# SUPER MARIO

**JavaFX Project**

# UMLs

## MainPage

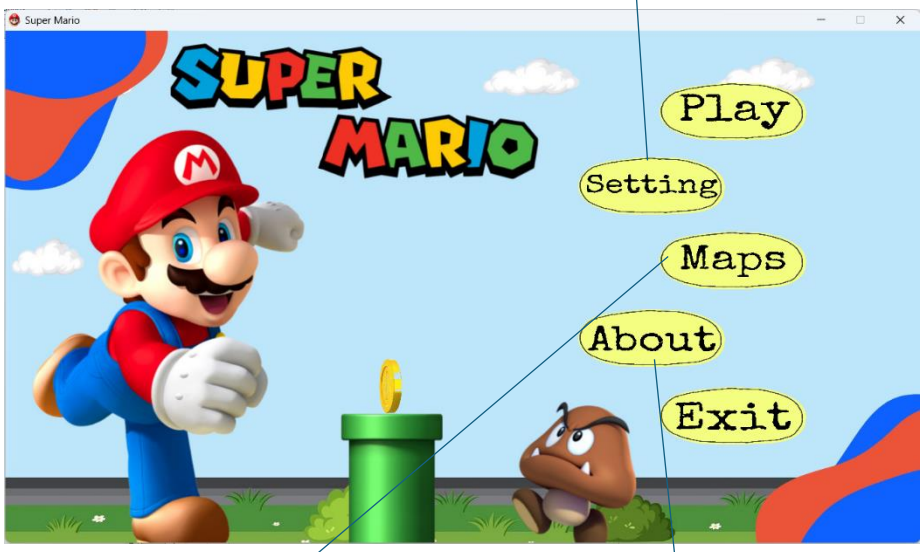
- MainStage: Stage
- WELCOME_TEXT: String
- Second_TEXT: String
- CHAR_DELAY_MS: int
- charIndex: int
- icon: Image
- Main_Scene: Scene

+ logout(stage: Stage): void
+ start(stage: Stage): void
+ thirdScene(Main_Stage: Stage): void
+ main(args: String[]): void

## SettingWindow

+ SUPER_MARIO: String
+ MARIO_LUIGI: String
+ FIRE_MARIO_LUIGI: String
+ MUTE_ON: String
+ MUTE_OFF: String
+ selectedCharacter: String
+ selectedMute: String

+display(primaryStage :Stage, previousScene: Scene) : void




## MapsWindow

- Crystal: String
- coralreefs: String
- horror: String
- selectedbackground: String

+ display(primaryStage:Stage, previousScene:Scene): void

## AboutWindow

**+ display(primaryStage : Stage, previousScene : Scene) : void**

## SuperMario

- MARIO_WIDTH: int
- MARIO_HEIGHT: int
- MARIO_START_X: int
- MARIO_START_Y: int
- LEVEL_GROUND: int
- GRAVITY: int
- JUMP_VELOCITY: int
- MOVE_SPEED: int
- marioX: double
- marioY: double
- marioVelocityY: double
- isJumping: boolean
- isMovingLeft: boolean
- isMovingRight: Boolean
- BLOCK_SIZE:int
- currentStandingImage: Image
- currentRunningImages: Image[]
- currentJumpingImages: Image[]
- currentSleepingImage: Image
- currentFrame: int
- frameCounter: int
- lastMoveTime: long
- SLEEP_TIMEOUT: long
- obstacles: ArrayList<Block>
- root1: Pane
- animation: Timeline
- PlayWindowScene: Scene
- marioImageView: ImageView
- backgroundImageView: ImageView
- textScore: Text
- hearts: int
- ScoreImage: Image
- HealthImage: Image
- ScoreImageView: ImageView
- HealthImageView: ImageView
- HealthText: Text
- FRAME_DELAY :int
- LEVEL_NUMBER :int
- root2 :pane
- backgroundCrystal :Image
- backgroundhorror :Image
-backgroundcoralreefs :Image
-Coins : Coin[]

……

…….

## SuperMario

…….

……..

+ getMarioimageview(): ImageView
+ setx(int x) :void
+ setY(int x) :void
+ MARIO_START_X(): int
+ MARIO_START_Y(): int
+ setbackgroundImageView(int x) :void
+ getx(): double
+ gety(): double
+ setVisiblemario(boolean state) :void
+ getMarioWidth(): double
+ getMarioHeight(): double
+ resetPlayer() :void
+ resetPalyWindow() :void
+ addToRoot2(ImageView heart) :void
+ DisplayPlayMode(Stage primaryStage) :void
+ setupGame() :void
+ setupAnimation() :void
+ update() :void
+ draw() :void
+ handleKeyPress(KeyEvent event) :void
+ moveBackground() :void
+ handleKeyRelease(KeyEvent event) :void
+ CheckWin() :void
+ handleMovement() :void
+ handleJumping() :void
+ handleAnimation() :void
+ handleSleeping() :void
+ checkBlockCollision(): boolean
+ checkBlockCollision(double newX, double newY): boolean
+ putBlocks() :void
+ makeCoins() :void
+ putCoins():void
+ CoinsScorePrint() :void
+ checkCoinsCollision() :void
+ setHoles() :void
+ getRoot1() :void
+ selectBackground() :void
+ selectCharacter() :void
+putHealthText() :void
+putBlocks() :void

# GameSounds

- jumpSound: MediaPlayer
- coinSound: MediaPlayer
- winSound: MediaPlayer
- loseSound: MediaPlayer
- levelUpSound: MediaPlayer
- fallSound: MediaPlayer
- playSound: MediaPlayer
- menuSound: MediaPlayer
- playSound_media: Media
- menuSound_media: Media
- jumpSound_media: Media
- coinSound_media: Media
- winSound_media: Media
- loseSound_media: Media
- levelUpSound_media: Media
- fallSound_media: Media

---

+ GameSounds()
+ playplaySound(): void
+ stopplaySound(): void
+ playmenuSound(): void
+ stopmenuSound(): void
+ playJumpSound(): void
+ playCoinSound(): void
+ playWinSound(): void
+ playLoseSound(): void
+ playlevelUpSound(): void
+ playFallSound(): void
+ stopAllSounds(): void

# LevelData

- LEVEL1: String[]
- levels: String[][]

---

# Hearts

- numHearts: int
- heartWidth: int
- heartHeight: int
- hearts: int
- heartimage: Image
- heartImgView: ImageView[]

---

+ Hearts()
+ displayHearts(): void
+ decreaseHearts(): void
+ getHearts(): int

# Block

- blocksImg: Image
- block: ImageView

---

+ Block(blockType : BlockType, x : int,y : int)

# Coin

- NUM_COINS: int
- Score: int
- GOOMBA_KILL_SCORE: int
- COIN_SCORE: int
- COIN_WIDTH: int
- COIN_HEIGHT: int
+ coin: Image
+ coinImgView: ImageView[]

---

+ getGOOMBA_KILL_SCORE(): int
+ getCOIN_SCORE (): int
+ getScore (): int
+ setScore(Score:int): void
+ Get_NUM_COINS (): int
+ setImagesView() : void
+ getCoinsView() : ImageView[]

# Bird

- startlineX: int
- startlineY: int
- endlineX: int
- endlineY: int
- bird: Image
- birdview: ImageView
- line: Line
- path: PathTransition

---

+ Bird(startX :int, startY : int, endX : int, endY : int)

# Enemy

- goombaImage: Image
- goombaImageView: ImageView
- dx: int
- Xstart: int
- Xend: int
- goombaMotionAnimation: Timeline

---

+ Enemy(Xstart : int, Xend : int)
+ goombamotion():void
+ checkMarioAndGoombaCollision() :void
+ displayGoomba() :void

# GameWin

+ WinScene: Scene

---

+ display(): Scene

# GameOver

+ overScene: Scene

---

+ display() : Scene

# UMLs explanation

## • MainPage

The MainPage class is the central controller of the Super Mario game application. It extends Application from JavaFX and contains the start method to initialize and display the main scene. This scene features buttons for playing the game, accessing settings, viewing maps, learning about the game, and exiting. The class handles user interactions with these buttons and utilizes JavaFX features for GUI layout and image display. Additionally, it provides functionality for logging out the user. Overall, MainPage serves as the primary interface for navigating the game's features and managing user interactions.

## • AboutWindow

- The code defines a method called display() in the AboutWindow class to show an "About" window in a JavaFX application.

- The window contains a background image and clickable images representing team members or contributors.

- Each clickable image is associated with a URL that opens the respective team member's LinkedIn profile when clicked.

- There's a close button to return to the previous scene.

- Sound handling is included to stop and play menu sounds based on a mute option.

- Overall, the code creates an interactive About window displaying team member information and providing links to their LinkedIn profiles.

## • SettingWindow

This Java code defines a SettingWindow class for a JavaFX application, presenting a settings window with character selection and mute options. It initializes character and mute settings, displays corresponding images, and handles user interactions for changing these settings. The window includes buttons to select different characters (Super Mario, Mario Luigi, Fire Mario Luigi) and toggle mute on/off. Sound handling stops and plays menu sounds based on the mute option. Additionally, a close button allows users to return to the previous scene. Overall, it creates an interactive settings window for configuring game options.

- ## SuperMario

The SuperMario class is the core component of a JavaFX-based Super Mario game application. It manages player movement, animation, collision detection, game state, and graphical rendering. Key features include handling player controls, setting up the game environment, managing animation frames for player movement, detecting collisions with obstacles, managing game state transitions, and integrating audio effects. It encapsulates the game's logic and mechanics, providing the foundation for an interactive Super Mario gaming experience.

- ## GameSounds

The GameSounds class in Java manages game sound effects using JavaFX's MediaPlayer. It initializes different sound files and provides methods to play, stop, or manage the volume of these sounds. Each sound effect, such as jump, coin, win, lose, level up, and fall, has its own MediaPlayer instance, and there's also a method to stop all sounds at once.

- ## LevelData

The LevelData class contains data for different levels of a game. It defines a two-dimensional array levels, where each element represents a level, stored as a string array. Each string in the array represents a row of blocks in the level. The levels are represented using characters where each character corresponds to a specific type of block or element in the game environment. This class allows for easy management and loading of level data, facilitating the design and implementation of game levels.

- ## Block

The Block class manages various types of blocks in a game environment. It loads an image file containing block textures and defines an enumeration for different block types. Its constructor initializes blocks based on type, position, and dimensions. Blocks are added to the game scene, each with its corresponding texture based on the block type. This class provides a structured approach for creating and handling blocks, contributing to the development of diverse and interactive game environments.

- ## Coin

The Coin class manages the coins in the game. It defines properties such as the number of coins, score values, and image dimensions. The class also provides methods to get and set the score, get the number of coins, and initialize and retrieve coin image views. Additionally, it contains constants for the score gained from killing a character (GOOMBA_KILL_SCORE) and collecting a coin (COIN_SCORE). This class facilitates the management and display of coins in the game environment.

- ## Bird

The Bird class manages bird entities in the game by defining their movement along a straight path from a starting point to an ending point. It utilizes JavaFX's PathTransition for animation, with a duration of 25 seconds, and repeats the animation indefinitely. The class extends Pane and uses an image view to display the bird's image.

- ## Enemy

The Enemy class manages the behavior of the Goomba enemy in the game. It extends the Pane class and uses an image view to display the Goomba's image. The class includes methods for animating the Goomba's movement within a specified area and detecting collisions with the player character, Mario. Upon collision, appropriate actions are taken, such as resetting Mario's position or updating the player's score. Sound effects are played during certain interactions. Overall, the Enemy class controls the Goomba's behavior and interactions within the game environment.

- ## GameWin

The GameWin class is responsible for displaying the win screen in the game. It creates a scene with a background image representing the win screen and an exit button. When the win scene is displayed, it stops all sounds if the mute option is turned off and plays the win sound. The exit button allows the player to return to the main page of the game. Overall, this class manages the visual representation of the win screen and user interaction with it.

- **GameOver**

The GameOver class is responsible for displaying the game over screen in the game. It creates a scene with a background image representing the game over screen and an exit button. When the game over scene is displayed, it stops all sounds if the mute option is turned off and plays the lose sound. The exit button allows the player to return to the main page of the game. Overall, this class manages the visual representation of the game over screen and user interaction with it.