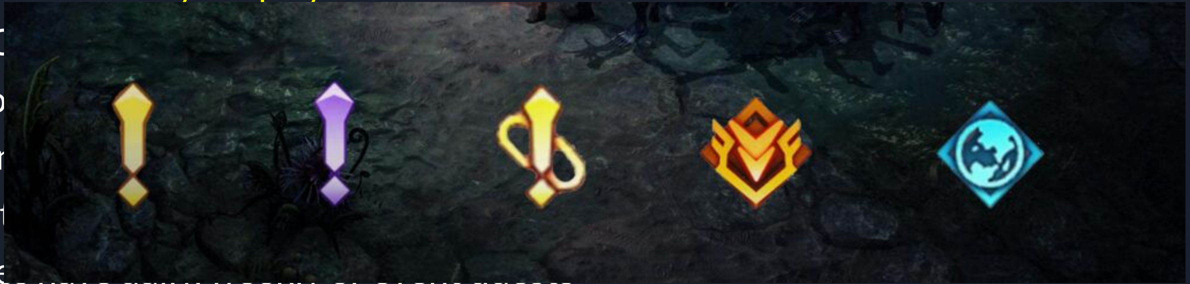
Quest Manager

Pol Maresch

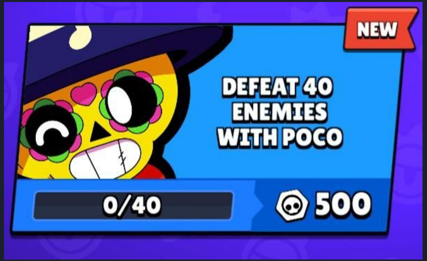
What is a Quest?

- Task that the player or players need to do for obtain something
  - This can give:
  - Objects/Equipment
  - Experiencie for your player LVL
  - For C
- Can be coo
- Can be Prir
- Sometime t
- Some game
- Normaly that quest are in Rol game or Multiplayer games

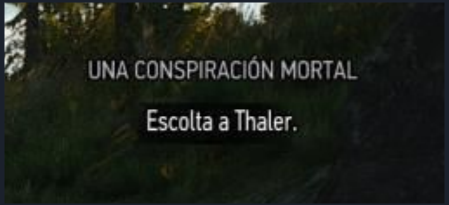


LostArk Quest

Type of Quest



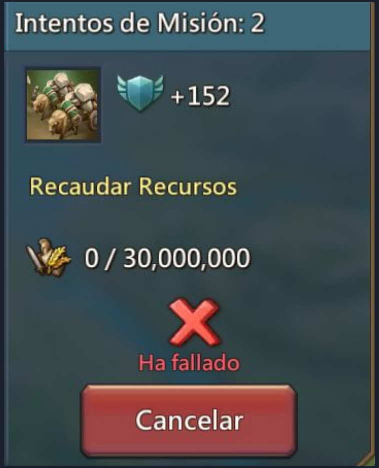
Kill Quest



UNA CONSPIRACIÓN MORTAL

Escolta a Thaler.

Protection Quest



Intentos de Misión: 2

+152

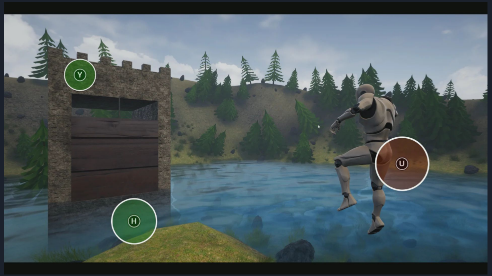
Recaudar Recursos

0 / 30,000,000

Ha fallado

Cancelar

Collect Quest

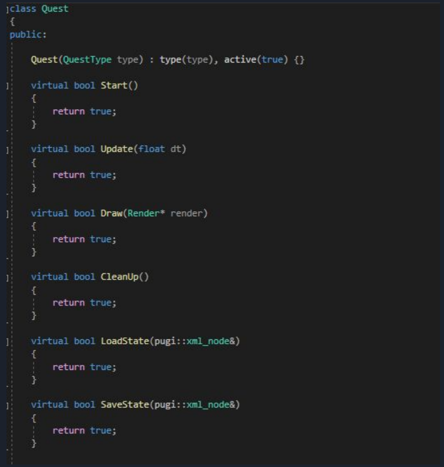


Combo Quest

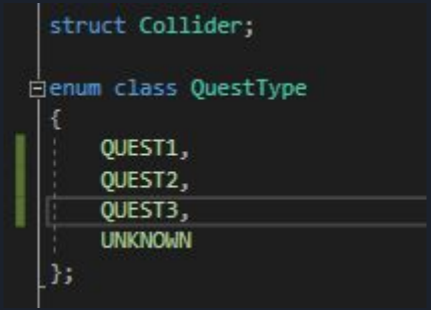
Quest Manager code

First of all we first need to create the Quests.h module, this module will be the parent of the other quest modules, since the quest modules will derive from it.

The module contains a class, with the missions that you are going to use, like the following one:



```cpp
class Quest
{
public:

    Quest(QuestType type) : type(type), active(true) {}

    virtual bool Start()
    {
        return true;
    }

    virtual bool Update(float dt)
    {
        return true;
    }

    virtual bool Draw(Render* render)
    {
        return true;
    }

    virtual bool CleanUp()
    {
        return true;
    }

    virtual bool LoadState(pugi::xml_node&)
    {
        return true;
    }

    virtual bool SaveState(pugi::xml_node&)
    {
        return true;
    }
```

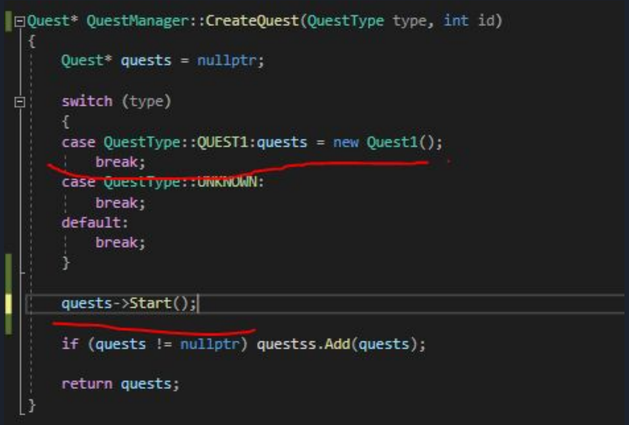


```cpp
struct Collider;

enum class QuestType
{
    QUEST1,
    QUEST2,
    QUEST3,
    UNKNOWN
};
```

Quest Manager code

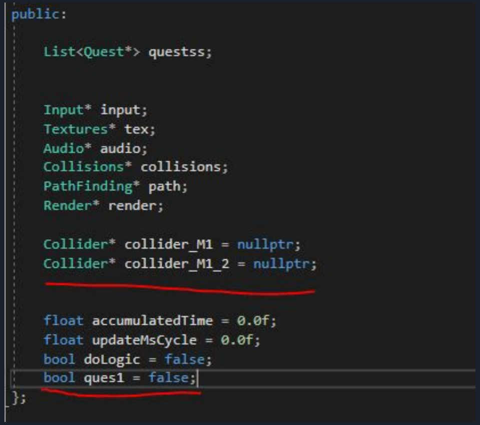
Then we are going to create Quest_manager.h and Quest_manager.cpp, these modules will create the entities in the scene and organize them in the game. This module will have everything needed to function to create or destroy the entities in the scenes.

First of all we are going to create the quests, on the function CreateQuest,its important that on this function, we need to call the quest Start, then we will see (first we need the quest modules created).

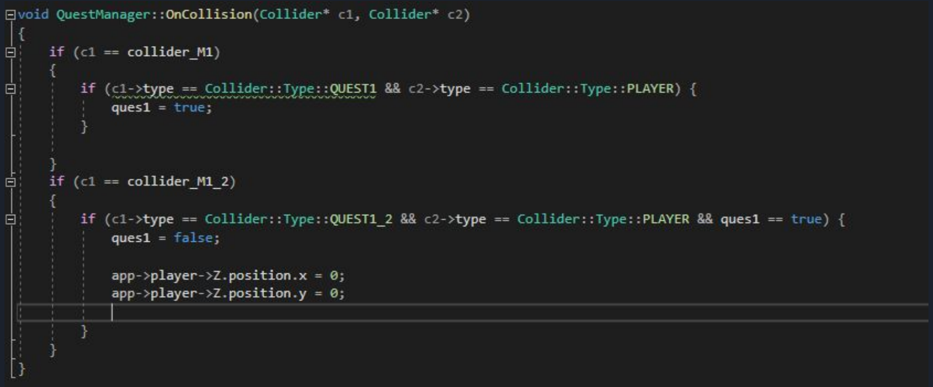

```
Quest* QuestManager::CreateQuest(QuestType type, int id)
{
    Quest* quests = nullptr;

    switch (type)
    {
    case QuestType::QUEST1:quests = new Quest1();
        break;
    case QuestType::UNKNOWN:
        break;
    default:
        break;
    }

    quests->Start();

    if (quests != nullptr) questss.Add(quests);

    return quests;
}
```

Quest Manager code

This module will have two important variables, the mission collisions and the bool that activates them, since they will be used in the Oncollision and then also called in the mission module.



```
public:

    List<Quest*> questss;


    Input* input;
    Textures* tex;
    Audio* audio;
    Collisions* collisions;
    PathFinding* path;
    Render* render;

    Collider* collider_M1 = nullptr;
    Collider* collider_M1_2 = nullptr;


    float accumulatedTime = 0.0f;
    float updateMsCycle = 0.0f;
    bool doLogic = false;
    bool ques1 = false;
};
```



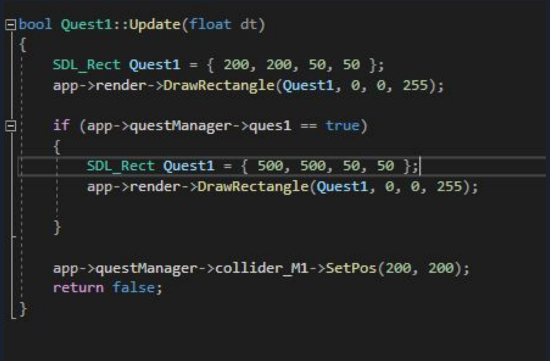
```
void QuestManager::OnCollision(Collider* c1, Collider* c2)
{
    if (c1 == collider_M1)
    {
        if (c1->type == Collider::Type::QUEST1 && c2->type == Collider::Type::PLAYER) {
            ques1 = true;
        }

    }
    if (c1 == collider_M1_2)
    {
        if (c1->type == Collider::Type::QUEST1_2 && c2->type == Collider::Type::PLAYER && ques1 == true) {
            ques1 = false;

            app->player->Z.position.x = 0;
            app->player->Z.position.y = 0;

        }
    }
}
```

The Quest

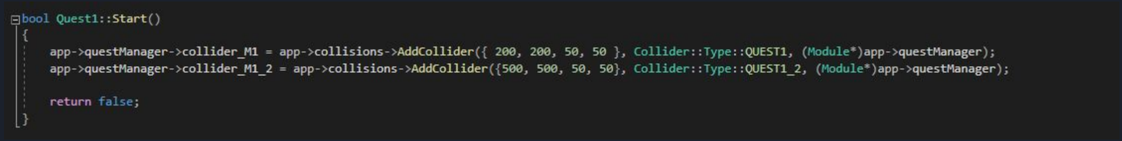
When we create the Quest, we are going to make the quest 1.cpp and quest1.h, here we are going to make the quest.

- this module need:
    - Start
    - Update

we are going to create the collisions in the start, and then in the update we activate or deactivate them



```
bool Quest1::Update(float dt)
{
    SDL_Rect Quest1 = { 200, 200, 50, 50 };
    app->render->DrawRectangle(Quest1, 0, 0, 255);

    if (app->questManager->ques1 == true)
    {
        SDL_Rect Quest1 = { 500, 500, 50, 50 };
        app->render->DrawRectangle(Quest1, 0, 0, 255);

    }

    app->questManager->collider_M1->SetPos(200, 200);
    return false;
}
```



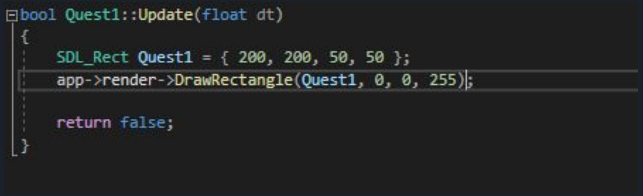
```
bool Quest1::Start()
{
    app->questManager->collider_M1 = app->collisions->AddCollider({ 200, 200, 50, 50 }, Collider::Type::QUEST1, (Module*)app->questManager);
    app->questManager->collider_M1_2 = app->collisions->AddCollider({500, 500, 50, 50}, Collider::Type::QUEST1_2, (Module*)app->questManager);

    return false;
}
```

Quest Manager code

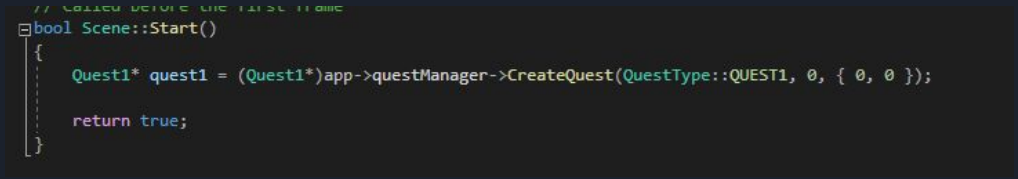
We need collider like a sensor, when player hits with the Quest collider, the quest start.

On the Oncollision, we put what happens when player hits with the Quest, on my case i made that active the Quest, and when hit with the other collider, player is TP on (0,0), to indicate that the Quest is over.

The only what we need to do for finish is initialize the quest on the scene.



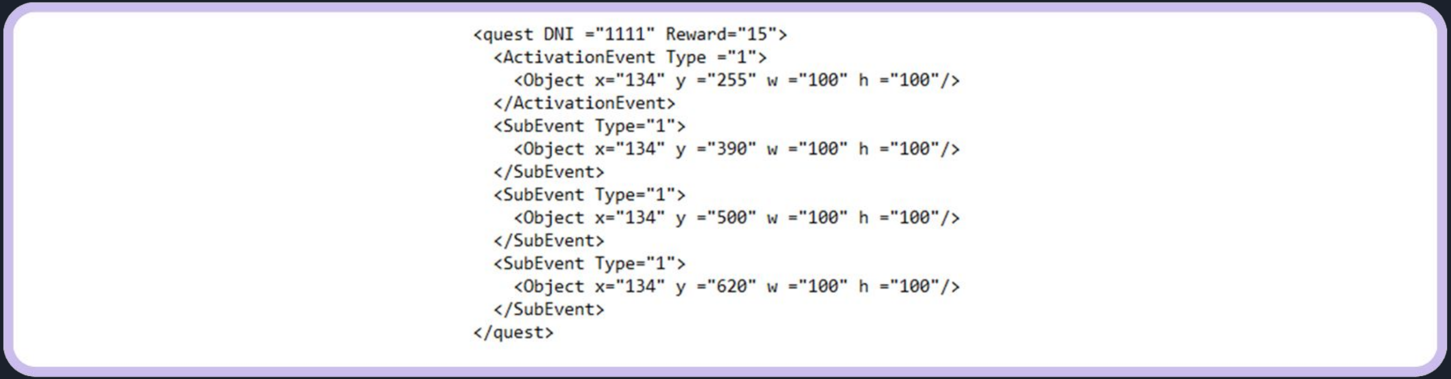
```
bool Quest1::Update(float dt)
{
    SDL_Rect Quest1 = { 200, 200, 50, 50 };
    app->render->DrawRectangle(Quest1, 0, 0, 255);

    return false;
}
```



```
// Called before the first frame
bool Scene::Start()
{
    Quest1* quest1 = (Quest1*)app->questManager->CreateQuest(QuestType::QUEST1, 0, { 0, 0 });

    return true;
}
```

XML

On XML, we are going to put the information of the quest, position if this are active or not, if any goal is met. and then we are going to initialize on the awaken, loadstate and save state.



```
<quest DNI ="1111" Reward="15">
  <ActivationEvent Type ="1">
    <Object x="134" y ="255" w ="100" h ="100"/>
  </ActivationEvent>
  <SubEvent Type="1">
    <Object x="134" y ="390" w ="100" h ="100"/>
  </SubEvent>
  <SubEvent Type="1">
    <Object x="134" y ="500" w ="100" h ="100"/>
  </SubEvent>
  <SubEvent Type="1">
    <Object x="134" y ="620" w ="100" h ="100"/>
  </SubEvent>
</quest>
```

Webgrafia

https://en.wikipedia.org/wiki/Quest_(video_games)