This documentation covers every major code and configuration file in your web-based game project. It explains the structure, purpose, and logic of each file, addressing both configuration and gameplay implementation.

This YAML file configures a GitHub Actions workflow for deploying static content to GitHub Pages when changes are pushed to the `main` branch.

- Automates deployment of your static site to GitHub Pages.
- Ensures that only one deployment runs at a time.
- Sets proper permissions for the deployment process.

- **Checkout**: Retrieves the latest source code from the repository.
- **Setup Pages**: Prepares the environment for GitHub Pages.
- **Upload Artifact**: Uploads the full repository as an artifact.
- **Deploy to GitHub Pages**: Deploys the artifact to GitHub Pages.

```yaml
name: Deploy static content to Pages
on:
  push:
    branches: ["main"]
  workflow_dispatch:
permissions:
  contents: read
  pages: write
  id-token: write
concurrency:
  group: "pages"
  cancel-in-progress: false
jobs:
  deploy:
    environment:
      name: github-pages
      url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Setup Pages
        uses: actions/configure-pages@v5
      - name: Upload artifact
        uses: actions/upload-pages-artifact@v3
        with:
          path: '.'
      - name: Deploy to GitHub Pages
        id: deployment
        uses: actions/deploy-pages@v4
```

This is the HTML entry point for the Cross Road game.

- Sets up the page structure and loads resources.
- Includes a root div (`#root`) for a React-based game.
- Loads a stylesheet and the JavaScript game module.

- **`#root`**: The React app mounts here.
- **`style.css`**: Applies the retro game style.
- **`script.js`**: Contains all gameplay and rendering logic.

```html
<!DOCTYPE html>
<html lang="en" >
<head>
  <meta charset="UTF-8">
  <title>Cross Road</title>
  <link rel="stylesheet" href="./style.css">
</head>
<body>
  <div id="root" />
  <script type="module" src="./script.js"></script>
</body>
</html>
```

This stylesheet controls the visual appearance and layout of the Cross Road game.

- **Retro Game Font**: Uses the "Press Start 2P" font for a pixel-art look.
- **Flex Layout**: Centers the game vertically and horizontally.
- **Control Layout**: Styles the arrow/WASD movement buttons at the bottom.
- **Score and Result**: Styles the floating score and game over modal.

```css
body {
  margin: 0;
  display: flex;
  min-height: 100vh;
}
#root { width: 100%; }
.game { position: relative; width: 100%; height: 100%; font-family: "Press Start 2P", cursive; }
/* ... more styles ... */
```

This is the main game logic for the Cross Road game, written in React and using react-three-fiber for 3D rendering.

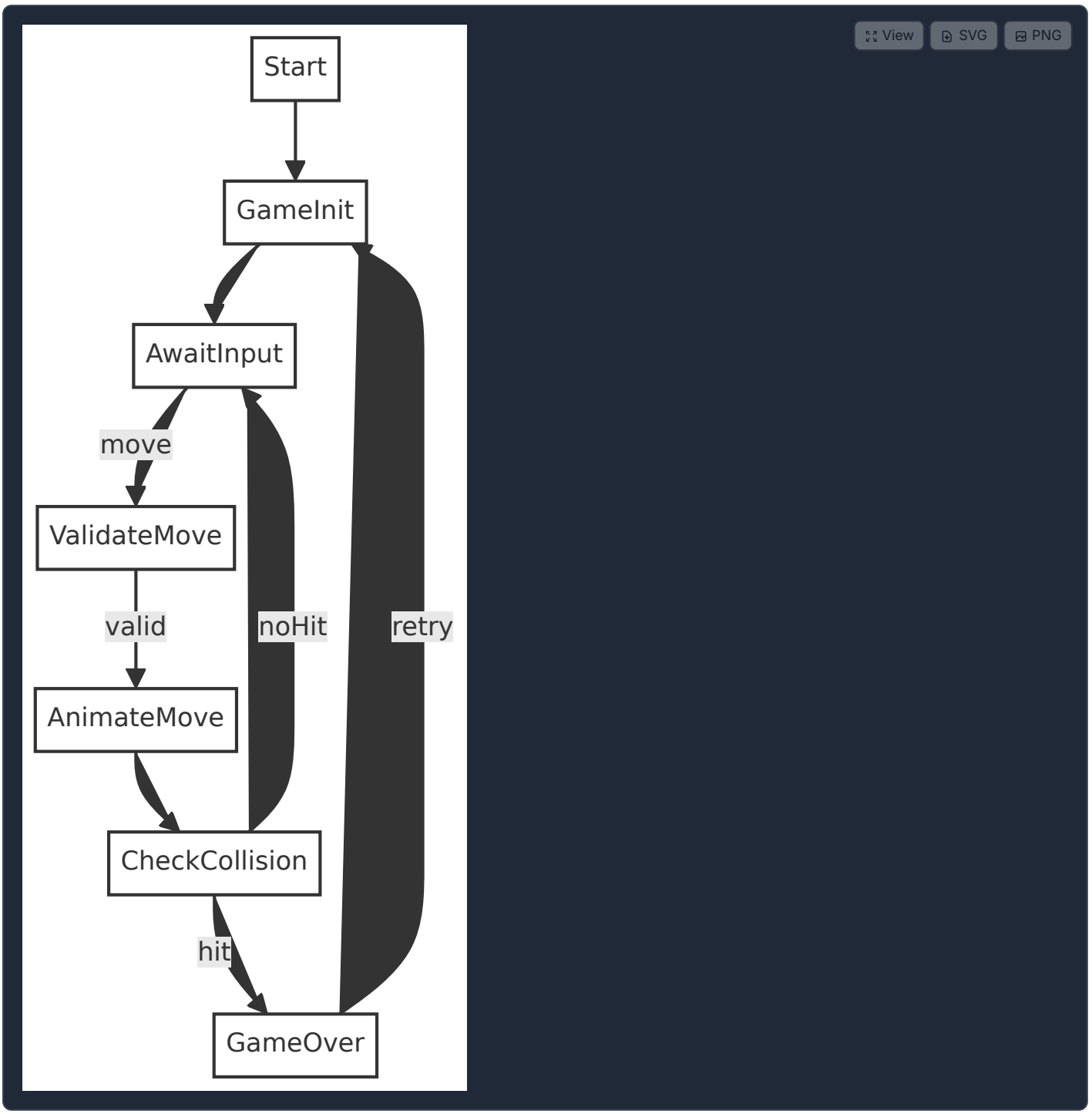
- Implements all gameplay, rendering, and state logic for a 3D Crossy Road-like game.
- Uses Zustand for state management, React for UI, and Three.js for 3D graphics.

- **Game**: Main React component orchestrating the game.
- **Scene**: 3D canvas for the game world.
- **Player**: Handles player position, movement, and state.
- **Map, Grass, Road, Vehicle, Tree**: Render the game board and obstacles.
- **useFrame**: Animates movement and vehicle logic.
- **useGameStore, useMapStore**: Zustand-based stores to manage global game and map state.

- Player movement is queued and validated.

- Vehicles are animated and loop around the road.
- Collision checks are performed using Three.js bounding boxes.

- Movement via arrow keys or WASD.
- Clickable UI buttons for movement.



```
function queueMove(direction) {
  // Check if game is running
  if (useGameStore.getState().status !== "running") return;
  // Validate move
  const isValidMove = endsUpInValidPosition({
    rowIndex: playerState.currentRow,
    tileIndex: playerState.currentTile,
  }, [...playerState.movesQueue, direction]);
  if (!isValidMove) return;
  playerState.movesQueue.push(direction);
}
```

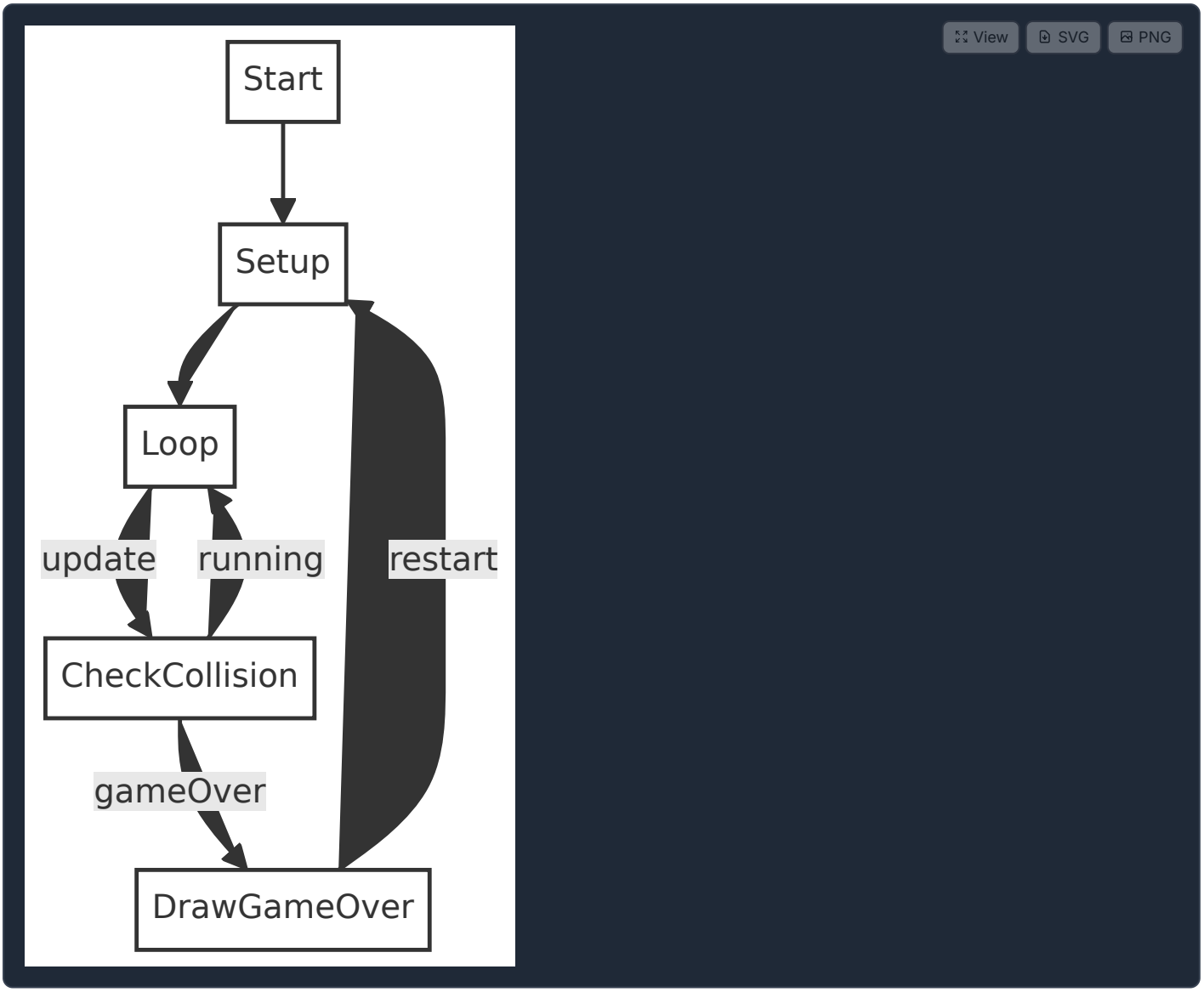
This file is a compiled or minified version of the Cross Road React/Three.js game. The core logic is the same as the previous file, but the code is converted to React.createElement and optimized for production.

- The logic and architecture are identical to the original Cross Road script.
- All UI, state management, and 3D rendering logic is present but optimized/minified.
- Maintains compatibility with React and Three.js APIs.

This is a **different** arcade runner game implemented using vanilla JavaScript and the Canvas 2D API.

- **Player**: Can jump, has shield, collects power-ups, and scores points.
- **Obstacles**: Regularly spawned, moving leftward.
- **Power-ups**: Random type (score multiplier, shield, high jump, slow time).
- **Parallax Background**: Simulated using moving stars and sky/ground.
- **Game Loop**: Handles all updates and drawing.
- **Controls**: `Space` to jump or restart after game over.

- `Player`: Tracks position, physics, shield, and jump status.
- `Obstacle`: Handles obstacle positions and movement.
- `PowerUp`: Handles types and temporary effects.
- `Game`: Contains game state, logic, and main loop.



Type	Color	Effect
Score Multiplier	#ffd700	x2 score for 5 sec
Shield	#00bfff	Blocks 1 collision
High Jump	#32cd32	Higher jumps
Slow Time	#8a2be2	Slows all obstacles

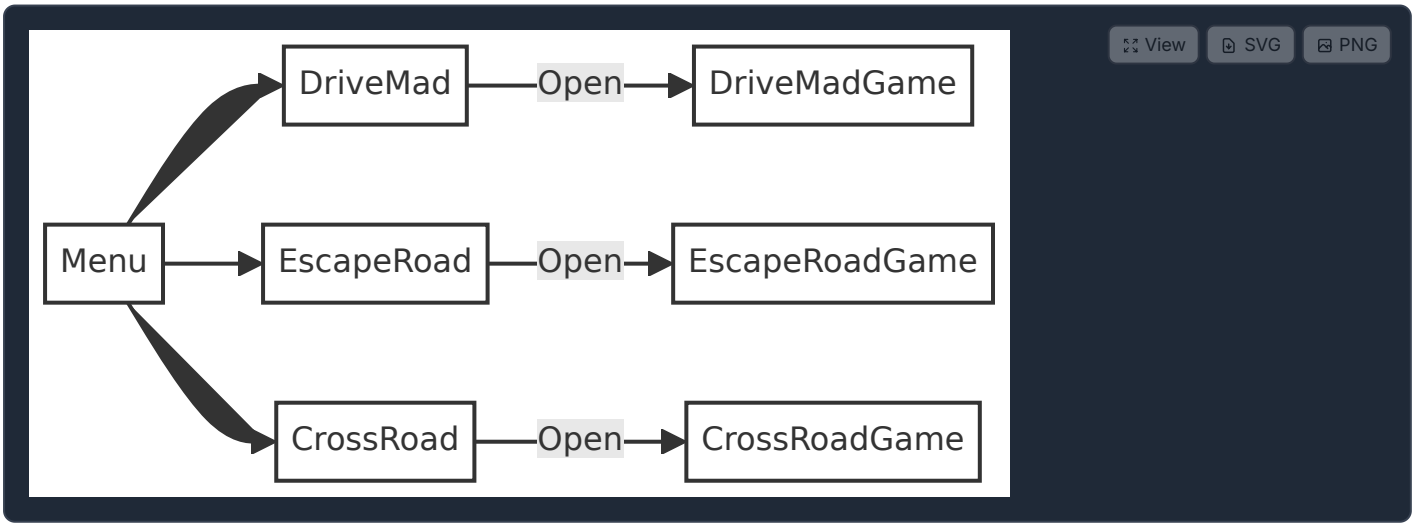
```
activatePowerUp(powerUp) {
  switch (powerUp.type) {
    case "scoreMultiplier":
      this.scoreMultiplier = 2;
      setTimeout(() => { this.scoreMultiplier = 1; }, powerUp.duration);
      break;
    // ... other cases
  }
}
```

Power-Up Effects

Power-ups provide temporary abilities (e.g. shield, double score, high jump, slow time) and expire after a few seconds.

This file builds a visually appealing, accessible menu for launching games.

- **Responsive Grid Menu**: Each game is a card with an icon, title, and description.
- **Accessibility**: Keyboard navigation, focus/hover effects, ARIA attributes.
- **Script Enhancements**: Keyboard activation for opening games in new tabs.
- **AdSense / Analytics**: Scripts for Google Ads and analytics.
- **Custom CSS**: Modern, dark theme with glass, accent, and responsive layout.



This file is the entry point for the "Drive Mad" game.

- **WebAssembly Loader**: Loads the game's WASM and supporting files.
- **Custom Styles**: Loads and applies game-specific fonts and styling.
- **Responsive Canvas**: Initializes a large canvas for 3D gameplay.
- **Loading Progress**: Shows progress while assets load.
- **Game Metadata**: Shows title, author, and cover art.
- **Firebase/Analytics**: Embedded scripts for analytics.

This file is the entry for the "Escape Road" game.

- **Unity Game Loader**: Loads the Unity WebGL build.
- **Responsive Canvas**: Sets up the Unity canvas, loading bar, and logo.
- **Firebase Analytics**: Sets up Firebase and analytics scripts.
- **Loader Scripts**: Handles Unity instance creation, loading, and progress display.
- **Custom Loader**: Draws a custom progress bar and logo animation as the game loads.

File	Purpose / Technology	Main Features
static.yml	Github Actions Workflow	Deploys static site to GitHub Pages
index.html (Cross Road)	React/Three.js Game Entry	Mounts 3D game, styles, JS import
style.css	CSS	Retro style, layout, controls
Cross Road Game	React/Three.js (JSX)	3D gameplay, Zustand state, UI
script.js (React, prod)	React/Three.js (JS, min)	Same as above, minified
script.js (Canvas)	Canvas 2D Game	2D runner, power-ups, physics
index.html (Menu)	HTML/CSS/JS	Games grid, accessibility, scripts
index.html (Drive Mad)	HTML/WebAssembly Loader	Loads WASM game, progress, styles
index.html (Escape Road)	HTML/Unity Loader	Loads Unity WebGL, loader canvas

---

Loading diagram...Failed to render diagram

Diagram Error

Parse error on line 2: ...D Menu[index.html (Menu)] -->|Launch| ----------^ Expecting 'SQE', 'DOUBLECIRCLEEND', 'PE', '-)', 'STADIUMEND', 'SUBROUTINEEND', 'PIPE', 'CYLINDEREND', 'DIAMOND_STOP', 'TAGEND', 'TRAPEND', 'INVTRAPEND', 'UNICODE_TEXT', 'TEXT', 'TAGSTART', got 'PS'

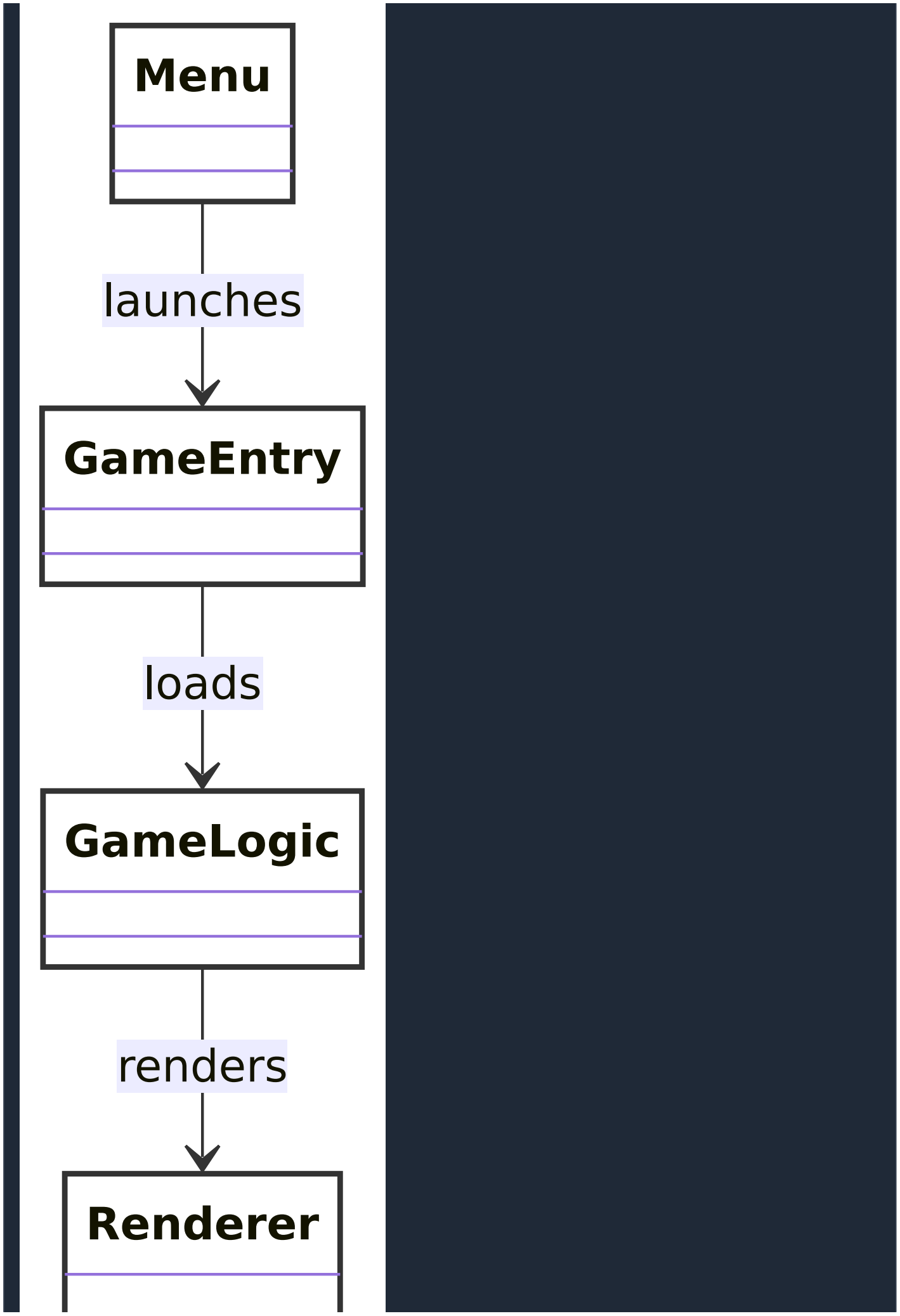
▶ Show details

Fix diagram

Best Practices

Keep workflow files up to date. Use Zustand or similar for game state. Use ARIA and keyboard navigation for accessibility. Separate game logic from UI.


View
SVG
PNG


Menu
launches
GameEntry
loads
GameLogic
renders
Renderer

- The project is well-structured for multiple games, each with its own loader and entry point.
- **Cross Road** uses modern JS frameworks for 3D gameplay; **Escape Road** uses Unity WebGL; **Drive Mad** uses WASM.
- The workflow automates deployment, and the menu improves user experience with accessibility.
- **No explicit API endpoints** are present in the code—this is a fully client-side/browser-based game collection.

If you need deeper details for any file or a new section (such as interactive API blocks for a future backend), let me know!