Bombermen Duel

Project Design

Yichun Deng (yd2348)

Hanyi Du (hd2342)

Murui Li (ml3815)

Wantong Li (wl2520)

CSEE 4840

Embedded System Design

Professor Steven Edwards

1. Project Overview

For the final project, our group hopes to implement the classic bomberman game based on the SoCKit platform and Linux system. The Bombermen Duel will be a two-player game where each player's goal is to defeat the other player through placing bombs on the map. As shown in figure 1, the screen will consist of the two players, bombs, destroyable and non-destroyable obstacles, special items, and the stationary background. The players will control the characters by either sharing the same keyboard or from two separate keyboards. The hardware used in this project will include VGA display, two keyboards, memory, and audio device. The Avalon Bus will be used as the interface between hardware and software.

Figure 1. From www.retrogamer.net

2. Game Rules

At the start of the game, the players will be born at the opposite sides of the screen, with numerous randomly-generated destroyable and non-destroyable obstacles created between them. The players will be capable of placing bombs that will detonate after a fixed amount of time, getting rid of the destroyable obstacles and killing any players that are within the bombing distance. The bombs do not distinguish players, meaning a bomb can also kill the player who placed this bomb. If one player is killed then the surviving player wins. If both players are killed together, it will be a draw game. When a game is finished, the players are reborn at the starting place with a brand new map.

To make the game more interesting, special items are hidden in destroyable obstacles and can be consumed by the players. The special items can be beneficial or detrimental, temporary or permanent for the duration of the game. Below is a list of items that we may implement:

Beneficial items:

- Increasing bomb blast distance (permanent)
- Increasing the maximum number of bombs placed (permanent)
- Increasing player speed (permanent)
- Placing ultra bombs that has whole-screen blast distance (temporary)
- Immunity against bombs (temporary)

Detrimental items:

- Decreasing player speed (temporary)
- Decreasing bomb blast distance (temporary)
- Reverse the direction control (temporary)

3. Architecture

Figure 2 describes the system architecture of this game.

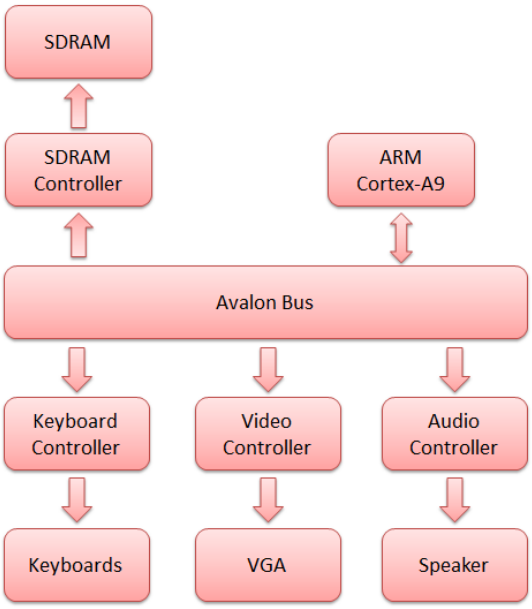

Figure 2. System Architecture.

4. Hardware Implementation

We will use hardware for sprite rendering and audio generation. For graphics, we have obtained sprites like in figure 3 specifically for bombermen games from open-source websites.

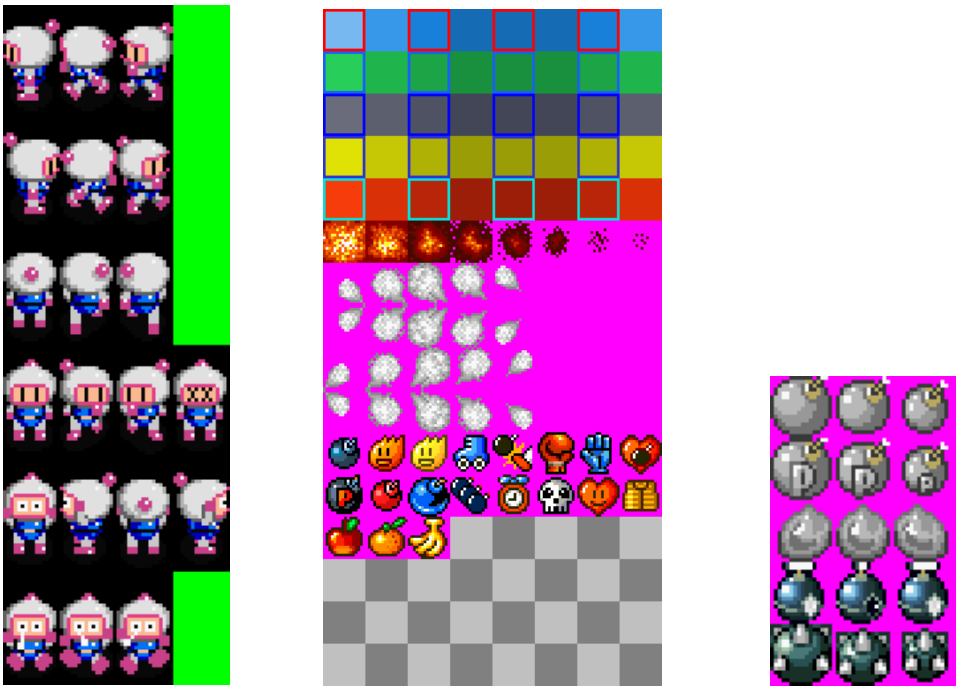

Figure 3. Open-source sprites

Because the screen is 640 x 480 pixels, we plan to make each of the objects (including background blocks, special items, and bombs) to be 32x32 pixels, and the characters to be 32 x 64 pixels. Thus, it makes sense to add a layer of abstraction in the SystemVerilog language to define grids, each of which is 32 x 32 pixels large. This abstraction will provide simpler mechanism for determining the color of each pixel in hardware, as well as using object-based matrix in software instead of a pixel-based one.

From a design point of view, we plan to include three “layers” of graphics. At the bottom there will be the background and immovable blocks, both of which stay constant for the duration of the game. The middle layer will include the special items, destroyable obstacles, and bombs, because they are all one grid large, but may be created or erased at any time. At the top layer will be the characters, who are also changing their positions constantly, but because they are 32 x 64 pixels large thus taking up two grids, each character will conceal part of an object with their “upper body”. This is why they need to be drawn at last to cover up the object or background that has been drawn before.

As for the audio, we will store several sound data into the system, and an audio controller will call the appropriate sound file depending on situations such as game start, game over, bomb explosion, or obstacle destruction. We will either synthesize the sounds, or obtain sound data in .wav files and transfer them into the appropriate format using Matlab.

5. Software Implementation

A. Character Control

Each of the bombermen will be controlled by a keyboard, using up, down, left, right arrows. We plan to include different angles of the characters, both standing and moving. Therefore at least eight states (four moving, four standing) will be needed to represent a character, and the states will be picked from sprites such as figure 3. An additional key for each character will be used for placing a bomb, under the same grid as the character is currently standing in. The keyboard controller will be written to simultaneously listen to both keyboards.

B. Character Status

Software will be used to keep track of the current status of each character, including maximum number of bombs placed, bomb blast distance, character speed, and immunity. Also, we may implement the game with each character having more than one life.

C. Obstacles

Software will also be used to determine where each character is allowed to go and where is forbidden. Specifically, the objects that characters cannot pass through include non-destructible obstacles, destructible obstacles, and bombs. Characters will be able to pass through anything else: special items and the opponent. Obstacle detection is checked every time a movement key is pressed for a character.

D. Map Layout

We plan to use the traditional Bombermen map setting for the arrangement of non-destructible and destructible obstacles, so every time a new game starts, the same map will be generated. However, the difference will be in the arrangement of special items, which will be scattered across the map hidden in destructible obstacles in a random fashion.