Gamelab II Report - Interactive Scenarios

1st Konstantin Passig
Games Engineering Bachelor 4th Semester
Julius-Maximilian-University
Würzburg, Germany
konstantin.passig@stud-mail.uni-wuerzburg.de

2nd Maximilian Sander
Games Engineering Bachelor 4th Semester
Julius-Maximilian-University
Würzburg, Germany
maximilian.sander@stud-mail.uni-wuerzburg.de

***Abstract*—Interactive Scenarios is a plugin for Unreal Engine 5.1 developed over the span of a year. It features a full blueprint editor to create quests using a visual state machine editor with dynamic blueprint function rebinding. This paper gives an overview about the design, creation and inner workings of the plugin and discusses the implemented features at the end.**

I. Task

In the third and fourth semester as Games Engineering students we were tasked to create a plugin for a game engine. The project where the plugin should fit into was provided by stakeholders with an existing project. The assigned project is ”The Circle”, a virtual reality game for architecture simulations. Users should pick a scenario to experience in their planned architectural object before construction begins. The scenarios are designed for the users to notice problems within the design of their homes before the building phase begins, avoiding expensive changes. It should also get the users to think about scenarios which might not be obvious while planning, such as moving in a wheel chair, not being able to lift ones legs high or having a baby. With an overview of the task we started working on formulating requirements and designing a basic structure of the project.

II. Requirements

Thus we were tasked to build a plugin allowing for easy development of short and simple quests. After some initial prototyping, we worked out the following requirements:

A. Blueprint accessibility

As the team behind the circle has only one programmer which is versed in C++, Unreal Engines programming language, we deemed it necessary to include a blueprint editor which made the plugin more accessible to all members of the team who are creating and maintaining assets.

B. Clean User Interfaces

As most people in our target audience are already familiar with the layout of the unreal engine editor we decided to mimic the default engine layout as closely as possible with our editor. The goal was to make the plugin look like it comes with a default unreal distribution.

C. Performance

As ”The Circle” is a virtual reality project our plugin needs to have a minimal performance impact. VR projects often have performance targets of $11ms$ or $8.3ms$ which is equivalent to a 90Hz or 120Hz refresh rate.

D. No deeply nested drop downs

The search function of the details panel [7] in unreal engine 5 is bad. This forces the user to click through all nested menus when they just want to quickly change a variable. This is not desirable for a smooth questmaking experience.

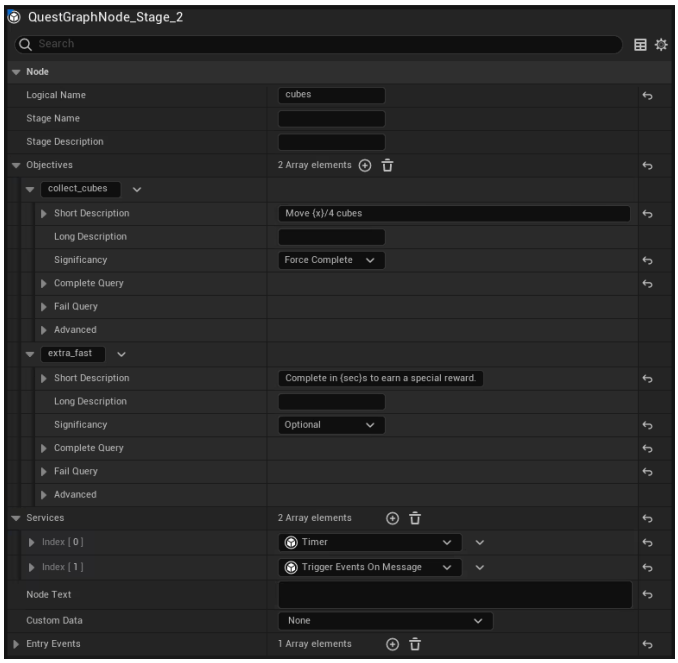

Fig. 1. Example of deeply nested values

E. Targeted platforms

for use in a VR project we made the plugin run on both common platforms: Windows and Android. Windows for PC-VR and development and Android for most standalone headsets.

III. Design of the plugin

In order to keep the plugin lightweight, the plugin is split into an editor and a runtime module.

A. Runtime module

The runtime module of the plugin contains all classes which are needed in order to run projects made using the plugin. All code which is present during runtime is stored in this module.

B. Editor module

The editor module contains all files of the plugin which are needed in order to edit and compile quests in the unreal editor. It is not included in builds of the game.

IV. Market research

For our market research we first started by classifying the different scenarios and worked out what kind of features a quest system should have to accomplish those tasks. Then we went to the epic games store and investigated different plugins. We found three main approaches:

A. Spreadsheet

Systems which are based on spread sheets are often implemented using unreal engines data tables [1]. Data tables in unreal engine are nothing more than an array of structs. When accessing a property inside of a struct it generates a dropdown. This can lead to very deeply nested values which is the main reason for us to not use this approach.

Pros:

- Easy portability with csv as file standard.
- Quick implementation as plugin as most of the code is already in the engine.

Cons:

- Relations between nodes are hardcoded and there are no functions which can be defined on a per edge or node basis.
- Deeply nested menus because of the data types.

B. Modified behaviour trees

One plugin which had quite impressive functionality is the "Fullstag Quest System [2]". It is a system which is based on the behaviour tree [3]. Behaviour trees execute from top to bottom and from left to right.

Cons:

- Layout of a behaviour tree is only from top to bottom.
- Execution order is dependent on the layout of the nodes in the editor.
- Deep nesting of menus in the detail pannel which makes it a hassle to find the right data.
- Does not use the capability of the engine. All services for the edges and nodes are custom built for the plugin.

Pros:

- Visual representation of quests nodes and edges

C. Primitive graphs

Some plugins used a graph oriented setup where the user can configure all properties of the graph via the detail panel [7]. One such plugin is the "Branching Quest & Dialogue System" [6] [11].

Cons:

- Uses Data Assets [9] for data storage and transition rules which has the problem of deeply nested menus
- The graph is only used as visual representation to connect the steps of the quest

Pros:

- More intuitive than the other two systems
- Delegates and overrides which can be bound to are intuitive
- Good layout of the data

D. Conclusion

With the knowledge about what other plugins are doing we implemented some minor adjustments to our requirements and started working on a first prototype.

V. First prototype

For the first prototype we developed a blueprint function library which could make any blueprint into a quest. We wanted to use the structure of blueprints for the visual representation of quests. This system proved to be usable but not where the plugin should head in terms of usability.

VI. Research for existing code bases

After the first prototype we discussed with one of our professors at university how quest systems are usually implemented. The answer was graphs. So we started investigating into graphs and how to implement them into unreal engine. A few sources suggested creating them from scratch but this would include the editor visual editor we put ourselves into the requirements. So after a quick prototype this was out of the question due to missing documentation for Unreal Engine. Then we discovered the plugin template GenericGraph by jinyuliao [4]. It is licensed under MIT license so it was free for us to use and modify.

A. What GenericGraph provides

The plugin features a node based editor which is able to manipulate the location of nodes on a grid. The graphs do not have any functionality. The added nodes were exactly what we were searching and look like in figure 2. It is only an empty data structure with its editor to interact with the nodes and edges.

VII. Implementation

A quest is described by a state machine, which in turn is represented by a graph ("Quest Flow") in our editor. The quest system ("Quest Subsystem") is designed to handle messages sent by any actor, propagating them to the running quests that have subscribed to them. These messages come in the form of gameplay tags [8] that describe events that happen in the world, which the quests use to determine whether to transition into a new phase. The Quest Subsystem can safely be informed about any events happening in the world, even if no related quest is active.

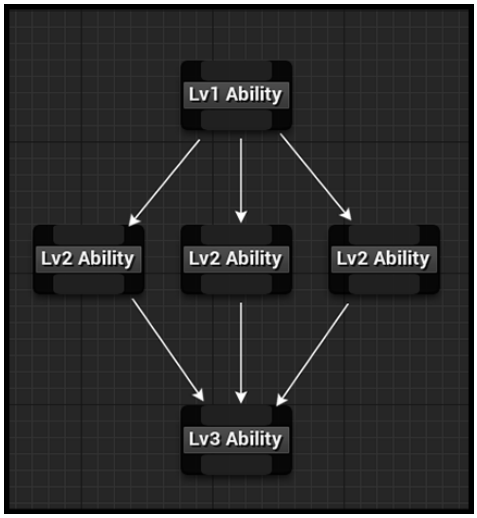

Fig. 2. Node system added by GenericGraph Plugin

A. Quest Subsystem

When a quest is started, the Quest Subsystem notes down all the tags that the quest wants to be informed about and dispatches an event to inform the world about the newly created quest. Upon receiving a tag and passing it on to interested quests, the Subsystem receives information on the effect of the message. It then dispatches an event if a quest either reached a new state or was terminated.

B. Quest Flow

Nodes in the Quest Flow graph represent the different phases of a quest, while the directional edges between them describe the conditions needed to progress into the next phase. Each node has the option to bind custom functions to it being entered and exited. The edges require a custom transition function and provide the option for dynamic description and progress indication functions to be used in user interfaces.
The usual course of a Quest Flow is as follows:
The Quest Flow starts by entering its designated entry node. As with any other node, its ”on entered” function is executed if bound. Whenever a tag is received by the quest, the transition functions of the outgoing edges from the current node are evaluated. If one of them returns true, the current nodes ”on exit” function is executed and the node following the respective edge is entered. When a node with no outgoing edges is reached, the quest is considered finished. Upon finishing, the quest runs its own finishing functions and subsequently gets destroyed by the Quest Subsystem.

C. Edges and Transition Functions

Each edge describes a transition condition in the state machine with its transition function. To save on performance, these functions are only evaluated when events relevant to the quest happen. A chronological list of messages that have been received since the activation of the edges parent node is passed into the transition function. This list can be evaluated in a number of ways, ranging from simply counting messages of a type to interacting with the world to gather more information on the most recent event.
Transition functions can also be used to affect the world, allowing dynamic reactions to events happening during a quest.

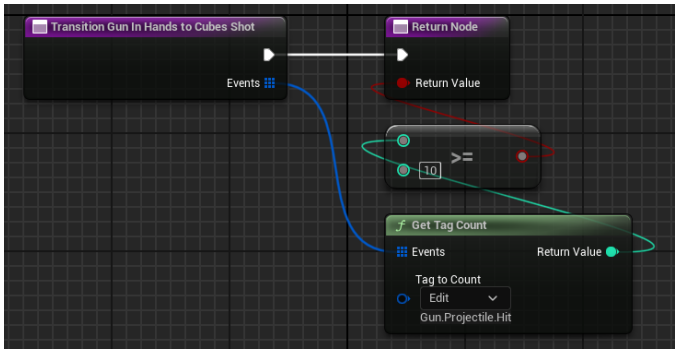

Fig. 3. Transition function

D. User Interfaces

While the plugin does not supply any UI widgets natively, it contains all the parts needed to sustain a fully automated UI containing all quests and their progression. An example of one is contained in the demo scene that comes with the plugin. While the Quest Subsystem dispatches events informing about any major changes to active quests, the edges can supply descriptions and percentage-based progress indication at any time based on data stored in the quest.

E. Blueprint Editor

The plugin features a custom blueprint editor to edit quests. Much like the widget editor, it is split into two tabs: one for editing the Quest Flow, and one for normal blueprint function editing. The Quest Flow Editor allows placing entry, exit and normal nodes. An entry node marks the unique entry point into the Quest Flow and is functionally the same as a normal node. An exit node is a normal node with reduced functionality, not allowing for outgoing edges or an exit function to be set. it is mostly meant to be a visual aid for developers.

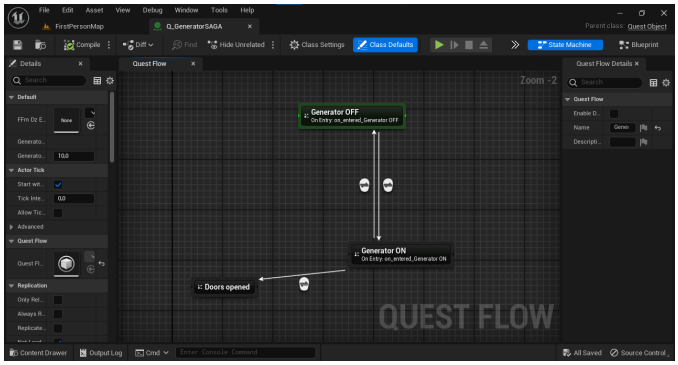

Fig. 4. A quest inside the quest flow editor

VIII. Evaluation

The plugin itself is fully functional and ready to use. It has some areas where further improvements are needed but those are mostly quality of life improvements. In hindsight the plugin is a bit too over engineered for the needs of our stake holders but not in a bad way.

A. User Interaction

During research the problem of deeply nested is mentioned every time the plugins in the market research use some kind of blueprint struct. Interactive Scenarios does not have any nested options but instead uses the default layout of the blueprint editor. This gives the user the freedom to keep track of their data as they want. The editor is contained in one single editor window so there is no switching between windows which increases the comfortability while using the plugin.

B. Blueprint accessibility

The custom editor enables users of all skill levels to work quick and efficiently with the plugin.

C. Targeted platforms

Android and Windows are fully supported with the plugin.

D. Performance

The event system where updates are sent to has the advantage that tags can get sent all the time even if there are no quests running. So updating quests works in a fire and forget which could update a quest can just tell the quest system in a fire and forget manner that something happened and move on as tags which are not used in any quests get voided by the quest subsystem.

IX. Future Improvements

A. Multiplayer

The plugin is designed for singleplayer use only. This means that if any project would want to create a multiplayer game the quest system would not be sufficient. It is mainly missing a concept for where the data is stored and how the data will get distributed to the clients.

B. Rewrite of Quest update map

The way that the Quest Subsystem messages quests could be rewritten. Currently, it subscribes the quest to all tags that all edges in the quest want to be informed of. an improved version would subscribe the individual edges in the quest to only the tags that they need, reducing unnecessary function calls. another improvement would be for edges to have the option to be updated on every game tick if that is deemed necessary.

C. Function selector improvements

The custom assignable functions can be created and opened from the context menu of the nodes and edges, while in the details panel of the nodes and edges they can be swapped out using a dropdown menu. Being able to create and edit them from the details panel would be a major improvement to the quest creation process

D. Automatic prompting for function renaming after creation

When a function is created for a node or edge, it is automatically named based on the nodes title and its role. keeping those names can get quite confusing, so the user being prompted to rename the function after creation would also improve the user experience.

E. Save and load functionality, level persistence

The current state of progression in a quest can not be saved or kept across level transitions. Implemementing this is necessary to use this plugin for projects with longer playtime.

References

[1] Epic Games, "Data Driven Gameplay Elements In Unreal Engine," Unreal Engine 5.1 Documentation, 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/data-driven-gameplay-elements-in-unreal-engine?application_version=5.1. [Accessed: Jul. 15, 2024].

[2] Epic Games, "FullStag Quest Framework," Unreal Engine Marketplace, 2024. [Online]. Available: https://marketplace-website-node-launcher-prod.ol.epicgames.com/ue/marketplace/en-US/product/fullstag-quest-framework. [Accessed: Jul. 15, 2024].

[3] Epic Games, "Behavior Trees in Unreal Engine," Unreal Engine 5.1 Documentation, 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/behavior-trees-in-unreal-engine. [Accessed: Jul. 15, 2024].

[4] J. Liao, "GenericGraph," GitHub repository, 2024. [Online]. Available: https://github.com/jinyuliao/GenericGraph/tree/master. [Accessed: Jul. 15, 2024].

[5] Epic Games, "Widget Blueprints in UMG for Unreal Engine," Unreal Engine 5.1 Documentation, 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/widget-blueprints-in-umg-for-unreal-engine?application_version=5.1. [Accessed: Jul. 15, 2024].

[6] Epic Games, "Branching Quest & Dialogue System," Unreal Engine Marketplace, 2024. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/branching-quest-dialogue-system. [Accessed: Jul. 15, 2024].

[7] Epic Games, "Level Editor Details Panel in Unreal Engine," Unreal Engine 5.1 Documentation, 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/level-editor-details-panel-in-unreal-engine?application_version=5.1. [Accessed: Jul. 15, 2024].

[8] Epic Games, "Using Gameplay Tags in Unreal Engine," Unreal Engine 5.1 Documentation, 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/using-gameplay-tags-in-unreal-engine?application_version=5.1. [Accessed: Jul. 16, 2024].

[9] Epic Games, "Data Assets in Unreal Engine," Unreal Engine 5.1 Documentation, 2024. [Online]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/data-assets-in-unreal-engine?application_version=5.1. [Accessed: Jul. 16, 2024].

[10] Epic Games, "Quest Editor Plugin," Unreal Engine Marketplace, 2024. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/quest-editor-plugin. [Accessed: Jul. 17, 2024].

[11] Epic Games, "Quest Editor Plugin," Unreal Engine Marketplace, 2024. [Online]. Available: https://www.unrealengine.com/marketplace/en-US/product/quest-editor-plugin. [Accessed: Jul. 15, 2024].