s

Bomberman Game README

Team Members

- Elena Hu
- Chengyu Yu

Changes

Gamestate Field Modification

Removed the boom-cor field from the gamestate structure. This change was made after realizing that handling the countdown (from 2 to 0) for bomb explosions directly within the layout field simplifies the implementation. Retaining boom-cor to pass explosion ranges introduced bugs, which were resolved by this adjustment.

Additionally, since the explosion effects are now directly updated in the layout, we adjusted the bomb countdown settings to start at 3 seconds and explode at 0 seconds, instead of the previous setup starting at 5 seconds, exploding at 2 seconds, and ending at 0 seconds.

Generating Layout

We use a rule function as the input of the generating-layout function, so that we can use just one function to generate both homepage-layout and random-layout.

We also aimed to make the game map size scalable by implementing the following code snippet:

```
1 (define SCALE Integer)
2 (define MAX-ROWS (* 11 SCALE))
3 (define MAX-COLS (* 15 SCALE))
```

When SCALE is set to 1, the game runs with the default 11x15 map size. By adjusting SCALE, we can expand the map while maintaining the same generation rules: fixed snowman positions, fixed safe zones in the top-left and top-right corners, and fixed player starting positions. However, we found that setting SCALE to 2 caused noticeable lag during gameplay. As a result, we ultimately decided to abandon this idea.

Modification of single-boom-range and extend-direction function

Initially, our ‘single-boom-range‘ and ‘extend-direction‘ functions were designed as follows:

```
[potential-boom-range-except-center
 (list
  ;; up
  (list (make-cor cor-x (- cor-y 1))
        (make-cor cor-x (- cor-y 2)))
  ;; down
  (list (make-cor cor-x (+ cor-y 1))
        (make-cor cor-x (+ cor-y 2)))
  ;; left
  (list (make-cor (- cor-x 1) cor-y)
        (make-cor (- cor-x 2) cor-y))
  ;; right
  (list (make-cor (+ cor-x 1) cor-y)
        (make-cor (+ cor-x 2) cor-y)))]

[boom-range-except-center
 (map
  (lambda (potential-list)
    (extend-direction potential-list layout))
  potential-boom-range-except-center)]
```

Since we aimed to achieve better scalability and avoid hardcoding (in this version, we had to manually list all potential coordinates in potential-list), we devised an alternative implementation:

```
[directions
 (list
  (lambda (n) (make-cor (cor-column center) (- (cor-row center) n)))
  (lambda (n) (make-cor (cor-column center) (+ (cor-row center) n)))
  (lambda (n) (make-cor (- (cor-column center) n) (cor-row center)))
  (lambda (n) (make-cor (+ (cor-column center) n) (cor-row center))))]

[boom-range-except-center
 (map
  (lambda (direction-fn)
    (extend-direction direction-fn layout))
  directions)]

(cons center
 (apply append boom-range-except-center))
```

However, this implementation turned out to be much less readable. Ultimately, we decided to prioritize readability over scalability and chose to use the initial version as a balance between the two.

Chain-explosion

We identified an issue in the previous implementation of the boom function, which failed to recursively compute chained explosions correctly. The function was rewritten to ensure that all chained explosions are properly handled. For example, a bomb at coordinate (0,0) cannot directly trigger a bomb at (0,4), but it can trigger a bomb at (0,2), which subsequently triggers the bomb at (0,4). The new implementation ensures that all affected bombs explode simultaneously through recursive propagation.

Updated Symbol System and modification in get-symbol function

The symbol system was updated to provide a clearer representation of the game state:

- Single-letter symbols represent cells without players. For example:
  - 'W: Walkable cell.
  - 'I: Indestructible wall.
  - 'D: Destructible wall.
- Three-letter symbols represent cells occupied by players. The rules are:
  - The first letter indicates the cell type (e.g., W, B for bombs).
  - The second letter indicates the player (1 for Player 1, 2 for Player 2).
  - The third letter indicates the player's current direction (U, D, L, R).
  - Example: 'W1L means Player 1 is on a walkable cell facing left.
- Two-letter symbols always start with E and represent the explosion effect's countdown timer.
  - Example: 'E2 indicates the explosion effect will last for 2 more ticks.

Since the keyhandler functions need to retrieve the new symbol, we added a new symbol 'ILEGAL to represent the symbol out of bounds. As a result, the get-symbol function was also modified.

Optimization of Minor Issues

We also addressed several minor issues, including the following:

1. The implementation logic of the end? function was simplified. Since end? returns a Boolean, there is no need to use ‘cond‘ to specify the return value for each case. Instead, we used ‘or‘ for a more concise implementation.

2. Improved the usage of let and let*. We ensured that let* is only used when necessary and avoided its use in situations where ‘let‘ suffices.

3. After updating the symbol system, we had to use the following code to handle certain logic (e.g., movement logic):

```
1 (string->symbol (string-append ...))
```

A better approach would have been to represent the layout as a vector of vectors of strings, thereby avoiding the need to convert between strings and symbols. However, considering that this issue had minimal impact, we ultimately decided not to make this change.

4. Split the previously monolithic file into separate modules:

- on-tick
- on-key
- render
- stop-when
- public(containing shared functions like get-symbol, convert, in-bound?, and complete data definitions)
- big-bang

This modularization improved the maintainability of the code and enhanced its future scalability.