2021

Multiplayer Game Project

REPORT – 2099 WORDS

ADAM WATSON (STUDENT)

18811629 | University of Brighton

Contents

Introduction

In this report I will describe my implementation of the project and deliver a critical review of my changes and additions to the supplied codebase, why I made those decisions, and how I believe they deliver a better user experience. I decided on a space theme for the client as I would be able to add space themed images like stars to create an attractive background and add space-oriented sounds and create an 8-bit track that fits the overall mood of the client. Below is the gameplay video that showcases that my assets have loaded in and that the sounds and music work as intended.

Gameplay Video - https://youtu.be/gkFz9j6bgOs

When deciding on how I would approach this project I looked back on previous C++ projects to get used to the language again as it would not be my first preference for coding language. The SDL documentation wiki was crucial to me when some module provided study content was not enough, but this was only an issue a few times. In my C++ project last year, I did not properly add my features as they were all added in the same cpp file, I made sure that my coding practice for this project was at the level it should be.

Implementation

This project started off as a working pong server with 2 paddles, a ball and updating scores, the client however started as just a rectangle representing the left 'player1' paddle. So, my first step was to show the right 'player2' paddle as it was not present in the client and did not have player control enabled. After checking the server and what command it receives, I noticed that the 'I' and 'K' keys were assigned to the right paddle, so basing controls off the provided codebase I was able to add control for a second player, this meant that the paddles would appear in the client and move within the server window due to the positions of the bats being within the update function. It is essential to utilise the commands for the server to apply the client changes within the server as the player could move the paddles within the client using the cpp code, but if the data was not sent to the server, the server would remain unaffected and therefore incomplete/unplayable. The GAME_DATA command is responsible for containing the inputs of the player and the actions taking place such as when the ball hits the wall to increase the point total.

```
case SDLK_i:
    send(event.type == SDL_KEYDOWN ? "I_DOWN" : "I_UP");
    break;
case SDLK_k:
    send(event.type == SDL_KEYDOWN ? "K_DOWN" : "K_UP");
    break;
```

The score was being updated on the server but on the client, there was no way of conveying score to the players. I needed to assign the score as an adjustable variable which I could display using unique fonts. To figure out how to start this process, I needed to discover which command within the server handled the score values so I could adjust this for the client side. Within the server log, the "Received: SCORES" was clearly the data I needed to work with to apply my score system. Within the scripts I created 2 variables to represent "player1score" and "player2score", this is like the way the game data structure is handled by the client as it creates variables to represent the server data, adding these points variables allows me to utilise these in various functions. I had to make sure that even though the scores were being received, there were not being updated, which leads to inconsistent results, placing the playerScore variable within an update function allows the score to constantly update with the correct score.

```
static struct playerScore {
    int Player1Score = 0;
    int Player2Score = 0;
}playerScore;
```

```
else if (cmd == "SCORES") {
    if (args.size() == 2) {
        playerScore.Player1Score = stoi(args.at(0));
        playerScore.Player2Score = stoi(args.at(1));
    }
}
```

Once this was done, I wanted to add some unique flair to the score by adding fonts to the client, I can do this by using a true type font (TTF) SDL feature which makes fonts easy to add to my application. I downloaded 'empathogenesis.ttf' as I believed this a unique font that perfectly matched the theme of my client. This process has a few steps, the first of which is initialising the font using TTF_Init. The font is loaded by setting the 'font' with a direct path to where the file is located.

```
//VISUAL

//Initialize TTF
if (TTF_Init() == -1) {
    printf("TTF_Init: %s\n", TTF_GetError());
    exit(2);
}

//Load Font
TTF_Font* font = TTF_OpenFont("Fonts/empathogenesis.ttf", 80);

//Font loaded check
if (font == nullptr) {
    std::cout << "Font creation failed " << TTF_GetError() << std::endl;
}
else {
    std::cout << "Font creation successful " << std::endl;
}
```

I ran into a lot of issues when first adding this as I was unfamiliar with some aspects of C++, I had initialised the font, but the client itself and no knowledge of it being present as I had not set it was not included in the render and set as a part of MyGame within the header file. To test this process, I added debug steps that show in the debug log so once there are more aspects in the game, I know exactly what is causing the issue, as if specific assets are not loaded properly, they return a nullptr.

```
std::string ScoreTextPlayer1 = std::to_string(playerScore.Player1Score);
std::string ScoreTextPlayer2 = std::to_string(playerScore.Player2Score);
SDL_Color textColour = { 0, 0, 0, 255 };

SDL_Surface* SurfaceText1 = TTF_RenderText_Blended(font, ScoreTextPlayer1.c_str(), textColour);
SDL_Surface* SurfaceText2 = TTF_RenderText_Blended(font, ScoreTextPlayer2.c_str(), textColour);

if (SurfaceText1 != nullptr || SurfaceText2 != nullptr) {
    SDL_Texture* TextTexture1 = SDL_CreateTextureFromSurface(renderer, SurfaceText1);
    SDL_Texture* TextTexture2 = SDL_CreateTextureFromSurface(renderer, SurfaceText2);


    if (TextTexture1 != nullptr || TextTexture2 != nullptr) {
        int a, b;
        int c, d;
        SDL_QueryTexture(TextTexture1, NULL, NULL, &a, &b);
        SDL_QueryTexture(TextTexture2, NULL, NULL, &c, &d);

        SDL_Rect d1 = { 50, 50, a, b };
        SDL_Rect d2 = { 580, 50, c, d };

        SDL_RenderCopy(renderer, TextTexture1, NULL, &d1);
        SDL_RenderCopy(renderer, TextTexture2, NULL, &d2);

        SDL_DestroyTexture(TextTexture1);
        SDL_DestroyTexture(TextTexture2);

    }
}
```

The next step was to create a texture of the font within the renderer painting the surface to text to the render and setting the texture from that line, utilising the string conversion for the score values, I was able to adjust the colour of the text and thanks to the texture creation I was able to add debugging features to ensure nothing went wrong with placing the updating score within the client. Lastly, I needed to destroy the textures when an instance of the texture has been created as if the this were not included, a new texture would be created each time the update function which would cause a constant increase in memory usage, leading to an eventual crash.

Now that the paddle and ball positions were working within the update, and scores were working as intended with the new font, it was time to adjust the appearance of the client visually. Similarly, to the text, I had to first initialise the image formats and setup the path to the image files. I used the SDL wiki to ensure I did this in a way that was supported and would not cause any unnecessary issues. Each file format needed to be separately initialised to ensure direct support was enabled. I based my client around a space theme, so I added a meteor image and coded a way for the image to constantly follow the position of the ball. I did this by creating the texture and assigning it the MeteorImage that was set in the header file and initialised in the cpp file, it is then applied to the renderer and instead of setting pre-determined co-ordinates for the image to be applied to, it uses the co-ordinates of ball which makes the image follow the balls position when it changes. Like the meteor image, I added images to each paddle in the same process, however this caused render issues which I will go into more detail in the critical review section.

```
int main(int argc, char** argv) {

    //Load image format support
    //Supply users with errors
    int loadImg = IMG_INIT_JPG | IMG_INIT_PNG | IMG_INIT_TIF | IMG_INIT_WEBP;
    int initialImg = IMG_Init(loadImg);
    if ((loadImg & initialImg) != loadImg) {
        printf("IMG_Init: Initialising image process failed, images unsupported!\n");
        printf("IMG_Init: %s\n", IMG_GetError());
    }
```

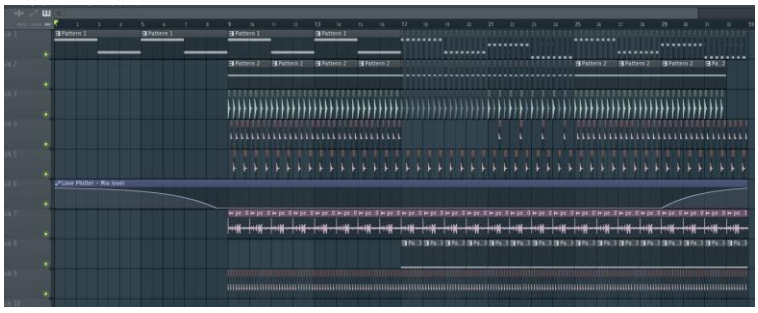
```
auto meteorTexture = SDL_CreateTextureFromSurface(renderer, MeteorImage);
SDL_RenderCopy(renderer, meteorTexture, NULL, &Ball);
SDL_RenderPresent(renderer);
```

```
auto trophyTexture = SDL_CreateTextureFromSurface(renderer, TrophyImage);

auto trophyTexture2 = SDL_CreateTextureFromSurface(renderer, TrophyImage);



if (playerScore.Player1Score > playerScore.Player2Score) {
    SDL_RenderCopy(renderer, trophyTexture, NULL, &trphy);

}
else if (playerScore.Player2Score > playerScore.Player1Score) {
    SDL_RenderCopy(renderer, trophyTexture2, NULL, &trphy2);

}
```

I wanted to have a visual way of showing the player who is winning without needing to constantly look at the score values. I did this by following the same image process as all images by initialising and setting it within the header file but with this I used the score values as an if statement to only render the image on a specific co-ordinate if the statement is true, in this case if a score is higher than the other score. The background image was an easy process as all I had to do was initialise the image, set the x and y co-ordinates to the origin and scale the image to correct height and width to the server window.

```
//Load specific audio, locate audio files

Mix_OpenAudio(22050, AUDIO_S16SYS, 2, 640);
Mix_Music* backgroundmusic = Mix_LoadMUS("Sounds/madescore.mp3");
Mix_Chunk* BallHitPaddle1 = Mix_LoadWAV("Sounds/Player1Hit.wav");
Mix_Chunk* BallHitPaddle2 = Mix_LoadWAV("Sounds/Player2Hit.wav");
Mix_Chunk* WallHits = Mix_LoadWAV("Sounds/WallHit.wav");

Mix_PlayMusic(backgroundmusic, -1);
Mix_VolumeMusic(60);

if (!backgroundmusic || !BallHitPaddle1 || !BallHitPaddle2 || !WallHits) {
    std::cout << "Music not loaded" << std::endl;
}
```

```
else if (cmd == "BALL_HIT_BAT1") {
    Mix_PlayChannel(-1, BallHitPaddle1, 0);
}

else if (cmd == "BALL_HIT_BAT2") {
    Mix_PlayChannel(-1, BallHitPaddle2, 0);
}

else if (cmd == "HIT_WALL_LEFTGAME_DATA" || cmd == "HIT_WALL_RIGHTGAME_DATA") {
    Mix_PlayChannel(-1, WallHits, 0);
}
```



Finally, to complete my ideal client, I added music and sounds. I made a short 8-bit sound score to act as the background music that plays when the client is opened as using Mix_PlayMusic, plays the sound with needing extra prompts for it to play. For the one-shot sounds, I needed to use commands from the server to receive certain actions for client to know when to play the sound. When the server receives BALL_HIT_BAT1 for example, it plays the sound assigned within the if statement.

Critical Review

This project helped in improving overall confidence in C++ coding, the successful completion of this project has provided the knowledge on how to utilise server code and manipulate client code in unison to create a fun gaming experience. Using the game data was done well to an extent as the client code was able to communicate with specific commands to allow visual and audio changes to take place to enhance the user's engagement within the game. To further enhance the competitiveness of the game, further graphical changes could be added to essentially add an end game, such as when a player's score is 20, the opposite players paddle image would change to show a damaged ship, then when it is 40, it shows the opposite paddle image on fire, and when the score is 60, the image is deleted and the game is stopped, with text indicating that a player has won.

Within this project, all features added worked as intended to a certain point. The theme of the game overall worked well as the image assets, sounds and music were all relevant in creating an engaging game level. More visuals could be added to enhance the theme such as a fire-like particle system that followed the co-ordinates of the ball, this would trigger when a player got 5 points in a row as this could be coded by either creating a variable that measures how many times "HIT_WALL_LEFTGAME_DATA" is received in the server and then applies the particle affect, or this could be hooked up to the score system by receiving "SCORES" and if a certain player score occurs uninterrupted, the effect is then triggered. This could be further adjusted by changing the colour of effect depending on how many times the 5 in a row occurs or could even link up to a change in colour within the players score.

A smaller positive aspect that came from this project was an improvement in researching skills, when the given tutorials and lectures caused me some issues, a noticeable change in how my overall coding skills have improved is that I knew specifically what to research and what to spot for when looking for the solution. This was achieved by referencing to the SDL wiki, it was a very reliable source for implementing features or when some specific features did not work, I was able to notice something I missed.

A massive issue that is immediately noticeable is the on-screen assets flickering, this only took place after adding the paddle images as beforehand it ran perfectly smooth, if the paddle images are removed from the coding, the issue is no longer present, this issue could have been caused by the textures being rendered and copied in an inappropriate way, perhaps making the assets essentially fight to render first. A solution could be ensuring that any repeating occurrences of the texture and image are destroyed when the client detects the texture is already present in the render.

In some tests, it was noticeable that the wall impact sounds would sometimes not play, this was odd as every time the server received the hit wall data, it should play the sound, this could have been an issue within the java server code as this could have meant the server was not receiving data on each wall hit. A possible client solution could be to add more conditions into the if statement, such as the client having to receive the hit wall data and the player hit data within a certain number of seconds to then perform the sound play as this ensures the ball is still present according to the game data.

Finally, the last thing I would try to improve if I had further attempted at this would be to adjust the java server code in some way to potentially allow for some additional client features and to try and make the client perform the same as the server with regards to animations like with the gentle paddle movements when the ball hits them.

Conclusion

From this project I believe my overall coding skills within cpp has greatly improved compared to the module last year, I am now more confident in my approaches to provided codebases and how a connection between a client and a server is established. If I had further time on this project I would add more visually impressive aspects like special particle systems or some sort of user interaction that allowed them to check high scores by storing the highest achieved score on the server.

Link to code scripts - https://github.com/awatson7777/cppupdate

Estimated Grade

Idea – B+

Demo – B-

Report – B+

Overall – B+

References

Font

Empathogenesis - https://freefontsdownload.net/free-empathogenesis-font-169362.htm

Images

Space - http://freevectorfinder.com/free-vectors/cartoon-space-background/

Spaceship – https://www.clipartmax.com/middle/m2i8G6G6N4i8m2N4_free-to-use-public-domain-spaceship-clip-art-space-ship-cartoon-png/

Meteor – https://www.seekpng.com/ipng/u2q8a9r5o0w7i1a9_meteor-png-asteroid-clipart/

Sounds

Wall Hit – http://www.contralogic.com/256-nes-samples/

Player Hit – http://www.contralogic.com/256-nes-samples/

Code assistance

RenderClear –

https://wiki.libsdl.org/SDL_RenderClear [SDL Wiki - Accessed 13 Jan 2021]

Loading Image Formats - https://lazyfoo.net/tutorials/SDL/06_extension_libraries_and_loading_other_image_formats/index.php [LazyFoo - Accessed 17 Jan 2021]

Audio - http://sdl.beuc.net/sdl.wiki/Mix_FadeInChannel [SDL Wiki - Accessed 13 Jan 2021]

Audio - https://www.libsdl.org/projects/SDL_mixer/docs/SDL_mixer_11.html [SDL Wiki - Accessed 13 Jan 2021]