**Introduction**

This document provides an overview of the Snake game developed using Vue.js framework, and TypeScript. The game has a grid of cells, where a snake moves and eats food randomly placed on the grid. The snake grows longer as it eats more food, and the game ends if the snake collides with the wall or itself.

**Components**

The game consists of the following components:

* page: The main component that contains the navigation bar, game grid, and a start button.
* navbar: The navigation bar that displays the current score and highest record.
* snake-container: The container that holds the grid of cells.
* snake-row: A row in the grid.
* snake-cell: A cell in the grid.
* btn-start: The button to start the game.

**Variables**

The following variables are used in the game:

* playing: A Boolean value to indicate if the game is in progress.
* speed: The speed at which the snake moves.
* size: The size of the game grid.
* windowWidth: The width of the browser window.
* windowHeight: The height of the browser window.
* cssSize: A computed property that sets the size of the cells in the grid based on the window size and game grid size.
* record: The highest score achieved in the game.
* score: The current score in the game.
* d: The direction in which the snake is moving.
* snake: An array that holds the coordinates of the snake's body.
* food: The coordinates of the food randomly placed on the grid.

**Functions**

The game uses the following functions:

* getRandomCoordinates(max: number): Returns random x and y coordinates within the game grid size.
* newFood(snake: Snake, size: number): Coordinate: Generates new coordinates for the food randomly placed on the grid, ensuring it is not placed on the snake.
* newSnake(size: number): Snake: Generates an initial array of coordinates for the snake's body at the center of the game grid.
* isCollision(c: Coordinate, size: number): Checks if the coordinate is outside the game grid.
* isCollisionWithItself(coordinate: Coordinate, snake: Snake): Checks if the coordinate is inside the snake and not the head or tail.
* isGoingBack(): Checks if the snake is going back on itself.
* startGame(): Starts the game.
* endGame(): Ends the game.
* play(): The main loop of the game, which updates the game grid, snake's position, and checks for collisions.
* isSnake(x: number, y: number): Checks if the given coordinates are part of the snake's body.
* isFood(x: number, y: number): Checks if the given coordinates are the food location.

**Usage**

The game can be started by clicking on the Start button, which calls the startGame() function. The snake can be controlled by using the arrow keys, which update the direction in which the snake moves. The game ends when the snake collides with the wall or itself, and the highest score achieved is saved as the record. The record can be reset by refreshing the page.

**Conclusion**

The Snake game is a simple yet addictive game that demonstrates the use of Vue.js and TypeScript in web development. With its intuitive user interface and engaging gameplay, the Snake game is a fun way to pass the time and improve hand-eye coordination.

**Improvments**  
One of the problems with Snake is that if a player makes multiple moves in a single frame, only the first valid move is counted and all other moves made during the same frame are lost. This can lead to frustration for the player, as he may expect his inputs to be fully recorded and accounted for in the snake's movement.

This problem will probably be solved in the future by efficiently composing all the moves from the previous and current turn.

Overall, by improving the processing of user inputs during gameplay, we can improve the player experience and make Snake more enjoyable and satisfying.