Project 1：Doodle jump

F84096192 葉翔恆、E64082246 宋俊熙

一、概念介紹

在遊戲架構的發想中，我們將大方向分為 doodle 和樓梯道具物件。首先，利用隨機函數預先隨機生成各物件的位置，再來設計 doodle 的跳躍，在 doodle 跳躍的過程中不斷將背景下拉，而樓梯物件也會同步隨著背景移動，因此在 doodle 上升的過程中其上升的距離會和物件下降的距離呈現相對運動，最後利用 if 指定只有在 0≦x≦540 && 0≦y≦960 的物件才會被顯示在畫面中，使不至於占用龐大記憶體空間而導致遊戲無法順利進行。而在過程中若是觸碰到道具物件，則會透過碰撞函數觸發並執行其相對應的動作。

在本次的 project 製作中，我們使用了三個檔案完成，分別是 main.cpp、mainwindow.h、mainwindow.cpp，而我們會在以下慢慢介紹每個檔案內部程式的發想。

二、程式介紹

首先是 main.cpp，而由以下程式我們可以知道，main function 的主要目的在於執行一個 QApplication a 並創建一個 mainwindow class 的 object，將其顯示在螢幕上。

```
1   #include "mainwindow.h"
2   #include <QApplication>
3
4   int main(int argc, char *argv[])
5   {
6       QApplication a(argc, argv);
7       MainWindow w;
8       w.show();
9       return a.exec();
10  }
```

而 mainwindow.h 檔內部包含了 mainwindow.cpp 所需要的 member fuction 和 member data，其中我們會用以下幾個 member fuction 逐一介紹，而 member data 將會再被用到時再介紹。

```
public:
MainWindow(QWidget *parent = 0);

protected:
void mouseMoveEvent(QMouseEvent *event);
void mousePressEvent(QMouseEvent *event);
void keyPressEvent(QKeyEvent *event);
void paintEvent(QPaintEvent *event);
void timerEvent(QTimerEvent *event);
void collide();
void stairposition();
void picture();
```

1. Mainwindow：

從 function 的名字我們可以得知其為一個 constructor，當我們在 main.cpp 生成 object w 時，constructor 內部的程式將會自動執行作為初始化。首先我們先設定遊戲的視窗大小，並且允許滑鼠座標的追蹤，並且呼叫 function stairposition()設定樓梯出現的位置(會在等下詳細介紹)，而 function picture()則是將在遊戲中所會用到的圖片以不同的形式載入並設定適當大小，最後則是將計時器所需要用到的時間參數初始化。

```
3   MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
4   {
5       setFixedSize(540, 960);
6       setMouseTracking(true);
7
8       //生成樓梯
9       stairposition();
10
11      //圖片載入
12      picture();
13
14      time1 = 0;
15      timerID = startTimer(1000 / 60); //1/60sec timer
16
17      time2 = 0;
18
19      time3 = 10;
20      timerID3 = startTimer(20);
21
22      time4 = 10;
23      timerID4 = startTimer(20);
24
25      time5 = 0;
26      timerID5 = startTimer(10);
27
28      time6 = 0;
29      time7 = 0;
30  }
```

2. MouseMoveEvent：

這個 function 的作用用於追蹤滑鼠的座標的位置並將其存入變數 x 和 y 中。

```
void MainWindow::mouseMoveEvent(QMouseEvent *event)
{
    x = event->x();
    y = event->y();
    update(); //call paintEvent
}
```

3. MousePressEvent

這個 function 的作用在於讀取滑鼠的點擊，在遊戲中只需讀取滑鼠的

```
//使用 childAt() 函數獲取在鼠標點擊位置上的子控件，然後判斷這個子控件是否是 play。如果是，則滿足條件
//一按下play就會開始動畫!!
QWidget *playc = childAt(event->pos());
if (playc && playc == play) {
    key = "讚哦";
    n = 1;
    i1 = 0;
    b = 0;
    Blood = 3;//用以生成一開始向左的doodle，再生成後令i1=1;

    play->hide();
    interface1->hide();

    //生成doodle初始位置
    doodlex = 210;
    doodley = 650;

    update();
}
```

```
//子彈發射條件

if (b > 0)
{
    eject = true;
    update();
}
b++;
```

左鍵動作，而其中包含了上圖兩種功能，左圖為點即開始鍵開始遊戲並初始化以些參數(後面細講)，右圖為遊戲進行中的子彈發射觸發條件。

4. KeyPressEvent

這個 function 的作用在於偵測鍵盤的動作，而在遊戲當中我們在按下鍵盤左右鍵時會隨之改變 doodle 的 x 方向位置，並且在到達邊界時，使其從另一側出現。

```
if (event->key() == Qt::Key_Left)
{
    key = "left arrow key";
    dir = 1;       //當dir==1，載入doodleL
    doodlex -= 10;

    if (doodlex <= -25)          //到達畫面最左時從最右出來
    {
        doodlex = 525;
    }
    update();//call paintEvent
}
```

```
else if (event->key() == Qt::Key_Right)
{
    key = "right arrow key";
    dir = 2;       //當dir==2，載入doodleR
    doodlex += 10;
    if (doodlex >= 525)          //到達畫面最右時從最左出來
    {
        doodlex = -25;
    }
    update();//call paintEvent
}
```

5. paintEvent

這個 function 的作用在於處理所需顯示在螢幕上之動作。而這個 function 特殊的地方在於期會在非常短的時間內重複執行 function 內的程式，有點類似一個巨大的迴圈，因此為了避免記憶體爆掉，若是和顯示無關的程式碼盡量不要放在裡面，否則每次刷新時都需要執行一

次的話會非常卡。而任何在 paintevent 外所需要改變 paintevent 內部數值的動作皆需要利用 update(); 這個 function 傳值進來刷新。首先我們創建要印出字型所需要的物件。

```
QPainter painter(this);
QFont font;
QPen pen;
painter.setRenderHint(QPainter::Antialiasing, true);
```

接著，我們利用兩個 if 迴圈 if (n == 1) 和 if (n == 2)分別代表遊戲進行中所需要的動作以及遊戲結束後所需的動作。而這裡要知道，先 painter 出來的物件會在較下層，因此欲使該物件出現在越上層則須將程式寫在越後面。

在 if (n == 1)中，我們首先利用 painter 生成背景，接著生成樓梯、道具和怪物。由於樓梯、道具和怪物的生成方式大同小異，因此這邊同樣類似的程式我只會介紹一次。以綠色梯子為例，由於我們是利用 array 預先生成每個相同的物件的所有位置，再利用 for 迴圈將 araay 的質疑個一個印出，因此為了避免有些尚未改到數值的 array 預設位置在(0,0)，我們使其(x,y)不在(0,0)時才生成，而為了減輕記憶體負擔我們只會在將出現在畫面中也就是 y 座標在 0~960 區間的物件 painter 出來。因此必須符合以上四個條件，樓梯物件才會被生成在螢幕上，而當期一條件未達成，物件就會消失釋放記憶體空間。

```
for (int j = 0; j < 2000; j++)
{
    if ((stairgreen_x[j] != 0) && (stairgreen_y[j] != 0) && ((stairgreen_y[j] + Score) > -30) && ((stairgreen_y[j] + Score) <= 960)) {
        painter.drawPixmap(stairgreen_x[j], stairgreen_y[j] + Score, stairgreen);
    }
}
```

其中有幾個較特殊的物件，例如 brokenstair 和倒數計時的樓梯(我們稱之為 stairred)會牽涉到消失的問題，因此我們在這邊加上其他條件。就 brokenstair 為例，當我們在後面會介紹的 collide function 中偵測到碰撞，會回傳一個布林值，這個布林值則會觸發消失的動畫，其中消失的時間 timebroken 會在計時器中觸發計算。

```
for (int j = 0; j < 500; j++)
{
    if ((stairbroken_x[j] != 0) && (stairbroken_y[j] != 0) && ((stairbroken_y[j] + Score) > -30) && ((stairbroken_y[j] + Score) <= 960)) {
        if (timebroken >= 1 && timebroken < 7)
        {
            touchbroken = true;
            painter.drawPixmap(stairbroken_x[j], stairbroken_y[j] + Score, stairbroken2);
        }
        else if (timebroken >= 7 && timebroken < 14)
        {
            touchbroken = true;
            painter.drawPixmap(stairbroken_x[j], stairbroken_y[j] + Score, stairbroken3);
        }
        else if (timebroken >= 14 && timebroken < 20)
        {
            touchbroken = true;
            painter.drawPixmap(stairbroken_x[j], stairbroken_y[j] + Score, stairbroken4);
        }

        if (touchbroken == false)
        {
            painter.drawPixmap(stairbroken_x[j], stairbroken_y[j] + Score, stairbroken1);
        }
    }
}
```

而 stairred 則是在 collide function 偵測到出現在螢幕中時，會回傳布林值 seered，並在四秒過後消失在螢幕中(使其 x 值變為 700 即消失在螢幕右側)。

```
for (int j = 0; j < 200; j++)                              ///在畫面出現4秒後消失
{
    if ((stairred_x[j] != 0) && (stairred_y[j] != 0) && ((stairred_y[j] + Score) > -30) && ((stairred_y[j] + Score) <= 960)) {
        if (timered <= 240)
        {
            seered = true;
            painter.drawPixmap(stairred_x[j], stairred_y[j] + Score, stairred);
        }
        else {
            stairred_x[j] = 700;
        }
    }
}
```

因此由上述可以知道，我們使物件消失的方式為將其 x 座標移到視窗外部，而選擇較遠的 700 而不是 540 是為了避免在 doodle 從螢幕左右側互換時會超出一點點並觸發碰撞，因此則 700 較為保險。

接著我們利用 function collide 進行碰撞檢測(於後方詳述)

```
//碰撞檢測
collide();
```

接著是 doodle 在螢幕上顯示的圖案選擇。而我們利用 if 迴圈分別將其區分為初始狀態(il == 0)、朝向左邊(dir == 1)以及朝向右邊(dir ==2)。而在三個情況下所欲呈現的不同狀態的 doodle 就可以利用 if 迴圈加上條件並在迴圈中以 painter 畫出，當持續為該狀態時，就會不停地畫出該狀態下 doodle 的圖片。以下是初始狀態的範例，當我條件未達 gameover 時則會 painter 出初始照片，當 gameover == true 則會執行 if 內的 painter。

```
if (i1 == 0)
{
    painter.drawPixmap(doodlex, doodley, doodleL);
    if (gameover == true)
    {
        painter.drawImage(0, 20, GameOver);
        Blood = 0;
    }
}
```

而朝左和朝右的內容類似因此在這裡只會解釋其中一個程式內容。已朝所為例，當 gameover == true 同樣印出 gameover 內容；當 touchprot 為 true，則印出有防護罩的 doodle；當 touchhat == true 時會隨著一個計時器印出 hat 的動畫，這裡要注意由於動畫做完一次就結束了，因此在最後要將 touchhat 變為 false 使其不會再重複進入這個 if 迴圈當中；而 jetpack 和 hat 類似因此不再贅述。

```
if (dir == 1) {
    if (gameover == true)
    {
        painter.drawImage(0, 20, GameOver);
        Blood = 0;
    }

    if (touchprot == true) {
        painter.drawPixmap(doodlex, doodley, doodleprotL);
    }
```

```
if (touchhat == true)
{
    if (time6 >= 1 && time6 < 90)
    {
        painter.drawPixmap(doodlex, doodley, doodleH1);
    }
    else if (time6 >= 90 && time6 < 180)
    {
        painter.drawPixmap(doodlex, doodley, doodleH2);
    }
    else if (time6 >= 180 && time6 < 270)
    {
        painter.drawPixmap(doodlex, doodley, doodleH3);
    }
    else if (time6 >= 270 && time6 < 360)
    {
        painter.drawPixmap(doodlex, doodley, doodleH4);
    }
    else if (time6 >= 360)
    {
        touchhat = false;
        mode = 1;
    }
}
```

接著是發射子彈的動畫，如同 mousepressevent 所說到點擊滑鼠左鍵會觸發 eject == true，因此這個動畫會被觸發，並且偵測 doodle 頭頂正中央的位置和點擊滑鼠的位置，並且利用計算兩點距離差乘以一常數作為飛行時間，這是為了達到等子彈不論距離為多少皆為等速率運動，而當偵測動畫完成後會使的子彈的位置變為(0,960)即移出畫面，

避免子彈動畫停留在畫面而使得怪物碰到會自動消失，最後將 eject 變為 false 使其跳出迴圈。

```
//發射子彈

if (eject == true)
{
    Bullet = new QPropertyAnimation(move_bullet, "pos");
    Bullet->setDuration(pow((pow((x - (doodlex + 40)), 2) + pow((y - doodley), 2)), 0.5) * 2);
    Bullet->setStartValue(QPoint(doodlex + 40, doodley));
    Bullet->setEndValue(QPoint(x, y));
    Bullet->start();

    connect(Bullet, &QAbstractAnimation::finished, [=]() {
        move_bullet->move(0, 960);
    });

    eject = false;

}
```

接著生成畫面的頂端以及血量，在不同的 Blood 值下印出不同顆數的愛心，而當血量歸零時會觸發 gameover 並且跳出 n == 1 的 if 迴圈，進入 n == 2 迴圈印出遊戲結束的畫面。

```
//生成頂端
painter.drawPixmap(0, 0, top);

//生成血量
if (Blood == 3) {
    painter.drawPixmap(380, 0, blood);
    painter.drawPixmap(410, 0, blood);
    painter.drawPixmap(440, 0, blood);
}
if (Blood == 2) {
    painter.drawPixmap(410, 0, blood);
    painter.drawPixmap(440, 0, blood);          }
if (Blood == 1) {
    painter.drawPixmap(440, 0, blood);
}
if (Blood <= 0) {
    gameover = true;
    n = 2;
}
```

最後則是在左上角印出當前分數。

```
font.setPointSize(20);
painter.setFont(font);
pen.setColor(Qt::black);
painter.setPen(pen);
painter.drawText(10, 30, QString("Score: %1").arg(Score));
```

接著介紹遊戲結束的迴圈，而其內容即為印出字串 Game over! 並且印出最後的分數。

```
if (n == 2) {
    painter.drawImage(0, 0, background);

    font.setPointSize(40);
    painter.setFont(font);
    pen.setColor(Qt::black);
    painter.setPen(pen);
    painter.drawText(150, 420, QString("Game over!"));
    painter.drawText(140, 480, QString("Score:%1").arg(Score));
}
```

6. timerEvent

在這個 function 裡，我們用了 4 種 timerID：timerID(1000/60)，timerID3(20)，timerID4(20)，timerID5(10)，其計時間隔分別為 1/60s，0.02s，0.02s，0.01s。我們主要利用個 function 來控制 doodle 的各種上跳時間、stairblue&stairdeepblue 的移動時間以及 stiarred 從在畫面出現到消失的時間，使用方法很簡單，首先要知道 timerID 是個計時器，不太可能利用它來完成我們的需求，所以我們另外創了 time1~7 來完成不同任務。

舉 time1 為例，我們知道 timerID 是個 1/60s 的計時器，所以我們是

```
if (event->timerId() == timerID && timerID != 0)
{
    time1 = (time1 == 360) ? 0 : time1 + 1;
```

透過以下程式碼進行秒數控制

這段程式碼是指當 timerID 開始計時候就進入此條件式，同時 time1 每 1/60s 增加 1，直到 time1=360 時，令 time1=0 重新計數，也就是說 time1 是個 6s 一循環的計時器。

接著我希望 doodle 在 mode=1,2,3(talk later)的受重力影響上跳時間為 1s，並在 doodle 到畫面一半時將 doodle 固定在畫面中央，所以我透過以下程式碼實現:(mode=1 為例)

```
if (time1 <= 59  && touchhat == false && touchjet == false)          //上升 1s
{
    down = false;
    if (mode == 1)   //mode 1為正常跳躍
    {
        if (doodley <= 440)
        {
            doodley = 440;
            update();
        }
        else   //未到螢幕中間時的上跳
        {
            doodley -= ((640.0 / 59 / 59) * 59 - (640.0 / 59 / 59) * time1 - (640.0 / 59 / 59 / 2));
            update();
        }
    }
```

關於上跳距離的公式我是用第(t+1)秒位置-第 t 秒位置求得，△y=第(t+1)秒位置-第 t 秒位置，經過計算可以得到△y 公式；

再來利用△y=$v_0 t - \frac{1}{2}at^2$，$v_0 = at$ 求得 a=(上跳距離/上跳時間^2/2)

這邊要注意因為畫面越上面數值越小，所以 doodle 在上跳時應用-= opeation 而非+=operation

- Score 計算

其實 Score 的計算跟 doodley 基本上是一樣的，只是要注意須把-=改成+=，否則 Score 會變成負的。

```
if (down == false)
{//mode不同計分不同
    if (mode == 1)
    {
        actualheight += ((640.0 / 59 / 59) * 59 - (640.0 / 59 / 59) * time1 - (640.0 / 59 / 59 / 2));
        highest = actualheight;
        if (actualheight > Score)
        {
            Score = highest;
            update();
        }
    }
```

```
if (down == true)
{
    //Score = highest;
    actualheight -= ((640.0 / 59 / 59) * time2 - (640.0 / 59 / 59));
    update();
}
```

舉 mode=1 為例，我們會需要一個變數 actualheight 來計算 doodle 與起點的距離，並透過 if(actualheight > Score){Score = actualheight}設定最高點為玩家的分數 Score。

- 特殊 stair

這段程式碼是用於計算 stairbroken 被踩到後的時間以及 stairred 出現在畫面後經過的秒數，用以執行後續動畫。

```
//特殊stairs
if (touchbroken == true)
{
    timebroken += 1;
}

if (seered == true)
{
    timered += 1;
}
```

- 補充

(1) mode = 1，doodle 正常的跳躍模式(上跳時間 1s，上跳 320px)，應用於踩到階梯及怪物時。

(2) mode = 2，doodle 踩到彈簧時的跳躍模式(上跳時間 1s，上跳 960px)。

(3) mode = 3，doodle 踩到跳床時的跳躍模式(上跳時間 1s，上跳 1920px)。

(4) mode = 4，doodle 踩到 propeller hat 的跳躍模式(上升時間 6s，上跳 3840px)，同時用 doodle 戴著帽子的圖檔取代原本的 doodle。

(5) mode = 5，doodle 踩到 jet pack 的跳躍模式(上升時間 8s，上跳 7680px)，同時用 doodle 揹著火箭背包的圖檔取代原本的 doodle。

7. collide

在這個 function 中我們主要利用 QRect 來對每個物件定義初期區域，並且在每種碰撞組合中輔以 if 迴圈來表示觸發碰撞所發出之訊號內容。

首先在一開始，我們率先定義了 doodle 和子彈的矩形位置，會這著做的原因是因為兩這皆需檢查大量的碰撞檢測並且物件只有一個，因此在最外面將其定義好避免了每次在 if 迴圈中皆須定義一次的麻煩。

```
//doodle位置
QRect rect2(doodlex + 20, doodley + 75, 40, 5);
//子彈位置
QPoint pos = move_bullet->pos();
QRect rect3(pos.x(), pos.y(), 20, 20);
```

接著是各種物件的碰撞偵測。同樣得由於樓梯、道具和怪物的生成方式大同小異，因此這邊同樣類似的程式我只會介紹一次。

以下是綠色階梯的碰撞偵測，我們利用 for 迴圈將 array 中每個位置的值皆偵測一次，如同 painter 一樣要符合四個條件才會進行矩形的繪製，而繪製完成後我們比較樓梯的矩形和 doodle 位置是否有進行碰撞，若是碰撞觸發，則會選擇計時器中 mode ==1 的跳躍，其中使得 time1 = 360 的目的是為了重製跳躍的計時器。

```
for (int j = 0; j < 2000; j++)
{
    if ((stairgreen_x[j] != 0) && (stairgreen_y[j] != 0) && ((stairgreen_y[j] + Score) > -30) && ((stairgreen_y[j] + Score) <= 960)) {
        QRect rect1(stairgreen_x[j], stairgreen_y[j] + Score, 120, 30); // 創建一个矩形(左上x，左上y，矩形寬，矩形高)
        if (down == true) {                                 //下降時
            if (rect1.intersects(rect2))                    //碰到
            {
                mode = 1;
                time1 = 360;                                //時間重製，重新跳躍
                //update();
            }
        }
    }
}
```

而其他種類階梯的碰撞偵測則和上述差不多。其中白色樓梯須加上碰撞後消失的條件，即 stairdissapear_x[j] = 540；而 stairbroken 則沒有跳躍動作而是觸發布林值 touchbroken = true 呼叫 paintevent 畫初期消失的動畫。

而 stairthorn 則較為特殊，從以下可以發現，當碰撞到他時，若是有防護罩的狀態(touchprot = true)會被改為 touchprot = false，而當碰撞時本身就為 touchprot = false 則會扣一滴血，而加上 touch = false 是避免同時偵測到第二碰撞使其一次扣兩滴血。

```
for (int j = 0; j < 200; j++)
{
    if ((stairthorn_x[j] != 0) && (stairthorn_y[j] != 0) && ((stairthorn_y[j] + Score) > -30) && ((stairthorn_y[j] + Score) <= 960)) {
        QRect rect1(stairthorn_x[j], stairthorn_y[j] + Score, 120, 30);
        if ((down == true)&&(touch == true)) {
            if (rect1.intersects(rect2))
            {
                mode = 1;
                if (touchprot == false)
                {
                    Blood--;
                }
                touch = false;
                touchprot = false;
                time1 = 360;
            }
        }
    }
}
```

接下來為道具的部分，其中內容和樓梯內容差不多只多了觸碰到會移出視窗，而彈簧和跳床跳躍的 mode 變為 2 和 3。而 hat 和 jetpack 則多了一些東西，首先他們也會觸發自己的 mode，並且由於一次只能吃一種道具，因此若原本有防護罩抑或是其他道具都會使其判斷布林值變為 false，以下以 hat 為例。

```
for (int j = 0; j < 200; j++)
{
    if ((hat_x[j] != 0) && (hat_y[j] != 0) && ((hat_y[j] + Score) > -30) && ((hat_y[j] + Score) <= 960)) {
        QRect rect1(hat_x[j], hat_y[j] + Score, 64, 40);
        if (rect1.intersects(rect2))
        {
            hat_x[j] = 700;
            mode = 4;
            touchprot = false;
            touchjet = false;
            touchhat = true;

            time6 = 0;
        }

    }
}
```

而防護罩則時觸碰時移出移螢幕並使得代表其狀態的布林值變為 true。

```
for (int j = 0; j < 200; j++)
{
    if ((protection_x[j] != 0) && (protection_y[j] != 0) && ((protection_y[j] + Score) > -30) && ((protection_y[j] + Score) <= 960))
        QRect rect1(protection_x[j], protection_y[j] + Score, 80, 72);

            if (rect1.intersects(rect2))
            {
                protection_y[j] = 700;
                touchprot = true;
            }
    }
}
```

再來是怪物，四種怪物都一樣因此這裡以 monster1 為例。首先是怪物和子彈的觸碰(注意這裡是 rect3)，當被子彈打到怪物就會消失。接著是與 doodle 的碰撞，當碰撞時 doodle 會和怪物打一架不論是被怪物扁因此扣血還是把她踩死，打完架都會讓怪物消失，而當在上升時

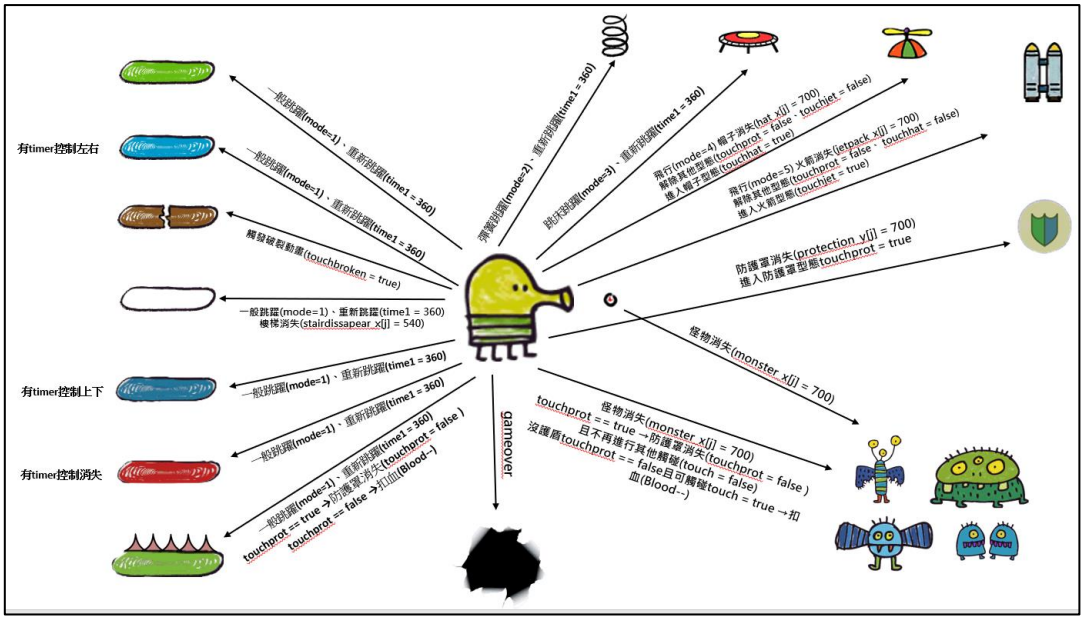
```
for (int j = 0; j < 200; j++)
{
    if ((monster1_x[j] != 0) && (monster1_y[j] != 0) && ((monster1_y[j] + Score) > -30) && ((monster1_y[j] + Score) <= 960)) {
        QRect rect1(monster1_x[j], monster1_y[j] + Score, 168, 105);
        if (rect1.intersects(rect3))
        {
            monster1_x[j] = 700;
        }

        if ((down == false) && (touch == true))
        {
            if (rect1.intersects(rect2))
            {
                if (touchprot == true) {
                    touchprot = false;
                    touch = false;
                }
                if ((touchprot == false) && (touch == true))
                {
                    Blood--;
                    touch = false;
                }
                monster1_x[j] = 700;
            }
        }
        if (down == true)
        {
            if (rect1.intersects(rect2))
            {
                monster1_x[j] = 700;
                time1 = 360;
            }
        }
        //*********碰到旁邊要扣血********
    }
}
```

碰到怪物時若是有防護則會讓其消失，並且同時讓 touch == false，這個目的在於不會再進行觸碰導致防護罩消失的瞬間還會再扣血，而當沒有防護罩時則會扣一滴血，而再下降過程中踩到怪物則可以再次跳起並且將其殺死(移出視窗)。

最後則是黑洞，當碰到黑洞時，會使得 gameover 布林值變為 true，因此結束遊戲。

```
for (int j = 0; j < 200; j++)
{
    if ((black_x[j] != 0) && (black_y[j] != 0) && ((black_y[j] + Score) > -30) && ((black_y[j] + Score) <= 960)) {
        QRect rect1(black_x[j], black_y[j] + Score, 150, 135);
        if (down == true) {
            if (rect1.intersects(rect2))
            {
                gameover = true;
            }
        }
    }
}
```

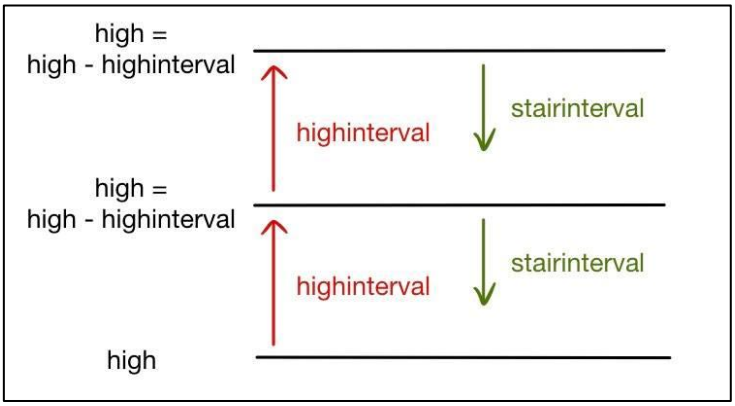
綜合以上可以畫出示意圖如下：



8. stairposition

這個 function 的作用為初始化所有階梯和道具和怪物的位置，我們從 y 值為 850 開始印出第一階綠色樓梯，接著利用隨機生成以及許多不同的高度參數，向 y 值為負的方向生成非常多的物件，並利用 painter 的過程中再每個物件的 y 座標加上遊戲分數，因為遊戲分數就是 doodle 的上升距離，也就是背景的下降距離，因此可以將其看做是我們生成了一個巨大長條的背景，並隨著 doodle 的上升將背景下移，呈現出不斷上升的感覺。

而設計出這個巨大背景的關鍵可以用以下示意圖解釋。我利用初始



化 high 的值(即 850)印出第一階的樓梯，接著利用 rand() % (自訂的範圍)生成一個隨機的 highinterval，並將 high 減去一個 highinterval 向上到達一個新的 high，再利用一個新的 stairinterval 變數在新的 high 位置向下一點成為下一個階梯的 y 值生成範圍，最後利用 high + 30 + (rand() % (stairinterval))在新的高度往下一個最大值為 stairinterval 的高度生成新的樓梯，這使得我的新樓梯位置會在舊的 high 和新的 high 中，而中+30 是為了

避免前一階剛好生成在區間內的最上層，而後一階生成在區間內的最下層而使兩者距離過於接近(因為樓梯之間需要最小 60，因此我選擇了 60/2 = 30)。

而後續的生成我們以 0~3000、C、6000~9000、9000~12000 以即 12000 以上做為區分，會這麼做的原因是因為這些區間所含有的樓梯種類以即出現的道具皆不一樣。

而在 .h 檔中，我們將每個欲生成的物件皆載入圖檔，若是有動畫的物件則載入多個(如 spring)，而每個物件類別會有自己的 x,y 座標 array 以即計算 array 元素的參數(正常從 0 開始，由於 stairgreen 第一階為固定因此從 1 開始算)。

```
QPixmap stairgreen;
int stairgreen_x[number];//greenstair位置
int stairgreen_y[number];
int green = 1;//計算greenstair數量
```

```
QPixmap spring1;
QPixmap spring2;
int spring_x[number];//spring位置
int spring_y[number];
int spr = 0; //spring數量計算
```

- 0~3000

首先，我們將第一階位置固定，接著印出後續樓梯位置，其中 x 值選擇讓它隨機生成，y 值則利用上述所說之原理印出。而其中利用參數 brokenhigh 累積高度 1000 印出一個 brokenstair，印出後將 brokenhigh 歸零重新計算。

```
//第一階固定位置
stairgreen_x[0] = 210;
stairgreen_y[0] = 850;
//隨機產生
srand(time(NULL));
//高度0~3000 較密集
while ((high <= 850) && (high > -2150))
{
    if (brokenhigh > 1000) {                    //每累積高度1000，生成一個brokenstair
        highinterval = 60;
        stairinterval = highinterval - 30;
        high = high - highinterval;
        stairbroken_x[broken] = rand() % 420;
        stairbroken_y[broken] = high + 30 + (rand() % (stairinterval));
        broken++;
        brokenhigh = 0;  //高度歸零
    }
    else {
        highinterval = 60 + rand() % 120;
        stairinterval = highinterval - 59;
        high = high - highinterval;
        brokenhigh = brokenhigh + highinterval;      //累積高度
        stairgreen_x[green] = rand() % 420;
        stairgreen_y[green] = high + 30 + (rand() % (stairinterval));
        green++;
    }
}
```

- 3000~6000

在這個區間中，我們多加入了怪物以及黑洞，因此為了區分甚麼時候要印什麼，我們在原本生成 stairgreen 的 else 迴圈中加入了一個 1~100 隨機變數

color = 1 + rand() % 100，利用 color 隨機生成出來的值決定要觸發哪種生成，以下圖為例，當 color 在 1~96 時，即大部分時間都生成單一 stairgreen，而這裡要注意我們在這個區間中開始加上道具，並且道具的生成只會隨著單一 stairgreen 的那層出現，我們利用 if (rand() % 100 <= 5)作為觸發道具生成的迴圈，意即生成機率為 5%，而在這個高度區間道具只有彈簧而已。而當 color 為 97、98 時會生成一個 stairgreen 配上怪物 1 或 2 或黑洞，而生成的機率用一個隨機變數 monster 決定，當 color 為 99、100 時會生成一個 stairgreen 配上怪物 3 或 4 或黑洞，而此時須注意由於怪物和黑洞的大小較大，因此此時的 highinterval 需較大避免怪物或黑洞和前後的樓梯重疊。

```
color = 1 + rand() % 100;
if (color < 97) {
    highinterval = 60 + rand() % 120;
    stairinterval = highinterval - 59;
    high = high - highinterval;
    brokenhigh = brokenhigh + highinterval;
    stairgreen_x[green] = rand() % 420;
    stairgreen_y[green] = high + 30 + (rand() % (stairinterval));
    //道具生成
    if (rand() % 100 <= 5) {                          //有15%的機率生成道具
        requirement = 1 + rand() % 100;
        if (requirement < 70) {                                          //70%生成彈簧
            spring_x[spr] = stairgreen_x[green] + 45;
            spring_y[spr] = stairgreen_y[green] - 20;
            spr++;
        }
        else if (requirement <= 100) {                                   //剩下30%生成防護罩
            protection_x[protection] = stairgreen_x[green] + 20;
            protection_y[protection] = stairgreen_y[green] - 72;
            protection++;
        }
    }
    green++;
}
```

```
//color在97、98時生成greenstair和monster1、2和黑洞
else if ((color >= 97) && (color < 99)) {
    highinterval = 250;
    stairinterval = highinterval - 100;
    high = high - highinterval;
    brokenhigh = brokenhigh + highinterval;
    stairgreen_x[green] = rand() % 220;
    stairgreen_y[green] = high + 30 + (rand() % (stairinterval));
    monster = 1 + rand() % 3;
    if (monster == 1) {
        monster1_x[monster1] = 340 + rand() % 32;
        monster1_y[monster1] = high - 20 + (rand() % (stairinterval));
        monster1++;
    }
    if (monster == 2) {
        monster2_x[monster2] = 340 + rand() % 72;
        monster2_y[monster2] = high - 40 + (rand() % (stairinterval));
        monster2++;
    }
    if (monster == 3) {
        black_x[black] = 340 + rand() % 50;
        black_y[black] = high - 20 + (rand() % (stairinterval));
        black++;
    }
    green++;
}
```

```
//color在99、100時生成greenstair和monster3、4和黑洞
else if ((color >= 99) && (color <= 100)) {
    highinterval = 250;
    stairinterval = highinterval - 100;
    high = high - highinterval;
    brokenhigh = brokenhigh + highinterval;
    stairgreen_x[green] = 270 + rand() % 15;
    stairgreen_y[green] = high + 30 + (rand() % (stairinterval));
    monster = 1 + rand() % 3;
    if (monster == 1) {
        monster3_x[monster3] = 30 + rand() % 80;
        monster3_y[monster3] = high - 20 + (rand() % (stairinterval));
        monster3++;
    }
    if (monster == 2) {
        monster4_x[monster4] = 30 + rand() % 80;
        monster4_y[monster4] = high - 15 + (rand() % (stairinterval));
        monster4++;
    }
    if (monster == 3) {
        black_x[black] = 30 + rand() % 90;
        black_y[black] = high - 20 + (rand() % (stairinterval));
        black++;
    }
    green++;
}
```

- 6000~9000

在這個區間中，最一開始一樣是生成 brokenstair，接著是生成其他階梯種類的 else 迴圈，而這個區間中生成單一綠色樓梯以及道具的機率為 80%(即 colorj 為 1~80)，而其中伴隨著道具生成的機率為 5%，而道具多了防護罩

和彈簧床其中 70%生成彈簧、20%彈簧床、10%防護罩，其中的機率我們利用隨機變數 requirement 決定。接著，在 color 在 81~96 時生成 bluestair，這裡沒有特別定義其 x 值是因為，其 x 值是會左右移動，因此我們將其定義在計時其中即可。而和先前一樣，當 color 為 97、98 時會生成一個 staircgreen 配上怪物 1 或 2 或黑洞，當 color 為 99、100 時會生成一個 staircgreen 配上怪物 3 或 4 或黑洞。

- 9000~12000

利用相同概念，在這個範圍內道具又多了帽子和彈簧鞋，機率一樣利用隨機變數 requirement 決定，而樓梯的部分，新增了上下移動的樓梯、觸碰到會消失的樓梯，出現的機率一樣利用 color 的區間定義。此時須注意上下移動的階梯 highinterval 也須加大，否則其在移動過程中可能會與其他露梯重疊，而其上下移動的 y 值會以我們初始化的 y 值搭配計時器中的時間變化將其作出緩慢的上下移動。

- 大於 12000

在最後這個區間以上，所有該出現的道具以及樓梯都出現了，我們在這個階段新增了倒數計時樓梯以及火箭，同樣的樓梯生成的機率以 color 表示，道具生成的機率以 requirement 決定，變數 monster 則是決定升成怪物還是黑洞。

9. Picture

這個 function 作用為載入所需要之圖片，其中我們分為以下不同形式討論。

1. QLabel（物件可被識別）

首先，先從資料夾中載入圖檔，接著創建一個 QLabel，並如圖上註解所示，設定 QLabel 相關特性。

```
playbutton.load("./dataset/images/playbutton.png");
playbutton = playbutton.scaled(188, 70, Qt::KeepAspectRatio);
play = new QLabel(this);                                    //創建一個新的 QLabel，命名為 play，並將它設定為當前視窗（this）的子元素。
play->setFixedSize(playbutton.size());                      //設置play的大小與playbuttom的大小一樣
play->setPixmap(playbutton);                                //將縮放後的playbuttom設置到QLabel對象play中
play->setObjectName("play");                               //指定物件名稱為"play"
play->move(83, 311);                                        //將QLabel物件移動至視窗(x,y)之位置
```

2. Qpixmap、QImage

兩者皆可以直接利用以下方法載入，若需要調整大小可以利用第二行程式將其縮放，其中 KeepAspectRatio 的用意在於當其中一個方向先縮放到所設值的時後即停止縮放，這會使得圖片和原圖為等比例縮放。

```
background.load("./dataset/images/background.png");
background = background.scaled(540, 960, Qt::KeepAspectRatio);
```

三、遇到的困難

- TimerEvent

一開始在計算 doodle 跳躍時的縱座標上一直不如我的預期，他總是一次大跳好一段，後來發現是因為一開始我用△y=v_0 t-1/2 at^2，v_0=at 去計算當下 doodle 的垂直位置，但只要這樣打，doodle 上跳距離就變得很誇張，在思考許久後發現，因為我用的是前 1/60 秒 doodle 所在的位置加上下一個 1/60 秒位移作為 doodle 的垂直座標，所以我要用的應該是△y=第(t+1)秒位置-第 t 秒位置。

- 關於 PLAY

一開始在思考怎麼點擊初始畫面的 PLAY 後遊戲開始，上網查了些範例後選擇將要被辨別的 PLAY 按鈕生成為 QLabel 的物件，然後程式就有辦法判斷我們現在點的是否為名為 PLAY 的這個 PLabel 這個物件。
但最後我們捨棄了這個方案，因為 QLabel 並沒有辦法被寫在 PainterEvent 裡，所以如果我們階梯也將他生成為 QLabel 物件，那他的生成位置無法隨著計時器或 doodle 的 y 值改變而改變。經過詢問助教後，我們選擇改用 QRect()這個函數來判斷點擊區域，而非判斷物件。

- 邏輯問題

自從改用 Painter 的方式顯示圖片後，剩下的問題大部分都是邏輯問題，像是要傳入何值進入 PainterEvent、何時要讓那些圖案出現或消失等。
因為 PainterEvent 本身就類似一個迴圈，所以只要在外部更新傳入值，圖片的改變就會自動顯示，不用在 PainterEvent 裡在寫個迴圈。也因此我們於外部利用了大量的 If() else 函數，並且創造了大量的 bool data 當作開關，以決定何時那些條件要被觸發並執行相關指令。
這邊主要困難的點是很多時候我們沒辦法一次考慮周全，當設下了某些條件並進行編譯後，會產生遊戲進行不如預期的狀況，這時候就需要獨自慢慢品味條件，並從頭梳理何種情況會要跑這個條件，何種情況則否。
還好通常自己慢慢思考過一段時間後就能順利整理出合適的條件。

- 資料誤用

但也因為大量的 bool data，我們常常會有某一天在改完 Code 後導致遊戲不如我們預期的進行，經過一段時間的 debug 後才發現原來是我們前一天新增了幾個 bool，而我們現在誤用他們了。後來我們的解決方法就是盡量在.h 建立 member data 時就先寫好他應該是用於何種條件判斷之類的。

- 子彈消失問題

  在製作子彈的動畫中我們發現，子代在動畫結束時會停留在結束動畫的位置，這導致子彈會持續出現在螢幕上，而當呼叫 collide 函數判斷時，留在螢幕上的子彈會生成其 QRect 導致怪物在出現的同時可能會碰到殘留的子彈而快速消失。我們試過在動畫後再寫一個塊蘇的消失動畫將其移出螢幕外，但後來發現 paintevent 的呼叫是瞬間的這代表第一個動畫才剛進行，第二個動畫就開始執行，使得子彈還沒飛行過去就直接被移出螢幕，因此在找尋許多資料後，我們發現了一個 connect 函數，它可以偵測動畫結束的時間在進行後續動作，因此我們在 connect 函數中打上子彈消失的程式碼，便順利的在子彈飛到定點時消失。