FOOP21 Final - LH-Tan

B06902019 洪佳生 B06902024 黃秉迦 B06902066 蔡秉辰 B06902074 柯宏穎

Introduction

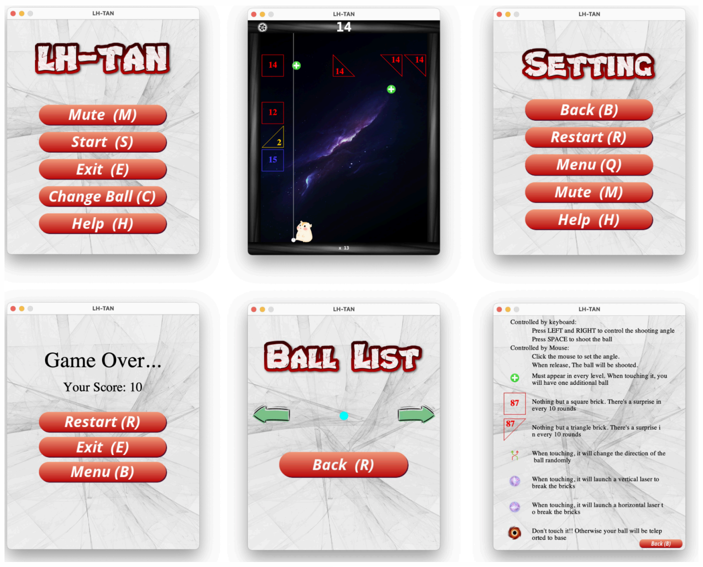
This is a little game similar to BB-Tan, a special brick elimination game. There are lots of bricks with certain counts. The player has to remove them before they fall to the bottom. There are several props in the game, some of them can help the player to eliminate the bricks faster, but some of them would interfere with the player.

Architecture

Introduction

All pages and Page Controller

The model is composed of 6 pages, Main Page, Game Page, Setting Page, Help Page, Settlement Page and Change-Ball Page, controlled by a static class called `Page Controller`.



Controlled by keyboard:
Press LEFT and RIGHT to control the shooting angle
Press SPACE to shoot the ball
Controlled by Mouse:
Click the mouse to set the angle.
When release, The ball will be shooted.
Must appear in every level. When touching it, you will have one additional ball
87 Nothing but a square brick. There's a surprise in every 10 rounds
87 Nothing but a triangle brick. There's a surprise i n every 10 rounds
When touching, it will change the direction of the ball randomly
When touching, it will launch a vertical laser to break the bricks
When touching, it will launch a horizontal laser t o break the bricks
Don't touch it!! Otherwise your ball will be telep orted to base

If each page wants to switch to another page, it can call `PageController.switchPanel([page_name])`; if it wants to show a page, it can call `PageController.showPanel([page_name])`. All pages will inherit from `ShowPage`, which inherits`JPanel` and implements `KeyListener` to bind key events.

- Main Page: The entry of the game. There are several options in the Main Page, muting the sound, starting a game, exiting the game, changing the ball and showing the help. All actions have their own function, binding with `keyRealease` or button `onclick`.

- Game Page: The core design of the game. One can enjoy our game on this page by continuously shooting the ball to smash the descending bricks to win a higher score. The detail shows in below:
  - It contains two controllers to decide the shooting angle and when to shoot. Once controllers receive the action to shoot (press SPACE or onclick mouse), it'll send an event `Shoot`. `Game.shoot()` will be invoked because `Game` register the event `Shoot` with `Game.shoot()` to `EventHandler`
  - In each round, `Game` will use `GenObstacle` to generate a row of bricks and obstacles.
  - After the user shoots the ball:
    - In each time of rendering, all of the obstacles will check if any ball touched it and perform some actions if they collide. All the balls will update their position. In addition, it'll check which obstacles need to vanish (e.g. the number of bricks is equal to 0).
    - After dealing with collisions and updating the state of balls and obstacles, it'll call `repaint()` to render.
    - The program would sleep 1 microsecond to avoid the ball running too fast.
    - While shooting, the ball would wait for the previous ball to run enough distance before it shot out. This is used to avoid the ball to stack together.
    - It will keep rendering until all of the ball is back and the character reaches a certain place.
- Setting Page: The page provides several options after the game is started. One can go back to the game, going back to the Main Page, restarting a game, muting the sound or showing the help in this page. All actions will have its own function, and bind with `keyRealease` or button `onclick`.
- Settlement Page: The page takes control when the player loses the game. Showing the score and providing several options for the player, such as restarting the game, exiting the game, or going back to the Main Page. All actions have their own function, binding with `keyRealease` or button `onclick`.
  - The score is acquired from `GameInfo`.
  - When the user goes back to the main page, it removes the `Game` from `PageController` first before switching to the main page.
- Change-Ball Page: The page to determine the ball played in the game.
- Help Page: Introduce the game, including how to control and what the objects are.
  - All bricks and props are generated from `GenObstacles`.
  - The descriptions of obstacles are obtained from `info()` of the obstacles.

Balls

The class `Ball` controls the ball's movement. It records each ball's position, radius, moving direction, moving speed, and color.
Notice that the ball will also need to implement the function called `draw`. This method will be called when rendering in the game page.

Brick and Props

- Introduction: all of the bricks and props will be inheritance from `Obstacle`; the `Obstacle` object should handle:
  - Vanish time: when should the obstacle vanishes.
  - Collision judge: whether the ball touches this obstacle.
  - Info: the description of this obstacle.
  - draw: draw the object when rendering.
- Brick:
  Our bricks have two shapes, square and triangle. When the number in the brick decreases, the color will change gradually. (e.g. change from red to yellow)

    - SquareBrick:
        - The brick will handle the collision as if it is a rounded rectangle.
    - TriangleBrick:
        - The brick will handle the collision as if it is a rounded isosceles right triangle.
        - This class has an additional variable called `direction`, since there are four different directions of the isosceles right triangle.
- Props:
  There are several props to make the game more interesting. Once you touch the prop except addBall, the prop would vanish at the end of the round.
    - AddBall: Add single ball for player. It must appear in each round.
    - RandomShoot: The ball will change direction randomly (without downard). When the ball's direction is closed to horizontal, this prop would appear around the ball to avoid the unlimited round.
    - Laser: There are two types, horizontal laser and vertical laser. Once the ball touches the laser, A light will hit all of the bricks horizontally or vertically to the laser. It will invoke an `Event` to reduce the brick's count simultaneously to avoid the ball hitting a brick which disappears after hitting by laser.
    - BlackHole: It will absorb the ball. The ball will be sent back to the starting point.

Event

`EventHandler` is a static class handling the events, such as shooting or invoking lasers. It works like a signal handler. When a component invokes the event with the `EventHandler`, the event listener which registers the same event on `EventHandler` will be invoked.

Controllers

`Controller` is responsible for drawing the shooting trajectory for the player, or the player can't ensure which direction the balls will be shooted. When the player shoots, `Controller` will invoke the event `Shoot`. When the player modifies the shooting direction, `Controller` will invoke the event `DrawTrajectory`.

- KeyboardController: A class listens to the keyboard events, decreasing the angle by 0.5 degrees when the player presses `Right` and increasing the angle by 0.5 degrees when the player presses `Left`. Shooting when the player presses `Space`.
- MouseController: Alternatively, you can directly click anywhere in the game window to redirect the shooting direction.

Game Info

A static class to store the game info and status. It contains the numbers of the ball, level, and the ball starting point.

Sound effects and background music

We have a static class called `MusicPlayer`. It'll add all `.wav` files in `music` folder into the mapping.

- When the program is started, it will load all the music in the `./music` by calling `addMusicFromDirectory("music")`. After loading all music, it constructs a mapping which maps the music name to the music for other components in the program.
- After the user opens our game, it'll play the background music by calling `MusicPlayer.playBackground([music_name])`.
- Each time when we want to let the event make some sound (e.g. when the ball touches the brick), it'll execute

`Musicplayer.play([sound_name])`.

- To avoid the effect of sound playing too frequently and making the game stuck, we design a function `isRunning()` to check whether this sound is still playing now. Our ears can't distinguish such close sounds while playing. It can reduce the load on playing music.
- When player asked to mute/unmute the musics in the Main Page or Setting Page, `Musicplayer.mute()` will be called to mute/unmute all the musics

Image Handling

We have a static class called `ImageHandler`. It'll add all image files in `image` folder into the mapping.

- When the program is started, it will load all the images in the `image` by calling `addImageFromDirectory("image")`. After loading all images, it constructs a mapping which maps the image name to the image for other components in the program.
- By calling `ImageHandler.getImage([image name])`, a component can get an image and draw it.
- By calling `ImageHandler.getOpacityImage([image name], alpha)`, a component can get an opacity-adjusted image and draw it.

Character

A character at the bottom. To avoid the character occluded by the wall, the character would stand by the right of the ball if the ball is at the left of the character, and vice versa. And The character would always face the ball.

Configuration

`Config` is a static class with public members which store all configurations, including the size, position, color and probability of appearance of the objects.

The advantages of the design

- We try to make all of the objects become OCP. It is easy to add a new obstacle or prop. Just inherit the abstract class we designed and modify their probability to appear in the game.
- If an object wants to play a new sound effect, just add a `.wav` file in `music`. That object only needs to call `MusicPlayer` to play it.
- The collision can be applied on any polygon. The only thing to do is setting the vertices clockwise.
- We use `PageController` to control all pages, it's very easy to add a new page and let it interact with other pages in our design.
- If we want to change some configurations, we don't need to access several classes to modify them. Instead, we only need to change the configuration in `Config`.

The disadvantages of the design

- Even we try hard to make the UI and setting be more flexible. Some of them are constant variables to beautify the game.
- If the balls touch the obstacles too frequently, the sound will play over and over, which will make the game lag. The only way to avoid it is to mute the music.

How to play

- Game page introduction:
  - The number on the top of the game page denotes the current level, at the same time, it's your current score.
  - Each time when the ball touches the brick, the number in the brick will be deduced by 1.
  - The game is over when the brick touches the bottom.
- Basic control:
  - To switch to another page, the user can either click the button or press the key (each button will show its respective key)
  - When you are on the game page, you can always click the upper left gear or press "s" to go to setting page.
- In the game page:
  The user has to adjust the shooting angle, then shoot the balls.
  - Control by keyboard: use the LEFT and RIGHT key to adjust direction; press SPACE to shoot!
  - Control by mouse: click the mouse to change the shooting direction, or drag the shooting line to the direction you want, the ball will be shooted when you release your mouse.
- Other suggestion:
  - If you think that the animation is lagging, you can mute the music (can be set on main page or setting page).
  - You can try different balls on the main page.
  - Every 10 levels the brick will double its count, be aware of that!
  - Try your best to get the highest score! Good Luck ~~~

Distribution

- Structure: all members
- Implementation: all members
- Interface design: all members
- Sound effects: all members