// RotateAnimator : 负责获取旋转展示过程中各种族位置

class RotateAnimator

{

A3DVECTOR3 m\_center; // 旋转中心位置

float m\_radius; // 目标与 m\_center 距离

int m\_count; // 围绕中心的目标个数

float m\_deltaDist; // 同一种族位置不同族内职业相对种族中心位置间距

float m\_selectMove; // 选中职业向相机移动距离（相对未选中时位置）

A3DVECTOR3 m\_dir; // 旋转角度为0时对应的轴向

A3DVECTOR3 m\_dirTangent; // m\_dir 的切向方向，用于计算位置偏移，达到一个位置附近获取多个位置的能力

double m\_angleStart; // 转向目标的开始位置

double m\_angle; // 当前角度

double m\_angleTarget; // 目标角度

DWORD m\_timeCost; // 转往目标花费总时间

DWORD m\_timeElapsed; // 转往目标位置已花费的时间（从m\_angleStart位置更新时算起）

bool m\_rotating; // 是否正在旋转中

int m\_firstIndicator; // 未旋转时转盘箭头指向的位置下标

int m\_targetIndicator; // 转盘箭头最终将指向的位置下标

enum RotateDirection

{

ROTATE\_CLOCKWISE, // 顺时针旋转

ROTATE\_ANTICLOCKWISE, // 逆时针旋转

};

RotateDirection m\_rotateDir;

RotateAnimator(int count);

RotateAnimator(const RotateAnimator &);

RotateAnimator & operator=(const RotateAnimator &);

public:

static const float RA\_DEFAULT\_RADIUS;

static const float RA\_DEFAULT\_DELTA\_DIST;

static const float RA\_DEFAULT\_SELECT\_MOVE;

static RotateAnimator & Instance();

double GetAngleBetween()const;

void SetCenter(const A3DVECTOR3 &center);

void SetDir(const A3DVECTOR3 &dir);

const A3DVECTOR3 & GetDir()const;

void SetRadius(float radius);

void SetDeltaDist(float dist);

void SetSelectMove(float dist);

float GetSelectMove()const;

void SetFirstIndicator(int indicator);

double GetBestAngleRepresentation(double angle);

void RotateToIndicator(int indicator);

bool Tick(DWORD dwDeltaTime);

void RotateFinished();

bool IsRotating()const{ return m\_rotating; }

A3DVECTOR3 GetCenteredIndicatorPos(int indicator);

A3DVECTOR3 GetIndicatorPos(int indicator);

A3DVECTOR3 GetIndicatorDeltaPos(int deltaIndex, float fMoveFrontRatio);

};

const float RotateAnimator::RA\_DEFAULT\_RADIUS = 2.0f;

const float RotateAnimator::RA\_DEFAULT\_DELTA\_DIST = 0.4f;

const float RotateAnimator::RA\_DEFAULT\_SELECT\_MOVE = 1.6f;

RotateAnimator & RotateAnimator::Instance()

{

static RotateAnimator s\_dummy(NUM\_RACE);

return s\_dummy;

}

RotateAnimator::RotateAnimator(int count)

: m\_center(0.0f), m\_radius(RA\_DEFAULT\_RADIUS), m\_count(count)

, m\_deltaDist(RA\_DEFAULT\_DELTA\_DIST), m\_selectMove(RA\_DEFAULT\_SELECT\_MOVE)

, m\_dir(0.0f, 0.0f, -1.0f), m\_dirTangent(-1.0f, 0.0f, 0.0f)

, m\_angleStart(0.0f), m\_angle(0.0f), m\_angleTarget(0.0f)

, m\_timeElapsed(0), m\_timeCost(1000), m\_rotating(false)

, m\_firstIndicator(0), m\_targetIndicator(0)

, m\_rotateDir(ROTATE\_ANTICLOCKWISE)

{}

double RotateAnimator::GetAngleBetween()const

{

return A3D\_2PI / m\_count;

}

void RotateAnimator::SetCenter(const A3DVECTOR3 &center)

{

m\_center = center;

}

void RotateAnimator::SetDir(const A3DVECTOR3 &dir)

{

m\_dir = dir;

m\_dir.y = 0;

m\_dir.Normalize();

m\_dirTangent = a3d\_RotatePosAroundY(m\_dir, A3D\_PI/2);

}

const A3DVECTOR3 & RotateAnimator::GetDir()const

{

return m\_dir;

}

void RotateAnimator::SetRadius(float radius)

{

m\_radius = radius;

}

void RotateAnimator::SetDeltaDist(float dist)

{

m\_deltaDist = dist;

}

void RotateAnimator::SetSelectMove(float dist)

{

m\_selectMove = dist;

}

float RotateAnimator::GetSelectMove()const

{

return m\_selectMove;

}

void RotateAnimator::SetFirstIndicator(int indicator)

{

if (indicator < 0 || indicator >= m\_count){

assert(false);

}else{

m\_firstIndicator = indicator;

m\_targetIndicator = indicator;

m\_angleStart = m\_angle = m\_angleTarget = 0.0f;

m\_timeElapsed = 0;

m\_rotating = false;

m\_rotateDir = ROTATE\_ANTICLOCKWISE;

RotateFinished();

}

}

double RotateAnimator::GetBestAngleRepresentation(double angle)

{

// 将 angle 转化到 [0, A3D\_2PI] 区间

angle = fmod(angle, A3D\_2PI);

if (angle < 0){

angle += A3D\_2PI;

}

return angle;

}

void RotateAnimator::RotateToIndicator(int indicator)

{

if (indicator == m\_targetIndicator){

return;

}

int lastIndicator = m\_targetIndicator;

m\_targetIndicator = indicator;

double newRotateAngle = -(indicator - lastIndicator) \* GetAngleBetween(); // 由于 indicator 按顺时针朝向摆放，所以取负值

double newTargetAngle = GetBestAngleRepresentation(m\_angleTarget + newRotateAngle);

double curAngle = GetBestAngleRepresentation(m\_angle);

double angleDiff = newTargetAngle - curAngle;

m\_angleTarget = newTargetAngle;

if (GetBestAngleRepresentation(angleDiff) > A3D\_PI){

// 如果顺时针，需要大半个圈，因此需用减少的方式

m\_rotateDir = ROTATE\_ANTICLOCKWISE;

m\_angle = curAngle;

if (m\_angle < newTargetAngle){

m\_angle += A3D\_2PI; // 保证 m\_angle 大于 m\_angleTarget 并逐渐减少

}

}else{

m\_rotateDir = ROTATE\_CLOCKWISE;

m\_angle = curAngle;

if (m\_angle > newTargetAngle){

m\_angle -= A3D\_2PI; // 保证 m\_angle 小于 m\_angleTarget 并逐渐增加

}

}

m\_angleStart = m\_angle;

m\_timeCost = 3000 \* fabs(m\_angle - m\_angleTarget) / A3D\_2PI;

if (m\_timeCost < 50){

m\_timeCost = 50;

}

m\_timeElapsed = 0;

m\_rotating = true;

}

bool RotateAnimator::Tick(DWORD dwDeltaTime)

{

if (!IsRotating()){

return false;

}

m\_timeElapsed += dwDeltaTime;

float t = m\_timeElapsed / (float)m\_timeCost;

a\_ClampRoof(t, 1.0f);

t = Ease::ExponentialOut(t);

m\_angle = m\_angleStart + t \* (m\_angleTarget-m\_angleStart);

if (m\_timeElapsed >= m\_timeCost){

m\_angle = m\_angleTarget;

m\_rotating = false;

RotateFinished();

}

return true;

}

void RotateAnimator::RotateFinished()

{

}

A3DVECTOR3 RotateAnimator::GetCenteredIndicatorPos(int indicator)

{

// 与 SetFirstIndicator 配合，保证不论如何旋转，同一参数 indicator 对应同一位置对象

if (indicator < 0 || indicator >= m\_count){

assert(false);

return A3DVECTOR3(0.0f);

}

int realIndex = (indicator - m\_firstIndicator + m\_count) % m\_count;

A3DVECTOR3 vZeroPos = m\_dir \* m\_radius;

return a3d\_RotatePosAroundY(vZeroPos, (float)(m\_angle + realIndex \* GetAngleBetween())); // 此函数决定了左手坐标系，indicator 顺时针旋转依次显示

}

A3DVECTOR3 RotateAnimator::GetIndicatorPos(int indicator)

{

if (indicator < 0 || indicator >= m\_count){

assert(false);

return A3DVECTOR3(0.0f);

}

return m\_center + GetCenteredIndicatorPos(indicator);

}

A3DVECTOR3 RotateAnimator::GetIndicatorDeltaPos(int deltaIndex, float fMoveFrontRatio)

{

A3DVECTOR3 delta = m\_dirTangent \* m\_deltaDist; // 距离 indicator 位置偏移

if (deltaIndex == 1){

delta = -delta;

}

delta += m\_dir \* m\_selectMove \* fMoveFrontRatio;

return delta;

}

// 倒计时一次性 Trigger

class ActionTrigger

{

bool m\_bTriggered;

DWORD m\_triggerTime;

DWORD m\_triggerLength;

public:

ActionTrigger();

void Reset();

void StartCounting(DWORD dwTime);

void ReverseCounting(DWORD dwTime);

void Tick(DWORD dwDeltaTime);

bool GetTriggered();

float GetLeftPercent()const;

};

ActionTrigger::ActionTrigger()

{

Reset();

}

void ActionTrigger::Reset()

{

m\_bTriggered = false;

m\_triggerTime = 0;

m\_triggerLength = 0;

}

void ActionTrigger::StartCounting(DWORD dwTime)

{

m\_bTriggered = false;

m\_triggerTime = dwTime;

m\_triggerLength = dwTime;

}

void ActionTrigger::ReverseCounting(DWORD dwTime)

{

if (m\_triggerLength > 0){

m\_triggerTime = m\_triggerLength - m\_triggerTime;

float ratio = m\_triggerTime/(float)m\_triggerLength;

m\_triggerLength = dwTime;

m\_triggerTime = (DWORD)(int)(m\_triggerLength \* ratio);

}

}

void ActionTrigger::Tick(DWORD dwDeltaTime)

{

if (m\_bTriggered || m\_triggerTime == 0){

return;

}

if (m\_triggerTime <= dwDeltaTime){

m\_bTriggered = true;

m\_triggerTime = 0;

m\_triggerLength = 0;

return;

}

m\_triggerTime -= dwDeltaTime;

}

bool ActionTrigger::GetTriggered()

{

if (m\_bTriggered){

Reset();

return true;

}

return false;

}

float ActionTrigger::GetLeftPercent()const

{

return (m\_triggerLength == 0) ? 0.0f : (float)m\_triggerTime/(float)m\_triggerLength;

}

class ProfessionTrigger;

// 职业行为控制器用到的状态

class ProfessionTriggerState

{

ProfessionTrigger\* m\_parent;

public:

virtual ~ProfessionTriggerState(){}

ProfessionTriggerState() : m\_parent(NULL){

}

void Init(ProfessionTrigger \*parent){

m\_parent = parent;

}

ProfessionTrigger \* GetParent(){

return m\_parent;

}

virtual void TriggerMoveFront(){}

virtual void TriggerGoback(){}

virtual void Enter(ProfessionTriggerState \*prev){}

virtual void Tick(DWORD dwDeltaTime){}

virtual void Rest(){}

virtual float GetBackToFrontRatio(){ return 0.0f; }

};

static const DWORD WAIT\_MOVEFRONT\_TIME = 100;

static const DWORD WAIT\_DISPLAY\_TIME = 300;

static const float PLAYER\_WALK\_SPEED = 2.0f; // 角色走路的速度、用于计算移动时间

static DWORD GetMoveFrontTimeCost()

{

// 从默认位置走到最前端所需时间

float s = RotateAnimator::Instance().GetSelectMove() / PLAYER\_WALK\_SPEED;

DWORD dwResult = (DWORD)(s\*1000);

return max(dwResult, 100);

}

static DWORD GetGobackTimeCost()

{

// 从最前端走回默认位置所需时间

return GetMoveFrontTimeCost();

}

class PTStateBack : public ProfessionTriggerState

{

// 默认状态：站在默认位置静默

public:

virtual void TriggerMoveFront();

virtual void Enter(ProfessionTriggerState \*prev);

};

void PTStateBack::TriggerMoveFront()

{

GetParent()->ChangeState(GetParent()->GetStateWaitRotator());

}

void PTStateBack::Enter(ProfessionTriggerState \*prev)

{

GetParent()->GetTrigger().Reset();

GetParent()->TriggerStandAction();

}

class PTStateWaitRotator : public ProfessionTriggerState

{

// 默认位置、已选中，等待旋转完成开启后续状态

public:

virtual void TriggerGoback();

virtual void Enter(ProfessionTriggerState \*prev);

virtual void Tick(DWORD dwDeltaTime);

virtual void Rest();

};

void PTStateWaitRotator::TriggerGoback()

{

GetParent()->ChangeState(GetParent()->GetStateBack());

}

void PTStateWaitRotator::Enter(ProfessionTriggerState \*prev)

{

GetParent()->GetTrigger().Reset();

GetParent()->TriggerStandAction();

}

void PTStateWaitRotator::Tick(DWORD dwDeltaTime)

{

if (!RotateAnimator::Instance().IsRotating()){

GetParent()->ChangeState(GetParent()->GetStateWaitMoveFront());

}

}

void PTStateWaitRotator::Rest()

{

GetParent()->ChangeState(GetParent()->GetStateWaitDisplay());

}

class PTStateWaitMoveFront : public ProfessionTriggerState

{

// 等待开始移动

public:

virtual void TriggerGoback();

virtual void Enter(ProfessionTriggerState \*prev);

virtual void Tick(DWORD dwDeltaTime);

virtual void Rest();

};

void PTStateWaitMoveFront::TriggerGoback()

{

GetParent()->ChangeState(GetParent()->GetStateBack());

}

void PTStateWaitMoveFront::Enter(ProfessionTriggerState \*prev)

{

GetParent()->GetTrigger().StartCounting(WAIT\_MOVEFRONT\_TIME);

}

void PTStateWaitMoveFront::Tick(DWORD dwDeltaTime)

{

GetParent()->GetTrigger().Tick(dwDeltaTime);

if (GetParent()->GetTrigger().GetTriggered()){

GetParent()->ChangeState(GetParent()->GetStateMoveFront());

}

}

void PTStateWaitMoveFront::Rest()

{

GetParent()->ChangeState(GetParent()->GetStateWaitDisplay());

}

class PTStateMoveFront : public ProfessionTriggerState

{

// 走近移动中

public:

virtual void TriggerGoback();

virtual void Enter(ProfessionTriggerState \*prev);

virtual void Tick(DWORD dwDeltaTime);

virtual void Rest();

virtual float GetBackToFrontRatio();

};

void PTStateMoveFront::TriggerGoback()

{

GetParent()->ChangeState(GetParent()->GetStateGoback());

}

void PTStateMoveFront::Enter(ProfessionTriggerState \*prev)

{

GetParent()->GetTrigger().StartCounting(GetMoveFrontTimeCost());

GetParent()->TriggerMoveFrontAction();

}

void PTStateMoveFront::Tick(DWORD dwDeltaTime)

{

GetParent()->GetTrigger().Tick(dwDeltaTime);

if (GetParent()->GetTrigger().GetTriggered()){

GetParent()->ChangeState(GetParent()->GetStateWaitDisplay());

}

}

void PTStateMoveFront::Rest()

{

GetParent()->ChangeState(GetParent()->GetStateWaitDisplay());

}

float PTStateMoveFront::GetBackToFrontRatio()

{

return 1.0f - GetParent()->GetTrigger().GetLeftPercent();

}

class PTStateWaitDisplay : public ProfessionTriggerState

{

// 等待播放展示动作

public:

virtual void TriggerGoback();

virtual void Enter(ProfessionTriggerState \*prev);

virtual void Tick(DWORD dwDeltaTime);

virtual float GetBackToFrontRatio(){ return 1.0f; }

};

void PTStateWaitDisplay::TriggerGoback()

{

GetParent()->ChangeState(GetParent()->GetStateGoback());

}

void PTStateWaitDisplay::Enter(ProfessionTriggerState \*prev)

{

GetParent()->GetTrigger().StartCounting(WAIT\_DISPLAY\_TIME);

GetParent()->TriggerStandAction();

}

void PTStateWaitDisplay::Tick(DWORD dwDeltaTime)

{

GetParent()->GetTrigger().Tick(dwDeltaTime);

if (GetParent()->GetTrigger().GetTriggered()){

GetParent()->ChangeState(GetParent()->GetStateDisplay());

}

}

class PTStateDisplay : public ProfessionTriggerState

{

// 播放展示动作中...

public:

virtual void TriggerGoback();

virtual void Enter(ProfessionTriggerState \*prev);

virtual void Rest();

virtual float GetBackToFrontRatio(){ return 1.0f; }

};

void PTStateDisplay::TriggerGoback()

{

GetParent()->ChangeState(GetParent()->GetStateGoback());

}

void PTStateDisplay::Enter(ProfessionTriggerState \*prev)

{

GetParent()->TriggerDisplayAction();

}

void PTStateDisplay::Rest()

{

GetParent()->ChangeState(GetParent()->GetStateWaitDisplay());

}

class PTStateGoback : public ProfessionTriggerState

{

// 走回默认位置移动中...

public:

virtual void TriggerMoveFront();

virtual void Enter(ProfessionTriggerState \*prev);

virtual void Tick(DWORD dwDeltaTime);

virtual void Rest();

virtual float GetBackToFrontRatio();

};

void PTStateGoback::TriggerMoveFront()

{

GetParent()->ChangeState(GetParent()->GetStateWaitRotator());

}

void PTStateGoback::Enter(ProfessionTriggerState \*prev)

{

if (prev == GetParent()->GetStateMoveFront()){

GetParent()->GetTrigger().ReverseCounting(GetGobackTimeCost());

}else{

GetParent()->GetTrigger().StartCounting(GetGobackTimeCost());

}

GetParent()->TriggerGobackAction();

}

void PTStateGoback::Tick(DWORD dwDeltaTime)

{

GetParent()->GetTrigger().Tick(dwDeltaTime);

if (GetParent()->GetTrigger().GetTriggered()){

GetParent()->ChangeState(GetParent()->GetStateBack());

}

}

void PTStateGoback::Rest()

{

GetParent()->ChangeState(GetParent()->GetStateBack());

}

float PTStateGoback::GetBackToFrontRatio()

{

return GetParent()->GetTrigger().GetLeftPercent();

}

class ProfessionTrigger

{

ProfessionTriggerState\* m\_curState;

PTStateBack m\_stateBack;

PTStateWaitRotator m\_stateWaitRotator;

PTStateWaitMoveFront m\_stateWaitMoveFront;

PTStateMoveFront m\_stateMoveFront;

PTStateWaitDisplay m\_stateWaitDisplay;

PTStateDisplay m\_stateDisplay;

PTStateGoback m\_stateGoback;

// 各状态共用的对象

ActionTrigger m\_counter;

bool m\_moveFrontActionTriggered;

bool m\_gobackActionTriggered;

bool m\_displayActionTriggered;

bool m\_standActionTriggered;

public:

ProfessionTrigger();

void Reset();

// 转发给各 State 的方法

void TriggerMoveFront();

void TriggerGoback();

void Tick(DWORD dwDeltaTime);

void Rest();

float GetBackToFrontRatio();

// 被各 State 对象使用的方法

void ChangeState(ProfessionTriggerState \*state);

ProfessionTriggerState \* GetStateBack(){ return &m\_stateBack; }

ProfessionTriggerState \* GetStateWaitRotator(){ return &m\_stateWaitRotator; }

ProfessionTriggerState \* GetStateWaitMoveFront(){ return &m\_stateWaitMoveFront; }

ProfessionTriggerState \* GetStateMoveFront(){ return &m\_stateMoveFront; }

ProfessionTriggerState \* GetStateWaitDisplay(){ return &m\_stateWaitDisplay; }

ProfessionTriggerState \* GetStateDisplay(){ return &m\_stateDisplay; }

ProfessionTriggerState \* GetStateGoback(){ return &m\_stateGoback; }

ActionTrigger & GetTrigger(){ return m\_counter; }

void TriggerMoveFrontAction();

void TriggerGobackAction();

void TriggerDisplayAction();

void TriggerStandAction();

void ClearTriggerAction();

// 外界查询方法

bool MoveFrontActionTriggered();

bool GobackActionTriggered();

bool DisplayActionTriggered();

bool StandActionTriggered();

bool IsGoingback();

};

ProfessionTrigger::ProfessionTrigger()

{

GetStateBack()->Init(this);

GetStateWaitRotator()->Init(this);

GetStateWaitMoveFront()->Init(this);

GetStateMoveFront()->Init(this);

GetStateWaitDisplay()->Init(this);

GetStateDisplay()->Init(this);

GetStateGoback()->Init(this);

Reset();

}

void ProfessionTrigger::TriggerMoveFrontAction()

{

m\_moveFrontActionTriggered = true;

m\_gobackActionTriggered = false;

m\_displayActionTriggered = false;

m\_standActionTriggered = false;

}

void ProfessionTrigger::TriggerGobackAction()

{

m\_moveFrontActionTriggered = false;

m\_gobackActionTriggered = true;

m\_displayActionTriggered = false;

m\_standActionTriggered = false;

}

void ProfessionTrigger::TriggerDisplayAction()

{

m\_moveFrontActionTriggered = false;

m\_gobackActionTriggered = false;

m\_displayActionTriggered = true;

m\_standActionTriggered = false;

}

void ProfessionTrigger::TriggerStandAction()

{

m\_moveFrontActionTriggered = false;

m\_gobackActionTriggered = false;

m\_displayActionTriggered = false;

m\_standActionTriggered = true;

}

void ProfessionTrigger::ClearTriggerAction()

{

m\_moveFrontActionTriggered = false;

m\_gobackActionTriggered = false;

m\_displayActionTriggered = false;

m\_standActionTriggered = false;

}

bool ProfessionTrigger::MoveFrontActionTriggered()

{

if (m\_moveFrontActionTriggered){

m\_moveFrontActionTriggered = false;

return true;

}

return false;

}

bool ProfessionTrigger::GobackActionTriggered()

{

if (m\_gobackActionTriggered){

m\_gobackActionTriggered = false;

return true;

}

return false;

}

bool ProfessionTrigger::DisplayActionTriggered()

{

if (m\_displayActionTriggered){

m\_displayActionTriggered = false;

return true;

}

return false;

}

bool ProfessionTrigger::StandActionTriggered()

{

if (m\_standActionTriggered){

m\_standActionTriggered = false;

return true;

}

return false;

}

bool ProfessionTrigger::IsGoingback()

{

return m\_curState == GetStateGoback();

}

void ProfessionTrigger::Reset()

{

ClearTriggerAction();

GetTrigger().Reset();

ChangeState(GetStateBack());

}

void ProfessionTrigger::TriggerMoveFront()

{

m\_curState->TriggerMoveFront();

}

void ProfessionTrigger::TriggerGoback()

{

m\_curState->TriggerGoback();

}

void ProfessionTrigger::Tick(DWORD dwDeltaTime)

{

m\_curState->Tick(dwDeltaTime);

}

void ProfessionTrigger::Rest()

{

m\_curState->Rest();

}

float ProfessionTrigger::GetBackToFrontRatio()

{

return m\_curState->GetBackToFrontRatio();

}

void ProfessionTrigger::ChangeState(ProfessionTriggerState \*state)

{

if (state == m\_curState){

return;

}

ProfessionTriggerState \*prev = m\_curState;

m\_curState = state;

m\_curState->Enter(prev);

}

class DisplayActionTrigger

{

ProfessionTrigger m\_trigger[NUM\_PROFESSION];

DisplayActionTrigger();

DisplayActionTrigger(const DisplayActionTrigger &);

DisplayActionTrigger & operator=(const DisplayActionTrigger &);

bool Validate(int prof);

public:

static DisplayActionTrigger & Instance();

void Reset();

void Rest();

void Trigger(int prof);

void Tick(DWORD dwDeltaTime);

bool MoveFrontActionTriggered(int prof);

bool GoBackActionTriggered(int prof);

bool DisplayActionTriggered(int prof);

bool StandActionTriggered(int prof);

void GetPosDir(int prof, A3DVECTOR3& vPos, A3DVECTOR3& dir);

};

DisplayActionTrigger & DisplayActionTrigger::Instance()

{

static DisplayActionTrigger s\_dummy;

return s\_dummy;

}

DisplayActionTrigger::DisplayActionTrigger()

{

Reset();

}

void DisplayActionTrigger::Reset()

{

for (int i(0); i < ARRAY\_SIZE(m\_trigger); ++ i)

{

m\_trigger[i].Reset();

}

}

void DisplayActionTrigger::Rest()

{

// 从当前状态直接到达最重要的状态（站到默认位置、或播放演武动画）

for (int i(0); i < ARRAY\_SIZE(m\_trigger); ++ i)

{

m\_trigger[i].Rest();

}

}

bool DisplayActionTrigger::Validate(int prof)

{

if (!CECProfConfig::Instance().IsProfession(prof)){

assert(false);

return false;

}

return true;

}

void DisplayActionTrigger::Trigger(int prof)

{

if (Validate(prof)){

for (int i(0); i < ARRAY\_SIZE(m\_trigger); ++ i)

{

if (i == prof){

m\_trigger[i].TriggerMoveFront();

}else{

m\_trigger[i].TriggerGoback();

}

}

}

}

void DisplayActionTrigger::Tick(DWORD dwDeltaTime)

{

for (int i(0); i < ARRAY\_SIZE(m\_trigger); ++ i)

{

m\_trigger[i].Tick(dwDeltaTime);

}

}

bool DisplayActionTrigger::MoveFrontActionTriggered(int prof)

{

if (Validate(prof)){

return m\_trigger[prof].MoveFrontActionTriggered();

}

return false;

}

bool DisplayActionTrigger::GoBackActionTriggered(int prof)

{

if (Validate(prof)){

return m\_trigger[prof].GobackActionTriggered();

}

return false;

}

bool DisplayActionTrigger::DisplayActionTriggered(int prof)

{

if (Validate(prof)){

return m\_trigger[prof].DisplayActionTriggered();

}

return false;

}

bool DisplayActionTrigger::StandActionTriggered(int prof)

{

if (Validate(prof)){

return m\_trigger[prof].StandActionTriggered();

}

return false;

}

void DisplayActionTrigger::GetPosDir(int prof, A3DVECTOR3& vPos, A3DVECTOR3& dir)

{

if (Validate(prof)){

int race = CECProfConfig::Instance().GetRaceByProfession(prof);

int showOrder = CECProfConfig::Instance().GetRaceShowOrder(race);

A3DVECTOR3 racePos = RotateAnimator::Instance().GetIndicatorPos(showOrder);

int showOrderInRace = CECProfConfig::Instance().GetProfessionShowOrderInRace(prof);

vPos = racePos + RotateAnimator::Instance().GetIndicatorDeltaPos(showOrderInRace, m\_trigger[prof].GetBackToFrontRatio());

if (m\_trigger[prof].IsGoingback()){

dir = -RotateAnimator::Instance().GetDir();

}else{

dir = RotateAnimator::Instance().GetDir();

}

}else{

vPos.Clear();

dir = g\_vAxisY;

}

}

class CECProfConfig

{

private:

CECProfConfig();

CECProfConfig(const CECProfConfig &);

CECProfConfig & operator == (const CECProfConfig &);

public:

static CECProfConfig & Instance();

bool IsProfession(int prof)const;

bool IsGender(int gender)const;

bool IsRace(int race)const;

bool IsExist(int prof, int gender)const;

bool CanShowOnCreate(int prof, int gender)const;

int GetRaceShowOrder(int race)const;

int GetRaceByProfession(int prof)const;

int GetProfessionShowOrderInRace(int prof)const;

bool ContainsAllProfession(unsigned int mask)const;

unsigned int GetAllProfessionMask()const;

int GetMaxBodyID(int prof)const;

};

CECProfConfig::CECProfConfig()

{

}

CECProfConfig & CECProfConfig::Instance()

{

static CECProfConfig s\_dummy;

return s\_dummy;

}

bool CECProfConfig::IsProfession(int prof)const

{

return prof >= 0 && prof < NUM\_PROFESSION;

}

bool CECProfConfig::IsGender(int gender)const

{

return gender >= 0 && gender < NUM\_GENDER;

}

bool CECProfConfig::IsRace(int race)const

{

return race >= 0 && race < NUM\_RACE;

}

bool CECProfConfig::IsExist(int prof, int gender)const

{

if (IsProfession(prof) && IsGender(gender)){

static bool s\_bExist[NUM\_PROFESSION][NUM\_GENDER] = {

{ true, true}, // 0:武侠

{ true, true}, // 1:法师

{ true, true}, // 2:巫师

{ false, true}, // 3:妖精

{ true, false}, // 4:妖兽

{ true, true}, // 5:刺客

{ true, true}, // 6:羽芒

{ true, true}, // 7:羽灵

{ true, true}, // 8:剑灵

{ true, true}, // 9:魅灵

{ true, true}, // 10:夜影

{ true, true}, // 11:月仙

};

return s\_bExist[prof][gender];

}else{

assert(false);

return false;

}

}

bool CECProfConfig::CanShowOnCreate(int prof, int gender)const

{

// 创建角色时是否应该显示此职业模型（每职业显示一种性别）

if (IsProfession(prof) && IsGender(gender)){

static bool s\_bShowMale[NUM\_PROFESSION][NUM\_GENDER] = {

{true, false}, // 0:武侠

{false,true}, // 1:法师

{false,true}, // 2:巫师

{false, true}, // 3:妖精

{true, false}, // 4:妖兽

{true, false}, // 5:刺客

{true, false}, // 6:羽芒

{false, true}, // 7:羽灵

{true, false}, // 8:剑灵

{false,true}, // 9:魅灵

{true, false}, // 10:夜影

{false, true}, // 11:月仙

};

return s\_bShowMale[prof][gender];

}

assert(false);

return false;

}

int CECProfConfig::GetRaceShowOrder(int race)const

{

// 值越小优先级越大

if (IsRace(race)){

static int s\_nRaceOrder[NUM\_RACE] = {

1,

2,

3,

4,

5,

0,

};

return s\_nRaceOrder[race];

}

assert(false);

return -1;

}

int CECProfConfig::GetRaceByProfession(int prof)const

{

if (IsProfession(prof)){

static int s\_nProfRace[NUM\_PROFESSION] = {

RACE\_HUMAN,

RACE\_HUMAN,

RACE\_GHOST,

RACE\_ORC,

RACE\_ORC,

RACE\_GHOST,

RACE\_ELF,

RACE\_ELF,

RACE\_LING,

RACE\_LING,

RACE\_OBORO,

RACE\_OBORO,

};

return s\_nProfRace[prof];

}

assert(false);

return -1;

}

int CECProfConfig::GetProfessionShowOrderInRace(int prof)const

{

// 值越小优先级越大

if (IsProfession(prof)){

static int s\_nRaceZeroShowOrderProf[NUM\_RACE] = {

PROF\_WARRIOR, // 人类：武侠

PROF\_ORC, // 妖族：妖兽

PROF\_ANGEL, // 羽人：羽灵

PROF\_MONK, // 汐族：巫师

PROF\_JIANLING, // 灵族：剑灵

PROF\_YEYING, // 胧族：夜影

};

int race = GetRaceByProfession(prof);

return prof == s\_nRaceZeroShowOrderProf[race] ? 0 : 1;

}

assert(false);

return -1;

}

bool CECProfConfig::ContainsAllProfession(unsigned int mask)const{

return (GetAllProfessionMask() & mask) == GetAllProfessionMask();

}

unsigned int CECProfConfig::GetAllProfessionMask()const{

static const unsigned int ALL\_PROFESSION\_MASK = (1 << NUM\_PROFESSION)-1;

return ALL\_PROFESSION\_MASK;

}

int CECProfConfig::GetMaxBodyID(int prof)const{

if (IsProfession(prof)){

// PROF\_ANGEL, PROF\_ARCHOR, PROF\_MAGE, PROF\_WARRIOR didn't have nBodyID

static int s\_maxBodyID[NUM\_PROFESSION] = {-1, -1, 4, 5, 3, 4, -1, -1, 4, 4, 1, 1 };

return s\_maxBodyID[prof];

}

return -1;

}

void ChangeShowModel(int prof, int gender){

int race = CECProfConfig::Instance().GetRaceByProfession(prof);

int order = CECProfConfig::Instance().GetRaceShowOrder(race);

if (!CECProfConfig::Instance().IsProfession(lastProfession)){

// 设置显示到最前方的职业

RotateAnimator::Instance().SetFirstIndicator(order); // 保证 RotateAnimator::GetIndicatorPos(order) 会获取到内部0的位置，即对应最前的位置

}else{

RotateAnimator::Instance().RotateToIndicator(order);

}

DisplayActionTrigger::Instance().Trigger(prof);

RefreshPlayerList();

TickRaceModels(0);

}

}

void TickRaceModels(DWORD dwDeltaTime){

RotateAnimator::Instance().Tick(dwDeltaTime);

DisplayActionTrigger::Instance().Tick(dwDeltaTime);

for (int i = 0; i < NUM\_PROFESSION; ++ i){

bool moveFrontTriggered = DisplayActionTrigger::Instance().MoveFrontActionTriggered(i);

bool goBackTriggered = DisplayActionTrigger::Instance().GoBackActionTriggered(i);

bool displayActionTriggered = DisplayActionTrigger::Instance().DisplayActionTriggered(i);

bool standActionTriggered = DisplayActionTrigger::Instance().StandActionTriggered(i);

A3DVECTOR3 pos(0.0f), dir(g\_vAxisY);

DisplayActionTrigger::Instance().GetPosDir(i, pos, dir);

for (int j = 0; j < NUM\_GENDER; ++ j){

if (!CECProfConfig::Instance().CanShowOnCreate(i, j) ||

m\_aModel[i][j] == NULL){

continue;

}

CECLoginPlayer \*pPlayer = m\_aModel[i][j];

pPlayer->SetPos(pos);

pPlayer->ChangeModelMoveDirAndUp(dir, g\_vAxisY);

if (moveFrontTriggered){

pPlayer->MoveFront();

}else if (goBackTriggered){

pPlayer->GoBack();

}else if (displayActionTriggered){

pPlayer->ShowDisplayAction();

}else if (standActionTriggered){

pPlayer->Stand(true);

}

pPlayer->Tick(0);

}

}

}

namespace Ease

{

float ExponentialIn(float t);

float ExponentialOut(float t);

float ExponentialInOut(float t);

float SineIn(float t);

float SineOut(float t);

float SineInOut(float t);

}

#define M\_PI 3.14159265358979323846

#define M\_PI\_2 1.57079632679489661923

namespace Ease

{

// 以下加速度效果参考自 cocos2d-x

float ExponentialIn(float t)

{

return t == 0 ? 0 : powf(2, 10 \* (t - 1)) - 0.001f;

}

float ExponentialOut(float t)

{

return t == 1 ? 1 : (-powf(2, -10 \* t) + 1);

}

float ExponentialInOut(float t)

{

t /= 0.5f;

if (t < 1)

{

t = 0.5f \* powf(2, 10 \* (t - 1));

}

else

{

t = 0.5f \* (-powf(2, -10 \* (t - 1)) + 2);

}

return t;

}

float SineIn(float t)

{

return -1 \* cosf(t \* (float)M\_PI\_2) + 1;

}

float SineOut(float t)

{

return sinf(t \* (float)M\_PI\_2);

}

float SineInOut(float t)

{

return -0.5f \* (cosf((float)M\_PI \* t) - 1);

}

}