Game of Life - Entwicklerdoku

▼ lib

▼ gol.ts

Die gol.ts Datei definiert die grundlegenden Bausteine für die Implementierung des Game of Life:

- **PixelType** `Enumeration`: Definiert die möglichen Zustände eines Pixels (tot, NPC, Spieler).
- **GameOfLifeImplementation** `Interface`: Beschreibt die Methoden und Eigenschaften, die eine Implementierung des Game of Life bereitstellen muss, wie das Abfragen und Setzen von Zellen, das Ausführen von Zeitschritten und das Registrieren von Callbacks für Updates.
- **makeGolImpl** `Funktion`: Erstellt eine Instanz der SimpleGolImpl Klasse, die das Interface implementiert, und markiert sie als "roh" für Vue, um die Reaktivität zu umgehen.

▼ simple.ts

Die simple.ts Datei enthält die SimpleGolImpl Klasse, eine konkrete Implementierung des Game of Life:

- **Private Felder**: Speichern die Grenzen des Spielfelds und die Zustände der Zellen.
- **Konstruktor**: Initialisiert die Klasse.
- **mapCoordinate**: Wandelt Koordinaten in einen eindeutigen Schlüssel um.
- **getX und getY**: Extrahieren die X- und Y-Koordinaten aus dem eindeutigen Schlüssel.
- **getCell und setCell**: Methoden zum Abfragen und Setzen des Zustands einer Zelle.
- **tick**: Führt einen Zeitschritt aus, berechnet die neuen Zustände der Zellen und aktualisiert das Spielfeld.
- **Getter für minX, maxX, minY, maxY**: Zugriff auf die Spielfeldgrenzen.
- **useUpdated**: Registriert und entfernt Callbacks für Updates.
- **hasUpdated**: Ruft die Update-Callbacks auf.

Diese Implementierung ist erforderlich, um die Logik des Game of Life zu verwalten, einschließlich der Zustände der Zellen und der Spielregeln. Sie ermöglicht es der Anwendung, das Spiel zu simulieren und auf Benutzerinteraktionen zu reagieren.

Die Grundregeln von Conway's Game of Life werden in der SimpleGolImpl Klasse in der Datei simple.ts festgelegt, insbesondere in der tick Methode. Hier wird für jede Zelle überprüft, wie viele lebende Nachbarn sie hat, und basierend darauf wird entschieden, ob die Zelle im nächsten Zeitschritt lebt, stirbt oder neu belebt wird.
Die relevanten Codezeilen, die die Regeln implementieren, sind:

```
tick(timesteps: number): void {
    const start = performance.now();
    for (let i = 0; i < timesteps; i++) {
        let delayed: (() => void)[] = [];
        for (let x = this._minX - 1; x <= (this._maxX + 1); x++) {
            for (let y = this._minY - 1; y <= (this._maxY + 1); y++) {
                let liveNeighbors = 0;
                let hasNpc = false;
                // Überprüfe alle Nachbarzellen für die aktuelle Zelle (x, y)
                for (let xOffset = -1; xOffset <= 1; xOffset++) {
                    for (let yOffset = -1; yOffset <= 1; yOffset++) {
                        // Überspringe die aktuelle Zelle selbst
                        if (xOffset === 0 && yOffset === 0) {
                            continue;
                        }

                        // Hole den Typ der Nachbarzelle
                        const cellType = this.getCell(x + xOffset, y + yOffset);

                        // Zähle lebende Nachbarn und prüfe, ob eine Nachbarzelle ein NPC ist
                        if (cellType !== PixelType.Dead) {
                            liveNeighbors++;
                            hasNpc ||= cellType === PixelType.Npc;
                        }
                    }
                }

                // Hole den aktuellen Zustand der Zelle
                const currentState = this.getCell(x, y);
                // Entscheide, ob die Zelle im nächsten Schritt lebt oder tot ist
                const isAlive = currentState !== PixelType.Dead
                ? liveNeighbors >= 2 && liveNeighbors <= 3
                : liveNeighbors === 3;

                // Bestimme den neuen Zustand der Zelle basierend auf den Game of Life Regeln
                const newState = isAlive
                               ? ((hasNpc || currentState === PixelType.Npc)
                                ? PixelType.Npc
                                : PixelType.Player)
                               : PixelType.Dead;

                // Wenn sich der Zustand geändert hat, plane die Aktualisierung der Zelle
                if (newState !== currentState) {
                    const xCopy = x;
                    const yCopy = y;
                    delayed.push(() => this.setCell(xCopy, yCopy, newState, true));
                }
            }
        }
        // Führe alle geplanten Zellaktualisierungen aus
        for (const updateCell of delayed) {
            updateCell();
        }
    }
    const end = performance.now();
    const diff = end - start;
    console.log(`tick performance = ${diff}`);

    // Benachrichtige über die Aktualisierung des Spielfelds
    this.hasUpdated();
}
```

In diesem Codeabschnitt wird für jede Zelle gezählt, wie viele lebende Nachbarn sie hat. Dann wird basierend auf dem aktuellen Zustand der Zelle und der Anzahl der lebenden Nachbarn der neue Zustand der Zelle bestimmt:

- Eine lebende Zelle mit weniger als zwei lebenden Nachbarn stirbt (Unterbevölkerung).
- Eine lebende Zelle mit zwei oder drei lebenden Nachbarn überlebt.
- Eine lebende Zelle mit mehr als drei lebenden Nachbarn stirbt (Überbevölkerung).
- Eine tote Zelle mit genau drei lebenden Nachbarn wird belebt (Fortpflanzung).

Diese Logik entspricht den klassischen Regeln von Conway's Game of Life.

▼ zoom

▼ index.ts

Die index.ts Datei im lib/zoom Ordner stellt Funktionen bereit, um Zoom- und Gesteninteraktionen auf einem HTML-Element zu handhaben. Hier ist eine vereinfachte Erklärung der wichtigsten Teile des Codes:

Hilfsfunktionen

- midpoint: Berechnet den Mittelpunkt zwischen zwei Touch-Punkten.
- distance: Berechnet die Distanz zwischen zwei Touch-Punkten.
- angle: Berechnet den Winkel zwischen zwei Touch-Punkten.
- limit: Begrenzt einen Wert auf ein Maximum.
- normalizeWheel: Normalisiert das Mausrad-Event, um konsistente Delta-Werte zu erhalten.

Koordinatentransformation

- clientToHTMLElementCoords: Transformiert Bildschirmkoordinaten in Koordinaten relativ zu einem HTML-Element.
- clientToSVGElementCoords: Transformiert Bildschirmkoordinaten in Koordinaten relativ zu einem SVG-Element.

Hauptfunktion: twoFingers

Die twoFingers Funktion ist das Herzstück des Moduls. Sie ermöglicht es einem HTML-Element, Gesten wie Zoom und Pan mit zwei Fingern zu erkennen und darauf zu reagieren. Die Funktion nimmt ein HTML-Element und ein Objekt mit Callback-Funktionen (onGestureStart, onGestureChange, onGestureEnd) entgegen.

- wheelListener: Reagiert auf Mausrad-Events und löst die entsprechenden Gesten-Callbacks aus.
- startGesture: Startet eine neue Geste, wenn der Benutzer den Bildschirm berührt.
- touchMove: Aktualisiert die Geste, wenn der Benutzer den Bildschirm mit einem oder zwei Fingern bewegt.
- watchTouches: Überwacht Touch-Events und fügt oder entfernt Event-Listener basierend auf der Anzahl der Finger auf dem Bildschirm.
- startDrag: Ermöglicht Drag-Gesten mit der Maus.

Safari-spezifische Gestenbehandlung

Für Safari-Browser, die GestureEvent unterstützen, werden zusätzliche Event-Listener hinzugefügt, um Multi-Touch-Gesten wie Pinch-to-Zoom und Rotation zu behandeln.

Aufräumfunktion

Am Ende der twoFingers Funktion wird eine Funktion zurückgegeben, die verwendet werden kann, um alle Event-Listener zu entfernen. Dies ist nützlich, um Speicherlecks zu vermeiden, wenn das Element entfernt wird oder die Gesteninteraktion nicht mehr benötigt wird.

Zusammengefasst bietet index.ts eine API, um Gesteninteraktionen wie Zoom und Pan auf einem Element zu ermöglichen, wobei sowohl Touch- als auch Maus-Events unterstützt werden.

▼ types.ts

Die types.ts Datei im lib/zoom Ordner definiert TypeScript Typen und Schnittstellen, die für die Handhabung von Gesten und Zoom-Interaktionen verwendet werden. Hier ist eine Erklärung der einzelnen Typen:

Coords

```
export type Coords = {
  x: number;
  y: number;
};
```

Coords ist ein einfacher Typ, der ein Koordinatenpaar mit einer x und einer y Komponente repräsentiert. Dieser Typ wird verwendet, um Positionen und Bewegungen auf dem Bildschirm zu beschreiben.

Gesture

```
export type Gesture = {
  origin: Coords;
  translation: Coords;
  scale: number;
  rotation?: number;
};
```

Gesture repräsentiert eine Geste, die vom Benutzer ausgeführt wird. Es enthält den Ursprung der Geste , die Verschiebung , den Skalierungsfaktor und optional eine Rotation .

GestureCallbacks

```
export type GestureCallbacks = {
  onGestureStart?: (gesture: Gesture) => void;
  onGestureChange?: (gesture: Gesture) => void;
  onGestureEnd?: (gesture: Gesture) => void;
};
```

GestureCallbacks ist ein Typ, der Funktionen für die verschiedenen Phasen einer Geste definiert: Start 5, Änderung 6 und Ende 7. Diese Funktionen werden aufgerufen, um die entsprechenden Ereignisse zu behandeln.

NormalizedWheelEvent

```
export type NormalizedWheelEvent = {
  dx: number;
  dy: number;
};
```

NormalizedWheelEvent ist ein Typ, der die normalisierten Delta-Werte eines Mausrad-Events (dx für horizontale Bewegung, dy für vertikale Bewegung) enthält.

Globale Erweiterungen

Die Datei erweitert auch einige globale Interfaces, um die Verwendung von nicht standardisierten Eigenschaften zu ermöglichen:

- TouchEvent: Erweitert um scale und rotation, die in manchen Browsern verfügbar sind, aber nicht standardisiert sind.
- GestureEvent: Definiert ein Interface für Gesten-Events, die in Safari verfügbar sind, aber nicht standardisiert sind. Es enthält zusätzliche Eigenschaften wie altKey, ctrlKey, metaKey, shiftKey, scale, rotation, clientX, clientY, screenX und screenY.
- ElementEventMap: Erweitert um gesturestart, gesturechange und gestureend Events.
- Window: Fügt eine optionale GestureEvent Eigenschaft hinzu, um die Existenz von Gesten-Events zu prüfen.

Diese Typdefinitionen sind entscheidend für die Typsicherheit und die korrekte Handhabung von Gesten-Events in der Anwendung. Sie ermöglichen es den Entwicklern, die Funktionen und Callbacks zu implementieren, die für die Interaktion mit Touch- und Gesten-basierten Eingaben erforderlich sind.

▼ App.vue

Die App.vue Datei ist die Hauptkomponente der Vue-Anwendung und dient als Einstiegspunkt. Sie enthält:

- **Script-Setup Bereich**: Importiert die GolPlayer Komponente und die Funktionen aus gol.ts. Initialisiert das Game of Life mit einer bestimmten Konfiguration.
- **Template-Bereich**: Definiert das HTML-Markup der Anwendung. Es gibt zwei Hauptteile:
- Eine div mit der Klasse main-div, die die GolPlayer Komponente enthält und ihr die impl Instanz übergibt.
- Eine offcanvas Sidebar, die Informationen und Regeln zu Noel's Game of Life anzeigt.
- **Style-Bereich**: Definiert CSS-Stile, um die main-div über den gesamten Bildschirm zu strecken.

Die App.vue Komponente ist notwendig, um die Anwendung zu strukturieren, die GolPlayer Komponente einzubinden und dem Benutzer eine Benutzeroberfläche mit zusätzlichen Informationen zum Spiel zu bieten.

▼ Components

Der components Ordner in deiner Codebase ist notwendig, um die Vue-Komponenten zu organisieren, die wiederverwendbare und unabhängige Einheiten der Benutzeroberfläche darstellen. Diese Strukturierung hilft dabei, die Anwendung modular und wartbar zu halten.

Funktionen der Dateien GolCanvas.vue und GolPlayer.vue:

▼ GolCanvas.vue

- **Darstellung des Spielfelds**: Nutzt ein HTML5 Canvas, um das Game of Life zu zeichnen.
- **Interaktion**: Erlaubt Benutzerinteraktionen wie Klicken auf das Canvas, um Zellen zu verändern.
- **Zoom und Pan**: Integriert die twoFingers Funktion aus lib/zoom, um Zoom- und Pan-Gesten zu ermöglichen.
- **Transformation**: Berechnet die Transformation von Bildschirmkoordinaten zu Spielkoordinaten, um korrekte Interaktionen zu gewährleisten.

▼ GolPlayer.vue

- **Benutzeroberfläche**: Stellt Steuerelemente zur Verfügung, um das Spiel zu steuern (z.B. Geschwindigkeit anpassen, Spiel pausieren/fortsetzen).
- **Integration von GolCanvas**: Bindet die GolCanvas Komponente ein und reagiert auf deren Ereignisse.
- **Spielsteuerung**: Verwaltet den Spielzustand, wie die Geschwindigkeit und ob das Spiel pausiert ist.

Beide Komponenten arbeiten zusammen, um die Benutzererfahrung des Game of Life zu ermöglichen, wobei GolCanvas für die Darstellung und GolPlayer für die Steuerung zuständig ist.