CROISSANT BOOM Documentation

Created by

Ratima Klabprasit 6733218921

Sinsuda Rakporpiang 6733271021

Jiratchaya Kunyaphila 6733032421

Puckanut Laoarpasuwong 6733190921

2110215 Programming Methodology

Semester 2 Year 2024

Chulalongkorn University

CROISSANT BOOM

Video presentation: https://youtu.be/s_bsMrSidFo?feature=shared

Introduction

" CROISSANT BOOM " is a game designed to enhance focus and patience. Players must bake croissants for the appropriate period of time, without underbaking or scorching them. The goal is to control baking times while avoiding obstacles such as tomato bombs, in order to score as many points as possible.

Rules

1. Players must control croissants and place them in the oven, timing the bake precisely.
2. Each type of croissant has a different baking time and point value:
    - Regular Croissant: 5 seconds (100 points if served perfectly)
    - Salad Croissant (Healthy): 6 seconds (120 points if served perfectly)
    - Rainbow Croissant: 8 seconds (250 points if served perfectly)
3. If the baking time exceeds the limit by 2 seconds, the croissant burns and must be discarded.
4. If the croissant is underbaked and removed from the oven, it must be placed back in and the timer restarts.
5. Penalties:
    - Serving an undercooked croissant deducts 100 points.
    - Serving a burned croissant deducts 150 points.
6. If a croissant touches a tomato bomb, it burns immediately and must be discarded.
7. All actions (baking, discarding, and serving) must be performed within designated areas.
8. Croissants may randomly spawn on a tomato bomb and burn instantly. While you cannot control the spawn point, you can discard the croissant and start again.

Game Modes

There are two difficulty modes:

1. Easy Mode: 2 tomato bombs are randomly placed on the board.
2. Hard Mode: 5 tomato bombs are randomly placed on the board.

Controls

- W, A, S, D: Move the croissant.
- E: Place a croissant in the oven (must be in the oven area).
- Q: Cancel baking and remove a croissant from the oven (must be in the oven area).
- F: Discard a croissant in the trash bin (must be in the trash area).
- R: Serve a croissant (can only be used in the serving area).

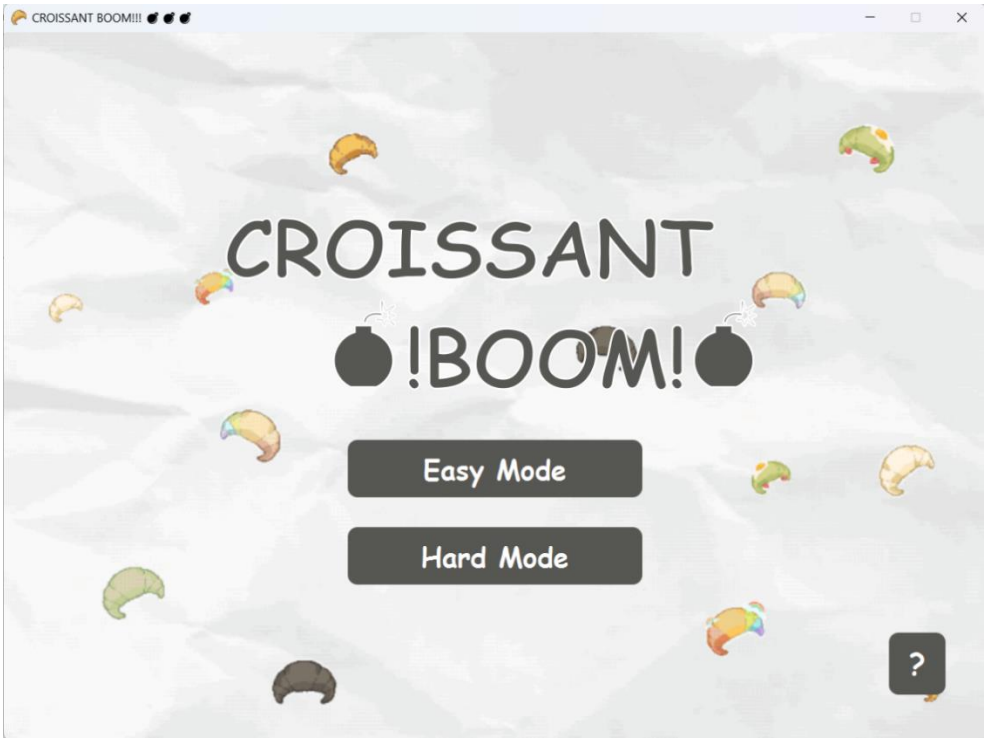
Objective

- Bake croissants perfectly to earn points.
- Avoid tomato bombs, as they will burn croissants.

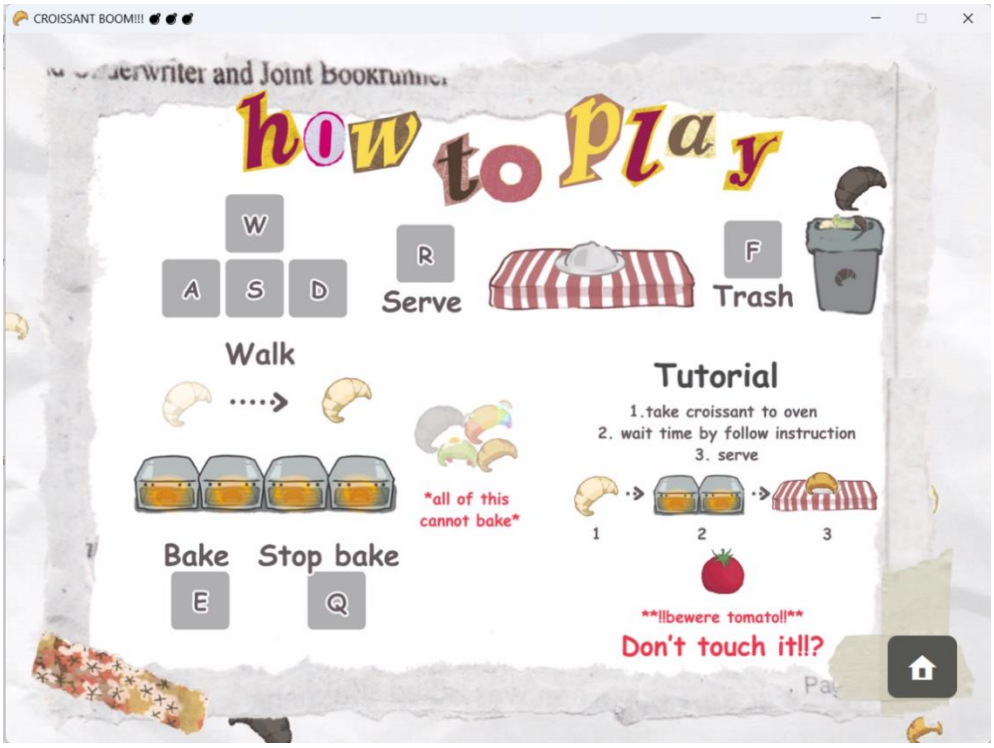
Game Over Conditions

- The game ends when the time limit of 1 minute and 30 seconds is reached.
- Points are calculated based on the number of successfully baked croissants, with longer baking times awarding higher scores

Main Menu (MenuCanvas)

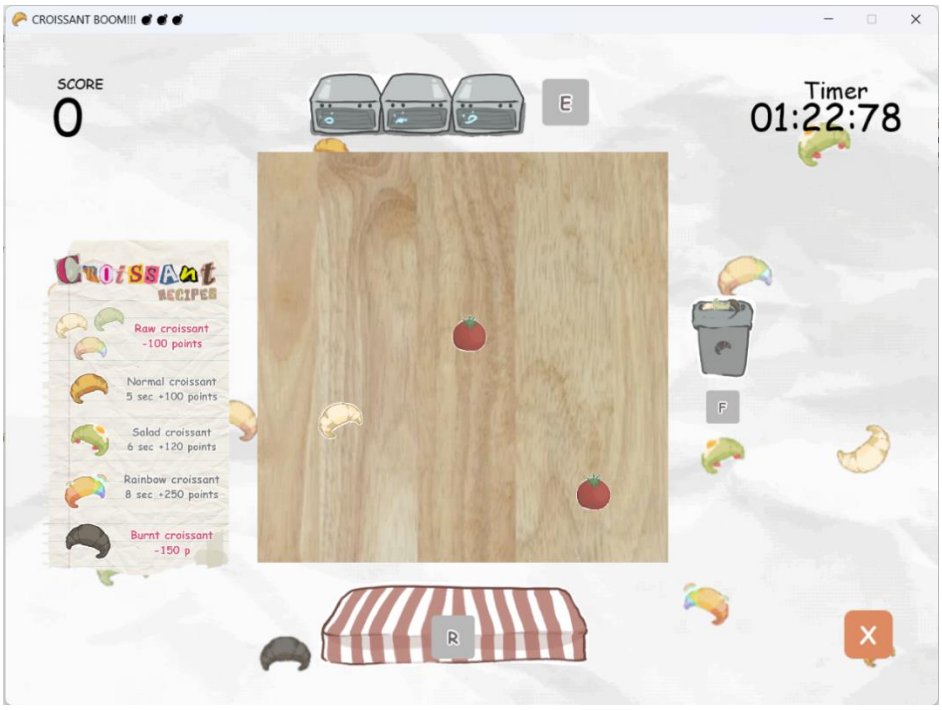


? Button: Displays instructions on how to play and a guide for the controls.

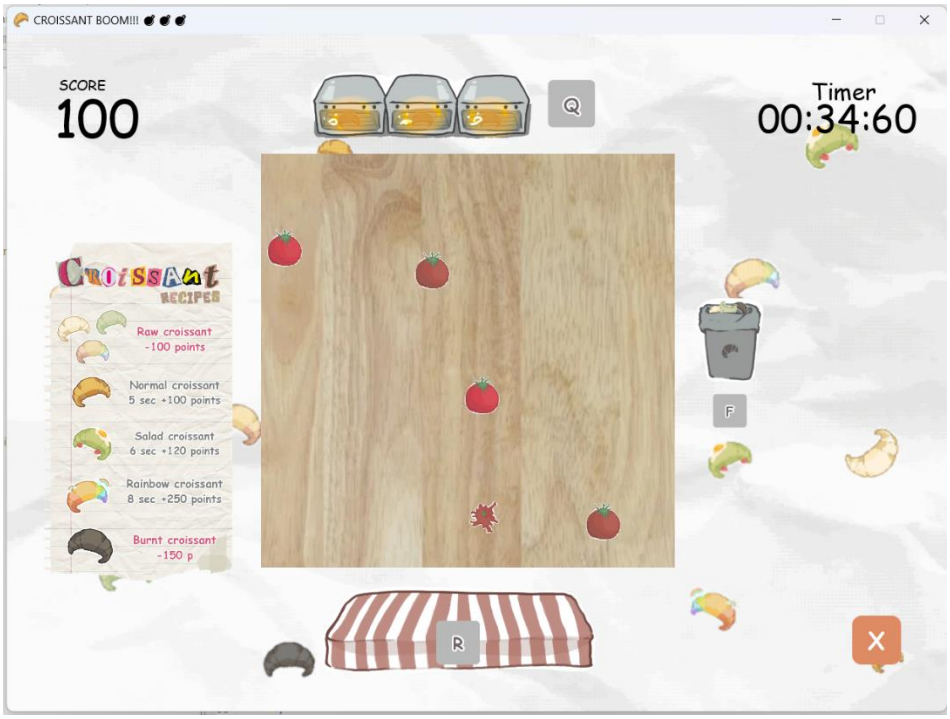


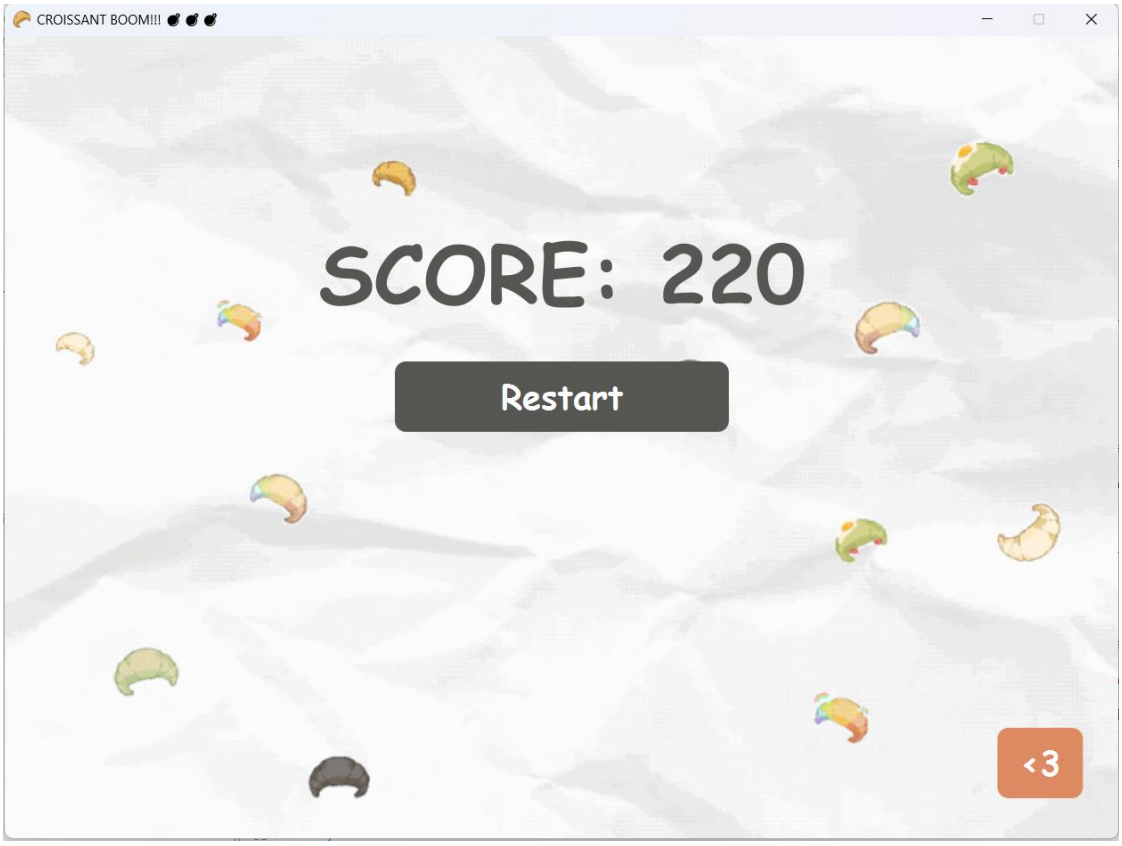
You can choose between two difficulty levels to play the game: Easy Mode and Hard Mode.

Easy Mode: 2 tomato bombs



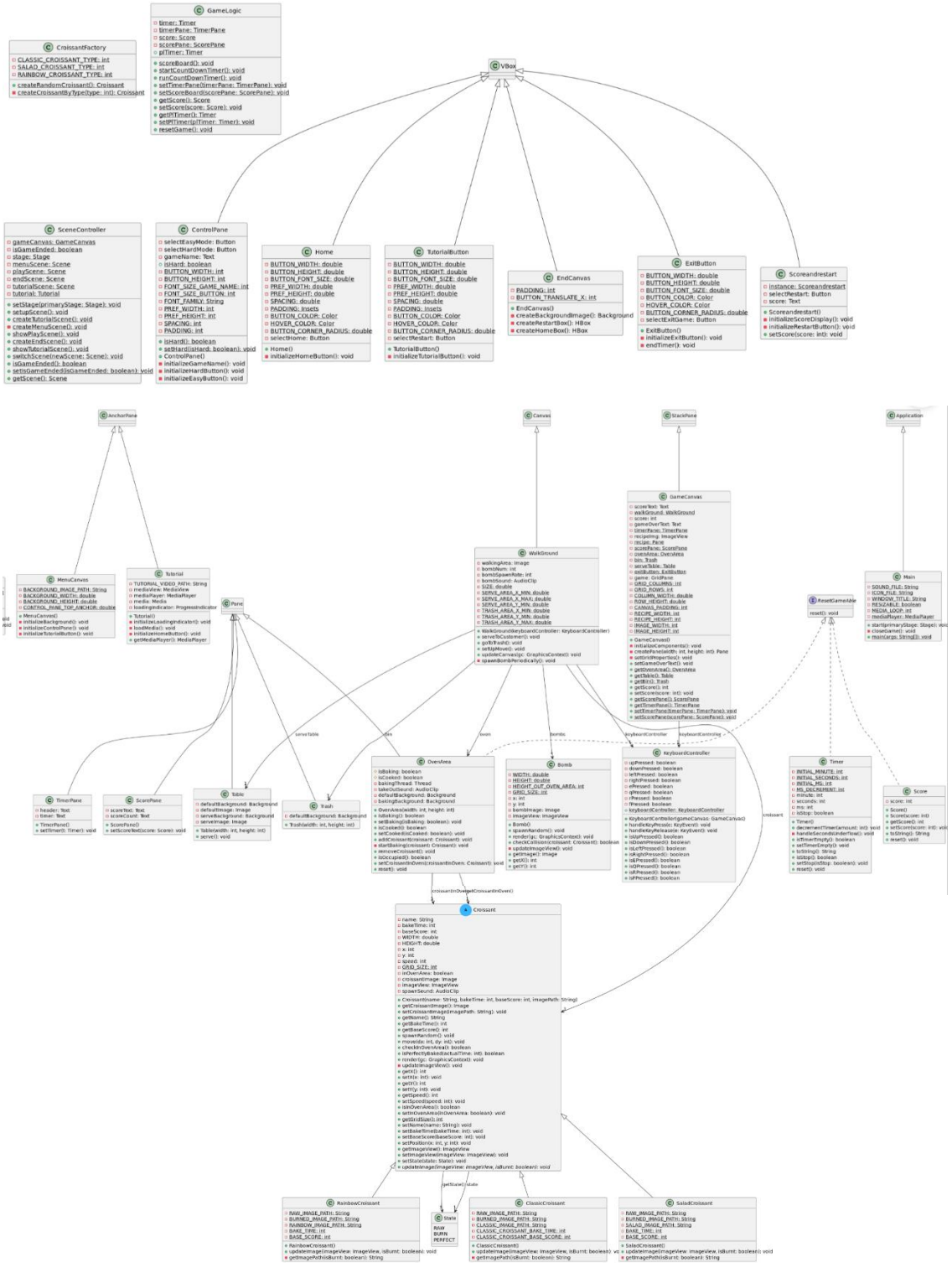
Hard Mode: 5 tomato bombs





At the end of the game, your total score for that round will be displayed. You can choose to Restart the game and try again to improve your score, or you can Return to the Home Screen to go back to the main menu.

Class diagram



Implementation Details

1. Package logic.game

1.1 public interface ResetGameAble

Interface for components that can be reset in the game. Any class that implements this interface must provide a reset method to restore its state to the initial configuration.

Methods

Name	Description
+ void reset()	This method will be responsible for resetting the state of the object to its initial configuration.

1.2 public class GameLogic

The GameLogic class is a part of the logic.game package and is responsible for managing the game's core mechanics, such as updating the score, handling the countdown timer, and resetting the game.

Field

Name	Description
- Timer timer	Represents the game's countdown timer.
- TimerPane timerPane	Represents the UI component responsible for displaying the countdown timer.
- Score score	Stores the player's current score.
- ScorePane scorePane	Represents the UI component responsible for displaying the player's score.
+ Timer plTimer	Timer for countdown timer method

Methods

Name	Description
+ void scoreboard()	Continuously updates the player's score on the UI while the game is running, and stops updating once the game ends.
+ void startCountDownTimer()	Initializes and starts the countdown timer in a separate thread.
+ void runCountDownTimer() throws InterruptedException	Initializes a local timer (plTimer) from the timer field. Decrements the timer value continuously and updates the UI in real-time.
+ void setTimerPane(TimerPane timerpane)	Assigns a TimerPane instance to the timerPane field.
+ void setScoreBoard(ScorePane scorepane)	Assigns a ScorePane instance to the scorePane field.
+ Score getScore()	Retrieves the current score stored in the score field.
+ void setScore(Score score)	Updates the score field with a new Score object.
+ void resetGame()	Resets the game state, including the score and timer.

1.3 public class KeyboardController

The KeyboardController class in Java (using JavaFX) is responsible for handling keyboard input in a game. It detects when specific keys are pressed and released, updating boolean flags accordingly. These flags can then be used by other game components to determine player actions.

Field : All fields are initialized to false.

Name	Description
- boolean upPressed	W key (move up).
- boolean downPressed	S key (move down).
- boolean leftPressed	A key (move left).
- boolean rightPressed	D key (move right).
- boolean ePressed	E key (used for entering the oven).
- boolean qPressed	Q key (used for exiting the oven).
- boolean rPressed	R key (used for serving food).
- boolean fPressed	F key (used for discarding trash).
+ KeyboardController keyboardController	A shared KeyboardController instance for global access.

Constructor

Name	Description
+ KeyboardController(GameCanvas gameCanvas)	Initializes the KeyboardController and sets up key listeners. Binds the event listeners to the provided GameCanvas object.

Methods

Name	Description
+ void handleKeyPress(KeyEvent e)	Checks for key codes (W, A, S, D, E, Q, R, F) and Sets the appropriate boolean field to true when a corresponding key is pressed.
+ void handleKeyRelease(KeyEvent e)	Checks for key codes (W, A, S, D, E, Q, R, F) and Sets the corresponding boolean field to false when a key is released.
+ boolean isUpPressed()	Returns the state of the W key (whether it is currently pressed).

+ boolean isDownPressed()	Returns the state of the S key (whether it is currently pressed).
+ boolean isLeftPressed()	Returns the state of the A key (whether it is currently pressed).
+ boolean isRightPressed()	Returns the state of the D key (whether it is currently pressed).
+ boolean isEPressed()	Returns the state of the E key (whether it is currently pressed). Used for entering the oven.
+ boolean isQPressed()	Returns the state of the Q key (whether it is currently pressed). Used for exiting the oven.
+ boolean isRPressed()	Returns the state of the R key (whether it is currently pressed). Used for serving food.
+ boolean isFPressed()	Returns the state of the F key (whether it is currently pressed). Used for discarding trash.

1.4 public class SceneController

The SceneController class is responsible for managing different scenes in a JavaFX-based game. It allows switching between various game states, including the main menu, gameplay, tutorial, and game-over screens.

Fields

Name	Description
- GameCanvas gameCanvas	Stores the canvas used during gameplay.
- boolean isGameEnded	Tracks whether the game has ended (true/false).
- Stage stage	Stores the primary stage where scenes are displayed.
- Scene menuScene	Stores the main menu scene.

- Scene playScene	Stores the gameplay scene.
- Scene endScene	Stores the end game scene.
- Scene tutorialScene	A reference to the Scene object representing the tutorial scene. Used for managing the tutorial screen.
- Tutorial tutorial	A reference to the Tutorial object, which manages the tutorial logic and content.

Methods

Name	Description
+ void setStage(Stage primaryStage)	Sets the primary Stage object where scenes are shown.
+ void setupScene()	Initializes the main menu scene and sets it as the active scene.
- void createMenuScene()	Creates the menu scene using the MenuCanvas class.
+ void showPlayScene()	Creates and switches to the gameplay scene.
+ void createEndScene()	Creates and switches to the end-game scene.
+ void switchScene()	Changes the active scene and updates the window.
+ void showTutorialScene()	Waits for the MediaPlayer to initialize, then stops and resets the media clip to start from the beginning, sets it to loop indefinitely, plays the media, and switches to the tutorial scene.
+ void createTutorialScene()	Initializes a new Tutorial object and creates a Scene with the Tutorial object as the root node, setting the scene size to 1000x720.
+ boolean isGameEnded()	Returns whether the game has ended.

+ void setIsGameEnded(boolean isGameEnded)	Updates the game-over status and displays a message if the game ends.
+ Scene getScene()	Retrieves the current active scene (play or menu).

1.5 public class Timer implements ResetGameAble

The Timer class is responsible for managing the in-game timer. It tracks time in minutes, seconds, and milliseconds, and allows for decrementing and resetting.

Fields

Name	Description
- int minute	Stores the number of minutes for the timer.
- int seconds	Stores the number of seconds for the timer.
- int ms	Stores the milliseconds for the timer.
- boolean isStop	Indicates whether the timer is stopped (true) or running (false).
- final int INITIAL_MINUTE	Stores the default initial number of minutes for the timer. Set to 1 minute.
- final int INITIAL_SECONDS	Stores the default initial number of seconds for the timer. Set to 30 seconds.
- final int INITIAL_MS	Stores the default initial number of milliseconds for the timer. Set to 0 milliseconds.
- final int MS_DECREMENT	Stores the value by which the milliseconds will be decremented in each step. Set to 100 milliseconds.

Constructor

Name	Description
+ Timer()	Initializes the timer with a starting value of 1 minute and 30 seconds. Initializes ms (milliseconds) to 0. Sets isStop to true (timer is initially stopped).

Methods

Name	Description
+ void decrementTimer(int amount)	Decreases the timer by a specified amount in milliseconds.
+ boolean isTimerEmpty()	Checks if the timer has run out (i.e., all time values are 0).
+ String toString()	Converts the timer's current time (minute, seconds, and milliseconds) into a formatted string.
+ boolean isStop()	Returns whether the timer is stopped or not.
+ void setStop(boolean isStop)	Sets the timer's stop state.
+ void reset()	Resets the timer to its default starting value (1 minute, 30 seconds, 0 milliseconds).

2. Package logic.components

2.1 public class Bomb

Bomb class that represents a bomb object in the game. It manages the bomb's image, position, and collision detection with a Croissant object.

Fields

Name	Description
- final double WIDTH	Defines the width of an object (likely the bomb) as 40 units.

- final double HEIGHT	Defines the height of an object as 40 units.
- final int HEIGHT_OUT_OVEN_AREA	Represents a height value outside the oven area, set to 60 units. (The exact purpose depends on the context of the code.)
- final int GRID_SIZE	Defines the size of the grid as 440 units. This likely represents the playable area.
- int x	Stores the x coordinate of the bomb on the grid.
- int y	Stores the y coordinate of the bomb on the grid.
- Image bombImage	Stores an image representing the bomb.
- ImageView imageView	Represents an ImageView that displays the bomb image in the UI.

Constructor

Name	Description
+ Bomb()	- Loads the bomb image (tomatoExplosion.gif) from the "images" folder. - Creates an ImageView to display the image. - Sets the image dimensions to match the defined WIDTH and HEIGHT (40x40). - Calls spawnRandom() to place the bomb randomly on the grid.

Method

Name	Description
+ void spawnRandom()	Ensures that bombs spawn randomly, stay within the game boundaries, and update their position correctly for gameplay mechanics.
+ void render(GraphicsContext gc)	- Displays the bomb image on the screen at the correct position.

	- The bomb is drawn with the specified width and height, ensuring it appears at the right size and location based on its coordinates (x, y).
+ boolean checkCollision(Croissant croissant)	- Detects collisions by comparing the positions of the bomb and the croissant. - If the absolute distance between their positions is less than the bomb's size, it returns true (indicating a collision). - If not, it returns false, meaning no collision.
- void updateImageView()	Updates the bomb’s image position on the screen, ensuring that the ImageView reflects the bomb’s current location on the grid.
+ Image getImage()	Returns the image (bombImage) of the bomb, making it accessible for other uses, such as displaying the image elsewhere or checking it for game logic.
+ int getX()	Returns the current x coordinate of the bomb.
+ int getY()	Returns the current y coordinate of the bomb.

2.2 public class ClassicCroissant extends Croissant

ClassicCroissant class extends Croissant to represent a specific type of croissant. It provides the properties and behavior of the Classic Croissant, including the image update when it is baked or burnt.

Fields

Name	Description
- final String RAW_IMAGE_PATH	Path to the raw (unbaked) croissant image: This is the file path for the image of a raw

	croissant (rawClassicCroissant.png), likely displayed before it is baked.
- final String BURNED_IMAGE_PATH	Path to the burned croissant image: This is the file path for the image of a burned croissant (burnedCroissant.png), likely displayed when the croissant is over-baked or ruined.
- final String CLASSIC_IMAGE_PATH	Path to the classic (perfectly baked) croissant image: This is the file path for the image of a classic croissant (classicCroissant.png), probably the final, desired version of the croissant.
- final int CLASSIC_CROISSANT_BAKE_TIME	Bake time for the classic croissant: The bake time is 5 seconds. This is how long the croissant needs to bake before it reaches its final, perfect state.
- final int CLASSIC_CROISSANT_BASE_SCORE	Base score for the croissant: The base score for a classic croissant is 100 points. This could represent the points awarded to the player when they bake a croissant successfully.

Constructor

Name	Description
+ ClassicCroissant()	- This constructor is used to initialize a ClassicCroissant object by providing specific details for that type of croissant, including its name, bake time, base score, and image path. - By calling super(...), it ensures that the parent class constructor is properly called to set up the common properties (likely for all

	croissant types) before setting the specific ones for the classic croissant.

Method

Name	Description
+ void updateImage(ImageView imageView, boolean isBurnt)	- Updates the croissant's image depending on whether it is burned or not. If the croissant is burned, it fetches the burned image path, and if it is not, it gets the normal or classic image path. - Then updates the croissant's visual representation on the screen with the correct image.
- String getImagePath(boolean isBurnt)	This method determines the appropriate image path based on the burned state of the croissant. • It returns the path to the burned image if the croissant is burned. • It returns the path to the classic image if the croissant is not burned.

2.3 public class ControlPane extends VBox

ControlPane class for managing the control buttons (Easy/Hard mode) and displaying the game name.

Fields

Name	Description
- Button selectEasyMode	A button that allows the player to select the easy mode in the game.
- Button selectHardMode	A button that allows the player to select the hard mode in the game.

- Text gameName	A text element that displays the game name on the screen.
+ boolean isHard	A static boolean variable that tracks whether the game is in hard mode (true) or easy mode (false). This is likely set when one of the mode buttons is clicked.
- final int BUTTON_WIDTH	The width of the buttons (used for both easy mode and hard mode buttons).
- final int BUTTON_HEIGHT	The height of the buttons for selecting game modes.
- final int FONT_SIZE_GAME_NAME	The font size for displaying the game name text.
- final int FONT_SIZE_BUTTON	The font size for the text on the mode selection buttons.
- final String FONT_FAMILY	The font family used for displaying text in the UI. In this case, "Comic Sans MS".
- final int PREF_WIDTH	The preferred width of the control pane (likely the section containing the buttons).
- final int PREF_HEIGHT	The preferred height of the control pane.
- final int SPACING	The spacing (vertical or horizontal) between UI elements, such as between buttons or text.
- final int PADDING	The padding around UI elements, like the text or buttons, for better alignment and spacing.

Constructor

Name	Description
+ ControlPane()	- Sets the preferred width of the control pane (from the constant PREF_WIDTH, which is 300).

	- Sets the preferred height of the control pane (from the constant PREF_HEIGHT, which is 600). - Aligns the child components (buttons and text) to the top center of the control pane. - Sets the spacing between the components to 30 pixels (from the constant SPACING). - Adds padding of 10 pixels (from the constant PADDING) to the top and bottom of the control pane. - Initializes the button for Easy Mode. - Initializes the button for Hard Mode. - Initializes the game name text. - Adds the game name text (gameName) and both the Easy mode button (selectEasyMode) and Hard mode button (selectHardMode) to the layout. These components will be displayed in the control pane.

Method

Name	Description
+ boolean isHard()	A getter method for the isHard variable. It returns the current value of the game mode (whether the game is in hard mode or not).
+ *void setHard(boolean isHard)*	A setter method for the isHard variable. It sets the value of the isHard static variable, which determines the current game mode.
- void initializeGameName()	Creates and styles the game name text, displaying "CROISSANT 💣 !BOOM! 💣 " with specific font and colors.

- void initializeHardButton()	Creates and configures the Hard Mode button with specific styles and actions (hover effects, click action to set hard mode).
- void initializeEasyButton()	Creates and configures the Easy Mode button with specific styles and actions (hover effects, click action to set easy mode).

2.4 public class Croissant

The Croissant class represents a base croissant object in the game. It contains properties like name, bake time, base score, and the croissant's image. It also handles movement, random spawning, and rendering of the croissant.

Fields

Name	Description
- String name	Name of the croissant.
- int bakeTime	Bake time in seconds.
- int baseScore	Base score for this croissant.
- final double WIDTH	Defines the width of the croissant as 48 units.
- final double HEIGHT	Defines the height of the croissant as 40 units.
- int x	Position of the croissant on the grid.
- int y	Position of the croissant on the grid.
- int speed	Stores the movement speed of the croissant as 20 units.
- final int GRID_SIZE	Defines the size of the game grid where the croissant can spawn as 392 units.
- boolean inOvenArea	Indicates whether the croissant is within the oven area as false.
- Image croissantImage	Image of the croissant.
- ImageView imageView	ImageView for displaying the croissant.

- State state	Current state of the croissant (e.g., RAW, BURNED).
- AudioClip spawnSound	Sound played when croissant spawns.

Constructor

Name	Description
+ Croissant(String name, int bakeTime, int baseScore, String imagePath)	- Sets the croissant's name using the name parameter, which can be useful for identification or display purposes. - Sets the base score of the croissant using the baseScore parameter. This score is typically awarded to the player when the croissant is successfully baked or interacted with. - Sets the bake time for the croissant, determining how long it needs to bake in the oven. The bakeTime parameter specifies the duration in seconds. - Loads the image of the croissant from the specified imagePath. The ClassLoader.getSystemResource() method retrieves the image file from the classpath (e.g., "images/classicCroissant.png"), and toString() converts it to a URI string for the Image constructor. - Creates an ImageView object to display the loaded croissantImage on the screen. This is used to render the image in the graphical user interface (GUI). - Sets the width of the imageView to the predefined constant WIDTH. This ensures the

	croissant image is scaled to the correct width in the game. - Sets the height of the imageView to the predefined constant HEIGHT, ensuring the image is scaled to the correct height. - Sets the initial state of the croissant to RAW. This state indicates that the croissant is in its raw, uncooked form at the beginning of the game. - Creates an AudioClip object to represent the sound effect played when the croissant spawns. The sound file (e.g., "spawnSound.mp3") is loaded from the classpath. - Calls the spawnRandom() method to position the croissant at a random location within the game grid, determining where it will appear in the game world.

Method

Name	Description
+ Image getCroissantImage()	Get the image for the croissant by specifying the image file path.
+ void setCroissantImage(String imagePath)	Set the image for the croissant by specifying the image file path.
+ String getName()	Return the name of the croissant respectively.
+ int getBakeTime()	Return the bake time of the croissant respectively.
+ int getBaseScore()	Return the base score of the croissant respectively.

+ void spawnRandom()	Spawns the croissant at a random position on the grid and plays the spawn sound.
+ void move(int dx, int dy)	Moves the croissant by a specified delta in the x and y direction, ensuring it stays within the grid bounds.
+ boolean checkInOvenArea()	Checks if the croissant is within the oven area (i.e., its y-position is within a certain range).
+ boolean isPerfectlyBaked(int actualTime)	Checks if the croissant is perfectly baked by comparing the actual bake time to the ideal bake time, allowing a margin of 1 second.
+ void render(GraphicsContext gc)	Renders the croissant on the screen using the provided GraphicsContext.
+ void updateImageView()	Updates the ImageView with the current position of the croissant.
+ int getX()	Returns the current x position of the croissant.
+ void setX(int x)	Sets the x position of the croissant to a new value. It updates the internal x position and also ensures the ImageView is updated with the new position.
+ int getY()	Returns the current y position of the croissant.
+ void setY(int y)	Sets the y position of the croissant to a new value. It updates the internal y position and ensures the ImageView is updated to reflect the new position.
+ int getSpeed()	Retrieves the current speed of the croissant. The speed determines how fast the croissant moves in the game when it is displaced along

	the x and y axis. The speed is used as a multiplier when the croissant is moved.
+ void setSpeed(int speed)	Set or change the speed of the croissant. You pass an integer value speed, which will then update the speed property of the croissant.
+ boolean isInOvenArea()	Checks if the croissant is currently in the "oven area". The oven area is determined based on the croissant's position (typically on the y axis). For instance, the croissant might be in the oven area if it's within the first few units of the y grid.
+ void setInOvenArea(boolean inOvenArea)	Sets whether the croissant is currently in the oven area. It uses a boolean value to mark the status.
+ int getGridSize()	Returns the size of the grid, which determines the bounds within which the croissant can move.
+ void setName(String name)	Set the name of the croissant. This could be used for identification or display purposes in the game.
+ void setBakeTime(int bakeTime)	Sets the bake time of the croissant. The bake time indicates how long the croissant should bake in the oven. It ensures that the bake time is at least 1 second, to prevent a non-positive bake time.
+ void setBaseScore(int baseScore)	Sets the base score of the croissant. The base score is the value awarded to the player when the croissant is successfully baked.
+ void setPosition(int x, int y)	Sets both the x and y positions of the croissant simultaneously. This allows you to

	move the croissant to a new position on the grid and update its graphical representation accordingly.
+ ImageView getImageView()	Getting the ImageView used to display the croissant.
+ void setImageView(ImageView imageView)	Setting the ImageView used to display the croissant.
+ State getState()	Returns the current state of the croissant.
+ void setState(State state)	Sets the current state of the croissant (e.g., RAW, BURNED), and updates the image if needed.
+ ***void updateImage(ImageView imageView, boolean isBurnt)***	This is an abstract method meant to be implemented by subclasses to update the image of the croissant based on its state (e.g., whether it's burnt).

2.5 public class CroissantFactory

CroissantFactory class that creates random Croissant objects. It provides a method to randomly generate a Classic, Salad, or Rainbow Croissant.

Fields

Name	Description
- final int CLASSIC_CROISSANT_TYPE	This constant represents the classic croissant type and is assigned the integer value 0.
- final int SALAD_CROISSANT_TYPE	This constant represents the salad croissant type and is assigned the integer value 1.
- *final int RAINBOW_CROISSANT_TYPE*	This constant represents the rainbow croissant type and is assigned the integer value 2.

Method

Name	Description
+ Croissant createRandomCroissant()	Creates a random croissant.
- Croissant createCroissantByType(int type)	Creates a croissant of a specific type based on the integer passed as the type argument.

2.6 public class EndCanvas extends VBox

EndCanvas class represents the layout shown at the end of the game. It displays the final score and provides options to restart or go home.

Fields

Name	Description
- final int PADDING	This line defines a constant PADDING with a value of 150.
- final int BUTTON_TRANSLATE_X	This line defines a constant BUTTON_TRANSLATE_X with a value of 130.

Constructor

Name	Description
+ EndCanvas()	- Calls super() to invoke the parent constructor, which is typically needed in subclasses to initialize the inherited properties. - Creates two UI components: restartBox and homeBox using the helper methods createRestartBox() and createHomeBox(). - Sets the background of the canvas using createBackgroundImage(). - Adds the restartBox and homeBox to the canvas layout using getChildren().add(). This ensures both elements appear in the UI.

Method

Name	Description
- Background createBackgroundImage()	Creates and returns a Background object with an image.
- HBox createRestartBox()	Creates an HBox containing the restart button and score display.
- HBox createHomeBox()	Creates an HBox containing the home button.

2.7 public class ExitButton extends VBox

Represents the Exit button.

Fields

Name	Description
- final double BUTTON_WIDTH	The width of the exit button (40 pixels).
- final double BUTTON_HEIGHT	The height of the exit button (40 pixels).
- final double BUTTON_FONT_SIZE	The font size for the button's text (25 pixels).
- final Color BUTTON_COLOR	The default color of the exit button (a light brown).
- final Color HOVER_COLOR	The color the button changes to when the mouse hovers over it (a reddish color).
- final double BUTTON_CORNER_RADIUS	The radius of the button's corners (10 pixels), making the button have rounded corners.
- Button selectExitGame	This is the main button representing the exit action in the UI. It will be displayed with the label "X" on it.

Constructor

Name	Description
+ ExitButton()	Initializes the ExitButton by calling initializeExitButton(), which configures the appearance and behavior of the exit button.

	The button is then added to the container (VBox).

Method

Name	Description
- void initializeExitButton()	Configures the selectExitGame button's appearance and behavior: • Font and Size: The text on the button is set to "X", using the Comic Sans MS font, with bold weight and a font size of 25. • Button Size: The button's width and height are set to 40px. • Background Color: The button's background color is set to a light brown (BUTTON_COLOR), and its corners are rounded with the specified BUTTON_CORNER_RADIUS. • Hover Effects: When the mouse enters the button, the background color changes to a reddish tone (HOVER_COLOR), and the text remains "X". When the mouse exits, the button reverts to its original color (BUTTON_COLOR). • Click Action: When the button is clicked, the endTimer() method is invoked to stop the timer and end the game.
- void endTimer()	Ensures that the game stops running when the exit button is clicked.

2.8 public class GameCanvas extends StackPane

The GameCanvas class represents the game screen, including the game grid, score, timer, recipe, and various game components like the oven and trash.

Fields

Name	Description
- Text scoreText	A Text object that will display the current score of the game.
- WalkGround walkGround	A static reference to a WalkGround object, likely representing the area where some gameplay occurs (such as a ground or surface in the game).
- int score	The score variable keeps track of the player's score. It is initialized to 0.
- KeyboardController keyboardController	A reference to a KeyboardController object, which likely handles player input via the keyboard.
- Text gameOverText	A Text object to display a "Game Over" message when the game ends.
- TimerPane timerPane	A static reference to a TimerPane object, likely representing a timer displayed during gameplay.
- ImageView recipeImg	An ImageView object that will display an image related to the recipe in the game.
- Pane recipe	A static reference to a Pane object that will contain the recipe UI (possibly displaying ingredients or instructions).
- ScorePane scorePane	A static reference to a ScorePane object, which might be a UI container that shows the player's score.

- OvenArea ovenArea	A static reference to an OvenArea object, likely representing the area where food (e.g., croissants) are baked in the game.
- Trash bin	A static reference to a Trash object, likely used to represent a trash can where players can discard items.
- Table serveTable	A static reference to a Table object, likely representing a surface where players can place items to serve them.
- ExitButton exitButton	A static reference to an ExitButton object, which allows players to exit the game.
- GridPane game	A GridPane layout container that arranges the game components in a grid. It's initialized here but is likely populated with the various UI components.
- final int GRID_COLUMNS	Defines the number of columns in the grid layout, which is 20. This likely divides the screen into 20 columns.
- final int GRID_ROWS	Defines the number of rows in the grid layout, which is 16. This divides the screen into 16 rows.
- final double COLUMN_WIDTH	This is the width of each column in the grid, calculated as 100% divided by 20 columns (i.e., each column takes up 4.166% of the total width).
- final double ROW_HEIGHT	This is the height of each row in the grid, calculated as 100% divided by 16 rows (i.e., each row takes up 5.88% of the total height).
- final int CANVAS_PADDING	Defines padding for the canvas or area that will contain the game content.

- final int RECIPE_WIDTH	Defines the width of the recipe section in the UI (possibly for displaying the recipe image or details).
- final int RECIPE_HEIGHT	Defines the height of the recipe section in the UI.
- final int IMAGE_WIDTH	Defines the width of an image displayed in the game (possibly for the background or certain game elements).
- final int IMAGE_HEIGHT	Defines the height of an image displayed in the game.

Constructor

Name	Description
+ GameCanvas()	- Configuring the grid properties. - Initializing game components. - Adding a background image to the game. - Creating a placeholder for Game Over text. - Setting padding for the canvas. - Adding the background image and grid layout to the scene. - Connecting game logic to the timer and score system.

Method

Name	Description
- void initializeComponents()	Initializes the components of the game canvas.Sets up various game elements like recipe, score pane, oven, etc.
- Pane createPane(String color, int width, int height)	Creates a Pane with the specified color and size.
- void setGridProperties()	Sets up the grid properties (columns and rows) for the game grid.

+ void setGameOverText()	Sets the "GAME OVER" text.
+ OvenArea getOvenArea()	Getter for OvenArea, initializes it if not created.
+ Table getTable()	Getter for Table (ServeTable), initializes it if not created.
+ Trash getBin()	Getter for Trash (Bin), initializes it if not created.
+ int getScore()	Getter for score.
+ void setScore(int score)	Setter for score.
+ ScorePane getScorePane()	Getter for ScorePane.
+ TimerPane getTimerPane()	Getter for TimerPane.
+ void setTimerPane(TimerPane timerPane)	Setter for TimerPane.
+ void setScorePane(ScorePane scorePane)	Setter for ScorePane.

2.9 public class Home extends VBox

Represents the Home screen component with a restart button.

Fields

Name	Description
- final double BUTTON_WIDTH	Button width size: 50 pixels
- final double BUTTON_HEIGHT	Button height size: 50 pixels
- final double BUTTON_FONT_SIZE	Font size: 30px
- final double PREF_WIDTH	Home screen's width preferred size: 400 pixels
- final double PREF_HEIGHT	Home screen's height preferred size: 400 pixels
- final double SPACING	Space between elements: 30px
- final Insets PADDING	Adds 10px padding on top and bottom
- final Color BUTTON_COLOR	Default button color: Grayish (rgb(87, 87, 82))
- final Color HOVER_COLOR	Hover color: Warm orange (rgb(222, 139, 100))

- final double BUTTON_CORNER_RADIUS	Button's rounded corners: 10px
- Button selectHome	It represents a button that allows the player to return to the home screen.

Constructor

Name	Description
+ Home()	- Set preferred size. - Center elements at the top. - Set spacing between elements. - Set padding. - Create and configure the button. - Add button to layout.

Method

Name	Description
- void initializeHomeButton()	- Creates a Button with a house emoji (🏠). - Uses "Comic Sans MS" with bold styling for a fun look. - Applies a gray background and white text. - Adds hover effects: • Changes color to orange. • Changes text from 🏠 ⟶ <3. - Handles click events by switching scenes.

2.10 public class MenuCanvas extends AnchorPane

The MenuCanvas class sets up the menu screen for the game. It initializes the background image and places the ControlPane at the bottom of the screen.

Fields

Name	Description
- final String BACKGROUND_IMAGE_PATH	- A constant for the file path of the background image.

	- "images/background.gif" represents the location of the image within the project. - Used to load the background image for the UI.
- final double BACKGROUND_WIDTH	- Defines the width of the background image as 1000 pixels. - Ensures the background image fits properly on the screen.
- final double BACKGROUND_HEIGHT	- Defines the height of the background image as 720 pixels. - Helps maintain the correct aspect ratio for the background.
- final double CONTROL_PANE_TOP_ANCHOR	- Specifies the vertical position of the control panel (buttons, menus, etc.). - 150 pixels from the top of the screen. - Ensures proper alignment of UI components.

Constructor

Name	Description
+ MenuCanvas()	- Calls the constructor of the parent class (which could be a class like Pane or Canvas), initializing the layout and properties that are inherited from the parent. - Sets the background for the canvas. It's likely that this method loads an image or a color to be used as the background for the menu screen. The initializeBackground() method would typically define a background image or color for the menu.

	- Adds the control pane to the layout. The control pane likely contains various UI elements like buttons, labels, or other interactive components for the user to interact with. It might include navigation buttons or game options. - Adds a tutorial button to the screen, which is typically used to guide the user on how to play or use the application. When clicked, it may show an instructional screen or dialog to help users understand the game or app.

Method

Name	Description
- void initializeBackground()	Initializes and sets the background image for the menu screen.
- void initializeControlPane()	Initializes the ControlPane and sets its position at the bottom of the screen.
- void initializeTutorialButton()	Initializes a TutorialButton and places it at the bottom-right corner of a layout (likely an AnchorPane). It sets the button's size to 100x100 pixels and positions it 20px from the bottom and 20px from the right. Then, it adds the button to the layout.

2.11 public class OvenArea extends Pane implements ResetGameAble

Represents an oven area where croissants can be baked. It handles baking, removing croissants, and updating their states.

Fields

Name	Description
- Croissant croissantInOven	Holds the current croissant that is in the oven. It is initialized to null because there might not be a croissant in the oven at the start.
# boolean isBaking	Tracks whether the croissant is currently baking in the oven. It's initialized to false, meaning the croissant is not baking when the program starts.
# boolean isCooked	Tracks whether the croissant has finished baking and is cooked. It starts as false, meaning the croissant isn't cooked initially.
- Thread bakingThread	Holds a Thread object that manages the baking process. By using a separate thread, the baking process can run concurrently without blocking the main program, allowing for background tasks like the baking timer or updates to the UI.
- AudioClip takeOutSound	Holds an audio clip that plays when the croissant is taken out of the oven. It could be an audio file representing the sound that is triggered when the baking process is completed or when the croissant is removed.
- final Background defaultBackground	Holds the default background for the oven. It is defined as final, meaning its value cannot be changed after being initialized. The defaultBackground might represent the background image or color of the oven when no croissant is being baked.

- final Background bakingBackground	Holds the background for the oven when a croissant is in the process of being baked. Similar to defaultBackground, it is defined as final and cannot be modified after it is initialized. The bakingBackground could represent an animated or special background when the baking process is underway.

Constructor

Name	Description
+ OvenArea(int width, int height)	- Sets the preferred size of the OvenArea to the given width and height. This controls how large the oven area will be displayed in the application. - Loads the default image (oven.png) for the oven. This image is used when the oven is not in use or when no croissant is baking. The image is loaded from the resources folder (via the class loader) and converted to a String URL for use in the program. - Loads the image (ovenCooking.png) for when the oven is actively baking a croissant. This background image represents the oven during the cooking process. - This background is set when the oven is in the process of baking. It uses the bakingImage loaded previously and specifies that the image should not repeat and should be centered. - This background is set when the oven is idle (not baking). It uses the defaultImage loaded

	earlier and also specifies that the image should not repeat and should be centered. - Sets the initial background of the oven area to the default background (defaultBackground). This will be the background displayed until the oven starts baking. - Loads a sound file (bellDing.mp3) which is played when the croissant is taken out of the oven. The sound is loaded from the resources folder and represented as an AudioClip.

Method

Name	Description
+ boolean isBaking()	Returns whether the oven is currently baking.
+ void setBaking(boolean isBaking)	Sets the baking status of the oven and updates the background.
+ boolean isCooked()	Returns whether the croissant in the oven is cooked.
+ void setCooked(boolean isCooked)	Sets the cooked status of the croissant.
+ void addCroissant(Croissant c)	Adds a croissant to the oven and starts the baking process.
- void startBaking(Croissant c)	Starts the baking process for a croissant. The baking time is based on the croissant's bake time.
+ void removeCroissant()	Removes the croissant from the oven and plays the sound.
+ boolean isOccupied()	Checks if the oven is currently occupied by a croissant.
+ Croissant getCroissantInOven()	Returns the croissant currently in the oven.

+ void setCroissantInOven(Croissant croissantInOven)	Sets the croissant to be in the oven.
+ void reset()	Resets the oven to its default state. This includes clearing the croissant, stopping the baking process, and resetting the background.

2.12 public class RainbowCroissant extends Croissant

RainbowCroissant class extends Croissant to represent a specific type of croissant. It provides the properties and behavior of the Rainbow Croissant, including the image update when it is baked or burnt.

Fields

Name	Description
- final String RAW_IMAGE_PATH	Stores the file path for the image of a raw (unbaked) rainbow croissant. It points to the rawRainbowCroissant.png image located in the images folder.
- final String BURNED_IMAGE_PATH	Stores the file path for the image of a burned croissant. If a croissant is left too long in the oven, it becomes burned, and this image is used to represent that state.
- final String RAINBOW_IMAGE_PATH	Stores the file path for the image of a fully baked rainbow croissant. After the baking process, this image is used to represent the perfectly baked croissant.
- final int BAKE_TIME	Defines how long (in seconds) it takes for a croissant to bake properly. In this case, the croissant takes 8 seconds to bake.
- final int BASE_SCORE	Defines the base score awarded for successfully baking a croissant. If the croissant is baked properly and is not burned, the

	player will earn this score. In this case, the base score is set to 250 points.

Constructor

Name	Description
+ RainbowCroissant()	Sets the name, bake time, base score, and image for the Rainbow Croissant.

Method

Name	Description
+ void updateImage(ImageView imageView, boolean isBurnt)	Updates the image of the croissant based on whether it is burnt or not.
- String getImagePath(boolean isBurnt)	Returns the correct image path based on whether the croissant is burnt.

2.13 public class SaladCroissant extends Croissant

SaladCroissant class extends Croissant to represent a specific type of croissant. It provides the properties and behavior of the Salad Croissant, including the image update when it is baked or burnt.

Fields

Name	Description
- final String RAW_IMAGE_PATH	Stores the file path for the image of a raw (unbaked) rainbow croissant. It points to the rawRainbowCroissant.png image located in the images folder.
- final String BURNED_IMAGE_PATH	Stores the file path for the image of a burned croissant. If a croissant is left too long in the oven, it becomes burned, and this image is used to represent that state.
- final String SALAD_IMAGE_PATH	Stores the file path for the image of a fully baked salad croissant. After the baking

	process, this image is used to represent the perfectly baked croissant.
- final int BAKE_TIME	Defines how long (in seconds) it takes for a croissant to bake properly. In this case, the croissant takes 6 seconds to bake.
- final int BASE_SCORE	Defines the base score awarded for successfully baking a croissant. If the croissant is baked properly and is not burned, the player will earn this score. In this case, the base score is set to 120 points.

Constructor

Name	Description
+ SaladCroissant()	Sets the name, bake time, base score, and image for the Salad Croissant.

Method

Name	Description
+ void updateImage(ImageView imageView, boolean isBurnt)	Updates the image of the croissant based on whether it is burnt or not.
- String getImagePath(boolean isBurnt)	Returns the correct image path based on whether the croissant is burnt.

2.14 public class Score implements ResetGameAble

Class representing a score in the game. Implements the ResetGameAble interface to allow resetting the score.

Fields

Name	Description
- int score	Instance variable to hold the score value.

Constructor

Name	Description
+ Score()	Default constructor that initializes the score to 0.
+ Score(int score)	Initializes the score with a specified value.

Method

Name	Description
+ int getScore()	Gets the current score.
+ void setScore(int score)	Sets the score value. If the provided score is negative, it will be set to 0.
+ String toString()	Returns the score as a string representation.
+ void reset()	Resets the score to 0.

2.15 public class ScorePane extends Pane

A class representing the score pane, which displays the score text and score count.

Fields

Name	Description
- Text scoreText	Text for "SCORE" label.
- Text scoreCount	Text for the score count.

Constructor

Name	Description
+ ScorePane()	- Initialize score text and score count. - Set font styles. - Set position for the score text. - Set position for the score count. - Set text color. - Update the score display by calling GameLogic. - Add the score texts to the pane.

Method

Name	Description
+ void setScoreText(Score score)	Updates the score count display with the new score.

2.16 public class Scoreandrestart extends VBox

A class that displays the score and provides a restart button. Implements a singleton design pattern for a single instance.

Fields

Name	Description
- Scoreandrestart instance	Singleton instance.
- Button selectRestart	Button to restart the game.
- Text score	Text to display the score.

Constructor

Name	Description
+ Scoreandrestart()	Initialize the layout, score, and restart button.

Method

Name	Description
- void initializeScoreDisplay()	Initializes the score display with the current score.
- void initializeRestartButton()	Initializes the restart button with hover effects and click action.
- void setScore(int score)	Updates the score displayed on the screen.

2.17 public enum State

Enum representing the possible states of an item (e.g., croissant) in the oven.

Name	Description
RAW	The item is raw and not yet baked.
BURN	The item is burned.

PERFECT	The item is perfectly baked.

2.18 public class Table extends Pane

Class representing a table with a default and a served state. The table background can change when it is served.

Fields

Name	Description
- Background defaultBackground	Holds a Background object that is used to represent the default background for an element in the game (such as a pane or a scene).
- Image defaultImage	Holds an Image object, which represents an image that can be used for the default state of an element (like an oven, table, or other in-game objects).
- Background serveBackground	Holds a Background object for the "serve" state, which is likely when the croissant (or another item) is ready to be served.
- Image serveImage	Holds an Image object representing the image for the "serve" state, likely showing an item (such as the croissant) in its final, served form.

Constructor

Name	Description
+ Table(int width, int height)	- Load images for different table states. - Create background for default table state - Create background for served table state. - Set the default background initially.

Method

Name	Description
+ void serve()	Changes the table background to the served state and resets after a short delay.

2.19 Class TimerPane extends Pane

Field

Name	Description
- Text header	Displays a label ("Timer") above the countdown.
- Text timer	Displays the current countdown time.

Constructor

Name	Description
+ TimerPane()	Sets the size of the pane to 200x80 pixels. Displays the label " Timer". Displays the countdown (initial value: "00:00:00").Sets the font to "Comic Sans MS" with sizes 25 and 40 respectively Sets the text color to black. setX and setY Controls the position of the text within the pane. Calls GameLogic.startCountDownTimer() to begin the countdown. Adds both header and timer to the pane for display.

Methods

Name	Description
+ void setTimer(Timer t)	Updates the displayed countdown using the Timer object. Assumes the Timer class provides a formatted String (e.g., "00:00:00").

2.20 Class Trash extends Pane

Field

Name	Description
- Background defaultBackground	Stores the background image of the trash bin.

Constructor

Name	Description
+ Trash(int width, int height)	Initialize the Trash pane with the given width and height. Load the image for the trash bin. Create a background using the trash bin image. Set the trash bin image as the background

2.21 Class Tutorial extends AchorPane

Field

Name	Description
- final String TUTORIAL_VIDEO_PATH	Path to the tutorial video file.
- MediaView mediaView	Displays the video content.
- MediaPlayer mediaPlayer	Controls video playback (play, pause, stop).
- Media media	Represents the media source (tutorial video).
- ProgressIndicator loadingIndicator	Shows a spinning circle while the video loads.

Constructor

Name	Description
+ Tutorial	Initializes the Tutorial pane by Displaying a loading spinner, Loading the tutorial video, Adding a Home button.

Methods

Name	Description
- void initializeLoadingIndicator()	Creates a loading spinner to show while the video is loading. Positions the spinner in the center.
- void loadMedia()	Loads the tutorial video and displays it when ready. Platform.runLater is used to handle JavaFX UI updates safely on the main thread. If the video fails to load, it retries by calling loadMedia() again.
- void initializeHomeButton()	Creates and positions a "Home" button in the bottom-right corner. The button always stays above the video.
+ MediaPlayer getMediaPlayer()	Returns the MediaPlayer object (used to control video playback). If mediaPlayer is null, Shows the loading spinner. (Commented out) Reloads the media.

2.22 Class TutorialButton extends VBox

Field

Name	Description
- final double BUTTON_WIDTH	Defines the width of the tutorial button.
- final BUTTON_HEIGHT	Defines the height of the tutorial button.
- final BUTTON_FONT_SIZE	Font size for the button label.
- final double PREF_WIDTH	Preferred width of the TutorialButton component.
- final double PREF_HEIGHT	Preferred height of the TutorialButton component.

- final double SPACING	Vertical spacing between child elements inside the VBox.
- final Insets PADDING	Padding for the VBox (top and bottom padding only).
- final Color BUTTON_COLOR	Default background color of the button.
- final Color HOVER_COLOR	Background color when the mouse hovers over the button.
- final double BUTTON_CORNER_RADIUS	Corner radius for rounded button edges.
- Button selectRestart	Represents the tutorial button (?). Clicking it navigates to the tutorial scene.

Constuctor

Name	Description
+ TutorialButton()	Sets the size, alignment, spacing, and padding for the VBox. Initializes and styles the tutorial button (initializeTutorialButton()). Adds the tutorial button to the VBox.

Methods

Name	Description
- void initializeTutorialButton()	Creates and styles the button. Hover Effect: Changes color and text to ¿. Click Event: Switches to the tutorial scene via SceneController.showTutorialScene().

2.23 Class WalkGround extends Canvas

Manages the croissant's movement and interactions with the oven, trash, bombs, and serving area. Handles game logic for serving croissants, bomb collisions, and trashing items.

Field

Name	Description
- final Image walkingArea	Background image of the walking area where the croissant moves.
- Croissant croissant	The current croissant controlled by the player.
- OvenArea oven	Reference to the oven where the croissant can be baked.
- ArrayList<Bomb> bombs	List of bombs spawned in the game.
- Trash bin	Reference to the trash bin for discarding croissants.
- Table serveTable	Area where the croissant is served.
- int bombNum	Maximum number of bombs allowed in the game (2 in normal, 5 in hard).
- int bombSpawnRate	Rate at which bombs spawn (4000 ms for normal, 2000 ms for hard).
- AudioClip bombSound	Sound played when the croissant hits a bomb.
- KeyboardController keyboardController	Handles keyboard input for player movement.
- final double SIZE	The size of the walking area (440x440 pixels).
- final double SERVE_AREA_X_MIN	Constants defining the serve area boundaries (x min).
- final double SERVE_AREA_X_MAX	Constants defining the serve area boundaries (x max).
- final double SERVE_AREA_Y_MIN	Constants defining the serve area boundaries (y min).

- final double TRASH_AREA_X_MIN	Constants defining the trash area boundaries (x min).
- final double TRASH_AREA_Y_MIN	Constants defining the trash area boundaries (y min).
- final double TRASH_AREA_Y_MAX	Constants defining the trash area boundaries (y max).

Constructor

Name	Description
+ WalkGround(KeyboardController keyboardController)	Initializes the walking area with game components. It sets up the croissant, oven, bomb list, trash bin, and serve table. It also handles bomb spawning and movement setup.

Methods

Name	Description
+ void serveToCustomer()	Serves the croissant when it's in the serving area and the 'R' key is pressed. Updates score based on croissant state (perfect, raw, burned).
+ void goToTrash()	Discards the croissant to the trash area when the 'F' key is pressed and the croissant is within the trash area.
+ void setUpMove()	Handles the movement of the croissant based on keyboard inputs (up (W), down (S), left (A), right(D)). Detects bomb collisions and updates croissant's position accordingly. Q Key: Removes the croissant from the oven if it's in the oven area. E Key: Puts a raw

	croissant into the oven if it's not already occupied.
+ void updateCanvas(GraphicsContext gc)	Refreshes the game canvas, drawing the background, croissant, and bombs. Runs periodically in a separate thread to update the display.
- void spawnBombPeriodically()	Spawns bombs at regular intervals based on the difficulty setting (hard mode spawns bombs faster). Removes bombs when the number exceeds the limit.

3 Package application

3.1 public class Main extends Application

Field

Name	Description
- final String SOUND_FILE	"sounds/backgroundMusic.mp3" is the path for the background music file in the resources/sounds/ folder.
- final String ICON_FILE	"images/classicCroissant.png" is the path for the icon image file in the resources/images/ folder.
- final String WINDOW_TITLE	The title of the game window, set as "CROISSANT BOOM!!! 💣💣💣".
- final boolean RESIZABLE	A boolean value set to false to make the window non-resizable.
- final int MEDIA_LOOP	Set to MediaPlayer.INDEFINITE to make the background music loop indefinitely.

- MediaPlayer mediaPlayer	The media player responsible for playing the background music.

Method

Name	Description
+ void start(Stage primaryStage)	Set up Stage and Scene through the SceneController. Initialize and play background music. Set window icon and title. Disable window resizing. Handle the close request to stop the music and exit the application.
- void closeGame()	Stops the background music when the game window is closed. Exits the JavaFX application and the Java Virtual Machine (JVM), ensuring the program terminates.
+ void main(String[] args)	The main method is the entry point for launching the JavaFX application. It calls launch(args), which starts the JavaFX application lifecycle.