EE 254 – Introduction to Digital Circuits

spring 2015 | may 1, 2015

BOMBMERMAN Final Project

Christin carter – computer engineering and computer science anthony asuquo – computer engineering and computer science

Contents

[Abstract 2](#_Toc418119568)

[Introduction and Background 2](#_Toc418119569)

[The Design 2](#_Toc418119570)

[Test Methodology 4](#_Toc418119571)

[Conclusion and Future Work 5](#_Toc418119572)

# Abstract

# Introduction and Background

Bomberman is a computer multiplayer video game where players compete to be the last-man standing on different playing fields. In the original game, players have the ability to produce “bombs” that explode in a cross-like fashion, destroying objects and players in its path. The game is enhanced through players’ abilities to pick up perks and power-ups that allow players to move faster, release multiple “bombs”, and other enhancement.

# The Design

Our version of the game is not at the graphical level, only incorporates a two-player system, and does not consist the variety of power-ups that the original Bomberman game contains. The objective is to destroy your opponent utilizing the bomb-dropping capabilities every player is given. Players start in a main menu and transition to the game play state once both users are ready. During gameplay, players navigate the map and strategically place bombs in an attempt to beat each other. First player to win three rounds wins the game, and the players are asked if they want to replay the game. Below is the state diagram and a more technical description.

. . .

READY

**p2\_** **joystickStartButton**

**WAIT\_P1**

LCD <= “Waiting for P1, P2 Ready!”

**joystickStartButton**

**GameStart**

**WAIT\_P2**

LCD <= “P1 Ready! Waiting for P2”

**p2\_** **joystickStartButton**

**MAIN\_MENU**

waitTimer1 <= 72 waiterTimer2 <= 48

countDigit <= 0 p1\_score <= 0

p2\_score <= 0 LCD <= “Pyromaster Press Start!”

//p1 variables

velocityX <= 0; velocityY <= 0;

direction <= 2'bXX; p1\_dead <= 0;

bombY <= 0; bombX <= 0;

bombRad <= 15; bombCount <= 0;

bombTimer <= 0; bombDelay <= 96;

explodeTimer <= 24; explode <= 0;

positionX <= 30; positionY <= 31;

//p2 has same variables

p2\_positionX <= 610 p2\_positionY <= 450

**WAIT**

LCD <= “Waiting for P1, Waiting for P2”

**GAME**

positionX <= positionX + velocityX;

positionY <= positionY + velocityY;

if (p1\_dead)

p2\_score <= p2\_score + 1;

else if (p2\_dead)

p1\_score <= p1\_score + 1;

if (joystick pushed down) //Set speed of player in specific direction

velocityY <= -3;

direction <=1;

else if (joystick pushed up)

velocityY <= 3;

direction <=0;

else if (joystick pushed left)

velocityX <= -3;

direction <=2;

else if (joystick pushed right)

velocityX <= 3;

direction <=3;

else

velocityX <= 0;

velocityY <= 0;

if (joystickBombButton && bombCount == 0 && bombTimer == 0 && explode == 0)

bombCount <= bombCount + 1; //Produce bomb where player pressed

bombTimer <= bombDelay; //the joystick button

bombY <= positionY;

bombX <= positionX;

if (!(bombTimer == 0)) //Countdown to detonation

bombTimer <= bombTimer - 1;

else if ((bombTimer == 0) && !(bombCount == 0)) //Displays the explosion of the

bombCount <= bombCount - 1; //bomb on screen

explode <= 1;

if (explode)

explodeTimer <= explodeTimer - 1;

if (explodeTimer == 0) //Resets the timers and state

explode <= 0; //of the players bomb

explodeTimer <= 24;

if (p1\_killed)

p1\_dead <= 1;

LCD <= “P1 Score –“ + p1\_score + “P2 Score –“ + p2\_score //display scores on LCD

**READY**

if(waitTimer1 == 0)

waitTimer <= 72

else

if (waitTimer1 <= 24)

countDigit <= 1

else if (waitTimer1 <= 48)

countDigit <= 2

else if (waitTimer1 <= 72)

countDigit <= 3

waitTimer1 <= waitTimer1 – 1

LCD <= “Game Starts in” + countDigit

**RND\_DRAW**

LCD <= “Round Draw”

**RND\_WON\_P2**

LCD <= “P1 wins Round!”

**RND\_WON\_P1**

LCD <= “P1 wins Round!”

**PLAY\_AGAIN**

LCD <= “Play Again, Yes No”

**GAME\_WON\_P1**

LCD <= “P1 wins Round!”

**GAME\_WON\_P2**

LCD <= “P2 wins Round!”

**GAME (Continued)**

if (positionY < 26) //Collisions with border of playing field

positionY <= 27;

else if (positionY > 454)

positionY <= 453;

else if (positionX < 25)

positionX <= 26;

else if (positionX > 615)

positionX <= 614;

else if (direction == 0) // Collisions when Moving Up

if (maze\_block\_0 || maze\_block\_1 || maze\_block\_2 || maze\_block\_3)

positionY <= 131; //Resets blocks position if player overlaps with block

else if (maze\_block\_4 || maze\_block\_5 || maze\_block\_6 || maze\_block\_7)

positionY <= 235;

else if (maze\_block\_8 || maze\_block\_9 || maze\_block\_10 || maze\_block\_11)

positionY <= 339;

else if (maze\_block\_12 || maze\_block\_13 || maze\_block\_14 || maze\_block\_15)

positionY <= 443;

else if (direction == 1) // Collisions when Moving Down

. . .

. . .

else if (direction == 2) // Collisions when Moving Left

. . .

. . .

else if (direction == 3) // Collisions when Moving Right

. . .

. . .

# Test Methodology

The main testing method we utilized was trial and error. One of the toughest parts to implement was collision detections.

# Conclusion and Future Work