Got the project files and code — thanks. Below is a concise, beginner-friendly plan plus ready-to-drop-in code to turn the current UI into a working Snake Xenzia game using the React Context API. I focused on clarity and small, safe changes so you can test quickly.

**Quick code issues to fix first**

1. Folder name Snake Xenzia contains a space — it works but is error-prone. I recommend renaming to SnakeXenzia or snake-xenzia.
2. grid-cols-13 (Tailwind) may not exist by default (Tailwind supports up to 12). Use inline gridTemplateColumns or add a config if you want a Tailwind utility.
3. main.jsx imports App but never uses it — safe but unused. Keep or remove later.

**What I will add (short)**

1. GameContext (React Context) to hold snake, direction, food, score, running, speed and controls (start/pause/reset).
2. Core game loop using setInterval (speed controlled).
3. Keyboard controls (Arrow keys + WASD).
4. Helper utils to convert id ↔ {row, col} ("a5" style IDs like in your current grid).
5. Small changes to SnakeInterface.jsx and SquareBlocks.jsx to read state from context and render snake/food/obstacles.

**Real-life example (how movement maps)**

If snake head is at "b5" and direction is RIGHT, new head becomes "b6" (column +1). If "b6" is an obstacle or part of the snake, game ends.

**Files to add / replace**

1. components/SnakeXenzia/GameContext.jsx (create this file)

import React, { createContext, useContext, useEffect, useRef, useState } from "react";

import { obstacleIds as defaultObstacles } from "./utils"; // adjust path if you renamed folder

const GameContext = createContext();

export const useGame = () => useContext(GameContext);

export const GameProvider = ({ children }) => {

const ROWS = Array.from({ length: 16 }, (\_, i) => String.fromCharCode(97 + i)); // a..p

const COLS = 13;

// create allIds once

const allIds = [];

for (let i = 0; i < ROWS.length; i++) {

for (let j = 1; j <= COLS; j++) {

allIds.push(ROWS[i] + j);

}

}

const idToCoord = (id) => {

const row = id[0];

const col = parseInt(id.slice(1), 10);

return { rowIndex: row.charCodeAt(0) - 97, colIndex: col - 1 };

};

const coordToId = (r, c) => String.fromCharCode(97 + r) + (c + 1);

const getRandomEmptyId = (blockedSet) => {

const empties = allIds.filter((id) => !blockedSet.has(id));

return empties.length ? empties[Math.floor(Math.random() \* empties.length)] : null;

};

// initial snake centered, length 3, heading RIGHT

const initialSnake = (() => {

const midR = Math.floor(ROWS.length / 2);

const midC = Math.floor(COLS / 2);

return [coordToId(midR, midC), coordToId(midR, midC - 1), coordToId(midR, midC - 2)];

})();

const [snake, setSnake] = useState(initialSnake);

const [direction, setDirection] = useState("RIGHT");

const [food, setFood] = useState(() => {

const blocked = new Set([...initialSnake, ...defaultObstacles]);

return getRandomEmptyId(blocked);

});

const [score, setScore] = useState(0);

const [running, setRunning] = useState(false);

const [speed, setSpeed] = useState(180); // ms per move

const timerRef = useRef(null);

const opposite = { LEFT: "RIGHT", RIGHT: "LEFT", UP: "DOWN", DOWN: "UP" };

const changeDirection = (newDir) => {

if (opposite[newDir] === direction) return; // ignore opposite

setDirection(newDir);

};

const step = () => {

setSnake((prev) => {

const head = prev[0];

const { rowIndex, colIndex } = idToCoord(head);

let r = rowIndex,

c = colIndex;

if (direction === "RIGHT") c++;

if (direction === "LEFT") c--;

if (direction === "UP") r--;

if (direction === "DOWN") r++;

// check bounds

if (r < 0 || r >= ROWS.length || c < 0 || c >= COLS) {

setRunning(false);

return prev;

}

const newId = coordToId(r, c);

// collision with obstacle or self

if (defaultObstacles.includes(newId) || prev.includes(newId)) {

setRunning(false);

return prev;

}

// eat food

if (newId === food) {

setScore((s) => s + 1);

const blocked = new Set([newId, ...prev, ...defaultObstacles]);

const nextFood = getRandomEmptyId(blocked);

setFood(nextFood);

return [newId, ...prev]; // grow

}

// normal move

return [newId, ...prev.slice(0, -1)];

});

};

useEffect(() => {

if (running) {

timerRef.current = setInterval(step, speed);

return () => clearInterval(timerRef.current);

} else {

if (timerRef.current) clearInterval(timerRef.current);

}

}, [running, speed, direction]); // step uses latest direction through closure

// keyboard controls

useEffect(() => {

const handler = (e) => {

const map = {

ArrowUp: "UP",

ArrowDown: "DOWN",

ArrowLeft: "LEFT",

ArrowRight: "RIGHT",

w: "UP",

s: "DOWN",

a: "LEFT",

d: "RIGHT",

W: "UP",

S: "DOWN",

A: "LEFT",

D: "RIGHT",

};

const dir = map[e.key];

if (dir) {

e.preventDefault();

changeDirection(dir);

}

};

window.addEventListener("keydown", handler);

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

}, [direction]);

const startGame = () => {

setSnake(initialSnake);

setDirection("RIGHT");

setScore(0);

const blocked = new Set([...initialSnake, ...defaultObstacles]);

setFood(getRandomEmptyId(blocked));

setRunning(true);

};

const pauseGame = () => setRunning(false);

const resetGame = () => {

setRunning(false);

setTimeout(startGame, 50);

};

return (

<GameContext.Provider

value={{

snake,

direction,

food,

score,

running,

speed,

setSpeed,

startGame,

pauseGame,

resetGame,

changeDirection,

obstacleIds: defaultObstacles,

allIds,

}}

>

{children}

</GameContext.Provider>

);

};

1. Replace components/SnakeXenzia/SnakeInterface.jsx with this simplified version (it now reads grid ids from context)

import React from "react";

import SquareBlocks from "./SquareBlocks";

import { useGame } from "./GameContext";

const SnakeInterface = () => {

const { allIds, score } = useGame();

return (

<div className="w-max">

<header className="text-center font-bold bg-green-400 text-white text-lg px-2">

Score: {score}

</header>

<div

className="grid gap-0"

style={{

gridTemplateColumns: "repeat(13, 30px)",

gridAutoRows: "30px",

}}

>

{allIds.map((id) => (

<SquareBlocks key={id} id={id} />

))}

</div>

</div>

);

};

export default SnakeInterface;

1. Replace components/SnakeXenzia/SquareBlocks.jsx so each cell decides its color from context

import React from "react";

import { useGame } from "./GameContext";

const SquareBlocks = ({ id }) => {

const { snake, food, obstacleIds } = useGame();

const isHead = snake[0] === id;

const isBody = snake.includes(id) && !isHead;

const isFood = food === id;

const isObstacle = obstacleIds.includes(id);

const base = "h-[30px] w-[30px] flex items-center justify-center border-[1px]";

let bg = "bg-blue-300";

if (isObstacle) bg = "bg-red-500";

else if (isFood) bg = "bg-yellow-400";

else if (isHead) bg = "bg-green-700";

else if (isBody) bg = "bg-green-500";

return (

<div id={id} className={`${base} ${bg}`}>

{/\* optional: show id for debugging \*/}

</div>

);

};

export default SquareBlocks;

1. Wrap the provider in layout/Root.jsx

import React from "react";

import SnakeXen from "../components/Snake Xenzia/SnakeXen"; // or adjusted path if renamed

import { GameProvider } from "../components/Snake Xenzia/GameContext";

const Root = () => {

return (

<GameProvider>

<SnakeXen />

</GameProvider>

);

};

export default Root;

**Small next steps (recommended)**

1. Add Start / Pause / New Game buttons in Sidebar.jsx that call startGame(), pauseGame(), resetGame() from useGame().
2. Save high score in localStorage when the game ends.
3. Add level/speed control (expose setSpeed in the sidebar).
4. Add sound toggles and mobile touch controls later.

If you want, I can now:

* (A) produce the exact modified files (full file contents) so you can copy-paste, **or**
* (B) implement Sidebar buttons and localStorage high-score saving next.

Which do you prefer? (If you want me to just continue without asking, I can implement all file replacements now.)