Doodle Jump Project Report

One. Functions and production methods:

1. Keystroke and mouse click event handling:

   QMainWindow inherits QWidget 'skeyPressedEvent andMousePressedEvent.

2. Photo display :

   GameObject has a QPixmap variable and a paint method, which will be displayed in QMainwindow

   was called.

3. Image Movement and Acceleration :

   GameObject has a mover, which provides a move() method to control movement and set the acceleration.

4. Bullet Launch:

   Create a Bullet object that inherits from the GameObject, and set the mover to move in the direction of the mouse.

5. After the stairs/monsters are stepped on, the Player will jump:

   i. Use the DetectionLanding class to detect if the stairs have been stepped on by the player

   ii. If it is stepped on, the react() method of the bound ReactionPlayerJump will be executed, let

   Player jumps up.

6. Broken stair：

   Animation after being stepped on: Set QTimer in ReactionChangeImage to modify the photo of the staircase.

7. When an item is touched, there will be a corresponding reaction:

   i. Use the DetectionTouching class to detect if the item has been touched by the player

   ii. If it is touched, a bound Reaction is executed. For example:

   A. Propeller-hat

   Call react() forReactionPlayerJump to move the Player up to the specified height

   Call react() of ReactionChangeImage and ask the Player to replace it with an image with a hat

   B. Jetpack

   Call react() for ReactionPlayerJump and move the Player up to the specified height for react() called ReactionChangeImage to swap the player with a backpack image

   C. SpringShoes

   Call react() for ReactionPlayerJump and move the Player up to the specified height for react() called ReactionChangeImage and let the Player change to a picture with a spring shoe called react() of ReactionPlayerBuffedJumpHeight, and let the Player defaultJumpHeight and turn it back after a while.

   D. Shield

   Call react() of ReactionPlayerShielded, set the player's shielded variable to true, and call resetShieldedAfter(5000) method, which changes the shielded variable back to false after 5 seconds.

8. monster
    i. If you are shot by a bullet, it will disappear
        A. Use the DetectionTouching class to detect if the monster has been Bullet Bump into.
        B. If it is touched, it will execute the react of the bound ReactionDisappear to make the monster disappear.
    ii. If you touch a player from below or left or right, the player will judge the health deduction.
        A. Use the DetectionTouchingWithoutLanding class to detect if a monster has been touched by the player from below or left or right.
        B. If it is touched, it will execute the react of the bound ReactionPlayerBeAttack , allowing the player to determine if they want to decharge health (if the player has a shield or does not decharge health during the debuff cooldown).
9. Move stairs horizontally and vertically
    i. Set the yVel and xVel of mover.
    ii. Set the BoundaryRection of mover torebounce to achieve the bounce effect.
10. When the player encounters Thorn Stair, they will judge the blood deduction
    i. Use the DetectiontLanding class to detect if the stairs have been stepped on by the player.
    ii. If so, it will execute the react of the bound ReactionPlayerBeAttacked , letting the player decide if they want to decharge health (if the player has a shield or will not deduct health during the debuff cooldown).
11. Countdown staircase
    i. Use the DetectionEnterWindow class to detect if the stairs have entered the window.
    ii. If so, react() of the bound ReactionChangeImage will be executed, so that the image of the staircase will be replaced with yellow and red in order, and finally disappear.
12. black hole
    i. Use the DetectionTouching class to detect if the black hole has been touched by the player.
    ii. If it is touched, it executes the react() of the bound ReactionGameover, ending the game.
13. Binary mode
    i. Blue doodler controls: A to the left and D to the right.
    ii. When the top doodler is halfway through the screen, the lower doodler will move down: move the signal that touches the halfway side of the screen and the other doodler down slots in QMainWindow .

Two. Code Explanation:

- Mainwindows.cpp
  - Initialize and generate buttons and images, and connect the generated buttons to the screen mode.
  - Create an empty destructor.
  - In the single-player screen (mode), clear the score first, and hit the player to the bottom line and no blood

    game over to connect the binding, start the timer and set whether the previously generated button wants to be displayed.
  - In the two-player screen (mode), clear the score first, and put the signal on the top player and the other doodler to move down to the slots Connect the binding, also bind the player to the bottom line and the game over, start the timer and set whether the previously generated button wants to be displayed.
  - On the game pause screen (mode), pause the time and display the buttons to continue the game and return to the main screen.
  - Single-player and two-player buttons are displayed on the main screen.
  - Pause the timer on the Game Over screen (mode), add an out sound and display the button on the main screen.
  - Use timerEvents to continuously update all features.
  - Use paintEvent to draw the desired object and display different objects depending on the mode you are in:
    - Display the cover in the main page mode.
    - In single-player mode, the score and current HP are displayed; The mode is out, and the final score and HP are displayed.
    - In the two-player mode, the score of two players is displayed; When a player hits the baseline and goes out, the score of the two players is shown and who wins.
  - If you are on the game screen (mode), press the mouse and shoot in the direction of the cursor.
  - The left and right arrow keys of the keyboard control the direction of the doodle of player1, and player2 if it is in two-player mode

    Control with the AD key of the keyboard, or press the esc key to pause .
  - Generate buttons.
  - Erase old data.

- detector.cpp
  - Declare and initialize
  - Create a destructor to clear mover, passenger, and timer
  - Link the detection command received by the string (target) with the reaction
  - Draw a picture and set objects to move and update

- gameobject.cpp
  - Declare and initialize
  - Create a destructer for deleting movers and images

- Pass the path of the image you want to paste into the profile
- Draw a profile
- Settings are updated and objects are moved

- mover.cpp
  - Initialize the argument and write out its set and get function
  - Write out the relationship between the object's position, horizontal displacement velocity, vertical displacement velocity, and acceleration.
  - Discuss the behavior of the subject, horizontal and vertical are the same concept, if it is none, no action is done; If it is rebound, change the original velocity to the opposite direction (rebound); If it is passThrough, the object crosses to the other end of the border.

- bullet.cpp
  - Initialize and announce first, set boundaries and set the case of the moving object's behavior, bind the update to time, and pass in the image.
  - Create an empty destructor
  - Set the bullet flight speed and frequency (update speed).

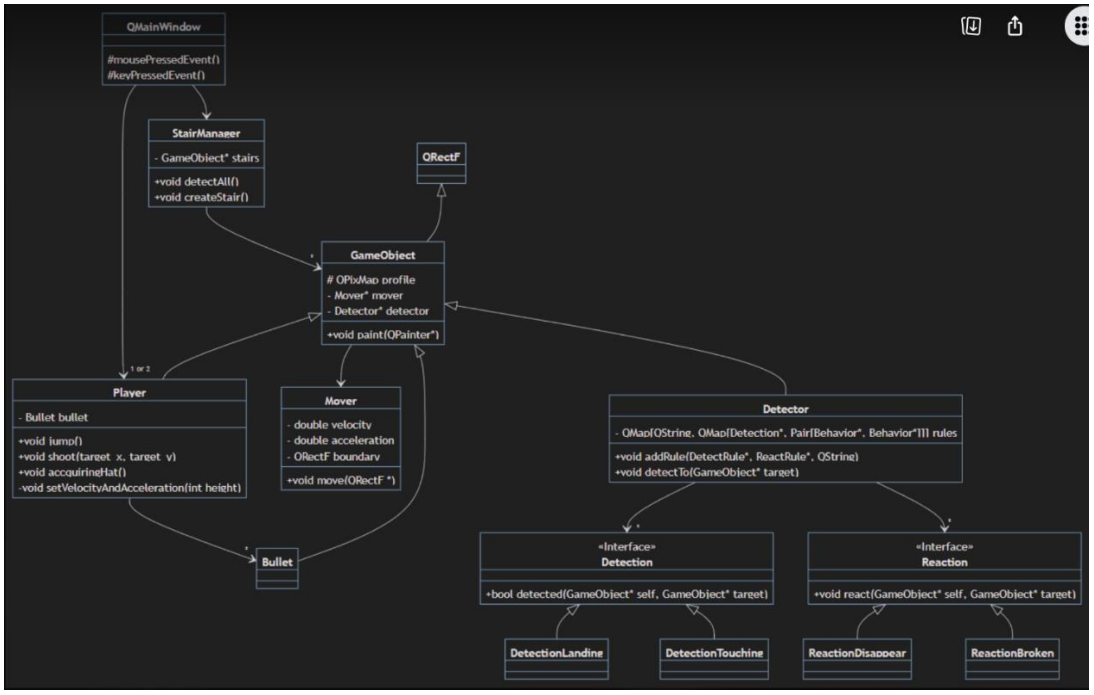
- player.cpp
  - Initialize and announce first, set boundaries and set the case of the moving object's behavior, pass in the image, and start jumping.
  - Establish a destructor to clear bullets
  - Bind bullet moveDown and exceedsTopBoundaryBy signals, apply bullet ejection frequency and velocity, and add sound effects.
  - Calculate the vertical acceleration using a physical formula
  - Set the shield and presence time, and execute the timer when calling.
  - Set the path of the image
  - Use physics formulas to figure out the speed of the jump
  - Set a cooldown time when attacked, and execute a timer when calling.
  - For character state discussions, the vertical velocity and acceleration are updated when falling; Continuously update the character height, when the character is more than half the height of the screen, the update speed; If it is not in the branded state, it will continue to move. And according to the direction of the character, the picture is updated, and the optimal height is continuously updated to facilitate the calculation of the score.
  - When the bottom is outside the lower boundary of the screen, it is judged to have touched the bottom.
  - If attacked, lose 1 HP and enter the cooldown of the attack, and the game ends when the HP is below 0; If there is a shield, it does not lose health, but the shield turns false (disappears).
  - Set the status of attack that can be attacked
  - Draw the shield
  - Character movement

- stair_manager.cpp
  - Initialize and set the top of the zero and new steps to be twice the height of the screen.
  - Fixed the zero ladder and randomly generated until the highest ladder exceeds the target height (twice the height of the screen).
  - Set the spawn height, type, unlock height, and spawn height interval of each ladder.
  - When the maximum height is less than the unlocked height but the next step spawn height is greater than the unlocked height, pass the size of the incoming ladder as the number. Otherwise, the different tiers will be discussed, and the stair.first used is used to deposit the object itself; stair.second is used to store object types and generate them. However, items and monsters will only spawn on the Basic Ladder and the Movement Ladder.
  - Items are divided into various types and randomly generated according to each category.
  - Print a stepped pattern
  - Deletes the object itself
  - Generate ladders and props: Follow the command (detection) and bound reactions to make movements, such as ladder automatic bounce (ascending and falling), sound when the character falls on the ladder, change the picture when the character falls on the ladder, and disappear the ladder. Or switch to airplane mode when you touch an object (move faster), and so on.
  - Generate a black hole: When the instruction is contacted, the reaction is out.
  - Spawn Monsters: There are four types of monsters, of which there are three commands, and when they land on monsters, the character will jump up, make a sound, and the monster will disappear; Touching from other directions will be attacked; Monsters will disappear when attacked by bullets.
  - The screen update is done by moving down the ladder and then continuing to create new ladders.
  - The destructor that creates the ladder is used to clear the ladder.

- Detection: It includes the following types of detection, focusing on the self (object) and the target (role).
  - detection_enter_window: Whether to enter the screen
  - detection_landing: Whether the character falls on the object.
  - detection_touching_without_landing: This applies to contact other than drop-offs.
  - detection_touching: Whether the character has been in contact with the subject.

- Reaction: Contains the following reactions, which also focus on the self (object) and the target (role).
  - reaction_change_image: Use the timer to make the image replaceable
  - reaction_disappear: Makes the object disappear from the frame
  - reaction_gameover: The character is out of the way to the reachBottomBoundary() signal.
  - reaction_make_sound: Sounds sounded.
  - reaction_player_be_attacked: Indicates the character's reaction when attacked, and executes the player
    beAttacked().
  - reaction_player_buffed_jump_height: The character performs a higher jump response.

- reaction_player_jump: The character reacts to jumping.
- reaction_player_set_flying_state: The character enters the flight state.
- reaction_player_shielded: A character's reaction when they have a shield.
- **Three Architecture Introduction:**



The above is our UML architecture diagram, and the following is a brief introduction to the substructures and the relationships between each file:

1. QMainWindow:

For the main presentation and execution of the game, include the details of the major functions including constants.h, gameobject.h, mover.h, player.h, detector.h, stair_manager.h,

reactions/reaction_make_sound.h, which connects the main subfiles to present the complete program content.

2. Stair_manager:

This file is a ladder construction, including the picture and the automatically generated part. The include file contains the objects that appear on the interface and the representations of the objects that react to them, including constants.h (to set the size of the object) and gameobject.h (to control the movement of the object)., player.h (including Doodle jumps, attacks, cooldowns, etc.), bullet.h (bullet behavior), detector.h (Detects if the doodle has come into contact with other objects.)
and rule_instances.h to detect Doodle or object movements, reactions, and game sounds
reactions /reaction_change_image.h (change the screen according to the Doodle reaction).
reactions/reaction_make_sound.h (substitution sound) to establish a link between each object and the interface.

3. GameObject:

Mainly controls the movement of objects, the include file contains mover.h (the behavior of the object's left and right translation, up and down acceleration motion, and bounce), so that it can make the movement aspects of objects in the GameObject meet the requirements.

4. Player:

This file includes constants.h, gameobject.h, bullet.h, detections/detection.h, reactions/reaction.h, player.hto combine the operation of the game with the objects in the gameobject, as well as bullet, detection and reaction.

5. Detector:

Dividedintodetection andreaction,Detection mainly detects whether the object has been touched,Behavior is divided into touching and landing,Monsters and stairs have to judge whether they have been stepped on,And item to determine whether it has been touched or not, so that it can react in subsequent reactions, this file include constants.h, gameobject.h, bullet.h, reactions/reaction_make _sound.h, LinkDoodle to objects.

Thereaction is based on the result of detection, so it needs to include gameobject.h, and Doodle's actions include changing images, disappearing, game over, andDoodles are attacked, jump height increased, flying, and shielded.

For the detection andreaction parts, we do it by dividing the behaviors into bins Touching andlanding detection and reactions .

6. Bullet:

This file constructs the behavior of the bullet, including the action of the bullet, the speed at which the bullet travels, and the frequency control of the bullet fired by the Doodle, including gameobject.h, mover.h, constants.h because the Doodle is usually in a moving state when firing the bullet.

7. Mover:

It is mainly responsible for the left and right translation, acceleration motion, bounce behavior, and when it touches the left and right sides of the screen, it will travel to the other side, including constants.h because it is combined with the object specifications.

Four. Difficulties encountered and solutions:

1. Stuck problems caused by Broken stair:

Issue: Ifthelocation of the Broken stair is not well designed, it may cause a stuck, as the distance between the two platforms may be greater than the Doodle's jump ability (320pixels) after the Broken stair in the middle disappears, causing the Doodle to be unable to move.

Solution: Design the location of the broken stair to avoid stuck

i. InStairManager::maintainAppearance(), use the switch to determine if the type of staircase to be created is broken stair (this part uses your own defined enum MantainedCategory).

ii. If it is a broken stair, it will determine whether the height difference between the front and rear stairs (exceeding the jump height set by the player) will be determined

iii. If the height difference is not too large, replace the stair with a broken stair

iv. If the height difference is too large, an additional building will be created in addition to the Broken Stair

ladder to solve the problem of stuck customs

2. Injury events and cooldown design:

Issue: Doodle may have a continuous charge of health when it touches a monster. Workaround:

i. In void Player::beAttacked(), first determine whether attackable() is true, if so, it means that Doodle is under attack, and then it will put hp minus one and call resetCoolingDownToAttackAfter(3000) to start a three-second cooldown.

ii. Attacking the player during the cooldown will not deduct HP, avoiding the issue of continuous health deductions.

3. Reuse code: Determine touching and landing

i. Monsters, stairs, and items have duplicate judgment events through observation, for example, both monsters and stairs have to determine whether they have been stepped on (landing), and both items and monsters have to judge whether they have been touched (touching).

ii. While it's usually intuitive to create classes for each of these three, there will be duplicate code between them, which can be difficult to maintain. Therefore, we encapsulate the behavior of judgment into a class, and encapsulate the detection of GameObjects into a class

Detector, each different detector can use the addRule function to add its own detection behavior, which solves the problem of duplicate code mentioned above, and thus optimizes the program content.

4. When the player rises halfway through the screen, the window needs to be updated and moved down

i. Using Qt's native signal and slots syntax, when the Player rises to halfway through the screen, it emit exceedsTopBoundaryBy() to turn the Player To get more than half of the value of the picture, this signal is connected to the stair_manager's viewDown() slot in MainWindow::initGame(), and thus, via viewDown(), all stairs and monsters or items on stairs will move down to the height specified by the player.

5. Detection and reaction Broken stair Crash issues:

Problem: In the beginning we wrote the entire Doodle reaction behavior together, but when we debug it, we found that it was difficult to correct the error when it appeared, and often the entire program architecture had to be changed as a result.

Solution:

The reaction of Doodle touching an object and giving a reaction is divided into detection and reaction, and the functions are subdivided and divided into bins to reduce the difficulties debugging.