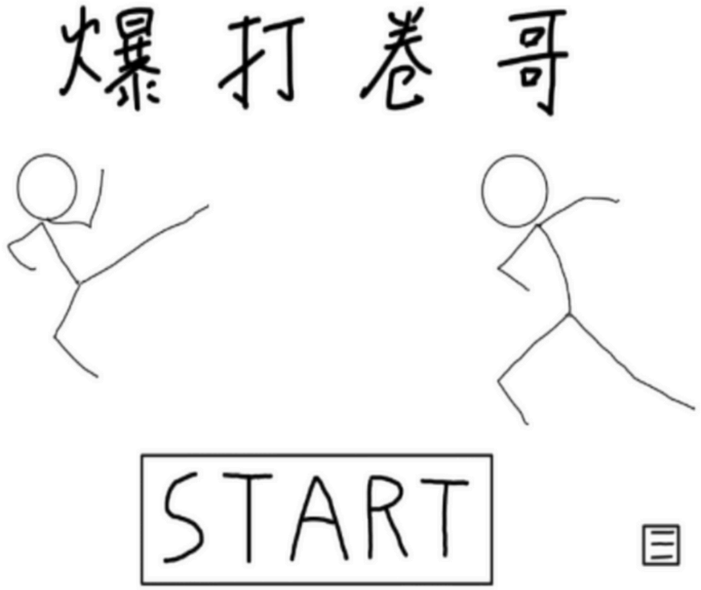
FINAL REPORT-
BATTLE GAME

B12901025 Po Hung Cheng

CONTENTS

- Planning
- Functional design
- Game introduction
- Art interface
- Programming architecture
- Experience and suggestions
- Work distribution
- Personal coding explanation
- character(class)
- player(class)
- main
- Object-oriented programming techniq
- Program optimization other tips



PLANNING

The actual completed part

- character(base class of the class player, Tool): contains position, velocity, index of the picture, flip or not
- Data(bass class): contains time and control the existence of the Tool and it is the member of the Tool(composition)
- Tool(derived class): contains the picture’s height, width, and four functions which represent four kinds of Tools

PLANNING

The actual completed part

- player(derived class): contains hp, attack , defence , functions about attacking another player, and functions about colliding with another player
- Lbutton: control the mouse event
- main function: handle the current state of the game, calls the functions written in the class, and render the correct picture and sound

FUNCTIONAL DESIGN

- p1: Use WASD to control up, down, left, and right. F is for attack and G is for defense.
- p2: Use the arrow keys to control. The period (.) is for attack and the slash (/) is for defense.

```
//Adjust the velocity
switch( e.key.keysym.sym )
{
    case SDLK_w: mVelY -= character_VEL; index[1] =10; break;
    case SDLK_s: mVelY += character_VEL; index[1] =10; break;
    case SDLK_a: mVelX -= character_VEL; index[1] =10; flip = SDL_FLIP_HORIZONTAL;break;
    case SDLK_d: mVelX += character_VEL; index[1] =10; flip = SDL_FLIP_NONE;break;
    case SDLK_f: index[1] = 2; break;   // press f to attack
    case SDLK_g: index[1] = 1; def_or_not = true; break;//press g to defence

}
//Adjust the velocity
switch( e.key.keysym.sym )
{
    case SDLK_UP: mVelY -= character_VEL;   index[1] =10;break;
    case SDLK_DOWN: mVelY += character_VEL; index[1] =10;break;
    case SDLK_LEFT: mVelX -= character_VEL; index[1] =10; flip = SDL_FLIP_HORIZONTAL;break;
    case SDLK_RIGHT: mVelX += character_VEL;index[1] =10; flip = SDL_FLIP_NONE; break;
    case SDLK_PERIOD: index[1] = 2; break; //press . to attack
    case SDLK_SLASH: index[1] = 1; def_or_not = true; break;// press / to defence
}
```

FUNCTIONAL DESIGN

- Four character choices (from the second parameter, they are health, attack power, defense power, character width, character height, speed)

1. balance 
2. attack 
3. agile 
4. defence 

```
case 1:p1->set(0,400,100,100,150,350,11,SDL_FLIP_NONE); render1 = false;break;  //balance
case 2:p1->set(1,400,135,80,150,350,10,SDL_FLIP_NONE); render2 = false;break;   //attack
case 3:p1->set(2,340,80,80,150,350,15,SDL_FLIP_NONE); render3 = false;break;   //agile
case 4:p1->set(3,420,90,140,150,350,5,SDL_FLIP_NONE); render4 = false;break;   //defence
```

FUNCTIONAL DESIGN

- Items: divided into four types

1. Square root multiplied by 10: Take the square root of health, attack power, and defense power and multiply it by 10
2. Starbucks: Add 20% of the original health, but cannot add more than the original health
3. Big copy: Attack power and defense power +10
4. Repair order: HP is directly reset to 0

```
void Tool::sqrt_10(player & p){
    if( p.live == true){
        double nhp = (p.hp/p.in_hp)* 100.0;
        int new_hp = 10*sqrt(nhp)*p.in_hp/100;
        p.set_hp( new_hp );
        p.set_atk ( 10*sqrt( p.atk));
        p.set_def ( 10*sqrt(p.def ));
    }
    data -> set_index(0);
    data -> set_exists(false) ;
}

void Tool::starbuck (player & p){
    if( p.live == true){
        if( p.hp + 0.2*p.in_hp <= p.in_hp)
            p.set_hp ( p.hp + 0.2*p.in_hp) ;
        else
            p.set_hp(p.in_hp);
    }
    data -> set_index(0);
    data -> set_exists(false) ;
}

void Tool::big_note(player &p){
    if( p.live == true){
        p.set_atk( p.atk + 10);
        p.set_def(p.def + 10) ;
    }
    data -> set_index(0);
    data -> set_exists(false) ;
}

void Tool::withdraw(player &p){
    p.set_hp(0);
    p.live = false;
    data -> set_index(0);
    data -> set_exists(false) ;
}
```

GAME INTRO

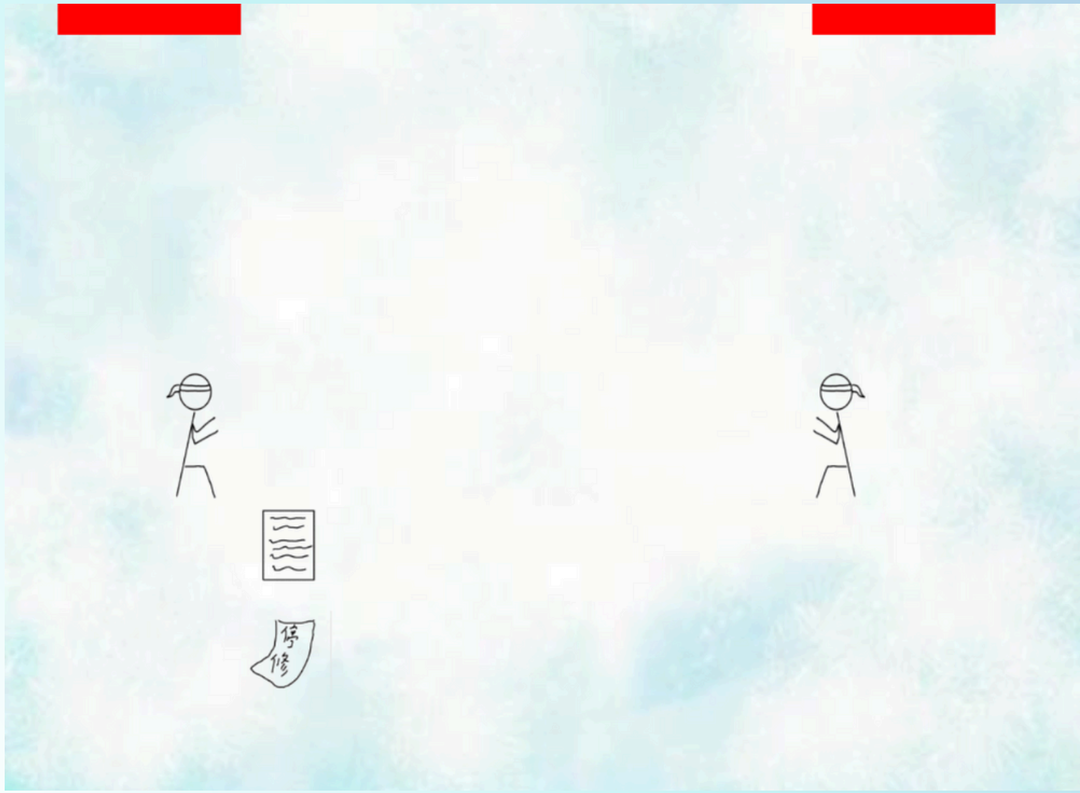
Both players claim to be the best player in the world, so you decide to duel to decide who is the real player. During the game, players can attack the opponent to make him lose health, and can also use defense to protect themselves. During the game, there will be random props. Eating the props may be able to turn the tide of the battle?

ART INTERFACE

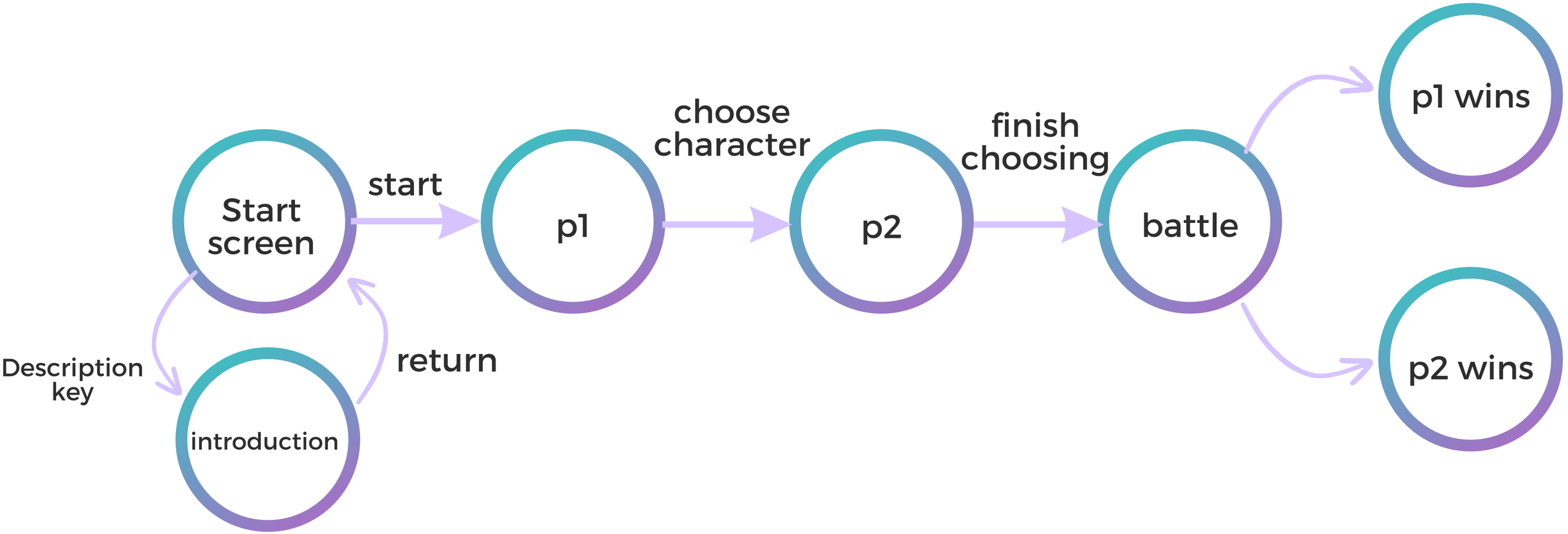
It is purely hand-painted, giving players a simple but not simple feeling.

The overall gameplay is very smooth and refreshing, and we can feel the joy of playing against each other during the process.

There are battle background music, battle sound effects and victory sound effects to make the battle more exciting.



PROGRAMMING ARCHITECTURE



Game description

gamestate = 0 uses a gButtons variable to determine whether the return button is pressed

Starting page

gamestate = 1 uses the two gButtons variables to confirm whether the mouse is in the instruction key, the return key, and whether the mouse is pressed.

choose character

gamestate = 2 or 3 uses the variables of the four gButtons to confirm whether the mouse is in the four character selection boxes and whether the mouse is pressed.

code

```
//User presses a key
switch (gamestate)
{
    case 0:
        gButtons[10].handleEventBack( &e );
        if (gButtons[10].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
        {
            gamestate = 1;
            break;
        }
}
```

```
switch (gamestate)
{
    case 1:
        gButtons[0].handleEventStart( &e );
        gButtons[9].handleEventGuide( &e );
        if (gButtons[0].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
        {
            Mix_PlayChannel( -1, mouse_over, 0 );
            render = true;
        }
        if (gButtons[0].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
        {
            Mix_PlayChannel( -1, button, 0 );
            gamestate = 2;
            render = false;
        }
        if (gButtons[0].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
        {
            render = false;
        }
        if (gButtons[9].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
        {
            Mix_PlayChannel( -1, mouse_over, 0 );
            render0 = true;
        }
        if (gButtons[9].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
        {
            Mix_PlayChannel( -1, button, 0 );
            gamestate = 0;
            render0 = false;
        }
        if (gButtons[9].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
        {
            render0 = false;
        }
        break;
```

```
case 2:
    for (int i=1;i<=4;i++)
    {
        gButtons[i].handleEventCharacter( &e );
        if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
        {
            Mix_PlayChannel( -1, mouse_over, 0 );
            render = true;
            switch(i)
            {
                case 1:render1 = true;break;
                case 2:render2 = true;break;
                case 3:render3 = true;break;
                case 4:render4 = true;break;
            }
        }
        if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
        {
            Mix_PlayChannel( -1, button, 0 );
            render = false;
            switch(i)
            {
                case 1:p1->set(0,400,100,100,150,350,11,SDL_FLIP_NONE); render1 = false;break;  //balance
                case 2:p1->set(1,400,135,80,150,350,10,SDL_FLIP_NONE); render2 = false;break;   //attack
                case 3:p1->set(2,340,80,80,150,350,15,SDL_FLIP_NONE); render3 = false;break;    //agile
                case 4:p1->set(3,420,90,140,150,350,5,SDL_FLIP_NONE); render4 = false;break;    //defence
            }
            gamestate = 3;
        }
        if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
        {
            switch(i)
            {
                case 1:render1 = false;break;
                case 2:render2 = false;break;
                case 3:render3 = false;break;
                case 4:render4 = false;break;
            }
        }
    }
    break;
```

battle

gamestate = 5 handles combat-related parts. Call many functions to implement the combat process. For example, collision is used to determine whether two characters will collide, be_attacked is used to determine the attack, and get_prop is used to determine whether the prop has been eaten.

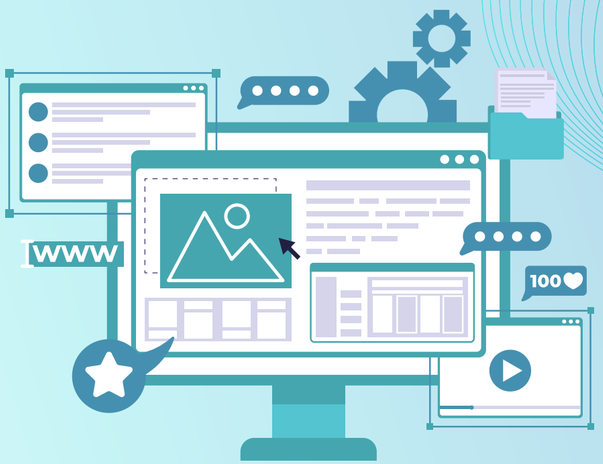
win/lose

gamestate = 6 or 7 will play the victory sound effect and determine when the ENTER key is pressed, the game will end.

code

```
case 5:
    {
        p1->handleEvent( 1,e );
        p2->handleEvent( 2,e );
        p1->collision(*p2);
        p2->collision(*p1);
        p1->be_attacked(*p2);
        p2->be_attacked(*p1);
        get_prop(*p1,item[0],SDL_GetTicks());
        get_prop(*p2,item[0],SDL_GetTicks());
        get_prop(*p1,item[1],SDL_GetTicks());
        get_prop(*p2,item[1],SDL_GetTicks());
        item[0].set_exists( item[0].collision(item[1]) );
        if (item[0].get_exists() == false){
            item[0].set_index(0);
        }
        item[1].set_exists(item[1].collision(item[0]));
        if (item[1].get_exists() == false){
            item[1].set_index(0);
        }
        break;
    }
case 6:
    {
        if (e.key.type==SDL_KEYDOWN)
        {
            switch(e.key.keysym.sym)
                case SDLK_RETURN:
                    return 0;
        }
    }
```

THOUGHTS AND SUGGESTIONS



I initially thought that making games was too difficult for me because it was difficult for me to even keep up with the courses. But after these few weeks of guidance from team members, teaching assistants, and professors, I gradually made a game that could move, which made me feel very accomplished. Although it is very rough, it also means that my programming ability has improved significantly.

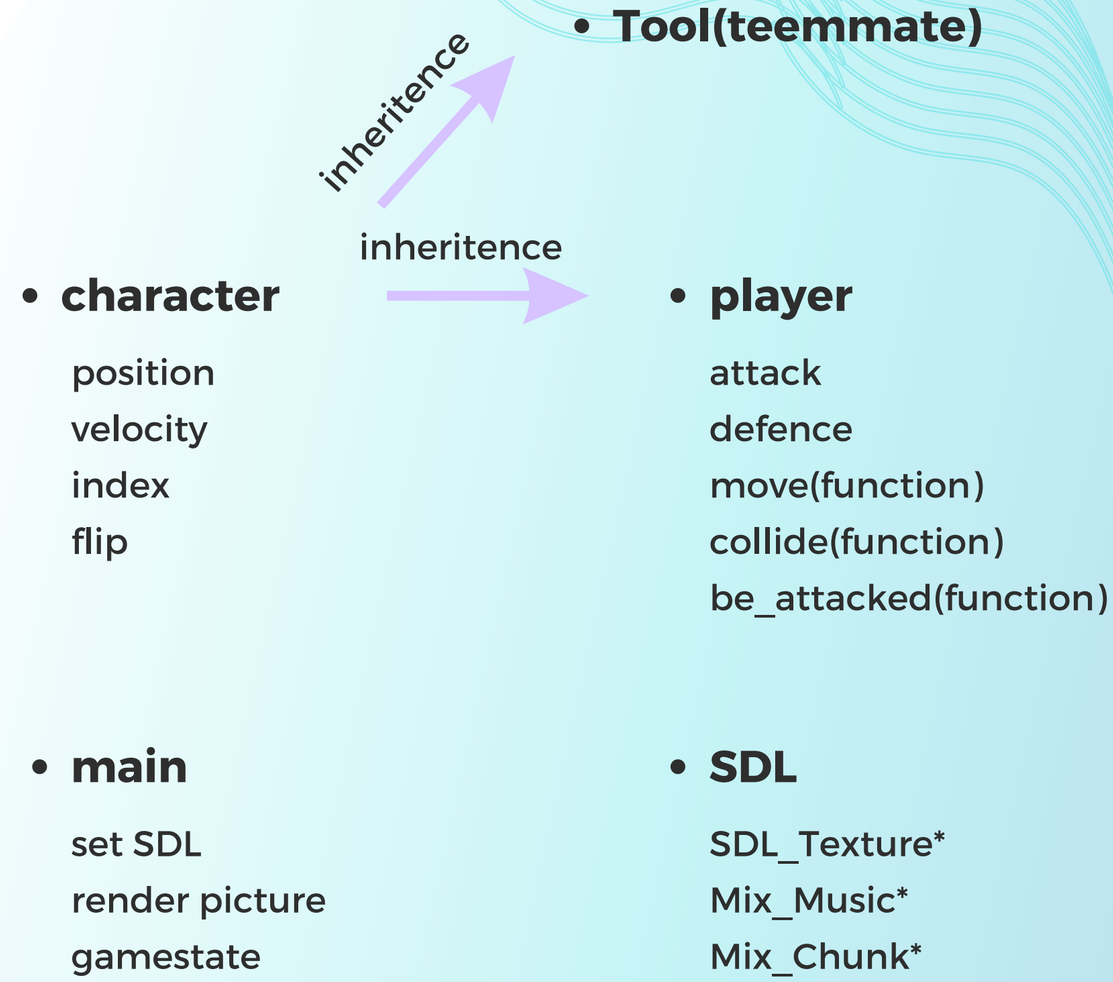
I also hope to continue studying programming in the future and consider software engineering as an option in my career plan.

WORK ASSIGNMENT

- Main program writing and organizing: Po Hung Cheng
- character: Po Hung Cheng
- player: Po Hung Cheng
- Data: Juming Gao
- Tool: Juming Gao
- load_picture_sound.h: Juming Gao
- Lbutton.h: Juming Gao

PERSONAL CODING EXPLANATION

- character(base class): contains position, velocity, index of the picture, flip or not
- player(derived class): contains hp, attack , defence , functions about attacking another player, and functions about colliding with another player
- main: combines teammate's header files and codes regarding SDL



character(player_tool.h)

- **public:**

1. get or set the value of the mPosX, mPosY, mVelX, mVelY(Encapsulation)
2. character_VEL: the maximum velocity of player(can not be modified once we setup)

- **protected: (it needs to be inherited by the player(derived class))**

1. displacement and current velocity
2. *index: a dynamic array contains two numbers, which determines the picture we want to render

```
class character
{

    public:

        //Maximum axis velocity of the dot
        int character_VEL;

        //the picture is flipped or not
        SDL_RendererFlip flip;

        //Initializes the variables
        character():mVelX(0),mVelY(0),def_or_not(false){
            index = new int [2];
            index[0] = 0;
            index[1] = 8;
        }
        ~character(){delete [] index ;}
        //Takes key presses and adjusts the dot's velocity
        void handleEvent( int index , SDL_Event& e );   //different player

        virtual int get_mPosX() {return mPosX;}
        virtual int get_mPosY() {return mPosY;}
        int get_mVelX() {return mVelX;}
        int get_mVelY() {return mVelY;}
        void set_mVelX( int v) { mVelX = v;}
        void set_mVelY( int v) { mVelY = v;}
        void set_character_v(int v) {   character_VEL = v;}

        void set_index2(int a)
        {
            index[1] = a;
        }

    protected:
        //The X and Y offsets of the dot

        int mPosX, mPosY;

        //The velocity of the dot
        int mVelX, mVelY;

        //the index of the picture
        int *index;

        //whether the defence button is pressed
        bool def_or_not;
};
```

character(player_tool.h)

- **constructor:** dynamic memory being allocated(line 19) and initialize the value(line18,20,21)
- **destructor:** free the space of the pointer(line 23)
- player and Tool are inherited from the character(multiple inheritance)

```
6   class character
7   {
8
9       public:
10
11          //Maximum axis velocity of the dot
12          int character_VEL;
13
14          //the picture is flipped or not
15          SDL_RendererFlip flip;
16
17          //Initializes the variables
18          character():mVelX(0),mVelY(0),def_or_not(false){
19              index = new int [2];
20              index[0] = 0;
21              index[1] = 8;
22          }
23          ~character(){delete [] index ;}
24          //Takes key presses and adjusts the dot's velocity
25          void handleEvent( int index , SDL_Event& e );   //different player
26
27          virtual int get_mPosX() {return mPosX;}
28          virtual int get_mPosY() {return mPosY;}
29          int get_mVelX() {return mVelX;}
30          int get_mVelY() {return mVelY;}
31          void set_mVelX( int v) { mVelX = v;}
32          void set_mVelY( int v) { mVelY = v;}
33          void set_character_v(int v) {   character_VEL = v;}
34
35          void set_index2(int a)
36          {
37              index[1] = a;
38          }
40      protected:
41          //The X and Y offsets of the dot
42
43          int mPosX, mPosY;
44
45          //The velocity of the dot
46          int mVelX, mVelY;
47
48          //the index of the picture
49          int *index;
50
51          //whether the defence button is pressed
52          bool def_or_not;
53  };
```

Data(player_tool.h)

- Tool has the Data(composition, line 77)
- Tool only needs one index to indicate the right picture . Therefore, we create another class called Data(line 55) and put *index in it.(line 59)
- Tool is a derived class from the class character(line 75) and has a Data pointer called data(line 77)

```
55 class Data{
56     private:
57         bool exists;
58         Uint32 initial_time;
59         int *index;
60     public:
61         Data():exists(false),initial_time(0) {
62             index = new int;
63             *index = 0;
64         }
65         void set_exists(bool t){ exists=t;}
66         void set_time(Uint32 t){ initial_time = t;}
67         void set_index(int i){ *index = i;}
68         Uint32 get_time(){return initial_time;}
69         bool get_exists(){return exists;    }
70         int get_index(){ return *index;}
71 };
```

```
75 class Tool:public character{
76     private:
77         Data *data;
78
79         const Uint32 duration;
80         const int character_WIDTH;
81         const int character_HEIGHT;
82     public:
83         static int tool_num;
84         friend void render_tool(SDL_Renderer* &gRenderer,Tool &item);
85         friend void get_prop(player&p,Tool &item,Uint32 time);
86         friend void tool_appear( long long int rtime, Uint32 tick_time,Too
87
88         Tool(): character_HEIGHT(80),character_WIDTH(80),duration(5)
89         {
90             data = new Data;
91             //index = new int;
92             //*index = 0;
93             //initial_time = 0;
94             //exists = false;
95         }
96         ~Tool(){delete data;}
97         bool get_exists(){
98             return data->get_exists();
99         }
```

player(player_tool.h)

- **public:**

1. character_WIDTH, character_HEIGHT: are declared as constant int (line 126)because they cannot and should not be modified
2. live: the player's hp is higher than zero or not
3. collision,be_attacked: functions

- **private(we don't want to access them easily) :**

1. static int player_num(line 158): to store the current number of the player on the field now
2. hp, atk, def, in_atk(original attack), in_def, in_hp

```
117 class player: public character{
118     public:
119         friend void render_blood(SDL_Renderer* &gRenderer,player &p,int type);
120         friend void get_prop(player&p,Tool &item,Uint32);
121         friend void Tool::sqrt_10(player &p);
122         friend void Tool::starbuck(player &p);
123         friend void Tool::big_note(player &p);
124         friend void Tool::withdraw(player &p);
125         friend void render_picture(SDL_Renderer* &gRenderer,player &p,int type);
126         const int character_WIDTH;
127         const int character_HEIGHT;
128         bool live;
129         player():live(false),character(),character_WIDTH(50),character_HEIGHT(120),start
130         }
131         ~player(){}
132         void set(int type,int blood,int attack,int defence,int xpos,int ypos,int speed,S
133         {
134             live = true;
135             index[0] = type;
136             hp = blood;
137             in_hp = blood;
138             atk = attack;
139             in_atk = attack;
140             def = defence;
141             in_def = defence;
142             mPosX = xpos;
143             mPosY = ypos;
144             character_VEL = speed;
145             flip = F;
146         }
147         void collision(player &p);  //move and collide
148         void be_attacked(player &p); //checked the player is hit
149         int get_VEL(){return character_VEL;}
150         void set_atk( int a) { atk = a;}
151         void set_def(int a) {def = a;}
152         void set_hp( int a){ hp = a;}
153         void set_VEL( int a){ character_VEL = a;}
154         double get_in_hp(){ return in_hp;}
155         int get_in_atk(){ return in_atk;}
156         int get_in_def(){ return in_def;}
157     private:
158         static int player_num;
159         double hp;
160         int atk, def;
161         int in_atk,in_def;
162         double in_hp;
163         Uint32 starttime;
164
165 };
```

player(player_tool.h)

- **friend function**

1. render_blood: access hp
2. get_prop: access mPosX, mPosY in character
3. sqrt_10: access hp, atk, def
4. starbuck: access hp
5. big_note: access atk, def
6. withdraw: access hp
7. render_picture: access *index in character and render the picture correctly

```
117 class player: public character{
118     public:
119         friend void render_blood(SDL_Renderer* &gRenderer,player &p,int type);
120         friend void get_prop(player&p,Tool &item,Uint32);
121         friend void Tool::sqrt_10(player &p);
122         friend void Tool::starbuck(player &p);
123         friend void Tool::big_note(player &p);
124         friend void Tool::withdraw(player &p);
125         friend void render_picture(SDL_Renderer* &gRenderer,player &p,int type);
126         const int character_WIDTH;
127         const int character_HEIGHT;
128         bool live;
129         player():live(false),character(),character_WIDTH(50),character_HEIGHT(120),sta
130         }
131         ~player(){}
132         void set(int type,int blood,int attack,int defence,int xpos,int ypos,int speed
133         {
134             live = true;
135             index[0] = type;
136             hp = blood;
137             in_hp = blood;
138             atk = attack;
139             in_atk = attack;
140             def = defence;
141             in_def = defence;
142             mPosX = xpos;
143             mPosY = ypos;
144             character_VEL = speed;
145             flip = F;
146         }
147         void collision(player &p);  //move and collide
148         void be_attacked(player &p); //checked the player is hit
149         int get_VEL(){return character_VEL;}
150         void set_atk( int a) { atk = a;}
151         void set_def(int a) {def = a;}
152         void set_hp( int a){ hp = a;}
153         void set_VEL( int a){ character_VEL = a;}
154         double get_in_hp(){ return in_hp;}
155         int get_in_atk(){ return in_atk;}
156         int get_in_def(){ return in_def;}
157     private:
158         static int player_num;
159         double hp;
160         int atk, def;
161         int in_atk,in_def;
162         double in_hp;
163         Uint32 starttime;
164
165 };
```

player(player_tool.h)

- **constructor:** initialize the value (line129)
- **destructor:** (line 131)

```
117 class player: public character{
118     public:
119         friend void render_blood(SDL_Renderer* &gRenderer,player &p,int type);
120         friend void get_prop(player&p,Tool &item,Uint32);
121         friend void Tool::sqrt_10(player &p);
122         friend void Tool::starbuck(player &p);
123         friend void Tool::big_note(player &p);
124         friend void Tool::withdraw(player &p);
125         friend void render_picture(SDL_Renderer* &gRenderer,player &p,int type);
126         const int character_WIDTH;
127         const int character_HEIGHT;
128         bool live;
129         player():live(false),character(),character_WIDTH(50),character_HEIGHT(120),sta
130         }
131         ~player(){}
132         void set(int type,int blood,int attack,int defence,int xpos,int ypos,int speed
133         {
134             live = true;
135             index[0] = type;
136             hp = blood;
137             in_hp = blood;
138             atk = attack;
139             in_atk = attack;
140             def = defence;
141             in_def = defence;
142             mPosX = xpos;
143             mPosY = ypos;
144             character_VEL = speed;
145             flip = F;
146         }
147         void collision(player &p);  //move and collide
148         void be_attacked(player &p); //checked the player is hit
149         int get_VEL(){return character_VEL;}
150         void set_atk( int a) { atk = a;}
151         void set_def(int a) {def = a;}
152         void set_hp( int a){ hp = a;}
153         void set_VEL( int a){ character_VEL = a;}
154         double get_in_hp(){ return in_hp;}
155         int get_in_atk(){ return in_atk;}
156         int get_in_def(){ return in_def;}
157     private:
158         static int player_num;
159         double hp;
160         int atk, def;
161         int in_atk,in_def;
162         double in_hp;
163         Uint32 starttime;
164
165 };
```

main

- **preset**

1. gamestate = 1
2. render,render0(bool): If the mouse is on the start button(guide button), we set the render(render0) to true and play the channel
3. we use one switch to realize a loop between gamestate 0 and gamestate 1
4. the other switch to control state 1 to state 7
5. in state 1, we call the function preset to render picture

```
22  //determine which picture is going to show
23  void preset(SDL_Renderer* &gRenderer,bool render,bool render0)
24  {
25      SDL_RenderCopy( gRenderer, menu, NULL, NULL );
26      if (render==true)
27      {
28          SDL_Rect renderQuad1 = { 287, 485, 360, 136};
29          SDL_RenderCopyEx( gRenderer, icon1 , NULL,&renderQuad1,0.0, NULL, SDL_FLIP_NONE);
30      }
31      if (render0==true)
32      {
33          SDL_Rect renderQuad1 = { 770, 545, 60, 60};
34          SDL_RenderCopyEx( gRenderer, icon2 , NULL,&renderQuad1,0.0, NULL, SDL_FLIP_NONE);
35      }
36  }
```

```
285                 //User presses a key
286                 switch (gamestate)
287                 {
288                     case 0:
289                         gButtons[10].handleEventBack( &e );
290                         if (gButtons[10].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
291                         {
292                             gamestate = 1;
293                             break;
294                         }
295                 }
296                 switch (gamestate)
297                 {
298                     case 1:
299                         gButtons[0].handleEventStart( &e );
300                         gButtons[9].handleEventGuide( &e );
301                         if (gButtons[0].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
302                         {
303                             Mix_PlayChannel( -1, mouse_over, 0 );
304                             render = true;
305                         }
306                         if (gButtons[0].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
307                         {
308                             Mix_PlayChannel( -1, button, 0 );
309                             gamestate = 2;
310                             render = false;
311                         }
312                         if (gButtons[0].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
313                         {
314                             render = false;
315                         }
316                         if (gButtons[9].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
317                         {
318                             Mix_PlayChannel( -1, mouse_over, 0 );
319                             render0 = true;
320                         }
321                         if (gButtons[9].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
322                         {
323                             Mix_PlayChannel( -1, button, 0 );
324                             gamestate = 0;
325                             render0 = false;
326                         }
327                         if (gButtons[9].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
328                         {
329                             render0 = false;
330                         }
331                         break;
```

```
458                 //Clear screen
459                 SDL_RenderClear( gRenderer );
460                 switch(gamestate)
461                 {
462                     case 0: guide(gRenderer); break;
463                     case 1:
464                         {
465                             preset(gRenderer,render,render0);
466                             break;
467                         }
```

main

- **guide**

1. gamestate = 0
2. render the introduction image

```
175  void guide(SDL_Renderer* &gRenderer)
176  {
177      SDL_RenderCopy( gRenderer, intro, NULL, NULL );
178  }
```

```
285                  //User presses a key
286                  switch (gamestate)
287                  {
288                      case 0:
289                          gButtons[10].handleEventBack( &e );
290                          if (gButtons[10].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
291                          {
292                              gamestate = 1;
293                              break;
294                          }
295                  }
```

```
458                  //Clear screen
459                  SDL_RenderClear( gRenderer );
460                  switch(gamestate)
461                  {
462                      case 0: guide(gRenderer); break;
463                      case 1:
464                          {
465                              preset(gRenderer,render,render0);
466                              break;
467                          }
```

main

- **choose1**

1. gamestate = 2
2. render1, render2, render3, render4: determine the mouse is on the four buttons
3. set the value of p1

```
void choose1(SDL_Renderer* &gRenderer,bool render1,bool render2,bool render3,bool render4)
{
    SDL_RenderCopy( gRenderer, choose, NULL, NULL );
    if (render1==true)
    {
        SDL_Rect renderQuad1 = { 17, 2, 150, 150};
        SDL_RenderCopyEx( gRenderer, picture[0][0] , NULL,&renderQuad1,0.0, NULL, SDL_FLIP_NONE);
    }
    if (render2==true)
    {
        SDL_Rect renderQuad1 = { 21, 190, 150, 150};
        SDL_RenderCopyEx( gRenderer, picture[1][0] , NULL,&renderQuad1,0.0, NULL, SDL_FLIP_NONE);
    }
    if (render3==true)
    {
        SDL_Rect renderQuad1 = { 18, 397, 150, 150};
        SDL_RenderCopyEx( gRenderer, picture[2][0] , NULL,&renderQuad1,0.0, NULL, SDL_FLIP_NONE);
    }
    if (render4==true)
    {
        SDL_Rect renderQuad1 = { 14, 600, 150, 150};
        SDL_RenderCopyEx( gRenderer, picture[3][0] , NULL,&renderQuad1,0.0, NULL, SDL_FLIP_NONE);
    }
}
```

```
                    case 2:
                        for (int i=1;i<=4;i++)
                        {
                            gButtons[i].handleEventCharacter( &e );
                            if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
                            {
                                Mix_PlayChannel( -1, mouse_over, 0 );
                                render = true;
                                switch(i)
                                {
                                    case 1:render1 = true;break;
                                    case 2:render2 = true;break;
                                    case 3:render3 = true;break;
                                    case 4:render4 = true;break;
                                }
                            }
                            if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
                            {
                                Mix_PlayChannel( -1, button, 0 );
                                render = false;
                                switch(i)
                                {
                                    case 1:p1->set(0,400,100,100,150,350,11,SDL_FLIP_NONE); render1 = false;break;  //
                                    case 2:p1->set(1,400,135,80,150,350,10,SDL_FLIP_NONE); render2 = false;break;   //
                                    case 3:p1->set(2,340,80,80,150,350,15,SDL_FLIP_NONE); render3 = false;break;   //a
                                    case 4:p1->set(3,420,90,140,150,350,5,SDL_FLIP_NONE); render4 = false;break;   //d
                                }
                                gamestate = 3;
                            }
                            if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
                            {
                                switch(i)
                                {
                                    case 1:render1 = false;break;
                                    case 2:render2 = false;break;
                                    case 3:render3 = false;break;
                                    case 4:render4 = false;break;
                                }
                            }
                        }
                        break;
```

```
                    case 2: choose1(gRenderer,render1,render2,render3,render4); break;
                    case 3: choose2(gRenderer,render5,render6,render7,render8); break;
                    case 4: startbattle(gRenderer); break;
                    case 5:
                    {
                        battlefield(gRenderer);
                        render_picture(gRenderer,*p1,1);
                        render_picture(gRenderer,*p2,2);
                        render_blood(gRenderer,*p1,1);
                        render_blood(gRenderer,*p2,2);
                        tool_appear( rand(),SDL_GetTicks(),item[0] );
                        tool_appear( rand(),SDL_GetTicks(),item[1] );
                        break;
                    }
```

main

- **battle**

1. gamestate = 5
2. call functions: handleEvent, collision, be_attacked, get_prop to deal with the player's movement, collide with the other player and boundary check, be attacked by the other player, get the game prop
3. I use the current blood to draw a red filled rectangle and original blood to draw a green rectangle

```
void render_blood(SDL_Renderer* &gRenderer,player &p,int type)
{
    int ori_blood;

    switch(p.index[0])
    {
        case 0:ori_blood = 200; break;
        case 1:ori_blood = 200; break;
        case 2:ori_blood = 170; break;
        case 3:ori_blood = 210; break;
    }
    SDL_Rect  rect1,rect2;
    if (type==1)
    {
        rect1 = {50,10,ori_blood,30};
        if (p.hp>0)
            rect2 = {50,10,p.hp/2,30};
        else
            rect2 = {50,10,0,30};
    }
    if (type==2)
    {
        rect1 = {750,10,ori_blood,30};
        if (p.hp>0)
            rect2 = {750,10,p.hp/2,30};
        else
            rect2 = {750,10,0,30};
    }
    //Render green outlined quad (original
    SDL_SetRenderDrawColor( gRenderer, 0x00, 0xFF, 0x00, 0xFF );
    SDL_RenderDrawRect( gRenderer, &rect1 );

    SDL_SetRenderDrawColor( gRenderer, 0xFF, 0x00, 0x00, 0xFF );
    SDL_RenderFillRect( gRenderer, &rect2 );

    SDL_SetRenderDrawColor( gRenderer, 0xFF, 0xFF, 0xFF, 0xFF );
}
```

```
                case 5:
                    {
                        p1->handleEvent( 1,e );
                        p2->handleEvent( 2,e );
                        p1->collision(*p2);
                        p2->collision(*p1);
                        p1->be_attacked(*p2);
                        p2->be_attacked(*p1);
                        get_prop(*p1,item[0],SDL_GetTicks());
                        get_prop(*p2,item[0],SDL_GetTicks());
                        get_prop(*p1,item[1],SDL_GetTicks());
                        get_prop(*p2,item[1],SDL_GetTicks());
                        item[0].set_exists( item[0].collision(item[1]) );
                        if (item[0].get_exists() == false){
                            item[0].set_index(0);
                        }
                        item[1].set_exists(item[1].collision(item[0]));
                        if (item[1].get_exists() == false){
                            item[1].set_index(0);
                        }
                        break;
                    }
                case 6:
                    {
                        if (e.key.type==SDL_KEYDOWN)
                        {
                            switch(e.key.keysym.sym)
                                case SDLK_RETURN:
                                    return 0;
                        }
                    }
```

```
                case 5:
                {
                    battlefield(gRenderer);
                    render_picture(gRenderer,*p1,1);
                    render_picture(gRenderer,*p2,2);
                    render_blood(gRenderer,*p1,1);
                    render_blood(gRenderer,*p2,2);
                    tool_appear( rand(),SDL_GetTicks(),item[0] );
                    tool_appear( rand(),SDL_GetTicks(),item[1] );
                    break;
                }
                case 6:
                    {
                        Mix_HaltMusic();
                        Mix_PlayChannel( -1, winn, 0 );
                        if (nextstate==true)
                        {
                            delete p1;
                            delete p2;
                            delete []item;
                            nextstate = false;
                            break;
                        }
                        over1(gRenderer);
                        break;
                    }
```

OBJECT ORIENTED PROGRAMMING TECHNIQUES

1 **constant**

the const keyword specifies that the object or variable isn't modifiable.

2 **static**

only one copy of the data is maintained for all objects of the class.

player(player_tool.h)

- **constant member:**

1. character_WIDTH, character_HEIGHT(public, line 126 127): It can prevents us from modifying it because it is a constant member.
2. we need the character's height and width to render part of the player(SDL_RenderCopyEx in main function)
3. collision and be_attacked need to access character_WIDTH and character_HEIGHT

```
117 class player: public character{
118     public:
119         friend void render_blood(SDL_Renderer* &gRenderer,player &p,int type);
120         friend void get_prop(player&p,Tool &item,Uint32);
121         friend void Tool::sqrt_10(player &p);
122         friend void Tool::starbuck(player &p);
123         friend void Tool::big_note(player &p);
124         friend void Tool::withdraw(player &p);
125         friend void render_picture(SDL_Renderer* &gRenderer,player &p,int type);
126         const int character_WIDTH;
127         const int character_HEIGHT;
128         bool live;
129         player():live(false),character(),character_WIDTH(50),character_HEIGHT(120),starttime(0){
130         }
131         ~player(){}
132         void set(int type,int blood,int attack,int defence,int xpos,int ypos,int speed,SDL_RendererFlip F)
133         {
134             live = true;
135             index[0] = type;
136             hp = blood;
137             in_hp = blood;
138             atk = attack;
139             in_atk = attack;
140             def = defence;
141             in_def = defence;
142             mPosX = xpos;
143             mPosY = ypos;
144             character_VEL = speed;
145             flip = F;
146         }
292 void player::collision(player &p)
293 {
294     int p_mPosX = p.get_mPosX();
295     int p_mPosY = p.get_mPosY();
296
297     //Move the character left or right
298     mPosX += mVelX;
299
300     //If the character went too far to the left or right
301     if( ( mPosX < 0 ) || ( mPosX + character_WIDTH > SCREEN_WIDTH ) )
302     {
303         //Move back
304         mPosX -= mVelX;
305     }
306
307     //Move the dot up or down
308     mPosY += mVelY;
309
310     //If the dot went too far up or down
311     if( ( mPosY < 50 ) || ( mPosY + character_HEIGHT > SCREEN_HEIGHT ) )
312     {
313         //Move back
314         mPosY -= mVelY;
315     }
316
317     if( abs(mPosX-p_mPosX) <= character_WIDTH-5 && abs(mPosY-p_mPosY) <= character_HEIGHT) //if they are going to touch
318     {
319         //Move back
320         mPosX -= mVelX;
321         mPosY -= mVelY;
322     }
323 }
```

```
325 void player::be_attacked(player &p)
326 {
327     int p_mPosX = p.get_mPosX();
328     int p_mPosY = p.get_mPosY();
329     if( abs(mPosX-p_mPosX) <= character_WIDTH+10 && abs(mPosY-p_mPosY) <= character_HEIGHT ) // be attacked by the other
330     {
331         if (p.flip==SDL_FLIP_HORIZONTAL&&p.get_mPosX()>get_mPosX())
332         {
333             if (p.index[1]==2)
334             {
335
336                 if (def_or_not == true)
337                 {
338                     Mix_PlayChannel( -1, atk_def, 0 );
339                     hp -= 0.2*p.atk*( 1.0 - def/200.0)*0.1;
340                     index[1] = 4;
341                 }
342                 else
343                 {
344                     Mix_PlayChannel( -1, atkk, 0 );
345                     hp -= p.atk *( 1.0 - def/200.0)*0.1 ;
346                     index[1] = 5;
347                 }
348                 if (hp<=0)
349                 {
350                     live = false;
351                     index[1] = 6;
352                 }
353             }
354         }
```

player(player_tool.h)

- **constant member:**

4. if don't declare them as constant, it might be confusing that these value can be modified, using constant can help maintain the program

```
117 class player: public character{
118     public:
119         friend void render_blood(SDL_Renderer* &gRenderer,player &p,int type);
120         friend void get_prop(player&p,Tool &item,Uint32);
121         friend void Tool::sqrt_10(player &p);
122         friend void Tool::starbuck(player &p);
123         friend void Tool::big_note(player &p);
124         friend void Tool::withdraw(player &p);
125         friend void render_picture(SDL_Renderer* &gRenderer,player &p,int type);
126         const int character_WIDTH;
127         const int character_HEIGHT;
128         bool live;
129         player():live(false),character(),character_WIDTH(50),character_HEIGHT(120),starttime(0){
130         }
131         ~player(){}
132         void set(int type,int blood,int attack,int defence,int xpos,int ypos,int speed,SDL_RendererFlip F)
133         {
134             live = true;
135             index[0] = type;
136             hp = blood;
137             in_hp = blood;
138             atk = attack;
139             in_atk = attack;
140             def = defence;
141             in_def = defence;
142             mPosX = xpos;
143             mPosY = ypos;
144             character_VEL = speed;
145             flip = F;
146         }
292 void player::collision(player &p)
293 {
294     int p_mPosX = p.get_mPosX();
295     int p_mPosY = p.get_mPosY();
296
297     //Move the character left or right
298     mPosX += mVelX;
299
300     //If the character went too far to the left or right
301     if( ( mPosX < 0 ) || ( mPosX + character_WIDTH > SCREEN_WIDTH ) )
302     {
303         //Move back
304         mPosX -= mVelX;
305     }
306
307     //Move the dot up or down
308     mPosY += mVelY;
309
310     //If the dot went too far up or down
311     if( ( mPosY < 50 ) || ( mPosY + character_HEIGHT > SCREEN_HEIGHT ) )
312     {
313         //Move back
314         mPosY -= mVelY;
315     }
316
317     if( abs(mPosX-p_mPosX) <= character_WIDTH-5 && abs(mPosY-p_mPosY) <= character_HEIGHT) //if they are going to touch
318     {
319         //Move back
320         mPosX -= mVelX;
321         mPosY -= mVelY;
322     }
323 }
```

```
325 void player::be_attacked(player &p)
326 {
327     int p_mPosX = p.get_mPosX();
328     int p_mPosY = p.get_mPosY();
329     if( abs(mPosX-p_mPosX) <= character_WIDTH+10 && abs(mPosY-p_mPosY) <= character_HEIGHT ) // be attacked by the other
330     {
331         if (p.flip==SDL_FLIP_HORIZONTAL&&p.get_mPosX()>get_mPosX())
332         {
333             if (p.index[1]==2)
334             {
335
336                 if (def_or_not == true)
337                 {
338                     Mix_PlayChannel( -1, atk_def, 0 );
339                     hp -= 0.2*p.atk*( 1.0 - def/200.0)*0.1;
340                     index[1] = 4;
341                 }
342                 else
343                 {
344                     Mix_PlayChannel( -1, atkk, 0 );
345                     hp -= p.atk *( 1.0 - def/200.0)*0.1 ;
346                     index[1] = 5;
347                 }
348                 if (hp<=0)
349                 {
350                     live = false;
351                     index[1] = 6;
352                 }
353             }
354         }
```

player(player_tool.h)

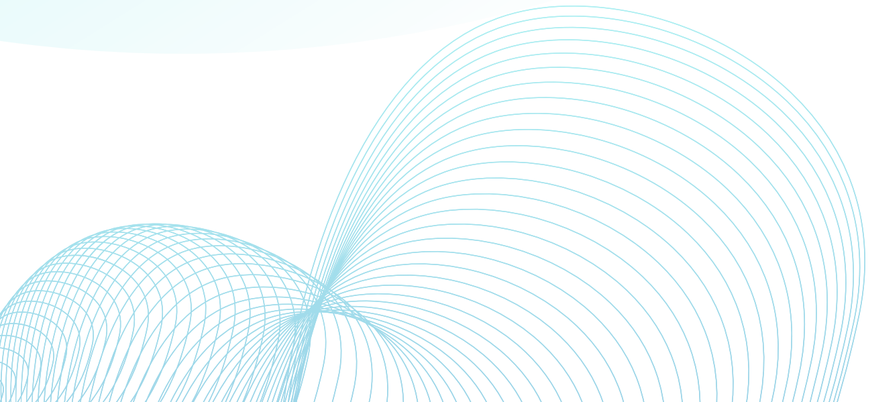
- **static member:**

1. player_num(private, line 158): it is declared as static member because we need to know the current player on the field.
2. initialize(line 166): has to be initialized outside the function
3. all player object share the same value, so we declare it as static

```
117 class player: public character{
118     public:
119         friend void render_blood(SDL_Renderer* &gRenderer,player &p,int type);
120         friend void get_prop(player&p,Tool &item,Uint32);
121         friend void Tool::sqrt_10(player &p);
122         friend void Tool::starbuck(player &p);
123         friend void Tool::big_note(player &p);
124         friend void Tool::withdraw(player &p);
125         friend void render_picture(SDL_Renderer* &gRenderer,player &p,int type);
126         const int character_WIDTH;
127         const int character_HEIGHT;
128         bool live;
129         player():live(false),character(),character_WIDTH(50),character_HEIGHT(120),s
130         }
131         ~player(){}
132         void set(int type,int blood,int attack,int defence,int xpos,int ypos,int spe
133         {
134             live = true;
135             index[0] = type;
136             hp = blood;
137             in_hp = blood;
138             atk = attack;
139             in_atk = attack;
140             def = defence;
141             in_def = defence;
142             mPosX = xpos;
143             mPosY = ypos;
144             character_VEL = speed;
145             flip = F;
146         }
```

```
158             static int player_num;
159             double hp;
160             int atk, def;
161             int in_atk,in_def;
162             double in_hp;
163             Uint32 starttime;
164
165 };
166 int player::player_num = 0;
```

PROGRAM OPTIMIZATION AND OTHER TECHNIQUES



main

```
238         player *p1, *p2;
239         Tool *item;
240         p1 = new player;
241         p2 = new player;
242         item = new Tool[2]
```

- **p1,p2,item:**

1. we declare them as pointer and delete them when the game ends
2. when we press enter button, it terminates the program faster than we declare p1,p2,item as object

```
436                     case 6:
437                         {
438                             if (e.key.type==SDL_KEYDOWN)
439                             {
440                                 switch(e.key.keysym.sym)
441                                     case SDLK_RETURN:
442                                         return 0;
443                             }
444                         }
445                     case 7:
446                         {
447                             if (e.key.type==SDL_KEYDOWN)
448                             {
449                                 switch(e.key.keysym.sym)
450                                     case SDLK_RETURN:
451                                         return 0;
452                             }
453                         }
454                 }
455             }
482                     case 6:
483                         {
484                             Mix_HaltMusic();
485                             Mix_PlayChannel( -1, winn, 0 );
486                             if (nextstate==true)
487                             {
488                                 delete p1;
489                                 delete p2;
490                                 delete []item;
491                                 nextstate = false;
492                                 break;
493                             }
494                             over1(gRenderer);
495                             break;
496                         }
497                     case 7:
498                         {
499                             Mix_HaltMusic();
500                             Mix_PlayChannel( -1, winn, 0 );
501                             if (nextstate==true)
502                             {
503                                 delete p1;
504                                 delete p2;
505                                 delete []item;
506                                 nextstate = false;
507
508                                 break;
509                             }
510                             over2(gRenderer);
511                             break;
512                         }
513                 }
```

main

- **sound effect:**

1. background music(line 271)
2. stops the background music if game ends(line 484, 499), and play a sound effect(line 485, 490)

```
269
270         //Play the music
271         Mix_PlayMusic( background, -1 );
272
273         //While application is running
274         while( !quit )
275         {
276             int count = 0;
277             //Handle events on queue
278             while( SDL_PollEvent( &e ) != 0 )
279             {
```

```
482             case 6:
483                 {
484                     Mix_HaltMusic();
485                     Mix_PlayChannel( -1, winn, 0 );
486                     if (nextstate==true)
487                     {
488                         delete p1;
489                         delete p2;
490                         delete []item;
491                         nextstate = false;
492                         break;
493                     }
494                     over1(gRenderer);
495                     break;
496                 }
497             case 7:
498                 {
499                     Mix_HaltMusic();
500                     Mix_PlayChannel( -1, winn, 0 );
501                     if (nextstate==true)
502                     {
503                         delete p1;
504                         delete p2;
505                         delete []item;
506                         nextstate = false;
507
508                         break;
509                     }
510                     over2(gRenderer);
511                     break;
512                 }
```

main

- **sound effect:**

1. when we move the button to the correct place, a sound effect will be played(line 338)
2. when we press the button, a different sound effect will be played(line 350)

```
332                 case 2:
333                     for (int i=1;i<=4;i++)
334                     {
335                         gButtons[i].handleEventCharacter( &e );
336                         if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
337                         {
338                             Mix_PlayChannel( -1, mouse_over, 0 );
339                             render = true;
340                             switch(i)
341                             {
342                                 case 1:render1 = true;break;
343                                 case 2:render2 = true;break;
344                                 case 3:render3 = true;break;
345                                 case 4:render4 = true;break;
346                             }
347                         }
348                         if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
349                         {
350                             Mix_PlayChannel( -1, button, 0 );
351                             render = false;
352                             switch(i)
353                             {
354                                 case 1:p1->set(0,400,100,100,150,350,11,SDL_FLIP_NONE); render1 = false;break;
355                                 case 2:p1->set(1,400,135,80,150,350,10,SDL_FLIP_NONE); render2 = false;break;
356                                 case 3:p1->set(2,340,80,80,150,350,15,SDL_FLIP_NONE); render3 = false;break;
357                                 case 4:p1->set(3,420,90,140,150,350,5,SDL_FLIP_NONE); render4 = false;break;
358                             }
359                             gamestate = 3;
360                         }
361                         if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
362                         {
363                             switch(i)
364                             {
365                                 case 1:render1 = false;break;
366                                 case 2:render2 = false;break;
367                                 case 3:render3 = false;break;
368                                 case 4:render4 = false;break;
373                 case 3:
374                     for (int i=5;i<=8;i++)
375                     {
376                         gButtons[i].handleEventCharacter( &e );
377                         if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OVER_MOTION)
378                         {
379                             Mix_PlayChannel( -1, mouse_over, 0 );
380                             switch(i)
381                             {
382                                 case 5:render5 = true;break;
383                                 case 6:render6 = true;break;
384                                 case 7:render7 = true;break;
385                                 case 8:render8 = true;break;
386                             }
387                         }
388                         if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_DOWN)
389                         {
390                             Mix_PlayChannel( -1, button, 0 );
391                             switch(i)
392                             {
393                                 case 5:p2->set(0,400,100,100,750,350,11,SDL_FLIP_HORIZONTAL);render5 = false;break;
394                                 case 6:p2->set(1,400,135,80,750,350,10,SDL_FLIP_HORIZONTAL);render6 = false;break;
395                                 case 7:p2->set(2,340,80,80,750,350,15,SDL_FLIP_HORIZONTAL);render7 = false;break;
396                                 case 8:p2->set(3,420,90,140,750,350,5,SDL_FLIP_HORIZONTAL);render8 = false;break;
397                             }
398                             gamestate = 4;
399                         }
400                         if (gButtons[i].getmyCurrentSprite() == BUTTON_SPRITE_MOUSE_OUT)
401                         {
402
403                             switch(i)
404                             {
405                                 case 5:render5 = false;break;
406                                 case 6:render6 = false;break;
407                                 case 7:render7 = false;break;
408                                 case 8:render8 = false;break;
409                             }
410                         }
```

player(player_tool.h)

- **sound effect:**

1. if a player is attacked and he is not defending himself, a sound effect will be played(line 344)
2. if a player is attacked and he is defending himself, a different sound effect will be played(line 338)

```
325  void player::be_attacked(player &p)
326  {
327      int p_mPosX = p.get_mPosX();
328      int p_mPosY = p.get_mPosY();
329      if( abs(mPosX-p_mPosX) <= character_WIDTH+10 && abs(mPosY-p_mPosY) <= character_HEIGHT ) //
330      {
331          if (p.flip==SDL_FLIP_HORIZONTAL&&p.get_mPosX()>get_mPosX())
332          {
333              if (p.index[1]==2)
334              {
335
336                  if (def_or_not == true)
337                  {
338                      Mix_PlayChannel( -1, atk_def, 0 );
339                      hp -= 0.2*p.atk*( 1.0 - def/200.0)*0.1;
340                      index[1] = 4;
341                  }
342                  else
343                  {
344                      Mix_PlayChannel( -1, atkk, 0 );
345                      hp -= p.atk *( 1.0 - def/200.0)*0.1 ;
346                      index[1] = 5;
347                  }
348                  if (hp<=0)
349                  {
350                      live = false;
351                      index[1] = 6;
352                  }
353              }
354          }
355          if (p.flip==SDL_FLIP_NONE&&p.get_mPosX()<get_mPosX())
```