Game: Ninja Dash
Members:
Asilbek (aismatil@u.rochester.edu)
Earvin (echen50@u.rochester.edu)
Kate (nkim32@u.rochester.edu)

Game Description

Application

In the application, we've imported necessary packages, such as javafx.application.Application, javafx.fxml.FXMLLoader, javafx.scene.Scene, javafx.stage.Stage, and import java.io.IOException to use later. Our HelloApplication class extends Application, the base class. Next, we used a start Method to enter the application. The FXMLLoader loads it into the application and allows us to use FX. The scene ensures that the FXML runs in the window. The following sets the title of the game and shows the stage. Additionally, we choose to not make the game resizable to conserve proportions.

Controller

At the top of the code, we imported all the necessary packages to utilize later. Next, in the main class HelloController we declare all variables, including ones imported from FXML. Our gameLoop helps us measure time as our game is animated - done through an import of AnimationTimer. Next, we declare variable r for random to be used for the randomized chainsaws later, double time to measure the game's time when in play, and int score_num to track the player's score. Rectangle time_block is used here to track the player's limited time, which decreases as the player plays. We then declared an ArrayList of Rectangles that contains all of the chainsaws the ninja must avoid.

We then imported images through Library Image such as our ninja character and the column log. We then call a method initialize that ensures the game actually plays. It runs once before the game begins and is mandatory. Within the method we use method update that updates parts of the game as time increases. Here is where we fill the limited time block with red color and add the rectangle to the display screen.

pressed is the function imported directly from FXML and is ran every time the user clicks a key on their keyboard. We implemented a condition if(game_on){, where the game is on when the ninja is still "alive" and when it collides with the chainsaw the game is off. No matter which key (left or right) the player clicks, the score increments by one with the if loop. Within the same loop, we have if(score_num % 5 == 0) time_decrease+=0.01 which increments the speed at which the time decreases and the following loop increases the time, but cuts it off when at its max to prevent the rectangle/box from overflowing. Next, if(ninja.getX() < 200) { checks the location of the ninja and changes the animation accordingly (facing right and left). Our create_obstacle() function creates new chainsaws, and then sets it to true to allow it to move. We then implemented the previous when the ninja is on the other side of the screen.

Our else statement runs when the player loses the game, it resets everything (the score the time, etc. to 0) and removes the start screen as well as the previous obstacles to prevent overflow. The following else statement allows players to enter a 3 letter “name” - a parallel to many retro video games - for the leaderboard. The conditions then state that the name be less than 3 indexes and the score to be larger than 0. We then receive a String of the user's name and display it in our leaderboard in the game. The following if statement creates a new leaderboard after the user types in 3 letter name, and we later call previous names and scores from previous plays into this ongoing leaderboard. The last thing in this block displays the leaderboard over everything in the background.

We then created a String make_leaderboard() to use library File to call previous scores and names and assigns it to variable “data.” Each name and score is “split” with data.split to display distinct scores/names and is used in an ArrayList that contains a sorted map. In this map, the key is the score and the value represents the name of the player. The map uses create__sorted_player_map, which receives ArrayList of a String of the player’s data and sorts the score from the highest to lowest. The following for loop ensures that the first three elements are letters while the last three are numbers, representing the name and score, respectively.

We then utilize a for loop that displays only the top (highest score) 3 places in the leaderboard by outputs the place of the current player’s score. Then, all of the scores and names are updated in the file which allows for future scores/names to be added and parsed.

Next, the handle_key_press method handles the input of the player name by allowing players to use the backspace key to edit their 3 letter name and prints out the updated one.

The method move_obstacles moves all of the obstacles - chainsaws - we have and incrementally moves it down by one step (125/3 pixels) using animations. It also removes chainsaws that are out of the screen/plane to prevent overflow.

Here we call the random variable r from before to create obstacles method that creates a new obstacle in our ArrayList by choosing left and right by random and creates certain amounts of obstacles on that side (0-3). We also imported self-drawn pixel pngs of our chainsaw, facing right and left depending on the side it’s on. When it switches sides, it doesn’t increment down for one step, allowing the player to react and avoid the chainsaw. Additionally, we imported the media player and libraries of JavaFX and used the duration library to create a whoosh and a game over sound effect.

Our collisonDection method is a boolean method that checks each obstacle through a for loop for collisions through the intersects method, running each frame.

Next, we have a load method that displays one time and at the start of the running of the code, setting the ninja's position to the default one.

Our update method runs each frame as declared earlier. It checks for the on/off (dead/alive) status of the game. If it’s on the time increments by one. We then nest multiple if statements to handle animations

(moving from left to right). This method also detects for collisions in addition to displaying the score's incrementation. It then decrements the limited time the player has with an if statement.

We then created a resetGame method that is called when the time less than 0 or when the ninja collides with a chainsaw.

The set_start_screen method displays the elements of the start screen such as the player name, score, leaderboard, starting text, background, and the additional texts that appear. The following is a remove_start_screen to remove the screen when the enter is pressed.

FXML

This contains all of the elements that are displayed on the screen, from the pixel imports to the background, to the rectangles.

Project Fulfillments

Flourishes

- **Randomized stationary obstacles that have negative effects:** our "chainsaws" that protrude out of the log are randomly generated and ensure that each play is a unique experience.
- **Sound effects:** we implemented sound effects for when the ninja moves as well as when it hits a saw.
- **High scores list:** after the first play, players are allowed to type 3 letter "names" that show up on the leaderboard list. The list appears each time the ninja touches a saw and "dies"
- **Make it look cool:** our time-dependent bar at the top left of the game, as well as the score counter on the top right both aim to create a more competitive environment and enhance user experience. Furthermore, our music and background scenery immerse users into the world of Ninja Dash

Outside resources

We used the internet to clarify concepts for us and help us implement things that weren't explicitly discussed in class. These are namely, music and image importation, many aspects of the animation, utilizing FXML, collision detection, and the usage of maps.