**Slap-A-Mole**

*Arduino Bar Game Table*

GAME 2334 – Project Development 2



**Index**

Index…………………………………

Overview ……………………………

Programming ………………………..

…………………………………...... 1

…………………………………... 2-3

…………………………………. 4-11

**Overview**

Slap-a-Mole is the final product of a pair of students at HCCS for the second of 2 projects for our Project Development 2 class, GAME 2334, for the DGS program. This valuable class is aimed at teaching some of the skills and teamwork needed to get a larger project done when working with more than just ourselves. Our project is an Arduino-driven game table styled after the arcade game Whack-A-Mole

**Gameplay**

The table has two game modes in its current state: single and two-player. In single-player, the player has 24 seconds to press as many of the lit-up buttons as they can. Their remaining time, and once it runs out their score, is displayed on the dial at the head of the table. In two-player, the players both begin with 12 points. Each time one pressed an LED of their color, red or blue, one point is taken from their opponent and given to them. The game ends when all 24 points are claimed by either of the players. The current point balance of the players is displayed on the dial.

**Controls**

* *Bottom Right & 2nd Bottom Right:* Start Single-Player
* *Top Right & 2nd Top Right:* Start Two-Player



 **Material Construction**

The table was made using yellow pine with cherry stain and sealed with a polyurethane glaze. The surface was sanded to bring it closer to a level surface (yellow pine is not known for being straight). The top is attached with hinges for easier access. Arcade switches are used for the buttons for a satisfying sound and feel, with red and blue LEDs embedded to create the glow.

**Tech**

- Arduino Uno

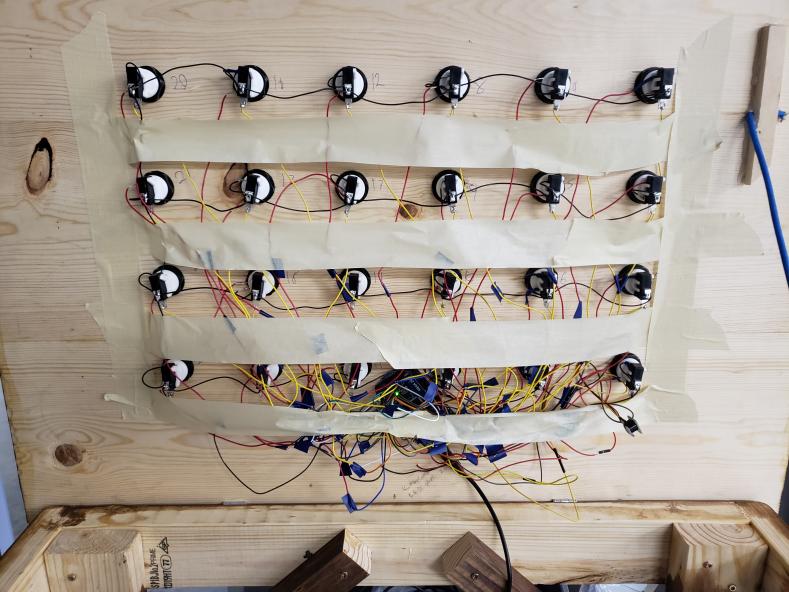
- C

**Team**

*Ryan Brown:* Design & Construction

*Daniel Molski:* Programmer

**Programming**



#include <Adafruit\_NeoPixel.h>

//system vars

const unsigned int rows = 6;

const unsigned int cols = 4;

const unsigned int area = rows \* cols;

unsigned int game\_state = 0;

unsigned long current\_time = 0;

unsigned long start\_time = 0;

unsigned long last\_ring\_update = 0;

unsigned int animation\_state = 0;

const unsigned int animation\_lag = 100;

//output vars

const unsigned int led\_offset = 2;

unsigned int led\_pins[area];

unsigned int led\_buffer[area];

const int ring\_pin = A1;

const unsigned int ring\_leds = 24;

Adafruit\_NeoPixel ring(ring\_leds, ring\_pin, NEO\_GRBW + NEO\_KHZ800);

char ring\_r[ring\_leds];

char ring\_g[ring\_leds];

char ring\_b[ring\_leds];

char ring\_w[ring\_leds];

const int buzzer\_pin = A0;

const int low\_buzz = 523;

const int high\_buzz = 784;

int buzz\_tone[8];

//input vars

const unsigned int but\_offset = 26;

unsigned int but\_pins[area];

unsigned int curr\_states[area];

unsigned int prev\_states[area];

int red\_but\_pins[area / 2];

int r\_curr\_states[area / 2];

int blue\_but\_pins[area / 2];

int b\_curr\_states[area / 2];

int \*blue\_leds[area / 2];

int \*red\_leds[area / 2];

//system functions

void check\_All\_Buttons()

{

for (int i = 0; i < area; i++)

{

prev\_states[i] = curr\_states[i];

pinMode(but\_pins[i], OUTPUT);

digitalWrite(but\_pins[i], HIGH);

pinMode(but\_pins[i], INPUT);

curr\_states[i] = digitalRead(but\_pins[i]);

}

}

void check\_RB\_Buttons()

{

for (int i = 0; i < area / 2; i++)

{

pinMode(red\_but\_pins[i], OUTPUT);

digitalWrite(red\_but\_pins[i], HIGH);

pinMode(red\_but\_pins[i], INPUT);

r\_curr\_states[i] = digitalRead(red\_but\_pins[i]);

pinMode(blue\_but\_pins[i], OUTPUT);

digitalWrite(blue\_but\_pins[i], HIGH);

pinMode(blue\_but\_pins[i], INPUT);

b\_curr\_states[i] = digitalRead(blue\_but\_pins[i]);

}

}

//buffer functions

void print\_Buffer()

{

for (int i = 0; i < area; i++) digitalWrite(led\_pins[i], led\_buffer[i]);

}

void fill\_Buffer(int state)

{

for (int i = 0; i < area; i++) led\_buffer[i] = state;

}

void fill\_Row(unsigned int row, int state)

{

for (int i = 0; i < cols; i++) led\_buffer[i + cols \* row] = state;

}

void fill\_Col(unsigned int col, int state)

{

for (int i = 0; i < rows; i++) led\_buffer[i \* cols + col] = state;

}

void toggle\_Row(unsigned int row)

{

for (int i = 0; i < cols; i++)

{

if (led\_buffer[i + cols \* row] == HIGH) led\_buffer[i + cols \* row] = LOW;

else led\_buffer[i + cols \* row] = HIGH;

}

}

void toggle\_Col(unsigned int col)

{

for (int i = 0; i < rows; i++)

{

if (led\_buffer[i \* cols + col] == HIGH) led\_buffer[i \* cols + col] = LOW;

else led\_buffer[i \* cols + col] = HIGH;

}

}

void roll\_Buf\_L()

{

int temp\_col[rows];

for (int i = 0; i < rows; i++) temp\_col[i] = led\_buffer[i \* cols];

for (int y = 0; y < rows; y++)

{

for (int x = 0; x < cols - 1; x++)

{

int index = y \* cols + x;

led\_buffer[index] = led\_buffer[index + 1];

}

led\_buffer[(y + 1) \* cols - 1] = temp\_col[y];

}

}

void roll\_Buf\_R()

{

int temp\_col[rows];

for (int i = 0; i < rows; i++) temp\_col[i] = led\_buffer[(i + 1) \* cols - 1];

for (int y = 0; y < rows; y++)

{

for (int x = cols - 1; x > 0; x--)

{

int index = y \* cols + x;

led\_buffer[index] = led\_buffer[index - 1];

}

led\_buffer[y \* cols] = temp\_col[y];

}

}

void roll\_Buf\_U()

{

int temp\_col[rows];

for (int i = 0; i < cols; i++) temp\_col[i] = led\_buffer[i];

for (int x = 0; x < cols; x++)

{

for (int y = 0; y < rows - 1; y++)

{

int index = y \* cols + x;

led\_buffer[index] = led\_buffer[index + cols];

}

led\_buffer[(rows - 1) \* cols + x] = temp\_col[x];

}

}

void roll\_Buf\_D()

{

int temp\_col[rows];

for (int i = 0; i < cols; i++) temp\_col[i] = led\_buffer[i + cols \* (rows - 1)];

for (int x = 0; x < cols; x++)

{

for (int y = rows - 1; y > 0; y--)

{

int index = y \* cols + x;

led\_buffer[index] = led\_buffer[index + cols];

}

led\_buffer[x] = temp\_col[x];

}

}

//ring functions

void print\_Ring()

{

for (int i = 0; i < ring\_leds; i++) ring.setPixelColor(i, ring\_r[i], ring\_g[i], ring\_b[i], ring\_w[i]);

ring.show();

}

void clear\_Ring()

{

for (int i = 0; i < ring\_leds; i++)

{

ring\_r[i] = 0;

ring\_g[i] = 0;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

}

void fill\_Ring(char r, char g, char b, char w)

{

for (int i = 0; i < ring\_leds; i++)

{

ring\_r[i] = r;

ring\_g[i] = g;

ring\_b[i] = b;

ring\_w[i] = w;

}

}

void rainbow\_Ring(char brightness)

{

for (int i = 0; i < ring\_leds / 3; i++)

{

ring\_r[i] = brightness - (brightness \* i / (ring\_leds / 3));

ring\_g[i] = brightness \* i / (ring\_leds / 3);

ring\_b[i] = 0;

ring\_w[i] = 0;

}

for (int i = ring\_leds / 3; i < 2 \* ring\_leds / 3; i++)

{

ring\_r[i] = 0;

ring\_g[i] = brightness - brightness \* (i - ring\_leds / 3) / (ring\_leds / 3);

ring\_b[i] = brightness \* (i - ring\_leds / 3) / (ring\_leds / 3);

ring\_w[i] = 0;

}

for (int i = 2 \* ring\_leds / 3; i < ring\_leds; i++)

{

ring\_r[i] = brightness \* (i - 2 \* ring\_leds / 3) / (ring\_leds / 3);

ring\_g[i] = 0;

ring\_b[i] = brightness - brightness \* (i - 2 \* ring\_leds / 3) / (ring\_leds / 3);

ring\_w[i] = 0;

}

}

void shift\_Ring\_CW()

{

char temp\_r = ring\_r[0];

char temp\_g = ring\_g[0];

char temp\_b = ring\_b[0];

char temp\_w = ring\_w[0];

for (int i = 0; i < ring\_leds - 1; i++)

{

ring\_r[i] = ring\_r[i+1];

ring\_g[i] = ring\_g[i+1];

ring\_b[i] = ring\_b[i+1];

ring\_w[i] = ring\_w[i+1];

}

ring\_r[ring\_leds - 1] = temp\_r;

ring\_g[ring\_leds - 1] = temp\_g;

ring\_b[ring\_leds - 1] = temp\_b;

ring\_w[ring\_leds - 1] = temp\_w;

print\_Ring();

}

void shift\_Ring\_CCW()

{

char temp\_r = ring\_r[23];

char temp\_g = ring\_g[23];

char temp\_b = ring\_b[23];

char temp\_w = ring\_w[23];

for (int i = 23; i > 0; i--)

{

ring\_r[i] = ring\_r[i-1];

ring\_g[i] = ring\_g[i-1];

ring\_b[i] = ring\_b[i-1];

ring\_w[i] = ring\_w[i-1];

}

ring\_r[0] = temp\_r;

ring\_g[0] = temp\_g;

ring\_b[0] = temp\_b;

ring\_w[0] = temp\_w;

print\_Ring();

}

//game functions

void start\_Game()

{

fill\_Buffer(LOW);

print\_Buffer();

fill\_Ring(30, 0, 0, 0);

print\_Ring();

tone(buzzer\_pin, low\_buzz);

delay(400);

clear\_Ring();

print\_Ring();

noTone(buzzer\_pin);

delay(200);

fill\_Ring(20, 0, 0, 0);

print\_Ring();

tone(buzzer\_pin, low\_buzz);

delay(400);

clear\_Ring();

print\_Ring();

noTone(buzzer\_pin);

delay(200);

fill\_Ring(10, 0, 0, 0);

print\_Ring();

tone(buzzer\_pin, low\_buzz);

delay(400);

clear\_Ring();

noTone(buzzer\_pin);

delay(200);

fill\_Ring(0, 30, 0, 0);

print\_Ring();

tone(buzzer\_pin, high\_buzz);

delay(600);

clear\_Ring();

print\_Ring();

noTone(buzzer\_pin);

delay(100);

}

void end\_2p(int winner)

{

fill\_Buffer(LOW);

print\_Buffer();

if (winner == 1)

{

fill\_Ring(30, 0, 0, 0);

for (int i = 0; i < area / 2; i++) \*red\_leds[i] = HIGH;

}

else if (winner == 2)

{

fill\_Ring(0, 0, 30, 0);

for (int i = 0; i < area / 2; i++) \*blue\_leds[i] = HIGH;

}

else

{

fill\_Ring(0, 30, 0, 0);

fill\_Buffer(HIGH);

}

print\_Ring();

print\_Buffer();

tone(buzzer\_pin, high\_buzz);

delay(200);

clear\_Ring();

print\_Ring();

fill\_Buffer(LOW);

print\_Buffer();

noTone(buzzer\_pin);

delay(100);

if (winner == 1)

{

fill\_Ring(30, 0, 0, 0);

for (int i = 0; i < area / 2; i++) \*red\_leds[i] = HIGH;

}

else if (winner == 2)

{

fill\_Ring(0, 0, 30, 0);

for (int i = 0; i < area / 2; i++) \*blue\_leds[i] = HIGH;

}

else

{

fill\_Ring(0, 30, 0, 0);

fill\_Buffer(HIGH);

}

print\_Ring();

print\_Buffer();

tone(buzzer\_pin, high\_buzz);

delay(200);

clear\_Ring();

print\_Ring();

fill\_Buffer(LOW);

print\_Buffer();

noTone(buzzer\_pin);

delay(100);

if (winner == 1)

{

fill\_Ring(30, 0, 0, 0);

for (int i = 0; i < area / 2; i++) \*red\_leds[i] = HIGH;

}

else if (winner == 2)

{

fill\_Ring(0, 0, 30, 0);

for (int i = 0; i < area / 2; i++) \*blue\_leds[i] = HIGH;

}

else

{

fill\_Ring(0, 30, 0, 0);

fill\_Buffer(HIGH);

}

print\_Ring();

print\_Buffer();

tone(buzzer\_pin, high\_buzz);

delay(200);

clear\_Ring();

print\_Ring();

fill\_Buffer(LOW);

print\_Buffer();

noTone(buzzer\_pin);

}

void end\_1p(unsigned int score)

{

fill\_Buffer(LOW);

print\_Buffer();

fill\_Ring(30, 0, 0, 0);

print\_Ring();

tone(buzzer\_pin, high\_buzz);

delay(200);

clear\_Ring();

print\_Ring();

noTone(buzzer\_pin);

delay(100);

fill\_Ring(30, 0, 0, 0);

print\_Ring();

tone(buzzer\_pin, high\_buzz);

delay(200);

clear\_Ring();

print\_Ring();

noTone(buzzer\_pin);

delay(100);

fill\_Ring(30, 0, 0, 0);

print\_Ring();

tone(buzzer\_pin, high\_buzz);

delay(200);

clear\_Ring();

print\_Ring();

noTone(buzzer\_pin);

fill\_Buffer(LOW);

print\_Buffer();

clear\_Ring();

print\_Ring();

start\_time = 0;

current\_time = millis();

char r = 0;

char g = 0;

char b = 0;

char w = 0;

for (int i = 0; i < score;)

{

current\_time = millis();

if (current\_time - start\_time >= animation\_lag)

{

start\_time = millis();

if (i % (3 \* ring\_leds) < ring\_leds)

{

r = 0;

g = 50;

b = 0;

w = 0;

}

else if (i % (3 \* ring\_leds) < 2 \* ring\_leds)

{

r = 0;

g = 0;

b = 50;

w = 0;

}

else

{

r = 50;

g = 0;

b = 0;

w = 0;

}

ring\_r[i % ring\_leds] = r;

ring\_g[i % ring\_leds] = g;

ring\_b[i % ring\_leds] = b;

ring\_w[i % ring\_leds] = w;

print\_Ring();

tone(buzzer\_pin, buzz\_tone[i % 8], 3 \* animation\_lag / 4);

i++;

}

}

delay(2000);

}

void single\_Player()

{

int lit\_led\_1 = rand() % area;

int temp\_lit\_led = rand() % area;

while (temp\_lit\_led == lit\_led\_1) temp\_lit\_led = rand() % area;

int lit\_led\_2 = temp\_lit\_led;

int score = 0;

start\_Game();

fill\_Ring(0, 30, 0, 0);

print\_Ring();

led\_buffer[lit\_led\_1] = HIGH;

led\_buffer[lit\_led\_2] = HIGH;

print\_Buffer();

current\_time = millis();

start\_time = current\_time;

last\_ring\_update = current\_time;

for(;;)

{

current\_time = millis();

if (current\_time - start\_time >= 24000) break;

if (current\_time - last\_ring\_update >= 1000)

{

unsigned long d\_time = (current\_time - start\_time) / 1000;

last\_ring\_update = current\_time;

if (d\_time < 8)

{

for (int i = d\_time; i < 24; i++)

{

ring\_r[i] = 0;

ring\_g[i] = 30;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

for (int i = 0; i < d\_time; i++)

{

ring\_r[i] = 0;

ring\_g[i] = 0;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

}

else if (d\_time < 16)

{

for (int i = d\_time; i < 24; i++)

{

ring\_r[i] = 20;

ring\_g[i] = 10;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

for (int i = 0; i < d\_time; i++)

{

ring\_r[i] = 0;

ring\_g[i] = 0;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

}

else

{

for (int i = d\_time; i < 24; i++)

{

ring\_r[i] = 30;

ring\_g[i] = 0;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

for (int i = 0; i < d\_time; i++)

{

ring\_r[i] = 0;

ring\_g[i] = 0;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

}

print\_Ring();

}

check\_All\_Buttons();

if (curr\_states[lit\_led\_1] == LOW)

{

tone(buzzer\_pin, 784, 100);

led\_buffer[lit\_led\_1] = LOW;

score++;

int new\_lit = rand() % area;

while (new\_lit == lit\_led\_1 || new\_lit == lit\_led\_2) new\_lit = rand() % area;

lit\_led\_1 = new\_lit;

led\_buffer[lit\_led\_1] = HIGH;

print\_Buffer();

}

if (curr\_states[lit\_led\_2] == LOW)

{

tone(buzzer\_pin, 784, 100);

led\_buffer[lit\_led\_2] = LOW;

score++;

int new\_lit = rand() % area;

while (new\_lit == lit\_led\_1 || new\_lit == lit\_led\_2) new\_lit = rand() % area;

lit\_led\_2 = new\_lit;

led\_buffer[lit\_led\_2] = HIGH;

print\_Buffer();

}

}

end\_1p(score);

}

void two\_Player()

{

start\_Game();

int timeout = 10000;

unsigned long last\_button\_press = millis();

int red\_lit = rand() % (area / 2);

\*red\_leds[red\_lit] = HIGH;

int blue\_lit = rand() % (area / 2);

\*blue\_leds[blue\_lit] = HIGH;

print\_Buffer();

for (int i = 0; i < ring\_leds / 2; i++)

{

ring\_r[i] = 30;

ring\_g[i] = 0;

ring\_b[i] = 0;

ring\_w[i] = 0;

}

for (int i = ring\_leds / 2; i < ring\_leds; i++)

{

ring\_r[i] = 0;

ring\_g[i] = 0;

ring\_b[i] = 30;

ring\_w[i] = 0;

}

print\_Ring();

int red\_score = 12;

int blue\_score = 12;

current\_time = millis();

while (blue\_score < 24 && red\_score < 24)

{

current\_time = millis();

if (current\_time - last\_button\_press > timeout) return;

check\_RB\_Buttons();

if (r\_curr\_states[red\_lit] == LOW)

{

last\_button\_press = millis();

\*red\_leds[red\_lit] = LOW;

tone(buzzer\_pin, high\_buzz, 100);

ring\_r[red\_score] = 30;

ring\_g[red\_score] = 0;

ring\_b[red\_score] = 0;

int new\_lit = rand() % (area / 2);

while (new\_lit == red\_lit) new\_lit = rand() % (area / 2);

red\_lit = new\_lit;

\*red\_leds[red\_lit] = HIGH;

print\_Buffer();

print\_Ring();

red\_score++;

blue\_score--;

}

if (b\_curr\_states[blue\_lit] == LOW)

{

last\_button\_press = millis();

\*blue\_leds[blue\_lit] = LOW;

tone(buzzer\_pin, high\_buzz, 100);

ring\_r[23 - blue\_score] = 0;

ring\_g[23 - blue\_score] = 0;

ring\_b[23 - blue\_score] = 30;

int new\_lit = rand() % (area / 2);

while (new\_lit == blue\_lit) new\_lit = rand() % (area / 2);

blue\_lit = new\_lit;

\*blue\_leds[blue\_lit] = HIGH;

print\_Buffer();

print\_Ring();

blue\_score++;

red\_score--;

}

}

if (red\_score >= 24) end\_2p(1);

else if (blue\_score >= 24) end\_2p(2);

}

int main\_Menu()

{

animation\_state = 0;

rainbow\_Ring(50);

print\_Ring();

int counter = 0;

for (;;)

{

check\_All\_Buttons();

if (curr\_states[0] == LOW && curr\_states[1] == LOW) return 1;

else if (curr\_states[2] == LOW && curr\_states[3] == LOW) return 2;

else if (curr\_states[4] == LOW && curr\_states[5] == LOW) return 3;

else if (curr\_states[6] == LOW && curr\_states[7] == LOW) return 4;

current\_time = millis();

if (current\_time - start\_time >= animation\_lag)

{

start\_time = millis();

shift\_Ring\_CW();

print\_Ring();

counter++;

if ((counter - 4) % 8 == 0)

{

if (counter < 8 \* 5)

{

led\_buffer[0] = HIGH;

led\_buffer[1] = HIGH;

}

else

{

led\_buffer[2] = HIGH;

led\_buffer[3] = HIGH;

}

}

else if (counter % 8 == 0)

{

led\_buffer[0] = LOW;

led\_buffer[1] = LOW;

led\_buffer[2] = LOW;

led\_buffer[3] = LOW;

}

if (counter % (8 \* 10) == 0)

{

counter = 0;

}

print\_Buffer();

}

}

}

//default functions

void setup()

{

pinMode(buzzer\_pin, OUTPUT);

ring.begin();

clear\_Ring();

print\_Ring();

for (int i = 0; i < area; i++)

{

led\_pins[i] = i + led\_offset;

pinMode(led\_pins[i], OUTPUT);

digitalWrite(led\_pins[i], LOW);

but\_pins[i] = i + but\_offset;

pinMode(but\_pins[i], INPUT);

curr\_states[i] = LOW;

}

buzz\_tone[0] = 523;

buzz\_tone[1] = 587;

buzz\_tone[2] = 659;

buzz\_tone[3] = 698;

buzz\_tone[4] = 784;

buzz\_tone[5] = 880;

buzz\_tone[6] = 988;

buzz\_tone[7] = 1047;

int blue\_offset = 0;

int red\_offset = 0;

for (int y = 0; y < rows; y++)

{

if (y % 2 == 0)

{

for (int x = 0; x < cols; x += 2)

{

blue\_but\_pins[blue\_offset] = but\_pins[y \* cols + x];

blue\_leds[blue\_offset] = &led\_buffer[y \* cols + x];

red\_but\_pins[red\_offset] = but\_pins[y \* cols + x + 1];

red\_leds[red\_offset] = &led\_buffer[y \* cols + x + 1];

blue\_offset++;

red\_offset++;

}

}

else

{

for (int x = 0; x < cols; x += 2)

{

red\_but\_pins[red\_offset] = but\_pins[y \* cols + x];

red\_leds[red\_offset] = &led\_buffer[y \* cols + x];

blue\_but\_pins[blue\_offset] = but\_pins[y \* cols + x + 1];

blue\_leds[blue\_offset] = &led\_buffer[y \* cols + x + 1];

blue\_offset++;

red\_offset++;

}

}

}

for (int i = 0; i < area / 2; i++)

{

r\_curr\_states[i] = LOW;

b\_curr\_states[i] = LOW;

}

}

void loop()

{

clear\_Ring();

print\_Ring();

fill\_Buffer(LOW);

print\_Buffer();

if (game\_state == 1)

{

single\_Player();

game\_state = 0;

}

else if (game\_state == 2)

{

two\_Player();

game\_state = 0;

}

else game\_state = main\_Menu();

}