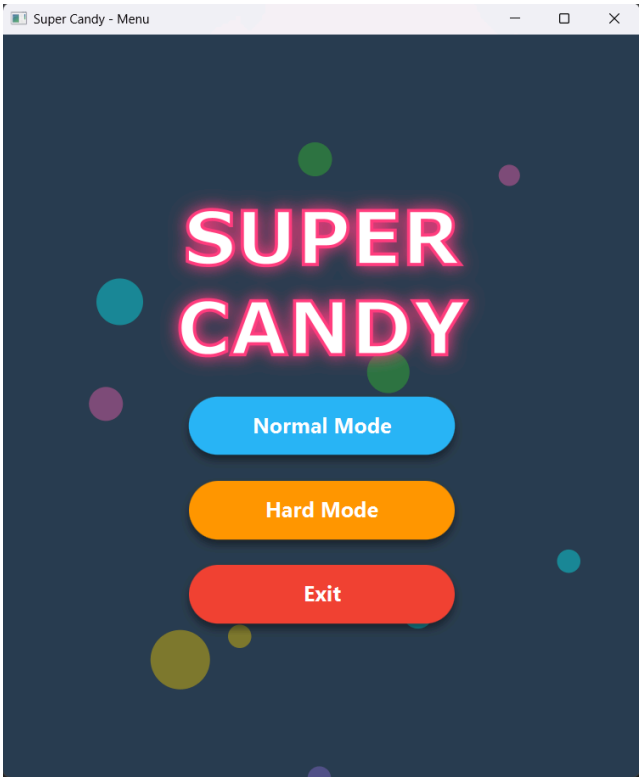
# Super Candy

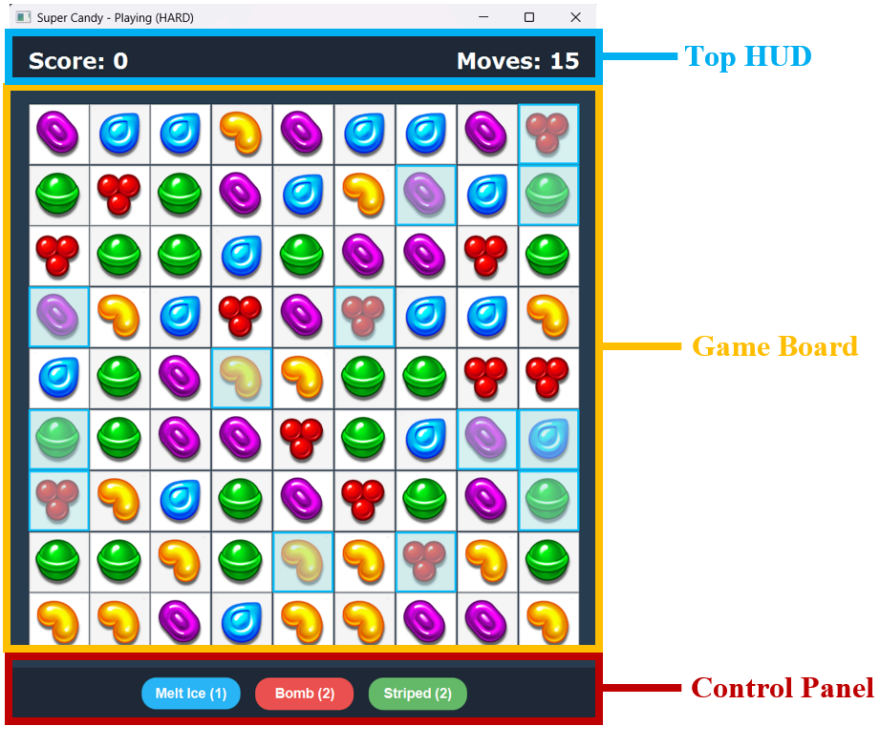
This game is a match-3 puzzle game where the objective is to swap adjacent candies to create lines of three or more matching colors. The game challenges players to achieve the highest possible score within a limited number of moves.




The application offers two distinct difficulty levels accessible from the Main Menu.

**1. Normal Mode** : The standard gameplay experience focused on score accumulation. The board is filled with standard candies, and the player has access to the "Color Bomb" item to clear specific colors.

**2. Hard Mode** : A more challenging variation that introduces "Frozen Candies" (Ice obstacles). Players must match frozen candies to break the ice. This mode provides the "Melt Ice" item to assist in clearing obstacles.




The game screen is divided into three functional areas :

1. **Top HUD**
    - Score : Displays the current accumulated points in the top-left corner.
    - Moves : Displays the remaining number of moves in the top-right corner. The game ends when this counter reaches zero.
2. **Game Board**

A central 9x9 grid containing various colored candies. This is the primary interactive area.

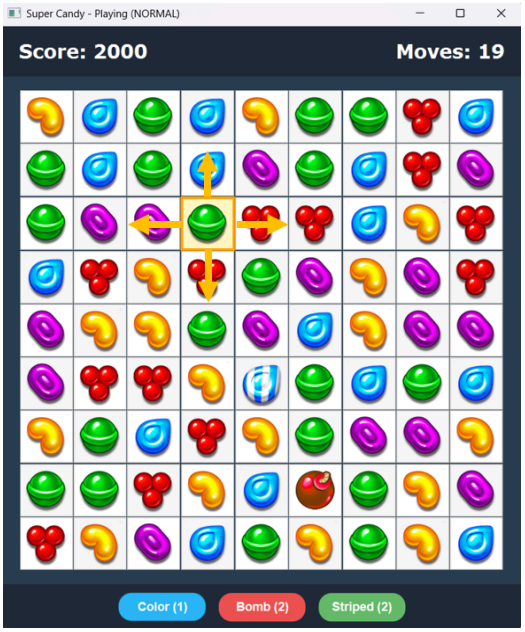
3. **Control Panel**

Located at the bottom of the screen, this panel contains buttons to activate special items. Each button displays the item name and the remaining quantity.

- Color Bomb (Randomly selects a color present on the board and destroys all candies of that color) / Melt Ice (Instantly melts and removes all ice layers from frozen candies on the board.)
- Bomb : Transforms a random candy into a Bomb. It explodes immediately, destroying a 3x3 area of surrounding candies.
- Striped : Transforms a random candy into a Striped Candy. It explodes immediately, clearing an entire row (Horizontal) or column (Vertical).

The game is controlled entirely via the mouse :

1. **Selection :** Click on a candy to select it. The selected tile is highlighted.
2. **Swapping :** Click on an adjacent candy (up, down, left, or right) to attempt a swap.
3. **Matching Logic :** If the swap results in a match of 3 or more identical candies (horizontally or vertically), the candies are eliminated, and points are awarded. If no match is made, the swap is invalid, the tile will not swap




**When candies are eliminated :**

Existing candies fall down to fill the empty spaces. New candies are generated at the top of the board. If the falling candies create new matches, they automatically explode , awarding bonus points and continuing the cycle until no more matches exist.

**Special Case when candies match patterns:**

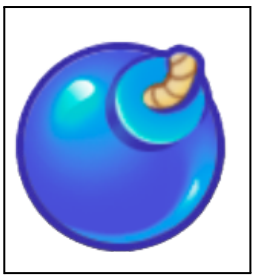
Players can create special candies on the board by matching more than 3 candies in specific patterns.

- **Striped Candy:** Created by matching 4 candies in a line.
  - **Effect:** When matched, it clears an entire row or column depending on the stripe direction.

Horizontal Striped

Vertical Striped

- **Area Bomb or Bomb Candy:** Created by matching 5 candies in an L or T shape.
  - **Effect:** When matched, it explodes twice, destroying all surrounding candies in a 3x3 area.



- **Color Bomb:** Created by matching 5 candies in a straight line.
    - **Effect:** Can be swapped with any adjacent candy to destroy all candies of that color on the board.

**Combo Rules (Special Candy Interactions):**

1. **Striped + Striped**
    - **Effect:**
      Triggers a Cross Blast, clearing both the entire row and the entire column at the point of activation.
2. **Bomb + Bomb**
    - **Effect:**
      Creates an Enhanced Explosion. While a regular Bomb clears a 3×3 area, this combo expands the blast to a 5×5 radius, dealing significantly increased area damage.
3. **Striped + Bomb**
    - **Effect:**
      Triggers a Mega Cross Blast, clearing three rows horizontally and three columns vertically centered on the activation point.
4. **Color Bomb + Striped**
    - **Effect:**
      All candies that share the striped candy's color are transformed into Striped Candies. All transformed candies are then immediately activated, causing a large chain reaction across the board.
5. **Color Bomb + Bomb**
    - **Effect:**
      All candies that share the bomb candy's color are transformed into Bomb Candies. All transformed bombs then explode, resulting in widespread board damage.
6. **Color Bomb + Color Bomb**
    - **Effect:**
      Activating two Color Bombs together clears the entire board, removing all candies regardless of color or type.

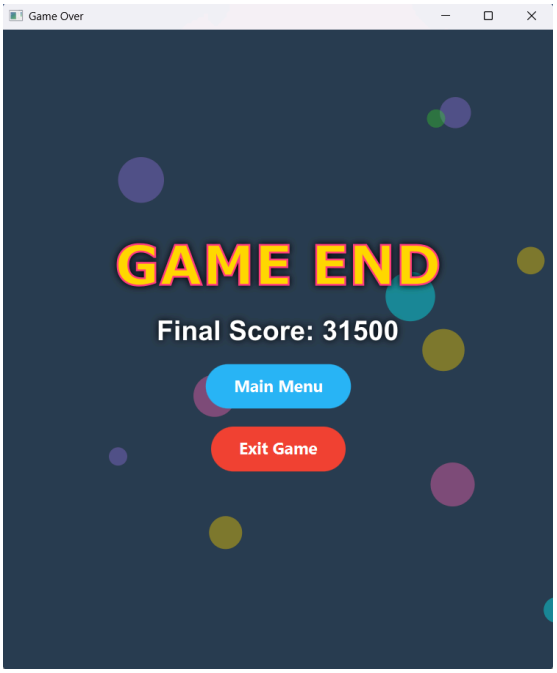
**Scoring System:**

- **Basic Match:** Each candy removed yields **100 points**.
- **Chain Reaction (Cascade):** If falling candies create new matches automatically, these "Chain Reactions" award additional points, encouraging players to plan moves that create cascades.
- **Item Usage:** Activating a special item from the inventory awards **500 points**.

**Board & Auto-Shuffle:**

To ensure the game is always playable:

- **Deadlock Prevention:** The system calculates potential moves after every turn. If no valid moves are possible (Deadlock), the board automatically **shuffles** all candies to create new possibilities without resetting the game state.
- **No-Waste Move:** If a player attempts an invalid swap (one that does not result in a match), the candies swap back to their original positions, and the "Moves" counter **does not decrease**.




The game session ends when the **"Moves"** counter reaches zero. Upon completion, the **"GAME END"** view is displayed, showing the Final Score. Then, Players can choose to return to the Main Menu to restart or Exit the application entirely.

# Implementation Details

---

**Overall of javadoc for this project:**
https://tanravikorn.github.io/JavaDoc/index.html

## 1.Package application

### 1.1 Class Main extends Application

The main entry point for the Candy Crush game application.

**javadoc for this package** :
https://tanravikorn.github.io/JavaDoc/application/package-summary.html

## 2.Package gui

### 2.1 Package base

2.1.1) **Class SoundManager** Manages audio playback for the application, including background music (BGM) and sound effects (SFX).

2.1.2) **Class ViewManager** Manages the navigation between different views (scenes) in the application.

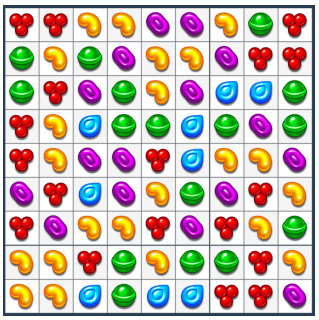
2.1.3) **Interface View** Defines the contract for all GUI views in the application.

**javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/gui/base/package-summary.html

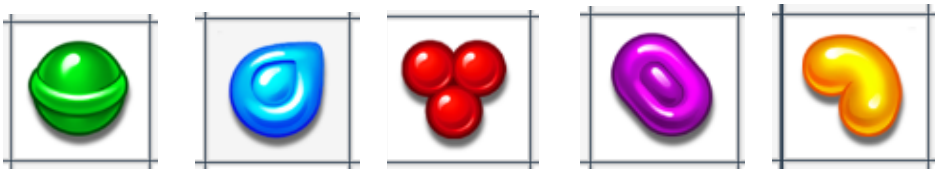
### 2.2 Package components

2.2.1) **Class AnimatedBackground** Creating a dynamic, visually engaging backdrop for the application's menu screens (such as the Start Menu and Game Over screen).

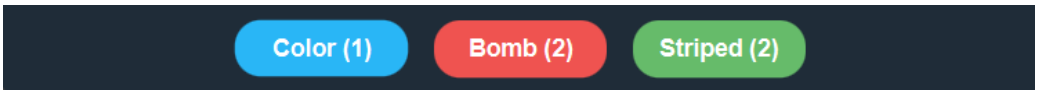
2.2.2) **Class BoardPane extends GridPane** Represents the visual grid layout of the game board.



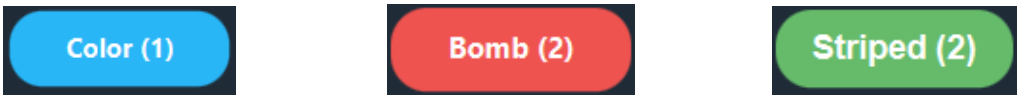
2.2.3) **Class CandyTile extends StackPane** Represents a single visual tile on the game board.



2.2.4) **Class ControlPane extends HBox** Represents the bottom control panel of the game interface.



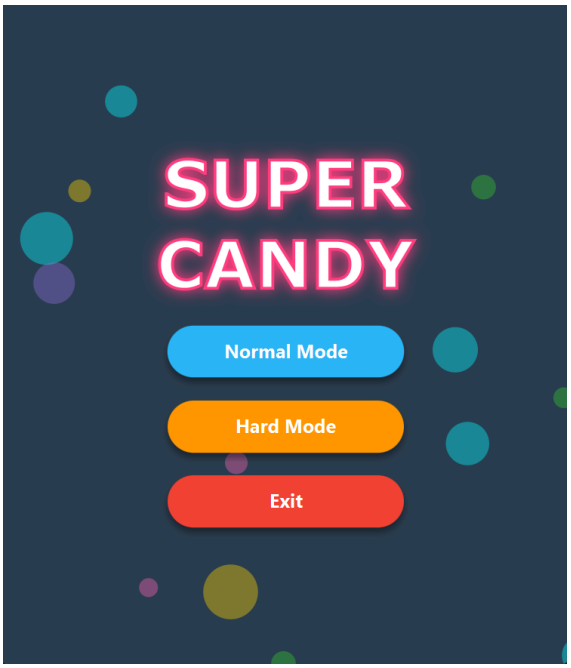

2.2.5) **Class ItemPane extends StackPane** This class handles user interaction, logic execution via the controller, and visual state updates.



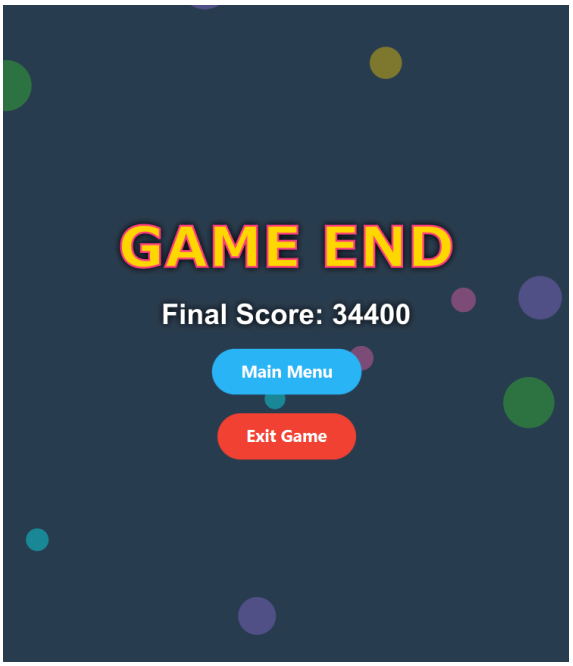

**javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/gui/components/package-summary.html

## 2.3 Package views

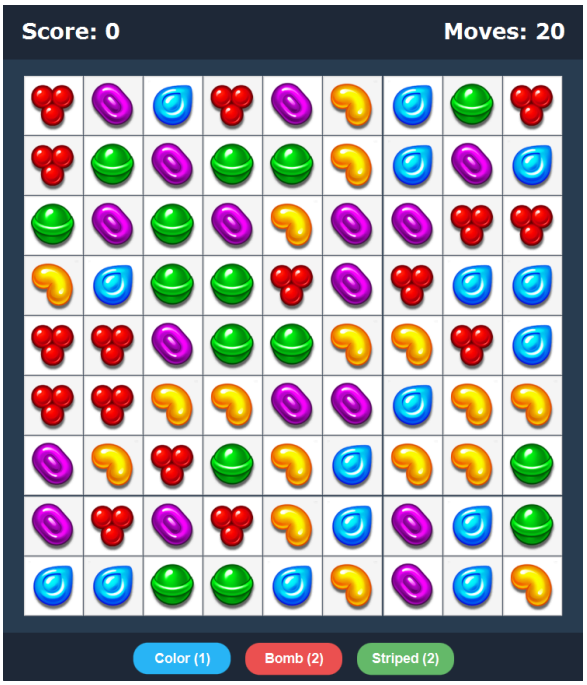
2.3.1) **Class StartView implements View** Represents the Main Menu screen of the application.



2.3.2) **Class EndView implements View** Represents the "Game Over" screen displayed when the game finishes.




2.3.3) **Class GameView implements View** Represents the primary gameplay screen of the application.




**javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/gui/views/package-summary.html

# 3.Package logic

## 3.1 Package board

3.1.1) **Class Board** Represents the game board (grid) containing all the candies.

3.1.2) **Class BoardInitializer** This class is responsible for setting up the initial state of the game board.

3.1.2) **Class BoardUpdater** This class responsible for updating the board state after a move or explosion.

**Javadoc for this package :**

https://tanravikorn.github.io/JavaDoc/logic/board/package-summary.html

## 3.2 Package candy

3.2.1) **Class Candy** Represents a single candy piece on the game board.

3.2.2) **Enum Class CandyColor** Enumerates the possible colors of a candy.

3.2.3) **Enum Class CandyType** Defines the specific type or power-up state of a candy.

**Javadoc for this package :**

https://tanravikorn.github.io/JavaDoc/logic/candy/package-summary.html

## 3.3 Package Explosion

3.3.1) **Interface Class ExplosionStrategy** Defines the strategy interface for explosion logic.

3.3.2) **Abstract Class ExplosionBase** An abstract base class providing common utility methods for explosion strategies. *implement ExplosionStrategy*

All of Classes below are extend form *ExplosionBase*

3.3.3) **Class NormalExplosion** Represents a standard match-3 explosion.

3.3.4) **Class LineExplosion** Represents a linear explosion logic, typically used by Striped Candies.

3.3.5) **Class AreaExplosion** Represents an area-of-effect explosion, typically used by Bomb Candy.

3.3.6) **Class ColorBombExplosion** Represents the explosion of a Color Bomb.

3.3.7) **Class CrossExplosion** Represents a cross-shaped explosion combo : **Striped + Striped**

3.3.8) **Class ColorStripedExplosion** Represents the combo explosion of a **Color Bomb + Striped Candy**.

3.3.9) **class ColorBombBombExplosion** Represents the combo explosion of a **Color Bomb + Candy Bomb**.

3.3.10) **class NukeExplosion** Represents a massive explosion that affects the entire board. **Color Bomb + Color Bomb**

**Javadoc for this package :**

https://tanravikorn.github.io/JavaDoc/logic/explosion/package-summary.html

## 3.4 Package item

3.4.1) **Abstract class Item** An abstract base class representing a usable item or power-up in the game.

All of Classes below are extend form *Class Item*

3.4.2) **Class BombItem** An item that transforms random candies into **Bomb Candies**.

3.4.3) **Class IceBreakerItem** An adaptive item that behaves differently based on the **GameMode**.

3.4.4) **Class StripedItem** An item that transforms random candies into **Striped Candies**.

**Javadoc for this package :**

https://tanravikorn.github.io/JavaDoc/logic/Item/package-summary.html

## 3.5 Package utils

3.5.1) **Class MatchFinder** Utility class responsible for detecting matches and checking for possible moves on the board.

3.5.2) **Class MatchProcessor** Handles the logic for processing matched candy clusters.

3.5.3) **Class SpecialCandyRules** Utility class defining the rules for creating special candies based on match shapes.

3.5.4) **Class CandyMixer** This class is responsible for determining the explosion strategy when two special candies are combined.

3.5.5) **Class Point** Represents a coordinate on the game board (row and column).

**Javadoc for this package :**

https://tanravikorn.github.io/JavaDoc/logic/utils/package-summary.html

### 3.6 Package controller

3.6.1) **Enum Class GameMode** Defines the available difficulty levels for the game.

3.5.2) **Enum Class GameState** Represents the current phase of the game loop.

3.5.3) **Class GameController** The central controller class responsible for orchestrating the entire game flow.

**Details:**

- Manages the game state, score, and move limits.
- Acts as the bridge between the data model (Board) and the user interface.
- Handles item inventory and item usage logic.
- Controls the game initialization and reset processes.

**Javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/logic/controller/package-summary.html

## 4.Package test

### 4.1 Package candy

4.1.1) **Class CandyTest** Test suite for the Candy class. Focuses on verifying the correct initialization of candy properties, state modification via setters, and safe execution of explosion methods.

**Javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/test/candy/package-summary.html

### 4.2 Package board

4.2.1) **Class BoardTest** Test suite for the Board class. Focuses on verifying the grid structure, boundary checks, and candy storage mechanisms.

4.2.2) **Class BoardInitializerTest** Test suite for BoardInitializer.Focuses on ensuring the initial board state is valid (no initial matches, contains moves) and respects the Game Mode settings.

4.2.3) **Class BoardUpdaterTest** Test suite for BoardUpdater.Focuses on testing the gravity mechanics, board refilling logic, and interaction with frozen obstacles.

**Javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/test/board/package-summary.html

## 4.3 Package explosion

4.3.1) **Class ExplosionTest** Test suite for individual ExplosionStrategy implementations. Focuses on verifying the blast radius and logic of single special candies (Striped, Bomb, Color Bomb) and their interaction with frozen obstacles.

4.3.2) **Class ComboTest** Test suite for complex Combo Explosions. Focuses on verifying the logic of mixing two special candies (e.g., Color Bomb + Striped, Striped + Striped).

**Javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/test/explosion/package-summary.html

## 4.4 Package gamecontroller

4.4.1) **Class GameControllerTest** Test suite for the GameController class. Focuses on verifying game flow, state management, inventory handling, and move validation.

**Javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/test/gamecontroller/package-summary.html

## 4.5 Package item

4.5.2) **Class ItemTest** Test suite for the Item subclasses.Focuses on verifying that items correctly identify target candies based on the GameMode and transformation rules.

**Javadoc for this package :**
https://tanravikorn.github.io/JavaDoc/test/item/package-summary.html

## 4.6 Package utils

4.6.1) **Class CandyMixerTest** Test suite for CandyMixer. Focuses on verifying that the correct ExplosionStrategy is returned when combining different types of special candies.

4.6.2) **Class MatchFinderTest** Test suite for MatchFinder. Focuses on detecting valid matches on the board and merging overlapping match clusters.

4.6.3) **Class MatchProcessorTest** Test suite for MatchProcessor. Focuses on verifying that matched clusters are correctly processed into special candies (Color Bomb, Striped, Bomb) or removed.

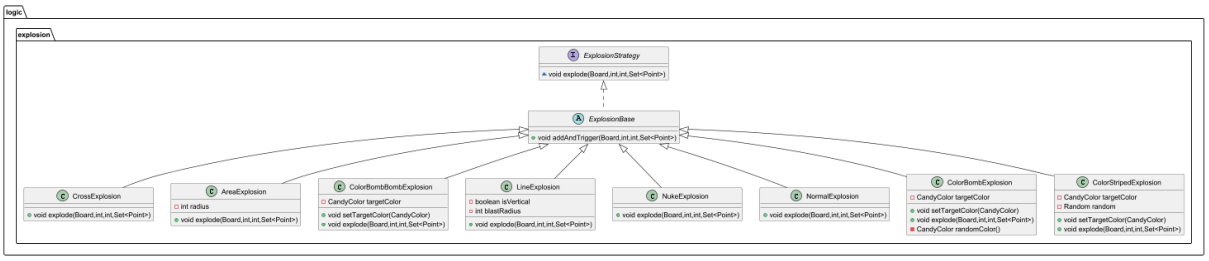
4.6.4) **Class PointTest** Test suite for the Point class. Focuses on verifying coordinate storage and value-based equality/hashing for use in Collections.

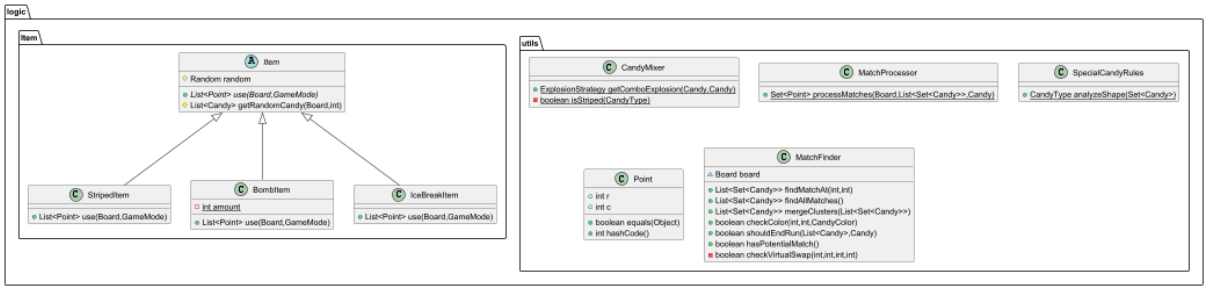
4.6.5) **Class SpecialCandyTest** Test suite for SpecialCandyRules. Focuses on verifying that the shape analysis logic correctly identifies patterns (Line, L-Shape, Match-3) and assigns the correct CandyType.

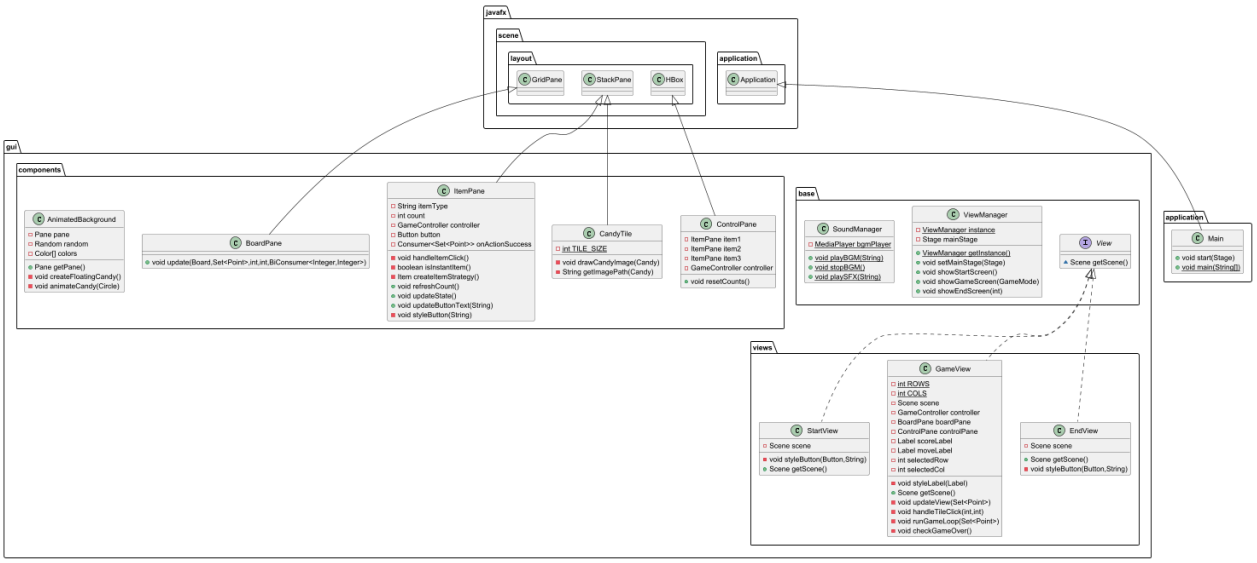
**Javadoc for this package :**

https://tanravikorn.github.io/JavaDoc/test/utils/package-summary.html

# Plant UML


logic
explosion
ExplosionStrategy
+ void explode(Board,int,int,Set<Point>)
ExplosionBase
+ void addAndTrigger(Board,int,int,Set<Point>)
CrossExplosion
+ void explode(Board,int,int,Set<Point>)
AreaExplosion
- int radius
+ void explode(Board,int,int,Set<Point>)
ColorBombBombExplosion
- CandyColor targetColor
+ void setTargetColor(CandyColor)
+ void explode(Board,int,int,Set<Point>)
LineExplosion
- boolean isVertical
- int blastRadius
+ void explode(Board,int,int,Set<Point>)
NukeExplosion
+ void explode(Board,int,int,Set<Point>)
NormalExplosion
+ void explode(Board,int,int,Set<Point>)
ColorBombExplosion
- CandyColor targetColor
+ void setTargetColor(CandyColor)
+ void explode(Board,int,int,Set<Point>)
- CandyColor randomColor()
ColorStripedExplosion
- CandyColor targetColor
- Random random
+ void setTargetColor(CandyColor)
+ void explode(Board,int,int,Set<Point>)


logic
item
Item
# Random random
+ List<Point> use(Board,GameMode)
+ List<Candy> getRandomCandy(Board,int)
StripedItem
+ List<Point> use(Board,GameMode)
BombItem
- int amount
+ List<Point> use(Board,GameMode)
IceBreakItem
+ List<Point> use(Board,GameMode)
utils
CandyMixer
+ ExplosionStrategy getComboExplosion(Candy,Candy)
- boolean isStriped(CandyType)
MatchProcessor
+ Set<Point> processMatches(Board,List<Set<Candy>>,Candy)
SpecialCandyRules
+ CandyType analyzeShape(Set<Candy>)
Point
# int r
# int c
+ boolean equals(Object)
+ int hashCode()
MatchFinder
- Board board
+ List<Set<Candy>> findMatchAt(int,int)
+ List<Set<Candy>> findAllMatches()
+ List<Set<Candy>> mergeClusters(List<Set<Candy>>)
+ boolean checkColor(int,int,CandyColor)
+ boolean shouldEndRun(List<Candy>,Candy)
+ boolean hasPotentialMatch()
- boolean checkVirtualSwap(int,int,int,int)


javafx
scene
layout
GridPane
StackPane
HBox
application
Application
gui
components
AnimatedBackground
- Pane pane
- Random random
- Color[] colors
+ Pane getPane()
- void createFloatingCandy()
- void animateCandy(Circle)
BoardPane
+ void update(Board,Set<Point>,int,int,BiConsumer<Integer,Integer>)
ItemPane
- String itemType
- int count
- GameController controller
- Button button
- Consumer<Set<Point>> onActionSuccess
- void handleItemClick()
- boolean isInstantItem()
- Item createItemStrategy()
+ void refreshCount()
+ void updateState()
+ void updateButtonText(String)
- void styleButton(String)
CandyTile
- int TILE_SIZE
- void drawCandyImage(Candy)
- String getImagePath(Candy)
ControlPane
- ItemPane item1
- ItemPane item2
- ItemPane item3
- GameController controller
+ void resetCounts()
base
SoundManager
- MediaPlayer bgmPlayer
+ void playBGM(String)
+ void stopBGM()
+ void playSFX(String)
ViewManager
- ViewManager instance
- Stage mainStage
+ ViewManager getInstance()
+ void setMainStage(Stage)
+ void showStartScreen()
+ void showGameScreen(GameMode)
+ void showEndScreen(int)
View
+ Scene getScene()
application
Main
+ void start(Stage)
+ void main(String[])
views
StartView
- Scene scene
- void styleButton(Button,String)
+ Scene getScene()
GameView
- int ROWS
- int COLS
- Scene scene
- GameController controller
- BoardPane boardPane
- ControlPane controlPane
- Label scoreLabel
- Label moveLabel
- int selectedRow
- int selectedCol
- void styleLabel(Label)
+ Scene getScene()
- void updateView(Set<Point>)
- void handleTileClick(int,int)
- void runGameLoop(Set<Point>)
- void checkGameOver()
EndView
- Scene scene
+ Scene getScene()
- void styleButton(Button,String)