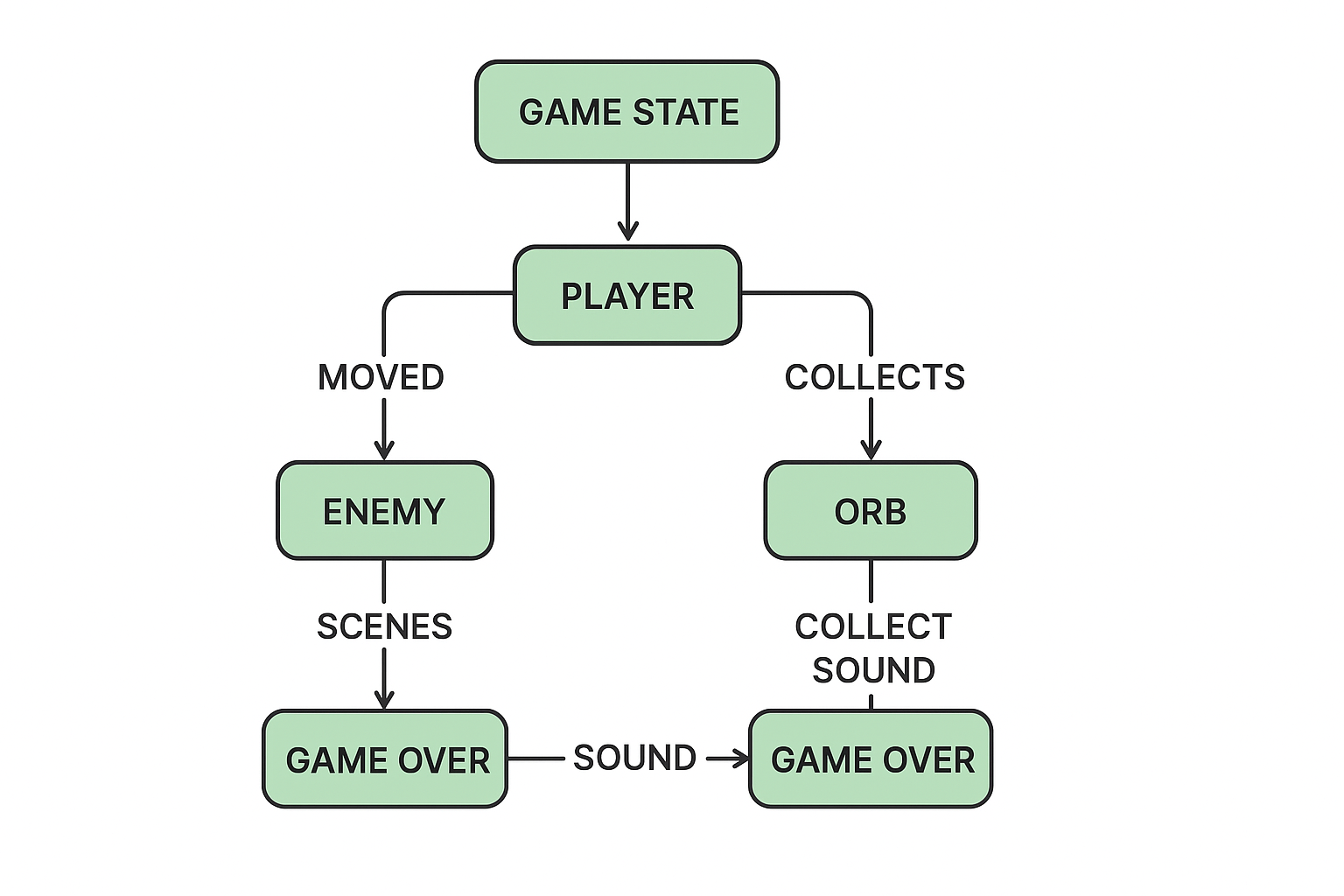
# **React Survival Game – Detailed Documentation**

## **Project Overview**

A 2D grid-based survival game built with **React** and **Redux Toolkit**.  
 **Goal:** Collect orbs, avoid enemies, survive as long as possible.  
 **Features:** Universe-style background, sound effects, animations, HUD, game over modal.



## **1️⃣ App.jsx – Main Game Container**

**Purpose:** Handles game start, HUD, board rendering, keyboard controls, and the main game loop.

## **1. Imports**

import React, { useEffect } from "react";

import { useDispatch, useSelector } from "react-redux";

import { startGame, resetGame, movePlayer, spawnEnemy, moveEnemy, spawnOrb, removeOrb, collectOrb, tick } from "./redux/gameSlice";

import Modal from "./components/Modal";

* React and useEffect → for component and side effects.
* useDispatch → to send actions to the Redux store.
* useSelector → to read values from Redux state.
* Redux actions from gameSlice.
* Modal → reusable modal popup for **Game Over**.

## **2. Reading Redux State**

const { player, enemy, orb, score, timeSurvived, difficulty, gameOver, gameStarted, enemyVisible } = useSelector((state) => state.game);

* Pulls the entire game state from Redux store.
* Makes it easy to **use state values** in the UI.

## **3. Keyboard Controls**

useEffect(() => {

const handleKey = (e) => {

if (!gameStarted || gameOver) return;

const moves = { ArrowUp: [0, -1], ArrowDown: [0, 1], ArrowLeft: [-1, 0], ArrowRight: [1, 0] };

if (moves[e.key]) {

const [dx, dy] = moves[e.key];

dispatch(movePlayer({ x: player.x + dx, y: player.y + dy }));

dispatch(collectOrb());

}

};

window.addEventListener("keydown", handleKey);

return () => window.removeEventListener("keydown", handleKey);

}, [player, gameStarted, gameOver, dispatch]);

* Adds **keyboard controls** for the player.
* Moves the player on arrow keys: ↑, ↓, ←, →.
* Calls collectOrb() to pick up an orb if the player moves to its position.
* The cleanup ensures no duplicate listeners are left when the component re-renders.

## **4. Game Loop**

useEffect(() => {

if (!gameStarted) return;

const interval = setInterval(() => {

dispatch(tick());

if (timeSurvived === 1) dispatch(spawnEnemy());

if (enemyVisible) dispatch(moveEnemy());

if (timeSurvived % 20 === 0 && timeSurvived !== 0) dispatch(spawnOrb());

if (timeSurvived % 20 === 10) dispatch(removeOrb());

}, 1000);

return () => clearInterval(interval);

}, [dispatch, gameStarted, timeSurvived, enemyVisible]);

* Runs every **1 second**:  
  + tick() → increases survival time and score.
  + spawnEnemy() → enemy appears after 1 second.
  + moveEnemy() → enemy chases player every second.
  + Spawns an orb every 20 seconds.
  + Removes the orb 10 seconds later (halfway through the 20-second cycle).

## **5. Rendering the Board**

const renderCell = (row, col) => {

const isPlayer = player.x === col && player.y === row;

const isEnemy = enemyVisible && enemy.x === col && enemy.y === row;

const isOrb = orb && orb.x === col && orb.y === row;

return (

<div

key={`${row}-${col}`}

style={{

width: 40,

height: 40,

backgroundColor: isPlayer ? "blue" : isEnemy ? "red" : isOrb ? "gold" : "#292930",

}}

/>

);

};

* Renders a **10x10 grid**.
* Colors each cell depending on whether it contains:  
  + Player → blue
  + Enemy → red
  + Orb → gold
  + Empty → dark background (#292930)

## **6. UI Layout**

<div style={{ textAlign: "center", padding: 20, fontFamily: "Arial" }}>

<h1>Survival Game</h1>

{!gameStarted && <button onClick={() => dispatch(startGame())}>Start Game</button>}

{gameStarted && !gameOver && (

<>

<p>Score: {score}</p>

<p>Time Survived: {timeSurvived}s</p>

<p>Difficulty: {difficulty}</p>

<div className="instructions"><strong>instructions:</strong> 🎮 Move: ↑, ↓, ←, → </div>

<div

style={{

display: "grid",

gridTemplateColumns: "repeat(10, 40px)",

gap: 2,

justifyContent: "center",

marginTop: 20,

}}

>

{Array.from({ length: 10 }).map((\_, row) =>

Array.from({ length: 10 }).map((\_, col) => renderCell(row, col))

)}

</div>

</>

)}

{gameOver && (

<Modal title="💀 Game Over" onClose={() => dispatch(resetGame())}>

<p>Score: {score}</p>

<p>Time Survived: {timeSurvived}s</p>

</Modal>

)}

</div>

### **Details:**

* Shows **Start Game button** before the game starts.
* During gameplay:  
  + Displays score, time survived, difficulty, and instructions.
  + Shows the **game board grid** using renderCell().
* When game ends:  
  + Shows a **Modal** with final score and survival time.
  + Clicking × resets the game.

## **7. Key Features**

1. Keyboard-controlled player movement.
2. Enemy chases player automatically.
3. Orbs spawn and can be collected.
4. Score increases based on time survived and collected orbs.
5. Game ends when enemy catches player.
6. Reusable modal for **Game Over** popup.

## **2️⃣ redux/gameSlice.js – Game State & Logic**

**Purpose:** Manages **all game state** and game logic using Redux Toolkit.

### **1. Imports**

* import { createSlice } from "@reduxjs/toolkit";
* createSlice is a function from Redux Toolkit (RTK) that helps you **define a slice of your Redux store** easily.
* A slice contains:  
  + state → data for that slice
  + reducers → functions to update the state
  + actions → automatically generated functions for each reducer

### **2. Constants & Helper Function**

* const BOARD\_SIZE = 10;
* const getRandomPosition = () => ({
* x: Math.floor(Math.random() \* BOARD\_SIZE),
* y: Math.floor(Math.random() \* BOARD\_SIZE),
* });
* BOARD\_SIZE defines a **10x10 game board**.
* getRandomPosition() returns a random (x, y) position on the board. This is used for spawning orbs.

### **3. Audio Setup**

* const collectAudio = new Audio("/sounds/collect.wav");
* const gameOverAudio = new Audio("/sounds/gameover.wav");
* These lines load sound files for **collecting orbs** and **game over**.
* They will be played using .play() inside the reducers when events happen.

### **4. Initial State**

* const initialState = {
* player: { x: 0, y: 0, score: 0 },
* enemy: { x: 9, y: 9 },
* orb: null,
* score: 0,
* timeSurvived: 0,
* difficulty: 1,
* gameOver: false,
* gameStarted: false,
* enemyVisible: false,
* };
* player → stores the player's position and individual score.
* enemy → stores the enemy’s position.
* orb → stores the current orb's position (null if none).
* score → total points (orb collection + time survived).
* timeSurvived → counts seconds of survival.
* difficulty → increases every 5 seconds.
* gameOver → true when the enemy catches the player.
* gameStarted → true when the game begins.
* enemyVisible → controls when the enemy appears.

### **5. Slice Definition**

* const gameSlice = createSlice({
* name: "game",
* initialState,
* reducers: { ... }
* });
* Creates a slice named "game" with initialState.
* reducers contain **functions that update state in response to actions**.

### **6. Reducers (Actions)**

#### **a) startGame**

* startGame(state) { ... }
* Initializes the game: resets score, player/enemy positions, time, difficulty, and orb.
* Sets gameStarted to true.

#### **b) resetGame**

* resetGame: () => initialState
* Resets the entire game state back to the original initialState.

#### **c) movePlayer**

* movePlayer(state, action) {
* const { x, y } = action.payload;
* if (!state.gameOver) {
* state.player.x = Math.max(0, Math.min(BOARD\_SIZE - 1, x));
* state.player.y = Math.max(0, Math.min(BOARD\_SIZE - 1, y));
* }
* }
* Moves the player to new coordinates (x, y) **within the board boundaries**.
* Only works if the game is not over.

#### **d) spawnEnemy**

* spawnEnemy(state) {
* state.enemyVisible = true;
* }
* Makes the enemy appear on the board.

#### **e) moveEnemy**

* moveEnemy(state) { ... }
* Moves the enemy **toward the player** one step at a time.
* If enemy touches the player → gameOver becomes true, and the **game over sound plays**.

#### **f) spawnOrb**

* spawnOrb(state) {
* state.orb = getRandomPosition();
* }
* Creates an orb at a random position.

#### **g) removeOrb**

* removeOrb(state) {
* state.orb = null;
* }
* Removes the orb from the board.

#### **h) collectOrb**

* collectOrb(state) {
* if (state.orb && state.orb.x === state.player.x && state.orb.y === state.player.y) {
* state.score += 10;
* state.orb = null;
* collectAudio.play();
* }
* }
* If the player is on the same cell as the orb:  
  + Adds 10 points to score
  + Removes the orb
  + Plays collection sound

#### **i) tick**

* tick(state) {
* if (!state.gameOver && state.gameStarted) {
* state.timeSurvived += 1;
* if (state.timeSurvived % 5 === 0) {
* state.difficulty += 1;
* }
* state.score += 1;

}  
}

* Called every second (or game tick):  
  + Increments survival time
  + Increases difficulty every 5 seconds
  + Adds 1 point per second survived

### **7. Exports**

* export const { startGame, resetGame, movePlayer, spawnEnemy, moveEnemy, spawnOrb, removeOrb, collectOrb, tick } = gameSlice.actions;
* export default gameSlice.reducer;
* Exports **action creators** (startGame(), movePlayer(), etc.) so you can dispatch them in React components.
* Exports the **reducer** to use in the Redux store.

✅ **Summary**:  
 This slice manages a **grid-based survival game**:

* Player moves around to collect orbs.
* Enemy chases the player after appearing.
* Score is based on survival time + collected orbs.
* Game over occurs on collision with enemy.
* Sounds play for collecting or losing.

## **3️⃣ components/HUD.jsx – Game Stats Display**

import React from "react";

import { useSelector } from "react-redux";

import "./styles.css";

export default function HUD(){

const { score, timeSurvived, difficulty } = useSelector(state => state.game);

return (

<div className="hud">

<div><strong>Score:</strong> {score}</div>

<div><strong>Time:</strong> {timeSurvived}s</div>

<div><strong>Difficulty:</strong> {difficulty}</div>

</div>

);

}

**Why:**

* Separates **HUD logic** from game board for modularity.

## **4️⃣ components/Modal.jsx – Reusable Modal**

import React from "react";

import "../styles.css";

export default function Modal({ title, children, onClose }){

return (

<div className="modal-overlay-white">

<div className="modal-content-white">

<button className="modal-close-x" onClick={onClose}>×</button>

<h1>{title}</h1>

{children}

</div>

</div>

);

}

* Reusable for **Game Over** or future messages.

## **5️⃣ redux/store.js – Redux Store**

import { configureStore } from "@reduxjs/toolkit";

import gameReducer from "./gameSlice";

export const store = configureStore({

reducer: { game: gameReducer },

});

* Provides the **Redux store** for the app.

## **6️⃣ index.js – App Entry Point**

import React from "react";

import ReactDOM from "react-dom/client";

import { Provider } from "react-redux";

import { store } from "./redux/store";

import App from "./App";

import "./index.css";

ReactDOM.createRoot(document.getElementById("root")).render(

<Provider store={store}><App /></Provider>

);

* Wraps the app with **Redux Provider**.

## **✅ Key Features Implemented**

1. Player movement with **keyboard**.
2. Enemy spawns and chases player.
3. Orbs spawn and can be collected.
4. Score, time, and difficulty tracking.
5. **Sound effects**: orb collect, game over.
6. **Animations**: player pulse, enemy shake, orb glow.
7. Modular, reusable **HUD** and **Modal** components.