

1. FOR STRING INPUT

```
class StringInput
{
    public:
        //The storage string
        std::string str;

        //The text surface
        SDL_Surface *text;

        public:
            //Initializes variables
            StringInput();

            //Does clean up
            ~StringInput();

            //Handles input
            void handle_input();

            //Shows the message on screen
            void show_centered();
};

StringInput::StringInput()
{
```

```

//Initialize the string

str = "";


//Initialize the surface

text = NULL;


//Enable Unicode

SDL_EnableUNICODE( SDL_ENABLE );
}


StringInput::~StringInput()
{
    //Free text surface

    SDL_FreeSurface( text );


    //Disable Unicode

    SDL_EnableUNICODE( SDL_DISABLE );
}


void StringInput::handle_input()
{
    //If a key was pressed

    if( event.type == SDL_KEYDOWN )
    {
        //Keep a copy of the current version of the string

        std::string temp = str;
    }
}

```

```

//If the string less than maximum size
if( str.length() <= 16 )
{
    //If the key is a space
    if( event.key.keysym.unicode == (Uint16)' ' )
    {
        //Append the character
        str += (char)event.key.keysym.unicode;
    }

    //If the key is a number
    else if( ( event.key.keysym.unicode >= (Uint16)'0' ) && ( event.key.keysym.unicode <=
(Uint16)'9' ) )
    {
        //Append the character
        str += (char)event.key.keysym.unicode;
    }

    //If the key is a uppercase letter
    else if( ( event.key.keysym.unicode >= (Uint16)'A' ) && ( event.key.keysym.unicode <=
(Uint16)'Z' ) )
    {
        //Append the character
        str += (char)event.key.keysym.unicode;
    }

    //If the key is a lowercase letter
    else if( ( event.key.keysym.unicode >= (Uint16)'a' ) && ( event.key.keysym.unicode <=
(Uint16)'z' ) )
    {
        //Append the character
        str += (char)event.key.keysym.unicode;
    }
}

```

```

    }
}

//If backspace was pressed and the string isn't blank
if( ( event.key.keysym.sym == SDLK_BACKSPACE ) && ( str.length() != 0 ) )
{
    //Remove a character from the end
    str.erase( str.length() - 1 );
}

//If the string was changed
if( str != temp )
{
    //Free the old surface
    SDL_FreeSurface( text );

    //Render a new text surface
    text = TTF_RenderText_Solid( font, str.c_str(), textColor );
}
}

void StringInput::show_centered()
{
    //If the surface isn't blank
    if( text != NULL )
    {

```

```

        //Show the name

        apply_surface( ( SCREEN_WIDTH - text->w ) / 2, ( SCREEN_HEIGHT - text->h ) / 2, text, screen );

    }

}

bool PlayerDetails(int num)

{

    SDL_FreeSurface(NumPlayers);

    SDL_WM_SetCaption( "Player Details", NULL );

    char* Names = new char[16*num];

    SDL_Surface* BackgroundInput = load_image("BckIn.png");

    if( BackgroundInput == NULL )

    {

        return false;//false;

    }

    apply_surface(0,0,BackgroundInput,screen);

    SDL_Flip(screen);

    bool nameEntered = false;

    bool quitPD = false;

    if( TTF_Init() == -1 )

    {

        return false;// false;

    }

    StringInput name;

    int counter = 0;

    //Open the font

    font = TTF_OpenFont( "lazy.ttf", 42 );

    if( font == NULL )

```

```

{
    return false;
}

message = TTF_RenderText_Solid( font, "Enter name for plyaer 1 :", textColor );

while( quitPD == false )
{
    //While there's events to handle
    while( SDL_PollEvent( &event ) )
    {
        //If the user has Xed out the window
        if( event.type == SDL_QUIT )
        {
            //Quit the program
            quitPD = true;
        }

        //If the user hasn't entered their name yet
        if( nameEntered == false )
        {
            //Get user input
            name.handle_input();

            //If the enter key was pressed
            if( ( event.type == SDL_KEYDOWN ) && ( event.key.keysym.sym == SDLK_RETURN ) )
            {
                //Change the flag
                //nameEntered = true;
            }
        }
    }
}

```

```

//Free the old message surface
SDL_FreeSurface( message );

SDL_FreeSurface(name.text);

name.text = NULL;

for(int lv = 0 ; lv < num ; lv++)

{
    Names[(16*counter)+lv] = name.str[lv];
}

name.str="";

counter++;

//Change the message
char s[23] = "Enter name for player ";

s[23] = counter+1;

message = TTF_RenderText_Solid( font, s , textColor );

if (counter==num)

{
    nameEntered=true;

    SDL_FreeSurface(BackgroundInput);

    SDL_FreeSurface(message);

    return false;
}

}

}

```

```

apply_surface( ( SCREEN_WIDTH - message->w ) / 2, ( ( SCREEN_HEIGHT / 2 ) - message->h ) / 2,
message, screen );

```

```

if( SDL_Flip( screen ) == -1 )

```

```

{

```

```

        return false;// 1;
    }
}

return false;
}

bool HowToPlay()// Flashes a short paragraph on how to play the game
{
    SDL_Surface* H2P = load_image("H2P.bmp");
    apply_surface(0,0,H2P,screen);
    SDL_Flip(screen);
    int xH2P,yH2P;
    bool quith2P;
    while( quith2P == false )
    {
        //If there's events to handle
        if( SDL_PollEvent( &event ) )
        {
            //If the user has Xed out the window
            if( event.type == SDL_QUIT )
            {
                //Quit the program
                quith2P = true;
                //return false;
            }
            if( event.type == SDL_MOUSEBUTTONDOWN )
            {

```



```

        //Get the mouse offsets

        xH2P = event.button.x;

        yH2P = event.button.y;


        //return 0 for PLayer

        if(yH2P>405)

            return true;


    }}

}

return true;

}

```

2. ONE-PLAYER CODE

```

struct CompMove

{

    int x;

    int y;

    double Ratio;

};

void MoveComp(int M[][6][2],int a , int b, int ID)

{

    if(M[a][b][1]==ID)

    {

```

```

        M[a][b][0] += M[a][b][1];

        Blast(a,b,M,ID);

    }

    else

    {

        M[a][b][1] = ID;

        M[a][b][0] += M[a][b][1];

        Blast(a,b,M,ID);

    }

}

double FindRatio(int M[][6][2],int sID , int eID)

{

    int s=0,e=0;

    for(int A=0 ; A<8 ; A++)

    {

        for(int B=0; B<6 ; B++)

        {

            if(M[A][B][1]==sID)

                s+=M[A][B][0]/M[A][B][1];

            if(M[A][B][1]==eID)

                e+=M[A][B][0]/M[A][B][1];

        }

    }

    double R=0;

    if(e==0)

        R=142.0;

    else

```

```

    R = s/e;

    return R;
}

CompMove Simulate( int GR[][6][2], int enemyID, int selfID)
{
    int Map[8][6][2];

    int ii,jj;

    for(ii = 0 ; ii<8 ; ii++ )
    {
        for(jj=0; jj<6 ; jj++)
        {
            if(GR[ii][jj][1]==1)
            {
                Map[ii][jj][1] = 8;

                Map[ii][jj][0] = Map[ii][jj][0]* 8;
            }

            if(GR[ii][jj][1]==2)
            {
                Map[ii][jj][1] = 7;

                Map[ii][jj][0] = (Map[ii][jj][0]/2)* 7;
            }
        }
    }

    CompMove temp;

    CompMove TheMove;

    //copy array

    for(int i=0; i<8; i++)

```

```

{
    for(int j=0 ; j<6 ; j++)
    {
        if(Map[i][j][1]!=enemyID)
        {
            MoveComp(Map,i,j,selfID);

            int PMap[8][6][2];

            for(ii=0;ii<8;ii++)
            {
                for(jj=0;jj<6;j++)
                {
                    PMap[ii][jj][0] = Map[ii][jj][0];
                    PMap[ii][jj][0] = Map[ii][jj][0];
                }
            }

            int Swap;

            for(int l=0; l<8 ;l++)
            {
                for(int J=0; J<6 ; J++)
                {
                    if(PMap[l][J][1]!=selfID)
                    {
                        MoveComp(PMap,l,J,enemyID);

                        double r = FindRatio(PMap,selfID,enemyID);

                        if (TheMove.Ratio < r)
                        {
                            TheMove.x = j;

```

```

        TheMove.y = i;

        TheMove.Ratio = r;

    }

}

for(ii=0;ii<8;ii++)

{

    for(jj=0;jj<6;jj++)

    {

        PMap[ii][jj][0] = Map[ii][jj][0];

        PMap[ii][jj][0] = Map[ii][jj][0];

    }

}

}

}

for(ii = 0 ; ii<8 ; ii++ )

{

    for(jj=0; jj<6 ; jj++)

    {

        if(GR[ii][jj][1]==1)

        {

            Map[ii][jj][1] = 8;

            Map[ii][jj][0] = GR[ii][jj][0] *8;

        }

        if(GR[ii][jj][1]==2)

        {

            Map[ii][jj][1] = 7;

```

```

        Map[ii][jj][0] = (Map[ii][jj][0]/2)* 7;
    }
}
}
}
}
return TheMove;
}

```

3. MOUSEOVER HIGHLIGHT

```

/*if(event.type == SDL_MOUSEMOTION)
{
    x = event.motion.x;
    y = event.motion.y;
    if((x<360) || (y<480))
    x = x/60;
    x = x*60;
    y = y/60;
    y = y*60;
    SDL_Surface* highlight = load_image("MouseOver.bmp");
    apply_surface(x,y,highlight,screen);
    Y = x/60;
    X = y/60;
    if(Grid[X][Y][1]!=0)
    BlitBalls(Grid[X][Y][1],Grid[X][Y][0]/Grid[X][Y][1],x,y,Grid);
    SDL_Flip(screen);
}

```

```
if(((floor(event.motion.x/60)!=Y)) || (floor(event.motion.x/60)!=X))  
  
SDL_FreeSurface(highlight);  
  
SDL_Flip(screen);  
  
}*/
```