Documentation “CSVImporter“

Table Of Content

Overview

Tech Details

Engine: Unity 2021.3.18f1 or higher

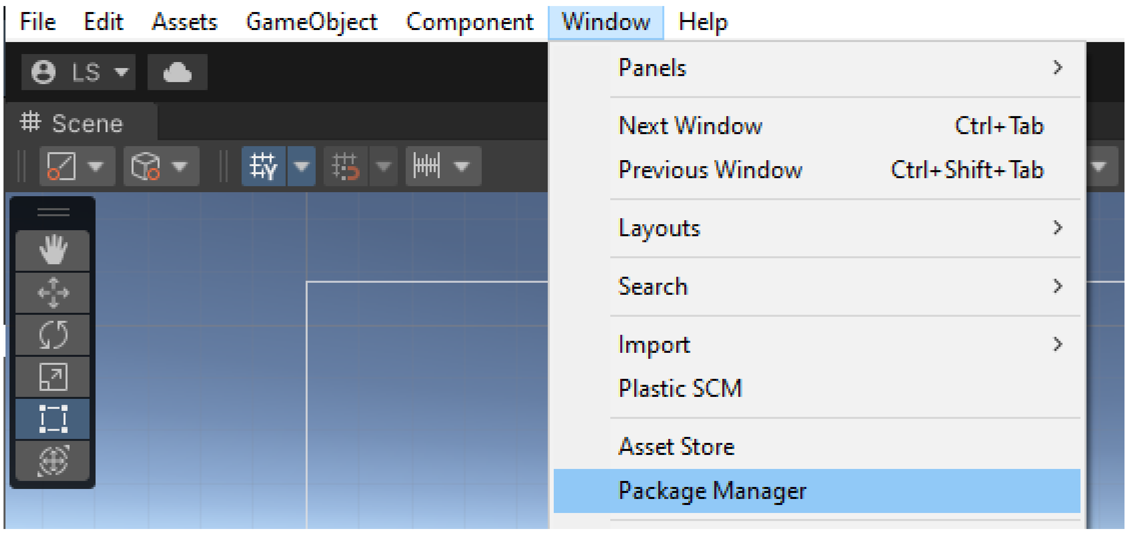
About the tool

This tool has two features. The first one is about converting any CSV file in unity and creating scriptable objects. The second part in this tool is about a basic NPC quest system. It is easy to build on and it contains three types of quest givers (Collect, Kill, Meet)

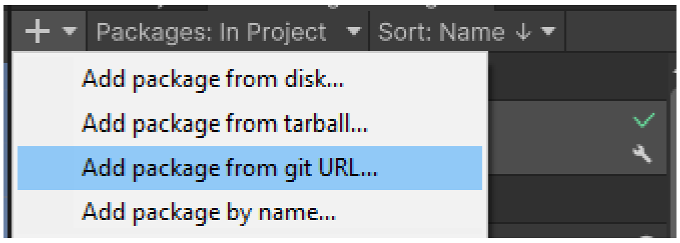
Installation and Import

Installation

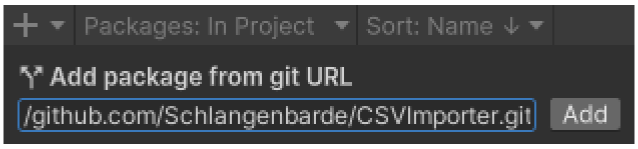
- Inside of Unity open the package manager



- Add package from git url

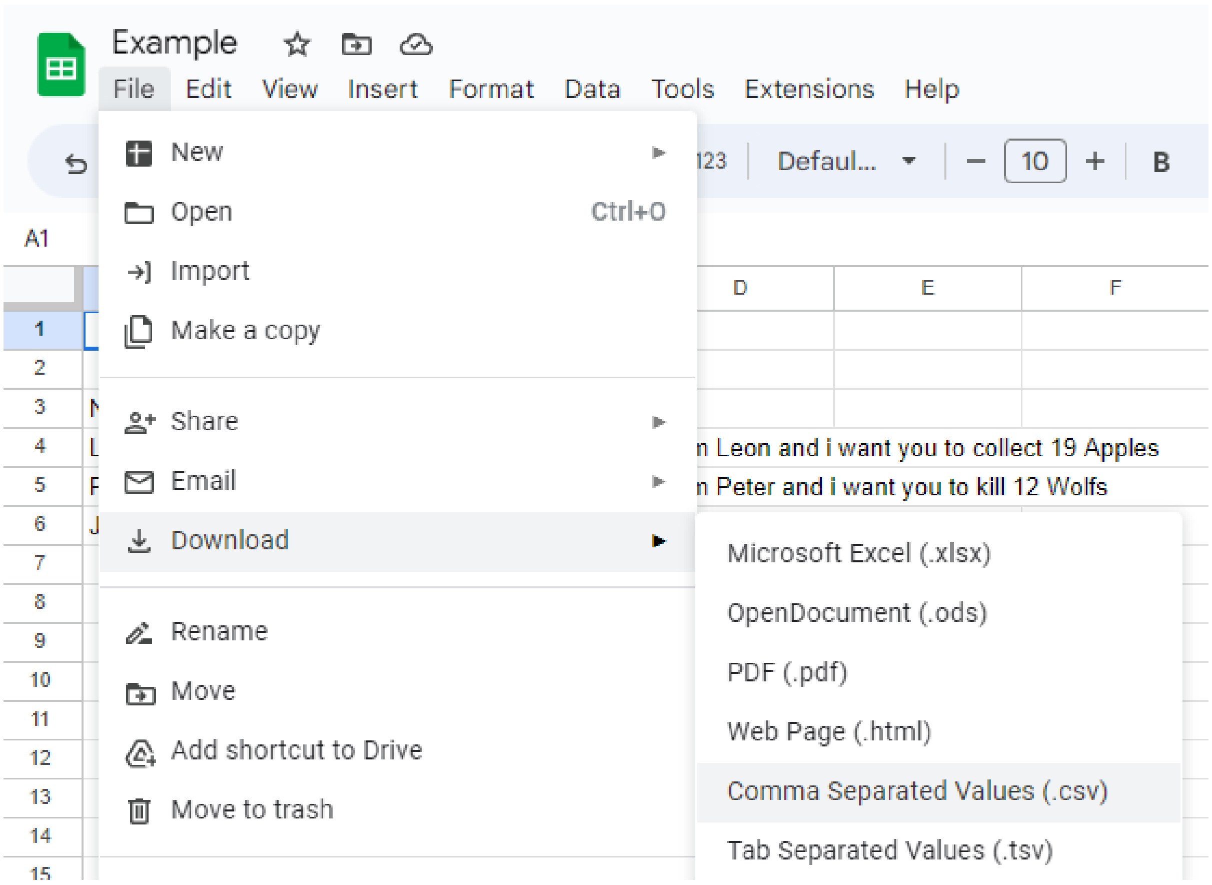


- Enter this URL: https://github.com/Schlangenbarde/CSVImporter.git and click on Add



Export CSV File

Google Sheets

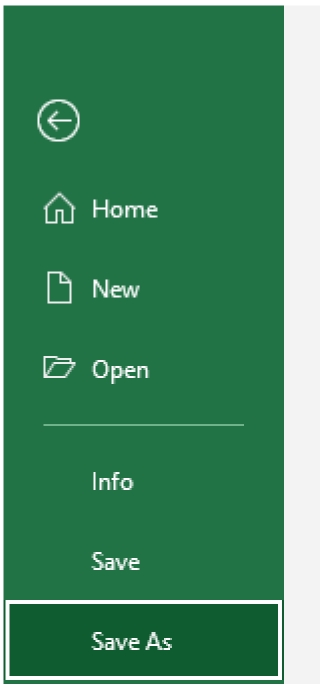


n Leon and i want you to collect 19 Apples
n Peter and i want you to kill 12 Wolfs

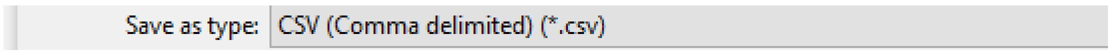
- *Go To File*
- under the tab *Download*
- select *Comma Separated Values (.csv)*

Excel

- *Select File*
- *Go to Save as*



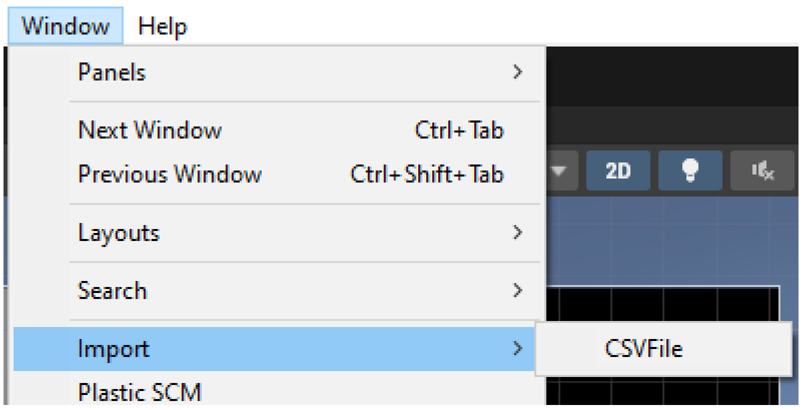
- Choose your path where you want to save your File
- Make sure to select the type *CSV*


Save as type: CSV (Comma delimited) (*.csv)

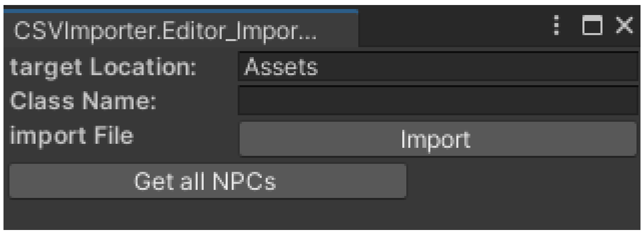
Import

This step explains how to import your CSV file.

- Click on the window tab at the top and select "Import/CSVFile"



- *A new window is going to open up*



Window:

target Location

- This is the data path where the scriptable objects should be created. (Example: Assets/Data/ScriptableObjects/)

Class Name

- This is the name of the class which is used for the scriptable objects. IMPORTANT: This class needs to derive from BaseImportObject.

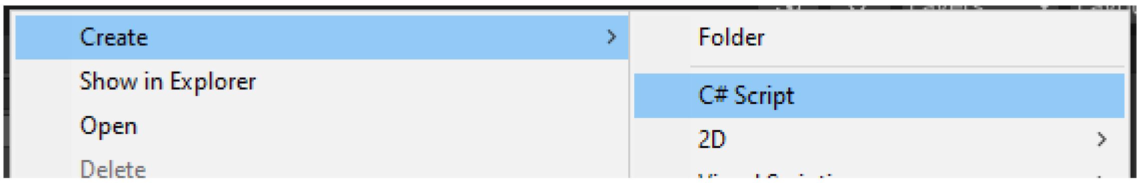
Import File

- This allows you to select a CSV file from your computer and creates scriptable objects.

Get all NPCs

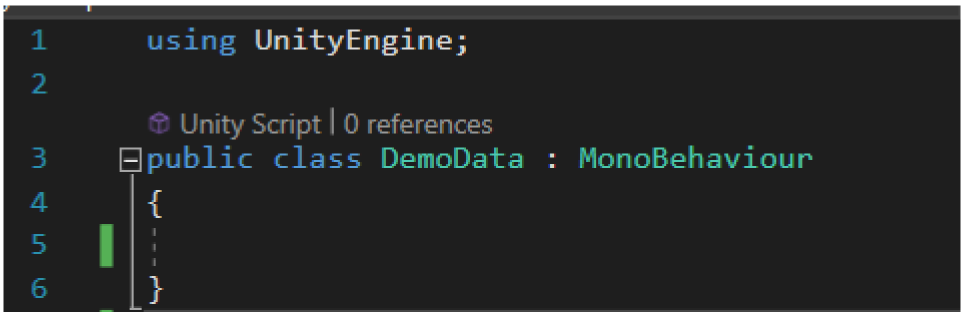
- This creates a list of all objects with the NPC script on it

Create new data



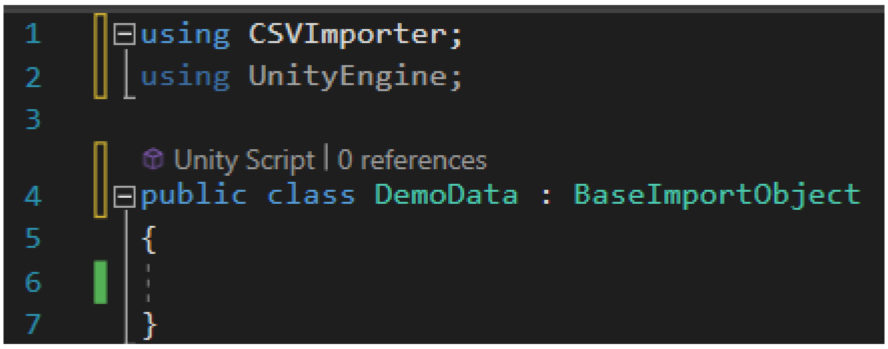
- *Create a new C# script*

- Remove everything unnecessary (Start and Update methods. All unused nameSpaces)



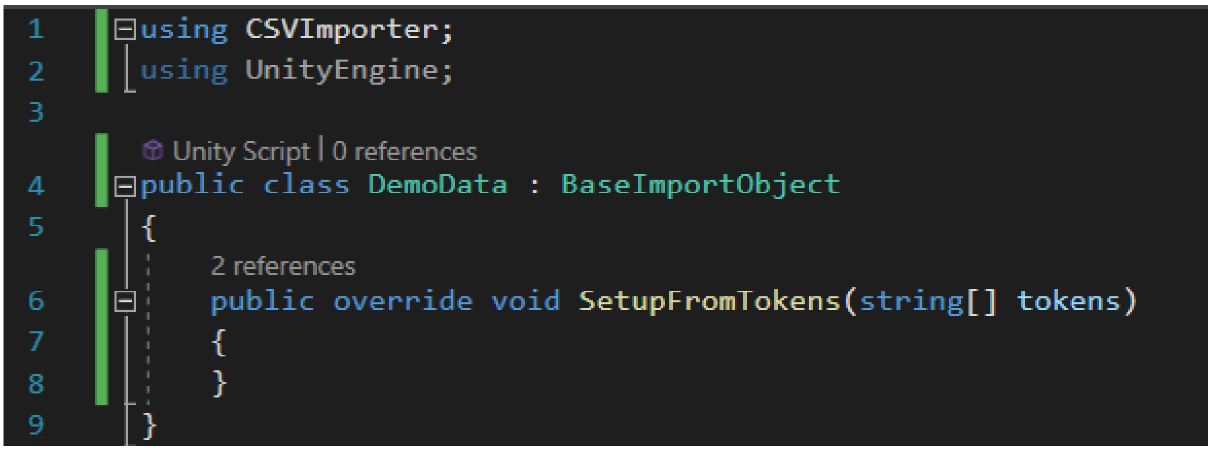
```
using UnityEngine;

Unity Script | 0 references
public class DemoData : MonoBehaviour
{

}
```

- IMPORTANT: This script needs to derive from "**BaseImportObject**" otherwise it won't work



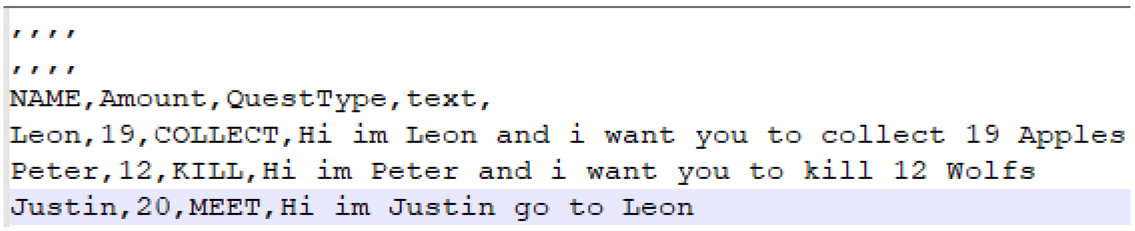
```
using CSVImporter;
using UnityEngine;

Unity Script | 0 references
public class DemoData : BaseImportObject
{

}
```

- There is a method (SetupFromTokens) which you can overrite. This is used to set up your data using the csv file importer.



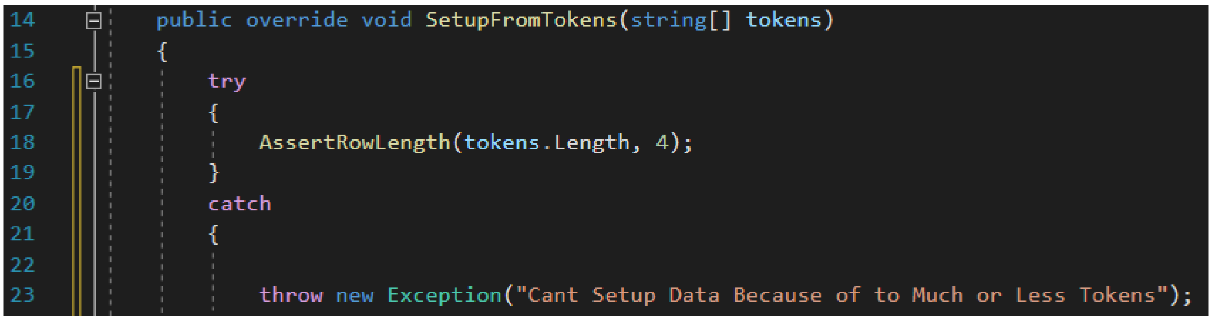
```
using CSVImporter;
using UnityEngine;

Unity Script | 0 references
public class DemoData : BaseImportObject
{
    2 references
    public override void SetupFromTokens(string[] tokens)
    {
    }
}
```

- After exporting your data. Example:



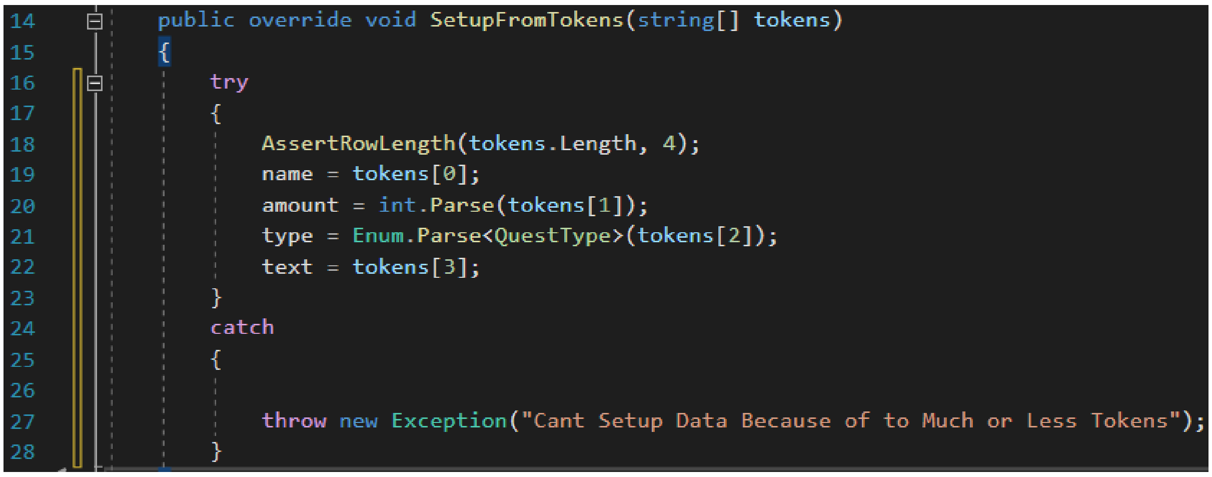
```
,,,,
,,,,
NAME,Amount,QuestType,text,
Leon,19,COLLECT,Hi im Leon and i want you to collect 19 Apples
Peter,12,KILL,Hi im Peter and i want you to kill 12 Wolfs
Justin,20,MEET,Hi im Justin go to Leon
```

- We also want to make sure there is enough data to import. There is a method in the base class which throws an exception. (The amount needed is dependent on how much data you have. In my example there are 4 (Name, Amount, QuestType and text)



```
    public override void SetupFromTokens(string[] tokens)
    {
        try
        {
            AssertRowLength(tokens.Length, 4);
        }
        catch
        {

            throw new Exception("Cant Setup Data Because of to Much or Less Tokens");
```

- Be sure to keep the correct order when assigning the variables. One string array is equal to one row in your csv file.



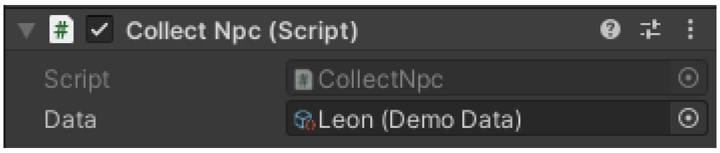
```
public override void SetupFromTokens(string[] tokens)
{
    try
    {
        AssertRowLength(tokens.Length, 4);
        name = tokens[0];
        amount = int.Parse(tokens[1]);
        type = Enum.Parse<QuestType>(tokens[2]);
        text = tokens[3];
    }
    catch
    {

        throw new Exception("Cant Setup Data Because of to Much or Less Tokens");
    }
```

Quest System

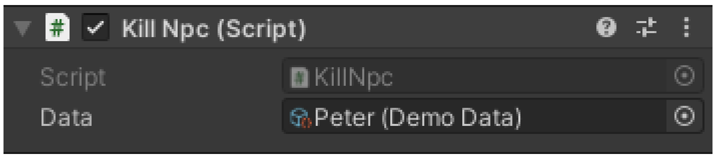
NPC Types

CollectNpc

- This Npc has a quest which has you collect x amount of DemoCollectable

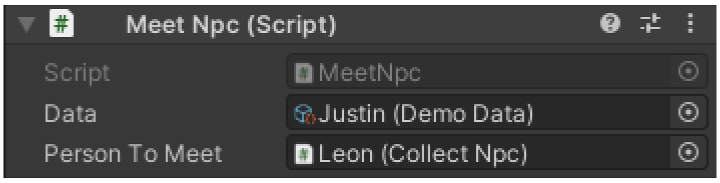

Collect Npc (Script)
Script CollectNpc
Data Leon (Demo Data)

KillNpc


Kill Npc (Script)
Script KillNpc
Data Peter (Demo Data)

- This Npc has a quest which has you kill x amount of DemoKillable

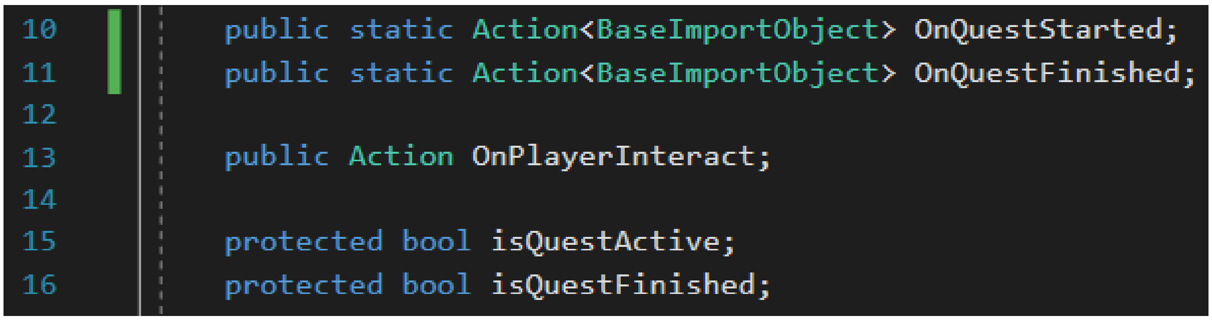
MeetNpc


Meet Npc (Script)
Script MeetNpc
Data Justin (Demo Data)
Person To Meet Leon (Collect Npc)

- This Npc has a quest which has you meet a certain Npc

Base Npc class

Variables

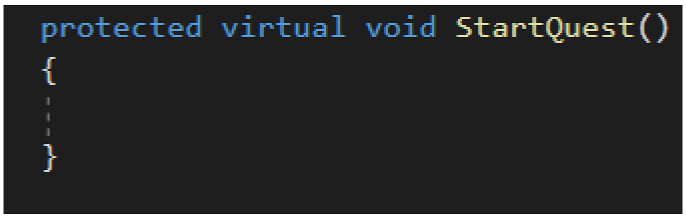


```
10      public static Action<BaseImportObject> OnQuestStarted;
11      public static Action<BaseImportObject> OnQuestFinished;
12
13      public Action OnPlayerInteract;
14
15      protected bool isQuestActive;
16      protected bool isQuestFinished;
```

- In **line 10/ 11** those are events which get called when the quest starts or finishes. This is used for the quest log.
- **Line 13 is** an event which other objects can subscribe to in order to react if the player is interacting with one Npc.
- **Line 15/ 16** are bools to check if the quest is active or finished.

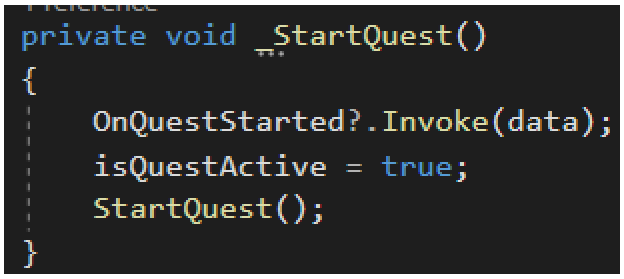
Methods

- **Naming Conventions**
  - "MethodName" Methods without an underscore at the beginning are virtual methods which can be overwritten.



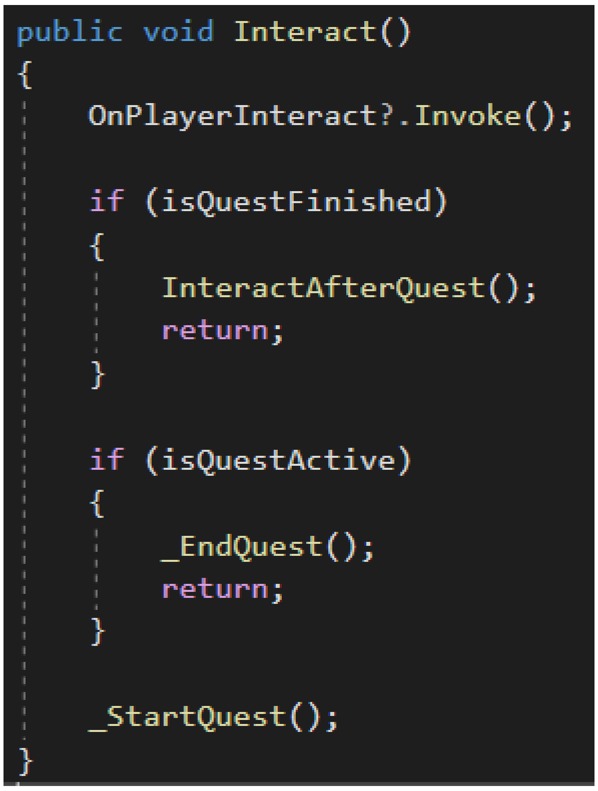
```
protected virtual void StartQuest()
{

}
```

  - "_MethodName" Methods with an underscore at the beginning are methods for the npc logic and which are getting called.



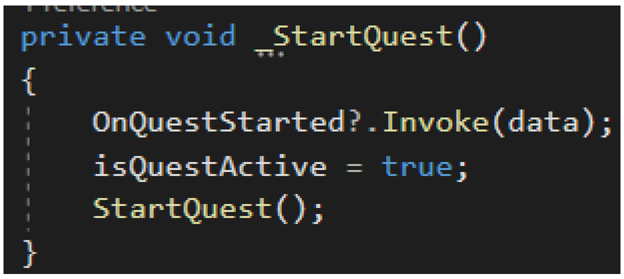
```
private void _StartQuest()
{
    OnQuestStarted?.Invoke(data);
    isQuestActive = true;
    StartQuest();
}
```

- **Interact()**
    - This Method is the start point. This can be called outside. The method checks if the quest has already started or finished.
    - If the quest is already active it calls the funktion _EndQuest().
    - If the quest has finished it calls InteractAfterQuest().
    - If the quest hasn't started or finished the method _StartQuest() gets called.



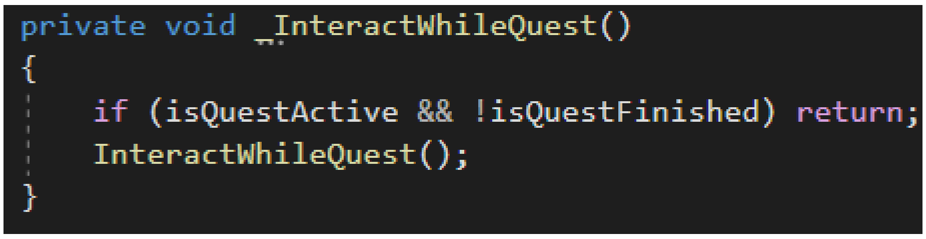
```
public void Interact()
{
    OnPlayerInteract?.Invoke();

    if (isQuestFinished)
    {
        InteractAfterQuest();
        return;
    }

    if (isQuestActive)
    {
        _EndQuest();
        return;
    }

    _StartQuest();
}
```

- **_StartQuest()**
    - This Method…
    - … gets called when the quest is going to start.
    - … invokes the event OnQuestStarted.
    - … sets the bool parameter isQuestActive on true.



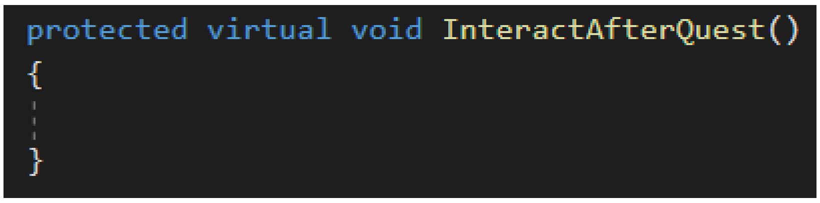
```
private void _StartQuest()
{
    OnQuestStarted?.Invoke(data);
    isQuestActive = true;
    StartQuest();
}
```

- **_InteractWhileQuest()**
    - This method is called when you interact with an npc where the quest is already started but not finished.



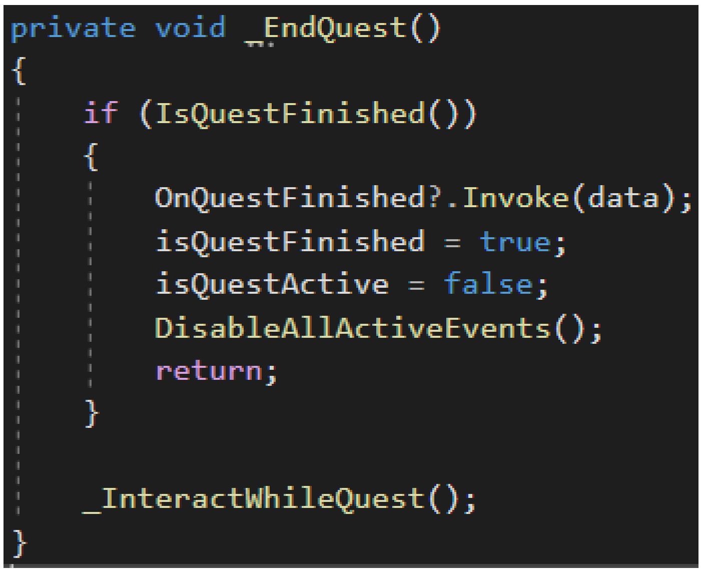
```
private void _InteractWhileQuest()
{
    if (isQuestActive && !isQuestFinished) return;
    InteractWhileQuest();
}
```

- **InteractAfterQuest()**
    - This method is used to implement logic for what happens after a quest is finished.



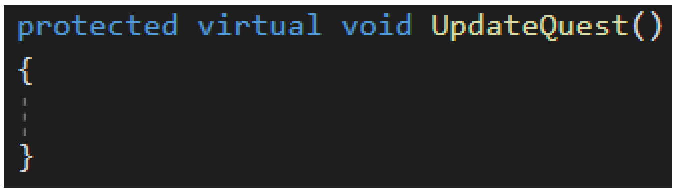
```
protected virtual void InteractAfterQuest()
{

}
```

- **_EndQuest()**
    - If IsQuestFinished() returns true
    - OnQuestFinshed gets invoked
    - Bools are getting set
    - Method DisableAllActiveEvents()



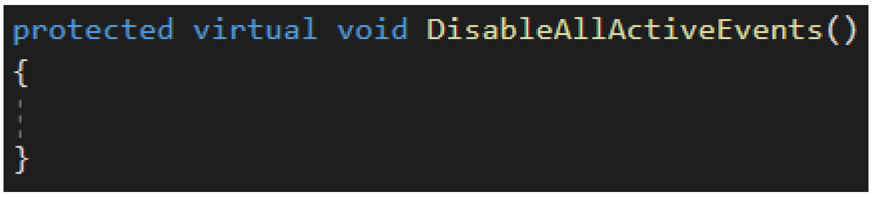
```
private void _EndQuest()
{
    if (IsQuestFinished())
    {
        OnQuestFinished?.Invoke(data);
        isQuestFinished = true;
        isQuestActive = false;
        DisableAllActiveEvents();
        return;
    }

    _InteractWhileQuest();
}
```

- **UpdateQuest()**
    - This Method is used to update the questlog



```
protected virtual void UpdateQuest()
{

}
```

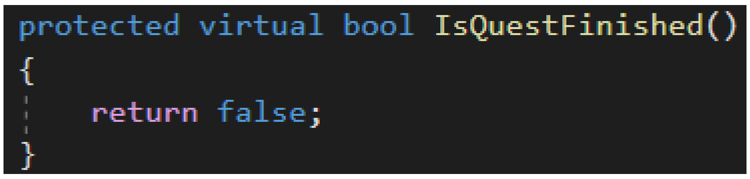
- **DisableAllActiveEvents()**
    - This method is used to disconnect all active events



```
protected virtual void DisableAllActiveEvents()
{

}
```

    - Gets called on the EndQuest() and OnDisable()

- **IsQuestFinished()**
    - Here goes all the logic needed to determine if all quest

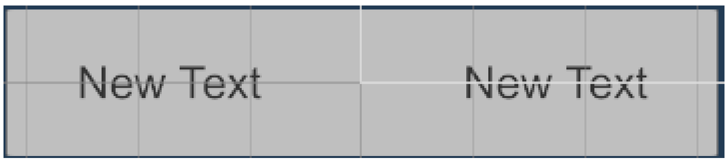


```
protected virtual bool IsQuestFinished()
{
    return false;
}
```

conditions are finished

QuestLog

QuestLogTemplate

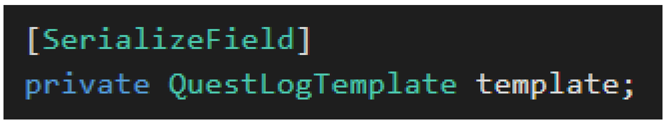


- Example for a quest log Item
- On this template is the script “QuestLogTemplate” attached.
- This script contains three methods…
    - …SetupData: This is the initial method. It gets called to fill in the correct text
    - …UpdateData: These are methods which are linked with events. It is used to update the strings.
    - …OnQuestFinished: If the quest finishes this gets called. It is used to change the color of this template.

QuestLog

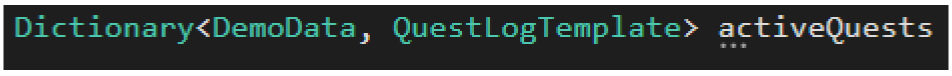
- Contains all active quests and its according QuestLogTemplate

Variables



```
[SerializeField]
private QuestLogTemplate template;
```

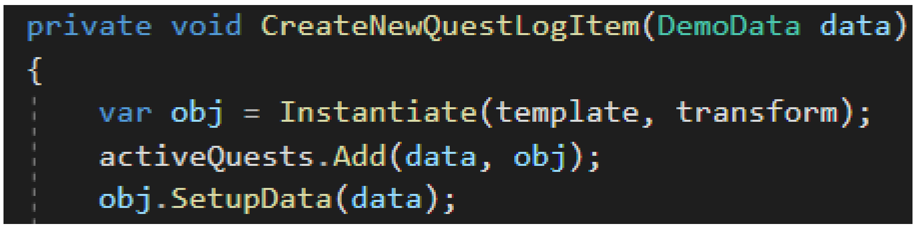
- template: It is a reference to the QuestLogTemplate
- activeQuests: This is used to store all activeQuests and its according log item



```
Dictionary<DemoData, QuestLogTemplate> activeQuests
```

Methods

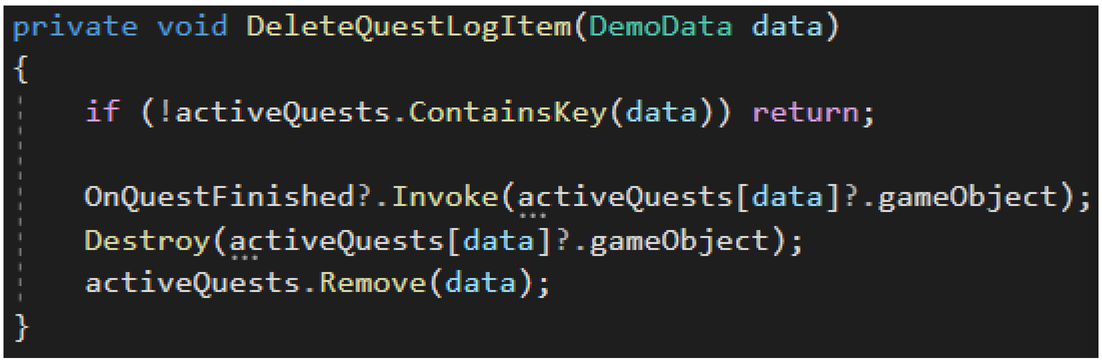
- CreateNewQuestLogItem:



```
private void CreateNewQuestLogItem(DemoData data)
{
    var obj = Instantiate(template, transform);
    activeQuests.Add(data, obj);
    obj.SetupData(data);
```

  - This Gets called when OnQuestStarted gets invoked
  - The template is getting instantiated and added to the dictionary.
  - The method SetupData gets called from the template

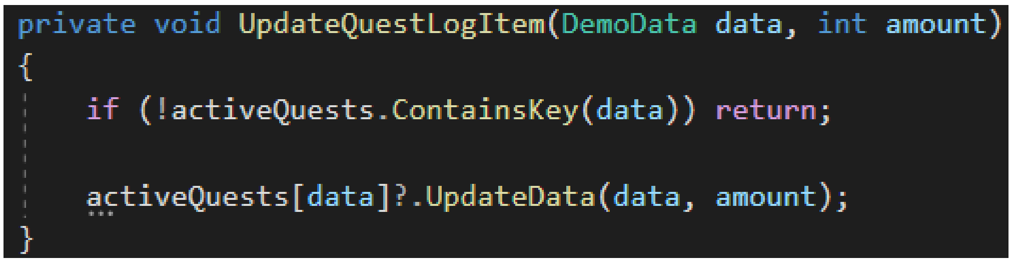
- DeleteQuestLogItem:



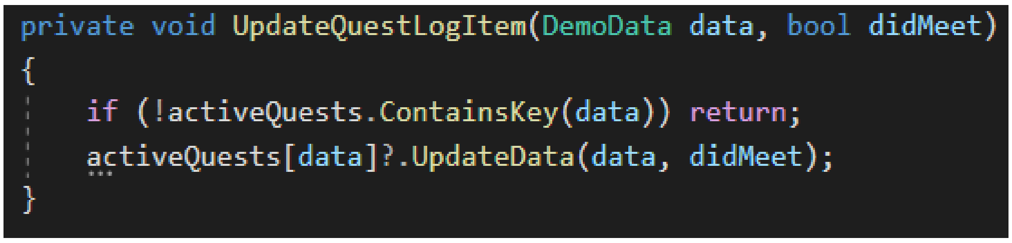
```
private void DeleteQuestLogItem(DemoData data)
{
    if (!activeQuests.ContainsKey(data)) return;

    OnQuestFinished?.Invoke(activeQuests[data]?.gameObject);
    Destroy(activeQuests[data]?.gameObject);
    activeQuests.Remove(data);
}
```

  - This gets called when OnQuestFinished gets invoke
  - It first makes sure that the data exists inside of activeQuests.
  - It invokes an event which creates an LogItem inside of the finished quest log tab
  - It destroys the old log item gameobject and remove it from the activeQuest List

- UpdateQuestLogItem:



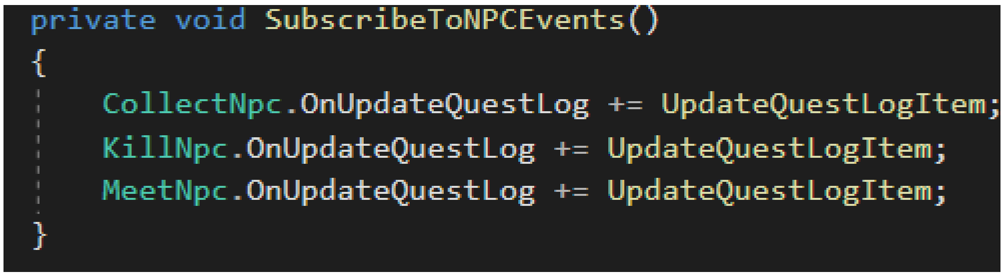
```
private void UpdateQuestLogItem(DemoData data, int amount)
{
    if (!activeQuests.ContainsKey(data)) return;

    activeQuests[data]?.UpdateData(data, amount);
}
```



```
private void UpdateQuestLogItem(DemoData data, bool didMeet)
{
    if (!activeQuests.ContainsKey(data)) return;
    activeQuests[data]?.UpdateData(data, didMeet);
}
```

  - These methods are used to update the questlogItems
  - First checks if there is a quest
  - Then call the update method on the log item.

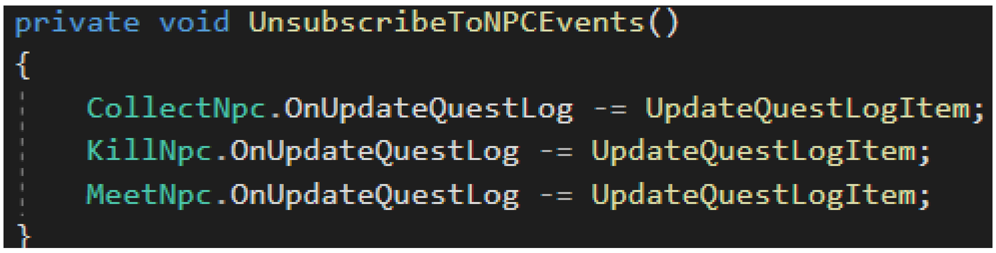
- SubscribeToNPCEvents:



```
private void SubscribeToNPCEvents()
{
    CollectNpc.OnUpdateQuestLog += UpdateQuestLogItem;
    KillNpc.OnUpdateQuestLog += UpdateQuestLogItem;
    MeetNpc.OnUpdateQuestLog += UpdateQuestLogItem;
}
```

  - This method is used to subscribe to all NPC events

- UnsubscribeToNPCEvents:



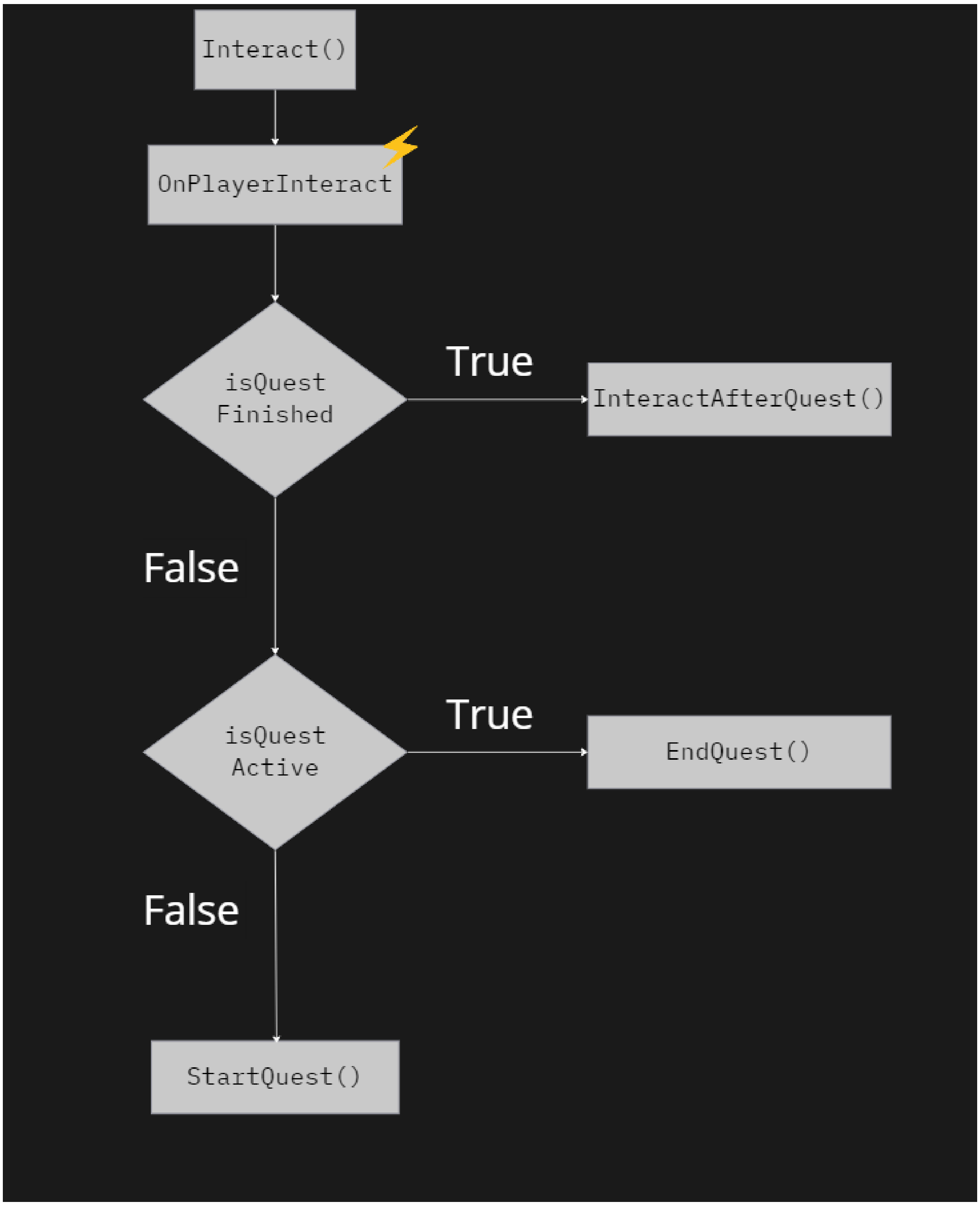
```
private void UnsubscribeToNPCEvents()
{
    CollectNpc.OnUpdateQuestLog -= UpdateQuestLogItem;
    KillNpc.OnUpdateQuestLog -= UpdateQuestLogItem;
    MeetNpc.OnUpdateQuestLog -= UpdateQuestLogItem;
}
```

  - This method is used to unsubscribe to all NPC events

- FinishedQuestLog
    - This is used to instantiate a QuestLogTemplate into the finishedQuestLog tab

Execution order

Npc


Interact()
OnPlayerInteract
isQuest Finished
True
InteractAfterQuest()
False
isQuest Active
True
EndQuest()
False
StartQuest()