Bomberman Developer Guide

Public

This part contains data definitions, generate-layout function and its helper functions, some constant definitions and some public functions such as convert, get-symbol,in-bound?.

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 serves as the core logic for the on-tick process, responsible for updating the game state at 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 by invoking **check-roundtimer?** and **add-maximum**. If any bomb's countdown reaches zero, it calls the **boom** function to handle chain explosion effects. Within **boom**, the **single-boom-range** function calculates the explosion range of a single bomb, and **extend-direction** extends the explosion range in all directions. Subsequently, the player and layout states are updated accordingly. Auxiliary functions such as **check-I?** and **check-D?** are used to determine whether a specific cell is an indestructible or destructible wall, while **chain-explosion** handles chain reactions and updates the bomb list. After processing in **boom** is complete, the **remove-bomb** function removes bombs that have already exploded. Through the close coordination of

these functions, **timehandler** dynamically updates all critical game states at each tick.

Big-bang

The **big-bang** function ties together the following components:

- **on-tick:** Calls the **timehandler** function to update the gamestate 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.

Stop-when

This part 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.