Project1_Group16

> Website View : https://jtsai1120.notion.site/Project1_Group16-fdab43eff7734a8a8fbbee8c2f420d48

> 蔡承希、蔡辰鑫、黃士洵

Main

Main.cpp

僅呼叫一個 MainWindow 物件名為 mainwindow，並將其顯示。

MainWindow

> mainwindow.h
> mainwindow.cpp
> mainwindow_game_init.h

MainWindow 物件生成時，我們首先讓 Ctor 初始化一些內容：

- 設定顯示視窗 `view` 大小

> 💡 `view` 為 QGraphicsView 的物件，可以視為視窗。我們可以透過將不同的 QGraphicsScene 物件 (場景) 傳入 `view` 以達成切換場景的效果。

- 將開始遊戲按鈕連接訊號槽到 `MainWindow::on_start_button_clicked()` 來呼叫 `MainWindow::game_init()` 以初始化遊戲畫面
- 利用 QTimer 建立一個計時 10ms 的計時器，再利用 訊號槽與觸發機制 設定每 10ms 呼叫一次 `MainWindow::update_frame()` 來更新畫面內容
- 呼叫成員函數 `MainWindow::start_init()` 來初始化開始畫面

> 💡 我們將 `gamestatus` 分為 3 種狀態：
> 0 ⇒ 開始畫面
> 1 ⇒ 遊戲中
> 2 ⇒ 結算畫面

▼ MainWindow::update_frame()

被誰呼叫：每 10ms 會被計時器連接的訊號槽呼叫

功能：偵測目前遊戲狀態來更新畫面內容

- 當遊戲狀態為 0 (開始畫面) 時，將 `view` 設定為 `start_scene`
- 當遊戲狀態為 1 (遊戲中) 時，呼叫：
  - `MainWindow::all_move_detection()` 決定該讓馬力歐相對畫面移動或是讓畫面相對馬力歐移動

    > 💡 我們讓馬力歐在移動時能夠固定在畫面中央的方法是讓畫面上的所有物件相對馬力歐去做移動，但當馬力歐在最左邊的場景時，因為最左邊的場景物件不能再往右移動 (意即露出場地外的部分)，因此需要讓馬力歐在畫面上的絕對座標去移動。`all_move_detection()` 就是用來決定是誰要相對誰移動的函數

  - 所有需要重力加速度的物件的成員函數 `move()` 以達成重力加速度的效果
  - 呼叫所有子彈物件的成員函數 `fly()` 以達成讓子彈依特定斜率飛行的效果

> 我們讓物件(馬力歐、方塊、蘑菇等)實現重力加速度的方式是為每個物件額外定義 dx 和 dy 兩個變數，而遊戲狀態中每次 update_frame() 被呼叫時，就會接著去呼叫這些需要實現重力加速度的物件的成員函數 move()，讓它們的 x 和 y 座標分別增加 dx 和 dy，而重力加速度的實現只需要在每次 move() 後讓 dy 改變固定值，就如同將速度值加上一個加速度值一般。

    - 偵測若達到 GameOver 的其中之一個條件即呼叫 `Mainwindow::end_init()` 來初始化結算畫面。
- 當遊戲狀態為 2 (結算畫面) 時，將 `view` 設定為 `game_scene`

▼ MainWindow::start_init()

被誰呼叫：`MainWindow::MainWindow()`，即 Ctor

- 將遊戲狀態設為 0 (開始畫面)
- 將要提交給視窗 `view` 的場景 `cur_scene` 設為 `start_scene`
- 載入開始畫面的圖片(QPixmap 物件)，並生成一個開始畫面的物件(QGraphicsPixmapItem 物件)，再將圖片傳入物件，並設定物件的座標後加入目前的場景 `cur_scene` 當中
- 載入開始遊戲按鈕的圖片(QPixmap 物件)，並生成一個開始遊戲按鈕(QPushButton 物件)，再生成一個開始遊戲按鈕的物件(ButtonItem 物件)，再將圖片與按鈕傳入物件，並設定物件的座標後加入目前的場景 `cur_scene` 當中

> ButtonItem 是我們自定義的類別，功能是將 QPushButton 物件轉成 QGraphicsItem 物件

▼ MainWindow::end_init()

被誰呼叫：`MainWindow::update_frame()`，當遊戲結束的任一條件被滿足後被觸發

- 將遊戲狀態設為 2 (結算畫面)
- 將結算畫面的三個物件 (白底背景、GameOver圖片、結算文字) 的 x 座標設為 1800 (剛好在畫面外)
- 將白底背景物件 (QGraphicsRectItem 物件)、GameOver背景物件 (QGraphicsPixmapItem 物件)、文字物件 (QGraphicsTextItem 物件) 加入目前的場景 `cur_scene` 當中。

> 文字物件會去偵測遊戲結束的原因以決定顯示的內容。

- 新增一個 fade_in_timer (QTimer 物件) 來讓結算畫面的三個物件 (白底背景、GameOver圖片、結算文字) 能夠逐漸移動到視窗中央，透過將 fade_in_timer 設定為每 1ms 觸發一次並將訊號槽連接到 `Mainwindow::game_over_fade_in` 來實現。

▼ MainWindow::game_over_fade_in()

被誰呼叫：當 fade_in_timer 開始計時時，每 1ms 被 fade_in_timer 所連接的訊號槽呼叫一次

每次被呼叫：

- 讓結算畫面的三個物件 (白底背景、GameOver圖片、結算文字) 的 x 座標往左移一點點。
- 偵測是否已經移到底(左側)：

- 刪除 fade_in_timer 計時器
- 載入重新遊戲按鈕的圖片(QPixmap 物件)，並生成一個重新遊戲按鈕(QPushButton 物件)，再生成一個重新遊戲按鈕的物件(ButtonItem 物件)，再將圖片與按鈕傳入物件，並設定物件的座標後加入目前的場景 `cur_scene` 當中。
- 將重新遊戲按鈕連接訊號槽到 `MainWindow::on_restart_button_clicked()` 來呼叫 `MainWindow::game_restart()` 進行重新遊戲的參數更新。

▼ MainWindow::on_restart_button_clicked()

被誰呼叫：每當重新遊戲按鈕被按下時，觸發訊號槽呼叫

移除結算畫面的三個物件 (白底背景、GameOver圖片、結算文字)，並將所有此前遊戲中被移除的物件歸位。

▼ MainWindow::keyPressEvent(QKeyEvent *event)

被誰呼叫：每當有任何按鍵被按下時，Qt 會呼叫並傳入 QKeyEvent* 類型參數 (即被按下的鍵的資訊)

- 如果是 a 鍵被按下，將 bool 類型參數 left_key_state 改為 1
- 如果是 d 鍵被按下，將 bool 類型參數 right_key_state 改為 1
- 如果是 w 鍵被按下，將 bool 類型參數 up_key_state 改為 1

▼ MainWindow::keyReleaseEvent(QKeyEvent *event)

被誰呼叫：每當有任何按鍵被放開時，Qt 會呼叫並傳入 QKeyEvent* 類型參數 (即被按下的鍵的資訊)

- 如果是 a 鍵被放開，將 bool 類型參數 left_key_state 改為 0
- 如果是 d 鍵被放開，將 bool 類型參數 right_key_state 改為 0
- 如果是 w 鍵被放開，將 bool 類型參數 up_key_state 改為 0

> 💡 我們將 keyPressEvent() 和 keyReleaseEvent() 同時使用而非只使用前者的原因是，我們發現若按住一個按鍵不放，keyPressEvent() 雖然會在按下去的瞬間被觸發，但是在下一瞬間會突然停止觸發約 0.3 秒的時間，導致馬力歐的移動會卡頓。因此，只要將按下的按鍵以狀態的方式呈現給馬力歐，在按下時轉為1，在放開時轉為0，就能解決在按下後會有短暫時間無法被觸發的問題 ⇒ 如此一來馬力歐就能平滑順暢的移動。

▼ MainWindow::mousePressEvent(QMouseEvent *event)

被誰呼叫：每當滑鼠鍵被按下時，Qt 會呼叫並傳入 QMouseEvent* 類型參數

- 如果滑鼠左鍵被按下，則利用傳入的 QMouseEvent* 類型參數取得鼠標當前在畫面中的座標，再呼叫 `Mario::shoot()` 來告知馬力歐發射子彈。

▼ MainWindow::all_move_detection()

被誰呼叫： `MainWindow::update_frame()` ，當遊戲狀態為 1 (遊戲中) 時，每 10ms 會被間接呼叫一次

功能：

1. 決定該 讓馬力歐相對畫面上所有物件移動 或是 讓畫面上所有物件相對馬力歐移動
2. 偵測馬力歐移動過程中是否碰撞特定物件 (Items、Mashrooms、Flag pole)

決定移動方式的主要偵測的項目有：

- **left_key_state 是否為 1** (代表 a 鍵被按下)
- **right_key_state 是否為 1** (代表 d 鍵被按下)
- **up_key_state 是否為 1** (代表 w 鍵被按下)
- **Mario::movable 是否為 1** (馬力歐目前是否可移動)
- **最 [左/右] 邊的場景是否可以再往 [右/左] 移動** (不能露出場地外的區域)
- **Mario::is_hit_right_side、Mario::is_hit_left_side 是否皆為 0** (代表馬力歐的左、右邊都沒有方塊擋住去路)

判斷流程：

▼ **left_key_state 為 1 或 right_key_state 為 1**

1. 先決定 **最 [左/右] 邊的場景是否可以再往 [右/左] 移動**
   - 可以 ⇒ 讓畫面上所有除了馬力歐的物件移動 (藉由呼叫 `MainWindow::all_horizontal_move()`)，畫面移動量由後面決定。
   - 不行 ⇒ 讓馬力歐移動 (藉由改變 `Mario::dx`)，值由後面決定。
2. 偵測 **Mario::movable 為 1** 且 **Mario::is_hit_right_side、Mario::is_hit_left_side 皆為 0**
   - 成立 ⇒ 畫面移動量、`Mario::dx` 設為 ±5 pixels
   - 不成立 ⇒ 畫面移動量、`Mario::dx` 設為 0 pixels (即不可移動)

> 💡 將移動量作為引數傳入 `MainWindow::all_horizontal_move(int moving_unit)`

▼ **up_key_state 為 1**

- 直接呼叫 `Mario::jump()`

▼ **偵測移動過程中是否碰撞特定物件**

- 若碰到 coin，則將該 coin 轉移至視窗外，並將 score +1
- 若碰到 toxic mushroom，則呼叫 `Mario::is_taller()` (將馬力歐調回小 Size)
- 若碰到 super mushroom，則呼叫 `Mario::touch_super_mushroom()` (將馬力歐調成大 Size)，並將呼叫該 super mushroom 的成員函數 `Super_mushroom::used()` (將該 super mushroom 移至視窗外)
- 若碰到 fire flower，則呼叫 `Mario::touch_fire_flower()` (告知馬力歐增加3枚子彈)，並將呼叫該 fire flower 的成員函數 `Fire_flower::used()` (將該 fire flower 移至視窗外)
- 若碰到 flag pole (遊戲結束的其中一種方式)，則將 `Mario::movable` 改為 0 (讓馬力歐暫時不能移動)，並等待呼叫 `Flag::fall()` (讓 flag 緩緩落下，直到碰到馬力歐後呼叫 `end_init()` 以顯示結算畫面)

> 💡 我們也有考慮到關於馬力歐沒有移動時，可能會被移動中的 coin、toxic mushroom、super mushroom 碰到的可能性，因此也有額外寫這種情況的碰撞偵測，寫法大致上與上述相同。

▼ MainWindow::all_horizontal_move(int moving_unit)

被誰呼叫： `MainWindow::all_move_detection()` ，每當需要畫面上所有除了馬力歐的物件移動時被呼叫

功能：呼叫所有除了馬力歐的物件的成員函數 `move(moving_unit, 0)`

▼ MainWindow::game_init()

被誰呼叫：`MainWindow::MainWindow()`，即 Ctor

功能：布置遊戲關卡

- **對於每個類別的物件大致上都是幾乎相同的步驟：**
  1. 將 QPixmap 物件傳入 QGraphicsPixmapItem 物件
  2. 設定 QGraphicsPixmapItem 物件的座標位置
  3. 將 QGraphicsPixmapItem 物件加入 cur_scene (目前的場景) 中
- **對於有多個物件要放入的類別的處理方式：**
  1. 建立一個 vector 存入所有要放入的物件的位置資訊
  2. 將此 vector 中的所有物件依照上述的方式布置進場景中

▼ MainWindow::game_restart()

被誰呼叫：`MainWindow::game_over_fade_in()`

功能：重新布置遊戲關卡

核心概念：

> 由於所有被創造過的物件都只有被移出畫面，因此只需要再移動回來，即可在不需更動任何記憶體的前提下，開始全新的遊戲。

- **對於每個類別的物件大致上都是幾乎相同的步驟：**
  1. 讀取 `MainWindow::game_init` 已經儲存過的物件的座標位置
  2. 將這些物件重新布置進場景中
  3. 有些物件需要啟動 `reset()` 以還原初始狀態 (因為 Ctor 只能發動一次)
- **對於有多個物件要放入的類別的處理方式：**
  1. 讀取 `MainWindow::game_init` 在 vector 中已經儲存過的物件座標位置資訊
  2. 將此 vector 中的所有物件依序布置進場景中
  3. 有些物件需要啟動 `reset()` 以還原初始狀態 (因為 Ctor 只能發動一次)

Mario

About Size Changing

▼ Mario::move()

被誰呼叫：`MainWindow::update_frame()`

功能：切換馬力歐靜止跟跳躍的圖片

- 靜止圖片
  1. 依照面對的方向切換成站立姿勢

- 跳躍切換圖片
  - 小馬力歐
    1. 依照正在上升或是下降切換跳躍的圖片
  - 大馬力歐
    1. 依照跳躍的移動距離切換跳躍的四張圖片

▼ Mario::change()

被誰呼叫：`MainWindow::all_move_detection()`

功能：切換馬力歐走路的圖片

- 走路跳躍切換圖片
  - 小馬力歐
    1. 如果是在地面上，則透過行走的單位距離切換跑步的兩張圖片
  - 大馬力歐
    1. 如果是在地面上，則透過行走的單位距離切換跑步的三張圖片

▼ Mario::is_taller(int i)

被誰呼叫：`MainWindow::all_move_detection()`

功能：將馬力歐切換成小馬力歐模式

- 高度檢查
  1. 確定馬力歐碰到毒蘑菇時，自己比該個毒蘑菇的當前 y 座標少 (高) 46 像素，且處於下降的狀態，且該毒蘑菇不處於`count_immune > 0`的狀態 (表示免疫該毒蘑菇)。
  2. 若是，呼叫該毒蘑菇並將其`dead`改成`true`。
  3. 若非，自己受傷，呼叫該毒蘑菇並將其`count_immune`改成 200。
- 切換大小
  1. 若馬力歐碰到毒蘑菇，且確定要扣血時，將`cur_size`設為`small`以及將`bool big`設為`false`

▼ Mario::touch_super_mushroom()

被誰呼叫：`MainWindow::all_move_detection()`

功能：若馬力歐觸碰到super mushroom，則將馬力歐調為長大模式

- 切換大小
  1. 若馬力歐觸碰到super mushroom，則將`cur_size`設為`big`以及`bool big`設為`true`

Hp

▼ Mario::is_taller(int i)

被誰呼叫：`MainWindow::all_move_detection()`

功能：將馬力歐血量減一

- 血量減一
    1. 若馬力歐碰到毒蘑菇，且確定要扣血時，將hp扣一

▼ Mario::touch_super_mushroom()

被誰呼叫：`MainWindow::all_move_detection()`

功能：若馬力歐觸碰到super mushroom，則將馬力歐血量加一

- 血量加一
    1. 若馬力歐觸碰到super mushroom，則若血量不為3的情況下，將血量加一

Score

> score.h
> score.cpp

物件被創造時，Ctor 初始化了：

- 設定分數初始值 0
- 外觀
- 指定畫面位置

▼ Score::add_score(int sc = 1)

被誰呼叫：`MainWindow::all_move_detection()`、`Coin::move()`

功能：增加分數 1 分

- 呼叫`Score::set_text()`

▼ Score::reset_score()

被誰呼叫：`MainWindow::game_restart()`

功能：重置分數

- 呼叫`Score::set_text()`

▼ Score::set_text()

被誰呼叫：`Score::reset_score`、`Score::add_score`

功能：設定分數字樣`"Score: "`

Bullet

> bullet.h
> bullet.cpp

當馬力歐吃到fire flower，且子彈數量大於0時，按下右鍵可以觸發

▼ Bullet::shoot(float m , int x0, int y0, int tx)

被誰呼叫：`Mario::` **`shoot`** `(int tx, int ty)`

功能：透過傳送馬力歐位置以及滑鼠點擊的x座標，計算出子彈飛行的路線

- 推算飛行路徑：
  1. 回傳的 `float m` 即為子彈飛行的斜率，`x0` 為馬力歐所在位置，`tx` 為目標所在位子。以這兩者判斷子彈應該往左飛還是往右飛
  2. 將子彈是否發射 `bool already_shot` 設為 true

▼ Bullet::fly()

被誰呼叫：`MainWindow::` **`update_frame`** `()`

功能：讓子彈移動

- 子彈移動：
  1. 將 `already_shot` 為 true 的子彈進行移動，若子彈在飛行途中仍保持在畫面之內，且並未撞到牆壁磚塊等障礙物，則飛行；但若撞到，則移出畫面之外。
- 射擊毒蘑菇
  1. 若和毒蘑菇碰撞，則將該毒蘑菇設為死亡，並同時將子彈移出畫面外。

Blocks

Floor brick

> floor_brick.h
> floor_brick.cpp

物件被創造時，Ctor 初始化了：

- 外觀

▼ Floor_brick::set_x(int new_x)

被誰呼叫： `MainWindow::game_init` 、 `MainWindow::game_restart` 及 `Floor_brick::` **`move`**

功能：使方塊移動到指定的 x 座標位置

- floor_brick 的高度固定，因此移動僅會改變 x 座標。此函式會接收呼叫時傳進的變化量 `new_x`，並使方塊的 x 座標改成 `new_x`，實現移動 (定位)。

▼ Floor_brick::move(int dx)

被誰呼叫： `MainWindow::` **`all_horizontal_move`**

功能：當按鍵按下，做出對應的水平移動

- 呼叫函式 `Floor_brick::` **`set_x`**，傳入 x 座標變化 `dx` 之後的位置。

Normal brick

> normal_brick.h
> normal_brick.cpp

物件被創造時，Ctor 初始化了：

- 套用 normal_brick 外觀
- stone_brick 外觀備用

▼ Normal_brick::set_xy(int new_x, int new_y)

被誰呼叫： `MainWindow::game_init` 、 `MainWindow::game_restart` 及 `Normal_brick::move`

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 new_x、new_y，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Normal_brick::move()

被誰呼叫： `MainWindow::all_horizontal_move`

功能：

1. 當按鍵按下，做出對應的水平移動
2. 重力掉落

- 若不在原本的 y 座標值 `double init_y` ：
  1. 受重力影響，以加速度值 `ay` 不斷加入速度值 `dy`，實現重力掉落。
  2. 若等於原本的 y 座標值 `init_y` ，則設定 `dy` = 0， `crack` 轉為 `false`，不再掉落。
- 呼叫函式 `Normal_brick::set_x y`，讀取 x 座標變化 `dx` ， y 座標變化 `dy` 之後的位置。

▼ Normal_brick::crack()

被誰呼叫： `Mario::is_crack_head`

功能：當馬力歐跳起，頭頂到方塊時，做出的相應互動

- `bool crack` 用以記錄自身的狀態， `true` 為彈跳中， `false` 為靜止。
- 剛彈跳時，設置初速度值 `dy`， `crack` 轉為 `true`。
- 判斷是否本來就存有金幣
  1. 若本來就不存在金幣，則正常彈跳。
  2. 若還存有金幣，則彈跳同時呼叫 `Coin::fly(x, y)`， `no_more_coins` ++。
  3. 若本來存有金幣，但現在沒有了，外觀轉為 stone_brick 並且不能再彈跳。

> 💡 利用變數 `int no_more_coins` 來存取每個方塊的金幣儲藏量。
> -1 ⇒ 本來就沒有金幣
> 0 ~ 4 ⇒ 已經觸發的金幣數量
> 5 ⇒ 本來有金幣，但已全部觸發完

▼ Normal_brick::reset()

被誰呼叫： `MainWindow::game_restart`

功能：將 `no_more_coins`、`crack`、外觀還原

Box brick

> box_brick.h
> box_brick.cpp

物件被創造時，Ctor 初始化了：

- 套用 box_brick 外觀
- stone_brick 外觀備用

▼ Box_brick::set_xy(int new_x, int new_y)

被誰呼叫： `MainWindow::game_init` 、 `MainWindow::game_restart` 及 `Box_brick::move`

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 `new_x`、`new_y`，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Box_brick::move()

被誰呼叫： `MainWindow::all_horizontal_move`

功能：當按鍵按下，做出對應的水平移動

- 呼叫函式 `Box_brick::set_x` `y`，讀取 x 座標變化 `dx` ， y 座標變化 `dy` 之後的位置。

▼ Box_brick::crack()

被誰呼叫： `Mario::is_crack_head`

功能：當馬力歐跳起，頭頂到方塊時，做出的相應互動

- `bool opened` 用以記錄自身的狀態， `true` 為已打開， `false` 為未打開；撞擊後 `opened` 轉為 `true`。
- 若 `bool have_coin` 為 `true` ，呼叫 `Coin::fly(x, y)` ，表示此 box_brick 存放的是金幣。

▼ Box_brick::reset()

被誰呼叫： `MainWindow::game_restart`

功能：將 `opened`、外觀還原

Broken brick

> broken_brick.h
> broken_brick.cpp

物件被創造時，Ctor 初始化了：

- 外觀

▼ Broken_brick::set_xy(int new_x, int new_y)

被誰呼叫：`MainWindow::game_init` 、 `MainWindow::game_restart` 及`Broken_brick::` **`move`**

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 `new_x`、 `new_y`，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Broken_brick::move()

被誰呼叫：`MainWindow::` **`all_horizontal_move`**

功能：

1. 當按鍵按下，做出對應的水平移動
2. 重力掉落

- 若不在原本的 y 座標值`double init_y`：
  1. 受重力影響，以加速度值 `ay` 不斷加入速度值 `dy`，實現重力掉落。
  2. 若高於原本的 y 座標值`init_y` ，則設定`dy` = 1000 移到螢幕之外，實現消失的動畫。
- 呼叫函式`Broken_brick::` **`set_x`** `y`，讀取 x 座標變化 `dx` ， y 座標變化 `dy` 之後的位置。

▼ Broken_brick::crack()

被誰呼叫：`Mario::` **`is_crack_head`**

功能：當馬力歐跳起，頭頂到方塊時，做出的相應互動

- `bool broken` 用以記錄自身的狀態， `true` 為已破壞， `false` 為未破壞。
- 剛彈跳時，設置初速度值 `dy`， `broken` 轉為`true`。

▼ Broken_brick::reset(int new_x, int new_y)

被誰呼叫：`MainWindow::game_restart`

功能：將`broken`、座標位置還原

Invisible brick

> invisible_brick.h
> invisible_brick.cpp

物件被創造時，Ctor 初始化了：

- 外觀

實現隱形的關鍵即在此，我們用和背景一樣的顏色畫出一塊 50 x 50 像素的方塊。

▼ Invisible_brick::set_xy(int new_x, int new_y)

被誰呼叫：`MainWindow::game_init` 、 `MainWindow::game_restart` 及`Invisible_brick::` **`move`**

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 `new_x`、`new_y`，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Invisible_brick::move(int dx)

被誰呼叫： `MainWindow::` **`all_horizontal_move`**

功能：當按鍵按下，做出對應的水平移動

- 呼叫函式 `Invisible_brick::` **`set_x`** `y`，傳入 x 座標變化 `dx` 之後的位置。

Stone brick

> stone_brick.h
> stone_brick.cpp

物件被創造時，Ctor 初始化了：

- 外觀

▼ Stone_brick::set_x(int new_x, int new_y)

被誰呼叫： `MainWindow::game_init` 、 `MainWindow::game_restart` 及 **`Stone_brick`** `::` **`move`**

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 `new_x`、`new_y`，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Stone_brick::move(int dx)

被誰呼叫： `MainWindow::` **`all_horizontal_move`**

功能：當按鍵按下，做出對應的水平移動

- 呼叫函式 **`Stone_brick`** `::` **`set_x`** `y`，傳入 x 座標變化 `dx` 之後的位置。

Water pipe

> water_pipe.h
> water_pipe.cpp

物件被創造時，Ctor 初始化了：

- 外觀

▼ Water_pipe::set_x(int new_x, int new_y)

被誰呼叫： `MainWindow::game_init` 、 `MainWindow::game_restart` 及 `Water_pipe::` **`move`**

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 `new_x`、`new_y`，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Water_pipe::move(int dx)

被誰呼叫： `MainWindow::` **`all_horizontal_move`**

功能：當按鍵按下，做出對應的水平移動

- 呼叫函式`Water_pipe::set_x y`，傳入 x 座標變化 `dx` 之後的位置。

Items

Coin

> coin.h
> coin.cpp

物件被創造時，Ctor 初始化了：

- 外觀

▼ Coin::set_xy(int new_x, int new_y)

被誰呼叫：`MainWindow::game_init` 、 `MainWindow::game_restart` 及`Coin::move`

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 new_x、new_y，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Coin::move()

被誰呼叫：`MainWindow::all_horizontal_move`

功能：

1. 當按鍵按下，做出對應的水平移動
2. 重力掉落

- 若不在原本的 y 座標值`double init_y`：
  1. 受重力影響，以加速度值 `ay` 不斷加入速度值 `dy`，實現重力掉落。
  2. 若高於原本的 y 座標值`init_y` ，則設定 y = 1000，移到螢幕外，不再掉落。`flying` 轉為 `false`，呼叫`Score::add_score (int sc = 1)` 加 1 分。
- 呼叫函式`Coin::set_x y`，讀取 x 座標變化 `dx` ， y 座標變化 `dy` 之後的位置。

▼ Coin::fly(int new_x, int new_y)

被誰呼叫：`Normal_brick::crack` 、 `Box_brick::crack`

功能：當藏有金幣的方塊被觸發時，做出彈跳動畫

- 呼叫`Coin::set_xy(new_x, new_y - 60)`，將要彈跳的金幣移到目標方塊上方。
- `bool flying` 用以記錄自身的狀態，`true` 為飛行中，`false` 為靜止。
- 剛彈跳時，設置初速度值 `dy`， `flying` 轉為`true`。

💡 遊戲中 `std::vector<Coin*> coins` 不同號碼的 `coin_item` 有不同用處。在初始化 `MainWindow::game_init` 中，呼叫一串金幣，其中 `coin[0]` 及 `coin[1]` 分別負責所有 normal_brick 及 box_brick 的動畫，讓其共用；其餘金幣則是正常出現在地圖中。

Fire flower

> fire_flower.h
> fire_flower.cpp

物件被宣告時，將 fire flower 的圖片隱藏在 box brick的圖片之下

▼ Fire_flower::show()

被誰宣告：`Mario::is_crack_head()`

功能：讓 fire flower 顯現出來

- fire flower 顯現
  1. 當馬力歐碰撞到 box brick 且那格剛好是 fire flower 時，以透過比對座標方式判斷是哪個box brick 與此 fire flower 重和，並將此 fire flower 的y座標下移50，集會出現在box brick的上方
  2. 將該 fire flower 的 `bool open` 設為 `true`，避免重複開啟

▼ Fire_flower::move()

被誰呼叫：`MainWindow::all_horizontal_move(int moving_unit)`

功能：隨著畫面移動

- 移動
  1. 將所有fire flower 的水平移動隨著場景移動

▼ Fire_flower::used()

被誰呼叫：`MainWindow::all_move_detection()`

功能：將已被觸碰過的 fire flower 移動到視窗之外

- 移動到視窗之外
  1. 若馬力歐已觸碰到此 fire flower ，則執行該 function，將fire flower 移動到遊戲畫面之外

Flag pole

物件被創造時，Ctor 初始化了：

- 外觀

偵測：在 `MainWindow::all_move_detection()` 中，只要馬力歐在移動過程中碰撞到 Flag pole 物件，則：

- 將 `Mario::movable` 設為 0 (讓馬力歐無法移動)
- 呼叫 `Flag::fall()` (讓 Flag 緩緩降落直到碰到馬力歐)

Flag

物件被創造時，Ctor 初始化了：

- 外觀

▼ Flag::fall()

被誰呼叫：`MainWindow::all_move_detection()`，馬力歐在移動過程中碰撞到 Flag pole 物件的當下被呼叫

- 建立一個訊號槽將 QTimer 物件 `Flag::fall_timer` 連接至 Slot `Flag::fall_until_touch_mario()`，並將 `Flag::fall_timer` 設定為每 5ms 觸發一次後開始計時。(目的是讓旗子能有緩緩下降直到碰到馬力歐的動畫)

▼ Flag::fall_until_touch_mario()

被誰呼叫：每 5ms 被 `Flag::fall_timer` 所連接的訊號槽呼叫

- 每次被呼叫，就讓 Flag 的 y 座標增加 3 (下降 3 pixels)
- 如果在某次下降之後 y 座標 ≥ 馬力歐的 y 座標 (即碰到馬力歐了)，則將 `Flag::fall_timer` 停止計時，並將 `Flag::is_touched_mario` 設為 1 (用來使 `MainWindow::update_frame()` 觸發 `MainWindow::end_init()`)

Mushrooms

Super Mushroom

> super_mushroom.h
> super_mushroom.cpp

物件被創造時，Ctor 初始化了：

- 外觀

▼ Super_Mushroom::set_xy(int new_x, int new_y)

被誰呼叫：`MainWindow::game_init()` 、 `MainWindow::game_restart()` 及 `Toxic_mushroom::move()`

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 new_x、new_y，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Super_mushroom::move()

被誰呼叫：`MainWindow::update_frame()`

功能 : 讓 super mushroom 被顯示之後開始移動

- 水平移動
  1. 偵測是否碰撞牆壁，若碰撞牆壁則反轉方向
  2. 若沒有碰撞牆壁，則維持水平等速移動

- 垂直移動
  1. 偵測是否在地面上，若沒有則開始掉落
  2. 包含重力掉落
- 是否被開啟
  1. 只有被撞擊箱子後顯示的 super mushroom 的 `bool open` 會是 `true`，所以多判斷只有 `open` 為 `true` 的 super mushroom 會開始移動

▼ Super_mushroom::normal_move()

被誰呼叫：`MainWindow::` **`all_horizontal_move`** `(int moving_unit)`

功能 : 讓 super mushroom 隨著場景水平移動

- 水平移動
  1. 跟隨馬力歐的 moving unit 移動水平，讓 super mushroom 跟場井內的東西一起移動

▼ Super_mushroom::is_grounded

被誰呼叫： `Super_mushroom::` **`move()`**

功能：判斷是否踩在任何地板類型上方

- 呼叫 `Super_mushroom::c` **`heck_whether_ground_brick`** `(QGraphicsPixmapItem *PixmapItem)` 測試。

▼ Super_mushroom::check_whether_ground_brick(QGraphicsPixmapItem *PixmapItem)

被誰呼叫： `Super_mushroom::is_grounded()` 、 `Super_mushroom::is_hit_left_side()` 、 `Super_mushroom::is_hit_right_side()`

功能：判斷撞擊方塊

- 調動所有磚塊物件位置進行座標檢查。

> 💡 broken_brick 在 `broken == true` 時不會吃到判斷，保證動畫會穿透過去；
> normal_brick 在 `crack == true` 時不會進入 `lock` 狀態，避免因為 normal_brick 彈跳而造成 toxic_mushroom 在奇怪位置來回。

▼ Super_mushroom::is_hit_left_side()

被誰呼叫： `Super_mushroom::` **`move()`**

功能：判斷是否撞擊右側板塊

- 呼叫 `Super_mushroom::c` **`heck_whether_ground_brick`** `(QGraphicsPixmapItem *PixmapItem)` 測試。

▼ Super_mushroom::is_hit_right_side()

被誰呼叫： `Super_mushroom::` **`move()`**

功能：判斷是否撞擊左側板塊

- 呼叫 `Super_mushroom::c` **`heck_whether_ground_brick`** `(QGraphicsPixmapItem *PixmapItem)` 測試。

Toxic Mushroom

> toxic_mushroom.h
> toxic_mushroom.cpp

物件被創造時，Ctor 初始化了：

- 兩個不同外觀、一個死亡外觀

▼ Toxic_mushroom::set_xy(int new_x, int new_y)

被誰呼叫： `MainWindow::game_init()` 、 `MainWindow::game_restart()` 及 `Toxic_mushroom::move()`

功能：使方塊移動到指定的 x、y 座標位置

- 此函式會接收呼叫時傳進的變化量 new_x、new_y，並使方塊的 x 座標改成 `new_x`，y 座標改成 `new_y`，實現移動 (定位)。

▼ Toxic_mushroom::move()

被誰呼叫： `MainWindow::all_horizontal_move()`

功能：

1. 當按鍵按下，做出對應的水平移動
2. 重力掉落

- `bool still` 若 toxic_mushroom 不在 mario 視野，`still == true` 則定格; `bool dead` 記錄 toxic_mushroom 死亡。兩者皆為 public bool 並由 mario 判斷及操控。
- `int count_immune` 用以記錄 mario 對該個 toxic_mushroom 的剩餘免疫時間，每次將減少 1 直至 0 (因為畫面更新為 10 ms ，200 × 10 ms = 2s)。
- 移動規則：
  1. 每走 30 像素來回更換造型，實現走路動畫。
  2. `lock` 狀態：當 toxic_mushroom 重複來回走，則記錄撞擊點，在撞擊點自動來回而不進入後面的判斷，節省CPU。直到 broken_brick 被破壞則解除。
  3. 若 `dead == true` 則不會做板塊判斷，並轉換皮膚，實現踩扁掉落穿過板塊的動畫。
  4. 底部板塊判斷：呼叫 `Toxic_mushroom::is_grounded` 測試是否踩在任何地板類型上方，否則受重力影響。以加速度值 `ay` 不斷加入速度值 `dy`，實現重力掉落。
  5. 左右撞擊板塊判斷：呼叫 `Toxic_mushroom::is_hit_left_side` 、 `Toxic_mushroom::is_hit_right_side` 測試是否撞擊左右板塊，若有則更改移動方向。
- 呼叫函式 `Toxic_mushroom::set_xy`，讀取 x 座標變化 `dx` ， y 座標變化 `dy` 之後的位置。

▼ Toxic_mushroom::is_grounded

被誰呼叫： `Toxic_mushroom::move()`

功能：判斷是否踩在任何地板類型上方

- 呼叫 `Toxic_mushroom::check_whether_ground_brick(QGraphicsPixmapItem *PixmapItem)` 測試。

▼ Toxic_mushroom::check_whether_ground_brick(QGraphicsPixmapItem *PixmapItem)

被誰呼叫：`Toxic_mushroom::is_grounded()` 、 `Toxic_mushroom::is_hit_left_side()` 、 `Toxic_mushroom::is_hit_right_side()`

功能：判斷撞擊方塊

- 調動所有磚塊物件位置進行座標檢查。

> 💡 broken_brick 在 `broken == true` 時不會吃到判斷，保證動畫會穿透過去；
> normal_brick 在
> `crack == true` 時不會進入 `lock` 狀態，避免因為 normal_brick 彈跳而造成 toxic_mushroom 在奇怪位置來回。

▼ Toxic_mushroom::is_hit_left_side()

被誰呼叫：`Toxic_mushroom::` `move()`

功能：判斷是否撞擊右側板塊

- 呼叫 `Toxic_mushroom::c` `heck_whether_ground_brick` `(QGraphicsPixmapItem *PixmapItem)` 測試。

▼ Toxic_mushroom::is_hit_right_side()

被誰呼叫：`Toxic_mushroom::` `move()`

功能：判斷是否撞擊左側板塊

- 呼叫 `Toxic_mushroom::c` `heck_whether_ground_brick` `(QGraphicsPixmapItem *PixmapItem)` 測試。