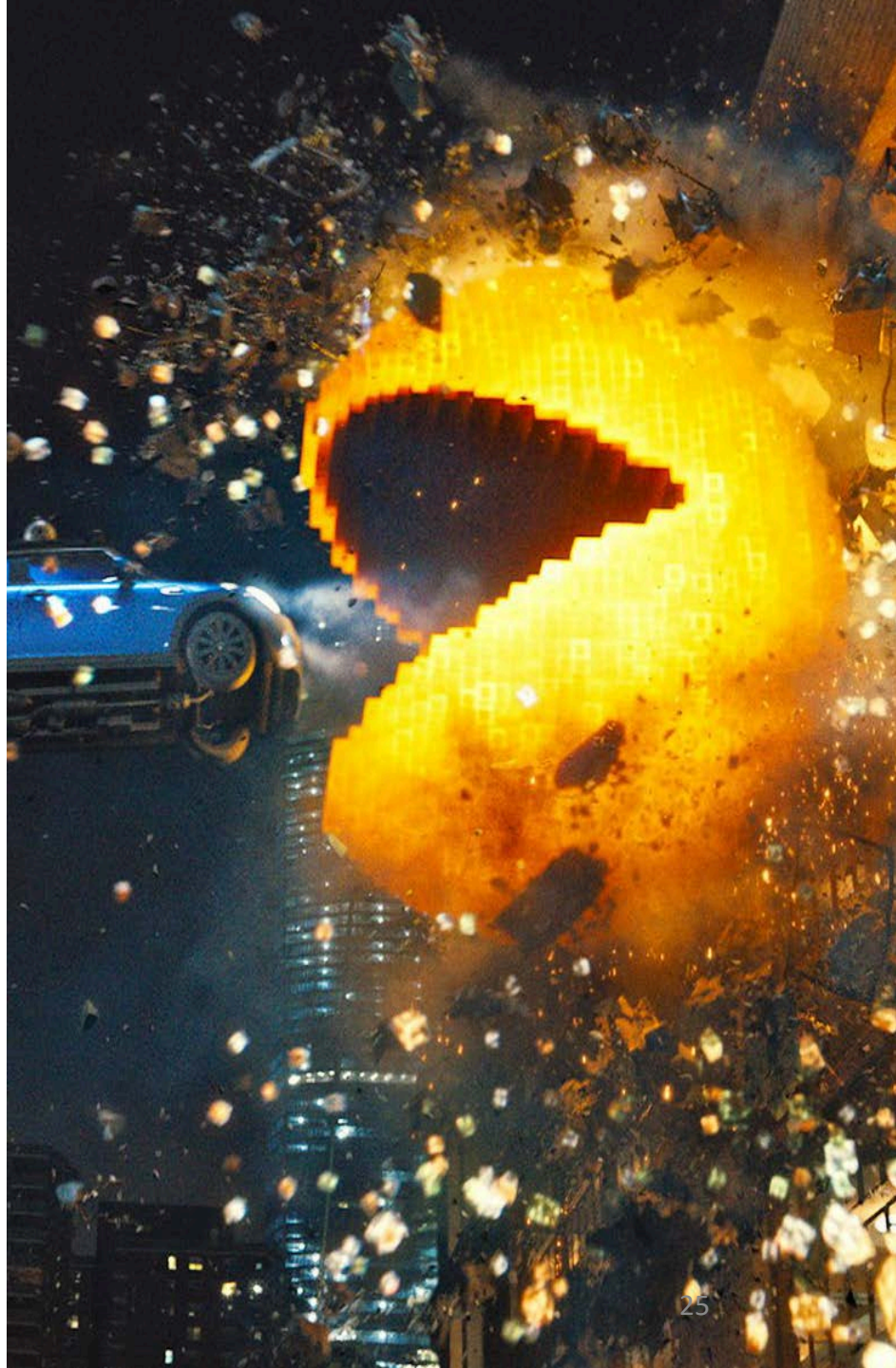
Setter的職責

Getter/Setter的小結論

- 透過Getter/Setter,
  - 我們可以明確分責函式
  - 並且易於實踐封裝的特性
  - 進而防止異常情況發生

```
#include "../include/pacman.hpp"

int Pacman::getPoint(){
    return point;
}

int Pacman::getLive(){
    return live;
}

void Pacman::decreaseLive(){
    this->live -= 1;
}

void Pacman::addPoints(){
    this->point += 100;
}

void Pacman::bonus(int code){
    if(code == 1){
        blinkBackground();
    }else if(code == 2){
        dilatePacman();
    }else if(code == 3){
        speedUpPacman();
    }else{
        throw NotImplementCodeExcep
    }
}
```

補充資料



Good Practice

- 對值進行特殊處理法，不要讓使用者直接更改值（思考題）
  - 讓 `pacman.live` 與 `pacman.point` 這樣的存取被擋下
  - 將 `setLive()` 與 `setPoint()` 使用 `decreaseLive()` 與 `addPoints()` 所替代。
  - 讓所有異常行為能夠被防範，以及能夠依然滿足需求。

```hpp
#ifndef PACMAN_HPP
#define PACMAN_HPP

#include <string>
#include <vector>

class Pacman {
private:
    int live = 3;
    int point = 0;
public:
    int getPoint();
    void addPoints();
    int getLive();
    void decreaseLive();
};

#endif

```

pacman.hpp

```cpp
#include "../include/pacman.hpp"

int Pacman::getPoint(){
    return point;
}

int Pacman::getLive(){
    return live;
}

void Pacman::decreaseLive(){
    this->live -= 1;
}

void Pacman::addPoints(){
    this->point += 100;
}
```

pacman.cpp

Bonus：如何檢驗封裝是好的。

- 會不會有異常情況沒有被考慮到?
- 寫（單元）測試來測試使用案例
  - 邊界檢查
  - 非法值
  - 可能還要有一點通靈成分
- 像極了在做紅隊或打 CTF，對吧?

```cpp
#include <gtest/gtest.h>

#include "../include/pacman.hpp"

TEST(PacmanTest, UseDecreaseLiveShouldDecrease1Live){
    Pacman pacman;
    int live = pacman.getLive();

    pacman.decreaseLive();

    ASSERT_EQ(pacman.getLive(), live - 1);
}

TEST(PacmanTest, UseAddPointsShouldAdd100Points){
    Pacman pacman;
    int point = pacman.getPoint();

    pacman.addPoints();

    ASSERT_EQ(pacman.getPoint(), point + 100);
}
```

ut_pacman.cpp

補充：struct

- C++的Class與Struct都可以：
  - 宣告成員變數與成員函式
  - 繼承、多型、建構子、介面
- 差異：
  - Struct預設屬性為Public；Class預設屬性為Private
  - Struct預設Public繼承；Class預設Private繼承
  - Struct不能用在template；Class可以用在template
- 使用時機：
  - 單純只有資料，不會針對內部成員做複雜處理的時候，使用 struct
  - 其他時候都使用 class

```cpp
#include <iostream>

// 定義一個名為 Point 的結構
struct Point {
    int x; // x 座標
    int y; // y 座標

    // 建構函數
    Point(int xVal, int yVal) : x(xVal), y(yVal) {}

    // 成員函數用於顯示點的座標
    void display() const {
        std::cout << "Point(" << x << ", " << y << ")\n";
    }
};

int main() {
    Point p1(5, 7);  // 建立一個 Point 物件並初始化其 x 和 y 值
    p1.display();   // 輸出: Point(5, 7)

    return 0;
}
```

補充：Enum

- 應該使用Enum來實作Bonus
  - 提高程式碼的可讀性
  - 確保變數只能取一組特定的值
  - 與整數相比，使用具有描述性名稱的值更具語意

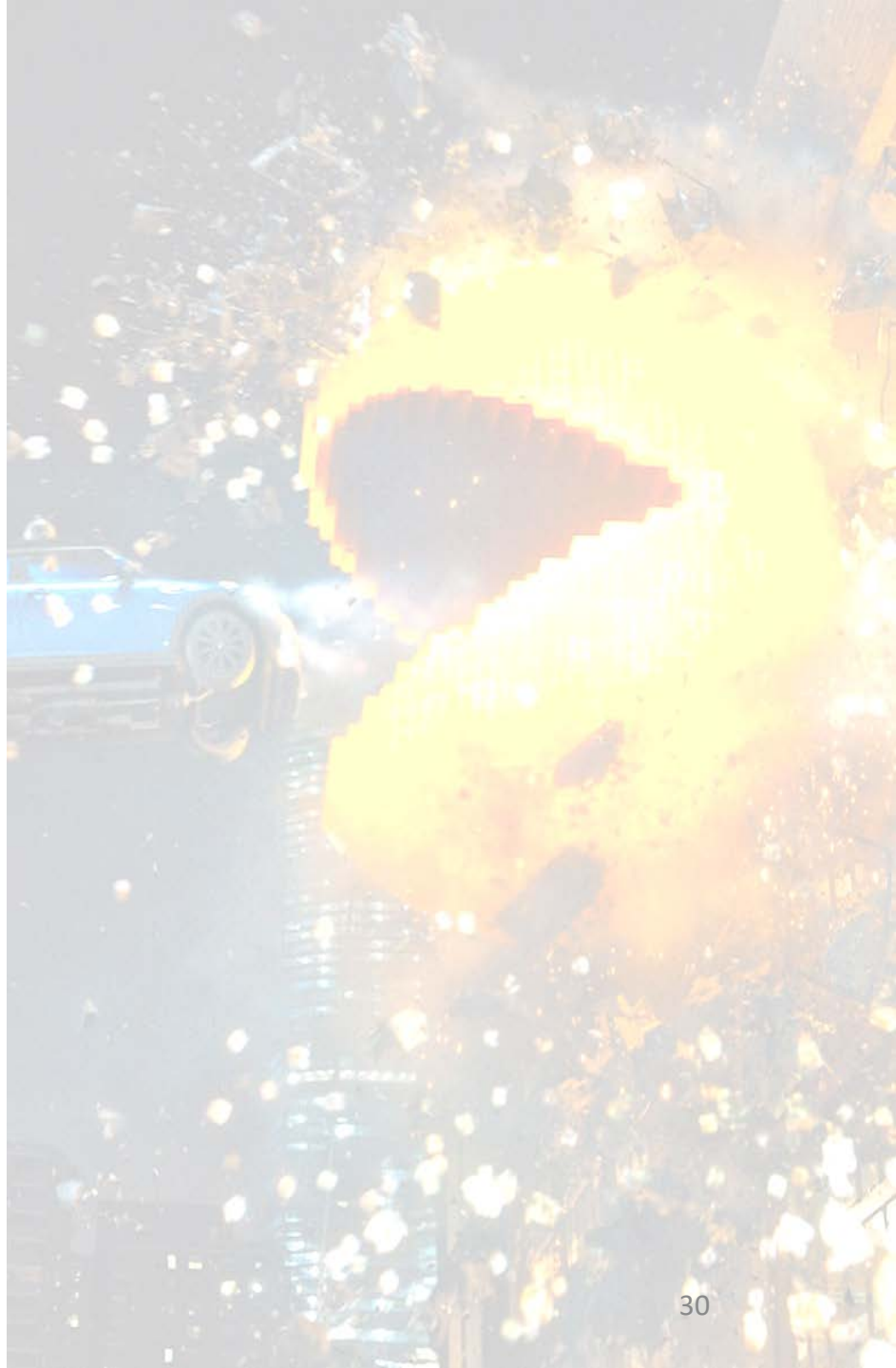
```cpp
enum BonusCode {
    BLINK_BACKGROUND = 1,
    DILATE_PACMAN,
    SPEED_UP_PACMAN
};

void Pacman::bonus(BonusCode code) {
    switch (code) {
        case BLINK_BACKGROUND:
            blinkBackground();
            break;
        case DILATE_PACMAN:
            dilatePacman();
            break;
        case SPEED_UP_PACMAN:
            speedUpPacman();
            break;
        default:
            throw NotImplementCodeException();
    }
}
```

```cpp
#include "../include/pacman.hpp"

int Pacman::getPoint(){
    return point;
}

int Pacman::getLive(){
    return live;
}

void Pacman::decreaseLive(){
    this->live -= 1;
}

void Pacman::addPoints(){
    this->point += 100;
}

void Pacman::bonus(int code){
    if(code == 1){
        blinkBackground();
    }else if(code == 2){
        dilatePacman();
    }else if(code == 3){
        speedUpPacman();
    }else{
        throw NotImplementCodeException();
    }
}
```

封裝（Source: 臺大開放式課程）

- 類別將狀態變數與功能函式封裝起來
  - 當作一個整理來思考運用
  - 使類別內容的修改更容易
  - 把類別變成一個黑箱（Black box）模組
    - Programmer不用瞭解成員變數與成員函式的細節，就能應用類別的功能
    - 程式碼重複使用（Code reuse）

Private



封裝（ Source: 臺大開放式課程）

- 硬體IC的封裝
  - 內部的電路封裝在內，不輕易顯露
  - 電子電路工程師只要在接腳施加正確的電壓電流訊號，IC就能正常運作
  - 大大減輕電子電路工程師的負擔

- 軟體類別上的封裝
  - 軟體工程師不需要了解private成員變數與成員函式
  - 只要懂得public成員的使用，也能正確應用既有類別程式
  - 減輕工作負荷，增進效能

https://en.wikipedia.org/wiki/Integrated_circuit

封裝（Source: 臺大開放式課程）

- 通常狀態變數為private
  - 只供同類別之功能函式取用
  - 不可以由類別外直接取用
- 若有必要讓外界取得或設定其值
  - 通常另外再寫public成員函式處理
  - 避開同名變數問題
  - 避免被不慎改動，破壞狀態的一致性
  - 容易維護，不影響外界程式
- 功能函式可能Public亦可能Private
- 一個類別至少有一個公開函式，以便應用

封裝 (Source: 臺大開放式課程)

```
1  public int width;
2  public int height;
3  public int area;
4
5  Rectangle a = new Rectangle();
6  a.width = 3;
7  a.height = 5;
8  a.area = a.width * a.height;
```

- 問題(一)：程式邏輯中，或許計算Area不需要給別人看不想（保護細節）
- 問題(二)：程式碼的Area計算方式，或許不是width*height

封裝 (Source: 臺大開放式課程)

```
1  private int width;
2  private int height;
3  private int area;
4
5  Rectangle a = new Rectangle();
6  a.width = 3;
7  a.height = 5;
8  a.area = a.width * a.height;
```

- 若改成private
  - Line 6 – 8 編譯不會過

封裝（Source: 臺大開放式課程）

```
1  private int width;
2  private int height;
3  private int area;
4
5  Public SetWidth(int w){
6      width = w;
7  }
8  Public SetHeight(int h){
9      height = h;
10     area = width * height;
11 }
12 Public GetArea(){
13     return area;
14 }
15
16 Rectangle a = new Rectangle();
17 a.SetWidth(3);
18 a.SetHeight(5);
19 a.GetArea();
```

- 要新增Setter/Getter
- 不能有SetArea
  - 避免被User破壞狀態的一致性

https://en.cppreference.com/w/cpp/utility/initializer_list
https://en.cppreference.com/w/cpp/utility/optional

Thanks!