C++ Project：Super Mario

陳祈恩(Chi-En Chen)

Team Leader/ Software Developer/ Project Manager

Team members:萬庭瑜、楊珽鈞

Introduction

A. Starting Interface

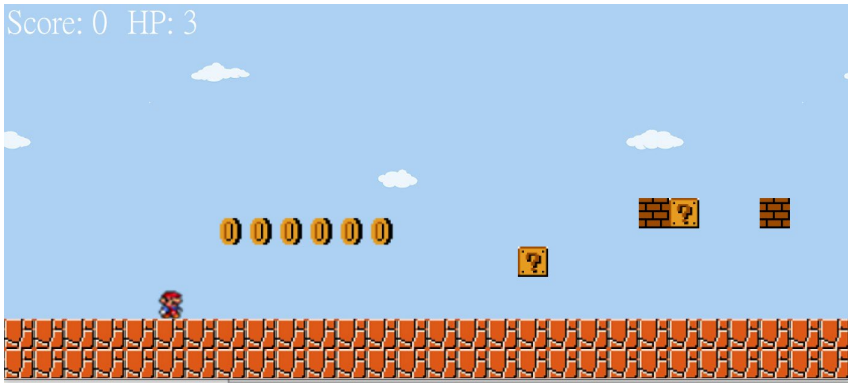
Click “START GAME” button to play the game

B. Ending

Show GAME OVER when Mario is dead

Show YOU WIN when Mario reaches the finishing flag and gains 20 coins

C. How to Play



Control Mario:

- Keyboard left and right keys: Mario moves left and right.
- Keyboard space key: Mario jumps.
- Mouse click: After eating the Fire Flower, Mario can fire the bullet three times.

Lose Condition:

The ending Game Over is determined by **Score**, **HP** in the upper left, and **Hole** on the ground.

1. Score

   The number of gold coins collected by Mario. If Mario reaches the finishing flag successfully, the game will automatically calculate whether the Score reaches 20 points or more. YOU WIN will be shown on the screen if the score exceeds 20 points, otherwise GAME OVER will be shown.

2. HP

The number of lives of Mario, up to 3. If HP reaches 0, the game will show GAME OVER. After the game is over, click the ending window with the mouse and then the game will automatically reset to the start interface.

3. Hole

   Mario falls into the hole and ends the game.

D. Items

1. Floor Brick (the ground in the game)
2. Broken Brick

   It will disappear if Mario hits it from the bottom.

3. Normal Brick (the coin containers)

   If there is no coin inside, it is a normal brick with no special functions; if there are coins, Mario can push up from bottom to get up to 5 coins, and the Normal Brick will automatically turn into a Stone Brick with no special functions.

4. Water Pipe

   It only appears on Floor Brick. Mario needs to jump over it.

5. Box Brick

   When Mario hits it from bottom, one of the three items (Super Mushroom, Fire Flower, and coin) will appear and automatically turn into a Stone Brick without special functions.

6. Fire Flower and Bullet

   Mario has 3 bullets after encountering Fire Flower, which can be used to shoot Toxic Mushroom. Bullets will disappear after touching any bricks.

7. Super Mushroom

   When Mario encounters Super Mushroom, it will increase 1 HP and Mario will grow up, enhancing his jumping ability.

8. Toxic Mushroom

   Mario will lose 1 HP when touching the left, right, and bottom of the Toxic Mushroom. The large Mario will shrink and cannot shoot. If Mario encounters the same Toxic Mushroom again within two seconds, he will not lose 1 HP.

9. Flag

   After Mario encounters the Flag, the game determines whether the player wins or loses based on the Score.

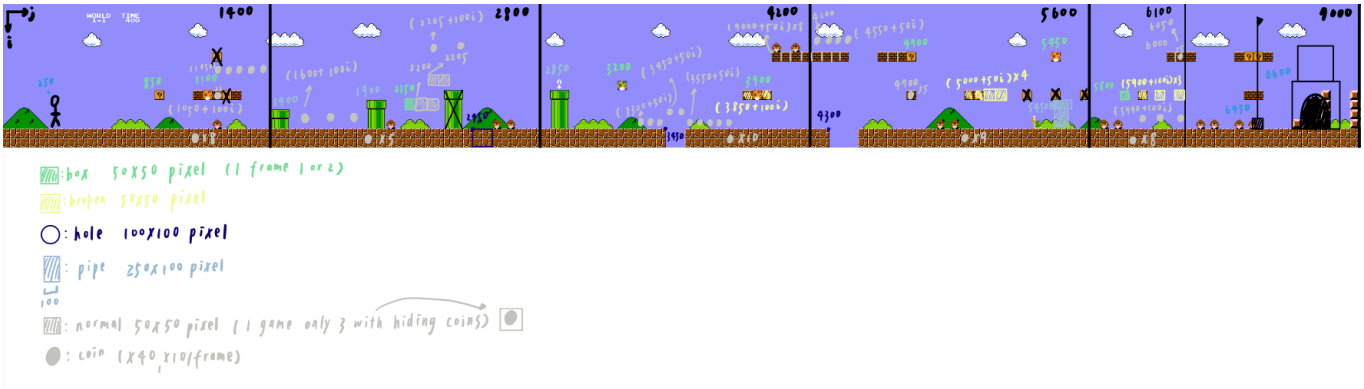
Programming

A. Hierarchy

Excluding background_scene.pro, background_scene_resource.qrc, and main.cpp, the game is divided into 33 .cpp and .h files.

B. Scene Design

Based on the original Super Mario World 1-1 level, we designed and constructed a new level that meets the project requirements. By utilizing the pixel dimensions of props and the view, we calculated and determined the corresponding coordinates for each prop.



Load an image into a QPixmap and then add it to a QGraphicsScene at a specified position using a QGraphicsPixmapItem

C. Mario Implementation

1. Horizontal Movement
   - Using QKeyEvent, when the right or left arrow key is pressed, Mario's x-coordinate is adjusted by a fixed number of pixels to move right or left, respectively. Simultaneously, the corresponding Mario sprite is updated to reflect the movement direction.
2. Jumping
   - Initiation: Upon pressing the spacebar, an upward velocity is applied to Mario.
   - Gravity Simulation: Utilizing QTimer, Mario's position is updated at fixed intervals, simulating the effects of gravity. A jumpstep function is implemented to gradually decrease the upward velocity and increase the downward velocity until Mario collides with a platform or the ground.
3. Collision Detection and Handling
   - Collision Detection: The collidingItems() function is used to check for overlaps between Mario and other objects in the scene.
   - Collision Handling: Based on the relative positions of Mario and the colliding object, different actions are taken. For example, when colliding with a block:
     - Above: Gravity is negated, and Mario is placed on top of the block.
     - Below: The block might be destroyed if it's breakable.
     - Sides: Mario's movement is halted.
   - Interactions with other objects: For instance, colliding with a toxic mushroom from above might trigger the mushroom's break function, while colliding with its sides could decrease Mario's health.
4. Size Transformation
   - Growing: When Mario collides with a super mushroom, the grow function is triggered, increasing Mario's size, adjusting the ground level for jumping, and changing his sprite. Other behaviors and calculations are also modified based on the isBig flag.
   - Shrinking: When a grown Mario collides with a toxic mushroom, the shrink function is triggered, reversing the effects of the grow function.
5. Shooting
   - Acquiring Ability: Mario gains the ability to shoot after collecting a fire flower.
   - Shooting Mechanism: Clicking the left mouse button fires a bullet towards the clicked position.

- Bullet Behavior: The bullet's trajectory is calculated using a line from Mario's center to the clicked position.
- Implementation: The mousePressEvent in the game class captures mouse clicks. The coordinates of the click are converted from view coordinates to scene coordinates and passed to the player class. The player class's shoot function calculates the angle between Mario and the target, creates a bullet object, and reduces the bullet count.

D. Bricks, Items, Score and Health Implementation

1. Broken Brick
   - Functionality: The BrokenBrick class is responsible for its creation, destruction, and visual representation.
   - Destruction: When a BrokenBrick is destroyed, it is removed from the scene using delete this to free up memory. The collision detection and subsequent actions are handled by the Player class.
2. Box Brick
   - Types: BoxBricks can contain three types of items: coins, Super Mushrooms, and Fire Flowers, which are stored in the ItemType property.
   - Creation: The game class calls CreateBoxBrick() to create and position BoxBricks in the scene. The constructor determines the specific object to create based on the ItemType.
   - Bouncing: When a BoxBrick is hit by Mario, the setBounce() function is called. This function sets the initial position of the BoxBrick and starts a timer. The timer triggers the bounce() function, causing the BoxBrick to bounce once.
   - Item Creation: The createItem() function creates the corresponding item object based on the ItemType.
3. Normal Brick
   - Coin Containment: The isNormalBrickWCoin boolean member indicates whether the NormalBrick contains a coin.
   - Creation: The game class calls CreateNormalBrick() to create and position NormalBricks in the scene.
   - Collision Handling: The handleCollision() function handles collisions. If the NormalBrick contains a coin and the coin count is less than 5, the brick bounces, the score is incremented, and a Coin object is created and its setFly() function is called. When the coin count reaches 5, the NormalBrick is transformed into a stone brick.
   - Bouncing: The setBounce() function sets the initial position and starts a timer. The timer triggers the bounce() function, causing the NormalBrick to bounce once.
4. Fire Flower and Bullet
   - Fire Flower Bouncing: The setBounce() function sets the initial position and starts a timer. The timer triggers the bounce() function, causing the fire flower to bounce once.
   - Bullet Movement: The move() function handles the bullet's flight. Since the firing angle is set when the bullet object is created, the move() function can calculate the dx and dy values for each step using trigonometry. The bullet's position is updated every 30ms. The bullet disappears upon hitting a toxic mushroom or any type of brick, or after a certain amount of time.
5. Coin
   - Deletion: When Mario collides with a coin, the deleteCoin() function finds and deletes the

coin.

- Flying and Disappearing: When Mario collides with a normal brick containing a coin, the setFly() function is called to set the coin's initial position and start a timer. The timer triggers the Fly() function, causing the coin to fly a certain height before being deleted.

6. Super Mushroom
   - Movement: Using QTimer, the Super Mushroom's position is updated at fixed intervals, creating the illusion of movement. It can fall, stop on the ground, and bounce off objects, similar to the player. However, when colliding with other objects, it moves in the opposite direction. Additional functions are implemented to handle this behavior.
   - Interaction with Mario: When colliding with Mario, the Super Mushroom is destroyed, and Mario's grow function is called.
7. Toxic Mushroom
   - Movement: The movement behavior is similar to the Super Mushroom, including falling, bouncing, and reversing direction when colliding with objects.
   - Interaction with Mario: When colliding with Mario from the sides and Mario is not jumping, the shrink function in the Player class is called to decrease Mario's size, and the decrease function in the Health class is called to reduce Mario's health.
8. Score
   - Singleton Pattern: A static data member 'instance' is used to ensure that only one instance of the Score class exists. Other classes can access the instance's member functions through the static member function 'getInstance'.
   - Functionality: The getScore() function returns the current score, and the increase() function increments the score by 1 and updates the display.
9. Health
   - Initialization: The constructor initializes the health to 3.
   - Singleton Pattern: Similar to the Score class, a singleton pattern is used to ensure only one instance of the Health class exists. The getInstance() function provides access to the instance's member functions.
   - Functionality: The getHealth() function returns the current health, the increase() function increments the health by 1 and updates the display, and the decrease() function decrements the health by 1.

E. Game Over and Reset Implementation

1. Game Over Process: When the game ends (determined by specific conditions), the GameOver() or YouWin() function is called. This opens a game over or game win window. When the player clicks on the window, it closes, and all objects in the scene are cleared. Finally, the start window is opened, and scene objects are recreated.
2. Game Over Conditions: In the Game::keyPressEvent function, three conditions are checked for game over: Mario reaches the flagpole, Mario falls into a hole, or Mario's HP reaches 0. Depending on the condition, either GameOver() or YouWin() is called.
3. GameOver and YouWin Functions: Both functions create a QDialog window and set a clickable icon as the window's content. The only difference is the image used for each dialog.
4. clearScene Function: After the player clicks on the game over or game win window, the clearScene() function is called to remove all items from the scene, including Mario, score, HP, and items.

5. setStart and CreateElement Functions: After clearing the scene, the setStart() and CreateElement() functions are called to open the start screen and recreate all the objects in the scene.

Challenges

A. Qt Version and Hardware Compatibility

Initially, following the instructor's guidance, we attempted to use Qt 5.12. However, we encountered compatibility issues when running it on an M2 chip. After reinstalling the software multiple times, we switched to Qt 6 as suggested by the instructor, and this successfully resolved the compatibility problem.

B. Overloaded main.cpp and Extra Window Issue

During the early and middle stages of the project, we concentrated all code within the main.cpp file, leading to memory overload and the unexpected appearance of an extra blank window during execution. As a solution, we first moved the code to the MainWindow class, as suggested by the instructor. However, this introduced a new problem: an additional, empty window appeared. We concluded that this was a default window created by the framework. To address this, we created a new class named Game to encapsulate the core game logic, including creation, deletion, and resetting of game elements. This effectively solved the aforementioned issues.

C. Centering Mario and Scrolling the Scene

Initially, we manually adjusted the scroll bar to center Mario within the view. However, this approach was not ideal as the view would continue to scroll, making it difficult to focus on Mario. To address this, we implemented two solutions:

- Solution 1: Automatically adjust the scroll bar based on Mario's position. While this approach worked to some extent, it resulted in the view scrolling indefinitely and issues with mouse clicks.
- Solution 2: Create a Game class to handle the core game logic, including event handling and scene movement. By limiting Mario's movement and properly handling mouse clicks within this class, we successfully resolved the problems encountered in the first solution.