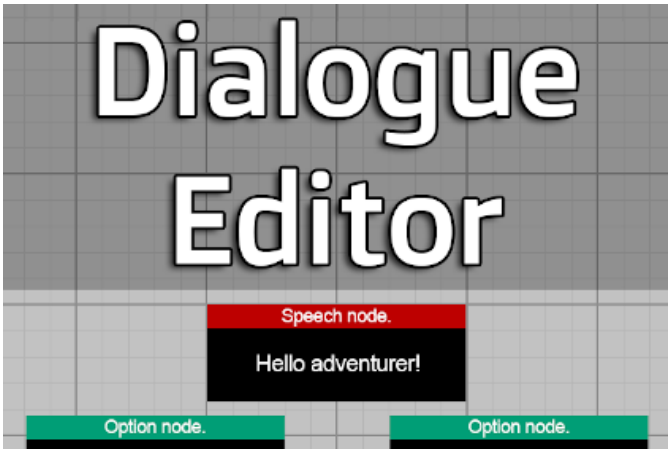
Unity工具：对话编辑器



1个链接

- 资产存储
- 在线文档
- 视频教程播放列表

2个书面教程

- 什么是对话框编辑器？
- 编辑器窗口
- 对话管理器UI预制件
- 触发对话
- 自定义输入
- 回调
- 对话数据结构

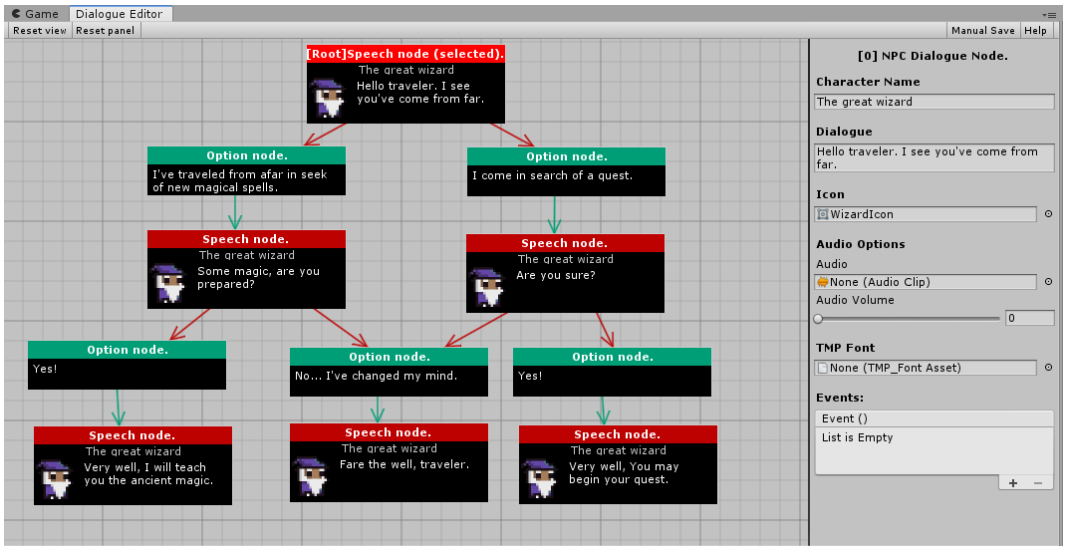
3.什么是对话框编辑器？

对话编辑器是一个Unity工具，它可以让你快速、轻松地将对话添加到游戏中。该工具带有一个编辑器窗口，允许您创建和编辑对话。
这个工具还附带了一个预先制作的、可定制的UI预制件，因此不需要进行UI编程。但是，如果您熟悉编程，并希望创建自己的UI实现，那么每个会话都可以作为一个简单的数据结构来访问。

4编辑器窗口

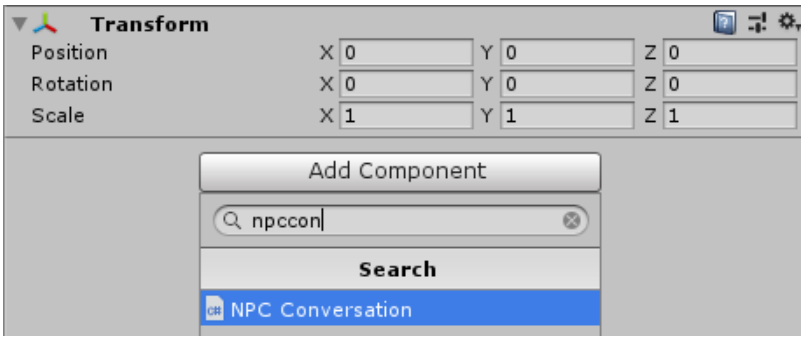
4.1 介绍

对话由语音节点和选项节点组成。语音节点表示角色会说的话，选项节点表示玩家可用的选项。这些节点之间的连接显示了对话的流程。



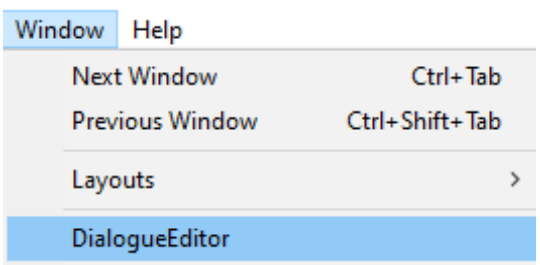
4.2 创建对话对象

为了创建一个会话，请创建一个新的游戏对象，并添加脚本NPC会话。



4.3 打开编辑器窗口

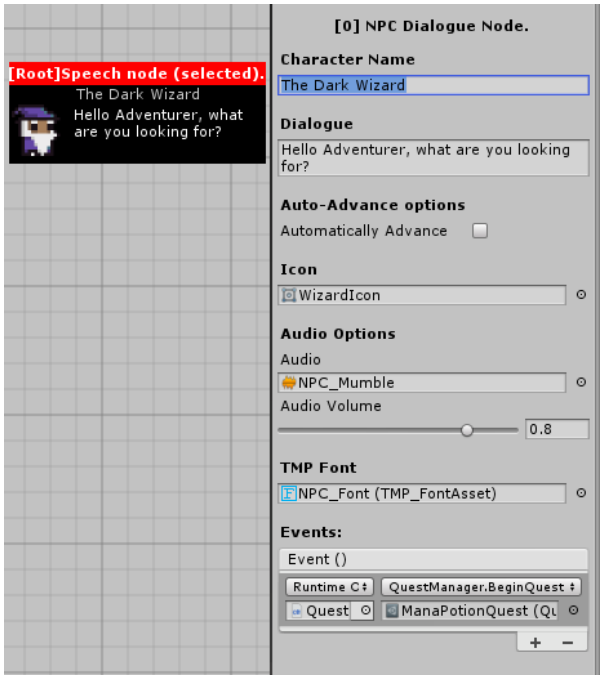
要打开编辑器窗口，请选择窗口→对话框编辑器。在层次结构中选择一个对话，以便在编辑器窗口中编辑该对话。



4.4 语音节点

创建新对话时，它将包含单个语音节点-这是对话的开始。
单击一个语音节点以对其进行编辑。语音节点具有以下变量：

- ●角色名称：这是正在说话的角色的名称。
- ●对话框：这是对节点的演讲。
- ●自动前进：如果一个语音节点引到另一个语音节点，或没有，此选项可用。当选择此选项时，对话框将自动继续，而用户不需要单击任何内容。
  - -显示继续选项：“继续”/“结束”选项是否仍然显示？
  - -对话时间：等待多久对话才能自动进行。
- ●图标：这是将出现在演讲旁边的NPC的图标。
- ●音频：这是一个可选的变量，您可以用这个语音播放音频。
- ●TMPFont：这是这个演讲的文本MeshPro字体。您可以逐个节点地设置字体。
- ●事件：这些是统一的事件，将在播放对话中的这个语音节点时运行。



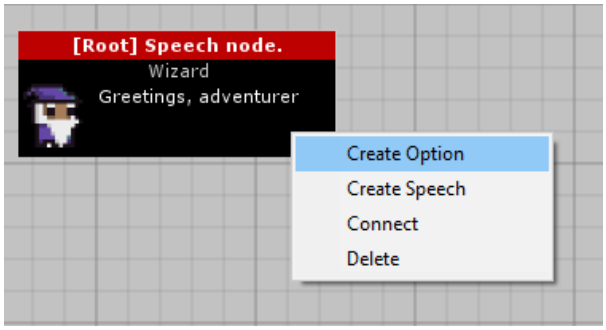
4.5 选项节点

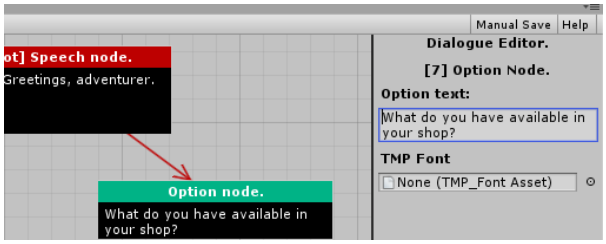
选项节点表示用户可以选择的选项。
单击一个选项节点以对其进行编辑。选项节点具有以下变量：

- ●选项文本：这是该选项的文本。
- ●TMP字体：这是选项文本将使用的文本字体。

4.6 创建节点

若要创建新节点，请右键单击现有节点。选择“ 创建语音 ”或“ 创建选项 ”。然后，左键单击某个地方以放置节点。





4.7 连接节点

要连接两个现有节点，请右键单击一个节点，然后单击“ 连接 ”。然后，左键单击您希望将其连接到的节点。
语音节点可以连接到选项节点或其他语音节点。

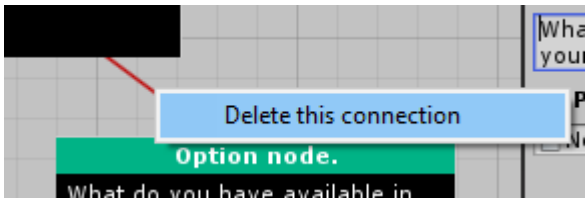
- ●如果一个语音节点连接到选项节点，这些选项将显示给玩家。
- ●如果一个语音节点连接到另一个语音节点，则之后会出现以下语音节点。
- ●如果一个语音节点没有连接，它标志着对话的结束。

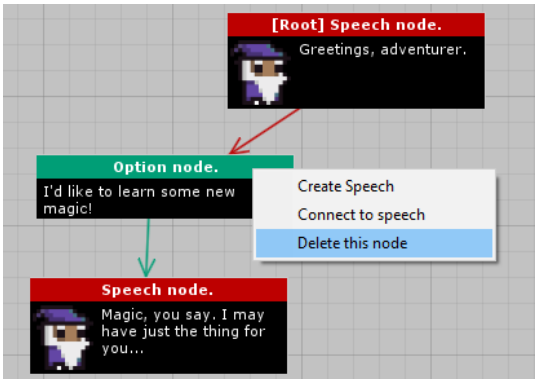
选项节点只能连接到语音节点。

- ●如果选项节点连接到语音节点，则在选择该选项后将出现以下语音节点。
- ●如果选项节点没有连接，对话将在选择该选项后结束。

4.8 删除节点和连接

通过右键单击箭头并单击“ 删除此连接 ”，可以删除节点之间不需要的连接
同样地，也可以通过右键单击该节点并点击“ 删除此节点 ”来删除不需要的节点。删除一个节点也将删除此节点之间的任何连接。



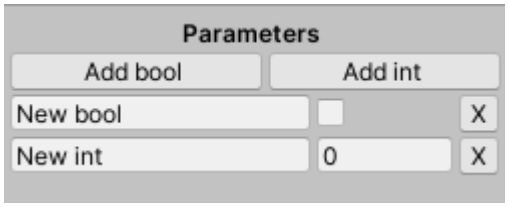


4.9 参数和条件

一个对话可以有一些参数。这些值将具有名称和值；该值可以进行更新。节点之间的连接可以有条件，因此只有在满足条件时，连接才会有效。条件需要一个参数值来满足某些要求。

4.9.1 添加参数

通过在对话中没有选择任何内容，您可以看到这些参数。可以通过单击“添加Int”和“添加Bool”按钮分别添加Int和Bool参数。您可以重新命名这些参数，并给它们提供一个默认值。



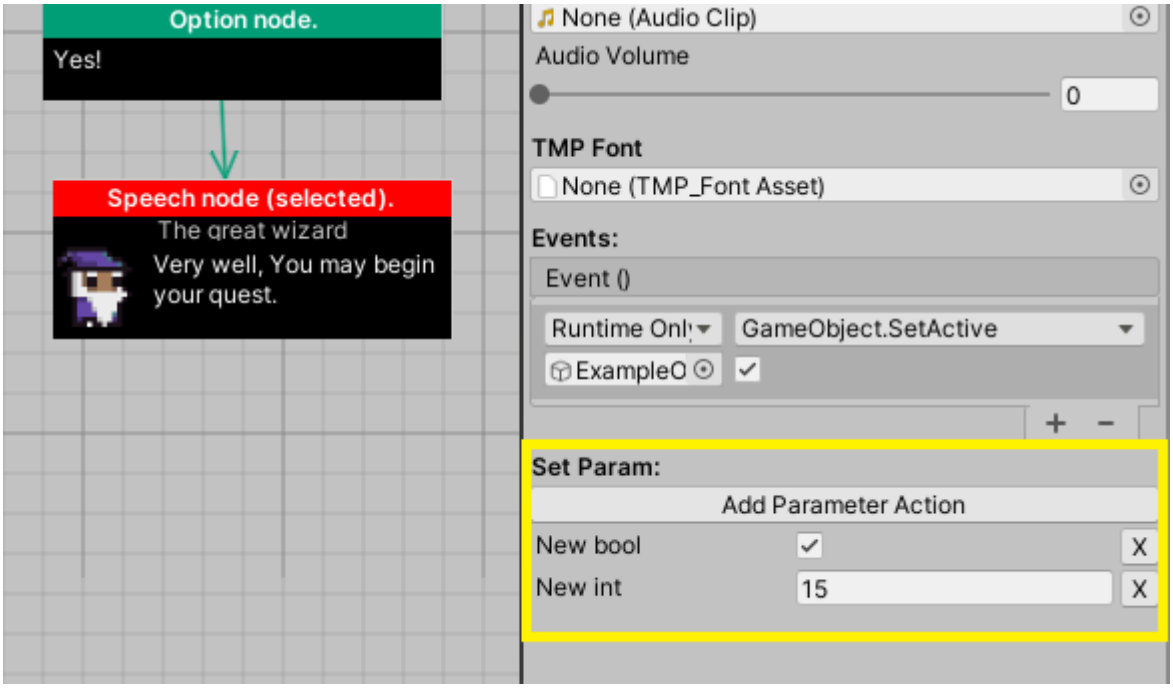
4.9.2 添加条件

通过单击一个连接，您可以设置条件。此连接只有在满足所有条件时才有效。这允许您创建一些场景，其中特定的选项或对话片段将只显示给满足需求的玩家。

4.9.3 设置参数值

按行动

通过单击语音节点或选项节点，您可以单击“ 添加参数操作 ”。这允许您在激活语音或选择该选项时设置任何参数的值。



按代码

您还可以通过代码获取和设置运行时的参数值。您可以通过调用对话管理器上的适当函数来实现这一点。

注意：您需要将“ 对话框编辑器 ”添加到脚本的名称空间。这可以通过
在顶部添加以下几行：

```
使用对话框编辑器；
```

您可以调用以下函数来获取/设置参数值：

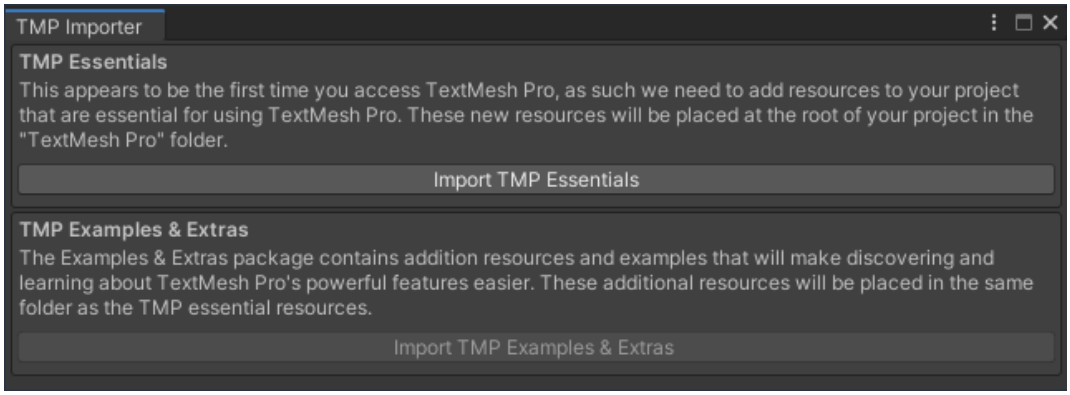
```
//集
ConversationManager .示例SetBool （“ BoolName ”，真）；
ConversationManager .示例SetInt("IntName", 1); 4

//获取
bool bVal = ConversationManager.Instance.GetBool("BoolName");
int iVal = ConversationManager.Instance.GetInt("IntName");
```

5对话经理

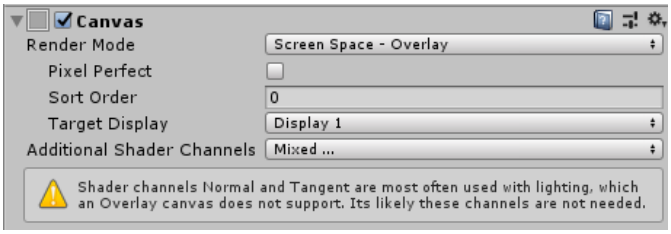
提供了一个预先制作的、可定制的UI预制件。对话管理器包含所有的UI，因此，要将其添加到场景中，您可以将其作为UI画布的子元素进行拖动。

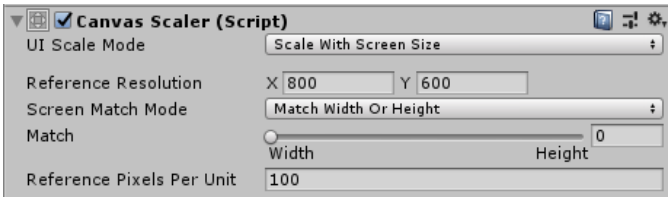
当你拖动对话主页预制件到你的场景时，如果你在Unity项目中没有TextMeshPro，它将提示你导入TMP必需品。导入后，请将对话管理器从层次结构中删除，然后将其重新添加到层次结构中，以便使用TMP正确呈现。



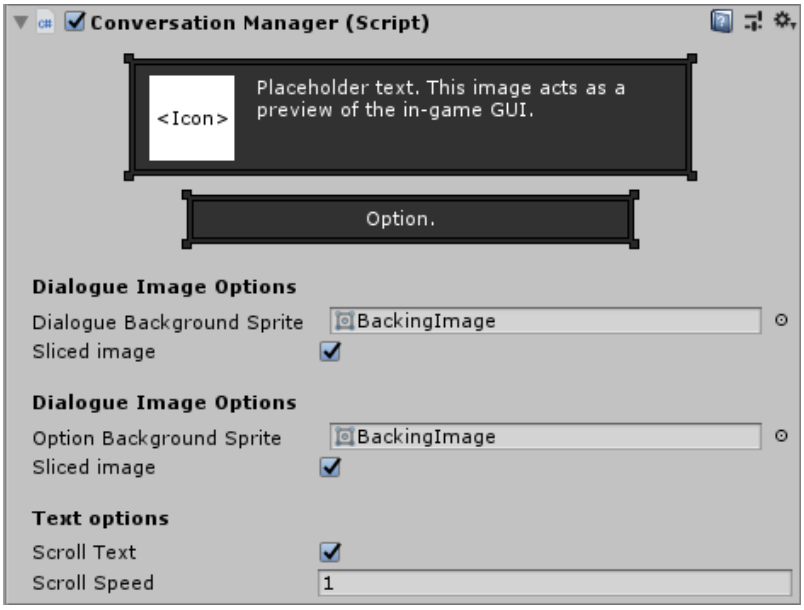
推荐的画布设置：

- ●画布-渲染模式：屏幕空间-覆盖
- **●画布缩放器- UI缩放模式：按屏幕大小比例**





对话管理器为对话框和选项框的背景图像提供了选项。这些图像可以选择是9个切片的图像。预览渲染将显示在这些选项的上方。您还可以选择文本滚动选项。



6触发对话

如果您正在使用会话管理器UI预制，可以通过调用单个函数触发对话，并通过NPC对话变量：

```
ConversationManager .示例StartConversation(/ *NPCConversation */);
```

注意：您需要将“ 对话框编辑器 ”添加到脚本的名称空间。这可以通过
在顶部添加以下几行：

```
使用对话框编辑器；
```

下面是一些示例代码，它显示了一个非常基本的NPC类，当点击NPC时，它会开始一个对话：

```
使用Unity引擎；
使用对话框编辑器；

public class NPC : MonoBehaviour
{
   //NPC会话变量（在检查员中分配）公开的NPC会话对话；

   private void OnMouseOver()
   {
      如果（输入。GetMouseButtonDown (0)）
      {
         ConversationManager .示例开始对话（对话）；}
   }
}

你还有一些其他的属性和功能：

//：目前正在进行一场对话吗？
ConversationManager .示例IsConversationActive;

//当前会话（如果没有活动的会话，则为空）。
ConversationManager .示例CurrentConversation;

//提前结束一个对话(e。 g .玩家会离开)。
ConversationManager .示例EndConversation();
```

7自定义输入

对话编辑器提供了一些基本的功能，允许您与对话UI进行交互。这使您能够支持游戏支持的任何输入方法，如键盘+鼠标或控制器。

三个基本功能允许您循环到下一个或上一个选项，并按当前选择的选项：

```
//循环到上一个选项对话管理器。示例选择首选选项（）；//循环到下一个选项
ConversationManager .示例选择Next选项（）；//按当前选择的选项对话管理器。示例PressSelectedOption();
```

下面是一些示例代码，它显示了键盘支持的对话UI：

使用Unity引擎；

使用对话框编辑器；

```
public class ExampleInputManager : MonoBehaviour
{
  private void Update()
  {
    if (ConversationManager.Instance != null)
    {
      如果（对话管理器。示例is对话活动）
      {
        如果（输入。GetKeyDown (KeyCode .UpArrow)) ConversationManager .
          示例SelectPreviousOption();

        else if (Input.GetKeyDown(KeyCode.DownArrow))
          ConversationManager .示例SelectNextOption();

        else if (Input.GetKeyDown(KeyCode.F))
          ConversationManager .示例PressSelectedOption();}
    }
  }
}
```

在对话管理器预制件上还有一个选项，它允许您选择是否应该启用鼠标交互。

8个回调

如果您使用对话管理器UI预制，则有两个回调可用，分别在对话开始和结束时调用。

```
DialogueEditor .ConversationManager .开始对话
DialogueEditor .ConversationManager .对话结束
```

注意：您需要将“ 对话框编辑器 ”添加到脚本的名称空间。这可以通过
在顶部添加以下几行：

```
使用对话框编辑器；
```

示例用例：

```
使用Unity引擎；
使用对话框编辑器；

public class ExampleClass : MonoBehaviour
{
  private void OnEnable()
  {
    ConversationManager.OnConversationStarted += ConversationStart;
    ConversationManager.OnConversationEnded += ConversationEnd;
  }

  private void OnDisable()
  {
    ConversationManager.OnConversationStarted -= ConversationStart;
    ConversationManager.OnConversationEnded -= ConversationEnd;
  }

  private void ConversationStart()
  {
    调试。日志(“ A␣交谈␣有␣开始");}

  private void ConversationEnd()
  {
    调试。日志(“ A␣交谈␣有␣结束");}
}
```

9对话数据结构

本节包含了针对那些希望编写自己的自定义UI实现的人的信息。如果您想使用该工具附带的预先制作的UI，则可以忽略此部分。

9.1 去往化

注意：您需要将“ 对话框编辑器 ”添加到脚本的名称空间。这可以通过
在顶部添加以下几行：

```
使用对话框编辑器；
```

为了反序列化会话，NPC会话包含了一个用于这样做的函数：它返回一个类型为“ 会话 ”的对象：

```
NPCConversation NPCConv;
会对话= NPCConv。Deserialize();
```

以下是您在创建自定义UI实现时需要关注的类：

9.2 会话

NPC对话反序列化为树状数据结构。反序列化将返回一个类型为“ 对话 ”的对象。一个对话对象包含一个语音节点；对话的根
-从这里开始，节点以树状模式连接。会话对象还包含会话的所有参数，以及设置和获取参数值的功能。

```
公共类对话{
  public Conversation()
  {
    Parameters = new List<Parameter>();
  }

  ///<摘要>对话的开始</摘要>公共语音节点根；

  ///<摘要>此会话的参数及其值</摘要>
  公共列表<参数>参数；

  /*方法的实现和私有方法都省略了*/

  公共空白SetInt（字符串参数名称、int值、输出参数状态状态）；
  公共空白SetBool（字符串参数名称、bool值、输出参数状态状态）
  公共intGetInt（字符串参数名称，oute参数状态状态）；
  公共boolGetBool（字符串参数名称，输出e参数状态状态）；}
```

9.3 对话节点

会话节点可以是语音节点或选项节点。会话节点包含有关该节点的所有数据（例如，对话文本、TMP字体、单元事件、参数操作等）以及该节点具有的所有连接。

```
公共抽象类会话节点{
  公共枚举eNodeType {
    演讲，
    选择
  }

  public ConversationNode()
  {
    Connections = new List<Connection>();
    ParamActions = new List<SetParamAction>();
  }

  public abstract eNodeType NodeType { get; }
  public Connection.eConnectionType ConnectionType { get { /* Ommited
      */ } }

  ///<摘要>节点的正文文本。</摘要>公共字符串文本；

  ///<摘要>此节点具有的子连接。</摘要>公共列表<连接>连接；

  ///<摘要>此节点参数操作。</摘要>公共列表<设置参数操作>参数操作；

  ///<摘要>此节点的文本
     . </summary>
  公共TMPro。TMP_FontAsset TMPFont;}


public class SpeechNode : ConversationNode
{
  public override eNodeType NodeType { get { return eNodeType.Speech;
      } }

  ///<总结了>正在说话的全国人大的名字。</摘要>公共字符串名称；

  ///<摘要>应该这个语音节点转到下一个
     自动地</总结>公共图书馆自动推进；
```

```
42      ///<摘要>如果这个语音节点，虽然考虑到自动推进，也显示一个“ 继续 ”或“ 结束 ”选
            项，供用户点击
        通过更快？</summary>
43      公共bool自动推进应显示选项；
44
45      /// <summary> If AutomaticallyAdvance==True, how long should this
            在进入下一个节点之前显示语音节点？</总结，>公共浮动时间，直到提前；
46
47      ///<总结>演讲的NPC图标</总结>公共雪碧图标；
48
49      公共AudioClip音频；
50      公众流动量；
51
52      ///<摘要>单元事件，将在此节点启动时触发。</ summary>
53      公共单位引擎。事件Unity事件事件
54
    public class OptionNode : ConversationNode
55  {
56      public override eNodeType NodeType { get { return eNodeType.Option;
57          } }
58  }
59
60
61
62
```

9.4 连接

每个对话节点都有一个包含0个或更多个连接的列表。连接包含一个语音节点或选项节点1以及条件列表。

```
公共抽象类连接{
  公共枚举e连接类型{
    没有一个
    演讲，
    选择
  }

  public Connection()
  {
    Conditions = new List<Condition>();
  }

  public abstract eConnectionType ConnectionType { get; }

  公共列表<条件>条件；}

public class SpeechConnection : Connection
{
  公共语音连接（语音节点节点）{
    Speech节点=节点；}

  public override eConnectionType ConnectionType { get { return
     eConnectionType.Speech; } }

  公共速度节点速度节点；}

public class OptionConnection : Connection
{
  公共选项连接（选项节点节点）{
    选项Node=节点；}

  public override eConnectionType ConnectionType { get { return
     eConnectionType.Option; } }

  公共选项节点选项节点；}
```

9.5 条件

每个连接都有一个包含0个或更多个条件的列表。每个条件都包含检查类型的名称（ 参数例如均衡、大于）和要求值。

```
公共抽象类条件{
  公共enum环境类型{
    IntCondition,
    BoolCondition
  }

  public abstract eConditionType ConditionType { get; }

  公共字符串参数名称；}

public class IntCondition : Condition
{
  公共枚举eCheckType {
    均等的
    lessThan,
    greaterThan
  }

  public override eConditionType ConditionType { get { return
     eConditionType.IntCondition; } }

  公共电子检查类型检查类型；
  公共int要求值；}

public class BoolCondition : Condition
{
  公共枚举eCheckType {
    均等的
    notEqual
  }

  public override eConditionType ConditionType { get { return
     eConditionType.BoolCondition; } }

  公共电子检查类型检查类型；
  公共bool要求值；}
```

9.6 参数

每个对话都可以有唯一的参数。每个参数都有一个名称和一个值。这些参数用于创建连接的条件

```
公共抽象类参数{
  公共参数（字符串名称）{
    参数名称=名称；}

  公共字符串参数名称；}

public class BoolParameter : Parameter
{
  public BoolParameter(string name, bool defaultValue) : base(name)
  {
    BoolValue = defaultValue;
  }

  公共bool值；}

public class IntParameter : Parameter
{
  public IntParameter(string name, int defalutValue) : base(name)
  {
    IntValue = defalutValue;
  }

  公共intIntValue；
```

9.7 设置Param操作

参数类定义参数名称和值。每个对话节点都有一个0或多个参数操作的列表-当播放这个对话节点（语音节点）或由用户选择（选项节点）时，应该执行参数名称定义的参数应该设置为指定的值。

公共抽象类设置参数动作{

```
  公共单位eParamaction类型{
    Int,
    弯曲件
  }

  public abstract eParamActionType ParamActionType { get; }

  公共字符串参数名称；}

public class SetIntParamAction : SetParamAction
{
  public override eParamActionType ParamActionType { get { return
     eParamActionType.Int; } }

  公共int值

public class SetBoolParamAction : SetParamAction
{
  public override eParamActionType ParamActionType { get { return
     eParamActionType.Bool; } }

  公共票据价值；
```