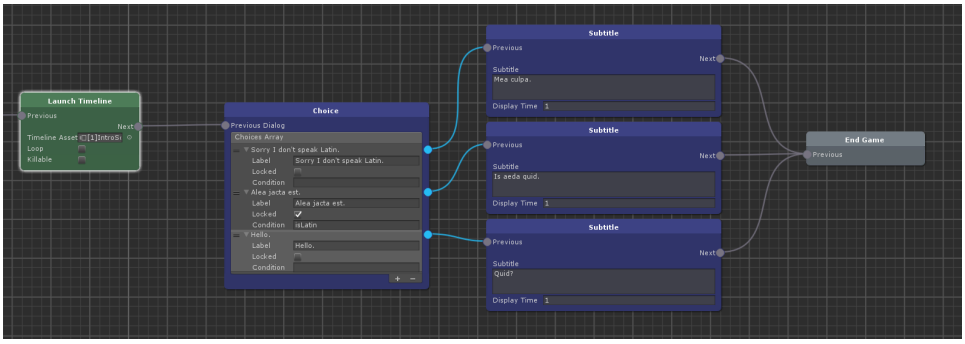
What have we done to each other? - Technical document

Florian Alazais - Emeline Berenguier

List of technical problems of the project

- Third-person camera controller
- Shader of the memory zone (displays objects depending on the distance and radius of the memory zone).
- Various situations possible for objects (exist only in a precise timeline, have two different versions of themselves depending on the timeline, can be manipulated, trigger cinematics, have special behaviors...)
- Cinematics and dialog nodal tools for the game designers, compatible with Cinemachine.
- QTE manipulation
- UI in world space : must be in front of everything except the character. Using a stencil test and a ghost version of the character to fix it.
- Wwise events on the inspector. (Lost when the project is updated).

Description of a major technical problem - Nodal editor cinematic tool



Context of creation

What have we done to each other was defined as a narrative game with cinematics and exploration phases. The creation of the cinematics was an important part of the development and need to implement these features :

- Subtitles
- Choices, branching dialogs, condition-locked choices depending on choices or in-game actions
- Multiple cinematics that can be launched from everywhere
- Launch sounds and trigger animations
- Manipulate cameras to create any plan

Two choices were possible at time : Timelines with *Cinemachine* or a custom tool for game designers.

Comparison between a custom tool and Cinemachine

Aa Criterias	Custom Tool	Cinemachine
Library	xNode	Unity
Cameras	**Must be done / coded at hand**	**Systems of timelines to control cameras**
Trigger systems	**Specific nodes with parameters chosen by the user. One node for type of signal. Infinity parameters of any type.**	**Signal events put at hand in timelines to trigger code functions. One signal per parameter. Functions called can only have one parameter of primitive type.**
Debugging	**Each node contains custom code. xNode base code.**	**Impossible. Library too big, random errors can appear.**
Workflow	**Straight logic (begin, start cinematic, play sound, add choice...). Nodes easily created and filled.**	**First, create of the cameras' plans. Then add triggers and events. One channel for each action (enable one specific object, send one signal to code...)**
Precision	**Using wait nodes to trigger subtitles or voices to fit the cinematic.**	**Voices and animations are triggered at the exact timing.**

Aa Criterias	Custom Tool	Cinemachine
Team workflow	**One person can use the dialog asset, no merge possible. The dialog asset is only needed to work.**	**Need the scene to work, nobody can use the scene when timelines are done. Add new cinemachine objects to the scene.**
Condition-based cinematics	**Supported**	**Unsupported**
Untitled		

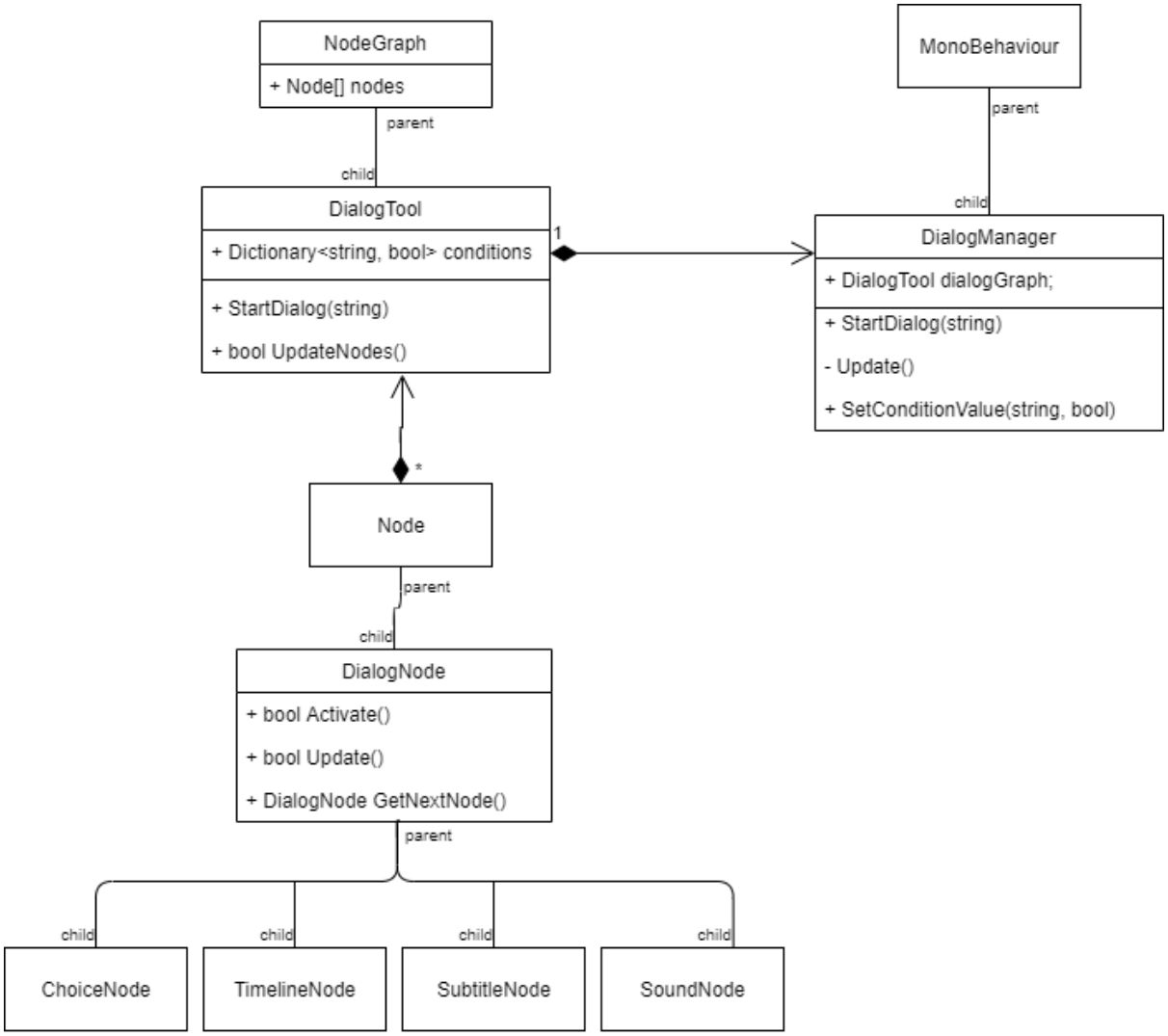
The previous comparison leads to this conclusion : a custom tool is easier to use but doesn't support a camera system and a timeline view to trigger events at the perfect timing.

However, *Cinemachine* doesn't support condition-based cinematics which is one of the pillars of the game.

We split the cinematics into two parts :

- **Straight cinematics**. These cinematics are played from start to end without intervention of the player or game event. They just need to be launched from anywhere. Managed by Cinemachine only.
- **Interactive cinematics.** These cinematics need input from the player. It can be cinematics during QTE or choice dialogs. Managed with custom tools and created with Cinemachine.

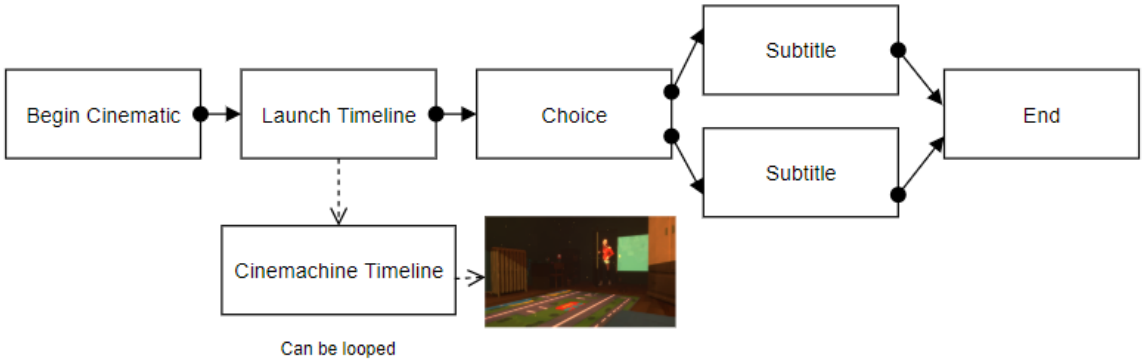
Functioning of the tool



UML schema of the cinematic Tool

The cinematic tool is a visual-nodal editor done with the xNode library (https://github.com/Siccity/xNode) for Game Designers. The cinematic starts with a begin node (which you can refer by its name to launch the wanted dialog), pass through nodes to launch dialogs or timelines and ends at an end node. The tool can only support interactive cinematics.

This is a macro schema of the process of an interactive cinematic.



- **Nodes and NodeGraph**. Nodes and NodeGraph are classes from xNode. NodeGraph contains all the nodes created. We use this class to access and manage the nodes. Each node has 3 override functions : *Activate()* to execute the node's code, *Update()* to wait for a player's input and *GetNextNode()* to return the next node. The dialogTool has an Update function to call the current node functions.

```
public bool UpdateNodes()
    {
        if (currentNode.Update())
        {
            currentNode = currentNode.GetNextNode();
            if (currentNode != null)
            {
                currentNode.Activate();
            }
            else
            {
                return true;
            }
        }

        return false;
    }
```

  If the node only needs to execute a script, the Activate function is called and the Update function is always true, so the DialogTool can get the next node.

  However, if the node is waiting for an input, the Update will return false and the DialogTool can't go to the next node.

  When the cinematic is done, the Update function of the DialogTool returns true and the DialogManager ends the cinematic.

- **DialogManager.** DialogManager is the link between the xNode classes and the project. xNode classes don't inherit from MonoBehaviour so their Update() methods can't run on their own. It's its Update() which runs the functions. Depending on the state of the game, the DialogManager can start, stop and run a cinematic.
- **TimelineManager**. The cinematic tool is not used to create the camera sequences. These sequences are created in Cinemachine and saved as a TimelineAsset. The TimeAssets can only be played and stop by a PlayableDirector from Cinemachine. The TimelineManger uses its PlayableDirector reference to launch the wanted Timeline.

Comparison between goals and results

Aa Criterias	Goals	Results
Cameras	Game Designers can do any cinematic plan without depending on programming.	**Game Designers have developed plans by themselves.**
Trigger systems	Easy way to script events from nodes with personalization.	**Straight cinematics need more triggers than excepted. The nodal trigger system was less used than Cinemachine.**
Debugging	Have a bug-free system and easy debugging cases.	**Bugs from Cinemachine and xNode. xNode has bugs, especially in build mode.**
Workflow	Game Designers work on their own, new features needed can be easily implemented.	**The nodal editor was easy to use and Cinemachine was very complete. Everything can be done alone.**
Precision	Events are triggered at a perfect timing.	**Events are triggerred at a perfect timing.**
Team workflow	Creation of cinematics doesn't affect someone else's work.	**Using of Cinemachine blocks an Unity scene. However, update events in a timeline or nodes only block the modified object.**
Condition-based cinematics	Possibility to create conditions and branching system.	**Nodal editor allows condition-based cinematics creation.**
Untitled		

Analysis of the results

The using of both Cinemachine and custom tool leads to different results:

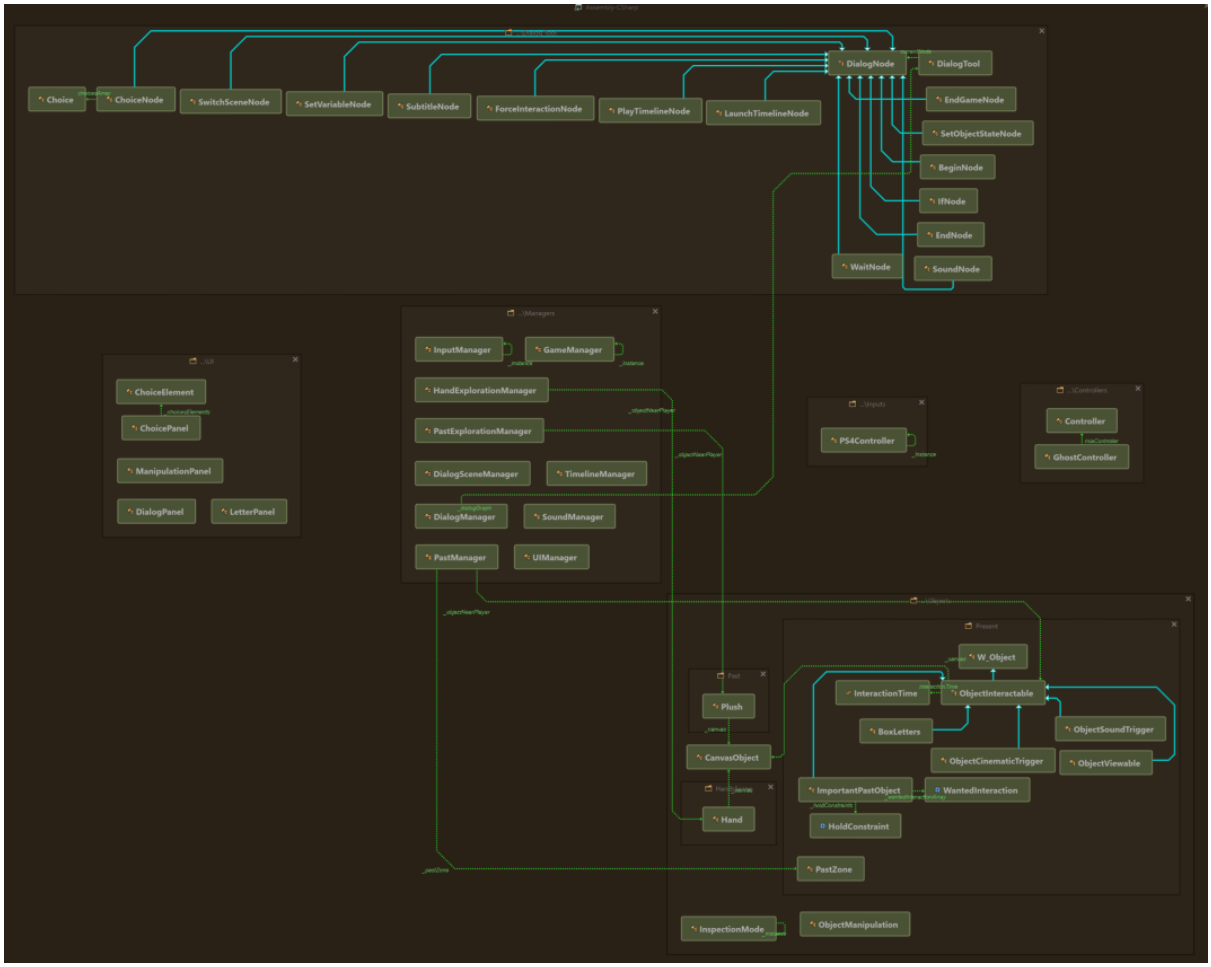
- **Good points:**
  - Every cinematic has to be done without technical problems
  - Tools easy to use, no appropriation time needed
- **Bad points :**
  - Bugs issues. Nodes are broken in build. Error from xNode, can't be fixed. Happen a few times so it wasn't a blocking bug. Cinemachine had some issues with parameters.
  - The synchronization between tool and Cinemachine. Timelines are played in parallel of the tool workflow, so small freeze moments at the end of the cinematic can happen if there is a desynch between the tool and cinematics.

It appears most of the problems comes from the utilization of both Cinemachine and a custom tool. Even if these two tools did their work, they created issues.

The best solution was to use only one tool : make a script system to create interactive cinematics with Cinemachine or to recode a Cinemachine system for our custom tool.

Creating timelines with Cinemachine was efficient but a handicap for team workflow (can't use Scenes, complicated timelines...) so we're thinking only develop a custom tool was the best way to implement cinematic.

Architecture of the game


Choice
ChoiceNode
SwitchSceneNode
SetVariableNode
SubtitleNode
ForceInteractionNode
PlayTimelineNode
LaunchTimelineNode
DialogNode
DialogTool
EndGameNode
SetObjectStateNode
BeginNode
IfNode
EndNode
WaitNode
SoundNode
ChoiceElement
ChoicePanel
ManipulationPanel
DialogPanel
LetterPanel
InputManager
GameManager
HandExplorationManager
PastExplorationManager
DialogSceneManager
TimelineManager
DialogManager
SoundManager
PastManager
UIManager
PS4Controller
Controller
GhostController
Plush
CanvasObject
Hand
W_Object
InteractionTime
ObjectInteractable
BoxLetters
ObjectSoundTrigger
ObjectCinematicTrigger
ObjectViewable
ImportantPastObject
WantedInteraction
HoldConstraint
PastZone
InspectionMode
ObjectManipulation