

# **VG101 Project Proposal**

**Yibo Wu & Xiaoyang Sheng & Zeyu Yang**

August 7, 2020

## 1 Group Introduction

‡ Yibo Wu

† Student ID: 519021910323

† Email: [jacky1319@sjtu.edu.cn](mailto:jacky1319@sjtu.edu.cn)

‡ Xiaoyang Sheng

† Student ID: 519021910884

† Email: [sheng6188@sjtu.edu.cn](mailto:sheng6188@sjtu.edu.cn)

‡ Zeyu Yang

† Student ID: 519021910525

† Email: [yangzeyu1026@sjtu.edu.cn](mailto:yangzeyu1026@sjtu.edu.cn)

## 2 Project Introduction

‡ **Name:** Simplified Bomberman

‡ **Intended language:** C/C++

‡ **Summary:** After learning the basic knowledge of C language, we are going to implement the simplified version of a classic game called bomberman by achieving the basic function of the game.

‡ **Motivation:** The Bomberman is a very classic game which was first published in 1983 and popular worldwide. After that, many other games with the similar mechanics are published and got popular in the

early age of the web game. After learning some basic knowledge of C language in this course, our group decided to implement a simplified version on our own so that we can both get trained and enjoy this classic game.

### 3 Design

#### ‡ File I/O and Map Designing:

We choose to use file I/O to fulfill our map loading. For one thing, it can effectively reduce our code in the main function. For another thing, we design several kinds of maps with different difficulty levels conveniently. Player can even design their own maps when necessary. (Please refer to README.md for further instructions.)

We use standard file input and output to accomplish our map load. To simplify our text file, which is the map file. We build a one to one mapping from the signs on the map to a range of numbers. We design three maps based on the numbers of walls and on how strong the wall is.

In the program, we use the variable FILE to read in the text file we have designed. FILE is a pointer variable, which can make input act similar to standard input on screen. We wrap all these steps in functions called readinmap and makes the code more well-structured.

#### ‡ Keyboard Manipulation: In our game, users can control the player roles by directly interacting with keyboard, for keyboard is diverse and

	player 1	player 2
upwards	w	↑
downwards	s	↓
to the left	a	←
to the right	d	→

Table 1: Prevailing Operator

fit for two players to play together. This also corresponds to the most popular games recently.

We use *kbhit* function to get instant input from the keyboard. This ensures players’ instant command. We use if-else statement to determine which character does the player type in. Nowadays, most games employ manipulation just as the table 1 shows, we did the same thing in our game. All these steps are in the main function since these are prelude to the main game codes.

‡ Two-player Mode:

We design two-player mode to make our game more competitive. Players can compare their manipulation as well as luck during the game.

We construct a structure array containing two elements standing for two players. In the structure, there are elements such as healt and power. As a result, all the elements in the structure array acts similarly. We also design functions to enable our players to choose various initialization, for some players may want more health level, some may want more power.

‡ props: Props are important because they add to the diverse game. In this game code, we employ several two-dimensinal arrays, integer type

or character type, to represent different layers of the map. We choose to display the layer we need to the screen.

If we set bomb to make some walls break, we display the prop after the breakable wall is broken down. If there happen to be no prop, we display nothing.

As to the randomness of the prop, we use classical random number generator where we set time as our random number seed. When it comes to use the random numbers, we use the random number to mod 5 to get a random integer in the range from 0 to 4, each number maps to a certain prop. If the player array happens to go through the prop array, which means their coordinates are identical, the player array will receive the command from the prop array. As a result, the feature of player arrays are changed.

#### ‡ Information Bar:

To get instant situation of each player, we must have something always reminding the players how is their health level, how is the power of their bomb, etc.

We display the information bar in the string. Since we apply the double buff screen, we have to set all our information in a string, which means we have to convert some integers into characters. We use the itoa function and we write our own string merge function to fit our use.

#### ‡ Timer:

Since our games is about explosion and the explosion time can be

changed by getting some props, it is necessary to design a timer throughout the entire game. After a player has settled the bomb, we therefore has a specific struct to recording explosion. We use *clock* function to record time, then modify what is being shown on the map.

‡ Double Buff Screen:

Cmd in windows is famous for its shimmering, which is bad for players' eyes as well as their game experience. We therefore have to find ways to solve such proplems.

We create another screen in the buff region. One screen reveals what needed to be shown, in the meantime, the other has already loaded what is going to be shown in a very short time. But it is this short time matters. By changing the two screens, we manage to make the screen not shimmer.

## 4 Outcome

‡ Bottom-line

† Map:

Maps are written in text files. In the text file, 1 stands for unbreakable wall and 2 stands for breakable wall. 3 stands for player 1 and 4 stands for player 2. The figure 1 gives an example of our txt file. We use file I/O to introduce the map. Besides, we have achieved self-design of the map.

† Manipulation:

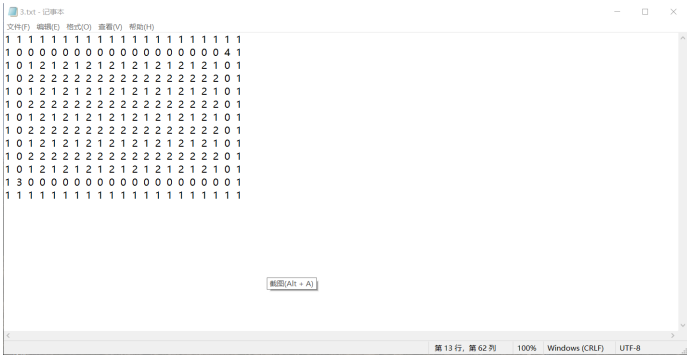


Figure 1: One example of our txt map

Player 1 use up,down,left,right to move and enter key to settle the bomb. Player 2 use WASD to move and space key to settle the bomb.

† Two-player Mode:

Actomatically there will be two players. The two players can simultaneously control their roles. Please see the interface in figure 2 (note that this corresponds to the map txt file before).

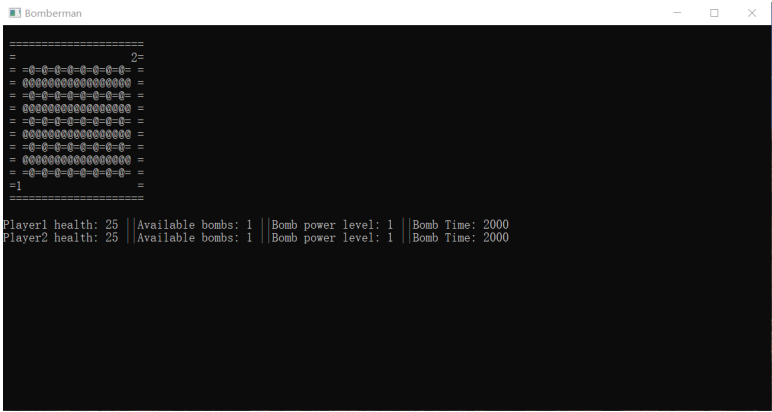


Figure 2: One sample interface

† How to start the game:

Players click on "bomberman.exe" in the directory (see the figure 3), the cmd windows will appear. When the game begin, the title of the cmd windows will be "bomberman". (Please refer to README.md file for detailed information.)










	1.txt	2020/7/30 10:29	文本文档	1 KB
	2.txt	2020/7/30 10:20	文本文档	1 KB
	3.txt	2020/7/30 10:14	文本文档	1 KB
	bomberman.cpp	2020/8/7 19:41	C++ Source File	18 KB
	functions.h	2020/8/1 20:58	JetBrains CLion	7 KB
	Game rule.txt	2020/8/7 19:25	文本文档	3 KB
	Group 11 analysis report.pdf	2020/8/7 22:28	Adobe Acrobat ...	241 KB
	Group11 demo.mp4	2020/8/7 20:29	媒体文件(.mp4)	212,356 KB
	headersvariables.h	2020/8/2 8:56	JetBrains CLion	2 KB
	README.md	2020/8/7 19:36	MD 文件	6 KB

Figure 3: Possible list of files

‡ Expected

† Props:

If the breakable walls are within the range of the bomb, it will reveal a,b,c,d or nothing. 'a' stands for additional 1 bomb; 'b' stands for increasing the power level of the bomb; 'c' stands for fastening the explosion by 150 ms(fastest 500ms);'d' stands for increasing health points by 5 points.

† Different Properties:

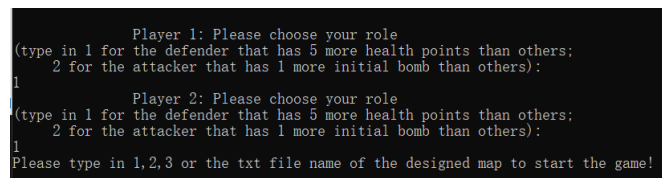
Players have attributes like health points and avaiable bomb numbers. The bombs have features including power level and explosion time.

† Map Choices:



Users type in 1 or 2 or 3 to choose the map they want. 1 stands for the basic map. 2 stands for the classic map with a higher interest. 3 is the most interesting one. Most exciting part is that users can design their own maps guided by the instructions in the README.md. (See the part of starting interface in figure 4)

† When the cmd window are on display, the screen will not shimmer. Since the content which will be shown on the screen is placed in another buff region, what is on the screen and what is going to be shown are not in the same memory space.



```
Player 1: Please choose your role
(type in 1 for the defender that has 5 more health points than others;
 2 for the attacker that has 1 more initial bomb than others):
1
Player 2: Please choose your role
(type in 1 for the defender that has 5 more health points than others;
 2 for the attacker that has 1 more initial bomb than others):
1
Please type in 1,2,3 or the txt file name of the designed map to start the game!
```

Figure 4: Part of our starting page

‡ Potential

† Game Diversity: The diversity of props, maps and roles work together to increase the diversity of the game.

† Before entering the game, for each player he or she can choose 1 or 2 mode. 1 stands for defender, which has 5 more default health points than others. 2 stands for attacker, which has 1 more available bombs at the beginning than others. (See the part of starting interface in figure 4)

† Graphic UI:

After a long period of learning cocos2dx, we found that this is not suitable for our project. So we decided to turn to OpenGL. However, due to the time limit, we cannot accomplish graphic part of our project.

## 5 Progress Timetable

Table 2: Timetable

Event	Data
Designing and writing the codes for keyboard input	Jul. 7 to Jul. 10
Designing the maps and finishing file I/O	Jul. 10 to Jul. 12
Beginning to piece codes, i.e. multiple files project	Jul. 12 to Jul. 13
Design various functions to fulfill more interesting parts	Jul. 13 to Jul. 20
Handing the progress report	Jul. 18
Learning double buff screen	Jul. 21 to Jul. 23
Piecing together all codes and writing README.md, report etc	Jul. 23 to Jul. 28
Handing the whole project	Aug. 3

## 6 Extra Learning Result

### ‡ Github

During the whole process of our project, we use sourcetree and github to help us with different versions of codes. We return to any version if we wish to. We also learn to merge and give others' codes comments by pull request during the journey. It really helps us a lot. The figure 5 is our github README page.

### ‡ Double Buff Screen:

We create another screen for the cmd windows, giving it the same

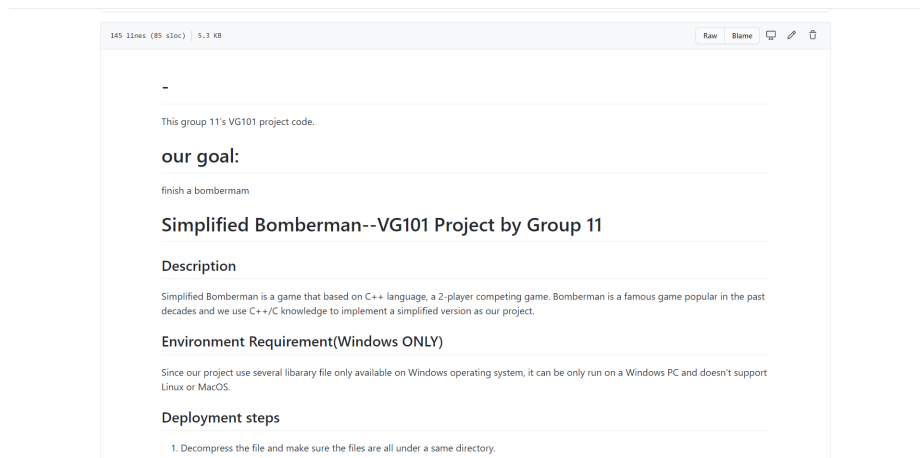


Figure 5: Our github README page

authorities as the original screen. Meanwhile, we remove the cursor to make the page more clear. What we displayed is a two-dimensional string on each screen. Some codes are listed in the appendeix.

## 7 Outcome Summary per Person

- ‡ **Xiaoyang Sheng:** README file, designing game rules and different levels of difficulty, realizing the game logic, debugging. the choices of the roles, demo video.
- ‡ **Zeyu Yang:** Mainly design the two-player mode and piece together all codes. The double buff screen, game interface, report, GitHub management, realizing MVC model.
- ‡ **Yibo Wu:** Learning more about coco2dx the map files and self-design of the map, the choices of the roles, design MVC model.

## 8 Appendix

### 8.1 Double Buff Screen Code

```
hOutBuf = CreateConsoleScreenBuffer(
    GENERIC_WRITE,
    FILE_SHARE_WRITE,//give the screen authorities
    NULL,
    CONSOLE_TEXTMODE_BUFFER,
    NULL
);//Set a new buff screen

hOutput = CreateConsoleScreenBuffer(
    GENERIC_WRITE,
    FILE_SHARE_WRITE,//give the screen authorities
    NULL,
    CONSOLE_TEXTMODE_BUFFER,
    NULL
);//Set the original screen

SetConsoleTitle(TEXT("Bomberman"));
SMALL_RECT rc = {0,0,50,50};
SetConsoleWindowInfo(hOutput,1,&rc);
SetConsoleWindowInfo(hOutBuf,1,&rc);
//change the title of the cmd
```

```
CONSOLE_CURSOR_INFO cci;  
  
//set a variable to control the cursor  
  
cci.bVisible = 0;// Hide the cursor on both screens  
  
cci.dwSize = 1;  
  
SetConsoleCursorInfo(hOutput, &cci);  
  
SetConsoleCursorInfo(hOutBuf, &cci);
```