Añadir codigo

fichero.h

```
15
16
17      struct Branch {
18          enum EntityState const next_state;
19          Branch(EntityState next) : next_state(next) {}
20      };
21
```

```
142
143     private:
144         void CameraMovement(float dt);
145         void GodMode(float dt);
146         void PlayerMovement(float dt);
147         void FishingDirecction(bool verticalMovement, bool horizontalMovement);
148         void MaskAttack(float dt);
149         void TakeDamage(float damage);
150         void stateMachine(float dt);
151
```

```
public:

    Branch transitionTable[static_cast<int>(EntityState::STATE_COUNT)][static_cast<int>(EntityState::STATE_COUNT)] = {

    //      IDLE                    RUNNING                 ATTACKING                   DEAD                    REVIVING                MASK_ATTACK                 DASHI                   NONE
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::NONE}, {EntityState::IDLE}}, // IDLE
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // RUNNING
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // ATTACKING
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // DEAD
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // REVIVING
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // MASK_ATTACK
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // DASHI
    { {EntityState::IDLE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::IDLE}} // NONE
    };

    EntityState currentState = state;
    EntityState desiredState = nextState;
    EntityState nextState = transitionTable[static_cast<int>(currentState)][static_cast<int>(desiredState)].next_state;
};
```

fichero.cpp

```cpp
bool Player::Update(float dt)
{


    b2Transform pbodyPos = pbodyFoot->body->GetTransform();
    pbodySensor->body->SetTransform(b2Vec2(pbodyPos.p.x, pbodyPos.p.y - 1), 0);

    if (!inAnimation) {
        desiredState = EntityState::IDLE;
    }

    if (app->input->GetKey(SDL_SCANCODE_F10) == KEY_DOWN)
    {
        godmode = !godmode;
        pbodyFoot->body->GetFixtureList()[0].SetSensor(godmode);
    }

    if (currentAnimation->HasFinished() && currentAnimation->getNameAnimation() == "dashiAnim") {
        inAnimation = false;
        dashiAnim.Reset();
    }

    if (godmode) { GodMode(dt); }
    else PlayerMovement(dt);

    CameraMovement(dt);

    //printf("\nposx:%d, posy: %d",position.x, position.y);

    stateMachine(dt);

    currentAnimation->Update();
    return true;
}
```

```cpp
void Player::stateMachine(float dt)
{
    //printf("\ncurrentState: %d, desiredState: %d", static_cast<int>(currentState), static_cast<int>(desiredState));
    nextState = transitionTable[static_cast<int>(currentState)][static_cast<int>(desiredState)].next_state;
    switch (nextState) {
    case EntityState::IDLE:
        DoNothing(dt);
        app->audio->StopFx(-1);
        break;
    case EntityState::RUNNING:

        Run(dt);
        app->audio->PlayFx(run_fx); // <--- Hay que arreglarlo
        break;
    case EntityState::ATTACKING:
        Attack(dt);
        break;
    case EntityState::DEAD:

        break;
    case EntityState::REVIVING:
        break;
    case EntityState::MASK_ATTACK:
        MaskAttack(dt);
        break;
    case EntityState::DASHI:
        if (isDashing) {
            Dashi(dt);
        }
        break;
    case EntityState::NONE:

        desiredState = EntityState::IDLE;
        break;

    default:
        break;
    }
    currentState = nextState;
}
```

Para cambiar a new state:
desiredState = Nuevo State que quieres

```
//Si pulsas J para atacar
if (app->input->GetKey(SDL_SCANCODE_J) == KEY_DOWN && timerAttack.ReadMSec() > cdTimerAttackMS + 100) {
    isAttacking = true;
    timerAttack.Start();
    desiredState = EntityState::ATTACKING;
}
```

Teoria:

Orden de tablero tiene que ser mismo como EntityState!!!
Cada vez añadir un nuevo state tiene que añadir todos tablero que tiene.
Orden de tablero de fichero.h es como siguiente.

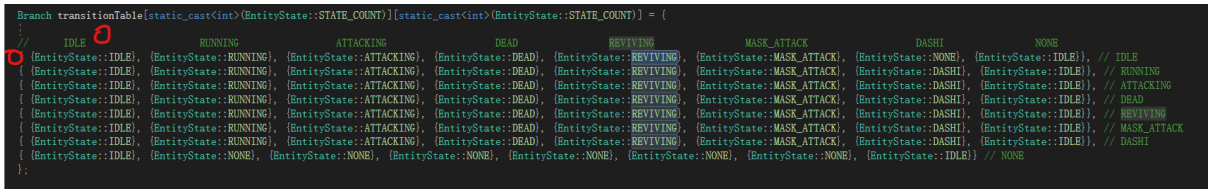
```
enum class EntityState
{
    IDLE,
    RUNNING,
    ATTACKING,
    DEAD,
    REVIVING,
    MASK_ATTACK,
    DASHI,
    NONE,
    STATE_COUNT
};
```

DLE = 0
RUNNING= 1
ATTACKING= 2
DEAD= 3
REVIVING= 4
MASK_ATTACK= 5
DASHI = 6
NONE = 7
STATE_COUNT= Siempre tiene que ser el ultimo!!!(no entra state, solo es para contar)

Como funciona statemachine?
Simplemente es buscar tablero

currentState = state actual
desiredState = siguiente state

si estamos idle, estamos state 0,0
currentState = 0
desiredState = 0



Si ahora queremos ir RUNNING

currentState = 0
desiredState = 0

Pulsar teclado de mover

```
if (horizontalMovement != 0 || verticalMovement != 0) {
    if (!inAnimation) {
        desiredState = EntityState::RUNNING;
    }
    isFacingLeft = (horizontalMovement < 0);
    lastMovementDirection = fPoint(horizontalMovement, verticalMovement);
}
```

desiredState = RUNNING
RUNNING= 1
Portanto ahora estamos como siguiente
currentState = 0
desiredState = 1

```
Branch transitionTable[static_cast<int>(EntityState::STATE_COUNT)][static_cast<int>(EntityState::STATE_COUNT)] = {
    //      IDLE                    RUNNING                 ATTACKING               DEAD                    REVIVING                MASK_ATTACK             DASHI                   NONE
0   { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::NONE}, {EntityState::IDLE}}, // IDLE
1   { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // RUNNING
2   { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // ATTACKING
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // DEAD
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // REVIVING
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // MASK_ATTACK
    { {EntityState::IDLE}, {EntityState::RUNNING}, {EntityState::ATTACKING}, {EntityState::DEAD}, {EntityState::REVIVING}, {EntityState::MASK_ATTACK}, {EntityState::DASHI}, {EntityState::IDLE}}, // DASHI
    { {EntityState::IDLE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::NONE}, {EntityState::IDLE}} // NONE
};
```

Pues es nextstate = RUNNING y entra switch de RUNNING

```
//printf("\ncurrentState: %d, desiredState: %d", static_cast<int>(currentState), static_cast<int>(desiredState));
nextState = transitionTable[static_cast<int>(currentState)][static_cast<int>(desiredState)].next_state;
switch (nextState) {
case EntityState::IDLE:
    DoNothing(dt);
    app->audio->StopFx(-1);
    break;
case EntityState::RUNNING:

    Run(dt);
    app->audio->PlayFx(run_fx); // <--- Hay que arreglarlo
    break;
case EntityState::ATTACKING:
    Attack(dt);
    break;
case EntityState::DEAD:
```