```
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import userEvent from '@testing-library/user-event';
import MinesweeperPage from './MinesweeperPage';
import * as gridUtils from '@/_util/grid';

// Mock the grid utility functions
jest.mock('@/_util/grid', () => ({
 createEmptyBoard: jest.fn(),
 placeMines: jest.fn(),
 cloneBoard: jest.fn(),
 floodFill: jest.fn(),
 computeAdjacency: jest.fn(),
}));

// Mock the RenderGrid component
jest.mock('./RenderGrid', () => {
 return function MockRenderGrid({ board, gridSize, reveal, flag }: any) {
   return (
     <div data-testid="mock-grid">
       {board.map((row: any, r: number) =>
         row.map((cell: any, c: number) => (
           <div
             key={`${r}-${c}`}
             data-testid={`cell-${r}-${c}`}
             onClick={() => reveal(r, c)}
             onContextMenu={(e: any) => flag(e, r, c)}
           >
             {cell.revealed ? (cell.isMine ? '💣' : cell.adjacentMines) : cell.flagged
? '🚩' : '□'}
           </div>
         ))
       )}
     </div>
   );
 };
});

describe('MinesweeperPage', () => {
 const GRID_SIZE = 10;
 const mockEmptyBoard: any = Array(GRID_SIZE)
   .fill(null)
   .map(() =>
```

```
    Array(GRID_SIZE).fill(null).map(() => ({
      isMine: false,
      revealed: false,
      flagged: false,
      adjacentMines: 0,
    }))
  );

beforeEach(() => {
  jest.clearAllMocks();
  (gridUtils.createEmptyBoard as jest.Mock).mockReturnValue(mockEmptyBoard);
  (gridUtils.cloneBoard as jest.Mock).mockImplementation((board) => board); //
Simplified for testing
  (gridUtils.placeMines as jest.Mock).mockImplementation((board) => board);
  (gridUtils.computeAdjacency as jest.Mock).mockImplementation((board) => board);
  (gridUtils.floodFill as jest.Mock).mockImplementation((board) => board);
});

// Test rendering and initial state
test('renders correctly with initial state', () => {
  render(<MinesweeperPage />);
  expect(screen.getByLabelText(/mines/i)).toHaveValue(15);
  expect(screen.getByText('Reset')).toBeInTheDocument();
  expect(screen.getByText('⏱ 0s')).toBeInTheDocument();
  expect(screen.getByText('🚩 15')).toBeInTheDocument();
  expect(screen.getByTestId('mock-grid')).toBeInTheDocument();
});

// Test reset functionality
test('reset button resets the game state', async () => {
  render(<MinesweeperPage />);
  const resetButton = screen.getByText('Reset');
  fireEvent.click(resetButton);
  expect(gridUtils.createEmptyBoard).toHaveBeenCalledWith(GRID_SIZE, GRID_SIZE);
  expect(screen.getByText('⏱ 0s')).toBeInTheDocument();
  expect(screen.getByText('🚩 15')).toBeInTheDocument();
});

// Test mine count input
test('updates mine count within valid range', async () => {
  render(<MinesweeperPage />);
  const mineInput = screen.getByLabelText(/mines/i);
```

```
  await userEvent.clear(mineInput);
  await userEvent.type(mineInput, '5');
  expect(mineInput).toHaveValue(5);
  expect(screen.getByText('🚩 5')).toBeInTheDocument();

  // Test max limit (20% of 10x10 = 20)
  await userEvent.clear(mineInput);
  await userEvent.type(mineInput, '25');
  expect(mineInput).toHaveValue(20);

  // Test min limit
  await userEvent.clear(mineInput);
  await userEvent.type(mineInput, '0');
  expect(mineInput).toHaveValue(1);
});

// Test revealCell - first click
test('revealCell on first click places mines and starts game', async () => {
  render(<MinesweeperPage />);
  const cell = screen.getByTestId('cell-0-0');
  fireEvent.click(cell);
  expect(gridUtils.placeMines).toHaveBeenCalledWith(
    expect.any(Array),
    15,
    { r: 0, c: 0 }
  );
  expect(gridUtils.computeAdjacency).toHaveBeenCalled();
  expect(gridUtils.floodFill).toHaveBeenCalledWith(
    expect.any(Array),
    GRID_SIZE,
    0,
    0
  );
  expect(screen.getByText('⏱ 0s')).toBeInTheDocument(); // Timer starts
});

// Test revealCell - hitting a mine
test('revealCell on mine ends game', async () => {
  const mockBoardWithMine = [...mockEmptyBoard];
  mockBoardWithMine[0][0] = { ...mockBoardWithMine[0][0], isMine: true };
  (gridUtils.createEmptyBoard as jest.Mock).mockReturnValue(mockBoardWithMine);
  render(<MinesweeperPage />);
```

```
  const cell = screen.getByTestId('cell-0-0');
  fireEvent.click(cell);
  await waitFor(() => {
    expect(screen.getByText('💣')).toBeInTheDocument(); // Mine revealed
  });
});

// Test toggleFlag
test('toggleFlag adds and removes flags correctly', async () => {
  render(<MinesweeperPage />);
  const cell = screen.getByTestId('cell-0-0');
  fireEvent.contextMenu(cell); // Right-click to flag
  await waitFor(() => {
    expect(screen.getByText('🚩')).toBeInTheDocument();
    expect(screen.getByText('🚩 14')).toBeInTheDocument(); // Flags left decremented
  });
  fireEvent.contextMenu(cell); // Remove flag
  await waitFor(() => {
    expect(screen.getByText('□')).toBeInTheDocument();
    expect(screen.getByText('🚩 15')).toBeInTheDocument(); // Flags left restored
  });
});

// Test checkWin
test('checkWin returns true when all non-mine cells are revealed', () => {
  const mockBoard = [...mockEmptyBoard];
  mockBoard[0][0] = { ...mockBoard[0][0], isMine: true, revealed: false };
  mockBoard[0][1] = { ...mockBoard[0][1], revealed: true };
  mockBoard[1][0] = { ...mockBoard[1][0], revealed: true };
  render(<MinesweeperPage />);
  const { container } = render(<MinesweeperPage />);
  const component = container.firstChild as any;
  const checkWin = component.__proto__.checkWin;
  expect(checkWin(mockBoard)).toBe(false); // Not all non-mine cells revealed
  mockBoard.forEach((row: any) =>
    row.forEach((cell: any) => {
      if (!cell.isMine) cell.revealed = true;
    })
  );
  expect(checkWin(mockBoard)).toBe(true); // All non-mine cells revealed
});
```

```
// Test revealMines
test('revealMines reveals all mines', () => {
  const mockBoard = [...mockEmptyBoard];
  mockBoard[0][0] = { ...mockBoard[0][0], isMine: true, revealed: false };
  mockBoard[1][1] = { ...mockBoard[1][1], isMine: true, revealed: false };
  render(<MinesweeperPage />);
  const { container } = render(<MinesweeperPage />);
  const component = container.firstChild as any;
  const revealMines = component.__proto__.revealMines;
  revealMines.call({ setBoard: jest.fn() }, mockBoard);
  expect(mockBoard[0][0].revealed).toBe(true);
  expect(mockBoard[1][1].revealed).toBe(true);
  expect(mockBoard[0][1].revealed).toBe(false); // Non-mine cell unchanged
});

// Test timer
test('timer increments when game is started', async () => {
  jest.useFakeTimers();
  render(<MinesweeperPage />);
  const cell = screen.getByTestId('cell-0-0');
  fireEvent.click(cell); // Start game
  expect(screen.getByText('⏱ 0s')).toBeInTheDocument();
  jest.advanceTimersByTime(1000);
  await waitFor(() => {
    expect(screen.getByText('⏱ 1s')).toBeInTheDocument();
  });
  jest.useRealTimers();
});

// Test edge case: no flags left
test('cannot flag a cell when no flags are left', async () => {
  render(<MinesweeperPage />);
  const cell1 = screen.getByTestId('cell-0-0');
  const cell2 = screen.getByTestId('cell-0-1');
  // Use all flags (15 mines)
  for (let i = 0; i < 15; i++) {
    fireEvent.contextMenu(cell1);
  }
  await waitFor(() => {
    expect(screen.getByText('🚩 0')).toBeInTheDocument();
  });
  fireEvent.contextMenu(cell2); // Try to flag another cell
```

```
  await waitFor(() => {
    expect(screen.getByTestId('cell-0-1')).toHaveTextContent('□'); // No flag added
  });
});

// Test win condition
test('game is won when all mines are flagged and non-mines revealed', async () => {
  const mockBoard = [...mockEmptyBoard];
  mockBoard[0][0] = { ...mockBoard[0][0], isMine: true, flagged: true };
  mockBoard.forEach((row: any) =>
    row.forEach((cell: any) => {
      if (!cell.isMine) cell.revealed = true;
    })
  );
  (gridUtils.createEmptyBoard as jest.Mock).mockReturnValue(mockBoard);
  render(<MinesweeperPage />);
  const cell = screen.getByTestId('cell-0-0');
  fireEvent.contextMenu(cell); // Trigger win check
  await waitFor(() => {
    expect(screen.getByText('💣')).toBeInTheDocument(); // Mines revealed on win
  });
});
});
```

Rendering and Initial State:

- Verifies that the component renders with the correct initial state (15 mines, 0 seconds, 15 flags, and the grid).

Reset Button:

- Tests that clicking the reset button resets the board, timer, and flags.

Mine Count Input:

- Ensures the mine count input updates correctly and respects min (1) and max (20% of grid) limits.

Reveal Cell (First Click):

- Tests that the first click places mines, computes adjacency, starts the timer, and reveals cells via flood fill.

Reveal Cell (Hitting a Mine):

- Simulates clicking a mine to ensure the game ends and mines are revealed.

Toggle Flag:

- Tests adding and removing flags, ensuring the flag count updates correctly.

Check Win:

- Verifies the checkWin function correctly identifies a win when all non-mine cells are revealed.

Reveal Mines:

- Ensures revealMines reveals only mine cells.

Timer:

- Tests that the timer increments every second after the game starts.

No Flags Left:

- Verifies that flagging is prevented when no flags remain.

Win Condition:

- Tests the win condition when all mines are flagged and all non-mine cells are revealed.