

PROYECTO FINAL



L

Lost Artifacts

YOUR ADVENTURE STARTS NOW

MARIBEL ITZEL MENDEZ ASCENCIO
MILKA YAMIL TRINIDAD GUTIÉRREZ

Introducción

- Juego 2D
- Un solo jugador
- Tipo arcade game
- Situado en un mundo de fantasía



Eres de los mejores mercenarios de Fiory, ciudad libre que queda justo en medio de los reinos Elvin, Seemar, HailFrost y el bosque oscuro del continente. Creciste y vives en esta ciudad cazando monstruos del bosque oscuro que logran infiltrarse en los reinos por dinero. Un día los reyes y reinas de cada reino te ofrecieron un trabajo que no puedes rechazar.



Información dada por los reyes de cada reino:

Un dragón, especie que se creía extinta desde hace años, ha robado los artefactos cuyos poderes protegen a cada reino de los monstruos del bosque oscuro, junto a sus guardianes (espíritu de animales poderosos que siempre están juntos a los artefactos). Hasta ahora, no se han podido recuperar los artefactos, pero saben que el dragón los ha colocado en el bosque oscuro, por lo que habrá monstruos al que te enfrentaras, así como un guardián que protege a cada artefacto.

MISIÓN

Recuperar los 3 artefactos de cada reino y enfrentarte al dragón. Si tienes el artefacto, el guardián aparecerá, y viceversa.

RECOMPENZAS

Vida de perezoso. Te darán una mina de minerales del reino en Elvin, Una mansión cerca de los mares en Seemar, y el arma de tu elección hecha por el mejor artesano de HailFrost.

FRACASO

La muerte.

PERSONAJES

Principal

Jugador quien desea recuperar los artefacto y obtener las recompensas prometidas.

Enemigo

Monstruo del bosque oscuro que ataca a humanos que entran a su territorio.

Guardian

Espiritu poderoso que protege a un artefacto. Como no es un ser vivo como tal, no se le puede hacer daño





Fuerza de ataque: 3 puntos

INVENTARIO:
Plantas/Shigmul:10
Gemas:0

JUGADOR

- Inicia el primer nivel con 100 puntos. Se bajan 1 punto cada que entra en contacto con un monstruo. Baja 1 punto si lo ataca un guardián.
- Gana 25 puntos cada que encuentre un artefacto.
- Si llega a cero en un nivel,pierde el juego y debe iniciar de nuevo.
- Los puntos se van acumulando cada nivel.



Enemigo: Mosutrus



Fuerza de ataque: 1 punto
Debilidad: Shigmul/Planta

- Sigue al jugador y le hace daño a este al entrar en contacto con ellos. Inicia con 50 puntos de vida en el primer nivel.
- Un ataque de él le resta 1 punto al jugador.
- Luego de atacar al jugador, reposa por 3 segundos y comienza a seguirlo de nuevo.
- Única debilidad son la hierba Shigmul:



Guardián: Búho



Fuerza de ataque: 1 punto
Debilidad: Nada

- Se mantiene en **estado IDLE** cuando inicia el nivel.
- Comienza a atacar al jugador cuando el jugador entra en contacto con él por primera vez.
- Se ataca al jugador en tiempos aleatorios entre 5-15 segundos.
- El artefacto que **protege**, “**Escudo de madera**”, aparece cuando el búho está despierto y atacnado.
- Su ataque le resta 1, 2 y 3 puntos al jugador en el nivel correspondiente.
- **El jugador no le puede hacer daño.**



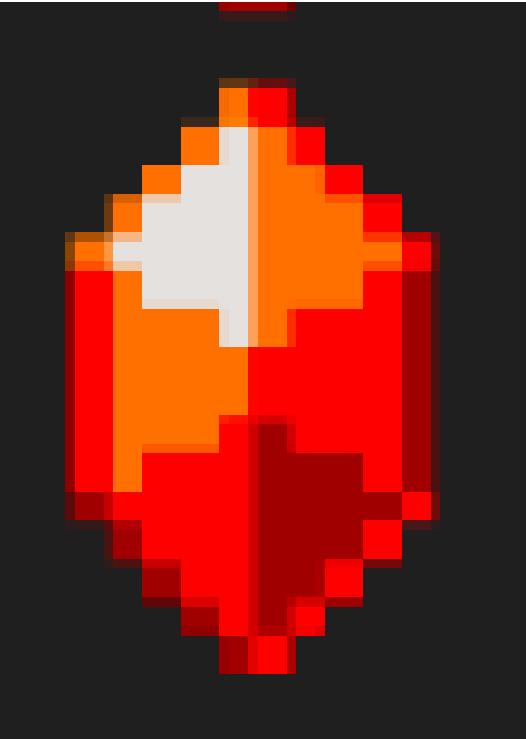
Artefacto: Escudo de Madera



Paleta de colores:
Diferentes tonos de gris,
café, rojo y blanco.

- Símbolo y artefacto mágico de madera que protege al reino Elvin, manteniendo sus fronteras a salvo y oculto de los monstruos.
- Le da al usuario el poder de protegerse contra cualquier ataque, incrementando la defensa del jugador de un 1.5 a 2.

Gemas



Inventario



- Gemas mágicas que solo se encuentran en el bosque oscuro y que los reinos dentro del juego consideran un lujo.
- Sirven para enfrentarse con el dragón y para comunicarse con él.
- Le da 2 puntos de vida al jugador cada gema que obtiene.

Inventario

- Hierba utilizada para combatir contra monstruos.
- Única debilidad que se ha encontrado de los monstruos.
- Crece en todo el bosque oscuro y en todas condiciones, incluyendo los diferentes niveles del juego.
- **La cantidad de puntos de vida que le quita al monstruo equivale a la fuerza de ataque del jugador**
 - Nivel 1= El shigmul les resta 3 puntos de vida.

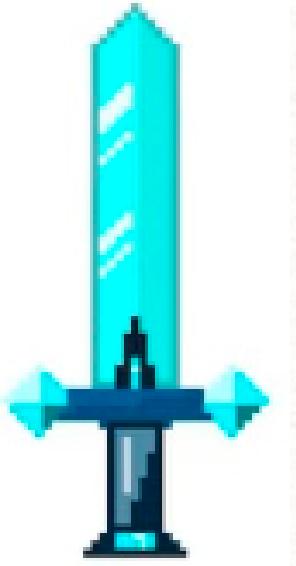
Shigmul / Plantas



Nivel 2

Guardian: Pinguino

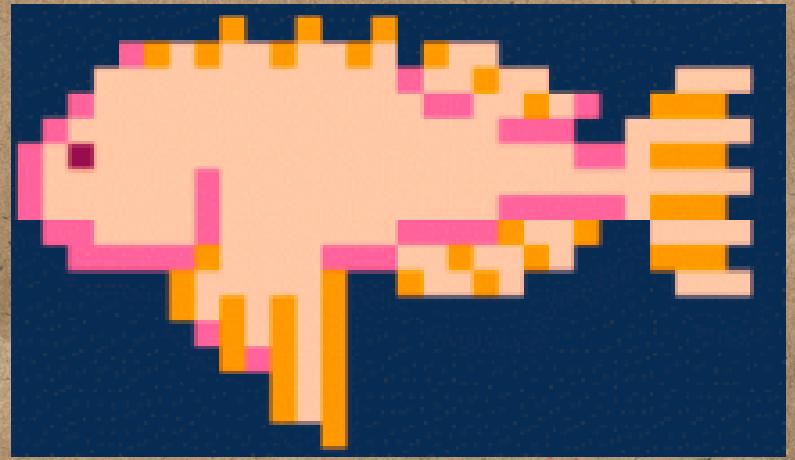
Artefacto: Espada de Hielo



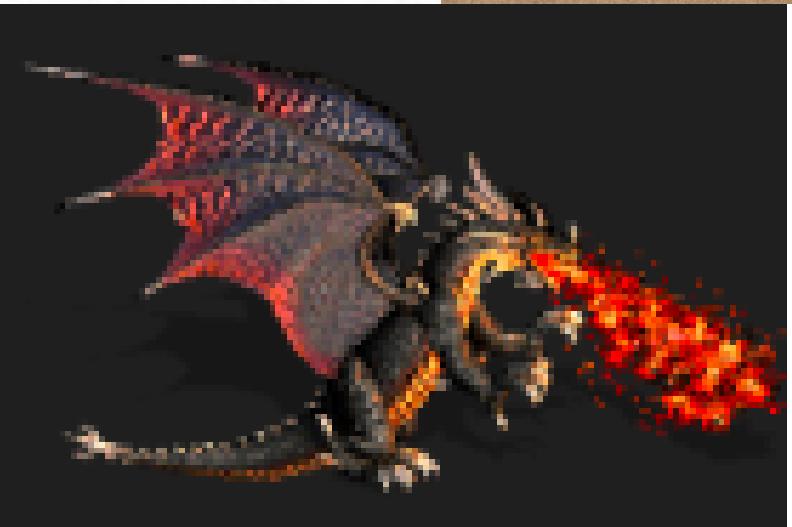
Nivel 3

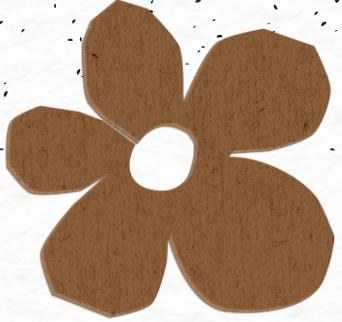
Guardian: Pez

Artefacto: Latigo de Agua



Dragon



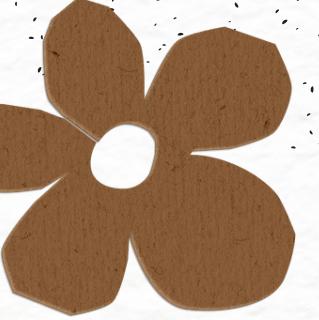


1er Nivel: laberinto

Este nivel consiste en un laberinto a la entrada de la guarida del dragón dentro del bosque oscuro hecho por el dragón para almacenar el artefacto Escudo de madera.



Se preciona L para iniciar el juego



Instrucciones:

- El protagonista debe de encontrar la ruta que lo lleve al primer artefacto.
- El nivel otorgará al protagonista gemas y Shigmul.,
- El Shigmul puede ser recogido y usado por el jugador para atacar al monstruo.
- Los ataques del jugador impactan a todo el cuarto donde esta, por lo que puede atacar desde cualquier lugar al monstruo, pero solo cada 10 segundos(el juego se lo indicara)





Instrucciones:

- El monstruo puede dañar al protagonista al momento de colisionar con él, por lo cual el jugador tendrá que decidir si matar al monstruo o escapar de él.
- Cuando el usuario haya encontrado al primer protector, este hará visible al artefacto, es decir al escudo.
- Y en cuanto haga contacto el jugador con el artefacto, el jugador acabará y ganara el nivel.
- Si el jugador muere antes de encontrar el artefacto, acaba el juego.



Características

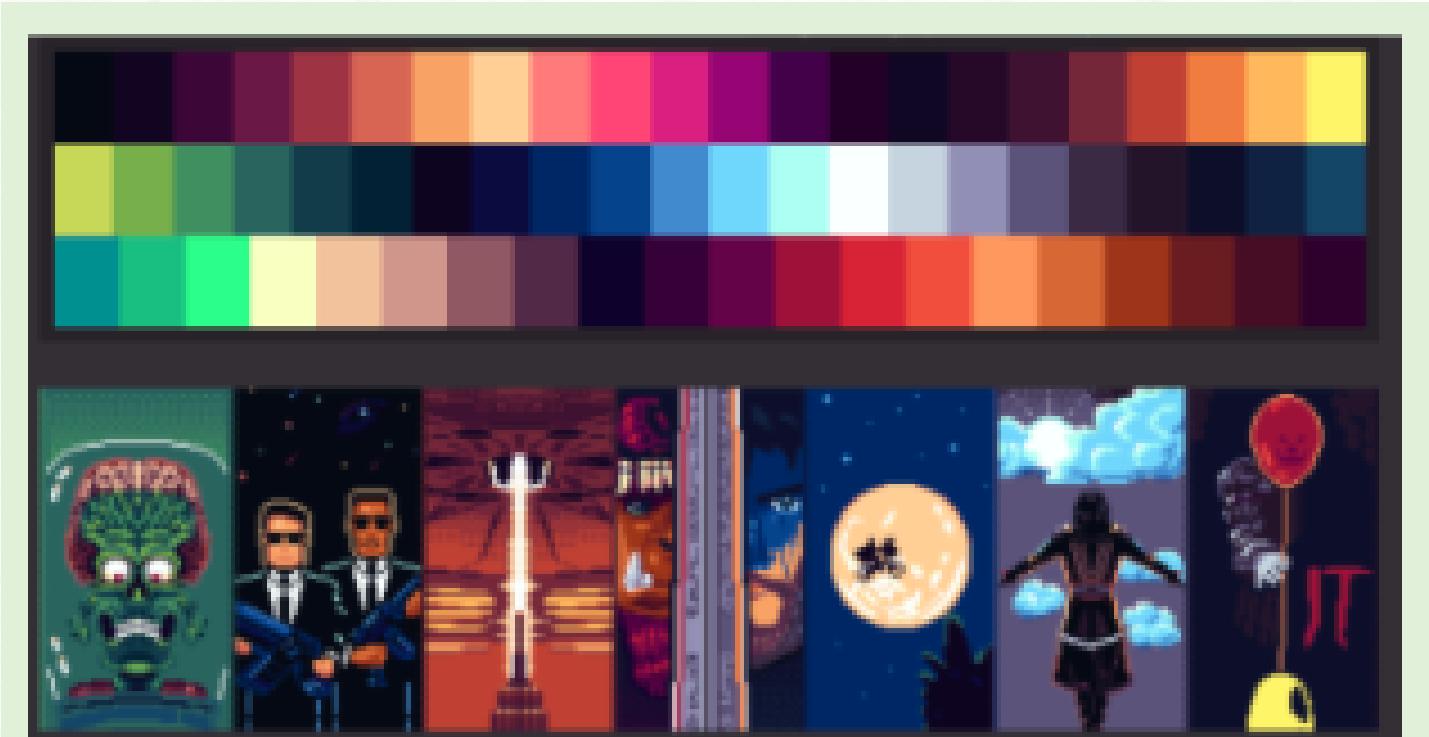


Figure: 6 bits (64 colores)

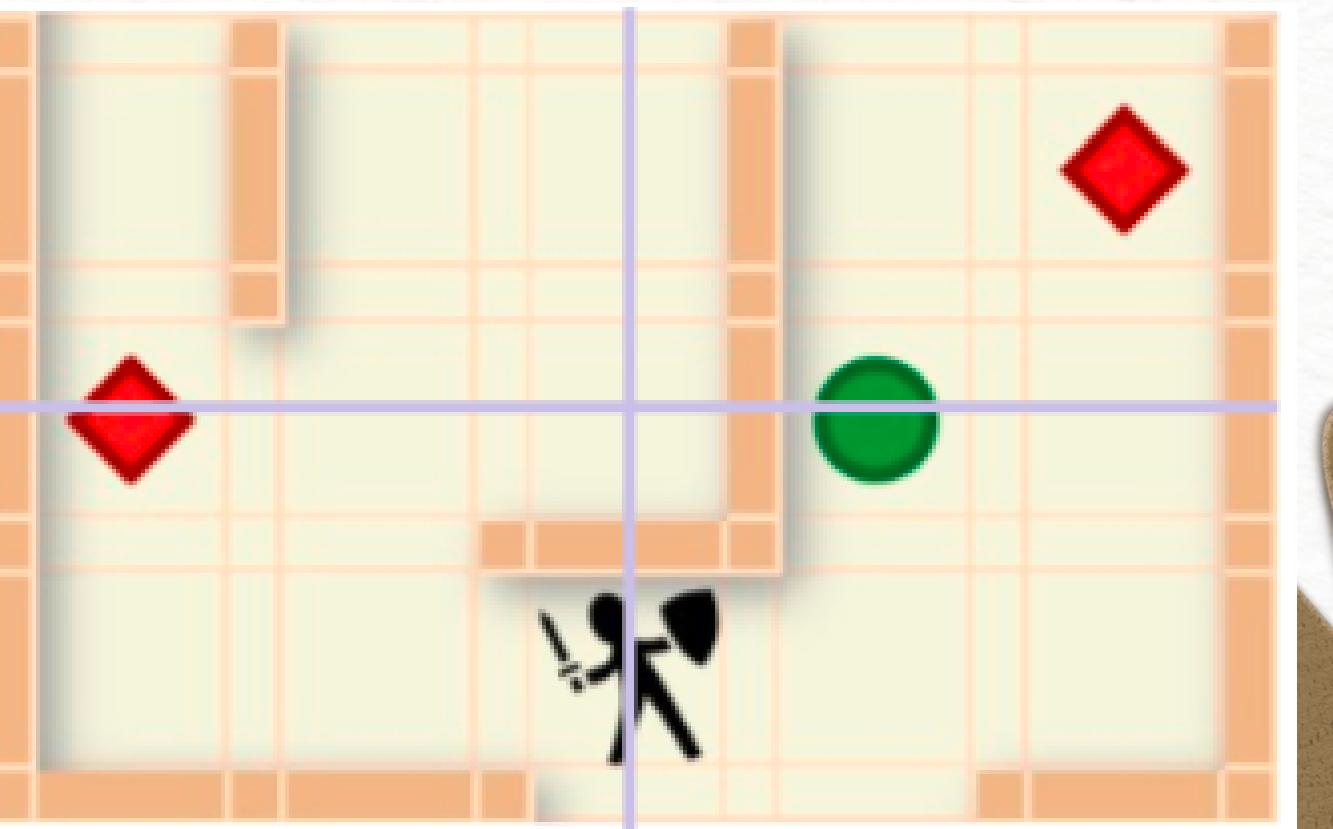
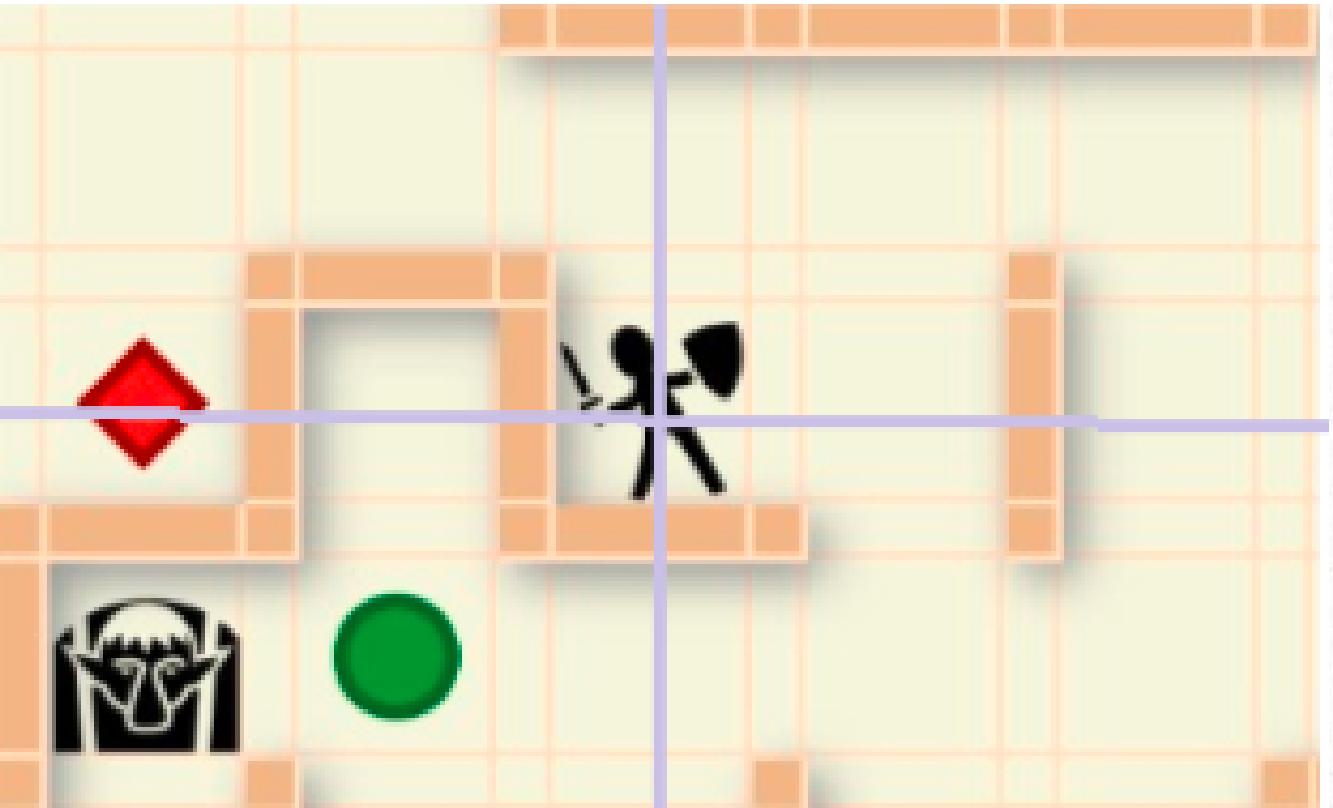
Tipo de arte: PIXEL ART

Escogimos el “pixel art” debido a que es un arte conceptual muy llamativo de ver y al ser un arte visualmente sencillo creemos que esto le puede dar un toque más fantasioso y clásico a nuestro juego cuyo mundo e historia es de fantasía.

Características

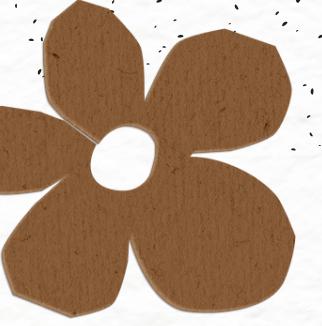
TIPO DE CÁMARA: Position-Locking

La cámara escogida para el videojuego fue la de Position-Locking, y se refiere a que la cámara se queda fija en la posición de un objeto, cuando dicho objeto se mueve, también lo hace la cámara, dando el efecto que lo va siguiendo. En el caso de nuestro juego, cada nivel del mundo se desarrolla en una especie de cuarto conforme a una temática relacionado a uno de los artefactos que debe de recuperar el jugador. Por lo tanto, el jugador podrá recorrer todo el cuarto hasta llegar a los bordes de este por lo que la camara igual solo mostrara las partes del mundo mas un espacio reducido de color blanco indicando que ya no existe mas informacion relevante fuera del cuarto.





Mecanismos de Clase



Camara

- Crear camara e inicializar atributos

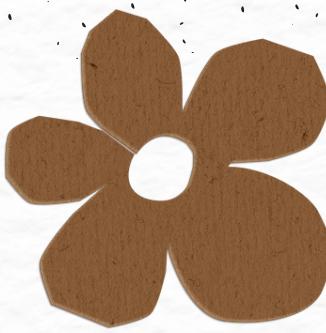
```
void Camara::input_handle(KeyOyente &input, MouseOyente &mouse)
{
    if(!estado_actual)
        return; //estado nulo al inicio
    FSMCamara* estado = estado_actual->input_handle(*this,input,mouse);
    if(estado) {
        estado_actual->on_salir(*this);
        delete estado_actual;
        estado_actual = estado;
        estado_actual->on_entrar(*this);
    }
}

void Camara::update()
{
    if(start==true) checar_pos_jugador();
    if(estado_actual)
        estado_actual->on_update(*this);
}
```

```
void Camara::checar_pos_jugador(){
    int x1=limites.at(0).x;
    int x2=limites.at(1).x;
    int y1=limites.at(0).y;
    int y2=limites.at(1).y;

    int posXObjeto = get_obj_lock()->get_posicion_mundo().x;
    int posYObjeto = get_obj_lock()->get_posicion_mundo().y;
    if(posXObjeto<x1 || posXObjeto>x2 || posYObjeto<y1 || posYObjeto>y2)
    {
        if(lock==0){ lock=1; }
    }else{
        if(lock==1){
            jugador=objeto_seguir;
            lock=0;
        }
    }
}
```

- Seguir o dejar de seguir un objeto dependiendo si esta dentro del limite o no



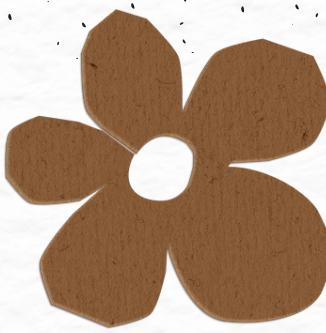
EstadoMover(IDLE)

- Quedarse en una posicion si la camara no esta siguiendo al objeto(lock=0)

```
EstadoCamaraMover::EstadoCamaraMover(Coordenadas dir)
{
    direccion = dir;
    strestado = "mover";
};

FSMCamara* EstadoCamaraMover::input_handle(Camara &cam, KeyOyente &keys, MouseOyente& mouse)
{
    if(cam.lock)
    {
        return new EstadoCamaraTransicion();
    }else
        return new EstadoCamaraMover({0,0});
};

void EstadoCamaraMover::on_entrar(Camara &cam){
    vel = cam.velocidad;
};
void EstadoCamaraMover::on_salir(Camara &cam){};
void EstadoCamaraMover::on_update(Camara &cam)
{
    Coordenadas p=cam.get_posicion_mundo();
    p.x =(int)(vel*direccion.x);
    p.y =(int)(vel*direccion.y);
    cam.set_posicion_mundo(p);
};
```



EstadoTransicion

- Cambiar de estado a atacar para estar en la misma posicion que el jugador si está una distancia de 10 del objeto que sigue

```
EstadoCamaraTransicion::EstadoCamaraTransicion()
{
    frames_actual =0;
    frames_maximo = 45;
    strestado = "transicion";
};

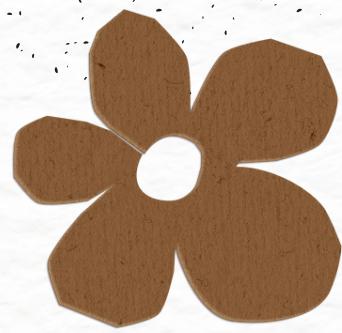
FSMCamara* EstadoCamaraTransicion::input_handle(Camara &cam, KeyOyente &keys, MouseOyente& mouse)
{
    if(!cam.lock){
        return new EstadoCamaraMover({0,0});
    }else if(frames_actual>frames_maximo)
    {
        if(distancia<10)
        {
            return new EstadoCamaraLock(*cam.get_obj_lock());
        }
        return new EstadoCamaraTransicion();
    }
    return NULL;
};
```

```
void EstadoCamaraTransicion::on_entrar(Camara &cam)
{
    cam.en_transicion = 1;
    vel = cam.velocidad;
    pos_inicial = cam.get_posicion_mundo();//(0,0)

    //seguro decir que este estado siempre entra en lock object
    // no es necesario comprobar que lock object no es nulo
    pos_final = cam.get_obj_lock()->get_posicion_mundo(); //o.x,o.y

    //convertir en posicion relativo a las coordenadas camara
    centro = cam.get_posicion_centro();
    pos_final.x -= centro.x;
    pos_final.y -= centro.y;

    //la diferencia es la cantidad de pixeles que se mueve
    ant_check={0,0};
    //checkar la distancia es pequena para ajustar los frames
    distancia = std::sqrt(std::pow(centro.x-(pos_final.x+centro.x),2)+std::pow(centro.y-(pos_final.y+centro.y),2));
    //std::cout<<"Distancia > "<<distancia<<"\n";
    float lim = cam.get_obj_lock()->get_avatar()->get_vertices()[3].x - cam.get_obj_lock()->get_avatar()->get_vertices()[0].x;
    if(distancia<lim){
        frames_maximo = 10;
    }
};
```

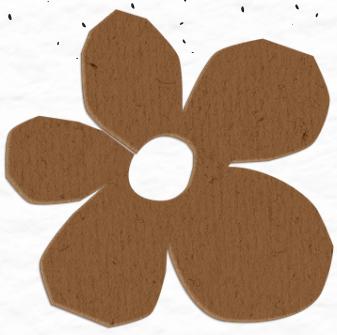


EstadoLock

- Cambia su posicion del mundo al del jugador para simular que lo esta siguiendo

```
EstadoCamaraLock::EstadoCamaraLock(Objeto &objlock)
{
    strestado = "lock";
    obj = &objlock;
};

FSMCamara* EstadoCamaraLock::input_handle(Camara &cam, KeyOyente &keys, MouseOyente& mouse)
{
    if(!cam.lock){
        return new EstadoCamaraMover({0,0});
    }
    return NULL;
};
void EstadoCamaraLock::on_entrar(Camara &cam)
{
    centro = cam.get_posicion_centro();
};
void EstadoCamaraLock::on_salir(Camara &cam){}
void EstadoCamaraLock::on_update(Camara &cam){
    // simplemente la camara obtiene la posicion del obj
    Coordenadas pos_mundo = obj->get_posicion_mundo();
    pos_mundo.x -= centro.x;
    pos_mundo.y -= centro.y;
    cam.set_posicion_mundo(pos_mundo);
};
```





Crear Ventana

```
bool SDLApp::on_init()
{
    //revisar que SDL inicio bien
    if(SDL_Init(SDL_INIT_EVERYTHING)<0){return false;};
    //crear la ventana
    get().vnt = SDL_CreateWindow(
        "Juego",           // Titulo de la ventana
        SDL_WINDOWPOS_UNDEFINED, // posicion X de la pantalla
        SDL_WINDOWPOS_UNDEFINED, // posicion Y de la pantalla
        get().WIDTH,        // ancho de la ventana
        get().HEIGHT,       // alto de la ventana
        SDL_WINDOW_OPENGL); // que use OPENGL

    TTF_Init();
    IMG_Init(IMG_INIT_PNG | IMG_INIT_JPG);
    //revisar que se creo bien la ventana
    if(get().vnt == NULL)
    {
        printf("No se creo la ventana por: %s",SDL_GetError());
        return false;
    }
    //la forma de procesar en GPU
```

Crear Render

```
    //la forma de procesar en GPU
    SDL_SetHint(SDL_HINT_RENDER_BATCHING,"opengl");
    //Creamos el 'canvas'
    get().render = SDL_CreateRenderer(
        get().vnt,           // la ventana
        -1,                  // driver
        SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC); // flags driver
    //revisamos si se creo correctamente
    if(get().render == NULL)
    {
        printf("No se creo el renderer por: %s",SDL_GetError());
        return false;
    }
    // si se creo correcto lo agregamos al Pipeline

    get().ensamble = new Pipeline(*get().render);
    // guardarMapa("assets/sprites/mundo/ids/FinalAtlas_ids.txt");
```

Crear mapa e inicializar booleanos

```
mapa = new Atlas("assets/sprites/mundo/ids/FinalAtlas_ids.txt");
mapa->generar_mapa(get().render,2,0);
posicionesPiso=mapa->get_posicionesPiso();
attack_on = false;
start = false;
game_over=false;
```

Generar posiciones aleatorias

```
//generar 3 posiciones random no repetidas de posicionesPiso
Coordenadas posJugador, posEnemigo, posGuardian;
    int random1, random2, random3;
    random1 = rand() % posicionesPiso.size();
    random2 = rand() % posicionesPiso.size();
    random3 = rand() % posicionesPiso.size();
    while(random1 == random2 || random1 == random3 || random2 == random3)
    {
        random1 = rand() % posicionesPiso.size();
        random2 = rand() % posicionesPiso.size();
        random3 = rand() % posicionesPiso.size();
    }
    posJugador = posicionesPiso[random1];
    posEnemigo = posicionesPiso[random2];
    posGuardian = posicionesPiso[random3];
```

Crear personajes

```
//07 Generar Jugador
player = new Jugador("assets/sprites/heroe/myHero.png",100,3, posJugador.x, posJugador.y,135,93,100,70,{255,0,255,255});
| | //80,1,300,400,135,93,100,70,{255,0,255,255});
get().ensamble->cargar_texturas(player->get_sprite());
player->set_plantas(10);

//Generar Enemigo
enemigo=new Enemigo("assets/sprites/heroe/monster.png",50,1, posEnemigo.x, posEnemigo.y,100,87,100,100,{255,0,255,255});//

//Generar Guardian
guardian=new Guardian("assets/sprites/heroe/buho3.png",15,1, posGuardian.x, posGuardian.y,60,50,100,100,{255,0,255,255});
get().ensamble->cargar_texturas(enemigo->get_sprite());
get().ensamble->cargar_texturas(guardian->get_sprite());
```

Generar inventario aleatorio(gemas, plantas, artefacto, tiles de ataque)

Inicializar tiles correspondientes como piso

```
//PINTAR GEMAS, plantas, artefacto y attackzones
plataformas = mapa->get_objetos_fisicos();
allTiles.push_back(mapa->get_tileGema());
allTiles.push_back(mapa->get_tilePlanta());
allTiles.push_back(mapa->get_tileArtefacto());
allTiles.push_back(mapa->get_tileBlanco());
allTiles.push_back(mapa->get_tilein());
for(auto &p:plataformas){
    int id = p->get_ID();
    if(id==ARTEFACTO && guardian->get_first_time()==0){
        p->set_tile(new Tile(allTiles[PISO-1]));
        p->set_ID(PISO);
    }
}
```

```
int numGemas = 0,numPlantas = 0,numBlancos=0;
bool existeArtefacto = false;
for(auto &p:plataformas){
    int id = p->get_ID();

    if(id==PISO && rand()%100<50){
        //HAY UN 20 PORCIETNO DE PROB DE QUE HAYA UNA GEMA
        if(rand()%100<5 && numGemas<=30){
            p->set_tile(new Tile(allTiles[GEMAS-1]));
            p->set_ID(GEMAS);
            numGemas++;
        }
        if(rand()%100<5 && numPlantas<=30){
            p->set_tile(new Tile(allTiles[PLANTAS-1]));
            p->set_ID(PLANTAS);
            numPlantas++;
        }
        //HAY UN 20 PORCIETNO DE PROB DE QUE HAYA UN ARTEFACTO
        if(rand()%100<5 && existeArtefacto==false){
            p->set_tile(new Tile(allTiles[ARTEFACTO-1]));
            p->set_ID(ARTEFACTO);
            existeArtefacto=true;
        } //HAY UN 20 PORCIETNO DE PROB DE QUE HAYA UN BLANCO
        if(rand()%100<5 && numBlancos<10){
            p->set_tile(new Tile(allTiles[ATTACK_ZONE-1]));
            p->set_ID(ATTACK_ZONE);
            numBlancos++;
        }
    }
}
```

- Crear camara principal
- Llenar arreglo de objetos
- Agregar el como del background frame

```
get().camara_principal = new Camara(0,0,get().WIDTH,get().HEIGHT,*get().render);
//get().camara_principal = new Camara(0,0,get().WIDTH,get().HEIGHT,enemigo,*get().render);
//camara_principal->lock_objeto(*player);

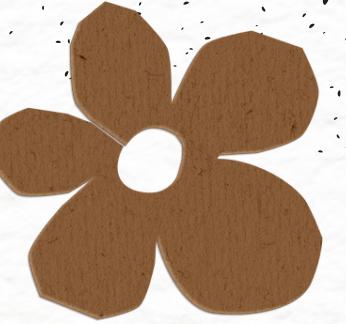
for(int i=0;i<plataformas.size();i++)
{
    //agregar todos los objetos en una lista para la camara
    objetos.push_back(plataformas[i]);
}

objetos.push_back(player);
objetos.push_back(enemigo);
objetos.push_back(guardian);
//Crear una plataforma ne medio de la pantalla

printf("Se crearon los test exitosamente\n");

//agregamos el color del background del frame
SDL_SetRenderDrawColor(
    get().render,          // canvas
    get().bg_color.r,     // rojo
    get().bg_color.g,     // verde
    get().bg_color.b,     // azul
    SDL_ALPHA_TRANSPARENT); // como usar el alpha

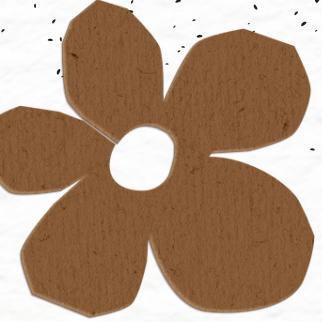
return true;
```



ATLAS
-Generar Mapa-

- Identificar cada tile y guardar la información del tile
- Guardar plataformas creadas

Atlas

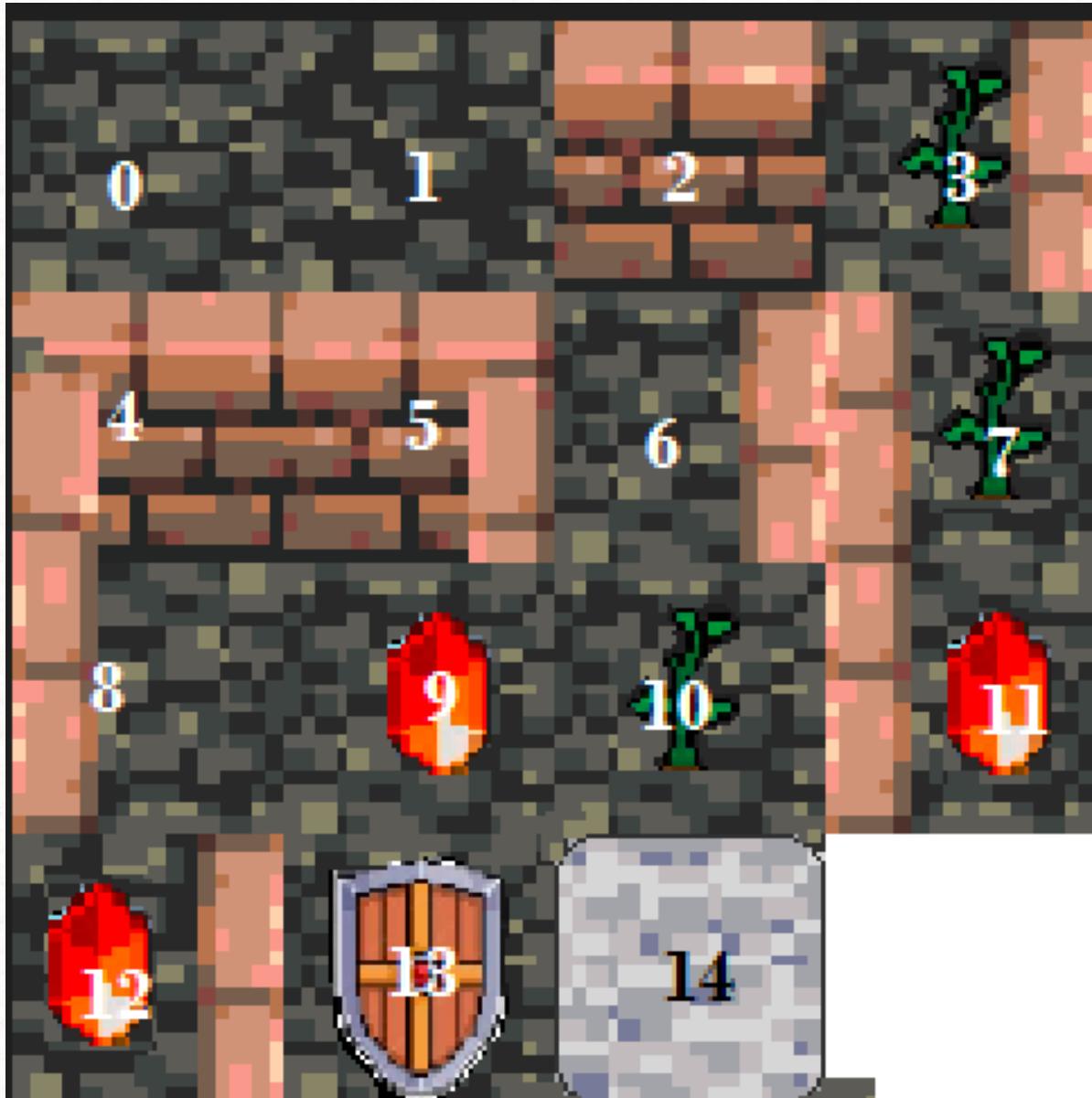


```

bool tiene_colision = false;
if(info.mapa_ids[i][j]==9 ){
    id=GEMAS;
    tileGema=tile;
}
else if(info.mapa_ids[i][j]==10 ){
    id=PLANTAS;
    tilePlanta=tile;
}
else if(info.mapa_ids[i][j]==1 ){
    id=PISO;
    tilein=tile;
    posicionesPiso.push_back({j*info.tile_width,i*info.tile_height});
}
else if(info.mapa_ids[i][j]==14 ){
    id=ATTACK_ZONE;
    tileBlanco=tile;
    // tilein=tile;
}
else if(info.mapa_ids[i][j]==13 ){
    id=ARTEFACTO;
    tileArtefacto=tile;
}
{
    id=MURO;
}

objetos_fisicos.push_back(new Plataformas(tile,tiene_colision,id));

```



A large, dark grey circular brushstroke is positioned in the center of the image, resembling a large drop or a splash. It has a textured, hand-painted appearance with visible brushstrokes.

SOLAPP

-Fisica Update-

Iniciar el juego

```
if(KeyOyente::get().estaPresionado(SDL_SCANCODE_L))
{
    start = true;
    camara_principal->start = true;
    enemigo->start = true;
    camara_principal->lock_objeto(*player);
    enemigo->lock_objeto(*player);
    guardian->lock_objeto(*player);
}

player->input_handle(KeyOyente::get(),MouseOyente::get());
```

Tiempo de ataque

```
double elapsed_tiempo = Tiempo::get_tiempo() - current_tiempo;
if (elapsed_tiempo > idle_tiempo) {
    if(attack_on==false)
    {
        attack_on = true;
    }
    else
    {
        attack_on = false;
    }
    player->set_attack_on(attack_on);
    current_tiempo = Tiempo::get_tiempo();
}
```



Checar si puede atacar
el jugador



Recorrer las plataformas
del mapa:

- Pintar tiles si es hora de atacar
- Checar si el jugador colisiona con un MURO

```
if(KeyOyente::get().estaPresionado(SDL_SCANCODE_J) && attack_on==true)
{
    if(player->get_plantas()>0){
        enemigo->set_hp(enemigo->get_hp()-player->get_ataque());
        player->set_plantas(player->get_plantas()-1);
        // printf("HP-enemigo: %2.f\n",enemigo->get_hp());
    }
}
```

```
for(auto &p:plataformas)
{
    // p->update(dt);
    int id=p->get_ID();
    if(id==ATTACK_ZONE){
        if(attack_on==true && start==true){
            p->set_tile(new Tile(allTiles[ATTACK_ZONE-1]));
        }else{
            p->set_tile(new Tile(allTiles[PISO-1]));
        }
    }
    if(id==MURO ){
        MotorFisico2D::get().diag_overlap_g(*player,*p);
        player->en_colision=false;
    }
}
```



Cambiar Gemas/Plantas a piso si colisionan con el jugador

```
if(id==GEMAS && start==true){  
    MotorFisico2D::get().diag_ovelap_g(*player,*p);  
    if(player->en_colision==true){  
        p->set_tile(new Tile(allTiles[PISO-1]));  
        p->set_ID(PISO);  
        player->en_colision=false;  
        player->set_hp(player->get_hp()+2);  
  
        player->set_colision_gemas(true);  
    }  
}  
  
if(id==PLANTAS && start==true){  
    MotorFisico2D::get().diag_ovelap_g(*player,*p);  
    if(player->en_colision==true){  
        p->set_tile(new Tile(allTiles[PISO-1]));  
        p->set_ID(PISO);  
        player->en_colision=false;  
        player->set_colision_plantas(true);  
        player->set_plantas(player->get_plantas()+1);  
    }  
}
```

Mostrar el artefacto si el jugador ya colisiona una vez con el guardian

```
if(id==ARTEFACTO && guardian->get_first_time()==0){  
    p->set_tile(new Tile(allTiles[PISO-1]));  
  
}else if(id==ARTEFACTO && guardian->get_first_time()==1){  
    p->set_tile(new Tile(allTiles[ARTEFACTO-1]));  
    MotorFisico2D::get().diag_ovelap_g(*player,*p);  
    if(player->en_colision==true){  
        p->set_tile(new Tile(allTiles[PISO-1]));  
        player->set_colision_artefacto(true);  
        player->set_hp(player->get_hp()+25);  
        player->game_over=true;  
        enemigo->game_over=true;  
        guardian->game_over=true;  
        camara_principal->game_over=true;  
        game_over=true;  
    }  
}
```

- Actualizar los objetos
- Checar si colisiona el jugador con algun otro personaje

```

for(auto &p:objetos)
{
    p->update(dt);
}

player->update(dt);
enemigo->update(dt);
MotorFisico2D::get().diag_ovelap_e(*player,*enemigo);
if(player->en_colision==true){
    player->set_hp(player->get_hp()-enemigo->get_ataque());
    player->en_colision=false;
}

guardian->update(dt);
MotorFisico2D::get().diag_ovelap_g(*player,*guardian);
if(player->en_colision==true){
    guardian->set_first_time(1);
    player->set_hp(player->get_hp()-guardian->get_ataque());
    player->en_colision=false;
}

```

- Checar si estan muertos el jugador y el monstruo
- Actualizar la camara y los objetos

```

if(player->get_hp()<=0){
    // printf("GAME OVER\n");
    game_over = true;
    player->game_over=true;
    enemigo->game_over=true;
    guardian->game_over=true;
    camara_principal->game_over=true;
}
if(enemigo->get_hp()<=0){
    // printf("ENEMIGO MUERTO\n");
    enemigo->set_muerto(true);
}

/*CAMARA al final para actualizar la proyección de los objetos*/
camara_principal->input_handle(KeyOyente::get(),MouseOyente::get());
camara_principal->update();
camara_principal->proyectar(objetos);
//printf("Update Fisica\n");

```

A large, dark gray circular brushstroke serves as the background for the central text.

SDLAPP
-Frame Update-

- Renderizar los objetos e imprimir las estadísticas si el juego inicio

```
//Renderizar todo a través de la camara
camara_principal->renderizar(objetos);
// camara_principal->render_cross();
std::string vidaJugador="Vida: "+std::to_string((int)player->get_hp());
std::string gemasJugador="Gemas: "+std::to_string((int)player->get_gemas());
std::string plantasJugador="Plantas: "+std::to_string((int)player->get_plantas());
std::string ataqueJugador="Fuerza de Ataque: "+std::to_string((int)player->get_ataque());
std::string vidaEnemigo="Vida: "+std::to_string((int)enemigo->get_hp());
std::string ataqueEnemigo="Fuerza de Ataque: "+std::to_string((int)enemigo->get_ataque());
std::string ataqueGuardian="Fuerza de Ataque: "+std::to_string((int)guardian->get_ataque());
// RenderTexto::get().render_texto(get().render,50,630,player->get_strEstado(),120,30,SDL_Color{0,0,0,255});
if(start==true){
    RenderTexto::get().render_texto(get().render,50,50,"JUGADOR",200,50,SDL_Color{255,0,0,255});
    RenderTexto::get().render_texto(get().render,50,90,vidaJugador,200,50,SDL_Color{255,0,0,255});
    RenderTexto::get().render_texto(get().render,50,130,gemasJugador,200,50,SDL_Color{255,0,0,255});
    RenderTexto::get().render_texto(get().render,50,170,plantasJugador,200,50,SDL_Color{255,0,0,255});
    RenderTexto::get().render_texto(get().render,50,200,ataqueJugador,200,50,SDL_Color{255,0,0,255});

    RenderTexto::get().render_texto(get().render,400,50,"ENEMIGO",150,30,SDL_Color{0,255,0,255});
    RenderTexto::get().render_texto(get().render,400,100,vidaEnemigo,150,30,SDL_Color{0,255,0,255});
    RenderTexto::get().render_texto(get().render,400,150,ataqueEnemigo,150,30,SDL_Color{0,255,0,255});

    RenderTexto::get().render_texto(get().render,700,50,"GUARDIAN",160,50,SDL_Color{0,0,255,255});
    RenderTexto::get().render_texto(get().render,700,100,ataqueGuardian,160,50,SDL_Color{0,0,255,255});
```



- Condicion que verifica si gano
- Condicion que verifica si perdio
- Mostrar mensaje de atacar
- Mostrar mensaje de como iniciar el juego

```
if(attack_on==true && player->get_colision_artefacto()==false && game_over==false && enemigo->get_muerto()==false){
    RenderTexto::get().render_texto(get().render,400,300,"ATTACK NOW",150,40,SDL_Color{255,255,0,255});
}
if(player->get_hp()<=0 && game_over==true){
    RenderTexto::get().render_texto(get().render,400,350,"GAME OVER",150,70,SDL_Color{255,0,255,255});
    enemigo->set_muerto(true);
    game_over=true;
}
if(player->get_colision_artefacto()==true){
    RenderTexto::get().render_texto(get().render,400,430,"YOU WIN",150,40,SDL_Color{255,0,255,255});
    enemigo->set_muerto(true);
    game_over=true;
}

if(start==false){
    RenderTexto::get().render_texto(get().render,400,300,"PRESS L TO START",200,90,SDL_Color{255,255,0,255});
}
```

- Esto es el metodo On_Correr que captura eventos y actualiza los metodos anteriormente explicados del juego mientras esta corriendo

```
int SDLApp::on_correr()
{
    //revisar que todo se inicializo bien
    if(get().on_init()==false){return -1;}
    SDL_Event eventos;
    double dt=0;
    double frecuencia = 1/get().maxFPS; // 1 frame a 60fps
    get().msfrecuencia = frecuencia*1000;

    while(get().estaCorriendo())
    {
        //double start = SDL_GetTicks();
        double inicio = Tiempo::get_tiempo();

        //printf("%lf <> %d\n",Tiempo::get_tiempo(),SDL_GetTicks());
        //captura eventos
        while(SDL_PollEvent(&eventos))
        {
            get().on_evento(&eventos);
        }
        //actualizamos si inicia o hay una diferencia de tiempo
        get().on_fisicaupdate(dt);
        get().on_frameupdate(dt);
        //calculamos el tiempo del frame
        dt = (Tiempo::get_tiempo() - inicio)/frecuencia*1000;

        inicio=dt; //el nuevo frame inicia en este tiempo
        //printf("<%d>[%lf][%d]\n",get().fps,dt,(int)Tiempo::get_tiempo());
        if(dt<get().msfrecuencia)
        {
            SDL_Delay(floor(get().msfrecuencia-dt));
        }
        //get().fps++;
    }
    //liberamos memoria
    get().on_limpiar();
    return 0;
};
```

Muchas gracias