Table of Contents

Introduction:

The aim of the given project was to introduce ourselves to the wider world of C programming in a practical manner. Though we had been taught how to handle small fragments of code through problems, this project was designed to teach us how to handle 1000+ lines of code , i.e., codes the size of real-world applications. It was also a tutorial to constructive programming.

Project Name:

Mr. Pac.

Project Objective:

The project was designed to give the user entertainment. Having worked in C programming for six months, it was an opportunity for us to implement some programming ideas in a practical application. It was also an opportunity to flex our creative muscles a bit.

The game is loosely based on old stealth action games and Pac-Man. Your objective is to collect all items in the room without getting detected by the patrolling guards. Upon detection, the guards would rush to your location to capture you, resulting in a gameover state.

Project Outline:

As mentioned before, the main objective here is to collect all and avoid all. The has some linear levels with one clear goal. It requires quick reflexes and proper decision-making to completely progress through the game. At any point in the game, the player can save his/her state, but death would instantly reset everything. The game has a scoring system based on how many objects the player has collected.

Main Features:

1.Main Menu:

a) New Game Tab:

This tab would take the player to the start of the game or, if the player has saved the game, take him/her to his/her last save state.

b) High Score Tab:

This tab would show the player the top five scores of all previous playthroughs.

c) Instructions Tab:

This tab would briefly describe how to play the game and what everything meant.

d) Quit Tab:

This tab would quit the game.

2) Player Movement:

The player movement is handled by the directional keys. Upon pressing, the player would start moving in the direction of the pressed button at a fixed velocity.

3) Enemy Movement:

The enemies have two different states, which will be discussed more broadly in the following section. The first state is the Idle state, where the enemy patrols a definite part of the map in a defined axis. And the other state is the chasing state, where all the enemies will start manically chasing you upon detection.

4) Game Map:

There are two stages in this game, all of which had been generated using a text file, where "03" denotes walls and "00" denotes passable planes.

5) Coins and Score:

The scoring system depends on the coins that are scattered throughout the level. The player has to collect all the coins without dying to proceed to the next stage and to finish the game.

Special Features:

1) Enemy States:

As mentioned before, the enemies in this game have two states. The Idle state and the Alerted state.

In the Idle state, the enemy would go back and forth along an axis at a definite speed until it has detected the player. In this state, the enemy has an extended collision box, denoting the how far the enemy will be able to see. If the player lands on that box, all of the enemies in the vicinity will be alerted which would initiate the Alerted state.

In the Alerted state, the enemy would start chasing the player. The enemy would take in the position of the player as its destination and start heading towards it. On its way if it finds a wall, it would stop its movement in that the direction of the wall and start scaling the wall. While scaling, the axis would depend on where the player is. If the player is a certain distance away along the y-axis, it would scale vertically in the direction of the player. Otherwise, it would scale horizontally.

2) Save States:

At any moment of the game, the player can save his/her progress by pressing the "S" key of the keyboard. This would save the player's location, the collected coins and bring the game back to the main menu screen. The player can comeback to this save file at any preferable time.

Coding Challenges:

1) Enemy Behavior:

This was the biggest challenge we faces when designing the game. The Idle state was relatively easy to design, but the enemy chasing when they would alerted was particularly difficult, as the enemies constantly went through walls in the y-axis when chasing. In the end, we resolved this matter a bit by constraining the enemy to follow the player in a defined axis and ignoring the other axis when the player is a certain distance away along the y-axis.

2) Sorting and updating high scores:

The scores were all saved into text files. Upon selection of the High Score tab in the menu, the designated function would read from the designated file and show the scores using the text rendering option of SDL2.

In case of updating, after getting the score, a function would compare this score with the previous entries of the file and add it if the score is more than a previous high score and sort it.

3) Save File:

The save file was done using a text file. The coins and the players position have all a designated value assigned to them. When the game is saved, these values are saved and the next game will start according to those values. There is also another value that denotes which stage the player is on and the game would start from that file.

Graphical Interface:
(MENU PICS)

Main Game:
(GAME PICS)

Project Layout and Code:

SDL2 is a very versatile language which allowed us to flex our creative muscles. Though we didn’t really focus on the graphical part that much, the addictive gameplay will surely hook players into this game. In this section, we will discuss the functions, the variables and a game chart to explain the coding aspect of our project.

Resources used:

We were tasked to make something using C/C++ and SDL2 was our supporting tool. SDL stands for Simple Directmedia Layer. It is a very powerful and versatile tool that enabled us to do creative and somewhat ambitions things as our project.

Functions used in the game:

bool init(): Initiates the game

bool loadMedia(Tile* tiles[]) : Load all the sprites and textures

void close(Tile* tiles[]) : Deletes all rendered objects upon closing

bool CheckCollision(SDL_Rect a, SDL_Rect b) : Checks collision between any two objects

void MouseEvent(SDL_Event &e) : Handles mouse events

bool touchesWall(SDL_Rect box, Tile* tiles[]) : Collision checker for the walls

bool setTiles(Tile *tiles[]) : Renders the walls
bool enterName() : Function for name input

int game() : Loads the game map

int scorefunc() : Handles scores

int menu() : Handles all menu functions

Classes and functions of those classes in the game

Tile

Tile(int x, int y, int tileType) : Initializer

void render() : Renders the tiles on a texture

int getType() : For selecting impassable tiles

SDL_Rect getBox() : Gets collision box

Texture

Texture() : Loads Texture

~Texture() : Unloads texture

bool LoadFromFile(string path) : Loads file from a folder

#if defined(_SDL_TTF_H) || defined(SDL_TTF_H)
bool LoadFromRenderedText(std::string TextureText, SDL_Color TextColor);
#endif
: Loads TTF files

void free() : Frees texture

void setColor(Uint8 red, Uint8 green, Uint8 blue) : Sets color

void setBlendMode(SDL_BlendMode blending) : Sets alpha blending

void setAlpha(Uint8 alpha) : Also sets alpha blending

void render(int x, int y, SDL_Rect* clip = NULL, double angle = 0.0, SDL_Point* center = NULL, SDL_RendererFlip flip = SDL_FLIP_NONE);
:Render function

int getWidth(): Returns width of the tile

int getHeight() : Returns height of the tile

<u>Player</u>

static const int PLAYER_WIDTH = 20
static const int PLAYER_HEIGHT = 20

Player(double, double) : Initializer

void HandleEvent(SDL_Event& e) :Handles input

void Move(Tile *tiles[]) : Movement function

void render() : Render function

<u>Enemy</u>

static const int ENEMY_HEIGHT = 20;
static const int ENEMY_WIDTH = 20;

Enemy(int , int) : Initializer

void Move(Tile *tiles[], char arr[][1000]) : Movement function

void render() : Render function

void Detected(SDL_Rect& a) : Sees if player has come in sight

<u>Items</u>

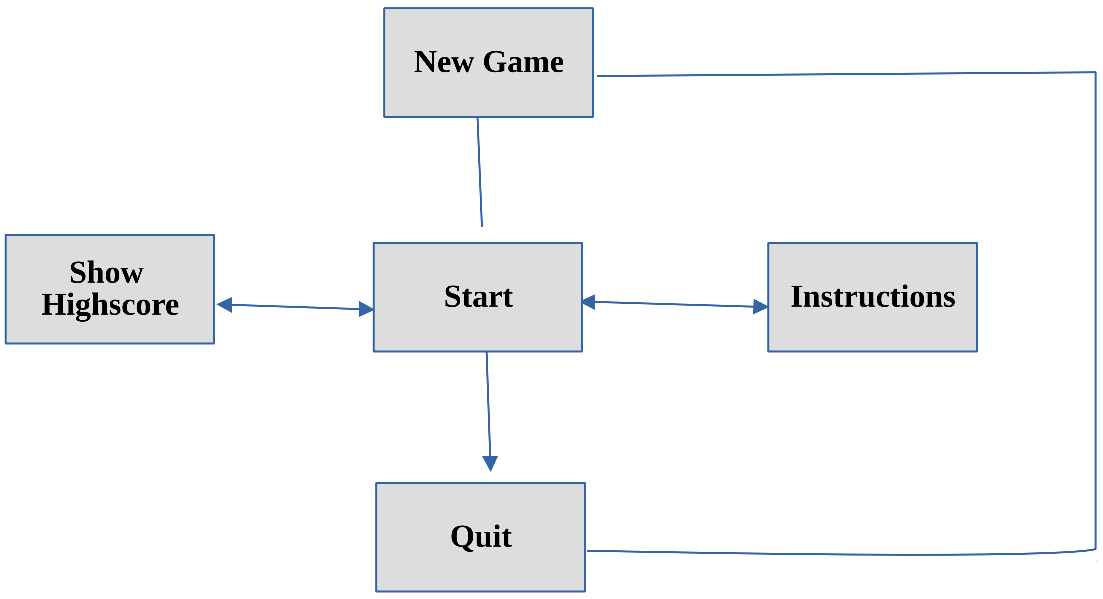
Items(int x, int y) : Initializer

void render() : Render function

bool Destroy() : Destroys itself if player collects it

bool Collect() : Would keep rendering if this returns true

Menu Flow Diagram:

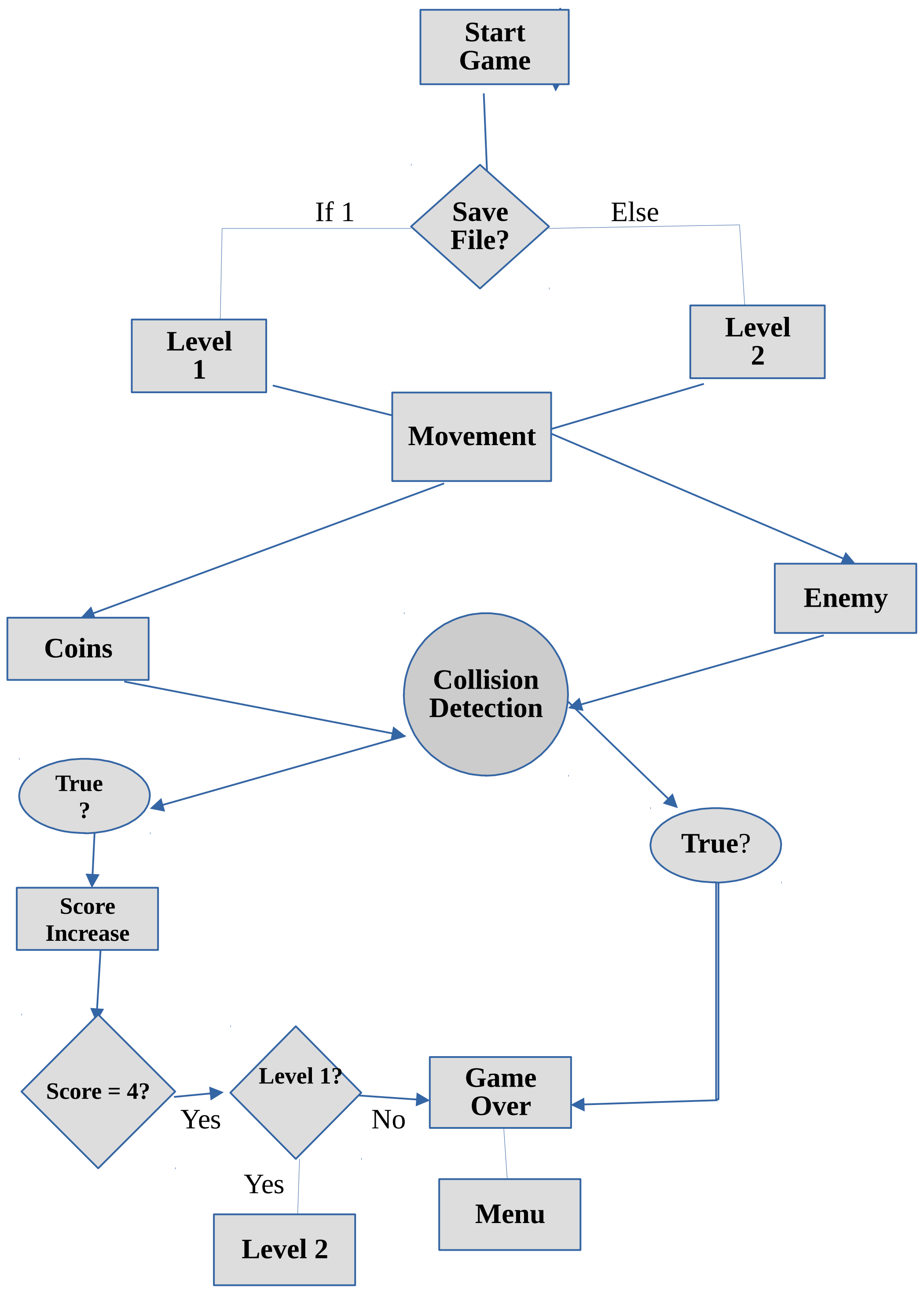


A Brief Explanation:

At first, the player would be greeted to the menu and would be shown the four tabs: New Game, Highscore, Instructions and Quit. The clicking onto those tabs would enable the functions game(), showhs(), instructfunc() and close() respectively.

From the game() function, the player can also access the menu after saving or dying in the game.

<u>Game Flow Diagram:</u>



A Brief Explanation:

After the game starts, the level will be selected based on which level had been saved in the previous save. If there were none, the game would start from level 1. After that, the player will progress and collect coins. Collecting four coins will take the player to Level-2 if he/she was on Level-1 or to the main menu otherwise. Collision with the enemy will result in a gameover state and the player will be sent to the main menu.

Overview and conclusion:

The project was a very good way to encourage young students to develop their skills and make something useful with their knowledge. This project is a perfect example of all the exceptional things that can be done with some basic C programming.

This project will also encourage young minds who want to be a game developer in the future. The project promotes originality and creativity and the skill of working in a team. The project also summarizes the basic tools of C language as they had to be implemented while making the game.

Source Code:

The main() function:

```
int main(int argc, char const *argv[])
{
	int playing = 1;
	int a, _game_ = 0;
	int savestart;
	if(!init())
	{
		printf( "Failed to initialize!\n" );
	}
	else
	{
		bool nametaken;

		name.clear();
		nametaken = enterName();
		while(playing)
		{
			a = menu();
			cout<< a;

			if(a==3)
			{
				a=instructfunc();

				if(a)
					continue;
			}


			if(a == 1)
			{
				if(nametaken)// && !saveStart)
					a = game();
			}

			if(a == 10)
				a = game2();

			if(a == 2)
			{
				show_hs();
				continue;
			}



	if (a == 0)
		    {
```

```
                SHESH = 5;
                int a = game();
                closeMenu();
                break ;
            }

            a = scorefunc();

            if (a == 0)
            {
                SHESH = 5;
                int a = game();
                closeMenu();
                break ;
          }
                cout << Score << endl;
          }
      }
      closeMenu();

      printf("END\n");

      return 0;
}
```

The menu() function:

```
int menu()
{
    int flag = 1;
    {
        //Load media
        if( !loadMediaStart() )
        {
            printf( "Failed to load media!\n" );
        }
        else
        {
            //Main loop flag
            bool quit = false;

            //Event handler
            SDL_Event e;
            SDL_Texture *selTexture = SDL_CreateTexture( gRenderer,
    SDL_PIXELFORMAT_RGBA8888, SDL_TEXTUREACCESS_TARGET, 300, 80 );
            SDL_Rect optRect, optRect2;

            //While application is running
            while( !quit )
            {
                //Handle events on queue
                while( SDL_PollEvent( &e ) != 0 )
                {
                    //User requests quit
                    if( e.type == SDL_QUIT )
                    {
                        return 0;
                    }
                    if (e.type == SDL_KEYDOWN)
                    {
                        switch( e.key.keysym.sym )
              {
                case SDLK_UP:
                flag--;
                flag %= 4;
                break;

                case SDLK_DOWN:
                flag++;
                flag %= 4;
                break;

                case SDLK_RETURN:
                cout<< flag;
                SDL_DestroyTexture (selTexture);
                selTexture = NULL;
        return flag;

    }
            }

        if(flag == -1) flag = 3;
        if(flag== -2) flag = 2;
        if(flag == -3) flag=1;


            if (e.type == SDL_MOUSEBUTTONDOWN)
            {
                int x,y;
                SDL_GetMouseState(&x, &y);
                //closeMenu();
                //click on quit
                if(x > SW/2 - SW/8 && x < SW/2 - SW/8 + SW/4 && y > SH/2 + SH /8
                    && y < SH /2 + SH /8 + SH/6)
                    return 0;
                //click on start/restart
                else if(x > SW/2 - SW/8 && x < SW/2 - SW/8 + SW/4 && y > SH/8 &&
                    y < SH/8 +SH/6)
                    {
                        return 1;
                    }
                //click on hs
                else if(x > 120 && x < 120+ SW/4 && y > SH/4 + SH/8 && y < SH/4
                    +SH/8+ SH/6 )
                {
                        return 2;
                }
                else if(x > 360 && x < 360+ SW/4 && y > SH/4 + SH/8 && y < SH/4
                        +SH/8+ SH/6 )
                {
                        return 3;
                }

            }
        }

        //Clear screen
    SDL_RenderClear( gRenderer );
        SDL_Rect StartRect = {SW/2 - SW/8,SH /8,SW/4,SH/6};
        SDL_Rect QuitRect  = {SW/2 - SW/8,SH /2 + SH /8,SW/4,SH/6};
      SDL_Rect HSRect    = {120,SH /4+ SH/8 ,SW/4,SH/6};
      SDL_Rect InsRect    = {360,SH /4+ SH/8 ,SW/4,SH/6};


        if(!GOV) // GOV=  game over flag

        {
            SDL_Rect go={0,0 ,SCREEN_WIDTH, SCREEN_HEIGHT };
      SDL_RenderSetViewport( gRenderer, &go );
      SDL_RenderCopy( gRenderer,GST , NULL, NULL );


      SDL_RenderSetViewport( gRenderer, &StartRect );
      SDL_SetTextureBlendMode( gStart, SDL_BLENDMODE_BLEND );
    SDL_SetTextureAlphaMod( gStart, 150 );
      SDL_RenderCopy( gRenderer, gStart, NULL, NULL );




      SDL_RenderSetViewport( gRenderer, &QuitRect);
      SDL_SetTextureBlendMode( gQuit, SDL_BLENDMODE_BLEND );
    SDL_SetTextureAlphaMod( gQuit, 150 );
      SDL_RenderCopy( gRenderer, gQuit, NULL, NULL );

      SDL_RenderSetViewport( gRenderer, &InsRect);
      SDL_SetTextureBlendMode( gInstruct, SDL_BLENDMODE_BLEND );
    SDL_SetTextureAlphaMod( gInstruct, 150 );
      SDL_RenderCopy( gRenderer, gInstruct, NULL, NULL );




      SDL_RenderSetViewport( gRenderer, &HSRect );
      SDL_SetTextureBlendMode( gHS, SDL_BLENDMODE_BLEND );
    SDL_SetTextureAlphaMod( gHS, 150 );
      SDL_RenderCopy( gRenderer, gHS, NULL, NULL );
}
else
{
      SDL_Rect go={0,0 ,SCREEN_WIDTH, SCREEN_HEIGHT };
      SDL_RenderSetViewport( gRenderer, &go );
      SDL_RenderCopy( gRenderer,GOT , NULL, NULL );


      SDL_RenderSetViewport( gRenderer, &StartRect );
      SDL_SetTextureBlendMode( gRestart, SDL_BLENDMODE_BLEND );
    SDL_SetTextureAlphaMod( gRestart, 150 );
      SDL_RenderCopy( gRenderer, gRestart, NULL, NULL );

      SDL_RenderSetViewport( gRenderer, &InsRect);
      SDL_SetTextureBlendMode( gInstruct, SDL_BLENDMODE_BLEND );
    SDL_SetTextureAlphaMod( gInstruct, 150 );
      SDL_RenderCopy( gRenderer, gInstruct, NULL, NULL );


      SDL_RenderSetViewport( gRenderer, &QuitRect);
      SDL_SetTextureBlendMode( gQuit, SDL_BLENDMODE_BLEND );
    SDL_SetTextureAlphaMod( gQuit, 150 );
      SDL_RenderCopy( gRenderer, gQuit, NULL, NULL );
```

```
                        SDL_RenderSetViewport( gRenderer, &HSRect );
                        SDL_SetTextureBlendMode( gHS,      SDL_BLENDMODE_BLEND );
                      SDL_SetTextureAlphaMod( gHS, 150 );
                        SDL_RenderCopy( gRenderer, gHS, NULL, NULL );




                    }
                    //SDL_SetRenderDrawColor( gRenderer, 0xFF, 0x00, 0xFF, 0x10 );

                    if(flag==1 || flag == -3) optRect = StartRect;
                    else if(flag==2 || flag == -2) optRect= HSRect;
                    else if(!flag) optRect = QuitRect;
                    else optRect = InsRect;HSRect;



                    optRect2 = {0, 0, 300, 80};
                    SDL_RenderSetViewport( gRenderer , NULL);
                    SDL_SetRenderDrawColor( gRenderer, 0x00, 0xff, 0x00, 0xFF );
                    SDL_SetRenderTarget ( gRenderer, selTexture );
                    SDL_RenderFillRect( gRenderer, &optRect2 );

                    SDL_SetTextureBlendMode( selTexture, SDL_BLENDMODE_BLEND );
                    SDL_SetTextureAlphaMod( selTexture, 100 );

                    SDL_SetRenderTarget( gRenderer, NULL );
                    SDL_RenderCopy( gRenderer, selTexture, NULL, &optRect );

                    SDL_RenderPresent( gRenderer );

                }
            }
        }
}
```

The game() function:

```
int game()
{
	char states[10000];

	double a,b;
	int c,d,g,f;

	FILE* game1;
	int q1 = 0, q2 = 0, q3 = 0, q4 = 0;

	int state = 0;
	// game1 = fopen("game1.txt", "w");
	// fprintf(game1, "%lf %lf\n",pPosX,pPosY );

	game1 = fopen("game1.txt", "r");

	fscanf(game1, "%d %lf %lf %d %d %d %d",&state, &a, &b, &q1, &q2, &q3, &q4);

	fclose(game1);

	nise = 0;

	nise += q1+q2+q3+q4;

	cout << nise << endl;

	if(state == 1)
		return 10;

	// fscanf(states, "%d %d %d %d %d %d", &pPosX, &pPosY, &)

	mapToArr(maparr);
	Caught = false;
	Alerted = false;
	Score = 0;

	Tile* tileSet[ TOTAL_TILES ];
	Player player(a,b);
	Enemy enemy1(200, 218);
	Enemy3 enemy2(95, 337);
	Enemy2 enemy3(536, 266);
	Enemy4 enemy4(380, 350);

	Items item1(60, 215);
	Items1 item2(135, 300);
	Items2 item3(379, 390);
	Items3 item4(626, 170);

	if(SHESH == 5)
	{
        close(tileSet);
        return 0;
    }

    if(!loadMedia(tileSet))
    {
        printf( "Failed to load media!\n" );
    }
    else
    {
        bool quit = false;

        SDL_Event e;

        while( !quit )
        {
            int x,y;
            SDL_GetMouseState(&x, &y);

                //cout << x << " " << y << endl;
            while( SDL_PollEvent( &e ) != 0 )
            {
                if( e.type == SDL_QUIT )
                {
                    quit = true;
                }

                player.HandleEvent(e);

                if(e.type == SDL_KEYDOWN)
                {
                    switch(e.key.keysym.sym)
                    {
                        case SDLK_s:
                        {
                            FILE* game1;
                            // game1 = fopen("game1.txt", "w");
                            // fprintf(game1, "%lf %lf\n",pPosX,pPosY );

                            // game1 = fopen("game1.txt", "r");

                            // fscanf(game1, "%lf %lf %d %d %d %d ", &a, &b, &c, &d, &e,
                                        &f);

                            game1 = fopen("game1.txt", "w");
                            // double a,b;
                            fprintf(game1, "%d %lf %lf %d %d %d %d\
                                        n",state,pPosX,pPosY,q1,q2,q3,q4 );

                            fclose(game1);
                            return 2;
                        }
                    break;
                }
            }
        }
        // cout << nise;

        player.Move(tileSet);

        // cout << pPosX << pPosY << endl;

        enemy1.Move(tileSet, maparr);

        enemy1.Detected(pCollider);

        enemy2.Move(tileSet, maparr);

        enemy2.Detected(pCollider);

        enemy3.Move(tileSet, maparr);

        enemy3.Detected(pCollider);

        enemy4.Move(tileSet, maparr);

        enemy4.Detected(pCollider);

        SDL_SetRenderDrawColor( gRenderer, 0xFF, 0xFF, 0xFF, 0xFF );
        SDL_RenderClear( gRenderer );

        for( int i = 0; i < TOTAL_TILES; ++i )
            tileSet[ i ]->render();

        player.render();

        enemy1.render();
        enemy2.render();
        enemy3.render();
        enemy4.render();

        if(!item1.Collect())
        {
            if(q1 == 0)
                item1.render();
            if(item1.Destroy())
                {
                    nise++;
                    q1++;
                }
        }

        if(!item2.Collect())
        {
```

```
        if(q2 == 0)
                item2.render();
        if(item2.Destroy())
                {
                        nise++;
                        q2++;
                }
}

if(!item3.Collect())
{
        if(q3 == 0)
                item3.render();
        if(item3.Destroy())
        {
                nise++;
                q3++;
        }
}



if(!item4.Collect())
{
        if(q4 == 0)
                item4.render();
        if(item4.Destroy())
                {
                        nise++;
                        q4++;
                }
}

SDL_SetRenderDrawColor( gRenderer, 0x00, 0x00, 0xFF, 0xFF );
SDL_RenderDrawRect( gRenderer, &Change );

SDL_RenderPresent( gRenderer );

if(Caught)// || nise == 8)
{
        nise = 0;
//      printf("Caught13 %d\n",Caught );
        // close(tileSet);
        //cout << "11111111" << endl;
        GOV = 1;

        FILE* game1 = fopen("game1.txt", "w");
        a = 0.0,b = 0.0;
        c = 0,d = 0,g = 0,f = 0;

        fprintf(game1,"%d %lf %lf %d %d %d %d", state, a, b ,c ,d ,g, f);
```

```
                fclose(game1);

                return 1;
            }

            if(nise == 4)
            {
                state = 1;

                FILE* game1 = fopen("game1.txt", "w");
                a = 0.0,b = 0.0;
                c = 0,d = 0,g = 0,f = 0;

                fprintf(game1,"%d %lf %lf %d %d %d %d", state, a, b ,c ,d ,g, f);

                fclose(game1);
                close(tileSet);
                return 10;
            }
        }
    }
    // cout << Score << endl;
    close(tileSet);

    return 0;
}
```

<u>The instructfunc() function:</u>

```
int instructfunc()
{
	SDL_RenderClear(gRenderer);
	SDL_Event e;
	SDL_SetRenderDrawColor(gRenderer,0,0,0,0);
	SDL_Texture* Instructionviewer = NULL;
	SDL_Rect instructview ={0,0, SW , SH};
	SDL_RenderSetViewport( gRenderer, &instructview );
	Instructionviewer = loadTexture("instruct.png");
	SDL_SetTextureBlendMode( Instructionviewer, SDL_BLENDMODE_BLEND );
	SDL_SetTextureAlphaMod( Instructionviewer, 150 );
	SDL_RenderCopy( gRenderer, Instructionviewer, NULL, NULL );
	SDL_RenderPresent(gRenderer);

	bool quit = false;
	while(!quit)
	{
		while(SDL_PollEvent(&e)!=0)
		{
			if(e.type== SDL_QUIT)
				{
					return 0;
				}

			if (e.type == SDL_KEYDOWN)
			{
				switch( e.key.keysym.sym )
		  {
		    case SDLK_RETURN:
				SDL_RenderClear(gRenderer);

			return 5;
		    break;
		  }
			}
		}
	}
```

The showhs() function:

```
int show_hs()
{
        FILE *p= fopen("hs.txt","r");
        int hiscore,score = Score;
        char _name[1000];
        int cnt;
        for(cnt=0;cnt<5;cnt++)
        fscanf(p,"%d %s",&card[cnt].s,card[cnt].n);
        SDL_SetRenderDrawColor(gRenderer,0,0,0,255);
        char s[10000];
        SDL_RenderClear(gRenderer);
        for(cnt=0;cnt<5;cnt++)
        {
                sprintf(s," %s : %d  ", card[cnt].n,card[cnt].s);

                SDL_Color textColor = {255,255,255};
                //gScore.render(0,0);

        //      SDL_RenderClear(gRenderer);
                //SDL_RenderCopy( gRenderer,GA , NULL, NULL );

                gScore.setAlpha(100);
                gScore.LoadFromRenderedText(s,textColor);
                gScore.render(300-60, 100+cnt*35);
        }
        SDL_RenderPresent(gRenderer);
        bool quit = false;
        SDL_Event e;
        while(!quit)
        {
                while(SDL_PollEvent(&e)!=0)
                {
                        if(e.type== SDL_QUIT)
                                {
                                        return 0;
                                }
                        // if (e.type == SDL_MOUSEBUTTONDOWN)
                        //      {
                        //              SDL_RenderClear(gRenderer);
                        //              return 5;
                        //      }
                        if (e.type == SDL_KEYDOWN)
                        {
                                switch( e.key.keysym.sym )
                    {
                       case SDLK_RETURN:
                                SDL_RenderClear(gRenderer);

                        return 5;
                       break;
                }
                    }
              }
        }
}
```

References:

- http://www.stereosue.com/
- http://www.thewallpapers.org
- http://www.stackoverflow.com
- http://www.programmingsimplified.com
- www.youtube.com
- http://lazyfoo.net/SDL_tutorials/
- https://wiki.libsdl.org/FrontPage
- C The Complete Reference By Herbert Schildt
- Programming in ANSI C By E.Balaguruswami