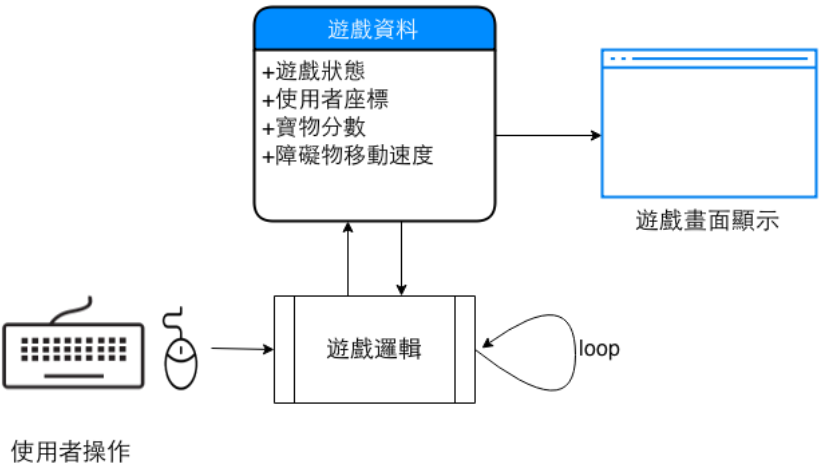
遊戲相關概念

2015 資訊之芽語法班 suhorng

小遊戲概念 (一般/粗略而言)

1. 遊戲邏輯
2. 遊戲資料
3. 使用者操作
4. 畫面生成
5. 資源控管

小遊戲概念 (一般/粗略而言)



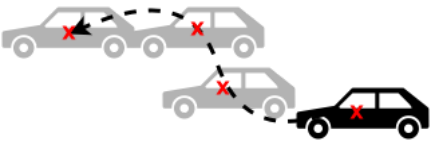
小遊戲概念: 遊戲資料 (example)

- 遊戲中每個東西所需要的屬性 資料
  - 例如磚塊就有長寬、位置 (、移動速度)
- 程式一開始時(或遊戲開始玩), 要**把一開始的資料設定好**
- 遊戲資料與畫面顯示是抽離的
- **想像中**、**簡化後** 的真實物件的比擬(?)
- 可能有整個遊戲的狀態

小遊戲概念: 遊戲邏輯

- 雖然概念上可能是連續的, 但...

- 取樣點跟電腦的計算是離散的



小遊戲概念: 遊戲邏輯 (example)

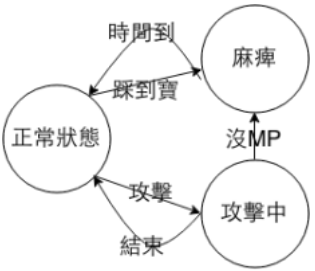
- 控制程式運作, 每個取樣點更新一次 (cf. 小遊戲 Extra). 使用迴圈 (或小遊戲(一)中用 timer)

```
while (game_running) {
    // ...
    DelaySomeTime();
}
```

```
void SystemTimer(int value){
    // ...
    glutTimerFunc(25, SystemTimer, 1);
}
```

- 依據收到的鍵盤/滑鼠輸入, 使用者與各個物件的狀態, 決定當下的行為與每個人的變動
    - 物體的移動, 狀態更新
    - 攻擊/生命值的更新

遊戲邏輯相關知識: 自動機 & 狀態




```
const int ST_NORMAL = 0, ST_PARALYZED = 1, ST_ATTACKING = 2;
int user_state = ST_NORMAL, user_time = 0;

// 程式迴圈
while (...) {
  if (user_state == ST_NORMAL) { // 處理輸入 ⇒ 移動
    if (IsInBox(x, y, block_x, block_y)) user_state = ST_PARALYZED;
  } else if (user_state == ST_PARALYZED) {
    if (user_time > 10) user_state = ST_NORMAL; // 結束麻痺
    else                ++user_time;
  } else if (user_state == ST_ATTACKING) { // 處理敵人生命值
    if (user_mp < 0) user_state = ST_PARALYZED;
  }
}
```

其他

- 使用者操作
  - 滑鼠, 鍵盤, ...
  - 看函式庫
- 畫面生成
  - 看函式庫
- 資源控管
  - 目前無須顧慮

案例回顧: 打磚塊; 遊戲資料 (concept)

- 磚塊設定(長寬) 球的設定(移動速度) 擊球板設定

```
float ball_vx, ball_vy;
float bar_vx, bar_vy, bar_width, bar_height;
float block_width, block_height;
```

- 遊戲物件相關資料

```
float ball_x, ball_y;
float bar_x, bar_y;
float block_x[100], block_y[100];
int block_cnt;
```

- 遊戲狀態

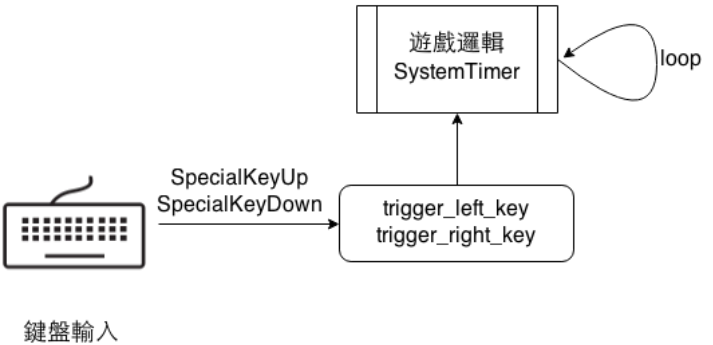
```
int game_state;
const int GAME_STATE_PLAY = 0;
const int GAME_STATE_END = 1;
```

案例回顧: 打磚塊; 遊戲資料

- 遊戲一開始時, 把各個物件就定位 ⇒ 設定好一開始玩的時候的資料

```
void Init(){
    bar_x = 0, bar_y = -0.5;
    bar_vx = 0.05, bar_vy = 0;
    bar_width = 0.3, bar_height = 0.015;

    ball_vx = 0.01, ball_vy = 0.012;
    ball_x = 0, ball_y = 0;

    block_width = 0.18, block_height = 0.09;
    block_cnt = 24;
    for(int lx = 0;lx < 4;lx++)
        for(int ly = 0;ly < 6;ly++)
            block_x[lx*6 + ly] = -0.5 + ly*0.2 ,
            block_y[lx*6 + ly] = 0.5 - lx*0.1;

    game_state = GAME_STATE_PLAY;
}
```

案例回顧: 打磚塊; 遊戲邏輯

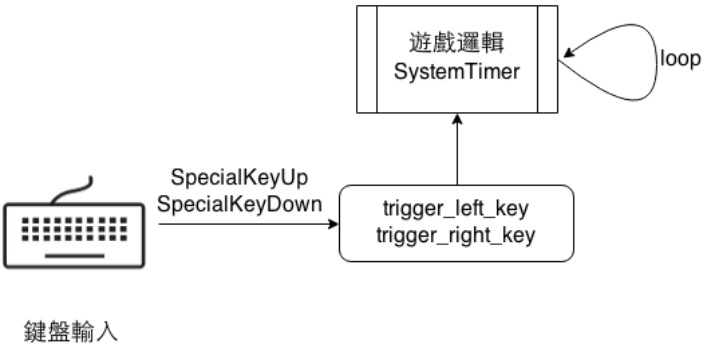


案例回顧: 打磚塊; 遊戲邏輯 (concept)

- 每個 sample 時間點, 檢視各個操作, 更新遊戲狀態

```
void SystemTimer(int value){
    if(game_state == GAME_STATE_PLAY){
        ball_x += ball_vx; ball_y += ball_vy;

        if(IsLegalPoint(ball_x + ball_vx, ball_y) == false){
            BlockDelete(ball_x + ball_vx, ball_y);
            ball_vx = -ball_vx;
        }

        if(IsLegalPoint(ball_x, ball_y + ball_vy) == false){
            BlockDelete(ball_x, ball_y + ball_vy);
            ball_vy = -ball_vy;
        }

        if(trigger_left_key) bar_x -= bar_vx;
        if(trigger_right_key) bar_x += bar_vx;

        if(ball_y < -1) game_state = GAME_STATE_END;

    } else if(game_state == GAME_STATE_END);
    // ...
```

案例回顧: 打磚塊; 使用者操作



案例回顧: 打磚塊; 使用者操作

```
bool trigger_left_key, trigger_right_key;

void SpecialKeyDown(int key, int x, int y){
    if(key == GLUT_KEY_LEFT) trigger_left_key = true;
    if(key == GLUT_KEY_RIGHT) trigger_right_key = true;
    glutPostRedisplay();
    return;
}

void SpecialKeyUp(int key, int x, int y){
    if(key == GLUT_KEY_LEFT) trigger_left_key = false;
    if(key == GLUT_KEY_RIGHT) trigger_right_key = false;
    glutPostRedisplay();
    return;
}

int main(int argc, char* argv[]){
    trigger_left_key = false, trigger_right_key = false;
    // ...
    glutKeyboardFunc(NormalKeyDown);
    glutKeyboardUpFunc(NormalKeyUp);
    glutSpecialFunc(SpecialKeyDown);
    glutSpecialUpFunc(SpecialKeyUp);
    // ...
```

案例回顧: 打磚塊; 畫面生成

- 依照遊戲資料, 把各個遊戲物件繪製到螢幕上

```
void Display(){
    glClear(GL_COLOR_BUFFER_BIT);

    if(game_state == GAME_STATE_PLAY){
        float wid = 0.05;
        DrawRectangle(ball_x, ball_y, wid, wid, 0.5, 0.5, 0.5);
        DrawRectangle(bar_x, bar_y, bar_width, bar_height,
                      0.5, 0.7, 0.7);

        for(int lx = 0;lx < block_cnt;lx++)
            DrawRectangle(block_x[lx], block_y[lx],
                          block_width, block_height,
                          0.5, 0.3, 0.3);

    }else if(game_state == GAME_STATE_END){
        // print GAME_END
        // ...
    }

    glFlush();
}
```

實例演練: pacman

1. 搞清楚這個遊戲怎麼玩 有什麼元素 有哪些東西
2. 遊戲資料 ⇒
    - 找到每個東西有哪些屬性要存 有什麼設定 (位置? 速度? 長寬? 被吃掉後的分數? ...)
    - 是否有整個遊戲相關的屬性 (幾條命? 遊戲現在狀態? 場面上有幾隻鬼? ...)
3. 遊戲邏輯的判斷與資料更新
4. 從遊戲資料繪製出螢幕畫面的程式 (計算每個物件的座標? 繪製的先後順序?)
5. 使用者的操作方式 (處理輸入的函式部份大同小異)

未來發展

- 資料封裝方式
  - 物件導向 ⇒ ...
  - 函數導向 ⇒ ...
- 使用者操作/螢幕繪圖/... 各個不同函式的包裝方式(e.g. 物件導向 ⇒ 每個物件提供外框 繪製方法等, 與畫布物件互動, ...)
  - 如何使用遊戲引擎?
- Functional Reactive Programming (?)
- 簡單遊戲也可以很好玩 agar.io

未來發展

- 作業出的各個東西都是超級簡化版 實際上...
- 遊戲邏輯: 更好的描述邏輯? 可抽換的邏輯(使用腳本(語言))? 連線遊戲? 記錄遊戲狀態?
- 使用者操作: 靈敏度? 介面設計? 不同的輸入來源? 防外掛?
- 畫面生成: 速度? 硬體支援? 特效/真實度? 3D 模型的顯示? 畫面的更新(髒矩形)? 避免閃爍?
- 資源控管: 控管記憶體用量? 各個資源是否總是需要存在記憶體中? 圖片的壓縮?