Mini Project-Arcade Like
2D Gaming System Based On ATmega328P

Tonghe Liu

November 30, 2023

1 Introduction

For my mini project, the ATmega328P is utilized as the center control unit and Atmel Studio as the compiler. We wrote a program to control the LCD1602, producing dynamic effects for our design. Our code did not include any external libraries for components and all the functionalities are coded and debugged by ourselves. The result is a obstacle avoidance shooting game featuring three game modes, three difficulty levels, and a total of six different character and enemy appearances. The game is not designed to be an infinite loop, where players do get a way to win the game but it's rather hard to do so. We will talk more about our game design and algorithms in the third part of this report. Our system also contains a Servo(TS90A), an ultrasonic distance sensor(HC SR04), a button for the activation interrupt service routine. Their functionalities will be elaborated in the second part of this report. The following schematic shows our circuit design of the whole system.

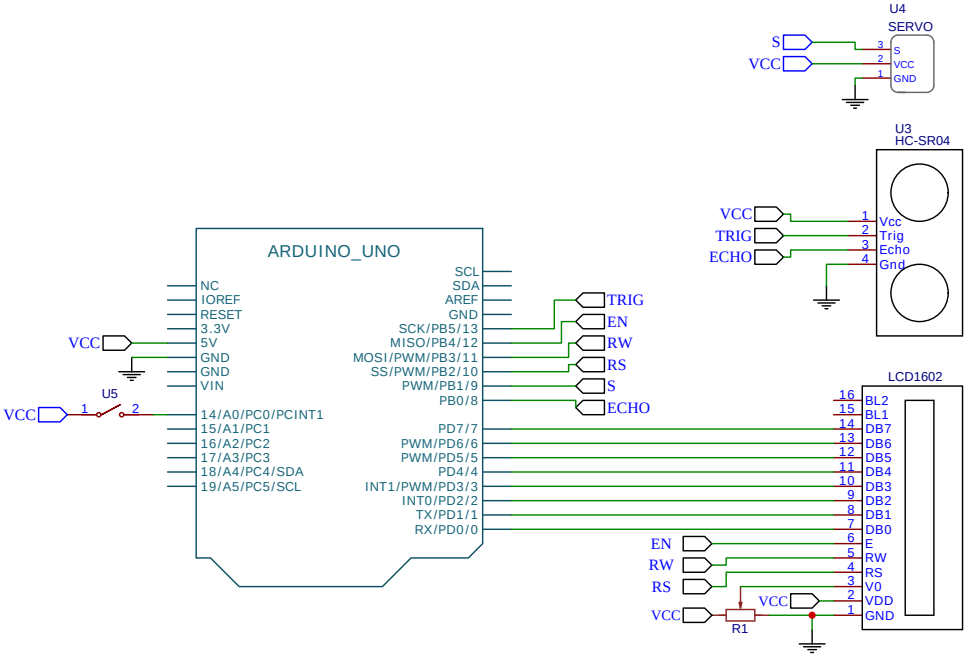

Figure 1: The project schematic

Our project incorporates six out of the seven specified functional requirements.

Although we utilize ultrasonic distance measurement and accurately read the continuous changes in distance into the chip, this process does not involve analog-to-digital conversion and so does our system. The functionality of each component will be detailed in the second section.

2 Functionalities of Components

2.1 LCD Display

The LCD Display is the most important part of system. To display custom characters on the screen, representing various objects such as enemies and bullets, we utilized the write functionality of the screen's RW(Write) pin. This meant that we could not continue using the given header file for LCD, but instead, we modified the file to be able to store and use our custom graphics on the LCD screen.

Here, we utilized the following code to save our custom characters into the CGRAM of the screen:

```
void LCD_Custom_Char(unsigned char location, unsigned char *msg)
{
  if(location<8) // CGRAM only has 8 locations
  {
    LCD_Command(0x40+(location*8)); // point to CGRAM address
    for(int i=0; i<8; i++)
    {
      LCD_Char(msg[i]);
    }
  }
}
```

Then we can define our own characters and save them to the CGRAM like following:

```
unsigned char Character1[8] = {0x0E, 0x0C, 0x06, 0x1F, 0x0E, 0
    x1B, 0x1B, 0x11};  // custom character 1, agent
LCD_Custom_Char(0, Character1);
```

The way we let the screen to display moving parts is to use LCD_Set_Cursor() and _delay_ms(). It seems to be a great idea if we utilize the LCD_Shift_Left() or LCD_Shift_Right() to do the job, however, that would cause severe timing problem if we want to create multiple moving objects on the screen(for example, what if bullets and enemy are all moving). The following code shows the general idea of moving object control.

```
while(1)
{
  while(playing)
  {
    LCD_Clear();
    LCD_Set_Cursor(object_row, object_column);
    LCD_Char(object_idx);
    ...// change position, logic operations, etc.
    _delay_ms(certainConstant/speed); // control the refresh
     rate
  }
}
```

2.2 Timer, PWM and Servo Control

For our project, we are using servo to rotate to mimic the arcade-styled gaming console. Here we are configuring Timer/Counter 1 to use the Fast PWM mode. OCR1A is set to control the rotation angle of the servo. The following code is wrote to control the servo:

```
void servo_spin()
{
  DDRB |= 1 << PINB1; // Set PB1 to output

  TCCR1A |= (1 << WGM11) | (1 << COM1A1); // enable the non-
    inverting PWM output on PB1
  TCCR1B |= (1 << WGM12) | (1 << WGM13) | (1 << CS11); //
    determining the timer clock frequency

  ICR1 = 24999;
  for(int i = 0; i < 3; i++)
  {
    OCR1A = 4899;

    _delay_ms(1000);

    OCR1A = 1199;

    _delay_ms(1000);
  }
}
```

2.3 Ultrasonic Distance Sensor

The way to use ultrasonic distance sensor is first to output a pulse signal for HC-SR04, then this pulse will enable the component to send out 8 ultrasonic waves and the echo pin will be turned on. Once the ultrasonic waves are detected to be echoing back, the echo pin will be turned off. By calculating the time interval of sending and receiving ultrasonic waves, we're able to calculate the distance. The following code shows our work:

```
void trigger_pulse()
{
  PORTB |= (1 << TRIGGER_PIN);
  _delay_us(10);
  PORTB &= ~(1 << TRIGGER_PIN);
}


uint16_t measure_distance()
{
  uint32_t count = 0;
  uint16_t distance = 0;
  trigger_pulse();
  while (!(PINB & (1 << ECHO_PIN)))
  {
    _delay_us(10);
    PORTB |= (1 << TRIGGER_PIN);
    _delay_us(10);
    PORTB &= ~(1 << TRIGGER_PIN);
  }

  while (PINB & (1 << ECHO_PIN))
  {
    count++;
    _delay_us(1);
  }

  distance = (uint16_t)(count * 0.034 / 2);
  return distance;
}
```

2.4 Interrupt

Our project uses button to activate the Interrupt Service Routine(ISR). Here, since the default pin of interrupt are all being used, we utilize the Pin Change

Interrupt 1 for Port C pins. Here, we use A0 as the interrupt pin.

```
// initialize interrupt pin
DDRC |= (1 << INTERRUPT_PIN);
PORTC &= ~(1 << INTERRUPT_PIN);

cli();
PCICR |=  0b00000010; // Enables Ports C Pin Change Interrupts
PCMSK1 |= 0b00000001; // PCINT0(PC0)
sei();
```

Our ISR is being written as below:

```
ISR(PCINT1_vect) //ISR for pin change external interrupt 1
{
  Jump_Flag = 1;
  if(Game_Paused)
  {
    Agent_Switch_Flag = 1;
  }
}
```

Here we are using interrupt to control the position of our protagonist. In order to avoid switch bounce, here we will only set the jump flag to be true, rather than changing the position immediately. And what should be noticed is that, if interrupt is triggered when game is paused, it allows you to change between different agents(protagonist). Again, details will be elaborate in section 3.

2.5 Serial Communication

In our project, we are using USART to send the score the player gets to the upper computer and save it in a text file. We configured the specifications of the the communication by the following code:

```
void initSerialPort  (void)
{
  UCSR0B = 0x08;  // enabling the transmitter
  UCSR0C = 0x06;  // 8 data bits, no parity, and 1 stop bit.
  UBRR0L = 0x67;  // set baud rate to 9600
}
void sendData (unsigned char character)
{
  while (!(UCSR0A & (1<<UDRE0))); // wait until the USART Data
    Register is empty
  UDR0 = character;
}
```

3 The Main Loop, Algorithms & Logic Operations

Limited by the hardware, it seems to be not possible to program an interesting, playable and challenging game using LCD1602, but we tried our best to do so. The game has the following features:

- There're three main character with different appearance for you to choose from.
- The three characters have different features.
- The game with the first character will be in the normal mode, where you avoid being hit by enemies appearing in random rows of the screen.
- The second character is able to shoot bullets; If you kill one of the enemies, your score will be added a certain value.
- The third character is invincible. The game is in debug mode and the player will not be killed.
- There're three levels of difficulty. The enemy appearance, enemy health will change accordingly; for the final level, enemy will be able to shoot bullets.
- There are bonus round after you finish a certain round, where there's a bonus for you to catch to gain scores.
- If the player survived the final round, they will be in the ultimate level, where a bonus will appearing randomly on the screen. If they manage to catch it, they won. Or they will be challenging level 3 again.
- You can use the ultrasonic sensor to pause the game by decreasing the distance; while in pause mode, you can freely change any characters that you like.
- Scores will be displayed on the screen all the way till the end of the game. Once the game ends, your score will be automatically sent to the upper computer.
- There's a welcome page at the beginning of the game. There will be page to indicate which level are you in once you're about to level up. Also, there will be an ending page to tell your total scores.

In order to build game with these features, we spend most of the programming functions and the main loop. Basically, the game loop and logic is shown below:

```
while(1)
{
  while(Game_Paused) // detect pause first
  {
    ... // let the player choose different characters
  }
  if(collision_detection()) // ends the game, however, not going
    to work in bonus round or if the character is invincible
  {
    game_end();
    ... // send data
  }
  if(!playing)
  {
    show_welcome();
    ...
    if(Start_flag)
    {
      playing = 1;
    }
  }
  if(playing)
  {
    if(level_up_flag)
    {
      ...
    }
    if(bonus_round)
    {
      ...
    }
    else if(final_round)
    {
      ...
    }
    else if(game_level == 1) // different levels of playing
    ...
  }
}
```

The complete version of the code is shown in the appendix.

In this process, we have developed our own methods and functions to carry out different task and some of them are tricky. For example, to find a way to generate

random numbers(how to pick the random seed) and ways to detect collision(once there're multiple objects moving, this can be tricky since they are able to moving towards each other but not appear in the same position of the screen determined by the refresh rate).

4 Result

Some in-game pictures are shown below:

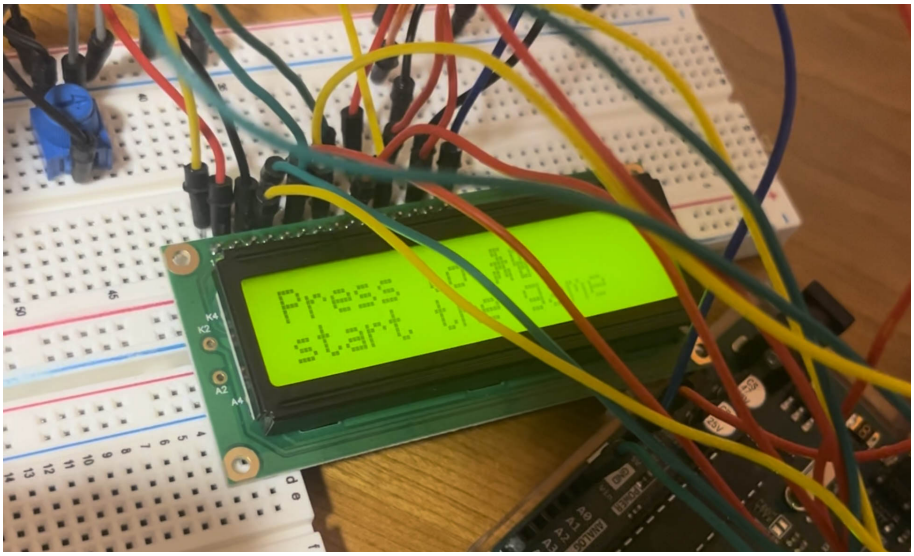

Figure 2: Welcome Page

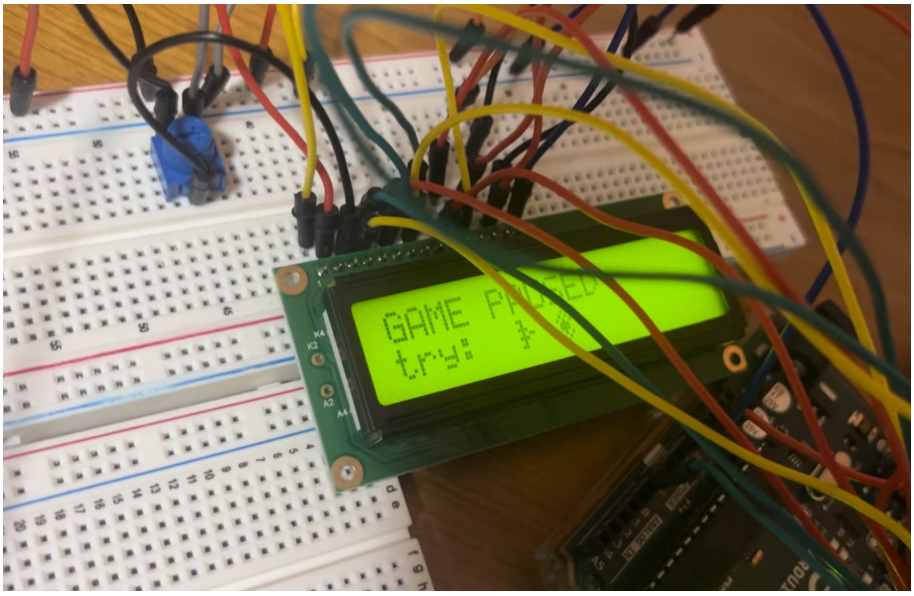

Figure 3: Game Paused Page(agent can be changed in this mode)

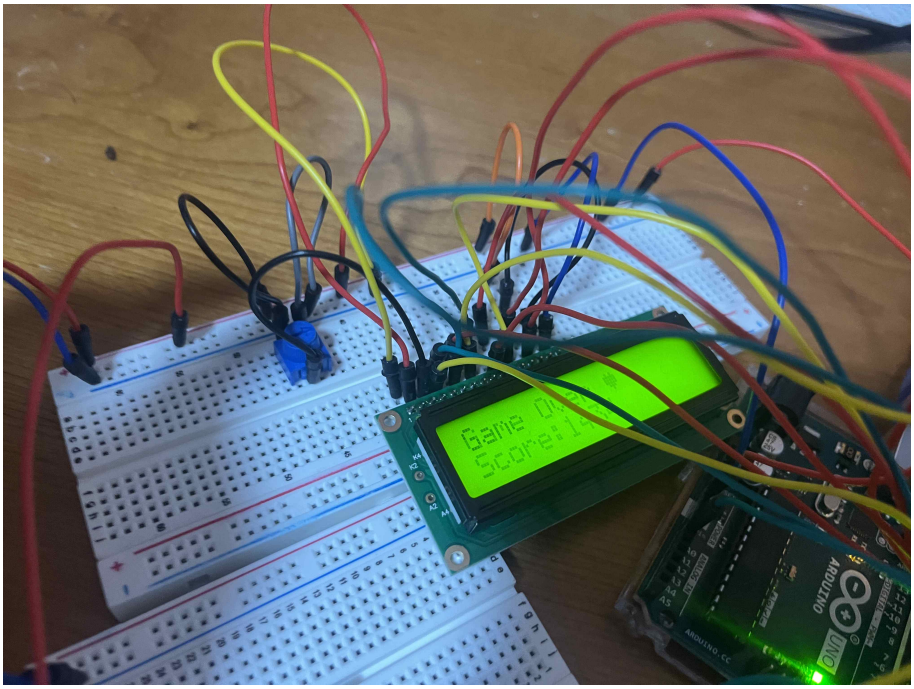

Figure 4: Game over page

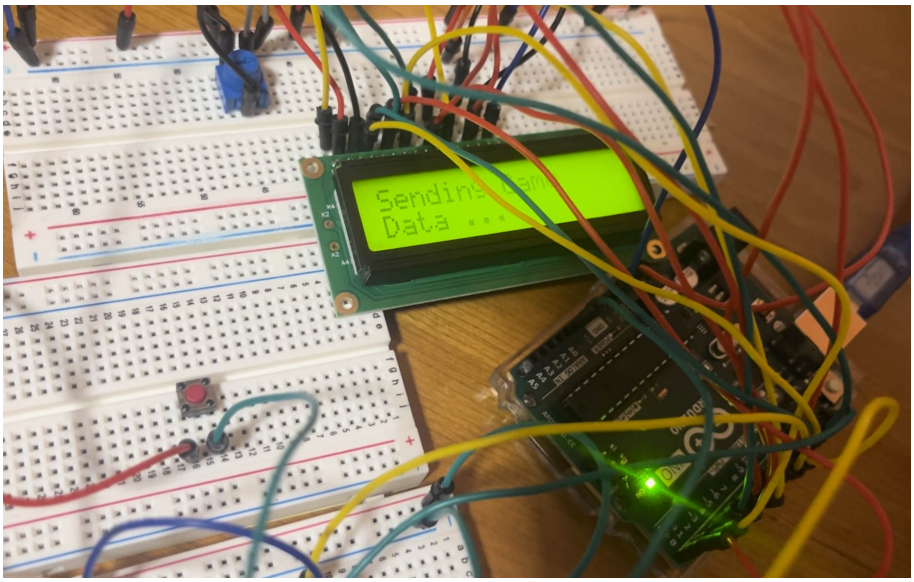

Figure 5: Sending data using serial port

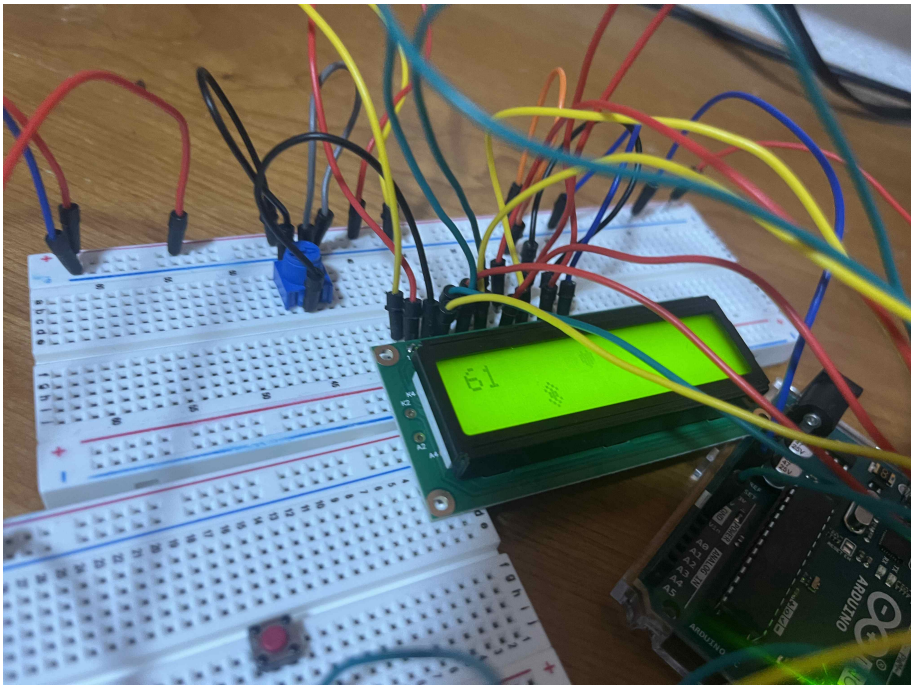

Figure 6: Bonus round

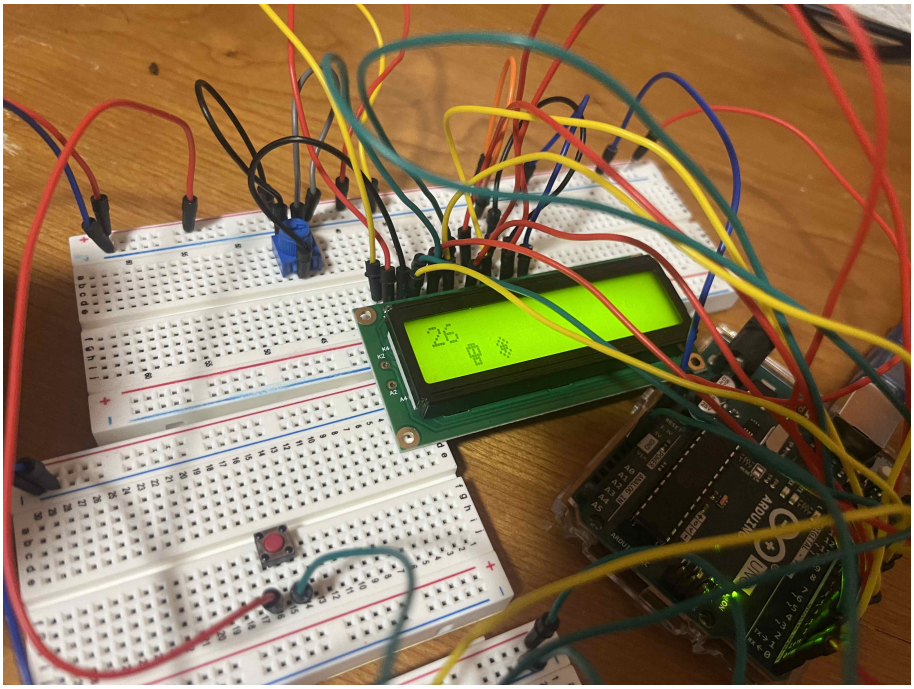

Figure 7: Invincible mode

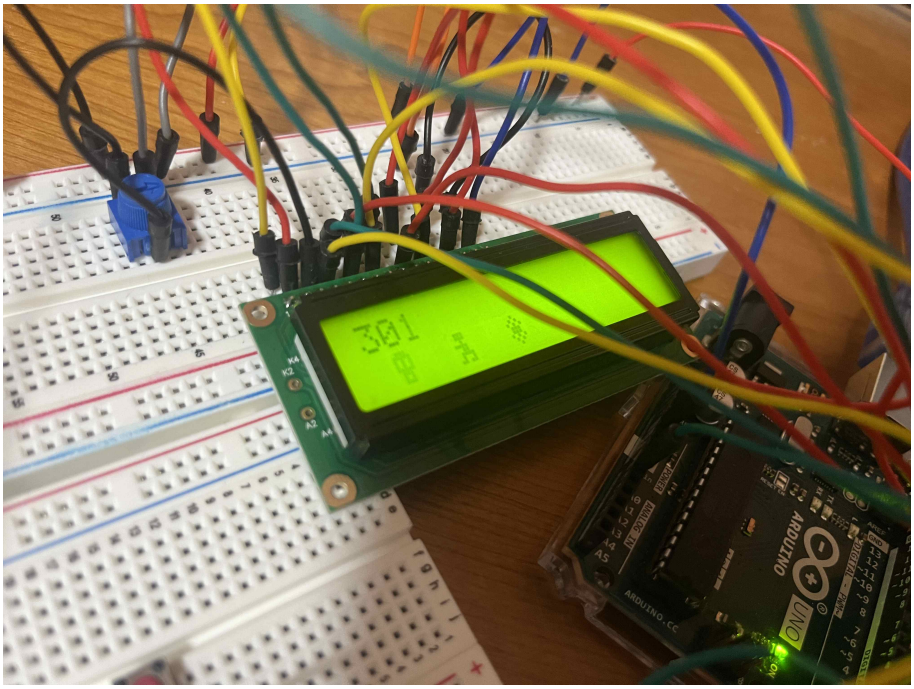

Figure 8: For the final round, the enemies can shoot bullets

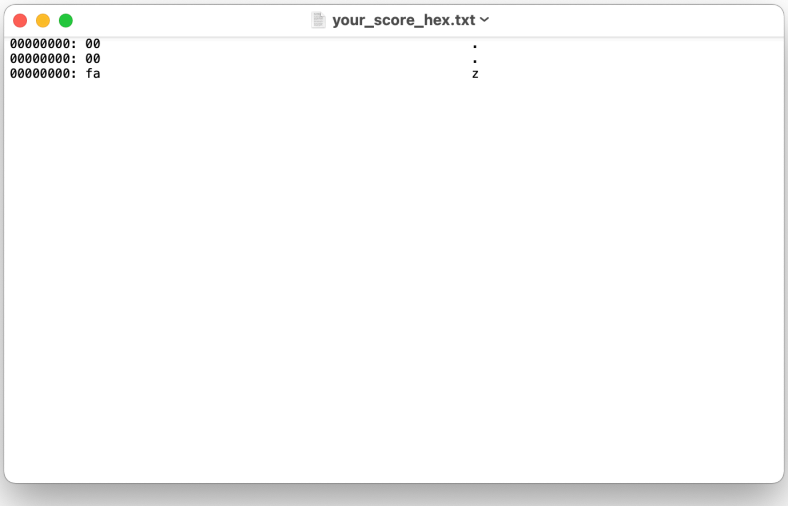

Figure 9: Score send to computer, saved to a file