Exploring Alternatives with Unreal Engine’s Blueprints Visual Scripting System

ABSTRACT
We present BPAlt – a system which allows game developers to create and manage alternatives for Unreal Engine’s Blueprints Visual Scripting System. BPAlt allows the user to create, save, organize and swap Blueprint alternatives for rapid testing and experimentation. BPAlt also supports level alternatives, alternatives of alternatives through alternative scenarios, and allows merging of alternatives. We conducted a preliminary user study with 10 moderately skilled participants where we compared the usability, usefulness and creativity support of BPAlt to Unreal Engine alone for prototyping alternatives of game objects and mechanics in four different games. The findings suggest that the support for alternatives with the main features of BPAlt is beneficial in the game developers’ workflow as it supports exploration particularly well.

CCS CONCEPTS
•Human-centered computing → Graphical user interfaces; Software and its engineering → Visual languages

Author Keywords
Alternatives, Visual Programming, Creativity Support, Unreal Engine, Blueprints, Game Development

INTRODUCTION
Due to its efficiency and flexibility, C++ is among the most popular languages used in game development – an industry which often pushes the hardware limits to deliver quality products to stand out in fierce competition. However, recently visual programming has been gaining popularity in modern game engines. Recent examples include visual scripting plugins for the popular Unity Engine [45] such as FlowCanvas [46], Playmaker [47], Bolt [48] and Amplify Shader Editor [49]. Initially these efforts have been directed to support rapid prototyping, but this is changing.

Unreal Engine (UE) [50] is a suite of integrated tools for game developers to design and build games, simulations, and visualizations. It is one of most widely used game engines among both hobbyists and professionals. The engine features Blueprint Visual Scripting [51] or simply Blueprints, which was created to support the workflow of designers and artists by enabling the full range of concepts and tools generally only available to programmers. Blueprints is a fully functional object-oriented visual programming system which is mainly used to create gameplay elements by defining classes and objects. Until recently a typical professional game studio which uses UE would employ a combination of C++ and Blueprints in their workflow. Traditionally, Blueprints were used by designers for rapid prototyping of game mechanics or for tasks like creating and positioning elements in a widget and C++ was used for developing the final product efficiently. Starting with version 4.15, the support for Blueprint Nativization [52] was added to reduce virtual machine overhead in the runtime by generating native C++ code from Blueprints. This process was first used successfully during the development of Epic Games’ Robo Recall [53]. As a result, the system is now suitable not only for prototyping but also for creating the final commercial products thus making game development truly accessible to those with limited coding skills.

Despite this, Blueprints and similar visual scripting systems do not support well the exploration of alternative ideas, such as, e.g., exploring multiple variants of a game mechanic, game objects, different scenarios that can be seen in the game, or multiple combinations of game mechanics. We define an alternative as a potential solution to a given problem that can be compared to other potential solutions. In the context of this work alternatives are defined as gameplay classes that can be potentially used.

Alternatives are an integral part of conceptual design [44]. In creative fields, experts typically generate sets of alternative solutions when solving ill-defined problems [36]. This has been shown to result in higher quality outcomes [7]. This can be seen in the workflow of architects [1,30], web designers [33], and software engineers [38] who generate sketches of designs as potential solutions before deciding on the final choice. These sketches help to externalize knowledge, better understand the problem, and explore a space of potential solutions [2]. Evidence has been found that design alternatives also improve exploration of the design space in various creative fields (see e.g., [28,32]).
including alternatives for visual programming environments (see e.g., [42–44]).

In software development, iteration is used when working on a project: be it larger iterations or smaller changes [21]. Game development is a creative field that also utilizes a development cycle similar to software development, where developers and artists iterate through alternative ideas. In game development the iteration process is applied during almost every aspect of design: from the initial conception through the final quality assurance testing [12]. Research in prototyping for game development has been previously done through user testing and iterating based on data and user feedback (see e.g., [6,9]). However, creating prototypes on a micro scale for an individual or a team remains a neglected area of research.

To fill this gap, we developed BPAlt (Blueprint Alternatives) – an extension for UE, to enable exploration of alternatives. Our system introduces methods for swapping between Blueprint alternatives, which allows for their rapid testing, experimentation, streamlined saving and organization. To demonstrate the usability, usefulness and creativity support of BPAlt, we conducted a comparative user study with moderately skilled participants who were tasked to prototype alternative game objects and mechanics in four different games.

RELATED WORK

Prototyping in Game Development

Two approaches to improving the quality of video games can be identified: improving the product directly or improving the development process. To track and measure this, often game analytics are applied in the context of game development and game user research. These analytics are directed at both: the analysis of the game as a product (e.g., whether it provides a good user experience) and as a project (e.g., the process of developing the game) [43]. Measuring the progress of improvement is traditionally done in iterations with a release of a build of the game.

Improvements to the iteration process of game developers between each user test have been partially addressed through the use of game engines such as Unity and UE, which enable environments to develop and test products more efficiently. However, we argue that the process of developers creating iterations of their own work has still room for improvement. Namely, we believe that the developers’ iterative workflow can be particularly improved through the use of alternatives.

Earlier Work on UIs for Alternatives

The work on alternatives can be traced investigating creative needs in UIs for experimentation, exploration and evaluation of alternatives by Terry et al. [39]. In this work, Terry et al. proposed the design horizon – a view to complement the “normal” document window for users to place snapshots of their work to support the creation of alternatives. Subsequently, Terry et al. [40] presented Parallel Paths, a model of interaction that facilitates generating, manipulating, and comparing alternative solutions. The model was implemented in Parallel Pies, a UI mechanism for image manipulation, which allowed for creation of alternatives; embedding of the alternatives in the same workspace; manipulation and side-by-side comparisons. Lunzer and colleagues introduced alternatives in a variety of applications: comparing queries over a multi-attribute dataset [23,24], gathering and comparing results from alternative resources that offer nominally the same processing [25,26], exploratory access to online resources [11,22], exploratory e-learning [18] and for information access, real-time simulation, and document design [27]. Design Galleries by Marks et al. [28] was an early work on representing multiple alternatives in a single view by automatically generating and organizing alternatives of 3D graphics or animations allowing a designer to consider alternatives through navigation of the solution space.

Alternatives for Parametric and Node-Based Design Tools

Parametric Computer Aided Design (CAD) is a domain where the use of alternatives has experienced growing popularity in recent years as evidenced below. CAMBRIA [20] is a multi-state design tool for simultaneously managing multiple 2D vector graphics alternative design models which can be explored in parallel. Woodbury et al. [41] introduced a prototype gallery system on a web browser, which supports saving alternatives from three graph-based parametric modeling tools where users can retrieve alternatives from the gallery, share them with others, and combine them to generate more alternatives. Mohiuddin et al. [31] presented an online gallery system for design alternatives in parametric modeling, which supports multiple commercially available parametric modelers. Cristie and Joyce [5] introduced a workflow plugin for Grasshopper [35] which enables parametric structures to be tracked in a similar way to Git [54] with branching support and allowing users to save the current parametric design state onto the cloud. The recorded options and data are visualized in a graph. Matejka et al. [29] presented DreamLens – an interactive visual analysis tool for exploring and visualizing large-scale generative design datasets. The system automatically generates alternatives within the given design criteria constraints. GEM-NI [42,44] – is a node-based generative design tool which enables the user to quickly generate sets of alternative solutions through branching, merging, Cartesian products, and history recall. It also allows users to edit alternatives in parallel with undo capabilities and support for multiple displays. The MACE extension [43] to GEM-NI also enables interactive comparisons of more than two alternatives using active and subtractive encodings for difference visualizations. The concept of subjunctive nodes has been introduced in Shiro [13] – a declarative language which allows a parametric system to represent multiple alternatives in a single system definition. In Shiro, subjunctive nodes—nodes with multiple possible outcomes [27]—allow expressing both alternative values for properties and alternative computations for specifying parametric systems containing alternatives thus providing a multi-state document model [13]. Elkhaldi and
Woodbury [8] introduced Alt.Text – a node-based tool for creating text documents, which supports tasks for creating alternatives through a hierarchical multi-state document model. It also offers a subjunctive user interface to support parallel editing and viewing of alternatives.

BPAlt is an extension to Unreal Blueprints, which is a node-based system. Our support for alternatives builds upon previous work [8,13,44] on alternatives for node-based systems. In BPAlt, we implement a number of interaction techniques for alternatives in node-based systems for use in visual game development.

Alternatives in Other Contexts
Juxtapose [16] presents a parallel code editor and runtime parameter environment for designing multiple alternatives of application logic and interface parameters which uses hardware board sliders. Bueno et al. [3] evaluated the idea of rewriting history to manage alternatives and explorations of a design and found that users understand the approach and would like to use it in their own creative work. Hailpern et al. [14] evaluated Team Storm – a system which allows to work with multiple design ideas collaboratively and in parallel. Hailpern et al. found that design teams can effectively utilize the system to create, organize, and share multiple design ideas during creative group work. Smith et al. [37] evaluated computational sketching tools by comparing three interaction models for working with alternatives in early design stages: a tab interface, a layered canvas, and spatial maps. Spatial maps were found to be used the most for reflection, analysis and decision because of the ability to compare designs side-by-side. Smith et al. concluded that tabs, spatial maps and layers are useful. d.note [15] is a revision tool for UIs expressed as control flow diagrams, which introduces a command set for modifying and annotating their appearance and behavior. d.note defines execution semantics allowing proposed changes to be tested immediately. Kazi et al. [19] presented DreamSketch – a 3D design interface for early stages of design where a user roughly defines the problem by sketching the design context and a generative design algorithm produces multiple alternative solutions that are augmented as 3D objects in the sketched context. The user can then navigate through these solutions. O’Leary et al. [34] presented Charrette – a system that allows designers to curate design iterations, attach meeting notes to the relevant content, and navigate sequences of design iterations with the associated notes to facilitate in-person discussions. O’Leary et al. found that using the system correlates with increased confidence and recall in discussing previous design decisions.

As it will be demonstrated, the concepts behind the early work on alternatives described above lay out the foundations of BPAlt, which also adapts the ideas introduced in Juxtapose [16] for use in the workflow of game developers.

Game developers work in teams and communicate design choices between team members and therefore supporting alternatives has a potential to also improve this collaborative aspect of developer’s workflow as found in previous work [14,34].

BPAlt

Figure 1: First version of the target movement in the event graph of the “Target” Blueprint.

The interface of BPAlt was designed to be minimally intrusive to the workflow of game developers who are already familiar with UE. Using BPAlt, the user creates and edits alternatives in a single-state document [39] but still can easily explore different ideas in parallel or individually through UE’s Level Editor.

In UE, objects that can be placed or spawned into the level are referred to as actors, which are made up of components that contain all the properties and functionality of the actor entity. Since UE works on an object-oriented system, every Blueprint class that is saved contains all the information of the Blueprint besides default values that can be edited on the actors placed or spawned into the level. This information includes components, component properties, variables, graphs and default properties.

To demonstrate the capabilities of BPAlt we will use the example in Figures 1-4.

Creating Alternatives
BPAlt creates alternatives from within a tab attached to each Blueprint Editor. When an alternative is saved it creates a copy of the Blueprint that is being edited. This copy is saved into a list of alternatives which we refer to as alternatives set. When creating the first alternative from a Blueprint, an alternatives set is created, which is a list of all the alternatives that are created from either the original or alternative Blueprint.

Imagine Tim, a game developer, is trying to create and test moving targets for a first-person shooter (FPS) game using Unreal Blueprints. Tim wants to create multiple types of targets to determine what versions he wants to use. After creating a base Target Blueprint class (Figure 1), which is just a white sphere that moves back and forth, he places four of these targets in the level, see Figure 3 (top).
Tim then uses BPAlt to create an alternative Blueprint of the Target. To do this, Tim navigates to the menu bar in the Blueprint Editor to open the Blueprint Alternatives tab (Figure 2a) via the Window menu (Figure 2b). The Blueprint Alternatives tab opens at the bottom of the screen (Figure 2e) but can be undocked and moved around. Tim then saves an alternative of the currently opened Target Blueprint via the “Save Alternative” button (Figure 2c), which will add a new alternative to the alternatives set, which appears in the list of created alternatives in the Blueprint Alternatives tab (Figure 2d), from which he can open or delete any of the alternatives.

By following this procedure Tim creates an alternative where the target’s color is changed to red and the path is changed to move along a sinusoidal curve, see Figure 3 (bottom).

Creating an alternative of a Blueprint class, saves a copy of all the data within a given Blueprint (relative transform, variables, components, graphs, functions, etc.). The process is similar to duplication. However, unlike duplication the process of swapping and testing alternatives is much more streamlined using BPAlt.

**Swapping Alternative Blueprints**

Swapping between alternatives for testing is the core feature that distinguishes BPAlt’s approach from simple duplication. BPAlt streamlines the process of testing between different Blueprint alternatives to improve the workflow of developers. To swap between alternatives Tim selects an instance of the target Blueprint actor in the level editor through the world outliner (Figure 3a), which has a set of alternatives associated with it. He then switches between the target Blueprint alternatives by going into the Details panel of the selected target and, under the Blueprint Alternatives section, he swaps alternatives by using the drop-down menu containing all alternative Blueprints in the set (Figure 3b). Once a selection is made to the new target alternative the actor is replaced with a new actor, which uses the Blueprint class of the newly selected target alternative (Figure 3c). The actor’s original world transform is preserved.

In Tim’s case, he placed four targets in the level to test them side-by-side. After he creates a few alternatives that he wants to test out, he can go into the level and swap between Target Blueprints individually via the Details panel (Figure 3b).

The next example shows another way of swapping between alternatives. Imagine Jen, a game designer, is tasked with designing a special power which players can activate in a Tetris-like game. A programmer on her team sets up a Blueprint class called Special Power which contains the event node (Figure 4c) to allow her to create functionality for the special power. Jen starts by making the special power clear all the blocks in the bottom row of the grid. Jen decides to create another iteration of the special power by using BPAlt to create another alternative of the special power Blueprint class. Jen changes the functionality of the
alternative Blueprint to clear two bottom lines instead. To test her new alternative Jen presses the “Play” button (Figure 4b) in the alternative’s Blueprint Editor, which swaps out the special power actor in the level. This method will swap out all instances in the level and is optional. In Jen’s case, it is used for a Blueprint which controls the special power mechanic.

Figure 3: FPS Target game before (top) and after (bottom) swapping between Target Blueprint alternatives. a) Target Blueprint actors in the world outline. b) Drop-down menu to swap between alternatives. c) The second from the top Target is swapped for an alternative with modified color and path.

Figure 4: Special Power Blueprint of Tetris worked example: the original (top) and an alternative (bottom). a) Blueprint Editor tabs for the original and alternative Blueprints. b) Play button. c) Special power event node.

Selective Merging
Native to the Unreal Engine there is functionality to merge together Blueprint graphs which was designed to resolve conflicts when merging Blueprints using Unreal Engine’s native source control. However, this process of resolving conflicts is slow and tedious. In contrast, our selective merging is designed to overcome this problem and to be used more regularly.

Selective merging involves moving nodes from a graph in one Blueprint alternative to another. We will use the previous worked example with Jen in Figure 4 (top) to demonstrate how selective merge works. Consider that Jen creates a new alternative to the “SpecialPower” Blueprint called “SpecialPower_Alternative 1” and wants to merge over a portion of the Blueprint back over to the original Blueprint alternative (“SpecialPower”). Figure 5 presents the alternative to the SpecialPower Blueprint with some changes to the Blueprint graph. Jen does not want to merge all the different nodes, so she selects a specific node from the graph (Figure 5c) and opens the selective merge menu (Figure 5a). She then selects the alternative she wants to perform the selective merge on.

Figure 6 presents the result of the selective merge where the FORLOOP node was merged over and was connected properly through all three pins. Note that the system had to figure out the proper connection to make since there was no immediate connection to the left of the selected node (Figure 5c). There was another node, LogTEXT, in the way (Figure 5b).

There is an algorithm in place to find the nearest applicable connections for merged nodes in the target Blueprint if they exist. This can be seen in Figure 7, Figure 8 and Figure 9. In this example the same process of selecting nodes in the alternative and merging them over to another Blueprint is demonstrated. In this case the nodes from “Test_Alternative 1” (Figure 8) are merged to the target Blueprint “Test” (Figure 7). The “DrawBox” node was merged and placed accordingly. However, in this case two more features are demonstrated: merging over nodes which represent references to class variables (Figure 8b) and dealing with
nodes that already exist in the target Blueprint. Note that in Figure 7 and Figure 8 the “Test Blueprint” does not have any of the variables that “Test_Alternative 1” has (see Figure 8a). After the merge (Figure 9) all the variable nodes are placed in the Blueprint and associated variables are also added to the Blueprint. All the selected nodes that already exist in Test are untouched except for changes in the nodes such as default values of pins and new pins added to certain nodes. E.g., in Figure 7 and Figure 9, there is a change in the switch node adding an additional pin and, in the top “Set Actor Location” node changing the “Sweep” Boolean parameter. In case there are no valid nodes for the merged nodes to connect to, the nodes are still merged over, but remain unconnected.

Figure 5: An alternative of the SpecialPower Blueprint “Special_Alternative 1” was created and changes were made. a) Selective merge menu b) Log Text node which does not exist in the “Test” Blueprint c) Selected for-loop node.

Figure 7: Test Blueprint before selective merge.

Figure 8: Test_Alternative 1 Blueprint is selectively merging to the Test Blueprint. 11 nodes are selected. a) Blueprint variables. b) Variable reference nodes.

Figure 9: Test Blueprint after selective merge.

Level Alternatives & Alternative Scenarios
We also implemented support for level alternatives, which allow the developer to save the current state of a level in the context of Blueprint alternatives. The currently active alternatives are saved for all actors in the level, so that the developer can experiment with different variations of levels by mixing and matching alternatives from different Blueprints.

Alternative scenarios is a supplemental feature that we added to BPAlt, allowing the user to create and load alternative “scenarios” (alternatives of alternatives). Scenarios are a given level’s current usage of Blueprint alternatives saved so that it can be restored later. Alternative scenarios collect references to all the actors in each level that are instances of a Blueprint class and saves what alternative is being used for each Blueprint actor. Scenarios are saved and loaded using a menu that can be opened in the Level Editor. Users can save the current state of the level as a scenario. Upon loading a scenario, the actors in the scene will swap to the alternative Blueprints that were saved in the scenario.

USER STUDY
We designed a comparative user study to test if BPAlt improves the game developers’ workflow for creating alternatives. BPAlt was compared to the unenhanced version of UE 4.17.2, which we refer to simply as Unreal from now on.
Participants
10 paid participants (two females) were recruited from undergraduate and graduate students in technical programs. We targeted participants with experience using Unreal and other game engines. The backgrounds of the participants varied from game developers, programmers, and robotic engineers. The participants’ ages ranged from 20–33 years old ($M = 24.3, SD = 4.27$). All participants were experienced game developers ($M = 4.71, SD = 3.26$). Three participants frequently used Unreal, the remaining participants were Unity programmers. Eight participants used Unity regularly for 2-9 years ($M = 4.75, SD = 3.08$). Eight participants used Unreal regularly for 1-2 years. All the participants had experience using data comparison or differencing tools: six participants used them regularly (at least once a week). Nine participants used flowcharts and diagrams in their work, six participants used them regularly (at least once a month). The same nine participants had experience using visual programming systems to most common being Scratch and Unreal Blueprints.

Apparatus

Figure 10: The multi-monitor setup in the user study. a) Monitor for the task instructions, could be used by the participants. b) Monitor controlled by a separate computer to guide participants through the tasks. c) Primary monitors used for the tasks.

We used a workstation with 16 GB RAM, AMD Ryzen 7 1700 3.9GHz 8 Core(s), Nvidia GTX 1060 3GB with Microsoft Windows 10. The PC was connected to a five-monitor setup: three horizontal monitors on the bottom and two monitors on the top (Figure 10). The monitor in the top left displayed specific information on each task including special functionality of the template and what the controls were for the game for participants to play test. The participants were free to use the top left monitor as they desired. During the study we gave tutorials on each task going through the functionality and what the goals would be. The participants had control over 4 of 5 monitors. The top middle monitor was reserved for the investigator helping the participants to understand the tasks. The investigator used a separate laptop computer to show the participants how to use Unreal and BPAlt, test the level, navigate Blueprints, and demonstrate an example of how to use each task template. Pen and paper were provided if requested (which happened twice). Interviews were recorded, and questionnaires were filled out on a separate laptop computer. Screen capturing was used.

Procedure
We designed four tasks to cover multiple use cases for BPAlt. Participants were given premade Unreal game project templates which contained Blueprints that had to be edited. Each task was preceded by a tutorial where the corresponding template was thoroughly explained. The aspects of the template that had to be modified during the task were identified to participants. Participants were then asked to pick a single premade Blueprint class from a selection of 1 to 9 different Blueprints. The actual number varied depending on the task. Participants were asked to create three alternatives of the Blueprint and test them.

The user study was a $2 \times 2$ mixed factorial design. The independent within-subject variable was System (Unreal, BPAlt). Each system was evaluated with two different game types (Block and FPS). For each game type there were two game templates: Tetris (Block), Match3 (Block), Target (FPS), Obstacles & Enemies (FPS). The two vastly different game types enabled us to cover more use cases of the system thus increasing external validity of the findings. For Block games (Tetris and Match 3) the participants were working with Blueprints which controlled game events and only one game object was needed in the level per event (abstract game object). This was different for the FPS tasks in which the participants had to work with Blueprints that they had to have multiple copies of in the level. We did not want participants to re-use the same game template with the second system they tested so only one game template of each type was used with either system. This was done to minimize the learning effect and to provide more options for participants to express their creativity. Furthermore, System was counterbalanced. System Order (BPAlt first, Unreal first) was the independent between-subject variable. Game type order and template order were randomized. Creativity Support Index (CSI) is a quantitative psychometric survey which assesses how well a system assists creativity in the design process [4]. Specifically, participants provided ratings for six dimensions of creativity support: Enjoyment, Exploration, Expressiveness, Immersion, Results Worth Effort, and Collaboration. Collaboration was not rated. CSI score was the dependent variable.

Before each task the participants were guided through the template project to learn how the game is played and tested. The participants were shown which aspects they had to change. We asked the participants to come up with three different alternatives for a given game template with no time limit. These options included changing enemies, targets, obstacles for the FPS tasks and changing reactionary game play mechanics in the Block tasks.
premade Blueprint classes were available. See Figure 11a. The four project templates used for the tasks: a) Tetris, b) Match 3, c) Target, d) Obstacles & Enemies.

Tetris
Like in the Tetris worked example described above, in the Tetris task there were predefined event nodes for clearing different numbers of lines and an added special power that the player could activate. The participants were asked to choose one of the available events and create three different alternatives for that event. Additionally, participants had access to event nodes that we created for them, which included blocking a predefined number of rows, clearing blocked rows, and clearing spaces in the Tetris grid. See Figure 11b.

Match 3
Similar to Tetris, there were premade events that the participants had to define. The events enabled special pieces that appear in the game when the player matched more than three pieces. The events also determined what happens when those pieces are cleared. There were nine premade events as follows: four, five or cross-match gem cleared and matching of special pieces (e.g., 4-match + 5-match gem). The participants were guided through an example of what kind of behavior the gems could have. The participants were then asked to create three different alternatives for one of the events with specific functionality for the task available. See Figure 11b.

FPS Target
Similar to the FPS Target worked example described above, the participants were asked to create three alternatives for a Target Blueprint class. An existing Target class was given to them where the target simply moved back and forth and was destroyed with one shot. There were four targets in the level. The participants were told to maintain this number of targets but make sure that they were all different in terms of behavior, appearance and properties. See Figure 11c.

FPS Obstacles & Enemies
In this task the participants were working with a template of an FPS game where the player must make it to the end of the level while avoiding obstacles and fighting enemies. Similar to the FPS Target task, we asked the participants to create three alternatives of a given class. In this case, participants chose either the Enemy or the Obstacle classes for which premade Blueprint classes were available. See Figure 11d.

Task Constraints
To keep the study consistent across participants and to increase internal validity, we imposed the requirements for all the experimental conditions as follows. For the FPS tasks we required the participants to have exactly four targets, enemies and obstacles in the level. This is to ensure that they end up with three alternatives and the original copy of the Blueprint. In the case of using BPAlt, the participants were shown how to use the alternatives system in the Blueprint Editor and how to swap between alternatives in the Level Editor using the Details panel. In the Unreal condition, they had to manually create separate copies of the Blueprint class and replace them in the level. For the Block tasks the participants had to create alternatives for predefined game events, so only one instance of that Blueprint class was required in the level. When the participants were creating alternatives in the Unreal condition for the block tasks, we gave them an option to work within one Blueprint and simply rewire a given graph to create alternatives instead of creating a new copy of the Blueprint, which all the participants opted for. Once a part of a graph is not connected to an event node it effectively becomes dead code. When participants used BPAlt they were instructed on how to swap between alternatives. The participants could either use the Details panel to swap between alternatives or simply press “Play” in the Blueprint Editor of the corresponding alternative (Figure 4b). The original gameplay object in the level is then automatically swapped to the alternative from the Blueprint Editor where “Play” was pressed.

After completing the task with each system, the participants completed the CSI’s agreement statements. After completing all the tasks, the participants completed the CSI’s paired-factor comparison test in compliance with same task, tool comparison repeated measures designs [4]. In addition, the participants left freeform feedback and ranked each system on a 7-point Likert scale for efficiency, ease of use, chance of future use and overall. We then conducted brief semi-structured interviews with the participants where we asked them further questions about BPAlt. The whole procedure took on average 3 hours, with each task taking on average 20.57 minutes to complete.

RESULTS
CSI Survey
The collected CSI data passed all the assumptions tests for two-factor mixed ANOVA. The main effect of System was significant, \( F(1,8) = 8.35, p < 0.05, \eta^2_p = 0.27 \). The CSI score for BPAlt \((M = 79.2, SD = 13.93)\) was higher than for Unreal \((M = 57.3, SD = 21.75)\). The breakdown of CSI survey results is shown in Table 1. The implications of these findings are discussed in the section below. The main effect of System Order \((F(1,8) = 0.02, \text{ns})\) was not significant, indicating that counterbalancing was successful.
Feedback from Participants
We asked participants to rank how they felt using Unreal versus BPAlt on a 7-point Likert scale (1-lowest, 7-highest) in terms of efficiency, ease of use, chance of future use and overall. The results appear in Figure 12.

Here are some of the most mentionworthy comments from the freeform feedback that we received. After the first task which was with BPAlt, P2 stated that creating alternatives was very simple and straightforward to understand, while P4 stated that their inexperience was making it difficult to perform the task. This speaks for the diversity of skill level among participants. After the second task, P2 stated that although working with Blueprints is straightforward and intuitive on its own, when trying to test different ideas, it can be difficult to work without overriding previous progress. This comment confirms the main findings of our study. P2 also stated that it was definitely more convenient to perform the tasks while having the alternatives and that the best part was the ability to swap out individual instances within the level view because it makes it very easy to compare all the ideas simultaneously. P3 stated that the tool made the tasks easier. P9 stated that it was much faster and easier to generate alternatives and apply them to different elements in a game. Two participants stated that BPAlt was well integrated into the system.

Semi-Structured Interviews
The goal of the interviews was to find out how the participants felt about BPAlt, and how the tasks affected their performance and feelings to give as an idea about the usability and usefulness of the system. In contrast to the freeform feedback, this interview was a more targeted approach to get a more detailed explanation through dialog with the participants.

All the participants stated that BPAlt was helpful for the tasks. The main points the participants brought up were as follows:
- **BPAlt** was straightforward to learn and use.
- The Details panel feature was useful for testing alternatives side-by-side.
- For Block tasks, working in one Blueprint made the graphs disorganized, so having the option to create alternatives was very helpful.

P1 said "I would love to have this in my Unreal", P2 and P4 stated that they “missed” BPAlt because the order of their user study featured BPAlt first followed by Unreal.

We tested two different use cases for BPAlt (FPS vs. Block). BPAlt was shown to support creating alternatives for abstract game objects that influenced games mechanics (Block) and for game objects in the form of targets, enemies and obstacles (FPS). P1, P6 and P9 stated that they preferred to use BPAlt for the FPS tasks. P1, P3, P4 and P8 preferred it for the Block tasks. P5, P7, P10 thought that BPAlt was equally useful in both FPS and Block tasks. However, all the participants stated that they still thought that BPAlt was helpful for both types of tasks. While this underlines that the BPAlt can be used in vastly different situations and still be useful, in the hindsight, we wish we asked participants who preferred BPAlt for specific kind of tasks why they thought this was the case.

### Table 1: Average results of CSI Survey after using Unreal (top); BPAlt (bottom).

<table>
<thead>
<tr>
<th>System</th>
<th>Factor `Scale</th>
<th>Enjoyment</th>
<th>Exploration</th>
<th>Expressiveness</th>
<th>Immersion</th>
<th>Results worth effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unreal</td>
<td>Factor counts (SD)</td>
<td>3.3 (1.5)</td>
<td>4.1 (0.6)</td>
<td>2.6 (1.1)</td>
<td>1.5 (1.1)</td>
<td>3.0 (1.5)</td>
</tr>
<tr>
<td>Unreal</td>
<td>Factor score (SD)</td>
<td>11.5 (2.8)</td>
<td>12.0 (2.4)</td>
<td>11.8 (2.3)</td>
<td>9.9 (1.7)</td>
<td>13.7 (1.7)</td>
</tr>
<tr>
<td>Unreal</td>
<td>Weighted factor score (SD)</td>
<td>38.0 (8.3)</td>
<td>49.2 (10.1)</td>
<td>30.7 (10.2)</td>
<td>14.9 (5.3)</td>
<td>41.1 (15.1)</td>
</tr>
<tr>
<td>BPAlt</td>
<td>Factor counts (SD)</td>
<td>3.3 (1.5)</td>
<td>4.1 (0.6)</td>
<td>2.6 (1.1)</td>
<td>1.5 (1.1)</td>
<td>3.0 (1.5)</td>
</tr>
<tr>
<td>BPAlt</td>
<td>Factor score (SD)</td>
<td>17.6 (1.7)</td>
<td>17.3 (1.6)</td>
<td>15.4 (1.9)</td>
<td>11.0 (2.2)</td>
<td>15.5 (1.8)</td>
</tr>
<tr>
<td>BPAlt</td>
<td>Weighted factor score (SD)</td>
<td>58.1 (15.3)</td>
<td>70.9 (9.1)</td>
<td>40.0 (10.6)</td>
<td>16.5 (6.5)</td>
<td>46.5 (15.4)</td>
</tr>
</tbody>
</table>

### Table 2: Results of robust mixed ANOVA based on trimmed means on the participants’ rankings of the systems.

We used robust mixed ANOVA based on trimmed means, a method recommended by Andy Field [10] (see p. 643) as a non-parametric alternative to mixed ANOVA. The main effect of System was significant for all ranked categories. BPAlt was consistently ranked higher. The results for each ranked category are summarized in Table 2 below. The main effect of System Order was not significant for all ranked categories.

<table>
<thead>
<tr>
<th>System</th>
<th>Efficiency</th>
<th>Ease of use</th>
<th>Future use</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>U</td>
<td>B</td>
<td>U</td>
<td>B</td>
</tr>
<tr>
<td>Median</td>
<td>5</td>
<td>6.5</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Main effect of System</td>
<td>$Q = 9.17$, $p = 0.028$</td>
<td>$Q = 18$, $p = 0.0031$</td>
<td>$Q = 13.33$, $p = 0.007$</td>
<td>$Q = 12.85$, $p = 0.014$</td>
</tr>
</tbody>
</table>

Figure 12 Participants’ rankings of Unreal and BPAlt.

**Table 3:** The results of a mixed ANOVA for the CSI survey.
All the participants except P2 said that the first two tasks really helped them to warm up to using Unreal and that they were more comfortable during the last two tasks regardless if BPAlt was used first or not.

We asked participants if during the user study they felt like there was a need to selectively merge a part of the event graph in one of the alternatives to another. All participants showed interest in having selective merging being a feature, but three participants stated that it would not be a necessary addition to BPAlt.

**DISCUSSION**

Here we discuss the design decisions made behind BPAlt, the findings of the user study.

In our user study and the worked example, the Block tasks were portrayed as if a programmer put together special events for the designer to work with. This is a normal practice when using Unreal Engine since one of its features is creating custom Blueprint events which are defined and triggered in C++ or other Blueprints. This allows the designer to work modularly and not have to unnecessarily deal with complex code.

The reason that the participants were given a limited number of Blueprints to work with was to make sure that a comparable amount of work was done across all the participants, which increases the internal validity.

We gave the participants a lot of work space in the form of multiple monitors, however most participants opted to use only one monitor to work with. This can be attributed to the short and focused nature of the tasks. However, in other creative domains multiple monitors for supporting exploration of alternatives were shown to be useful (e.g. [31,41]).

We allowed the participants to take as much time as they needed to complete the tasks since we anticipated that many of them will need time to get used to using Unreal and we did not want to stifle their creative process. This was found to be the case since our participants had varying levels of experience with Unreal. Some participants had trouble getting started during the study using Unreal due to lack of experience or not having used Unreal for months or years. P3 and P4 expressed frustration with the interface of Unreal for being hard to get used to due to the differences from Unity which they were more familiar with. However, no complaints were received about the interface of BPAlt. It is important to note that some participants that used BPAlt first thought that it was a native feature of Unreal.

From the breakdown of the CSI survey results, we conclude that the support for exploration with BPAlt was particularly well received by our participants, since the exploration was the most important factor to them and the weighted factor score for exploration was significantly higher for BPAlt than for Unreal. This underlines that our system supports exploration of alternatives well. Based on the CSI results and overall comments made by the participants it can be implied that the functionality supported by BPAlt will be desirable to have in the existing tools that game developers use. In addition to the CSI survey, we designed our own post-questionnaire where participants ranked the two systems. In all ranked categories participants felt more positive about BPAlt in their workflows. In this questionnaire we also asked the participants what their preferred system was to use during the tasks. All 10 participants indicated that they preferred using BPAlt. Overall, we found that using alternatives in game development can improve the creative process of prototyping significantly. This is consistent with previous work done on alternatives in other creative domains.

We believe the functionality that BPAlt provides for supporting alternatives will also be desirable in visual programming in general as there is evidence that similar functionality can work with text programming as well [17]. Although our user study focused specifically on the game development aspect of Unreal, the approach can also be used in other applications such as industrial simulations and animated films.

**CONCLUSION, LIMITATIONS & FUTURE WORK**

We presented BPAlt—an extension of the Unreal Engine that supports the exploration of Blueprint alternatives. BPAlt addressed the process in the iteration cycle of game developers for creating and testing different possibilities for gameplay classes. We tested the main features of the system in a user study and found that BPAlt supports developers’ workflow better than the unenhanced version of Unreal and that it was also preferred by all participants. In particular, BPAlt was found to support exploration better. Exploration is an integral part of creativity support, and thus we argue that BPAlt has a potential to support creativity well. It is important to note that the results are preliminary as were unsuccessful in recruiting the targeted sample size of 24 participants.

In the future, a longitudinal study can be conducted to evaluate selective merging, level alternatives and alternative scenarios. In our study we focused on visual scripting in the context of game objects and mechanics accessible through Blueprints. We did not cover all forms of visual scripting available in Unreal such as, e.g., Material, Animation, Widget, or Sound. Future studies can focus on investigating if the benefits of alternatives are transferable to these other visual scripting systems of Unreal.

**REFERENCES**


