Bomberman Developer Guide

Public

This section defines frequently used functions that are shared across the modules. These include the following:

`get-symbol:` Retrieves the symbol at a given coordinate in the layout.

`convert:` Updates the layout by replacing the symbol at a given coordinate or a list of coordinates with a new symbol.

`in-bound?:` Checks whether a given coordinate is within the valid bounds of the layout.

Render

This section explains the rendering logic, including how the game layout, players, and other visual components are displayed on the screen.

Keyhandler

This section discusses the handling of user inputs (keyboard events), which affect player movement, bomb placement, and other game actions.

Timehandler

The timehandler function is the core logic for the on-tick process, responsible for updating the game state during each tick. It first calls the updated-layout and updated-bomb functions to update the game layout and the countdown of bombs, while also handling the round timer and the maximum number of bombs. If any bomb's countdown reaches 0, it invokes the boom function to handle the chain explosion effect. Within boom, the single-boom-range function calculates the explosion range of a single bomb, and extend-direction is called to extend the explosion range in each direction. The player and layout states are then updated accordingly. Auxiliary functions like check-I? and check-D? are used to determine whether a specific cell is an indestructible or destructible wall, while chain-explosion processes chain reactions and updates the bomb list. Through the close cooperation of these functions, timehandler dynamically updates all critical game states during each tick.

Big-bang

This section describes the main game loop, which integrates all modules using the `big-bang` framework. It initializes the game state, handles user inputs, updates the game state on each tick, and renders the game layout.

The `big-bang` function ties together the following components: - `on-tick:` Calls the `timehandler` function to update the game state dynamically at each tick. - `on-key:` Invokes the keyhandler functions to process user inputs, such as player movements and bomb placements. - `to-draw:` Uses render functions to display the current game layout, including players, bombs, and other elements. - `stop-when:` Determines when the game should end by calling the `end?` function. - `final:` Displays the appropriate endgame screen when the game ends.

This structure ensures smooth integration of all modules and provides a responsive gameplay experience.

Stop-when

This section describes the logic for determining when the game should stop and how the final game state is displayed. It consists of the following key functions:

`end?:` This function checks whether the game should stop based on the current game state. The game ends if any of the following conditions are met: - The player has chosen to quit the game. - The round timer reaches 0. - Both players have died. - Player 1 has died. - Player 2 has died. The function returns `#t` if any of these conditions are true and `#f` otherwise.

`final:` Once the game ends, this function determines the final image to display based on the game state. - If the player has chosen to quit, it returns `quit-image`. - If both players have died or the round timer has finished, it returns `tie`. - If only Player 1 has died, it returns `player2-win`. - If only Player 2 has died, it returns `player1-win`. This ensures that the appropriate message is displayed to reflect the game's outcome.

Auxiliary functions such as `check-player1-died?`, `check-player2-died?`, and `check-all-died?` are used by `end?` to evaluate the players' statuses.

Additionally, helper functions like `player1-win`, `player2-win`, and `tie` are invoked by `final` to generate specific endgame images based on the result.