

Bomberman Game Discussion

requirement in the instruction.pdf

1. An overview of the project topic and of the group members, mentioning the main tasks that each of them primarily worked on during project development

2. A brief demonstration of what your project does (that is, you run the project to show how it's used: bring your laptop)

3. The main data types used in the project's implementation

4. How the main project functionalities of the project are implemented and by what functions

5. What kinds of tests you wrote and which functions you tested more thoroughly

6. Whether the project went through several revisions or was designed develop- ing the same idea from the start

part1:main tasks that each of them primarily worked on during project development

Elena Hu worked for the public,render,on-key,test2(render and key-handler test)

Chengyu Yu worked for the public,on-tick,stop-when,test(stop-when,public,on-tick test)

My friend designed the decorations part for us(If TAS ask this question)

part2

Bring computer and play game for them



part4

See them in code and also prepare some questions for each other and talk about them tomorrow



part3 and part 6:Gamestate Field Modification

compare the gamestate structure

Previous gamestate:

```
1 (define-struct gamestate [layout bomb player1 player2 roundtimer maximum
   quit?])
2 layout is a vector of vector of symbol but just have 1-length legal
   symbol
3 bomb is a list of bombstate,and bombstate is:(define-struct bombstate
   [cor countdown])
4 each player: (define-struct player1 [posn dead? bombcount])
```

Milestone gamestate:

```
1 (define-struct gamestate [layout bomb player1 player2 roundtimer maximum
   quit? boom-cor])
2 Change compared to Previous gamestate:
3 Add a boom-cor field
4 Add three-length legal symbol in the layout
5 Modify each player:(define-struct player1[cor direction])
6 Bombstate:(define-struct bombstate [cor countdown owner])
```

Submit gamestate:

```
1 (define-struct gamestate [layout bomb player1 player2 roundtimer maximum
   quit?])
2 Change compared to Milestone gamestate:
3 Delete boom-cor field
4 Add two-length legal symbol 'E1 'E0 in the layout
```

Why we add direction field of each player?

Since we wanna to make sure player can show different image when they move.

Why we use cor field to replace the posn field of each player?

At first , we wanna use Posn(floating numbers) to represents the location of each player and use Cor(Integer) to represents each cell.

So we design a simple convert function:

```
1 (define (nearest-cor posn)
2   (make-cor
3     (floor (/ (posn-x posn) CELL-SIZE))
4     (floor (/ (posn-y posn) CELL-SIZE))))
```

Obviously convert Posn to Cor we need use floor function to convert the floating Number to Integer, so it caused a lot of problems such as:

- 1.Move-predicator and put-bomb-predicator
- 2.Check-die functions

At that time we have two choice:

1. Giving up the first idea of Cor (it simplifies a lot of logic in our game) and use Posn to represents all the locations in this game
2. Use Cor to represent the location of player

At last, we choose the second choice.after that ,the keyhandler part and stop-when part are very easy to achieve.

The related video is named as Posn in backup.

Why we delete died? field of each player field

The reason for designing died? field of each player is serving for the stop-when part of this game before we change the Posn field to Cor of each player.

When the nearest-cor function's ouput Cor is in the range of 'E1 or 'E0, the player will die.

However, since the change of Posn to Cor, we update the symbol system.So that the died player will be represented directly in the layout such as:

-'E1R(E+who+its direction)

So that we just need to use get-symbol function to get the symbol of each player's Cor in the layout to know their state.

Why we Add a boom-cor field of the gamestate in the milestone but delete it in the final submit?

This reason is related to a issue in the timehandler part, and it is also the reason of changing the range of countdown of each Bombstate Structure.

```

1 ;countdown:
2 ;represents the countdown of the each bombstate
3 ;5 is the time for bomb just added to game
4 ;2 is the time for bomb start to explode
5 ;0 is the time for bomb end explode
6
7 (define (timehandler gamestate)
8   (let*
9     (
10      [updated-gamestate
11        (make-gamestate (gamestate-layout gamestate)
12                          (updated-bomb (gamestate-bomb gamestate))
13                          (gamestate-player1 gamestate)
14                          (gamestate-player2 gamestate)
15                          (updated-roundtimer (gamestate-roundtimer
16                                                gamestate))
17                          (if (check-roundtimer? (gamestate-roundtimer
18                                                    gamestate))
19                              (add-maximum
20                                (gamestate-maximum
21                                  gamestate))
22                                (gamestate-maximum
23                                  gamestate)))]
24
25      ;;;;;;This part is responsible for updating all the
26      ;countdown(roundtimer,whether add maximum,countdown of
27      ;each Bombstate);;;;;;
28
29      [boom-end-state
30        (if (check-zero? (gamestate-bomb updated-gamestate))
31            (boom-end updated-gamestate)
32            updated-gamestate)]
33
34      ;;;;;;This part is responsible for handling the
35      ;countdown=0 bomb (update layout from 'E->'W,remove
36      ;exploded bomb)
37
38      [bomb-boom-state
39        (if (check-two? (gamestate-bomb boom-end-state))
40            (boom boom-end-state)
41            boom-end-state)]
42
43      ;;;;;;This part is responsible for handling the
44      ;countdown=2 bomb (convert the cor in boom-range in layout
45      ;to 'E, kill player,similar to submit version)
46
47      )
48      bomb-boom-state))

```

In this version: We need to calculate the total boom-range in the boom-function, and it is the real boom-range. But boom-end function cannot get it,so we have to calculate it again in boom-end function to make sure which range needed to be converted to 'W.

However, this method make the code so long and complex.and it also cause a bug:

Since one of the boom-range rule is: the explosion of the bomb could dismental the tree('D) but it should stop extending in this direction.However,since the 'D has been converted in the boom function, So in the boom-end function when we recalculate the boom-range,it will regard it as 'E,so in fact ,it will still extend in this direction to make the next 'D to 'W.

(This can be seen in the video named as "Double calculating boom-range")

To solve this issue, my first idea is to use some trick so that boom-function can pass the boom-range to the boom-end-function, so we add a boom-cor field (a list of Cor) the gamestate so that boom-end-function can reuse the boom-range that boom-function calculated before.

But this bring another issue:

-1.Some 'E cannot disappear(This can be seen in the vedeo named as "Passing boom-range").

-2.In fact, in almost all the bomberman games, we wanna see that after bomb boom, the player's bombcount can be removed after the bomb-boom but not boom-end.

And inspired by the issue 2. (I guess the boom and boom-end is divided to handle in bomberman games in the markets).We decided to divide the logic of boom and boom-end(in fact its just the convert of E to W),so we rewrite the countdown of Bombstate setting and delete the boom-cor field.and add two functions to handle the convert of E to W in the layout.

```
1 ;countdown:
2 ;represents the countdown of the each bombstate
3 ;3 is the time for bomb just added to game
4 ;0 is the time for bomb start to boom
5
6 (define (update-E-symbols symbol)
7   (cond
8     [(equal? symbol 'E1) 'E0]
9     [(equal? symbol 'E0) 'W]
10    [else symbol]))
11
12 (define (updated-layout layout)
13   (vector-map
```

```
14 | (lambda (row)
15 |   (vector-map update-E-symbols row))
16 | layout))
```

So that, we deleted the boom-end function, and handle all the issue about boom in the boom-function. and after boom, we remove the countdown=0 Bombstate from the list.

Why we delete the bombcount field of each player and add owner field of bombstate?

This is not a very big issue. But remove-bomb function will remove the countdown=0 Bombstate from the list, so that if we use owner field in Bombstate, we don't need to access player field in the remove-bomb function. It's better than using bombcount field of each player.

Why we update symbol system (the symbol that is legal in layout)?

For adding three length symbols('E1R 'E2L) it is because of the change in Posn convert to Cor part.

For adding two length symbols('E1 'E0) it is because of the change in Why we Add a boom-cor field of the gamestate in the milestone but delete it in the final submit? part.

part5:test part

I need to make dinner, talk about it tomorrow.

