University of Dhaka

Department of Computer Science and Engineering

Project Report:
Fundamentals of Programming Lab(CSE-1211)

Project Name:
NFS - DU DAYS

Team Members

Intesar Tahmid (AE - 38)
Tasnimul Hossain Tomal (EK - 58)

- ## Introduction

NFS - DU DAYS is a running and shooting game that has two different modes. The first one is a classic endless running one where the runner has to avoid any type of collision with the obstacles.To make it more interesting there's a feature where gamers can shoot the plane that is coming towards the runner.This will result in extra points than just avoiding.

The other mode is our special feature and is called "Treasure Hunt Mode". Likewise old fashioned treasure hunts the gamer has to go to specific destinations and those places are based on the campus of University of Dhaka. When entered in the mentioned mode the gamer will be given a brainstorming riddle with 4 options. The riddle refers to the destination where the gamer has to reach. With every right answer there will be a level up with higher difficulty. The final level refers to our very own origin, CSEDU.

- ## Objectives

There are two different significant objectives from two perspectives. From our developers' side we wanted to explore the world of gaming. Computer games have always been an entertaining part of life since our childhood. We wanted to know how it is made from scratch. Our main goal was to enrich our coding skills and learn new things over time.
From a gamer perspective we wanted to represent our university in a fun way. To answer the riddles one must have at least basic knowledge about the varsity campus. And even if one doesn't have, repetition of the levels if encountered with wrong answers will surely give them an idea.

- ## Project Features

As mentioned above, the project consists of two specific modes along with basic features that are prevalent in most games that we play.

Menu :

When the individual user will run the initiating command in the terminal this window will appear that refers to the menu.

Figure : Menu

The menu window has all the basic information on where to go for specific needs. In need of exiting directly from the menu one has to click on “EXIT” using mouse motion.

Instruction :

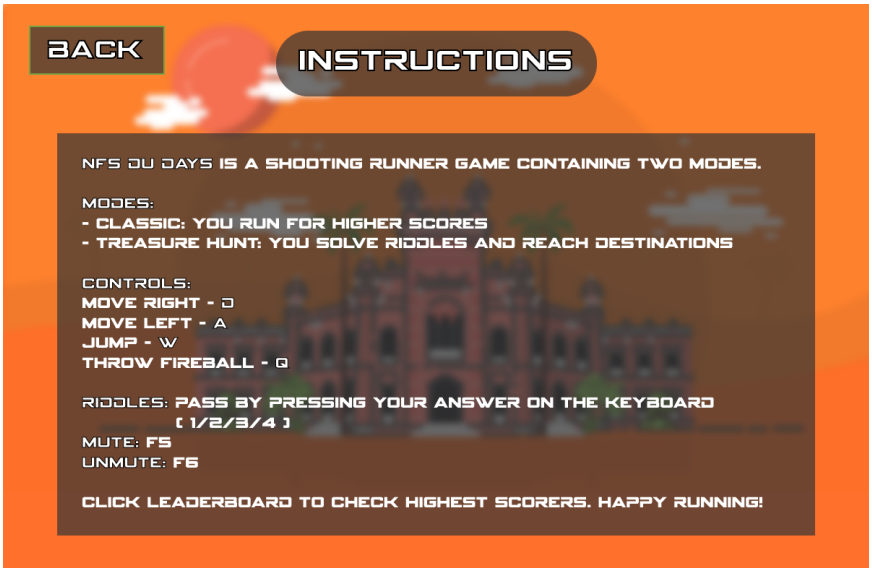

Figure : Instructions

This window has all the detailed information on how to play the game comfortably. SInce it has everything precisely, going through this only once will help the gamer to play as long as he or she wants without rechecking.

Leaderboard :

Inserted below is the leaderboard of the gamers that have played previously. The leaderboard window contains information about players with a given handle and final score.

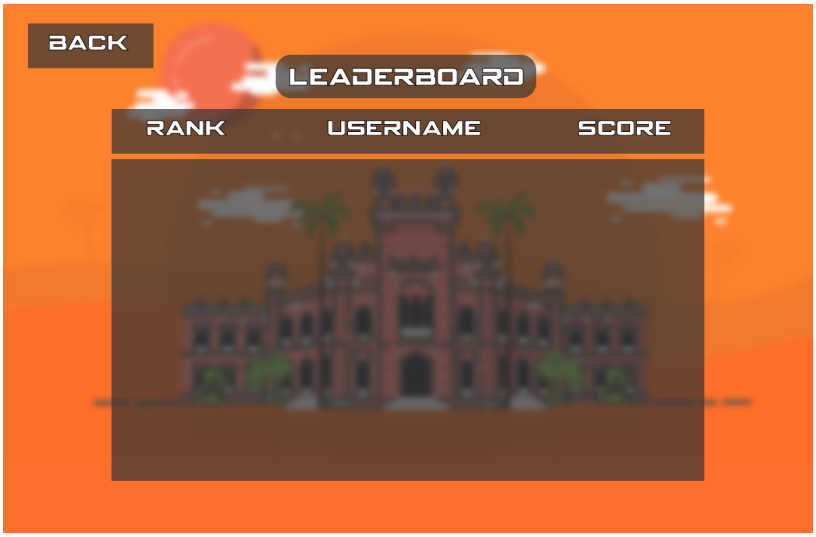

Figure : Leaderboard

We have two separate windows each containing information about ten players with the increasing order of scores.

Username :

When pressed “PLAY” the game will redirect to the username window like this below. The gamer has to input the desired handle through this window. It has been made extremely easy since one just has to type the handle and press the “Enter” key and this will automatically take the person to the next window.

Figure : Username Window

Figure : Entered a handle

Modes :

After inputting the handle the user will face the mode option where the gamer will have to decide whether he or she wants to play the endless mode or the treasure hunt mode.

Figure : Mode Selection Window

Mode selection has to be controlled using the keyboard. As said in the image, one has to press the “c” key from the keyboard and for the treasure hunt mode it is the “k” key. Pressing the “Esc” key will take one directly to the main menu.

Classic Mode :

The below image is the opening window of the Classic mode. The green bar is the health or life indicator. We provide four lives in total. And the default score is always set to zero.

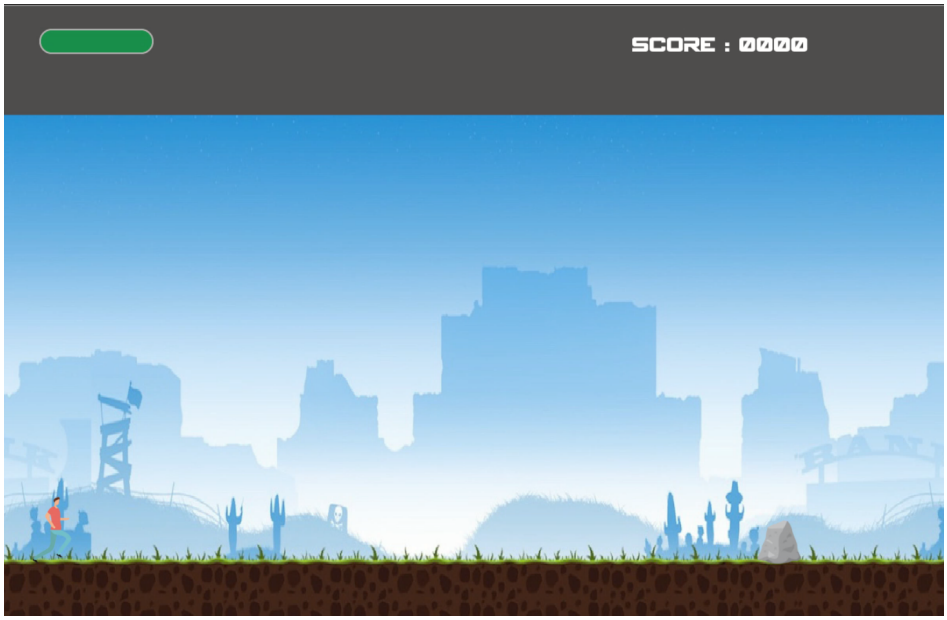

Figure : Opening Window

There will be two types of obstacles; rocks and planes. The basic rule is to avoid colliding with them. Collision will cost the gamer lives and eventually the game over. According to the instructions one can use “W”, “A”, “D” respectively for jumping, going backwards and moving forward. In each case of avoidance from collision the gamer will earn ten points.

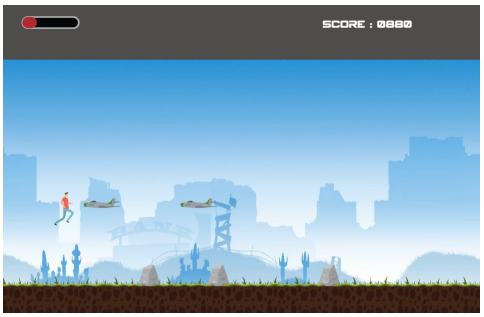

Figure : Life loss with rock collision

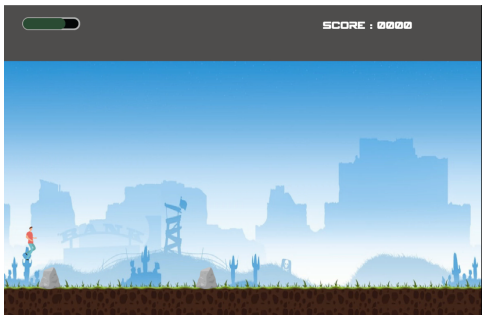

Figure : Life loss with plane collision

The gamer can shoot two fireballs at a time to destroy planes. In each destroy the reward is fifty points.

The scrolling background speed increases over time to make the game more challenging and fun.

Treasure Hunt Mode :

The main attraction of this game is this treasure hunt mode. This mode consists of seven innovative riddles each referring to a significant place of the University of Dhaka.

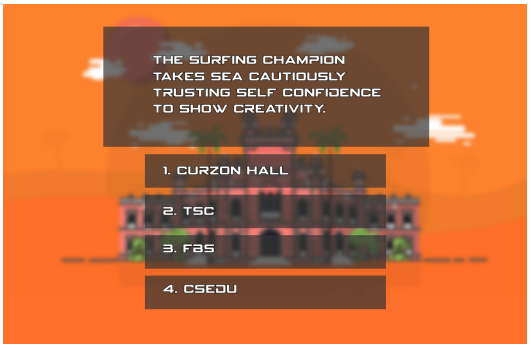

Figure : Riddle 1

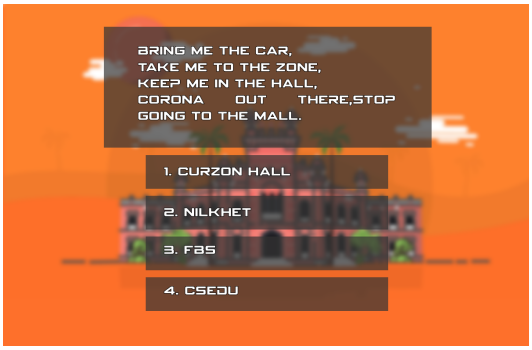

Figure : Riddle 2

The first riddle refers to the very familiar TSC and the second riddle talks about the historic Curzon Hall.

Figure : Riddle 3

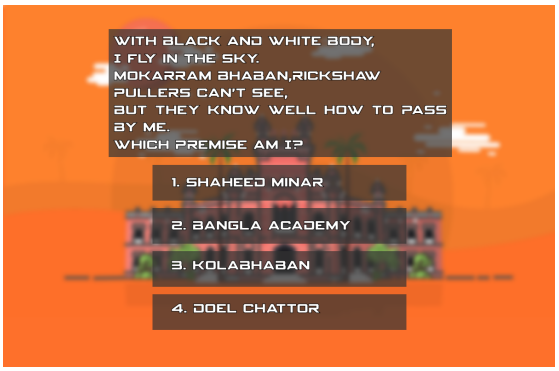

Figure : Riddle 4

In the above two images, we respectively talk about the iconic “Raju Bhashkarja” and the “Doel Chattor”.

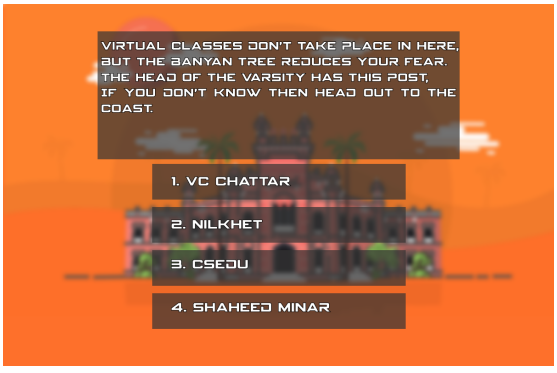

Figure : Riddle 5

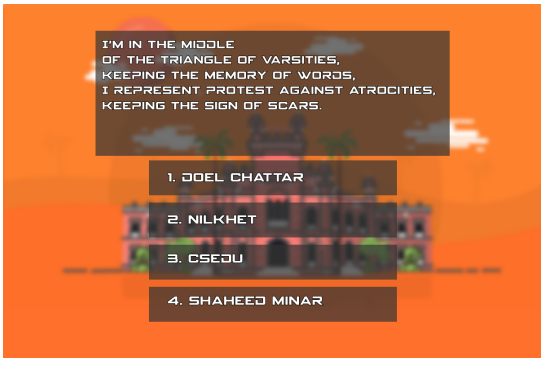

Figure : Riddle 6

In these two riddles, we mentioned the VC Chattar and the one only Shaheed Minar.

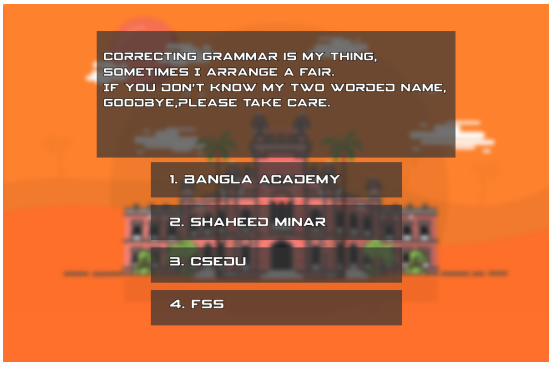

Figure : Riddle 7

The final riddle is about the hero behind the book fair, Bangla Academy.

We do believe that contemplating these riddles will provide the gamer a Dhaka University based experience.

This mode has been designed like real life treasure hunt events. When the user will face a specific riddle he or she has to choose an answer using the keyboard keys “1” or “2” or “3” or “4”. There is a flag kept in this module which stores whether the answer is correct or not but keeps the user unaware about that.

Figure : Right Answer Message

If the user answers the riddle correctly the above image is shown and the next riddle is automatically loaded on the window. This process keeps going on until the last level.

But if the gamers get the riddle wrong this window appears and asks to replay the level. The gamer has to replay it continuously to go to the next level until he gets the correct answer.

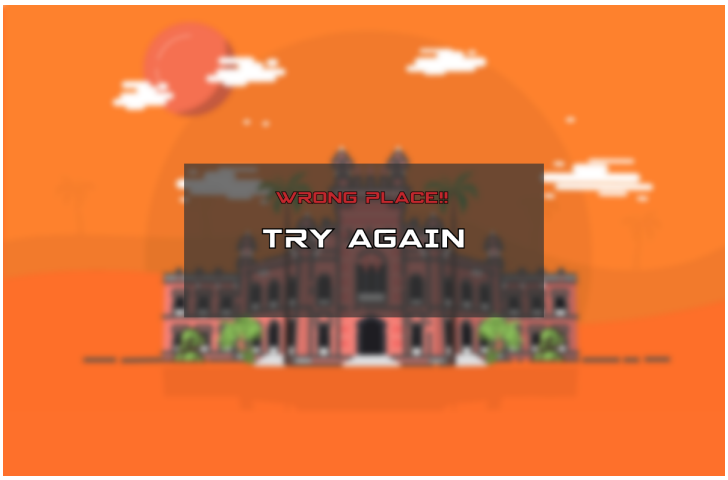

Figure : Wrong answer message

And lastly if the gamers succeed to complete all the designed levels and finish the last riddle as well the user will see this window.

Figure : Congratulations Message

This is officially the end of the game.

- Project Modules

main.cpp :

The main.cpp is the file where all the necessary functions and header files are called sequentially.

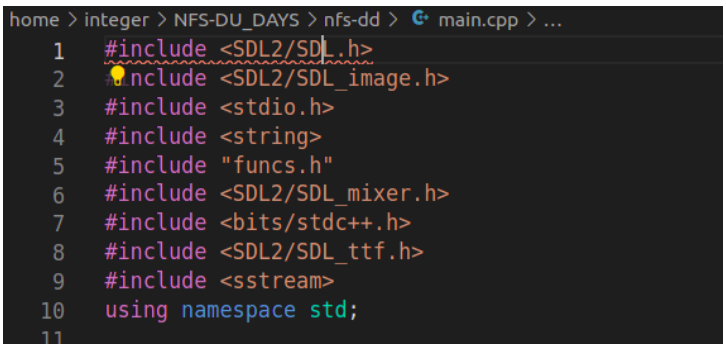

Figure : Calling the header files and all the necessary libraries

Later on, we directly started the main function according to the instructions.

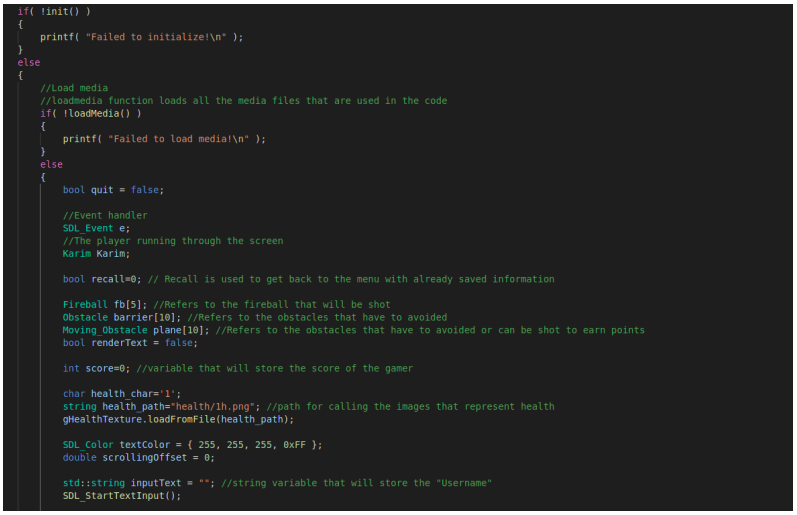

Figure : Initiation codeblock

This codeblock initiates preprocessor libraries. Then loads media and later on declares structures/classes and other necessary variables and pushes to the main loop.

This below mentioned block handles the event queue.

```
while( !quit )
{


    while( SDL_PollEvent( &e ) != 0 )
    {
        //User requests quit
        if( e.type == SDL_QUIT )
        {
            quit = true;
        }

        //Toggle music
        //Press F5 to turn the music off and F6 for turning on again
        if(e.key.keysym.sym == SDLK_F5)
        {
            musicOn = 0;

        }
        if(e.key.keysym.sym == SDLK_F6)
        {
            musicOn = 1;
        }

        if(where == MENU)
        {
            if(recall == 1)
            {
                scrollingOffset = 0; //Making the scrolling speed zero for starting over with this speed, not that of when the game was over
                startTime=0; //Flag to count the time spent from beginning again

                hello=0;

                level = 1;
                //initializes obstacles
                for(int i=0;i<10;i++)
                {
                    barrier[i].init();
                    plane[i].init();
                }

                score=0;
```

Figure : Event handler

```
if(where == MENU)
{
    if(recall == 1)
    {
        scrollingOffset = 0; //Making the scrolling speed zero for starting over with this speed, not that of when the game was over
        startTime=0; //Flag to count the time spent from beginning again

        hello=0;

        level = 1;
        //initializes obstacles
        for(int i=0;i<10;i++)
        {
            barrier[i].init();
            plane[i].init();
        }

        score=0;

        health_char='1';
        health_path="health/1h.png";
        gHealthTexture.loadFromFile(health_path);

        Karim.Life = 100; // Providing full life when restarting the game

        renderText = false;
        inputText = " ";
        Karim.init(); //Initializing the running character again

        SDL_StartTextInput();
        recall = 0;
    }

    handleMenuEvent(e);
}
```

Figure : Recalling codeblock

This code block sets all the information to default value when the game is over and the gamer has to start over again.

```
if(where == MENU)
{

    if(musicOn)
    {
        if( Mix_PlayingMusic() == 0 )
        {
            //Play the music
            Mix_PlayMusic( gMenuMusic, -1 );

        }
    }
    else
    {
        Mix_HaltMusic(); //This library function stops the music

    }

    if(whereInMenu == PLAY){
        SDL_SetRenderDrawColor( gRenderer, 255, 255, 255, 255 );
        SDL_RenderClear( gRenderer );

        //Render background
        gMenuPlayTexture.render( 0, 0 );
        gMenuPlayTexture.render( 0 + gMenuPlayTexture.getWidth(), 0 );
        SDL_RenderPresent( gRenderer );
    }
    if(whereInMenu == INSTRUCTION){
        SDL_SetRenderDrawColor( gRenderer, 255, 255, 255, 255 );
        SDL_RenderClear( gRenderer );

        //Render background
        gMenuInsTexture.render( 0, 0 );
        gMenuInsTexture.render( 0 + gMenuInsTexture.getWidth(), 0 );
        SDL_RenderPresent( gRenderer );
    }
```

```
    if(whereInMenu == LEADERBOARD){
        SDL_SetRenderDrawColor( gRenderer, 255, 255, 255, 255 );
        SDL_RenderClear( gRenderer );

        //Render background
        gMenuLeadTexture.render( 0, 0 );
        gMenuLeadTexture.render( 0 + gMenuLeadTexture.getWidth(), 0 );
        SDL_RenderPresent( gRenderer );
    }
    if(whereInMenu == EXIT){
        SDL_SetRenderDrawColor( gRenderer, 255, 255, 255, 255 );
        SDL_RenderClear( gRenderer );

        //Render background
        gMenuExitTexture.render( 0, 0 );
        gMenuExitTexture.render( 0 + gMenuExitTexture.getWidth(), 0 );
        SDL_RenderPresent( gRenderer );
    }

    if(whereInMenu == DEFAULT){
        SDL_SetRenderDrawColor( gRenderer, 255, 255, 255, 255 );
        SDL_RenderClear( gRenderer );

        //Render background
        gMenuTexture.render( 0, 0 );
        gMenuTexture.render(gMenuTexture.getWidth(), 0 );
        SDL_RenderPresent( gRenderer );
    }
}
```

Figure : Code base of menu

This specific block mentioned above calls the menu and graphically initiates the game for the users.

```
else if(where == LEADERBOARD)
{
    if(scoreLoaded == false) loadScoreFromFile();

    int Y = 260, rank = 0;

    mainFont = gFont;

    SDL_SetRenderDrawColor( gRenderer, 255, 255, 255, 255 );
    SDL_RenderClear( gRenderer );

    if(!whereInLeaderBoard){
        gLeaderBoard1.render( 0, 0 );
        gLeaderBoard1.render( 0 + gLeaderBoard1.getWidth(), 0 );
    }
    else {

        gLeaderBoard2.render( 0, 0 );
        gLeaderBoard2.render( 0 + gLeaderBoard2.getWidth(), 0 );
    }

    for(auto user : LeaderboardData) {
        const int total_width = 55;
        const int s_width = strlen(user.handle); //Determining the length of the handle given by the user
        const int field_width = (total_width - s_width) / 2 + s_width;
        sprintf(leaderboardScore, "%02d%*s%*s%4d",++rank,field_width + (s_width % 2), user.handle, field_width - s_width - 5 ,"", user.score);

        gLeaderBoardScores[rank - 1].loadFromRenderedText( leaderboardScore, textColor );
        gLeaderBoardScores[rank - 1].render( 221 , Y);

        Y += 37;
    }
    SDL_RenderPresent( gRenderer );
}
```

Figure : Leaderboard initiation

This specific code block initiates the leaderboard of the game.

```
else if(where == PLAY)
{


    SDL_SetRenderDrawColor( gRenderer, 255, 255, 255, 255 );
    SDL_RenderClear( gRenderer );
    gModeTexture.render( 0, 0 );
    gModeTexture.render(gModeTexture.getWidth(), 0 );
    SDL_RenderPresent( gRenderer );

    //Pressing the "Esc" key takes one back to the menu
    if(e.key.keysym.sym==SDLK_ESCAPE)
    {
        Mix_HaltMusic();
        cout<< "Back To Menu" << endl;
        where = MENU;
    }
    //Press "C" key on keyboard to enter the Classic Mode
    if(e.key.keysym.sym==SDLK_c)
    {
        cout<< "CLASSIC Mode" << endl;
        where = CLASSIC;
        startTime = SDL_GetTicks();

    }
    //Press "T" key on keyboard to enter the Treasure Hunt Mode
    if(e.key.keysym.sym==SDLK_t)
    {
        cout<< "Treasure Hunt Mode" << endl;
        where = LEVELS;
        lFlag = 0;
    }


}
```

Figure : Mode selection

This block is the mode selection part where the player gets the choice of choosing Classic or Treasure Hunt mode or whether he or she wants to get back to the main menu.

texturelib.h :

This header file consists of all the textures and media that had to be defined for specific needs.

```
//Texture Wrapper Class
class LTexture
{
    public:
        //Initializes variables
        LTexture();

        //Deallocates memory
        ~LTexture();

        //Loads image at specified path
        bool loadFromFile( std::string path );

        #if defined(SDL_TTF_MAJOR_VERSION)
        //Creates image from font string
        bool loadFromRenderedText( std::string textureText, SDL_Color textColor );
        #endif
        //Deallocates texture
        void free();

        //Set color modulation
        void setColor( Uint8 red, Uint8 green, Uint8 blue );

        //Set blending
        void setBlendMode( SDL_BlendMode blending );

        //Set alpha modulation
        void setAlpha( Uint8 alpha );

        //Renders texture at given point
        void render( int x, int y, SDL_Rect* clip = NULL, double angle = 0.0, SDL_Point* center = NULL, SDL_RendererFlip flip = SDL_FLIP_NONE );

        //Gets image dimensions
        int getWidth();
        int getHeight();

    private:
        //The actual hardware texture
        SDL_Texture* mTexture;

        //Image dimensions
        int mWidth;
        int mHeight;
};
```

Figure : LTexture

In the above mentioned image, there is a class we have declared with the name “LTexture”. LTexture is the general class used for all the textures that have been defined all through the code and header files.

```
bool LTexture::loadFromFile( std::string path )
{
    //Get rid of preexisting texture
    free();

    //The final texture
    SDL_Texture* newTexture = NULL;

    //Load image at specified path
    SDL_Surface* loadedSurface = IMG_Load( path.c_str() );
    if( loadedSurface == NULL )
    {
        printf( "Unable to load image %s! SDL_image Error: %s\n", path.c_str(), IMG_GetError() );
    }
    else
    {
        //Color key image
        SDL_SetColorKey( loadedSurface, SDL_TRUE, SDL_MapRGB( loadedSurface->format, 0, 0xFF, 0xFF ) );

        //Create texture from surface pixels
        newTexture = SDL_CreateTextureFromSurface( gRenderer, loadedSurface );
        if( newTexture == NULL )
        {
            printf( "Unable to create texture from %s! SDL Error: %s\n", path.c_str(), SDL_GetError() );
        }
        else
        {
            //Get image dimensions
            mWidth = loadedSurface->w;
            mHeight = loadedSurface->h;
        }

        //Get rid of old loaded surface
        SDL_FreeSurface( loadedSurface );
    }

    //Return success
    mTexture = newTexture;
    return mTexture != NULL;
}
```

Figure : loadfromfile() function

This above function enlists all the paths defined under the functions using SDL_Surface.

```
#if defined(SDL_TTF_MAJOR_VERSION)
bool LTexture::loadFromRenderedText( std::string textureText, SDL_Color textColor )
{
    //Get rid of preexisting texture
    free();

    //Render text surface
    SDL_Surface* textSurface = TTF_RenderText_Blended( mainFont, textureText.c_str(), textColor);
    if( textSurface != NULL )
    {
        //Create texture from surface pixels
        mTexture = SDL_CreateTextureFromSurface( gRenderer, textSurface );
        if( mTexture == NULL )
        {
            printf( "Unable to create texture from rendered text! SDL Error: %s\n", SDL_GetError() );
        }
        else
        {
            //Get image dimensions
            mWidth = textSurface->w;
            mHeight = textSurface->h;
        }

        //Get rid of old surface
        SDL_FreeSurface( textSurface );
    }
    else
    {
        printf( "Unable to render text surface! SDL_ttf Error: %s\n", TTF_GetError() );
    }


    //Return success
    return mTexture != NULL;
}
#endif
```

Figure : Text renderer

This above function renders all kinds of texts used in the game.

```
void LTexture::render( int x, int y, SDL_Rect* clip, double angle, SDL_Point* center, SDL_RendererFlip flip )
{
    //Set rendering space and render to screen
    SDL_Rect renderQuad = { x, y, mWidth, mHeight };

    //Set clip rendering dimensions
    if( clip != NULL )
    {
        renderQuad.w = clip->w;
        renderQuad.h = clip->h;
    }

    //Render to screen
    SDL_RenderCopyEx( gRenderer, mTexture, clip, &renderQuad, angle, center, flip );
}
```

Figure : Renders loaded texture

This function renders all kinds of loaded textures.

```
void Obstacle::render()
{
    gObstacleTexture.render(mPosX,mPosY);
}

void Moving_Obstacle::render()
{
    gPlaneTexture.render(mPosX,mPosY);
}

void Fireball::render()
{
    gFireballTexture.render(mPosX,mPosY);
}
```

These functions are declared here from other libraries as they include texture declared in this scope.

```cpp
bool checkCollision( SDL_Rect a, SDL_Rect b )
{
    //The sides of the rectangles
    int leftA, leftB;
    int rightA, rightB;
    int topA, topB;
    int bottomA, bottomB;

    //Calculate the sides of rect A
    leftA = a.x;
    rightA = a.x + a.w;
    topA = a.y;
    bottomA = a.y + a.h;

    //Calculate the sides of rect B
    leftB = b.x;
    rightB = b.x + b.w;
    topB = b.y;
    bottomB = b.y + b.h;

    //If any of the sides from A are outside of B
    if( bottomA <= topB )
    {
        return false;
    }

    if( topA >= bottomB )
    {
        return false;
    }

    if( rightA <= leftB )
    {
        return false;
    }

    if( leftA >= rightB )
    {
        return false;
    }

    //If none of the sides from A are outside B
    return true;
```

Figure : Collision detection algorithm

This function detects any kind of collision between the runner and the obstacles.

Karim.h :

This header file is designed specifically for the runner of both the modes used in our game.

```cpp
class Karim
{
    public:
        //The dimensions of the Karim
        static const int Karim_WIDTH = 63;
        static const int Karim_HEIGHT = 85;
        int mPosX, mPosY;
        //Maximum axis velocity of the Karim
        static const int Karim_VEL = 10;
        int jumped;
        int fireball_throwed;
        int Life;

        int hitten;
        //Initializes the variables
        Karim();

        //Takes key presses and adjusts the Karim's velocity
        void handleEvent( SDL_Event& e );

        //Moves the Karim
        void move();
        void jump();
        void init();
        //Shows the Karim on the screen
        void render(SDL_Rect* currentClip);
        double initial ;
        double velocity;
        double gravity;
        SDL_Rect Karim_Rect;
    private:
        //The X and Y offsets of the Karim


        //The velocity of the Karim
        int mVelX, mVelY;
};
```

Figure : class for Karim

The above mentioned image refers to the class dedicated to Karim, the runner character. In this class we have defined all the necessary variables that are related to the runner. Apart from declaration we have called all the necessary functions as well regarding movement, shooting, life and so on.

```
void Karim::init()
{
    mPosX = 40;
    mPosY = GROUND;

    //Initialize the velocity

    mVelX = 0;
    mVelY = 0;
    jumped=0;
    hitten = 0;
    fireball_throwed=0;
    initial = -13;
    velocity=initial;
    gravity=0.5;
    Karim_Rect.w=Karim_WIDTH;
    Karim_Rect.h=Karim_HEIGHT;
    Life=100;

}
```

```
Karim::Karim()
{
    //Initialize the offsets
    mPosX = 40;
    mPosY = GROUND;

    //Initialize the velocity
    hitten = 0;
    mVelX = 0;
    mVelY = 0;
    jumped=0;
    fireball_throwed=0;
    initial = -13;
    velocity=initial;
    gravity=0.5;
    Karim_Rect.w=Karim_WIDTH;
    Karim_Rect.h=Karim_HEIGHT;
    Life=100;

}
```

Figure : Two functions that initializes the runner activity

These two images contain the default information on how the runner should start every time when the user enters into any specific mode.

```
void Karim::jump()
{
    if(velocity >= abs(initial)+1)
    {
        velocity=initial;
        jumped=0;
    }
    else
    {
        mPosY+=velocity;
        velocity+=gravity;

    }
}
```

```
void Karim::move()
{
    //Move the Karim left or right
    if(jumped)
    {
        jump();
    }

    mPosX += mVelX;
    mPosX=max(mPosX, 40 );
    mPosX=min(mPosX, SCREEN_WIDTH/2 );
    //If the Karim went too far to the left or right


    //Move the Karim up or down
    mPosY += mVelY;

    //If the Karim went too far up or down
    if( ( mPosY < 0 ) || ( mPosY + Karim_HEIGHT > SCREEN_HEIGHT ) )
    {
        //Move back
        mPosY -= mVelY;
    }
    Karim_Rect.x=mPosX;
    Karim_Rect.y=mPosY;
}
```

Figure : Runner's jump and movement controlling functions

Change of activity when the gamer wants to jump or move is defined using the "jump()" and "move()" functions respectively. Necessary default information is enlisted in here as well.

The below images are two conditions inside the "handleEvent()" function. The first one with SDL_KEYDOWN occurs when one of the mentioned keys is pressed. And the second one refers to the situation when that specific key is released.

obstacle.h :

```
class Obstacle
{
    public:
        int Obstacle_WIDTH = 55;
        int Obstacle_HEIGHT = 55;
        double mPosX, mPosY;
        int mVelX;
        int flag_of_obstacle;
        int hitten;
        //Maximum axis velocity of the Obstacle
        static const int Obstacle_VEL = -4;
        //double Obstacle_VEL = -(8+hello);

        //Initializes the variables
        Obstacle();

        //Takes key presses and adjusts the Obstacle's velocity
        //Moves the Obstacle
        void move(double speed);
        void close();
        void init();
        //Shows the Obstacle on the screen
        void render();
        SDL_Rect Obstacle_rect;

    private:
        //The X and Y offsets of the Obstacle
        //The velocity of the Obstacle
        int f;
};
```

Figure : Obstacle Class

This class defines the obstacle that we plan to render. All the necessary variables are declared and the functions are called accordingly as well.

moving_obstacle.h

```
class Moving_Obstacle
{
    public:
        int Moving_Obstacle_WIDTH = 55;
        int Moving_Obstacle_HEIGHT = 55;
        double mPosX, mPosY;
        int mVelX;
        int flag_of_obstacle;
        int hitten;

        char path_of_Moving_Obstacle;
        //Maximum axis velocity of the Moving_Obstacle
        static const int Moving_Obstacle_VEL = -10;

        //Initializes the variables
        Moving_Obstacle();

        //Takes key presses and adjusts the Moving_Obstacle's velocity
        //Moves the Moving_Obstacle
        void move(double speed);
        void close();
        void init();
        //Shows the Moving_Obstacle on the screen
        void render();
        SDL_Rect Moving_Obstacle_rect;

    private:
        //The X and Y offsets of the Moving_Obstacle
        //The velocity of the Moving_Obstacle
        int f;
};
```

Figure : Class definition for moving obstacle

This class defines the moving obstacle. Like before all the necessary variables are declared and functions are called sequentially.

fireball.h :

```
class Fireball
{
    public:
        static const int Fireball_WIDTH = 20;
        static const int Fireball_HEIGHT = 20;
        int mPosX, mPosY;
        int mVelX;
        int flag_of_fireball;
        //Maximum axis velocity of the Naruto
        static const int Fireball_VEL = 12;

        //Initializes the variables
        Fireball();

        //Takes key presses and adjusts the Naruto's velocity
        //Moves the Naruto
        void move();
        void close();
        //Shows the Naruto on the screen
        void render();
        SDL_Rect Fireball_rect;

    private:
        //The X and Y offsets of the Naruto
        //The velocity of the Naruto
        int f;
};
```

Figure : Class for fireball

This class is defined for the fireball that will be shot by the gamer.

funcs.h :

This file is the combination of all the small functions that were needed to run the code perfectly. From menu to leaderboard and character movement, all the small functions are called in here with proper information inside them.

- ## Team Member Responsibilities

To expedite the execution process we planned the work distribution at first and evenly took responsibility for specific modules. The responsibilities are mentioned below according to the plan.

Intesar Tahmid :

- Developed the functions for the “Main Menu” page.
- Coded the sprite animation of the runner
- Made the scrolling background more lively where it was made sure that the scrolling speed increases over time.
- Build the classes and structures needed for the character movement and the relevant header file as well.

- Created all the riddles for the treasure hunt mode.
- Coded the scoring option and the “Leaderboard” using necessary structures and classes that enlists the highest scorers in the leaderboard.
- Developed health reduction in terms of collision.
- Wrote the project report including all necessary information about the code files.

Tasnimul Hossain Tomal :

- Connected the functions in the menu page so that the user can switch between them easily.
- Prepared all the images using illustrator.
- Coded both types of obstacles for two modes and made sure that the scrolling speeds looked realistic.
- Coded the levels of the treasure hunt mode.
- Prepared the header library named “func.h” which has all the functions used in the code.
- Coded the shooting of the runner and the header file for that.
- Took care of all the font related works like username, score and health graphics.

Platform, Library & Tools

The whole project has been coded by using the C++ language. For the graphical user interface we have used the SDL(Simple DirectMedia Layer) library. It is a cross platform development library that is designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D.It can be used to make animations and video games. The latest version SDL2 has been used for better portability and comfortable usage. Under the SDL2 we have used the image library, font library and music library altogether to bring together an entertaining game. For development we have used basic text editors like gedit, sublime text editor and vscode and the terminal for the compiling and running.

- ## Limitations

  From an honest perspective we believe there could be better graphics for the gameplay. Though generally endless running games have scrolling backgrounds and themes like this of ours. We think the user interface could be made more lively with more effort. The increase of variety in the obstacles could spice up the gameplay experience too.

  But the amount of regression considering the desired features that were left unimplemented is not that much. It is because we always tried to be realistic considering our own skill set and planned accordingly.

- ## Conclusions

  In the beginning we were very excited about making a game as the project. Our enthusiasm overwhelmed while planning the features. Needless to say, we were pretty skeptical about the implementation since graphics related programming was not a familiar topic to us. Therefore we decided to set smaller goals and implement them accordingly in order. That is when building the code base became a bit easier since we arranged the application from easy to hard and kept working in that way. Fortunately, we could achieve the goals that we had set for our project. Despite treasure hunt being the core feature we were struggling to execute that perfectly. But, after extensive research and effort we could improve the code and could do everything accurately. In essence we are very content about meeting the goals that we had set.

  The journey has been with lots of ups and downs as we did not have any sort of experience regarding the graphics programming. There were numerous tutorials online but very few were user friendly enough to teach from the basics. And so, for every single small detail such as : timing, animation, movement and so on, we had to run every single code after going through the tutorial. Later on we had to write code according to our own needs and debug it for hours. To us the most difficult part was bringing everything together to make the game work flawlessly. There are innumerable SDL functions that can be used for specific needs. But making sure that we are using them sequentially without errors is the true goal of this project. And after implementing we do believe that we have learned how to do that. We learned to make the best use of the existing library according to needs. Learning to use according to need is far better than memorizing extensively and then struggling to decide which to use and where to add.

  Though it is only a first year project and we surely have thousand other things to learn in the coming days to be an efficient software developer, we believe this project has worked as a seed for those. Apart from learning new library functions and coding skills we worked as a team this time. We made plans and set deadlines for us so that the workflow stays stable. Because of the pandemic we had to communicate only through online mediums. This gave us a slight experience of work from home as well. As different members we had differences in opinions

considering features. In every such case, we took time to discuss and resolve the issues with patience and respect to each other's perspectives.

Afterall despite immense struggle while coding and implementing the ideas, this project has worked as an opportunity to learn both coding and working in a team. And we are certain that these will come in handy in the near and distant future.

- Future plan

We are confident about the simplicity of this project. We do believe that the unique idea of treasure hunt will be loved by others except the varsity students as well. In our future updates the first thing we are planning to work on is bringing more realistic graphics. We plan to implement the real life images around the campus of the University of Dhaka so that gamers can connect to the places more easily.

To make the concept more familiar to players of any age we plan to use more riddles based on the world around us. Increasing levels in such a way will provide the eligibility for more popularity for sure. And so we have a plan to publish it with more features and confirmed stability. At present it is already pretty easy to play on a linux machine. One has to run a few commands in the terminal to install the libraries and then type “bash run.sh” to start playing. We plan to update the github repository with more features and then publish it publicly in Ubuntu Software Center. Considering the quality of games we have seen in the software center, with some extended features and a stable online we can surely turn out a simple and very fun game to play.

Repositories

GitHub Repository: *https://github.com/tomal66/nfs-dd*

Youtube Video: *https://youtu.be/pnS1ACaHIl4*

References

- Lazyfoo - https://lazyfoo.net/tutorials/SDL/index.php
- Parallelrealities - https://www.parallelrealities.co.uk/tutorials/#shooter
- SDL Wiki - https://www.libsdl.org/