Race Through the Night,
a Game Made with C++ Borland Graphics, Why?

Farrell Rafee Sudjatmiko
Stud. ID : 5024221011
Computer Engineering Department, Institut Teknologi Sepuluh Nopember

June 10, 2023

Abstract

Race Through the Night is a simple game that might attract weebs since the name itself came from a song by YOASOBI and the main premise of the game itself represents the story of the anime Oshi No Ko where Idols are facing real life social media problems. Game is created using graphics.h extension in C++

1 Introduction

In this chapter I'll explain and answer some questions regarding about problems that are encountered during the project and the back story of why I made the game.

1.1 Making a game using graphics.h on c++, Why?

Well, there actually are some benefits on doing that. Other than it is the order of our lecturer, the usage of Borland Graphics.h itself made us knowledgeable about the fundamentals of coding in C++ especially in the OOP space (Object Oriented Programming). The usage itself isn't that complicated although the there are lots of limitations including types of files that can be included, the inability to overlap audio tracks making it hard to put sound effects while maintaining the background music, and the complexity of memory allocating puting us in a hard position to mask images.

1.2 Why is it named Racing Through the Night?

Because it's based on a story where the main character ruby takes an adventure fighting through social media which is highly related to the disease called insomnia that many teens are mostly having nowadays. The name itself comes from the song called 'Yoru ni Kakeru' that translates as 'Racing into The Night'. These social media problems are really happening in real life and this game is one of the way to tell youngsters how dangerous social media is. The main menu of the game is shown in Figure 1.

Figure 1: Main Menu Interface

1.3 What is up with the usage of social media logos on the opponents?

Well it itself is a representation of the danger of social media to our lives right now. This makes it clear that the usage of the logos itself is really important. I used 5 logos as a representation including on the first stage the logo of youtube represented as a goblin and facebook as an orc boss, on the second stage we used instagram as a mage that continously attack us with electric bullets that symbolizes as this virtual reality we are living in and reddit as a knight holding a sword straight facing us that means reddit as a double-edged sword to our own lives that anything we post or share might get us back in trouble. The implementation of these are shown in Figure 2. The last boss is a manifestation of hatred among tiktok useres and that is why the colors of the final robot is colored like tiktok with white as main color and accents of cyan and light red. The boss figure is shown on Figure 3.

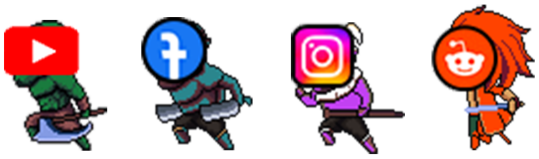

Figure 2: Social Media Logos as Opponents

Figure 3: TikTok Colors as Final Boss

2 Case Study

These are some case studies I've found related with my project :

1. Taken from *Game Development using C++* by Divya. S. N, that in the beginning, computer games were restricted due to memory limitations. As a result, game elements like maps had to be created dynamically using algorithms in real-time. The reason behind this approach was the insufficient capacity to store a vast number of pre-designed levels and artwork. By the mid-1980s, interactive fiction started to progressively incorporate more visual elements. This is connected to the making process of this game that memory of the compiler itself is limited and that is the reason of my usage of getimage instead of readimagefile for the background since a background can cost us 100-300kb of memory per loop while getimage saved those frames so it doesn't cost any.

2. Taken from Pixel Art for Game Developers by Daniel Silber, that the foundation of computer graphics involved manipulating individual pixels to create images, giving rise to the early days of Pixel Art. Initially, the control over these pixels was achieved through complex programming scripts. However, as time passed, computer art programs became more prevalent, leading to enhanced visual representations. Pixel Art emerged as a response to the technological limitations of an earlier era. Gaming systems had limited memory capacity and operated with smaller pixel dimensions compared to modern standards. Consequently, artists had to work within tight constraints of size and color, which became a technical necessity. This itself has encourage to me use pixel art as the main design of the game. I used the implementation

of pixel art using Adobe Photoshop CC 2019 which is easier for beginners and won't cost a lot of time learning a new skill shown in Figure 4.

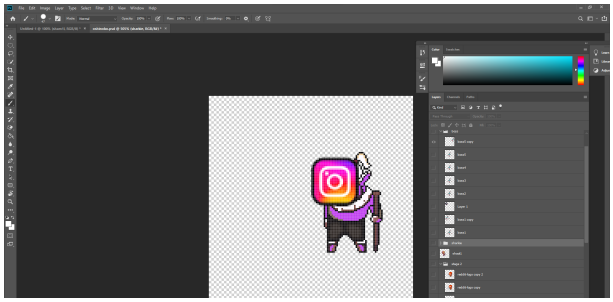

Figure 4: The Usage of Adobe Photoshop

3 The Process of Making a Game in C++ Using graphics.h (Borland Graphics Interface) Extension

In this chapter I'll explain the process of making a game in graphics.h from setting up the compiler until it's a done game.

3.1 Setting up the compiler

There are different ways on setting up a compiler to fit the graphics.h structure you can search up on your favorite platforms as well, but this is my way on doing things!

1. Firstly, when setting up the compiler make sure that your compiler is a 32-bit compiler otherwise the graphics.h library won't get a run on your text editor. I recommend using the 32-bit version of TDM-GCC Compiler which already elaborates the GNU GCC Compiler with little add-ons.

2. Then, you should download the needed files to run graphics.h on your compiler which you can find on the internet. On my case, we're already provided by our lecturer, Dr. Eko Mulyanto Yuniarno, S.T., M.T. with the required files.

3. Next, we are going to link the files into our compiler. Copy the files from the folders to your text editor file location (make sure to match the lib file to the lib folder and the header file to the include folder) then go to your compiler settings to link the libraries to your installation which includes '**-lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole3**' otherwise the library isn't going to get recognized by your text editor.

4. We're done setting up the compiler!

3.2 Making the game

Might be the most pain of your life but it is fun if your really enjoyed making it!

1. Now is the fun part, time to start making the game. To start making a window with your monitor size you can easily just run this command below

```
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
```

or you can acutally just customize it with something like this (mine is 1440X900)

```
initwindow(1440,900);
```

2. Then it is time to make the core of the game itself, the knowledge of basic C++ is really needed on this step since I won't be explaining the way to make stuff move or transform it is all up to your liking an example function of mine to make the character walk is like the command below it includes the staging of the game though.

```
void gerakl()
{
    for (int i=1;i<speed;i++)
    {
        midxt--;
        midxb--;
    }
    if (midxt < -150)
    {
        midxt = 1290;
        midxb = 1590;
        stage=stage-1;
        if (stage < 1)
        {
            midxt = -150;
            midxb = 150;
            stage = 1;
        }
    }
}
```

3. Using image backgrounds aren't as simple as putting an image and looping it thousands of times but more importantly is to make the code more efficient so that the game reduces the lag it created is to use the function getimage to memory allocate. The code below is my implementation of getimage for the stage background!

```
void *bg1,*bg2,*bg3;
unsigned int area;
void savebg()
{
    readimagefile("newbg1.gif",0,0,getmaxx(),getmaxy());
    area = imagesize(0,0,getmaxx(),getmaxy());
    bg1 = malloc(area);
    getimage(0,0,getmaxx(),getmaxy(),bg1);
    cleardevice();
    readimagefile("newbg2.gif",0,0,getmaxx(),getmaxy());
    area = imagesize(0,0,getmaxx(),getmaxy());
    bg2 = malloc(area);
    getimage(0,0,getmaxx(),getmaxy(),bg2);
    cleardevice();
    readimagefile("newbg3.gif",0,0,getmaxx(),getmaxy());
    area = imagesize(0,0,getmaxx(),getmaxy());
    bg3 = malloc(area);
    getimage(0,0,getmaxx(),getmaxy(),bg3);
    cleardevice();
}
```

4. One more important thing is to use keyboard inputs to control our characters this way is used GetAsyncKeyState as my main solution example below

```
if (GetAsyncKeyState('D'))
        {
            run();
            walkr();
            gerakr();
            dir = 0;
        }
```

5. So those are the basics of making a game core in c++ graphics.h!

4 Survey Results

Below is the survey result I've tested with 25 of my friends (either online or offline friends) regarding the game aspects which highlights the difficulty, simplicity, seamlessness, graphics, and controls!
This Graph shows that the game is rated overall good but it still needed to be improved on the controls and difficulty side since the game is actually too easy to beat and lacks of comlicated movements with more keyboard buttons.

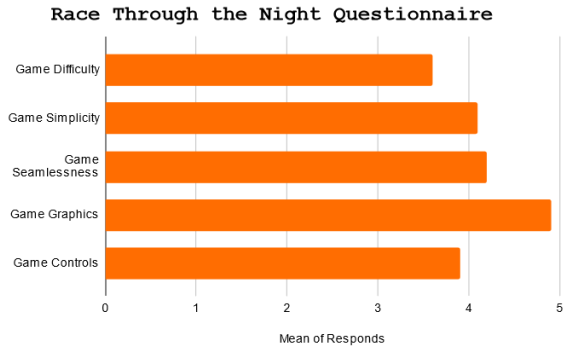

Figure 5: Questionnaire Result

5 Bibliography

Divya, S. N. (2014). Game Development using C++. International Journal of Emerging Technology and Advanced Engineering, 4(10), ISSN 2250-2459.
Silber, D. (2016). Pixel Art for Game Developers (Chapter 1).
YOASOBI. (2021, January 5). Yoru Ni Kakeru [Video file]. Retrieved from https://youtu.be/mJ1N7-HyH1A
Akasaka, A., Yokoyari, M. (Creators). (2023). Oshi no Ko (Episode 1) [TV series episode]. Oshi no Ko. Doga Kobo. (Anime)