Welcome to the comprehensive documentation for the Pixel Blast game! This project is a visually-rich, Gosu-powered puzzle game featuring animated effects, grid-based mechanics, and interactive UI. Below you'll find detailed explanations for each code file, including architecture diagrams, usage notes, and code highlights.

`run.rb`

This file boots up the game window and starts the game loop.

```ruby
require './lib/ui/window'

window = PixelblastWindow.new(800, 1000)
window.random_block_id = rand(8)
window.caption = 'Pixel Blast'
window.show
```

- **Purpose:** Launches the game by creating the `PixelblastWindow` and setting its caption and initial random block.
- **Usage:** Run this script to start the Pixel Blast game.

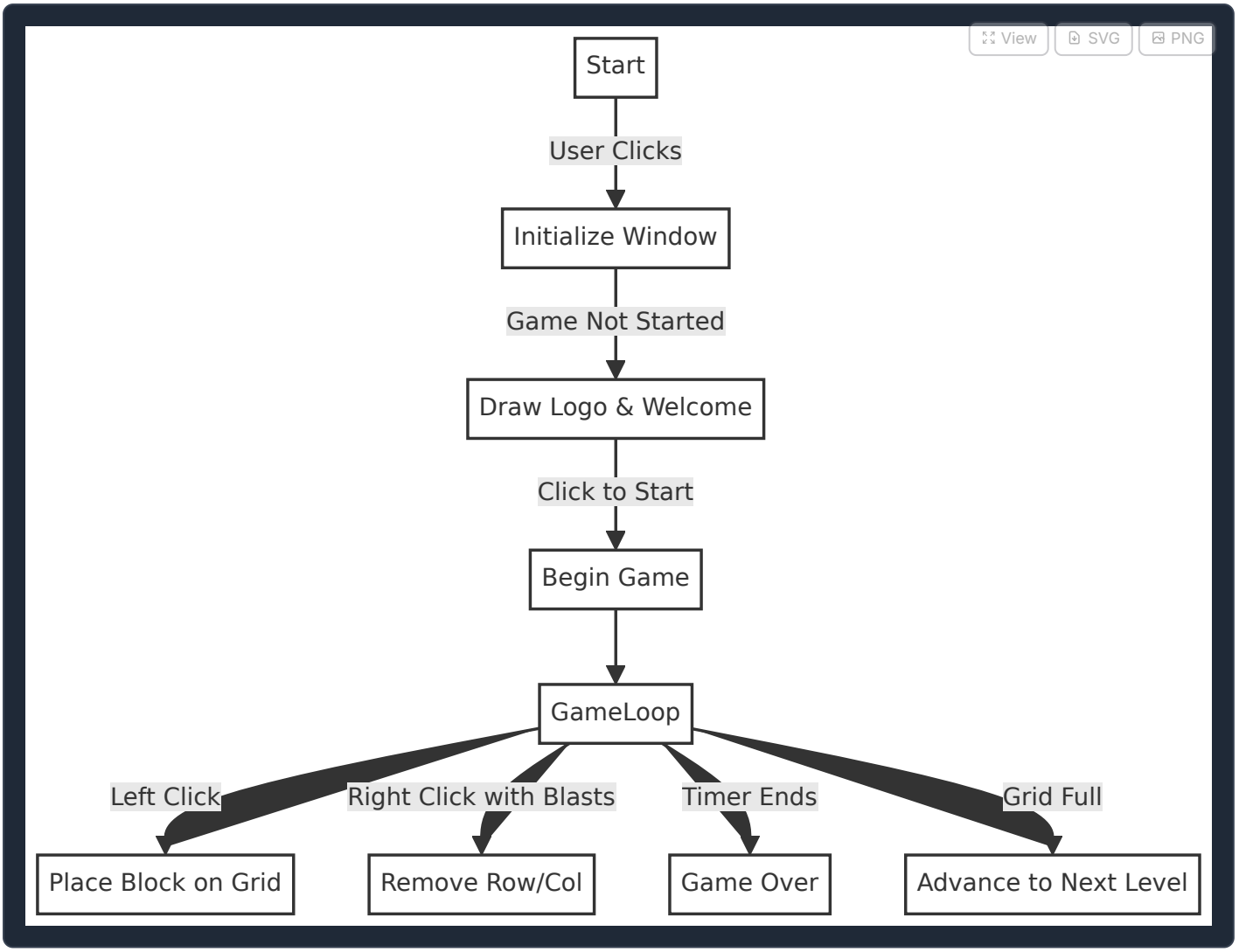
`window.rb`

Handles the core game window, event loop, and drawing/updating routines.

- Inherits from `Gosu::Window`.
- Manages UI, game state, and input events.
- Handles drawing of all game components.

```ruby
class PixelblastWindow < Gosu::Window
  # ... initialization and methods ...
end
```

- **Initialization:** Loads media, UI components, background effects, and the game grid.
- **Event Handling:** Listens for mouse clicks to start the game, place blocks, or use "blasts" for removing rows/columns.
- **Drawing:** Renders the game logo, statistics, current grid, next block, and special effects (starfield, code rain).
- **Game State:** Updates score, level, blasts, and checks for game-over conditions.
- **Timer:** Uses a custom timer for game pacing.



`code_rain.rb`

Implements the vertical streams of symbols (like The Matrix).

- Generates animated columns of random symbols.
- Each symbol has a color and animation offset.

```
class CodeStream
  # ... responsible for code rain effect ...
end
```

- **Usage:** Called in the main window to draw animated code rain in the background.

`gif_player.rb`

Handles loading and displaying animated GIFs.

- Loads frames from a pattern of images (with optional delay in filename).
- Supports tinting and scaling.
- Calculates which frame to display based on elapsed time.

```
class GifPlayer
  # ... animation and rendering logic ...
end
```

- **Usage:** Useful for animated UI elements or effects.

`starfield.rb`

Creates a dynamic, flickering starfield with optional "blast" animations.

- Randomly positions stars and animates their brightness to create a flicker effect.
- Occasionally triggers "blasts" (shooting stars with fading trails).

```
class Starfield
  # ... star and blast animation ...
end
```

- **Usage:** Drawn in the game background for visual appeal.

`media_loader.rb`

Centralizes loading of all game media assets.

- Loads sound effects, music tracks, and fonts.
- Randomly picks a song for each session.
- Loads a random background image from the assets.

```
class MediaLoader
  # ... asset references and initialization ...
end
```

- **Usage:** Used throughout the game for UI, sound effects, and music.

`timer.rb`

A simple timer for managing game time without threads.

- Provides `start`, `tick`, and `add_time` methods.
- Triggers an optional callback on completion.

```
class Timer
  # ... frame-independent timing logic ...
end
```

`matrix_2d.rb`

A module containing helper methods for grid (matrix) manipulation.

- Transposing matrices.
- Checking for full rows/columns.
- Clearing rows/columns.
- Checking sub-matrix placement.

```
module Matrix2D
  # ... various matrix/grid helper methods ...
end
```

`button.rb`

Draws interactive buttons on the screen.

- Draws a button with text, background, and hover effect.
- Detects mouse hover and clicks.

```
class Button
  # ... button drawing and events ...
end
```

`draw.rb`

Provides utilities for drawing game elements and grids.

- Draws pixel-perfect grids.
- Draws thick lines, rectangles, and image regions.
- Renders the game grid and next block preview.

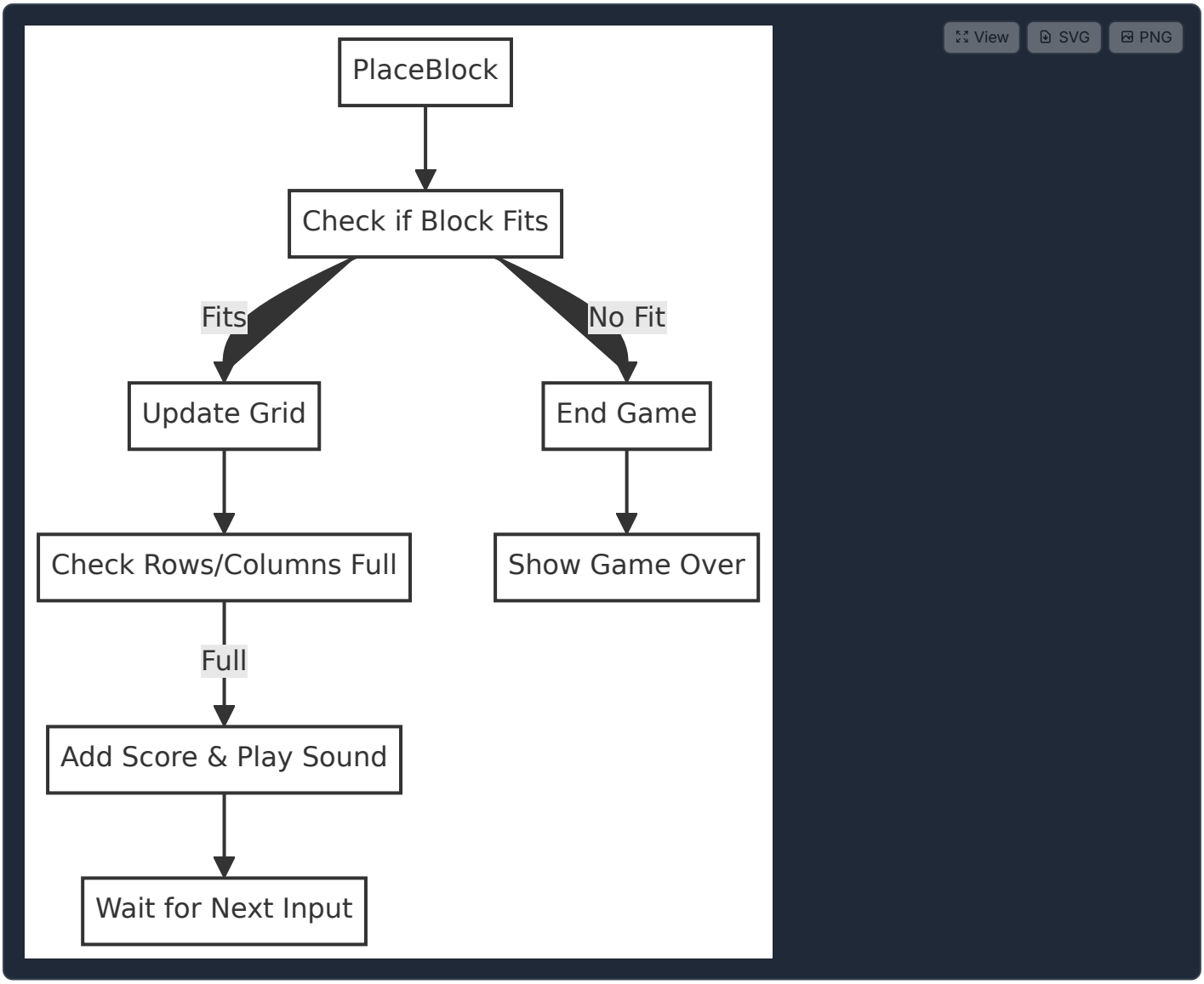
```ruby
module Draw
  # ... drawing utilities ...
end
```

`helper.rb`

Centralizes game logic and state management.

- Manages score, blasts, level, and other game state.
- Handles block placement logic and scoring.
- Integrates with audio cues for feedback.
- Checks for row/column completion, applies scoring, and triggers effects.

```ruby
class PixelblastHelper
  # ... core game logic ...
end
```



`random_generator.rb`

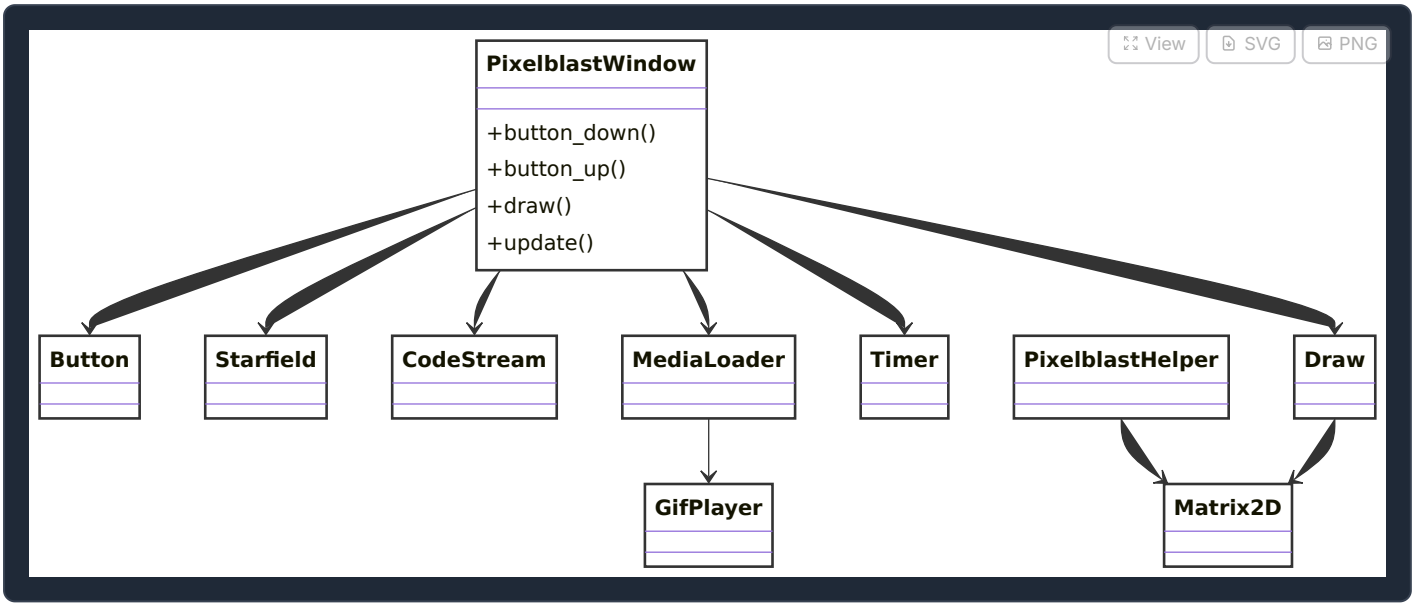
Handles random asset selection and color generation.

- Randomly selects files from folders (for backgrounds).
- Generates vivid random colors.
- Random unique number generation.

```ruby
module RandomGenerator
  # ... asset, color and number utilities ...
end
```

This game is built around a classic loop of:

- **Input Handling:** User clicks and key presses.
- **State Update:** Game logic and timer updates.
- **Rendering:** Drawing all game elements each frame.



- **Grid System:** The game board uses a 2D matrix to keep track of placed blocks and cleared lines.
- **Block Placement:** Players interact by placing blocks of different shapes onto the grid.
- **Clearing:** Rows or columns filled by blocks can be cleared for points and special effects.
- **Blasts:** Players can use limited "blasts" to clear a row or column.
- **Visual Effects:** Animated backgrounds, code rain, and starfields add polish.
- **Audio Feedback:** Sounds and music enhance user experience at key moments (placing blocks, clearing, game over).

> **Best Practice: Centralized State**
>
> All core game state is managed through PixelblastHelper for consistency and easier debugging.

`run.rb`

Purpose:
Launches the game window and sets up the initial game state.

- Instantiates `PixelblastWindow` with a fixed size.

- Assigns an initial random block.
- Sets window caption and runs the main show loop.

`window.rb`

Purpose:
Defines the main game window class, handles input, drawing, and updating the screen.

- **Initialization:** Loads media, fonts, background, UI elements, and effects.
- **Event Handlers:**
  - `button_down`: Handles left/right mouse clicks for gameplay and UI.
  - `button_up`: Places blocks on the grid, starts timers, and handles music.
- **Draw Routine:**
  - Renders logo, stats, grid, next block, code rain, and starfield.
- **Update Routine:**
  - Updates animation, timers, and checks for game-over or level transitions.

`code_rain.rb`

Purpose:
Draws a Matrix-style animated "code rain" for visual effect.

- Configurable font, character set, speed, and color.
- Continuously updates to animate the falling characters.

`gif_player.rb`

Purpose:
Handles animated GIF playback for UI or effects.

- Loads images matching a filename pattern.
- Supports per-frame delays, scaling, and tinting.
- Draws the current frame based on real-time.

`starfield.rb`

Purpose:
Animates a field of flickering stars and random shooting blasts.

- Each star flickers using a sine-wave brightness.
- Blasts are animated lines with a fading trail.

`media_loader.rb`

Purpose:
Manages loading of all sound, music, and font resources.

- Loads a random background image.

- Randomly selects a music track for variety.
- Provides references to all sounds and fonts for use elsewhere.

`timer.rb`

Purpose:
Implements a simple, update-loop-driven timer.

- Can be started, ticked, and queried for remaining time.
- Triggers a callback when finished.

`matrix_2d.rb`

Purpose:
Provides handy functions for manipulating a 2D grid (matrix).

- Transposing, checking, clearing, and setting matrix cells.
- Sub-matrix searching for block placement logic.

`button.rb`

Purpose:
Reusable button rendering and interaction detection.

- Draws a rectangle with a label and hover feedback.
- Detects clicks and hovers based on mouse coordinates.

`draw.rb`

Purpose:
Defines pixel-perfect grid and block drawing utilities.

- Grid lines, thick lines, rectangles, borders, and cropped image rendering.
- Main tool for drawing the game field and block previews.

`helper.rb`

Purpose:
Game state and logic manager.

- Tracks score, level, block placement, and grid state.
- Handles block placement, row/column checking, game-over logic, and sound triggers.

`random_generator.rb`

Purpose:
Random utility functions for asset selection and color generation.

- Chooses random files (for backgrounds).
- Generates vivid random colors for block rendering.
- Ensures random numbers aren't repeated when necessary.

> **Visual Effects for Engagement**
>
> Starfield and code rain effects enhance game atmosphere and keep the visual experience dynamic.

- **Pixel Blast** uses a modular codebase with clear separation between game logic, rendering, effects, and utilities.
- Visual and audio polish make for a compelling user experience.
- The helper modules and centralized state pattern simplify game logic and maintainability.
- The architecture is flexible, allowing easy addition of new block types, effects, or features.

For further information, please refer to the comments in each file or explore the code for customizations and enhancements!