

EXPLORING DESIGN ALTERNATIVES IN GAME
DEVELOPMENT ENGINES USING VISUAL
PROGRAMMING

ERIC CHU

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE
UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY
OSHAWA, ONTARIO

APRIL 2019

© ERIC CHU, 2019

THESIS EXAMINATION INFORMATION

Submitted by: **Eric Chu**

Master of Science in Computer Science

Thesis title: Exploring Design Alternatives in Game Development Engines Using Visual Programming

An oral defense of this thesis took place on April 8, 2019 in front of the following examining committee:

Examining Committee:

Chair of Examining Committee	Dr. Karthik Sankaranarayanan
Research Supervisor	Dr. Loutfouz Zaman
Examining Committee Member	Dr. Pejman Mirza-Babaei
Examining Committee Member	Dr. Jeremy Bradbury
External Examiner	Dr. Christopher Collins

The above committee determined that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate during an oral examination. A signed copy of the Certificate of Approval is available from the School of Graduate and Postdoctoral Studies.

ABSTRACT

We present *BPAIt* – a system which allows game developers to create and manage alternatives for *Unreal Engine*’s *Blueprints Visual Scripting System*. *BPAIt* allows the user to create, save, organize and swap Blueprint alternatives for rapid testing and experimentation. We conducted a user study with 10 moderately skilled participants where we compared *BPAIt* to *Unreal Engine* alone for prototyping alternatives of game objects and mechanics in four different games. We found evidence that supporting alternatives with *BPAIt* is beneficial in the game developers’ workflow. In response to the results of the user study we implemented new features for selectively merging parts of one alternative Blueprint to another. We also implemented an interface for alternative scenarios.

ACKNOWLEDGEMENTS

Thank you to everyone who supported me through my journey of graduate studies. My family who has supported me through my entire academic career. I hope I can make you guys proud.

Loutfouz, I will always be so thankful for the opportunities you gave me and the time and effort you have put towards me and my work. I will miss our weekly meetings, our discussion, situations where we'd discover some kind of disaster because of something that we overlooked. It was a fun ride.

Thanks to my friends from the lab, you guys made my time here much better whether you helped me out when I was stuck or was just around for long distracting chats, I'll miss you guys. Special thanks to James Robb for helping me out with the expert evaluation and for always being happy to help me out whenever I needed it. Thanks to UOIT and NSERC Discovery for funding this research.

TABLE OF CONTENTS

CHAPTER 1 Introduction	1
1.1 Game Engines	1
1.2 Prototyping.....	3
1.3 Research Contributions	6
1.4 Thesis Structure	7
1.5 Chapter 1 Summary	7
CHAPTER 2 Related Work	9
2.1 Prototyping in Game Development	9
2.2 Earlier Work on Design Alternatives.....	10
2.3 Alternatives in Creativity Support Tools	11
2.4 Chapter 2 Summary	14
CHAPTER 3 BPAIt: A System for Creating and Managing Alternatives in Unreal Engine’s Blueprints Visual Scripting System	15
3.1 Creating Alternatives	17
3.2 Swapping Alternative Blueprints	19
3.3 Design Choices	23
3.4 Chapter 3 Summary	24
CHAPTER 4 User Study	25
4.1 Research Questions and Hypotheses	25

4.1.1	Research Question 1	25
4.1.2	Research Question 2	26
4.1.3	Research Question 3	26
4.2	Participants.....	27
4.3	Apparatus	28
4.4	Procedure	30
4.4.1	Phase 1: Introduction	30
4.4.2	Phase 2: Creating Blueprint Prototypes	31
4.4.3	Phase 3: Interview and Debriefing.....	38
4.5	Discussion	38
4.6	Chapter 4 Summary	39
CHAPTER 5 User Study Results		40
5.1	Creativity Support Index.....	40
5.1.1	Assumption Tests.....	40
5.1.2	Mixed ANOVA Test.....	42
5.2	Feedback from Participants.....	43
5.3	Semi-Structured Interview	46
5.4	The Positive and Negative Affect Schedule (PANAS).....	49
5.5	Relevant Metrics	53
5.6	Expert Evaluation – Design Quality	54

5.7	Discussion	57
5.8	Chapter 5 Summary	59
CHAPTER 6 Overall Discussion and Conclusion		60
6.1	New Features Implemented in Response to the Results of the User Study	64
6.1.1	Selective Merging	64
6.1.2	Alternative Scenarios	69
6.2	Chapter 6 Summary	69
CHAPTER 7 Implementation		71
7.1	Blueprint Alternatives	71
7.1.1	Swapping between alternatives	72
7.2	Selective Merging	73
7.3	Chapter 7 Summary	84
CHAPTER 8 Future Work and Limitations		85
8.1	BPAIt.....	85
8.2	Limitations of the User Study	87

Bibliography	89
Appendix A CSI Survey	95
Appendix B Positive Affect Negative Affect Schedule (PANAS)	113
Appendix C Pre-Study Verbal Script.....	115
Appendix D Pre-Study Survey.....	116
Appendix E Post-Study Survey	124

LIST OF TABLES

Table 4-1: The independent variables with levels in the experimental design.	31
Table 4-2: The independent variables and levels used in our experimental design. The table also demonstrates how we counterbalanced the experimental condition between participants to decrease the effect of learning.....	33
Table 5-1: Shapiro-Wilk Normality Test of the CSI Score for System and System Order.	42
Table 5-2: Levene's Test of Equality of Error Variance of the CSI Score between <i>Unreal</i> and <i>BPAIt</i>	42
Table 5-3: Average results of CSI Survey after using Unreal (top); BPAIt (bottom).	43
Table 5-4: Robust mixed ANOVA results on the participants' rankings of the systems. 10,000 bootstrap samples were used. *-significant at $\alpha=0.1$, **-significant at $\alpha=0.05$	44
Table 5-5: The independent variables with levels in the PANAS analysis.	50
Table 5-6: Means and standard deviations of positive and negative PANAS sum scores.	53
Table 5-7: Expert evaluation: means and standard deviations.....	55
Table 5-8: Expert Evaluation: Mann-Whitney U test results.....	55

Table 6-1: Average time (in minutes) and number of tests for tasks regardless of task type.....	62
Table 7-1: Selective Merge Exception Cases	84

LIST OF FIGURES

Figure 1-1	4
Figure 3-1: The FPS Target worked example: Target Blueprint Graph (original blueprint)	16
Figure 3-2: The FPS Target worked example: Blueprint Editor of the Target Blueprint alternative using BPAlt. a) Window menu bar section. b) Blueprint Alternatives menu c) Save button for the alternative in the currently open Blueprint. d) Alternatives set. e) Blueprint alternatives tab.	17
Figure 3-3: Blueprint graph of the original “Special Power” Blueprint in the Tetris worked example.	20
Figure 3-4: An alternative of the” Special Power” Blueprint of Tetris worked example. a) Blueprint Editor tabs for the original and alternative Blueprints. b) Play button. c) Special power event node.	21
Figure 3-5: FPS Target game before swapping the actor for an alternative.	22
Figure 3-6: FPS Target game after swapping the actor for an alternative. a) Target Blueprint actors in the world outliner. 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.....	23
Figure 4-1: Experimental setup for BPAlt user study. a) The investigator’s computer that was being used to lead the participants through the tasks. The display was duplicated for	

the participants to follow along during the tutorial. b) The laptop computer for filling out questionnaires and transcribing the interviews. c) The computer with four external monitors the participants used to complete the tasks.....	29
Figure 4-2: The four-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.	30
Figure 4-3: The four project templates used for the tasks: a) Tetris, b) Match 3, c) Target, d) Obstacles & Enemies.....	34
Figure 5-1: Box Plot of CSI Scores comparing BPAIt and Unreal by itself. No outliers were identified.	41
Figure 5-2: Participants' rankings of <i>Unreal</i> and <i>BPAIt</i>	45
Figure 5-3: PANAS individual scores	51
Figure 5-4: Box Plot of Positive and Negative Affect of mean PANAS Scores	52
Figure 5-5: Expert evaluation: grades per participant.....	56
Figure 6-1: 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.	65
Figure 6-2: After selective merge completed in Test Blueprint.	65
Figure 6-3: Test Blueprint before selective merge.	66

Figure 6-4: Test_Alternative 1 Blueprint is selectively merging to the Test Blueprint. 11 nodes are selected. a) Blueprint variables. b) Variable reference nodes.	67
Figure 6-5: Test Blueprint after selective merge.	68
Figure 7-1: Blueprint alternative master list class and Blueprint alternative data class ...	72
Figure 7-2: Blueprint Alternatives Menu a) Checkbox to toggle swapping between alternatives in the Blueprint editor using the Play button.....	73
Figure 7-3: Classes for Blueprint graph nodes and node pins	74
Figure 7-4: Important Selective Merge classes.....	76

Chapter 1

Introduction

1.1 Game Engines

Due to its efficiency and flexibility, C++ is among the most popular languages used in game development – an industry which often pushes the limits of modern hardware to deliver quality products to stand out in fierce competition. In the past, it was typical to use frameworks or libraries to create video games purely through code.

In recent years, modern game engines such as the *Unity Engine*¹ and the *Unreal Engine*² have become the most popular option for game developers having millions of users respectively. These engines feature suites of integrated graphical user interfaces (GUIs) and various tools to streamline the game development process. Besides video games, these game engines are also used to create simulations, animated movies and even HCI research tools (see e.g., [41]).

Recently visual programming has been gaining popularity in modern game engines. Examples include visual scripting plugins for the popular *Unity Engine* such as *FlowCanvas* [57], *Playmaker* [58], *Bolt* [59] and *Amplify Shader Editor* [60]. *Unreal Engine* features a native visual programming system called *Unreal Blueprints*. Initially

¹ <https://unity3d.com/>

² <https://www.unrealengine.com>

these efforts have been directed to support rapid prototyping and aiding designers in their testing process, but this is changing.

The *Unreal Engine* 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 hobbyists and professional developers alike. The engine features *Blueprint Visual Scripting* [61] 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 *Unreal Engine* 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. On the other hand, C++ was traditionally used for developing the final product efficiently. Starting with version 4.15, the support for *Blueprint Nativization* [62] was added to reduce virtual machine overhead in the runtime by generating native C++ code Blueprints. This process was first used successfully during the development of *Robo Recall* [63], a virtual reality game by Epic Games. 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 the well 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.

1.2 Prototyping

In game development and particularly in game design the iteration process is applied during almost every aspect of design: from the initial conception through the final quality assurance testing [11]. See Figure 1-1. Research in regards to prototyping in game development has been previously done through user testing and iterating based on data and user feedback (see e.g., [6,8]). However, creating prototypes on a micro scale for an individual or a team remains a neglected area of research.

In other creative fields, experts typically generate sets of alternative solutions when solving ill-defined problems [47]. This has been shown to result in higher quality outcomes [9]. This can be seen the workflow of architects [1,33], web designers [38], and software engineers [49] who generate sketches of potential 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].

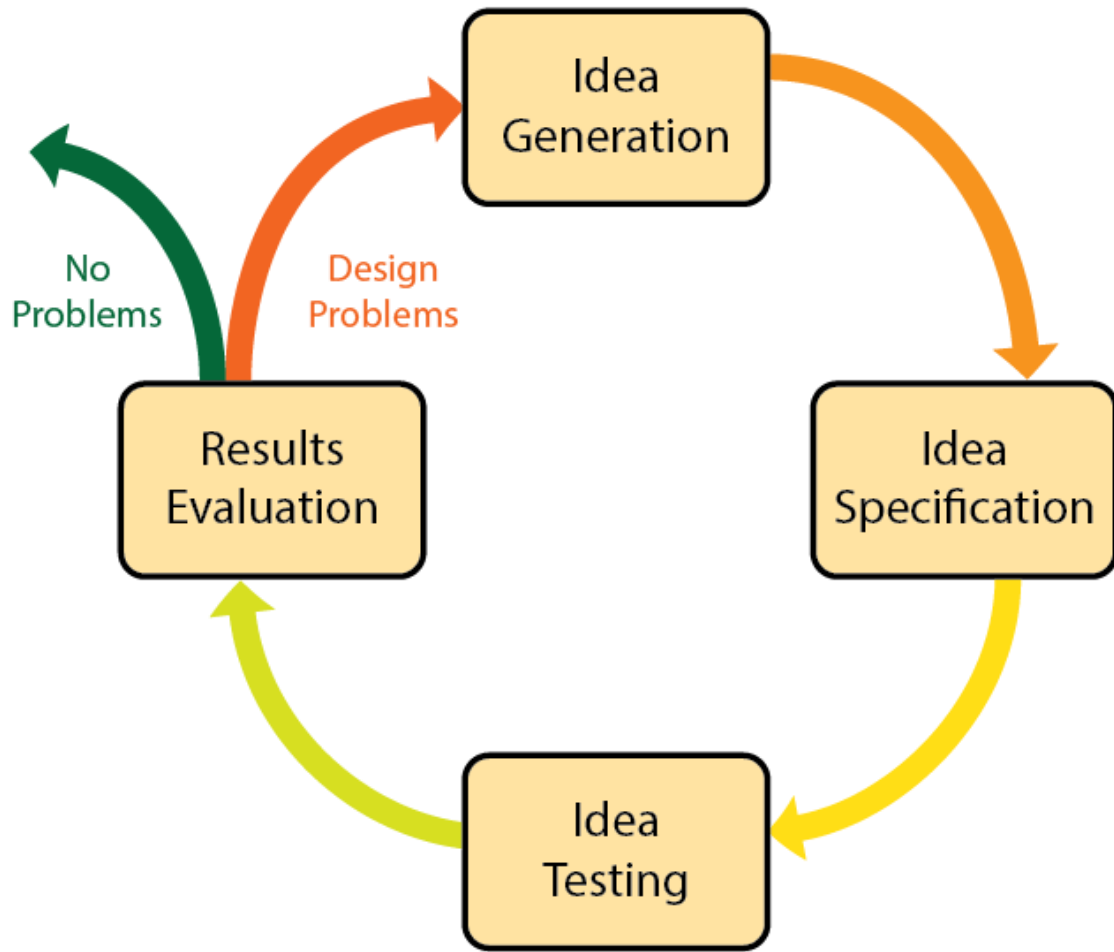


Figure 1-1: Diagram of the iterative process courtesy of Tracy Fullerton. [11]

Evidence has been found that design alternatives improve exploration of the design space in various creative fields (see e.g., [31,36]), including alternatives for visual programming environments (see e.g., [54–56]). This work investigates whether using design alternatives in the context of game development, specifically using *Unreal Blueprints* – a visual programming system, improves the workflow of game developers.

In software development, iteration is used when working on a project: be it larger iterations or smaller changes [23]. 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 [11]. Research in prototyping for game development has been previously done through user testing and iterating based on data and user feedback (see e.g. [6,8]). 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 *Unreal Engine 4*, which allows 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 and usefulness 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.

Alternatives are an integral part of conceptual design [56]. 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.

1.3 Research Contributions

We are the first to study this behavior in the context of game developers' workflow. To increase the ecological validity of this research we implemented our solution as an extension to *Unreal Engine 4* – a popular game engine used by the industry. It is currently the only game engine which places node-based visual scripting as the primary paradigm for developing games. Some of the commercially successful games made with *Unreal 4*, as of Q1 2019, include: *Fortnite Battle Royale*, *Abzû*, *Mortal Kombat X*, *Street Fighter V*, *Tekken 7*, *Injustice 2*, *Hellblade: Senua's Sacrifice*, *Gears of War 4*, *Batman: Arkham VR*, *Little Nightmares*, *Life Is Strange 1/2*, *Moss*, and the highly anticipated *Biomutant*, *Days Gone* and *Yoshi's Crafted World*, among many others. We developed *BPAlt* (**B**lueprint **A**lternatives) – an extension for *Unreal Engine 4*, which allows exploration of alternatives – an integral part of conceptual design [56], which is present in the workflow of game developers. An alternative is defined as a potential solution to a given problem that can be compared to other alternatives. In the context of this work alternatives are gameplay classes that can be potentially used. Our system introduces methods for creating and managing Blueprint alternatives and a streamlined way of swapping between Blueprint alternatives, which allows for their rapid testing, experimentation, streamlined saving and organization without removing any of the existing benefits from using the *Unreal Blueprints* system.

To demonstrate the usability and usefulness 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.

1.4 Thesis Structure

Chapter 2 discusses related works that used a similar approach of incorporating design alternatives in their systems.

Chapter 3 discusses *BPAlt*, a system for creating and managing alternatives in *Unreal Engine's Blueprints*, which incorporates the concept of design alternatives into the Unreal Blueprints visual programming system.

Chapter 4 discusses the user study conducted to test usability and usefulness of *BPAlt*. The participants, apparatus, and procedure involved in the user study are described.

Chapter 5 presents the evaluations of the results of the user study and includes an expert evaluation of the work done by participants.

Chapter 6 concludes the thesis with a summary of findings and discussion of the results. Changes to *BPAlt* since the user study are also described.

Chapter 7 specifies the implementation of *BPAlt's* features.

Chapter 8 describes the limitations of the work and future work that is planned to be done.

1.5 Chapter 1 Summary

This chapter presented an overview of the game development process and importance of the iterative process. The importance of prototyping and the limited prototyping methods currently available in game development, is highlighted. Current trends in game development programming paradigms are discussed. Visual programming is identified as a viable approach to game programming thanks to modern technology. The lack of

adequate support for exploration and experimentation of design alternatives in visual programming environments for game development is identified. A solution in the form of the research contribution is proposed and discussed. The structure of this thesis is outlined in this chapter.

The next chapter discusses related work in the use of design alternatives in different fields, none directly tying into game development. However, they all pertain to related fields: creatively inclined fields or programming applications.

Chapter 2

Related Work

2.1 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) [44]. Measuring the progress of improvement is traditionally done in iterations with a release of a build of the game.

Work has been done to improve the iteration process in game development by leveraging player data to measure if the current state of the product provides acceptable experience to players. Beyond traditional playtesting, Mirza-Babaei et al. [34] investigated new forms of obtaining results from users through the use of biometrics. Nacke [37] summarized physiological player metrics for evaluating games. Robertson et al. [42] introduced emotional reporting techniques for assessing gameplay experience.

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 *Unreal Engine*, 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.

2.2 Earlier Work on Design Alternatives

The work on alternatives can be traced to the work of Terry et al. [50] work on investigating creative needs in UIs for experimentation, exploration and evaluation of alternatives. 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. [51] presented *Parallel Paths*, a model of interaction that facilitates generating, manipulating, and comparing alternative solutions. The model was implemented in *Parallel Pies*, a user interface 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 [25,26], gathering and comparing results from alternative resources that offer nominally the same processing [27,28], exploratory access to online resources [10,24], exploratory e-learning [20] and for information access, real-time simulation, and document design [29]. In the work of Marks et al. [31] *Design Galleries* 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 solutions space.

2.3 Alternatives in Creativity Support Tools

In modern game engines, the workspace is built to allow parametric editing to enable faster testing and tweaking of the projects. This is not unique to game development. Parametric editing existed in other creativity support tools [45,46], such as 2D graphics and 3D model editors, long before modern game engines were introduced in mid 2000s.

Much of the work in this thesis draws the inspiration from recent advancements in Computer Aided Design (CAD) – a domain where the use of alternatives has experienced growing popularity in recent years. Some of the works described below also feature node-based interfaces which allow model definition through visual programming not unlike in *Unreal Blueprints*.

GEM-NI [54–56] 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, and to visualize differences between two or more alternatives in the graph, parameter and output views. *CAMBRIA* [22] is a multi-state design tool for simultaneously managing multiple 2D vector graphics alternative design models which can be explored in parallel. Matejka et al. [32] 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. Kazi et al. [21] 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. Cristie and Joyce [5] introduced a workflow plugin for *Grasshopper* [43] which enables parametric structures to be tracked in a similar way to *Git* [64] with branching support and allowing users to save the current parametric design state onto the cloud. The recorded options and data are visualised in a graph. Mohiuddin et al. [35] presented an online gallery system for design alternatives in parametric modeling, which supports multiple commercially available parametric modelers. Woodbury et al. [53] 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.

Elkhaldi and Woodbury [7] 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. *d.note* [14] 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. *Juxtapose* [15] 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. O’Leary et al. [40] 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. Hailpern et al. [13] 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. [48] 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.

Implications for BPAIt

The success and continued interest of integrating alternatives in creativity support tools as described above was behind our motivation to investigate the use of alternatives –an integral part of conceptual design – in game development. Our solution adopts the ideas introduced in *Juxtapose* [15] and *GEM-NI* [56] for use in the workflow of game developers. Game developers work in teams and communicate between team members about design choices and therefore supporting alternatives has a potential to also improve this collaborative aspect of developer’s workflow as found in previous work [13],[40].

2.4 Chapter 2 Summary

Chapter 2 started by describing the iterative process of game development and suggested that this process can be improved through the use of design alternatives in the developers' workflow. The chapter then describes related work on the use of alternatives in creativity support tools, primarily in design and programming software. None of the surveyed research directly involved game development. This presents us an opportunity to investigate the use of alternatives in game engines, which has not been tried before. We believe modern game engines such as *Unreal Engine* and *Unity* would be included into Schneiderman's [45,46] list of creativity support tools, if they were common at the time of these publications. Therefore, introducing the support of alternatives – an integral part of conceptual design – to game engines would be a natural adaptation of this practice to another creative domain. As a result, we adapted the concepts and methods tested in other creative fields in our work. The next chapter describes *BPAIt*, a system that we developed for creating and managing alternatives in *Unreal Engine*'s Blueprints visual scripting system. The chapter also describes the integration of the system into the *Unreal Engine* at a high level.

Chapter 3

BPAlt: A System for Creating and Managing Alternatives in Unreal Engine’s Blueprints Visual Scripting System

We created a system that incorporates design alternatives into a game engine. We describe how alternatives can be used with our system using a worked example. The *Unreal Engine* features widely used visual scripting system known as *Unreal Blueprints*. Previous work successfully demonstrated the benefits of using design alternatives in visual programming systems [54–56]. As a consequence, *Unreal Engine* with its visual scripting is a perfect target platform for our research. Beyond this, we chose the *Unreal Engine* because it is one of the most used game engines in the industry thus increasing the ecological validity of our research.

We developed *BPAlt* (*Blueprint Alternatives*) – an extension for *Unreal Engine 4*, which allows the support of alternatives. The interface of *BPAlt* was designed to be minimally intrusive to the workflow of game developers who are already familiar with *Unreal Engine*. Using *BPAlt* the user creates and edits Blueprint alternatives in a single-state document [50] (in the Blueprint Editor) but still can easily explore different ideas in parallel or individually through *Unreal*’s Level Editor.

In *Unreal Engine*, 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 *Unreal Engine* 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 *BPAIt* we will be using a use case example described in Sections 3.1 and 3.2.

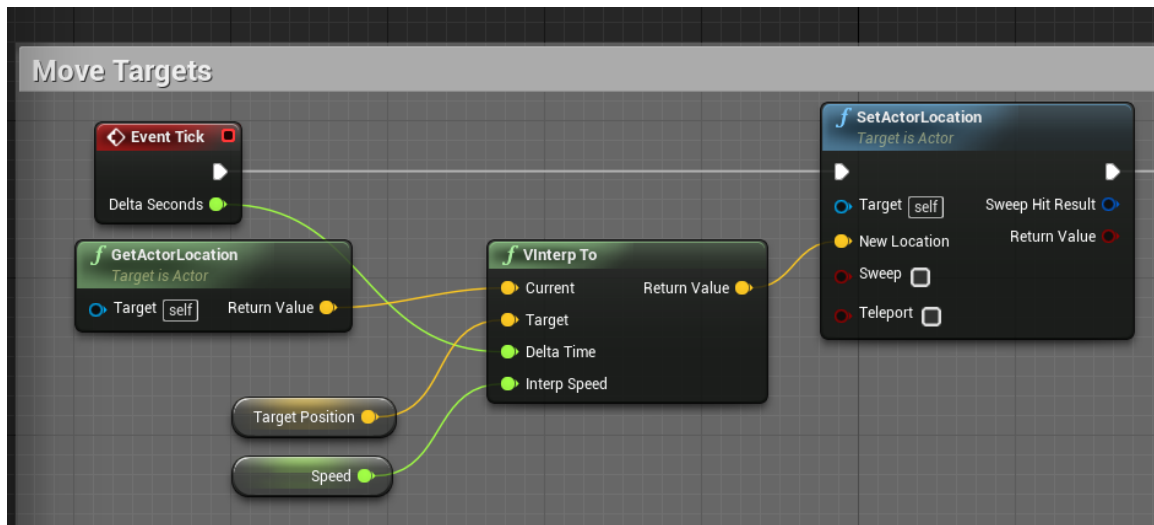


Figure 3-1: The FPS Target worked example: Target Blueprint Graph (original blueprint)

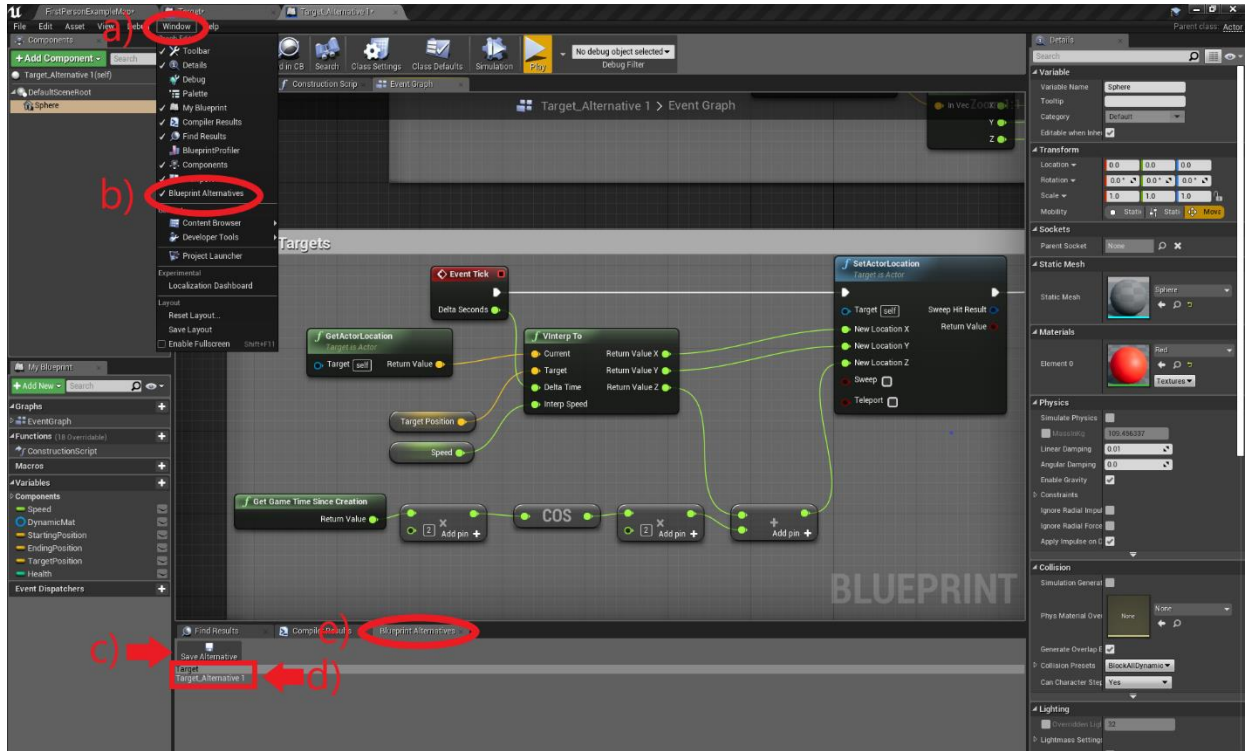


Figure 3-2: The FPS Target worked example: Blueprint Editor of the Target Blueprint alternative using BPAlt. a) Window menu bar section. b) Blueprint Alternatives menu c) Save button for the alternative in the currently open Blueprint. d) Alternatives set. e) Blueprint alternatives tab.

3.1 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 3-2), which is just a white sphere that moves back and forth, he places four of these targets in the level, see Figure 3-5. Tim then uses *BPAIt* 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 3-2a) via the Window menu Figure 3-2b). The Blueprint Alternatives tab opens at the bottom of the screen Figure 3-2e) but can be undocked and moved around. Tim then saves an alternative of the currently opened Target Blueprint via the “Save Alternative” button Figure 3-2c), which will add a new alternative to the alternatives set, which appears in the list of created alternatives in the Blueprint Alternatives tab Figure 3-2d), 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 3-6c).

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 like duplication. However, unlike duplication, the process is kept track of, recording which blueprint it is copied from and other alternatives within the same

alternative set. The recorded data is used for extra functionality such as making the process of swapping and testing alternatives much more streamlined using *BPAIt*.

3.2 Swapping Alternative Blueprints

Swapping between alternatives for testing is the core feature that distinguishes *BPAIt*'s approach from simple duplication. *BPAIt* 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 outline* (Figure 3-6a), 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 3-6b). 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 3-6c). 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 3-6b). *Note: there is a button called "Regenerate Alts" that regenerates the dropdown menu if the list is incorrect, specifically used for the user study in case there was an issue.*

The next example shows another way of swapping between alternatives. Imagine Jen, a game designer, is tasked with designing a *special power* which players activate in a

Tetris-like game. A programmer on her team sets up a Blueprint class called Special Power (Figure 3-3) which contains the event node Figure 3-4c) 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 then decides to create another iteration of the special power by using *BPAIt* 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 3-4b) 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.

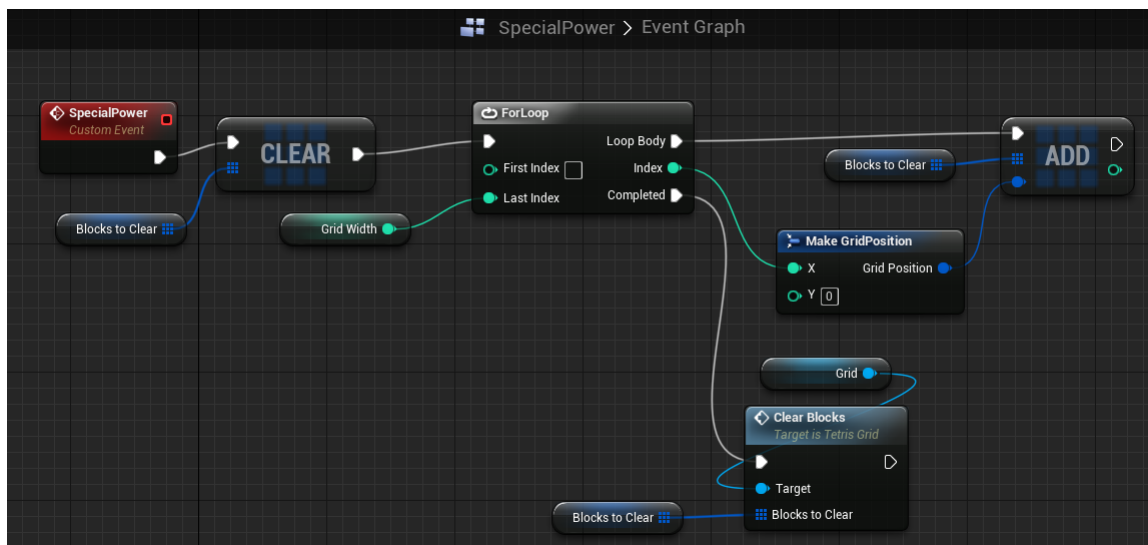


Figure 3-3: Blueprint graph of the original “Special Power” Blueprint in the Tetris worked example.

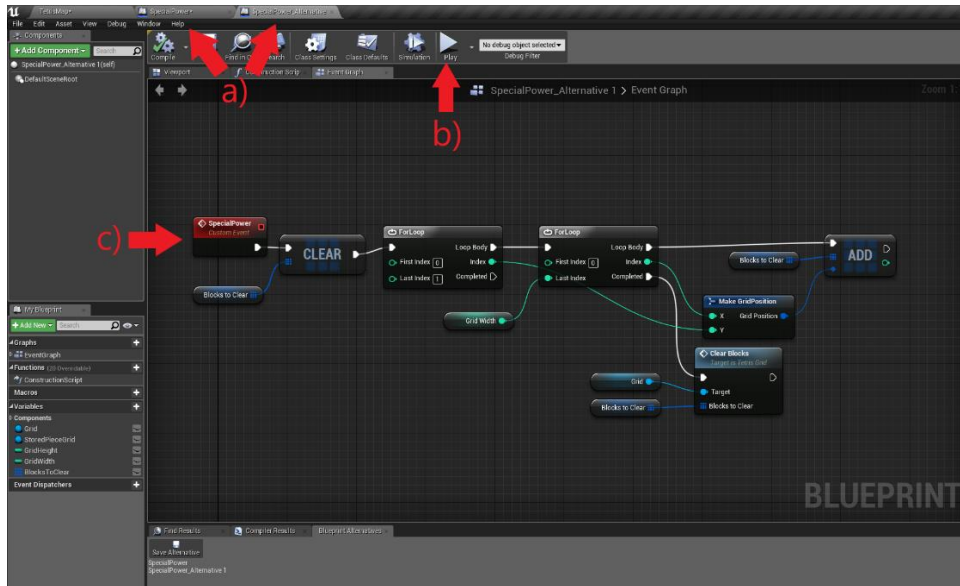


Figure 3-4: An alternative of the” Special Power” Blueprint of Tetris worked example. a) Blueprint Editor tabs for the original and alternative Blueprints. b) Play button. c) Special power event node.

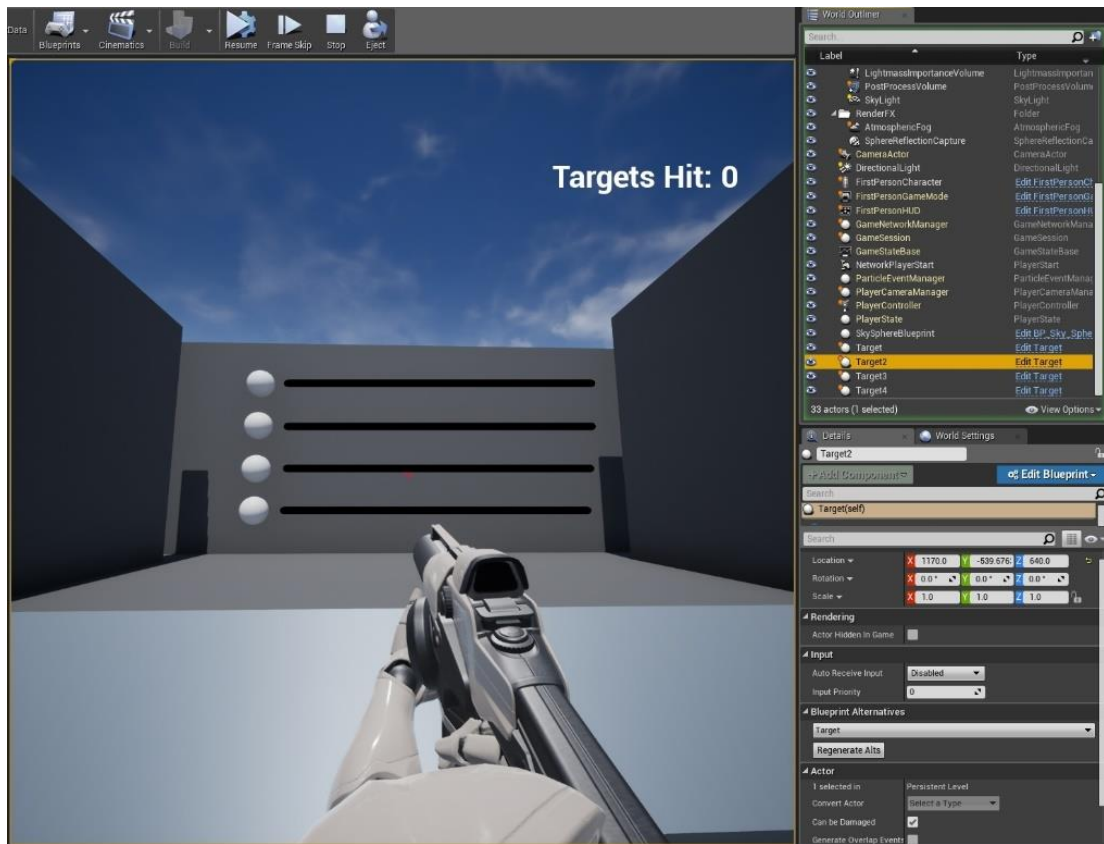


Figure 3-5: FPS Target game before swapping the actor for an alternative.



Figure 3-6: FPS Target game after swapping the actor for an alternative. a) Target Blueprint actors in the world outliner. 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.

3.3 Design Choices

All the design choices were made with the intention of being seamlessly integrated into the typical workflow for game developers.

The method to create alternatives was based on the idea of prototyping and the process of iterating, which builds upon previous work (see Chapter 2). Making the creation of alternatives dependant on the Blueprint Editor that the menu belongs to allows users to

create iterations of Blueprints easily. The method was also inspired by earlier systems that used alternatives in the design field, more specifically *GEM-NI* [56], *Juxtapose* [15] and *CAMBRIA* [22]. The interactions using *BPAIt* are meant to not be intrusive with the regular workflow of game developers. The way that the alternatives are saved and presented is akin to version control systems, but the system has extra functionality to help along the prototyping process. Swapping between alternatives using the details panel was meant to accommodate game developers by allowing easy access to swap between alternatives so that the process of rapid prototyping can be convenient.

3.4 Chapter 3 Summary

In this chapter we discussed the features and functionality of *BPAIt*: creating and managing Blueprint alternatives and swapping between Blueprint alternatives in the level. The support for Blueprint alternatives was implemented to add functionality for features that are tested in Chapter 4.

We also discuss the design choices made to accommodate the existing game development workflow and how they were inspired by the previous work on design alternatives.

The next chapter describes the comparative user study which we conducted to test the usability and usefulness of *BPAIt* compared to just using *Unreal Engine* by itself.

Chapter 4

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 *Unreal Engine 4.17.2*, which we refer to simply as *Unreal* from now on.

This study was approved by the Research Ethics Board at the University of Ontario Institute of Technology (REB# 14883).

4.1 Research Questions and Hypotheses

The user study captured metrics for user experience, preference, performance and system usability. These metrics were captured to answer three research questions below.

4.1.1 Research Question 1

Can design alternatives be integrated well into a game development environment to be non-intrusive with the existing workflow?

To answer this question, we measure the user experience how the users felt after using both *BPAlt* and *Unreal* on its own. Since there are many different approaches to measure user experience, we used several different methods. To compare the creative output we used the Creativity Support Index (CSI) survey [4] to measure how users felt using the system during a creative task. The results are reported in Section 5.1. To measure the user's mood we used Positive and Negative Affect Schedule (PANAS) [52], which is

discussed further in Section 5.4. To measure performance, we measured the iteration time. We measured the time it took each participant to complete each task and the number of tests (simulating the game) conducted during each task. The results can be seen in Section 5.5. The following research hypothesis was developed to investigate this research question.

Hypothesis 1 (H1): *BPAlt will not encumber users from completing tasks in comparison to Unreal on its own.*

4.1.2 Research Question 2

How does BPAlt compare to traditional prototyping methods in terms of user preference?

This question explores the user's preference and performance when using *BPAlt* compared to traditional prototyping in *Unreal*. To measure preference, we used the post-study questionnaire. The results can be seen in Section 5.2. To measure performance, we got an expert to evaluate the game prototype alternatives created by the participants. The following research hypothesis was developed to investigate this research question.

Hypothesis 2 (H2): *Using design alternatives will increase levels of creativity of the users.*

4.1.3 Research Question 3

Does BPAlt improve the iteration time when developing game prototype alternatives?

This question pertains to the user's performance in terms of the time it takes to complete an iteration. To measure the iteration time, we measured the time it took each participant to complete each task and the number of tests (simulating the game) conducted during each

task. The results can be seen in Section 5.5. The following research hypothesis was developed to investigate this research question.

Hypothesis 3 (H3): *Using BPAlt will either be the same or better than traditional prototyping methods in terms of iteration time.*

The discussion of these research questions can be seen in Section 5.7.

4.2 Participants

10 paid participants (2 females) were recruited from undergraduate and graduate students at the University of Ontario Institute of Technology. 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 ($M = 4.71$ years, $SD = 3.26$). 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 flowcharts or diagrams at least once a month. The same nine participants had experience using visual programming systems to most common being *Scratch* and *Unreal Blueprints*.

4.3 Apparatus

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 4-2). The monitor in the top left had 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 computer to show the participants how to use *Unreal* and *BPAIt*, test the level, navigate Blueprints, and demonstrate an example of how to use each task template. The view of the investigator was duplicated for the participants to easily see what they were doing (Figure 4-1b) Pen and paper were provided if requested (which happened twice). Interview answers were transcribed, and questionnaires were filled out on a separate laptop computer. Screen capturing was used during the duration of the study using Open Broadcaster Software (OBS) [65].

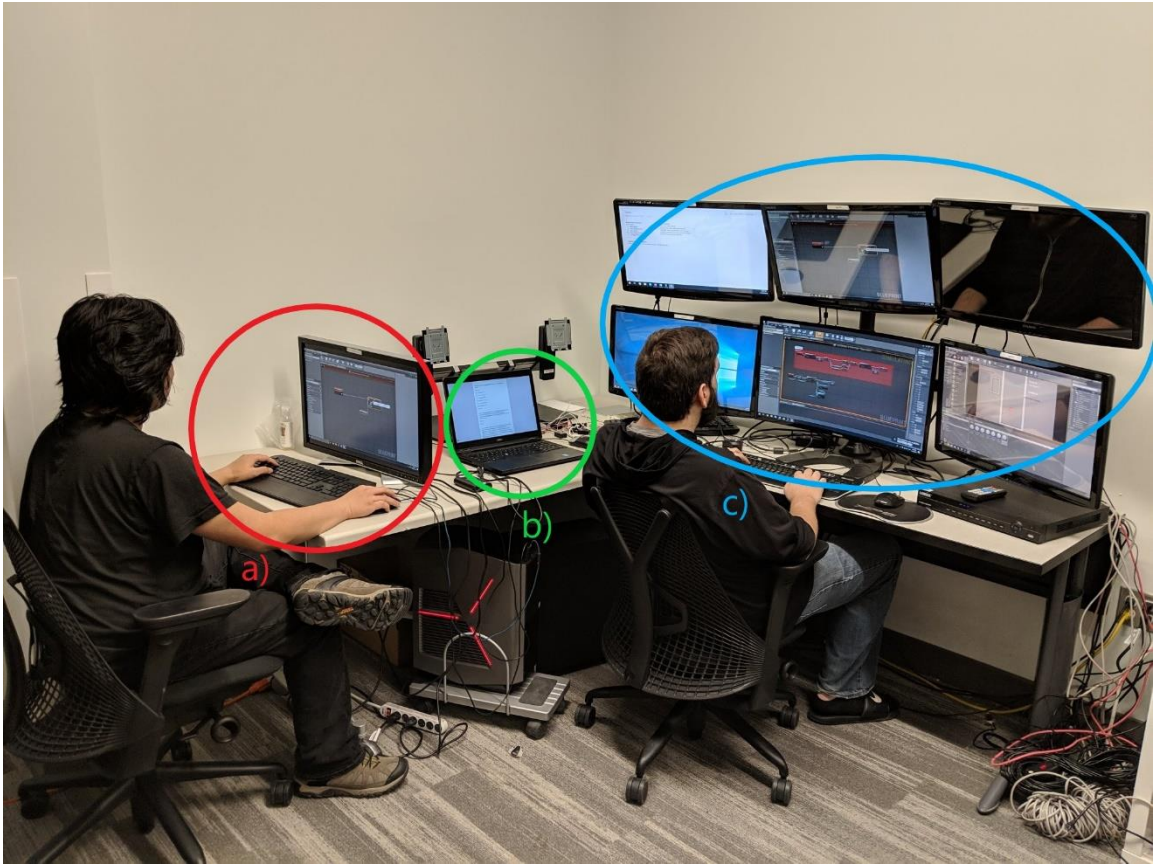


Figure 4-1: Experimental setup for BPAIt user study. a) The investigator's computer that was being used to lead the participants through the tasks. The display was duplicated for the participants to follow along during the tutorial. b) The laptop computer for filling out questionnaires and transcribing the interviews. c) The computer with four external monitors the participants used to complete the tasks.

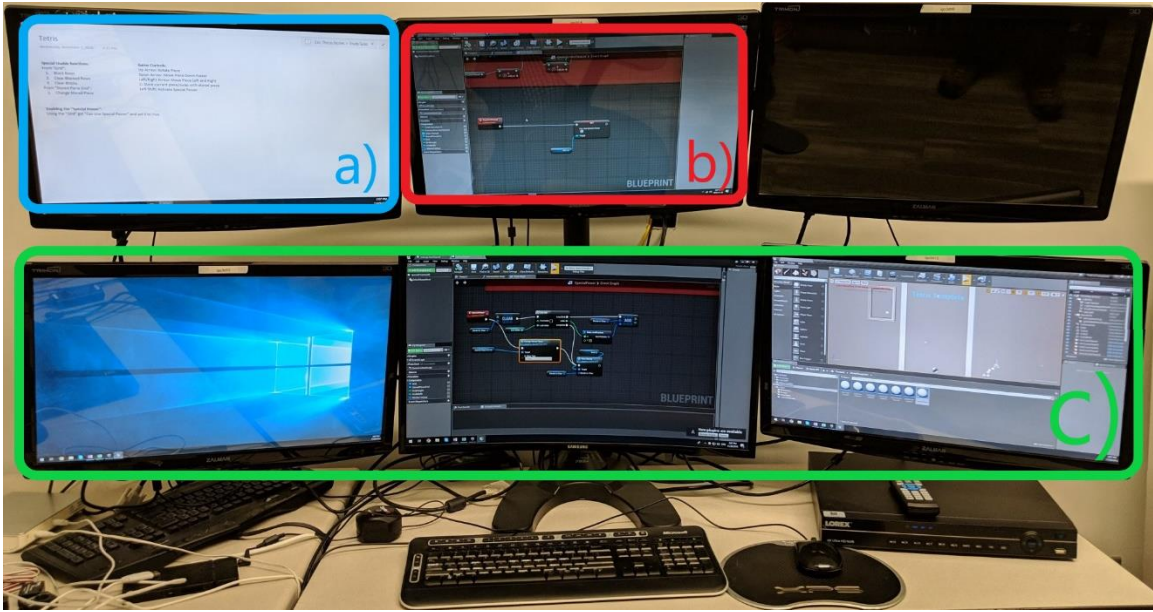


Figure 4-2: The four-monitor setup in the user study. (a) Monitor for the task instructions, could be used by the participants. b) Monitor controlled by a separate computer to guide participants through the tasks. c) Primary monitors used for the tasks.

4.4 Procedure

4.4.1 Phase 1: Introduction

When the participants arrived, we presented them with a pre-study questionnaire (Appendix D) to gather demographics and their experience using game engines and visual programming systems. We also gave the participants the PANAS questionnaire (Appendix B) for the first time to gauge their initial positive and negative emotions. The participants were read a script briefing them of the nature of the tasks (Appendix C) and told that the study should take roughly 3 hours to complete. The participants then started the tasks.

4.4.2 Phase 2: Creating Blueprint Prototypes

We designed four tasks to cover multiple use cases for *BPAIt*. 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 2×2 mixed factorial design. The independent variables and levels are listed in the Table 4-1 below:

Independent variable	Type	Level
System Order	Between-subject	<i>Unreal</i> first, <i>BPAIt</i> first
System	Within-subject	<i>Unreal</i> , <i>BPAIt</i>

Table 4-1: The independent variables with levels in the experimental design.

The independent within-subject variable was System (*Unreal*, *BPAIt*). 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 (*BPAIt* first, *Unreal* first) was the independent between-subject variable. Game type order and template order were randomized. See Table 4-2 for details. 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.

Participant #	System	Game Type	Game Template	System	Game Type	Game Template	System	Game Type	Game Template	System	Game Type	Game Template
1	<i>Unreal</i>	Block	Tetris	<i>Unreal</i>	FPS	Target	<i>BPAlt</i>	Block	Match3	<i>BPAlt</i>	FPS	Obstacle
2	<i>BPAlt</i>	Block	Match3	<i>BPAlt</i>	FPS	Obstacle	<i>Unreal</i>	Block	Tetris	<i>Unreal</i>	FPS	Target
3	<i>Unreal</i>	FPS	Target	<i>Unreal</i>	Block	Tetris	<i>BPAlt</i>	FPS	Obstacle	<i>BPAlt</i>	Block	Match3
4	<i>BPAlt</i>	FPS	Obstacle	<i>BPAlt</i>	Block	Match3	<i>Unreal</i>	FPS	Target	<i>Unreal</i>	Block	Tetris
5	<i>Unreal</i>	Block	Match3	<i>Unreal</i>	FPS	Obstacle	<i>BPAlt</i>	Block	Tetris	<i>BPAlt</i>	FPS	Target
6	<i>BPAlt</i>	Block	Tetris	<i>BPAlt</i>	FPS	Target	<i>Unreal</i>	Block	Match3	<i>Unreal</i>	FPS	Obstacle
7	<i>Unreal</i>	FPS	Obstacle	<i>Unreal</i>	Block	Match3	<i>BPAlt</i>	FPS	Target	<i>BPAlt</i>	Block	Tetris
8	<i>BPAlt</i>	FPS	Target	<i>BPAlt</i>	Block	Tetris	<i>Unreal</i>	FPS	Obstacle	<i>Unreal</i>	Block	Match3
9	<i>Unreal</i>	Block	Tetris	<i>Unreal</i>	FPS	Target	<i>BPAlt</i>	Block	Match3	<i>BPAlt</i>	FPS	Obstacle
10	<i>BPAlt</i>	Block	Match3	<i>BPAlt</i>	FPS	Obstacle	<i>Unreal</i>	Block	Tetris	<i>Unreal</i>	FPS	Target

Table 4-2: The independent variables and levels used in our experimental design. The table also demonstrates how we counterbalanced the experimental condition between participants to decrease the effect of learning.

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.

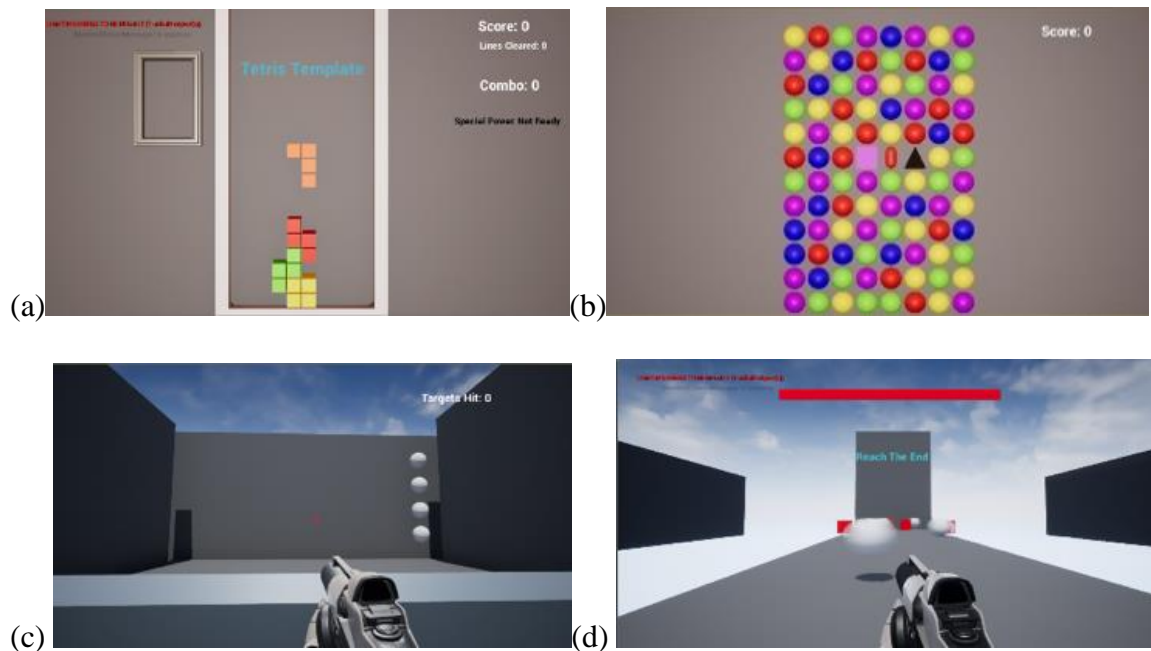


Figure 4-3: The four project templates used for the tasks: a) Tetris, b) Match 3, c) Target, d) Obstacles & Enemies.

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 4-3a.

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 4-3b.

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 4-3c.

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 4-3d.

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. In the templates given to the participants there already existed 4 copies of the same target, enemy, and obstacle Blueprint. This meant that participants were supposed to create alternatives for one of the possibilities per task (target, enemy/obstacle) and replace the existing copies in the level with their alternatives. This is to ensure that they end up with three alternatives and the original copy of the Blueprint.

The process of creating alternatives differed depending on both the system being used and the type of task that is being performed. When using *BPAIt*, 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 create alternative functionality for the blueprint. The participants would then simply rewire a given graph to test between 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 *BPAIt* 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 3-4b). The original gameplay object in the level is then automatically swapped to the alternative from the Blueprint Editor where “Play” was pressed.

After Each Task

After each task the participants were asked the following questions:

- How many alternatives did you make?
- Did your alternatives function as intended?
- What were your alternatives supposed to do?
- Do you think that your alternatives would be useable in a game similar to this? If not which ones?

The purpose of these questions was to see if the tasks were completed correctly, and to have easy reference to what the users did during the tasks for further analysis.

After Using Each System

After completing the tasks with each system, the participants completed the CSI's paired-factor comparison test in compliance with same task, tool comparison repeated measures designs [4]. They also completed the PANAS questionnaire to gauge their emotions after using each system for a total of three PANAS entries.

4.4.3 Phase 3: Interview and Debriefing

After completing all the tasks, 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 *BPAIt* which is discussed further in Section 5.3. After the interview, the participants were free to ask about the nature of the study. The whole procedure took on average 3 hours, with each task taking on average 20.57 minutes to complete.

4.5 Discussion

The reason that why participants were asked to perform these specific tasks was because in other HCI works, testing usability of systems followed a similar method of giving relevant tasks to participants, see e.g., [56]. However, in our case, we wanted to provide a proper example of how *BPAIt* can be used by the typical user of *Unreal* Blueprints. The FPS tasks were tailored towards developers that work with Blueprints to create full game object classes, and the Block tasks were tailored towards designers who were working on

gameplay elements given entry points by the programmers on the team, as this is an intended use of *Unreal* Blueprints.

4.6 Chapter 4 Summary

This chapter stated the following research hypotheses:

- **H1)** *Usability and user experience will either be the same or better using BPAIt compared to Unreal on its own.*
- **H2):** *Using design alternatives will increase levels of creativity of the users.*
- **H3):** *Using BPAIt will either be the same of better than traditional prototyping methods in terms of iteration time.*

The chapter then described the user study which was performed to test the hypotheses and describes the participants, apparatus and procedure.

The next chapter describes the results of the user study, breaking it down into the different evaluation methods.

Chapter 5

User Study Results

In this chapter we describe the results of the user study. We discuss all the methods of evaluation. We report the CSI Scores, feedback from the participants, and the PANAS scores. We analyze the data collected during the completion of the tasks, which includes the time it took the participants to complete the tasks and the number of tests that they did while completing the tasks. We also discuss the results from the expert evaluation done on all the alternatives created by the participants.

5.1 Creativity Support Index

Creativity Support Index (CSI) is a quantitative psychometric survey which assesses how well a system assists creativity in the design process [4]. Specifically, users provide ratings for six dimensions of creativity support: Enjoyment, Exploration, Expressiveness, Immersion, Results Worth Effort, and Collaboration. We used it to measure each participants' CSI score. Collaboration was not rated.

5.1.1 Assumption Tests

Before conducting a mixed ANOVA test on the results of the CSI survey we tested the assumptions of the test. There were no significant outliers in the data set as identified by

the boxplot function in R³ (Figure 5-1). There were also no significant results after running Shapiro-Wilk test for normality (Table 5-1). No significant results were also found after running Levene's test for homogeneity of independent variables (Table 5-2). Sphericity tests could not be conducted since there were only two levels in each independent variable. Thus, all the assumptions for mixed ANOVA test passed.

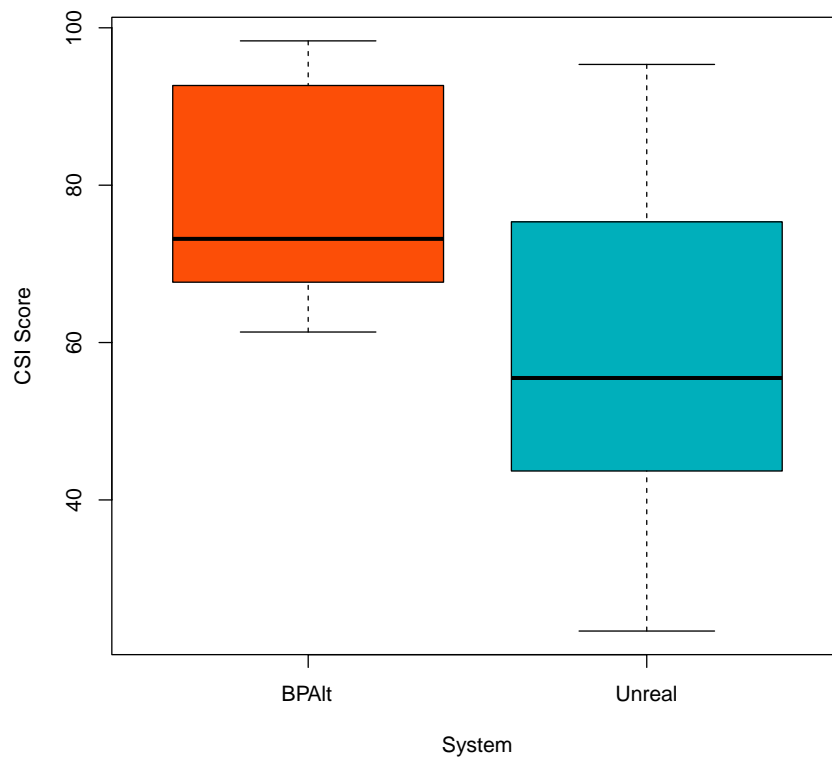


Figure 5-1: Box Plot of CSI Scores comparing BPAIt and Unreal by itself. No outliers were identified.

³ <https://www.r-project.org/>

System	System Order	W-Statistic	df	Sig.
<i>BPAIt</i>	<i>Unreal</i> first	0.910	5	0.468
<i>BPAIt</i>	<i>BPAIt</i> first	0.926	5	0.570
<i>Unreal</i>	<i>Unreal</i> first	0.977	5	0.920
<i>Unreal</i>	<i>BPAIt</i> first	0.944	5	0.694

Table 5-1: Shapiro-Wilk Normality Test of the CSI Score for System and System Order.

System	F	df1	df2	Sig.
<i>Unreal</i>	0.367	3	6	0.780
<i>BPAIt</i>	0.032	3	6	0.992

Table 5-2: Levene's Test of Equality of Error Variance of the CSI Score between *Unreal* and *BPAIt*.

5.1.2 Mixed ANOVA Test

We conducted a two-factor mixed ANOVA test. The main effect of System was significant, $F(1,8) = 8.35$, $p < 0.05$, $\eta^2_p = 0.27$. The CSI score for *BPAIt* ($M = 79.2$, $SD = 13.93$) was higher than for *Unreal* ($M = 57.3$, $SD = 21.75$). The main effect of System Order ($F(1,8) = 0.02$, ns) was not significant, indicating that counterbalancing was successful. The interaction effect between System and System Order was also not significant ($F(1,8) = 0.18$, ns). The breakdown of CSI survey results is shown in Table 5-3.

System	Factor/ Scale	Enjoyment	Exploration	Expressive ness	Immersion	Results Worth Effort
<i>Unreal</i>	Factor counts (SD)	3.3 (1.5)	4.1 (0.6)	2.6 (1.1)	1.5 (1.1)	3.0 (1.5)
<i>Unreal</i>	Factor Score (SD)	11.5 (2.8)	12.0 (2.4)	11.8 (2.3)	9.9 (1.7)	13.7 (1.7)

<i>Unreal</i>	Weighted Factor Score (SD)	38.0 (8.3)	49.2 (10.1)	30.7 (10.2)	14.9 (5.3)	41.1 (15.1)
<i>BPAIt</i>	Factor counts (σ)	3.3 (1.5)	4.1 (0.6)	2.6 (1.1)	1.5 (1.1)	3.0 (1.5)
<i>BPAIt</i>	Factor Score (SD)	17.6 (1.7)	17.3 (1.6)	15.4 (1.9)	11.0 (2.2)	15.5 (1.8)
<i>BPAIt</i>	Weighted Factor Score (SD)	58.1 (15.3)	70.9 (9.1)	40.0 (10.6)	16.5 (6.5)	46.5 (15.4)

Table 5-3: Average results of CSI Survey after using Unreal (top); BPAIt (bottom).

5.2 Feedback from Participants

We asked participants to rank how they felt using *Unreal* versus *BPAIt* on a 7-point Likert scale (1-lowest,7-highest) in terms of efficiency, ease of use, chance of future use and overall.

Classic inferential methods are not suitable for analysing data where assumptions have been violated as in the situation of Likert scales where the dependent variable is measured on the ordinal rather than continuous scale. Under general conditions these methods can have relatively poor power, yield inaccurate confidence intervals, and poorly characterize the extent groups differ [30]. Moreover, we conducted normality tests which revealed that normality was violated severely ($p < 0.0001$) for a number of combinations of the between and within subject variables in some of the ranked categories. Wilcoxon signed-ranked test would have been a suitable alternative to analyze the Likert scale

rankings of the systems. However, this test is only suitable for non-factorial designs. As a result, we used *robust mixed ANOVA*, a method recommended by Andy Field [9] (see p. 643) as a non-parametric alternative to mixed ANOVA. The method is available in WRS2 package for R [30]. For this analysis we used 10,000 bootstrap samples. The main effect of System Order was not significant for all ranked categories. The main effect of System was significant at either $\alpha = 0.1$ and 0.05 depending on the ranked category. Across all ranking categories *BPAlt* was consistently ranked higher, most importantly this includes the overall ranking, which was found significant ($\hat{\psi} = 1, p = 0.03$). The results for each ranked category are summarized in **Table 5-4** below.

Category	Efficiency		Ease of use		Chance of future use		Overall	
System	<i>Unreal</i>	<i>BPAlt</i>	<i>Unreal</i>	<i>BPAlt</i>	<i>Unreal</i>	<i>BPAlt</i>	<i>Unreal</i>	<i>BPAlt</i>
Median	5	6.5	5	6	4	5.5	5	6
Main effect of System Order (sppba)	$\hat{\psi} = 0.5,$ $p > 0.26$		$\hat{\psi} = -0.6,$ $p = 0.39$		$\hat{\psi} = -0.575,$ $p = 0.69$		$\hat{\psi} = 0.1,$ $p = 0.77$	
Main effect of System (sppbb)	$\hat{\psi} = 1.222,$ $p < 0.03^{**}$		$\hat{\psi} = 1.5,$ $p = 0.055^*$		$\hat{\psi} = 1,$ $p = 0.09^*$		$\hat{\psi} = 1,$ $p = 0.03^{**}$	
Interaction effect (sppbi)	$\hat{\psi} = -1,$ $p > 0.24$		$\hat{\psi} = 1,$ $p = 0.21$		$\hat{\psi} = 0,$ $p = 0.8$		$\hat{\psi} = 0,$ $p = 0.5$	

Table 5-4: Robust mixed ANOVA results on the participants' rankings of the systems. 10,000

bootstrap samples were used. *-significant at $\alpha=0.1$, **-significant at $\alpha=0.05$.

Diverging stacked bar charts are a recommended graphical display technique for Likert scale [17], which we used to display this data below in Figure 5-2.

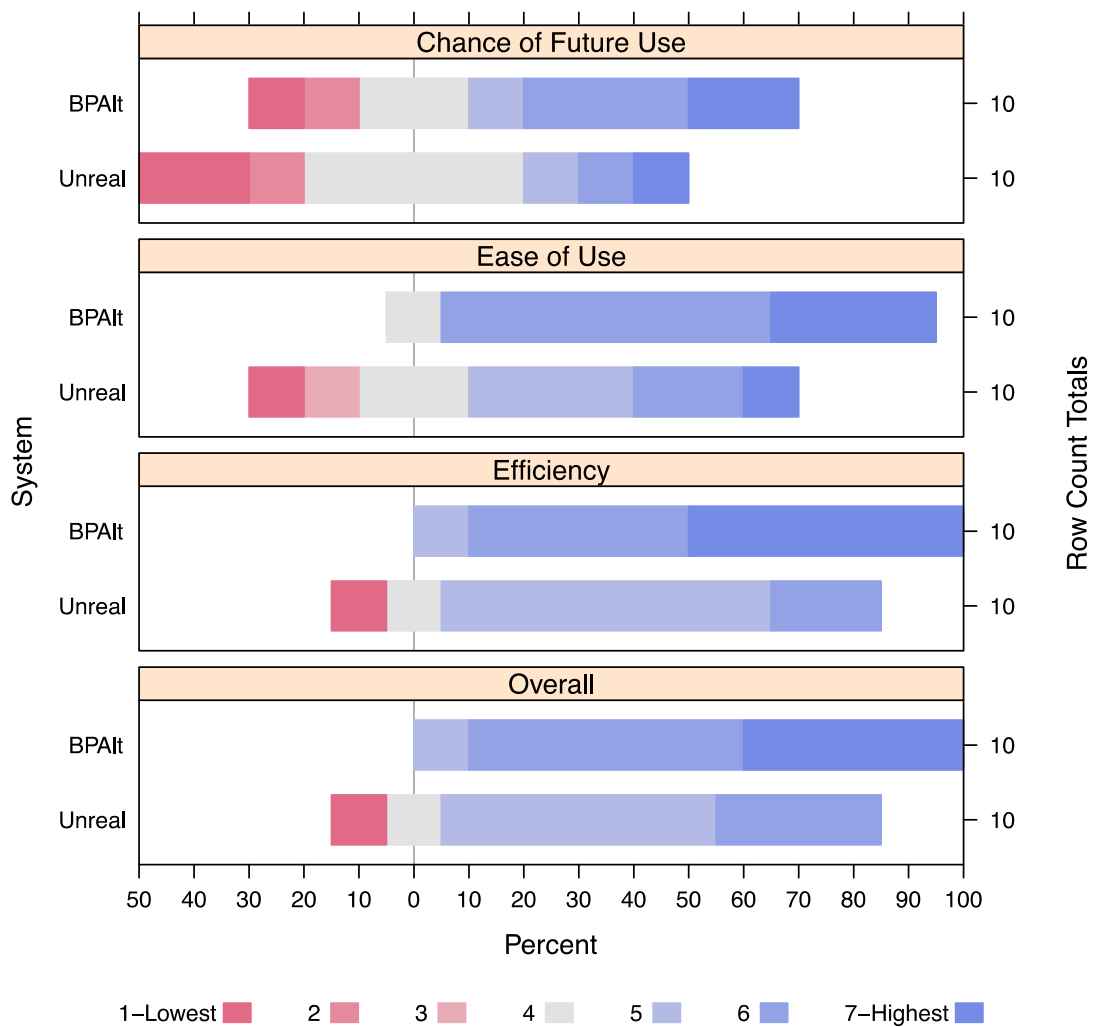


Figure 5-2: Participants' rankings of *Unreal* and *BPAIt*.

Here are some of the most mentionworthy comments from the freeform feedback that we received. After the first task which was with *BPAIt*, 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 *BPAIt* was well integrated into the system saying that they thought that it was a native function of the Unreal Engine.

5.3 Semi-Structured Interview

The goal of the interviews was to find out how the participants felt about *BPAIt*, and how the tasks affected their performance and feelings. In contrast to the freeform feedback, this interview was a more targeted approach to get a more detailed explanation through dialog with the participants.

During the interview we asked the following questions:

1. Did you find the interface for creating and swapping between alternatives (*BPAIt*) useful? (If yes, what did you like about it)?
2. While performing the tasks in this experiment, did you feel like there ever was a need for you to selectively merge a part of the Event Graph in one of the alternatives to another? Which task(s) was/were it/they, if so? Also, if so, do you believe that an interaction technique that could enable you to do this would be useful?
3. If the System Order was Unreal first: Do you believe that since you completed first two tasks without the alternatives plugin (*BPAIt*), this influenced your workflow during the last two tasks with the alternative plugin? If so, then how? Do you believe your results would be different if the order of the tasks was reversed? If so, then how?

If the System Order was *BPAIt* first: Do you believe that since you completed first two tasks using the alternatives plugin (*BPAIt*), this influenced your workflow during the last two tasks without the alternatives plugin? If so, then how? Do you believe your results would be different if the order of the tasks was reversed? If so, then how?
4. Did you find the alternatives plugin more useful for the Block task, FPS task, both tasks, or none of the tasks?

All the participants stated that *BPAIt* was helpful for the tasks. The main points the participants brought up were as follows:

- *BPAIt* 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*” *BPAIt* because the order of their user study featured *BPAIt* first followed by *Unreal*. We tested two different use cases for *BPAIt* (FPS vs. Block). *BPAIt* 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 *BPAIt* for the FPS tasks. P1, P3, P4 and P8 preferred it for the Block tasks. P5, P7, P10 thought that *BPAIt* was equally useful in both FPS and Block tasks. However, all the participants stated that they still thought that *BPAIt* was helpful for both types of tasks. While this underlines that the *BPAIt* can be used in vastly different situations and still be useful, in the hindsight, we wish we asked participants who preferred *BPAIt* for specific kind of tasks why they thought this was the case.

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 *BPAIt* was used first or not.

We asked participants if during the user study they felt like having the ability to selectively merge a part of the event graph in one of the alternatives to another would be useful. All participants showed interest in having selective merging being a feature, but three participants (P3, P6 and P8) stated that it would not be a necessary addition to *BPAIt*.

5.4 The Positive and Negative Affect Schedule (PANAS)

The Positive and Negative Affect Schedule (PANAS) [52] is a questionnaire which measures the positive and negative emotions of participants at the time of completion. Positive affect refers to positive emotions such as being alert or excited. Negative affect refers to negative emotions such as being upset or scared. PANAS consists of 20 questions, where 10 questions measure positive affect and the remaining 10 questions measure negative affect. Each question is in the form of a five-point Likert scale. The questionnaire can be found in Appendix B. To calculate the scores for both positive and negative affect, we add up the measures for the respective 10 questions corresponding to each affect type. This gives us scores between 10 and 50 for both positive and negative affect where a higher score represents a higher level of affect. The questionnaire was administered three times during the duration of the experiment, prior to performing either of the conditions, and after using each system. This gave us three snapshots of the participants' emotional state, which allowed us to measure the emotional affect after using the systems. We performed normality tests for each combination of between and within subject variable on both positive affect (PA) and negative affect (NA) scores. No violations of normality were observed. We checked for homoscedasticity using Lavenne's test. No violations of homoscedasticity were found. We performed a 2×3 mixed ANOVA on the positive and negative affect score sums separately. This is a standard practice in published research, which employs PANAS. The details of the experimental design are shown in Table 5-5.

Independent variable	Type	Level
Order	Between-subject	<i>Unreal</i> first, <i>BPAlt</i> first
Administration	Within-subject	Initial, After <i>Unreal</i> , After <i>BPAlt</i>

Table 5-5: The independent variables with levels in the PANAS analysis.

See Figure 5-3 for individual PANAS scores of participants.

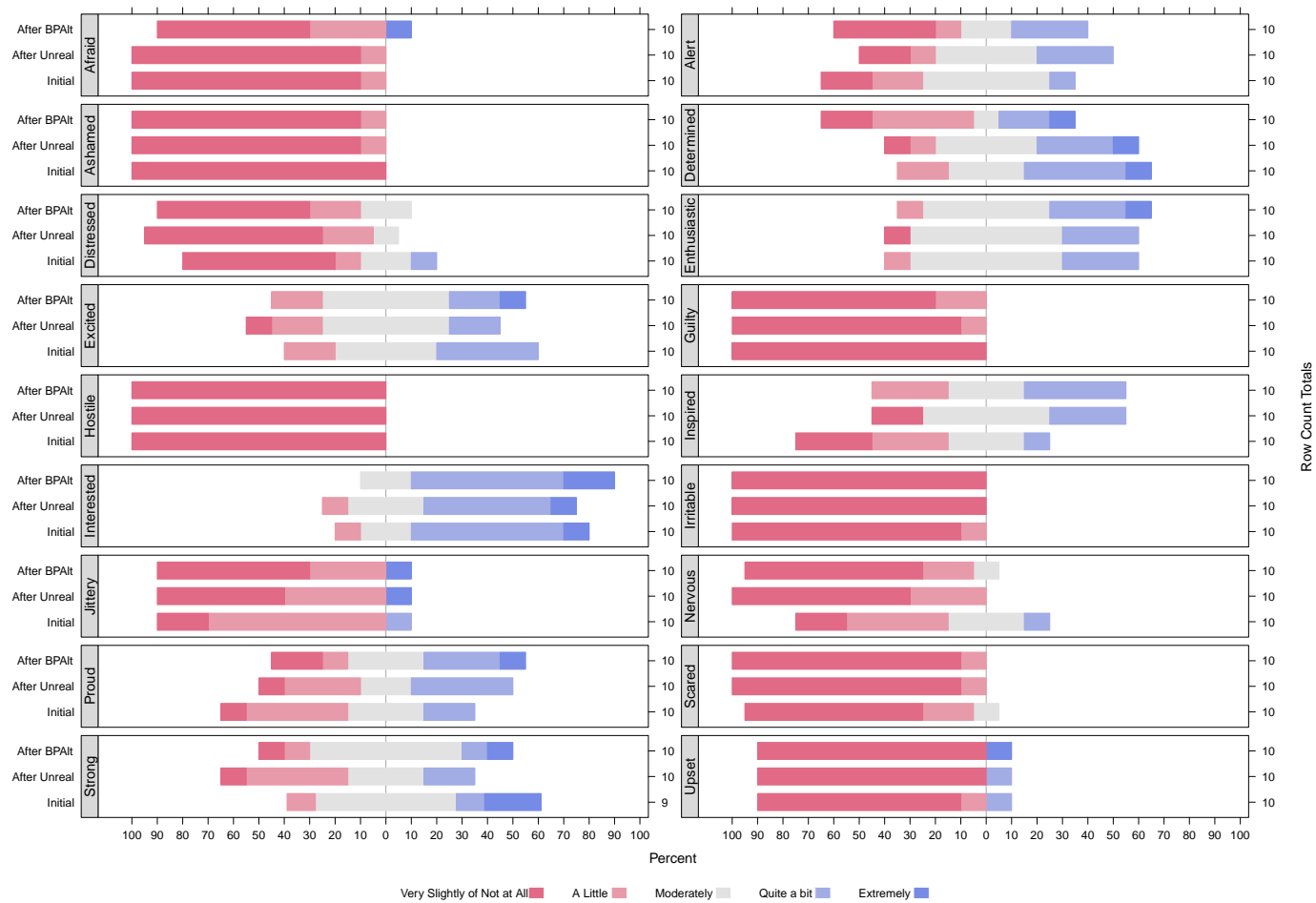


Figure 5-3: PANAS individual scores

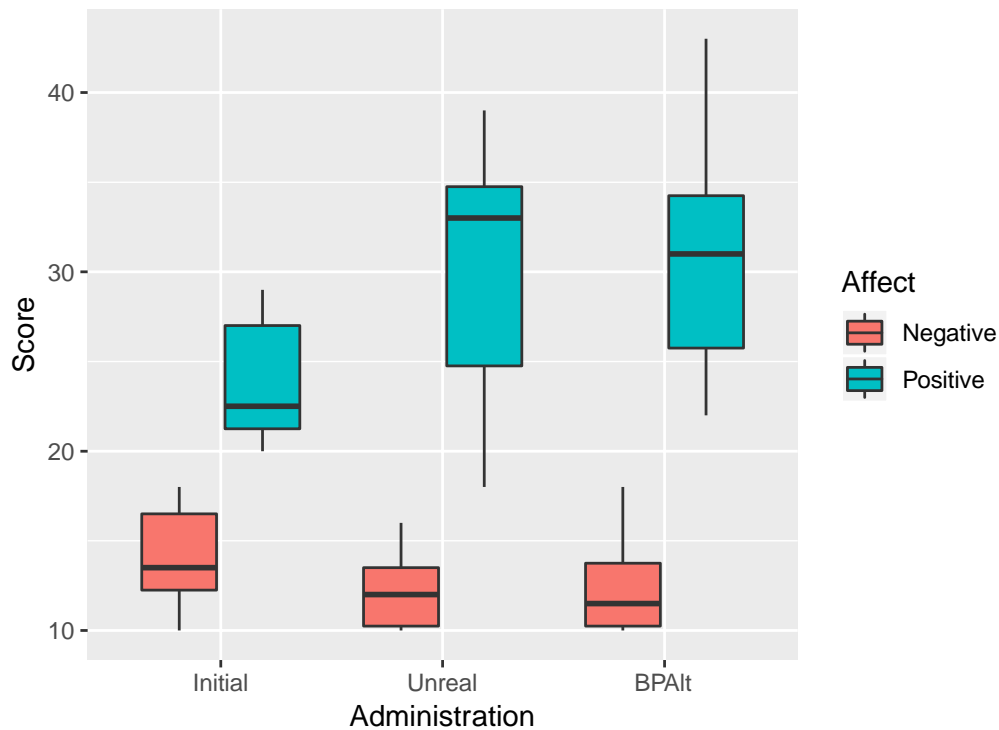


Figure 5-4: Box Plot of Positive and Negative Affect of mean PANAS Scores

The results of the systems are broken up to Positive Affect and Negative Affect for each condition. See Figure 5-4 for the affects after each condition. See Table 5-6 for the means and standard deviations of the sums of each emotional affect.

Positive Affect

The main effect of order of which the systems were introduced on total positive affect score was not significant ($F(1,8) = 0.07$, ns). The interaction with order and administration (Initial, After *Unreal*, After *BPAlt*) was also not significant, ($F(2,16) = 0.57$, ns). The main effect of administration was significant ($F(2,16) = 3.7$, $p < 0.01$, $\eta^2 = 0.25$). A pairwise post-hoc t-test with Bonferroni adjustments revealed that with the initial administration

($M = 23.8$, $SD = 3.5$) resulted in lower total positive affect score than both: administration after *Unreal* ($M = 30.2$, $SD = 6.5$), $p < 0.05$, and after *BPAIt* ($M = 30.9$, $SD = 6.7$), $p < 0.05$.

Negative Affect

The main effect of order of which the systems were introduced on total negative affect score was not significant ($F(1,8) = 0.29$, ns). The interaction with order and administration (Initial, After *Unreal*, After *BPAIt*) was also not significant, ($F(2,16) = 2.95$, $p > 0.05$, $\eta^2 = 0.07$). Mauchly's test indicated that the assumption of sphericity had been violated for Administration, $W = 0.35$, $p < .05$, therefore, the degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity ($\epsilon = 0.6$). After this correction the effect was not significant at $\alpha = 0.05$ ($F(1.2,9.6) = 3.59$, $p = 0.065$, $\eta^2 = 0.09$).

	Positive (M)	Positive (SD)	Negative (M)	Negative (SD)
Initial condition	23.80	3.584	14.00	2.708
After using <i>Unreal</i>	23.90	5.343	12.20	2.150
After using <i>BPAIt</i>	24.70	5.078	12.60	2.914

Table 5-6: Means and standard deviations of positive and negative PANAS sum scores.

5.5 Relevant Metrics

During the interviews, nine participants stated that they were more comfortable with the last two tasks regardless if *BPAIt* was used first or not. We looked further into this by analysing task completion times and number of play tests done by participants on the last two tasks. We found that it took less time per task and fewer tests on average using *BPAIt*

(time: $M = 33.6m$, $SD = 11.4 m$; tests: $M = 19.2$, $SD = 7.4$) compared to *Unreal* (time: $M = 38m$, $SD = 16.5m$, tests: $M = 25.6$, $SD = 13.7$). We conducted a One-Way ANOVA analysis, but the results were not significant (time: $F(1,8) = 1.73$, $p > 0.05$, tests: $F(1,8) = 1.52$, $p > 0.05$).

5.6 Expert Evaluation – Design Quality

In total there were 120 different alternatives created by the 10 participants. To evaluate the design quality of each task done we recruited an arm's length expert, James Robb, who is a teaching faculty member in game development at UOIT and who has been teaching game design and leading the game development workshops for several years, to rate each alternative for each task. Part of Mr. Robb's duties as an instructor for game development workshops is to grade the quality of video games of student teams. Our expert rated each alternative based on perceived creativity, effort, viability, and overall quality. The ratings were done on a scale of 0 – 10. Each set of tasks was shown to the expert at random to minimize any bias. The expert was given the intention of each task for each participant, which was taken from the questions answered in-between each task. Since our expert rated assigned grades with fractions, we assumed the data we collected was continuous. The means and standard deviations appear in Table 5-7. After the results were collected, we ran a Mann-Whitney U to investigate if System had any effect on the grades. We found that the main effect of the system was not significant for any graded category. See Table 5-8 for details. Grade breakdown per each participant are shown in Figure 5-5.

	<i>Unreal (M)</i>	<i>Unreal (SD)</i>	<i>BPAIt (M)</i>	<i>BPAIt (SD)</i>
Creativity	4.843	1.026	4.841	1.250
Effort	6.484	1.370	6.625	1.726
Viability	4.724	1.108	4.69	1.161
Overall	5.159	0.841	5.283	0.945

Table 5-7: Expert evaluation: means and standard deviations

	Mann-Whitney U Score	Z	Asymp. Sig. (2-tailed)
Creativity	1724.500	-0.084	0.933
Effort	1731.500	-0.046	0.963
Viability	1663.500	-0.417	0.677
Overall	1790.500	-0.051	0.959

Table 5-8: Expert Evaluation: Mann-Whitney U test results.

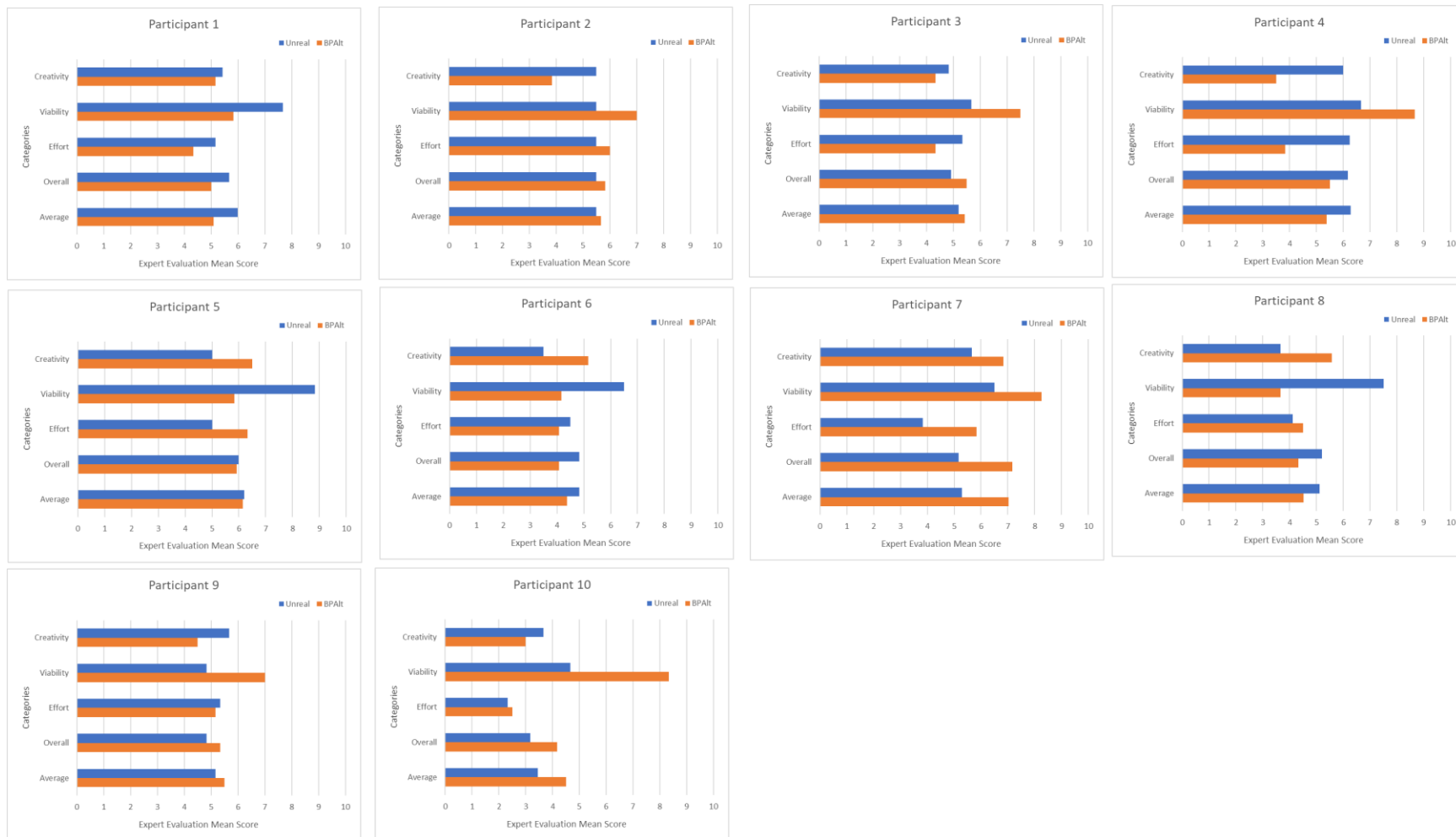


Figure 5-5: Expert evaluation: grades per participant

5.7 Discussion

During the interviews, nine participants stated that they were more comfortable with the last two tasks regardless if *BPAIt* was used first or not. Based on these comments we assumed that the first tasks would result in a learning effect. Luckily, we prepared for this by counterbalancing the order in which the participants were evaluating each of the two systems. All the statistical analyses showed that the order effect was not significant.

The mean CSI scores for *BPAIt* were higher than for *Unreal*. This difference was also found to be statistically significant. From the results of the post-study questionnaire we found that the participants preferred using *BPAIt* over using *Unreal* on its own in all ranked categories (efficiency, ease of use, chance of future use, and overall). These results were statistically significant with a varying significance level depending on the category. Thus, we were able to support our **Hypothesis 2** for user preference in favour of *BPAIt* rather is it comparable to *Unreal Engine* by itself. We are also able to support **Hypothesis 1** showing that participants felt that *BPAIt* was easy to use and useful in the existing system.

The analysis of PANAS ratings revealed that the positive affect increased for both systems compared to the initial condition. The negative affect slightly decreased in both systems compared to the initial condition, but the difference was not statistically significant. This result shows us that participants felt very similarly about both tools after using them, showing that *BPAIt* did not cause additional distress to the participants. The results of CSI combined with PANAS provide supporting evidence for **Hypothesis 1**.

The analyses of time required to complete tasks and number of play tests performed did not reveal any significant difference between the two systems. This shows that *BPAIt* is at least comparable to *Unreal Engine* by itself thus partially confirming **Hypothesis 3** and **Hypothesis 1** as we found no significant difference in terms of iteration time and number of play tests performed. Though the results were not significant there is an observable difference in the average times and tests in favour of *BPAIt*. We hypothesize that the lack of statistical significance is due to the short-term nature of the study. Participants were performing a total of four tasks each. The tasks were quite simple in nature, and so participants completed the tasks quickly regardless of the system. Since the tasks were open-ended and creative in nature the creative process will be unique for each participant leading to great variation in completion times and play tests, which is evident from the large standard deviations we obtained for time and number of play tests in the last two tasks participants performed. This underlines that methods like the CSI survey are a more reliable way to perform evaluation in user studies that involve open-ended tasks with time restrictions. Similarly, in the expert evaluation the results were found to be not significant. Thus, we were not able to support **Hypothesis 2** in terms of user performance.

Although some of the results we found were not significant, we believe that it is due to the limitations of the user study and with additional work more significant results may be seen. The expert evaluation, the general efficiency of performing tasks, and the difference between number of tests required to complete a task between the two systems would be more interesting, if we performed a longitudinal user study by participants with

more experience using the *Unreal Engine*. Giving them more time to complete a larger task over the course of a two-week period. In fact, this has successfully been recently applied in evaluation of other creativity support tools, see e.g., [18,19]

The limitations of the user study and what influence they might have had on the results are discussed further in Section 8.2.

5.8 Chapter 5 Summary

This chapter describes the evaluation of *BPAIt*. The CSI Scores showed that there is a clear preference from the participants towards *BPAIt*. The PANAS showed there was no significant difference between the systems in terms of emotional affect on participants, supporting **Hypothesis 1**. There was also no significant difference between the systems used in the expert evaluation of the tasks. However, the results of the user study showed that the participants preferred using *BPAIt* over using *Unreal* on its own. Thus, partially supporting **Hypothesis 2** for user preference but not for user performance. The analysis of the time it took the participants to complete each task and the number of tests each participant performed during the tasks showed there was no significant difference when comparing the two systems, thus not confirming **Hypothesis 3** but cementing **Hypothesis 1**. We believe that excluding the PANAS results, other insignificant results can be attributed to the circumstances of the user study.

The next chapter reiterates the discussion of the results of the study and addition points. As well as some changes to *BPAIt* made from user feedback.

Chapter 6

Overall Discussion and Conclusion

Here we discuss the design decisions made behind *BPAIt*, the findings of the user study and the implications for future work.

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 are considered to be more useful (see e.g., [35,53,54]) and could have potentially been beneficial to the users given tasks that took longer to complete.

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 *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 *BPAIt*. It is important to note that some participants that used *BPAIt* 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 *BPAIt* 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 *BPAIt* 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 we are confident that the functionality supported by *BPAIt* 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 (Efficiency, Ease of Use, Chance of Future Use, and Overall) participants felt more positive about *BPAIt* 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 *BPAIt*. 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.

The PANAS did not reveal any significant difference between *Unreal* and *BPAIt*. We can conclude that the two systems are comparable in terms of emotional impact, so that we can assume that *BPAIt* did not make participants feel worse while using it.

The remaining insignificant results indicate that our findings are inconclusive, and we attribute this to the nature of the study. The evaluation of the iteration speed and number of tests conducted ended up being not significant. However, we believe that this user study may not have been the best evaluation given that some of the participants were either not too familiar with using the *Unreal Engine* or had not used the engine regularly for some time. This would cause some of the times to be skewed. Most participants' longest task time was their first task, regardless of the system they were using which can be seen in Table 6-1. Regardless, no ordering effects were observed on participants' rankings of the systems or the CSI scores.

	Mean Time	Time SD	Mean Tests	Tests SD
First Task	27.9	9.99	15.8	10.19
Second Task	18.6	8.03	10.7	6.40
Third Task	18.6	7.29	9.2	5.35
Fourth Task	17.2	7.48	13.2	7.19

Table 6-1: Average time (in minutes) and number of tests for tasks regardless of task type.

Similarly, in the expert evaluation, the results were found to be not significant as well. We speculate that the results would differ if the participants had more experience using the *Unreal Engine* and if they were given more time to complete longer tasks. Ji-Young et al. [39] used a very similar form of expert evaluate their creativity support tool. Ji-Young et al. found their expert evaluation results to be positive. However, the circumstances of their evaluation were different. In their study they evaluated their participants after a tutorial period giving them more time to get comfortable using the software. They also used two longer tasks as opposed to the 4 shorter tasks that we used, which allowed the user's creativity to come out more through the work. Most importantly in their work, J-Young et al. evaluated drastically different systems.

During the user study multiple participants expressed that they appreciated the simple interface of *BPAIt* stating: “over complicating the system would take away from its usefulness” P3 and “I really liked that the tool was so straight forward to use” P2. The intention of the simple design of the tool was to be a non-intrusive extension to an existing system.

We believe the functionality that *BPAIt* 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 [16]. Although our user study focused specifically on the game development aspect of *Unreal*, the approach can also potentially be used in other applications such as industrial simulations, animated films, and in creating research tools.

6.1 New Features Implemented in Response to the Results of the User Study

During the semi-structured interview, we asked if participants thought having access to selective merging would be a useful feature and received a positive response. We have implemented selective merging of nodes between Blueprint alternatives for a future longitudinal study with professional game developers. 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. Evaluating this was beyond the scope of our user study, and we will evaluate this in the longitudinal study in the future as well.

6.1.1 Selective Merging

Selective merging involves moving nodes from a graph in one Blueprint alternative to another Blueprint alternative in the same alternative set. For a selective merge to take place there needs to be alternative(s) of a Blueprint to merge to. To show some of the base functionality of the selective merge a previous worked example with Tim in Figure 3-3 is used. Consider that Tim creates a new alternative to the “SpecialPower” Blueprint called “SpecialPower_Alternative 1” (Figure 6-1) and wants to merge over a portion of the Blueprint back over to the original Blueprint alternative (“SpecialPower”). Figure 6-1 presents the alternative to the Test Blueprint with some changes to the Blueprint graph.

Tim does not want to merge all the different nodes, so he selects a specific node from the graph (Figure 6-1c) and opens the selective merge menu (Figure 6-1a). He then selects the alternative he wants to perform the selective merge on.

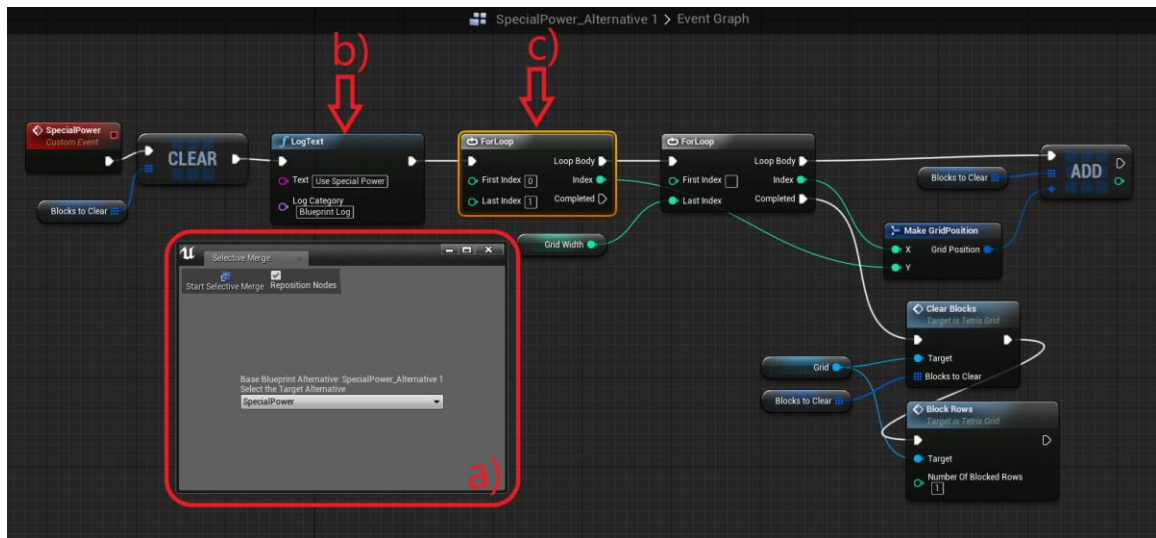


Figure 6-1: An alternative of the SpecialPower Blueprint “Special_Alternative 1” was created and changes were made. a) Selective merge menu b) Log Text node which does not exist in the “Test” Blueprint c) Selected FOR-LOOP node.

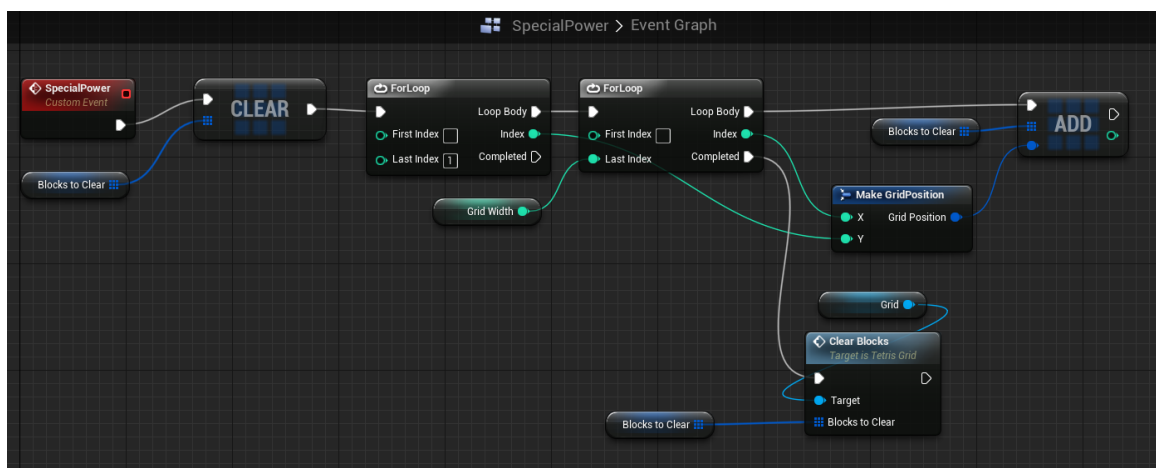


Figure 6-2: After selective merge completed in Test Blueprint.

Figure 6-2 presents the result of the selective merge where the FOR-LOOP 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 6-1 c). There was another node in the way (Figure 6-1 b). There is an algorithm in place to find the nearest applicable connection for merged nodes in the target Blueprint. This example demonstrated what the selective merge feature is supposed to be used for, but the system can handle much more complex situations. This can be seen in the following figures: Figure 6-3 Figure 6-4 Figure 6-5.

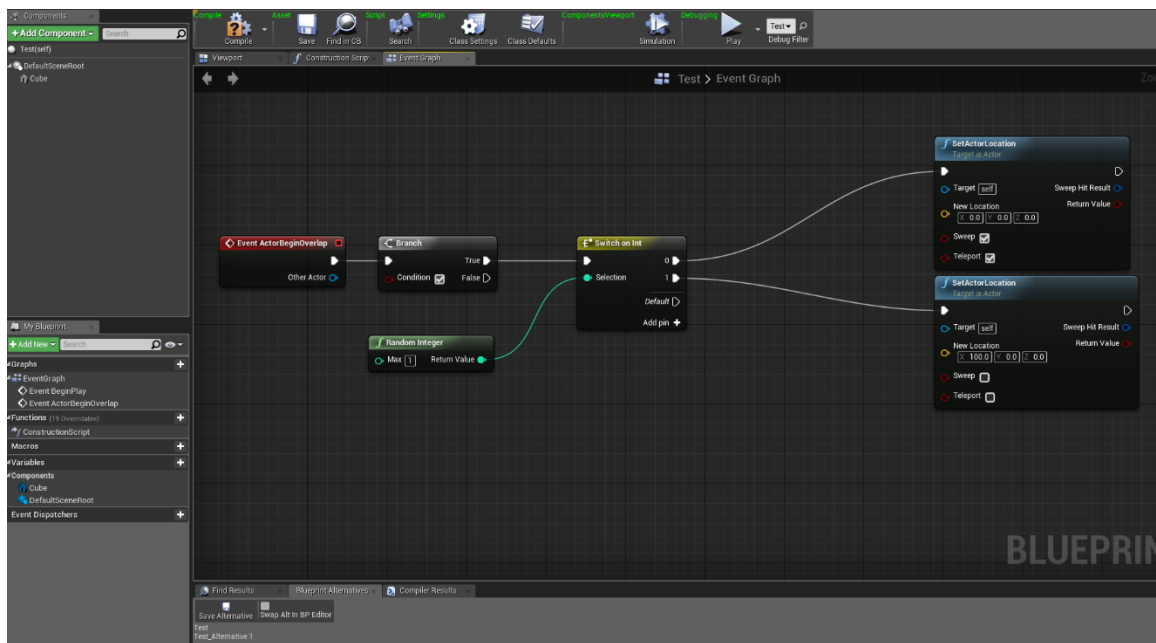


Figure 6-3: Test Blueprint before selective merge.

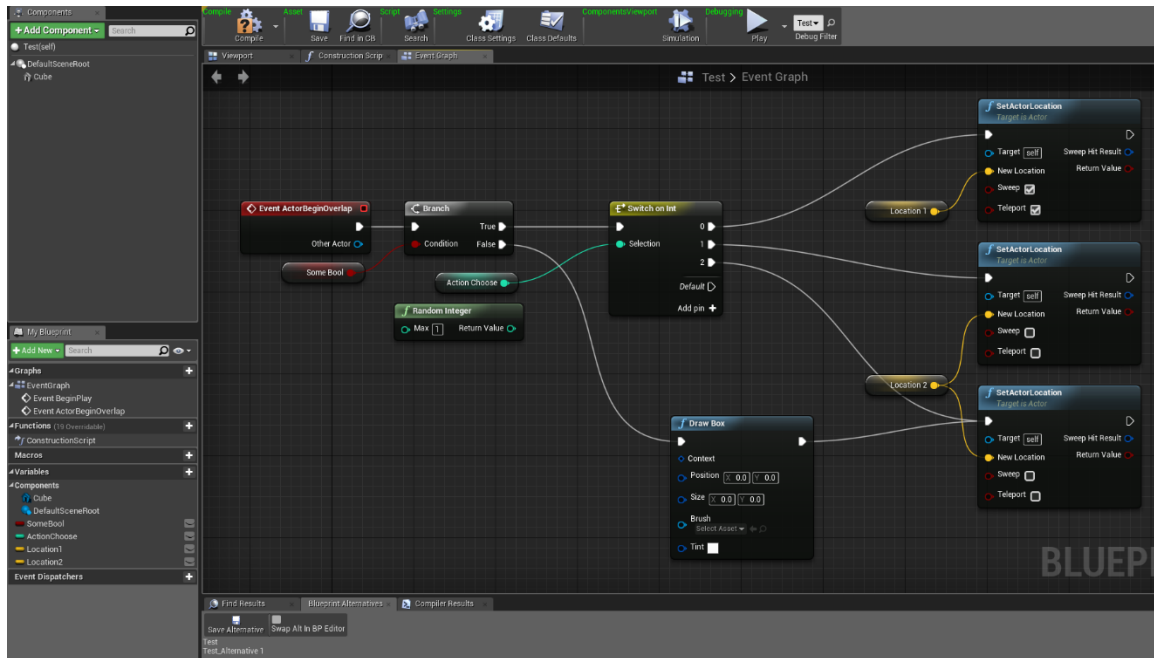


Figure 6-5: Test Blueprint after selective merge.

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” are merged to the target Blueprint “Test”. The “DrawBox” node was merged and placed accordingly. However, in this case there two more features are demonstrated: merging over variable nodes and dealing with nodes that already exist in the target Blueprint. Note that in Figure 6-3 and Figure 6-4 the Test Blueprint does not have many of the variables that Test_Alternative 1 has. After the merge (Figure 6-5) 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 6-3 and Figure 6-5, a change in the switch node and the top “Set Actor Location” node). See Section 7.2 for implementation details of Selective Merge. In a case in which there are no valid nodes for the merged nodes to connect to, the nodes are still merged over, but remain unconnected.

6.1.2 *Alternative Scenarios*

Alternative scenarios is a supplemental feature that we added to *BPAIt*, allowing the user to create and load alternative “scenarios”. 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. We plan to test the usability and usefulness of these features in the future work.

6.2 Chapter 6 Summary

This chapter discusses the overview of the user study and its findings. We found significant difference in the CSI scores in favour of *BPAIt*, proving that the participants showed preference for *BPAIt*. The PANAS survey results were inconclusive, both systems had a similar emotional affect on participants. The remaining results were found to be inconclusive. These include expert evaluations, the measurement of time it took the

participants to complete tasks, and the number of tests each participant performed. We attribute this to the nature of the study. We believe a study conducted with more experienced participants and tasks performed over long periods of time will yield significant results.

The chapter then describes some features that have been implemented to *BPAIt* since the user study. Blueprint node selective merging was added based on user feedback. The alternative scenarios feature was also added as an experimental feature.

Chapter 7

Implementation

In this section we discuss the techniques used to implement the features of *BPAIt*. This includes selectively merging nodes from one Blueprint alternative to another and creating and managing alternative scenarios of different alternatives to be used in each level. We also implemented alternative scenarios, which enable saving and loading different states of a level using Blueprint alternatives. Implementation details will be explained at a high level and accompanied by code examples.

7.1 Blueprint Alternatives

The Blueprint alternatives menu is attached to the Blueprint Editor and will vary for each Blueprint. When the user saves an alternative, they create a duplicate of the original Blueprint. All the information is collected via from the Blueprint Editor that the menu is attached to create the new alternative. Alternatives are saved into a 2D list of data structures which contain: a reference to the Blueprint object, the Blueprint's name and references to the other alternatives in the same alternative set. The first dimension of the list represents a list of alternative sets and the second dimension stores the individual Blueprint alternatives within each alternative set. This list is referenced wherever information regarding alternatives is relevant (e.g. swapping alternatives, creating the alternative menu, etc.). A UML representation of this list and the data stored can be seen in Figure 7-1.

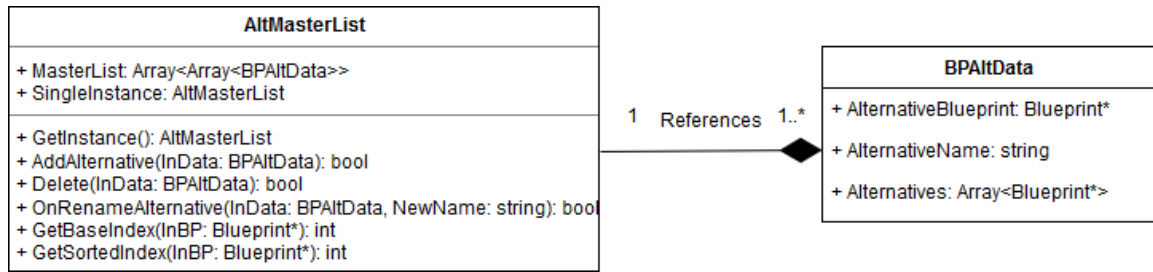


Figure 7-1: Blueprint alternative master list class and Blueprint alternative data class

7.1.1 Swapping between alternatives

Swapping between alternatives in the Level Editor involves replacing blueprint actors that have other Blueprint alternatives with an actor that is created from a selected Blueprint alternative. As discussed in Section 3.2 there are two different types of swapping from the details panel and directly from the Blueprint editor. Note that the user can choose whether to swap from the Blueprint Editor or not in the Blueprint alternatives menu in the Blueprint editor (Figure 7-2). We added a new feature added since the user study. This feature is the option to swap between all instances of actors in the opened level to the currently opened Blueprint editor via the PLAY button Figure 7-2a. In either method of swapping between Blueprint alternatives, the affected actor(s) in the scene are deleted and replaced with a new actor that has all the default values of the original actor but is an instance of the selected Blueprint alternative.

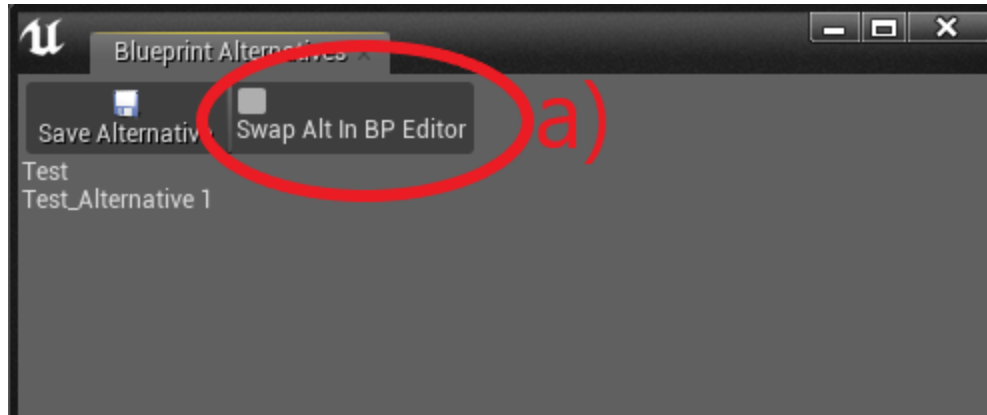


Figure 7-2: Blueprint Alternatives Menu a) Checkbox to toggle swapping between alternatives in the Blueprint editor using the Play button.

The details panel dropdown menu information is generated from the alternative set that the actor's Blueprint belongs to getting the names of the alternatives to populate the dropdown menu and the references to the Blueprint alternative classes to swap between them. When swapping between Blueprint alternatives in the Blueprint Editor using the play button, the system looks for any actors in the scene belonging to the same alternative set as the Blueprint being edited and swaps all of them to the Blueprint being edited right before play.

7.2 Selective Merging

The selective merge is the act of selecting some nodes from one graph of a Blueprint (Base Blueprint) and merging them to another Blueprint alternative within the same alternative set (Target Blueprint).

Here we outline how the current implementation of the selective merge is done. To give some context for implementation, each Blueprint class has a list of Blueprint graphs which can represent macros, functions, or event graphs. In each Blueprint graph there is a list of node objects that exist in the graph. Each node represents some functionality and has a list of pins which are the method of connecting to other nodes. Pins can represent either a value (bool, int, class object, etc.) or they can represent the flow of the code which are referred to as execution pins. Pins can have either input or output direction and can only connect to other pins of the opposite direction (input \rightarrow output, output \rightarrow input). Representations of Nodes and Node Pins can be seen in Figure 7-3.

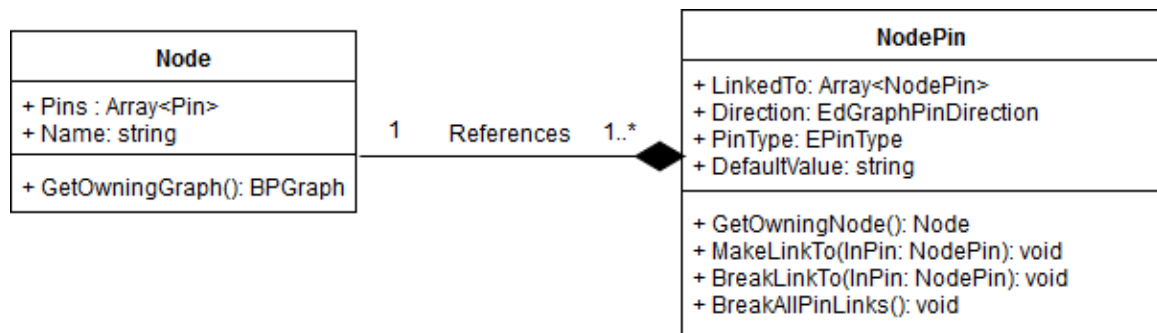


Figure 7-3: Classes for Blueprint graph nodes and node pins

The process of selectively merging in an ideal case with no exception cases is as follows:

1. Select the target Blueprint class.
2. Gather all the selected nodes in the base Blueprint.
3. Determine which nodes should be merged over.
4. Find the proper connections for all the nodes that are being merged over.

- a. Check for the nearest adjacent pin in the target Blueprint
 - b. If none exist check for other applicable actions.
5. Deal with any exceptions cases that need to be handled.
6. Paste over the merge nodes.
7. *Handle exception cases*
8. Connect nodes properly.

In *BPAIt* all nodes are given an ID and are kept track of in a master list of nodes.

Nodes that are shared across alternatives share the same ID which is essential for the merge process.

One of the most important steps in the selective merge is finding the proper pins to connect the merged nodes to. Using an algorithm to determine the appropriate connection we can find the proper pin to connect to. Once we do, we store the pin of the merged node that will connect, the node ID of the targeted node and the indexes of both of the pins in the respective merged and target nodes so that we can make the proper connections.

The important classes used in the selective merge are represented in a UML diagram as seen in Figure 7-4.

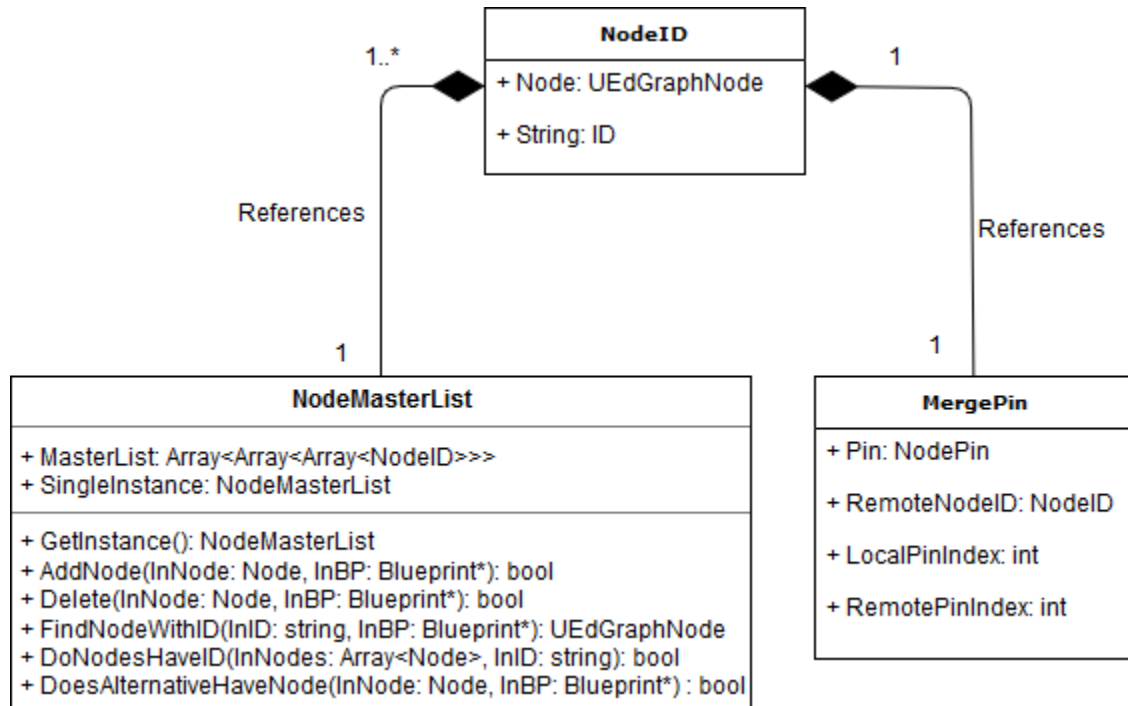


Figure 7-4: Important Selective Merge classes

Below is the pseudo-code to find the appropriate connection for the merged nodes in the target Blueprint.

```

Array<NodePin*> CheckForNearestAdjacentPin(NodePin* P, PinDirection
Dir)
{
    // New array of pins to
    TArray<NodePin*> APins;

    // Check to see if the pin IS NOT an execution pin
    if (P->PinType != PC_Exec)
    {
        for (auto LPin : P->LinkedTo)
        {
            if (AltNodeMasterList->DoesAlternativeHaveNode(
                LPin->GetOwningNode(), TargetBP))
            {
                APins.Add(LPin);
            }
        }
        return APins;
    }

    // If it IS a flow pin
    else
    {
        if (Dir == EGPD_Input)
        {
            // Check all the linked pins
            for (auto LinkedPin : P->LinkedTo)
            {
                Node* LinkedNode = LinkedPin->GetOwningNode();

                // Check all the pins for the linked node
                for (auto LinkedNodePin : LinkedNode->Pins)
                {
                    // Skip non-execution pins
                    if (LinkedNodePin->PinType.PinCategory !=
                        K2Schema->PC_Exec)
                        continue;

                    // Get the node that is connected to the node that you
                    // are already connected to
                    Node* LinkedNodePinNode =
                        LinkedNodePin->GetOwningNode();

                    if (NodeMasterList->
                        DoesAlternativeHaveNode(LinkedNodePinNode, TargetBP))
                    {

```

```

        // if we are looking at a pin with the wrong
        // direction just look at the next
        if (LinkedNodePin->Direction == P->Direction)
            continue;

        // if the target blueprint does have the node, add it
        // to the list to return.
        APins.Add(LinkedNodePin);
    }
    // If there is not an immediate connection available
    else
    {
        // if we are looking at a pin with the wrong
        // direction just look at the next
        if (LinkedNodePin->Direction != P->Direction)
            continue;

        // Check again through the next pin over recursively
        TArray<NodePin*> DeeperPins =
            CheckForNearestAdjacentPin(LinkedNodePin, Dir);

        for (auto DeeperPin : DeeperPins)
        {
            FString DeeperID = NodeMasterList->
                FindNodeID(DeeperPin->GetOwningNode());
            if (DeeperPin != NULL &&
                !AltNodeMasterList->DoNodesHaveID(
                    Editor->GetSelectedNodes(), DeeperID))
            {
                APins.Add(DeeperPin);
            }
        }
    }
}
return APins; // Return the list of pins found
}
else if (Dir == EGPD_Output)
{
    // Check all the linked pins
    for (auto LinkedPin : P->LinkedTo)
    {
        Node* LinkedNode = LinkedPin->GetOwningNode();
        for (auto LinkedNodePin : LinkedNode->Pins)
        {
            // if we are looking at a non-execution pin look at the
            // next pin
            if (LinkedNodePin->PinType.PinCategory

```

```

        != K2Schema->PC_Exec)
        continue;

// Get the node that is connected to the node that you
// are already connected to
Node* LinkedNodePinNode =
    LinkedNodePin->GetOwningNode();

// if the target blueprint does have the node with the
// same ID add it to the list
if (NodeMasterList->DoesAlternativeHaveNode(
    LinkedNodePinNode, TargetBP))
{
    // if we are looking at a pin with the wrong
    // direction just look at the next pin
    if (LinkedNodePin->Direction == P->Direction)
        continue;

    APins.Add(LinkedNodePin);
    break;
}
else
{
    // if we are looking at a pin with the wrong
    // direction just look at the next pin
    if (LinkedNodePin->Direction != P->Direction)
        continue;

    // Try to find an applicable pin by looking at the
    // next node over. (recursive)
    TArray<NodePin*> DeeperPins =
        CheckForNearestAdjacentPin(LinkedNodePin, Dir);

    for (auto DeeperPin : DeeperPins)
    {
        // Make sure that the found pins are valid
        FString DeeperID = NodeMasterList->
            FindNodeID(DeeperPin->GetOwningNode());

        if (DeeperPin != NULL &&
            !NodeMasterList->DoNodesHaveID(
                Editor->GetSelectedNodes(), DeeperID))
        {
            APins.Add(DeeperPin);
        }
    }
}
}
}

```

```
        }  
        return APins; // Return the list of pins found  
    }  
}  
return TArray<NodePin*>(); // Return empty array if we get here  
}
```

There is some chance the segment of Blueprint nodes that the user is trying to merge will be connected to different “root nodes”. Root nodes are defined as nodes which have no input executable pins but have output executable pins. They are usually event or function entry nodes. Finding the root node is essential if the merged nodes do not have a proper connection in the target Blueprint. The merged nodes can either generate the same root node if it does not exist in the target Blueprint. If the root node exists, connect to the root node. Below is the pseudo code for finding the root node for a given node.

```

Node* CheckForRootNode(Node* N, NodePin* P)
{
    for (auto Pin : N->Pins) // Check all of the Pins of the given node
    {
        if (Pin->Direction == EGPD_Input)
        {
            // Check all the linked pins
            for (auto LinkedPin : Pin->LinkedTo)
            {
                Node* LinkedNode = LinkedPin->GetOwningNode();
                // Check to see if the linked node is the root node
                if (LinkedObject->IsRootNode())
                    return LinkedNode;

                // If the linked node is any node types that are specified,
                // check the next pin
                else if (LinkedNode->HasOnlyOutputPins() &&
                        !LinkedNode->IsRootNode())
                    continue;

                // Is this a node that we can keep looking through?
                else
                {
                    // Recursively look for the Root Node
                    Node* DeeperNode = CheckForRootNode(LinkedNode, P);

                    if (DeeperNode != NULL)
                    {
                        P = LinkedPin;
                        return DeeperNode;
                    }
                }
            }
        }
        else
        {
            // If the node is already a root node just return it.
            if (N->IsRootNode())
                return N;

            continue;
        }
    }
    return NULL; // If there is no
t a root node return NULL
}

```

Since the *Unreal* Blueprints system is so well featured, there are many exception cases that must be dealt with when attempting to perform a selective merge. Some of the important examples (but not all) of these exception cases with potential solutions can be seen in Table 7-1.

<u>Exception Cases</u>	<u>Potential Solutions</u>
Variable nodes that are going to be merged into a target Blueprint which does not have the member variable that the variable nodes are referring to.	Create a new variable in the target Blueprint if it does not already exist and set the default value to be the same.
Custom Events, Function and Macro nodes that are going to be merged and the actual functions/macros being referenced do not exist in the target Blueprint alternative.	Copy the custom events, functions, and macros from the base Blueprint to the target Blueprint then add the custom event, function, macro nodes as required.
Blueprint components referenced in a Blueprint graph that do not exist in the target Blueprint.	Create the same components in the target Blueprint.
Function and Macro graphs have different properties from the event graph (parameters, return nodes, local variables) and they must be accounted for.	Need to accommodate for each difference individually.

Merging from a graph that does not exist in the target Blueprint.	Create the graph in the target Blueprint.
--	---

Table 7-1: Selective Merge Exception Cases

7.3 Chapter 7 Summary

This chapter describes the implementation of the key features of *BPAIt* including how the Blueprint alternatives are stored, swapping between Blueprint alternatives and how selective merging is done. Code examples are provided for some of the key algorithms ran during the selective merging process.

Chapter 8

Future Work and Limitations

8.1 BPAIt

Future work will include improving *BPAIt* and its current features, creating new systems within the *Unreal Engine* to find additional value of using alternatives in different areas of game development, generalizing the benefits of using alternatives to different areas of game development.

Add functionality to the existing features of the system and changing the interface based on user feedback can be explored in the future. Some features that can be added to the alternatives system include supporting history of all alternative Blueprints activity (creation, deletion, merging), adding a visualization of the history and adding functionality to revert alternatives and alternative sets to previous states. To compliment the Alternative Scenario system, creating a way to preview and edit multiple level scenarios at a time could help with the development process. Adding more options to help the user customize exactly how they want the merge to be done using the already created selective merge can also be explored in the future.

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, Behaviour Trees, Widget, and Sound. These subsystems use variations of Blueprint visual scripting system making the translation of *BPAIt* to other node-based interfaces within *Unreal* quite straight forward.

Future studies can also focus on investigating if the benefits of alternatives are transferable to these other visual scripting systems of *Unreal*.

In our research we only investigated the use of *BPAIt* in the context of game development. One could also test for the effects of using design alternatives on development of e.g., simulations, animations and research tools using the *Unreal Engine*. We also wanted to create a Blueprint library (a collection of Blueprint nodes) that would be able to communicate Blueprint alternative data to Blueprints to account for specific development scenarios. Examples include implementing a Blueprint node that returns all actors in an opened level that shares the same alternative set, implementing another node to trigger a commonly named event among actors in the same alternative set. This would enable users to create side-by-side alternative testing scenarios more easily.

There are features that could potentially be introduced into *BPAIt* in the future that have been done in other works on design alternative in other domains. The *MACE* extension [55] to *GEM-NI* enables interactive comparisons of more than two alternatives using active and subtractive encodings for difference visualizations. Difference visualization based on the additive encodings in *MACE* could be implemented to compare different alternatives of a Blueprint. The concept of subjunctive nodes has been introduced in *Shiro* [12] – 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 [29]—allow expressing both alternative values for properties and alternative computations for specifying parametric systems containing alternatives

thus providing a multi-state document model [12]. In the future, subjunctive nodes could be implemented in *Unreal* and compared against *BPAlt* in a user study.

8.2 Limitations of the User Study

Since some of the participants in the user study knew the researcher in varying capacities prior to participating in the study. We acknowledge that there is a possibility of bias from the participants.

We performed a power analysis to find the ideal number of participants given that we were looking for a power of 0.8 ($1 - \beta = 0.8$), and $\alpha = 0.05$ with an effect size of 0.26. The ideal number of participants was determined to be 32 but given the nature of the study finding and testing that many participants was unfeasible. After putting out the call for participants we ended up with 10 participants, so the results found should be considered preliminary ($1 - \beta = 0.32$). However, we still found the effect size to be desirable.

In the interview portion of the user study the questions could be seen as leading since we did not give them an opportunity to give negative comments via the questions. In hindsight, we wish we have included negatively constructed questions to force participants to provide negative feedback. However, we did allow participants to give freeform feedback in case they had any criticisms. Regardless we recognize this as a limitation of the interview questions.

While the user study that we conducted gave us good results in terms of user experience and the support of creativity there were several limitations. The participants all had experience making games or using game development engines, but many did not have

much experience using the *Unreal Engine* or systems like *Unreal Blueprints*. We speculate that the results would more likely be positive in favour of *BPAIt* if the systems were compared in a situation where the participants would already be familiar with the *Unreal Blueprints* system and are given more complex tasks that take longer to complete.

Based on the limitations of the first user study, we will be conducting a two-week longitudinal study on *BPAIt* with developers who have more experience using the *Unreal Engine* to help verify the value of the tool, potentially with a few changes to the tool based on user feedback. Specifically, we want to test the potential of using the selective merge feature, since participants from the previous study showed interest in it. We will be seeking out an expert with experience working with the *Unreal Engine* regularly in a professional manner. The study will be structured similarly to the first user study where the expert participant will be given a task to complete. Four meetings would take place between the researchers and the expert participant during the study to gather feedback and data. With this study we can gain a more accurate results in terms of user experience and usability.

Bibliography

1. Ömer Akin. 1978. How do architects design? *Artificial Intelligence and Pattern Recognition in Computer Aided Design*: 65–103.
2. Ömer Akin. 2001. Chapter 6 - Variants in Design Cognition. In *Design Knowing and Learning: Cognition in Design Education*, Charles M. Eastman, W. Michael McCracken and Wendy C. Newstetter (eds.). Elsevier Science, Oxford, 105–124. <https://doi.org/10.1016/B978-008043868-9/50006-1>
3. Carlo Bueno, Sarah Crossland, Christof Lutteroth, and Gerald Weber. Rewriting History: More Power to Creative People. In *OzCHI 2011*, 62–71.
4. Erin Cherry and Celine Latulipe. 2014. Quantifying the Creativity Support of Digital Tools Through the Creativity Support Index. *TOCHI 2014* 21, 4: 21:1–21:25. <https://doi.org/10.1145/2617588>
5. Verina Cristie and Sam Joyce. 2017. Capturing And Visualising Parametric Design Flow Through Interactive Web Versioning Snapshots. In *IASS Annual Symposium 2017 “Interfaces - Architecture. Engineering. Science.”*.
6. Heather Desurvire and Charlotte Wiberg. 2009. Game Usability Heuristics (PLAY) for Evaluating and Designing Better Games: The Next Iteration. In *Online Communities and Social Computing* (Lecture Notes in Computer Science), 557–566.
7. Maher Elkhaldi and Robert Woodbury. 2015. Interactive Design Exploration with Alt.Text. *International Journal of Architectural Computing* 13, 2: 103–122. <https://doi.org/10.1260/1478-0771.13.2.103>
8. Magy Seif El-Nasr. 2013. *Game analytics: maximizing the value of player data*. Springer, New York.
9. Andy Field, Jeremy Miles, and Zoe Field. 2012. *Discovering Statistics Using R*. SAGE Publications Ltd, London ; Thousand Oaks, Calif.
10. Jun Fujima, Aran Lunzer, Kasper Hornbæk, and Yuzuru Tanaka. 2004. Clip, connect, clone: combining application elements to build custom interfaces for information access. In *Proceedings of the 17th annual ACM symposium on User interface software and technology* (UIST '04), 175–184. <http://doi.acm.org.proxy.lib.sfu.ca/10.1145/1029632.1029664>
11. Tracy Fullerton. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games, Fourth Edition*. CRC Press.
12. Jeffrey Robert Guenther. 2016. Shiro - A language to represent alternatives. Simon Fraser University. Retrieved October 28, 2017 from <http://summit.sfu.ca/item/17048>
13. Joshua Hailpern, Erik Hinterbichler, Caryn Leppert, Damon Cook, and Brian P. Bailey. 2007. TEAM STORM: Demonstrating an Interaction Model for Working with Multiple Ideas During Creative Group Work. In *Proceedings of the 6th ACM SIGCHI Conference on Creativity & Cognition* (C&C '07), 193–202. <https://doi.org/10.1145/1254960.1254987>

14. Björn Hartmann, Sean Follmer, Antonio Ricciardi, Timothy Cardenas, and Scott R Klemmer. 2010. d.note: revising user interfaces through change tracking, annotations, and alternatives. In *CHI 2010* (CHI '10), 493–502.
<https://doi.org/10.1145/1753326.1753400>
15. Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R Klemmer. 2008. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *UIST 2008* (UIST '08), 91–100.
<http://doi.acm.org.proxy.lib.sfu.ca/10.1145/1449715.1449732>
16. Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R. Klemmer. 2008. Design As Exploration: Creating Interface Alternatives Through Parallel Authoring and Runtime Tuning. In (UIST '08), 91–100.
<https://doi.org/10.1145/1449715.1449732>
17. Richard Heiberger and Naomi Robbins. 2014. Design of Diverging Stacked Bar Charts for Likert Scales and Other Applications. *Journal of Statistical Software* 57, 1: 1–32. <https://doi.org/10.18637/jss.v057.i05>
18. Jennifer Jacobs, Joel Brandt, Radomír Mech, and Mitchel Resnick. 2018. Extending Manual Drawing Practices with Artist-Centric Programming Tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI '18), 590:1–590:13. <https://doi.org/10.1145/3173574.3174164>
19. Jennifer Jacobs, Sumit Gogia, Radomír Měch, and Joel R. Brandt. 2017. Supporting Expressive Procedural Art Creation Through Direct Manipulation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (CHI '17), 6330–6341. <https://doi.org/10.1145/3025453.3025927>
20. Klaus P. Jantke, Aran Lunzer, and Jun Fujima. 2005. Subjunctive Interfaces in Exploratory e-Learning. In *Proceedings of the Third Biennial Conference on Professional Knowledge Management* (WM'05), 176–188.
https://doi.org/10.1007/11590019_21
21. Rubaiat Habib Kazi, Tovi Grossman, Hyunmin Cheong, Ali Hashemi, and George Fitzmaurice. 2017. DreamSketch: Early Stage 3D Design Explorations with Sketching and Generative Design. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (UIST '17), 401–414.
<https://doi.org/10.1145/3126594.3126662>
22. Siniša Kolarić, Halil Erhan, and Robert Woodbury. 2017. CAMBRIA: Interacting with Multiple CAD Alternatives. In *Computer-Aided Architectural Design. Future Trajectories* (Communications in Computer and Information Science), 81–99.
https://doi.org/10.1007/978-981-10-5197-5_5
23. C. Larman and V. R. Basili. 2003. Iterative and incremental developments. a brief history. *Computer* 36, 6: 47–56. <https://doi.org/10.1109/MC.2003.1204375>
24. Aran Lunzer. 2004. Benefits of Subjunctive Interface Support for Exploratory Access to Online Resources. In *Proceedings of the 2004 International Conference on Intuitive Human Interfaces for Organizing and Accessing Intellectual Assets* (IHI'04), 14–32. https://doi.org/10.1007/978-3-540-32279-5_2

25. Aran Lunzer and Kasper Hornbæk. 2003. Side-by-side display and control of multiple scenarios: Subjunctive interfaces for exploring multi-attribute data. *Proceedings of OzCHI 2003*: 26–28.
26. Aran Lunzer and Kasper Hornbæk. 2004. Usability Studies on a Visualisation for Parallel Display and Control of Alternative Scenarios. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '04)*, 125–132. <https://doi.org/10.1145/989863.989882>
27. Aran Lunzer and Kasper Hornbæk. 2006. RecipeSheet: Creating, Combining and Controlling Information Processors. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06)*, 145–154. <https://doi.org/10.1145/1166253.1166276>
28. Aran Lunzer and Kasper Hornbæk. 2006. An Enhanced Spreadsheet Supporting Calculation-Structure Variants, and Its Application to Web-Based Processing. In *Federation over the Web (Lecture Notes in Computer Science)*, 143–158.
29. Aran Lunzer and Kasper Hornbæk. 2008. Subjunctive Interfaces: Extending Applications to Support Parallel Setup, Viewing and Control of Alternative Scenarios. *ACM TOCHI* 14, 4: 17:1–17:44. <https://doi.org/10.1145/1314683.1314685>
30. Patrick Mair and Rand Wilcox. Robust Statistical Methods Using WRS2. In *The WRS2 Package*.
31. J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. 1997. Design galleries: a general approach to setting parameters for computer graphics and animation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97)*, 389–400. <https://doi.org/10.1145/258734.258887>
32. Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. 2018. Dream Lens: Exploration and Visualization of Large-Scale Generative Design Datasets. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*, 369:1–369:12. <https://doi.org/10.1145/3173574.3173943>
33. Alexandre Menezes and Bryan Lawson. 2006. How designers perceive sketches. *Design Studies* 27, 5: 571–585. <https://doi.org/10.1016/j.destud.2006.02.001>
34. Pejman Mirza-Babaei, Lennart E. Nacke, John Gregory, Nick Collins, and Geraldine Fitzpatrick. 2013. How Does It Play Better?: Exploring User Testing and Biometric Storyboards in Games User Research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*, 1499–1508. <https://doi.org/10.1145/2470654.2466200>
35. Arefin Mohiuddin, Robert Woodbury, Narges Ashtari, Mark Cichy, and Völker Mueller. 2017. A Design Gallery System: Prototype and Evaluation. In *ACADIA 2017: DISCIPLINES & DISRUPTION [Proceedings of the 37th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA) ISBN 978-0-692-96506-1] Cambridge, MA 2-4 November, 2017*, pp. 414–425. Retrieved

- December 16, 2018 from http://papers.cumincad.org/cgi-bin/works/Show?acadia17_414
36. Tamara Munzner, François Guimbretière, Serdar Tasiran, Li Zhang, and Yunhong Zhou. 2003. TreeJuxtaposer: scalable tree comparison using Focus+Context with guaranteed visibility. *SIGGRAPH 2003* 22, 3: 453–462. <https://doi.org/10.1145/882262.882291>
 37. Lennart E. Nacke. 2013. An Introduction to Physiological Player Metrics for Evaluating Games. In *Game Analytics: Maximizing the Value of Player Data*, Magy Seif El-Nasr, Anders Drachen and Alessandro Canossa (eds.). Springer London, London, 585–619. https://doi.org/10.1007/978-1-4471-4769-5_26
 38. Mark W. Newman and James A. Landay. 2000. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In *DIS 2000* (DIS '00), 263–274. <https://doi.org/10.1145/347642.347758>
 39. Ji-Young Oh, Wolfgang Stuerzlinger, and John Danahy. 2006. SESAME: towards better 3D conceptual design systems. In *Proceedings of the 6th ACM conference on Designing Interactive systems - DIS '06*, 80. <https://doi.org/10.1145/1142405.1142419>
 40. Jasper O’Leary, Holger Winnemöller, Wilmot Li, Mira Dontcheva, and Morgan Dixon. 2018. Charrette: Supporting In-Person Discussions Around Iterations in User Interface Design. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI '18), 535:1–535:11. <https://doi.org/10.1145/3173574.3174109>
 41. Adrian Ramcharitar and Robert J. Teather. 2018. EZCursorVR : 2 D Selection with Virtual Reality Head-Mounted Displays. In *Graphics Interface 2018*, 114–121.
 42. Raquel Robinson, John Murray, and Katherine Isbister. 2018. You’re Giving Me Mixed Signals!: A Comparative Analysis of Methods that Capture Players’ Emotional Response to Games. LBW567. <https://doi.org/10.1145/3170427.3188469>
 43. David Rutten. Grasshopper™. Retrieved December 22, 2018 from <https://www.grasshopper3d.com/>
 44. Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa (eds.). 2013. *Game Analytics*. Springer London, London. Retrieved October 27, 2016 from <http://link.springer.com/10.1007/978-1-4471-4769-5>
 45. Ben Shneiderman. 2002. Creativity Support Tools. *Commun. ACM* 45, 10: 116–120. <https://doi.org/10.1145/570907.570945>
 46. Ben Shneiderman. 2007. Creativity Support Tools: Accelerating Discovery and Innovation. *Commun. ACM* 50, 12: 20–32. <https://doi.org/10.1145/1323688.1323689>
 47. Herbert Alexander Simon. 1996. *The sciences of the artificial*. MIT press.
 48. Brittany N. Smith, Anbang Xu, and Brian P. Bailey. 2010. Improving interaction models for generating and managing alternative ideas during early design work. In *Graphics Interface 2010* (GI '10), 121–128. Retrieved October 20, 2011 from <http://dl.acm.org/citation.cfm?id=1839214.1839236>

49. Matthew Stephan and James R Cordy. 2013. A survey of model comparison approaches and applications. *Conference on Model-Driven Engineering and Software Development*: 265–277.
50. Michael Terry and Elizabeth D Mynatt. 2002. Recognizing creative needs in user interface design. In *Creativity and Cognition 2002 (C&C '02)*, 38–44. <https://doi.org/10.1145/581710.581718>
51. Michael Terry, Elizabeth D. Mynatt, Kumiyo Nakakoji, and Yasuhiro Yamamoto. 2004. Variation in element and action: supporting simultaneous development of alternative solutions. In *CHI 2004 (CHI '04)*, 711–718. <https://doi.org/10.1145/985692.985782>
52. David Watson, Lee Anna Clark, and Auke Tellegen. 1988. Development and validation of brief measures of positive and negative affect: the PANAS scales. *Journal of personality and social psychology* 54, 6: 1063.
53. Robert Woodbury, Arefin Mohiuddin, Mark Cichy, and Volker Mueller. 2017. Interactive design galleries: A general approach to interacting with design alternatives. *Design Studies* 52: 40–72. <https://doi.org/10.1016/j.destud.2017.05.001>
54. Loutfouz Zaman, Christian Neugebauer, Wolfgang Stuerzlinger, and Robert Woodbury. 2018. GEM-NI+: Leveraging Difference Visualization and Multiple Displays for Supporting Multiple Complex Generative Design Alternatives. In *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems (CHI EA '18)*, LBW106:1–LBW106:6. <https://doi.org/10.1145/3170427.3188593>
55. Loutfouz Zaman, Wolfgang Stuerzlinger, and Christian Neugebauer. 2017. MACE: A New Interface for Comparing and Editing of Multiple Alternative Documents for Generative Design. In *Proceedings of the 2017 ACM Symposium on Document Engineering (DocEng '17)*, 67–76. <https://doi.org/10.1145/3103010.3103013>
56. Loutfouz Zaman, Wolfgang Stuerzlinger, Christian Neugebauer, Rob Woodbury, Maher Elkhaldi, Naghmi Shireen, and Michael Terry. 2015. GEM-NI: A System for Creating and Managing Alternatives In Generative Design. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*, 1201–1210. <https://doi.org/10.1145/2702123.2702398>
57. FlowCanvas - Visual Scripting for Unity. Retrieved December 22, 2018 from <http://flowcanvas.paradoxnotion.com/>
58. PlayMaker - Visual Scripting for Unity3D. *Hutong Games*. Retrieved December 22, 2018 from <http://hutonggames.com/>
59. Bolt: Visual Scripting for Unity. *Bolt: Visual Scripting for Unity*. Retrieved December 22, 2018 from <https://ludiq.io/bolt>
60. Amplify Shader Editor. Retrieved December 22, 2018 from <http://amplify.pt/unity/amplify-shader-editor/>
61. Blueprints Visual Scripting. Retrieved December 22, 2018 from <https://docs.unrealengine.com/en-us/Engine/Blueprints>

62. Nativizing Blueprints. Retrieved December 22, 2018 from <https://docs.unrealengine.com/en-US/Engine/Blueprints/TechnicalGuide/NativizingBlueprints>
63. Robo Recall. *Robo Recall*. Retrieved December 22, 2018 from <https://www.epicgames.com/roborecall/en-US/home>
64. Git. Retrieved December 22, 2018 from <https://git-scm.com/>
65. Open Broadcaster Software | Home. Retrieved February 18, 2019 from <https://obsproject.com/>

Appendix A

CSI Survey

The Creativity Support Index

What is the participant's ID?

How many times will the participant fill out the CSI?

Please choose a folder for saving the participant's data.

Please rate your agreement with the following statements:

I was satisfied with what I got out of the system or tool.

Highly Disagree Highly Agree

It was easy for me to explore many different ideas, options, designs, or outcomes, using this system or tool.

Highly Disagree Highly Agree

☐ N/A

The system or tool allowed other people to work with me easily.

Highly Disagree Highly Agree

I would be happy to use this system or tool on a regular basis.

Highly Disagree Highly Agree

I was able to be very creative while doing the activity inside this system or tool.

Highly Disagree Highly Agree

My attention was fully tuned to the activity, and I forgot about the system or tool that I was using.

Highly Disagree Highly Agree

Continue

Please rate your agreement with the following statements:

I enjoyed using this system or tool.

Highly Disagree

Highly Agree

The system or tool was helpful in allowing me to track different ideas, outcomes, or possibilities.

Highly Disagree

Highly Agree

What I was able to produce was worth the effort I had to exert to produce it.

Highly Disagree

Highly Agree

The system or tool allowed me to be very expressive.

Highly Disagree

Highly Agree

☐ N/A

It was really easy to share ideas and designs with other people inside this system or tool.

Highly Disagree

Highly Agree

I became so absorbed in the activity that I forgot about the system or tool that I was using.

Highly Disagree

Highly Agree

Continue

**When doing this task, it's most important
that I'm able to...**

Explore many different ideas,
outcomes, or possibilities

☐

☐ Work with other people

1/15

Continue

**When doing this task, it's most important
that I'm able to...**

Be creative and expressive ☐

☐ Produce results that are worth
the effort I put in

2/15

Continue

**When doing this task, it's most important
that I'm able to...**

Enjoy using the system or tool ☐

☐ Become immersed in the activity

3/15

Continue

**When doing this task, it's most important
that I'm able to...**

Become immersed in the activity ☐

☐ Produce results that are worth
the effort I put in

4/15

Continue

**When doing this task, it's most important
that I'm able to...**

Work with other people ☐

☐ Enjoy using the system or tool

5/15

Continue

**When doing this task, it's most important
that I'm able to...**

Produce results that are worth
the effort I put in

☐

Explore many different ideas,
outcomes, or possibilities

☐

6/15

Continue

**When doing this task, it's most important
that I'm able to...**

Be creative and expressive ☐

☐ Become immersed in the activity

7/15

Continue

**When doing this task, it's most important
that I'm able to...**

Work with other people ☐

☐ Produce results that are worth
the effort I put in

8/15

[Continue](#)

**When doing this task, it's most important
that I'm able to...**

Be creative and expressive ☐

☐ Enjoy using the system or tool

9/15

Continue

**When doing this task, it's most important
that I'm able to...**

Explore many different ideas,
outcomes, or possibilities

☐

☐ Become immersed in the activity

10/15

[Continue](#)

**When doing this task, it's most important
that I'm able to...**

Work with other people ☐

☐ Be creative and expressive

11/15

Continue

**When doing this task, it's most important
that I'm able to...**

Produce results that are worth
the effort I put in

☐

☐ Enjoy using the system or tool

12/15

Continue

**When doing this task, it's most important
that I'm able to...**

Explore many different ideas,
outcomes, or possibilities

☐

☐ Be creative and expressive

13/15

Continue

**When doing this task, it's most important
that I'm able to...**

Work with other people ☐

☐ Become immersed in the activity

14/15

[Continue](#)

**When doing this task, it's most important
that I'm able to...**

Explore many different ideas,
outcomes, or possibilities

☐

☐ Enjoy using the system or tool

15/15

Continue

Appendix B

Positive Affect Negative Affect Schedule (PANAS)

After task questionnaire

How do you feel after completing the task?

	Not at All	A Little	Moderately	Quite a Bit	Extremely
Interested	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Distressed	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Excited	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Upset	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Strong	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Guilty	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Scared	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Hostile	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Enthusiastic	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Proud	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Irritable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Alert	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ashamed	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Inspired	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Nervous	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Determined	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Attentive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Jittery	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Active	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Afraid	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Additional Comments about the task/tool used.

Your answer

BACK

NEXT

Never submit passwords through Google Forms.

Appendix C

Pre-Study Verbal Script

During this study you will complete a task using a plugin for the Unreal Engine that supports the creation and management of Unreal Blueprint alternatives and meeting with us twice a week for 2 weeks for an interview and to fill out questionnaires, which we can discuss the time for. We will not be recording any video or audio of your participation. If at any time you feel uncomfortable and would like to withdraw from the study, please let us know and this user study will be terminated. You can also request for the data that we collected to be withdrawn from publishing up to seven days from the end of your participation. By signing the consent form, you are agreeing to allow us to use the results of your participation in our research analysis and to use your name and level of experience in research publications to prove that your opinion as a professional is valuable. Please let us know if you have any questions before we start. To start the study, we are going to go over how to install and use the plugin. After the tutorial I will be giving you access to a git repository with the plugin and documentation for it.

Thank you for your participation.

Appendix D

Pre-Study Survey

Pre-Study Questionnaire Part 1

How long has it been since you started using a desktop/laptop computer? *

Your answer _____

How often do you use data comparison/differencing tools? *

- ☐ Never
- ☐ 1-2 times a year
- ☐ 1-2 times per month
- ☐ 3-7 times per month
- ☐ 1-2 times per week
- ☐ 3-7 times per week

What data comparing/differencing tools have you used?

- ☐ Microsoft Word Track Changes and Compare Documents features
- ☐ Linux/Unix diff command
- ☐ Built-in tools in GitHub and BitBucket for source code comparing
- ☐ Other: _____

How often do you use diagrams or flowcharts in your line of work? *

- ☐ Never
- ☐ 1-2 times per year
- ☐ 3-5 times per year
- ☐ 1-2 times per month
- ☐ 3-5 times per month
- ☐ 1-2 times per week
- ☐ 3-5 times per week
- ☐ Other: _____

Do you have experience with visual node-based programming languages? (Also known as diagrammatic or data-flow programming languages, which are based on the idea of "boxes and arrows") *

- ☐ Yes
- ☐ No

If you answered "Yes" to the previous question, list all the data-flow programming languages and tools that you have experience with. (E.g., NodeBox, GEM-NI, Unreal Blueprints, Playmaker for Unity, Amplify Shader Editor for Unity, CryEngine Flow Graph, Blender Node Editor, Blender Sverchok, Grasshopper 3D, Max/MSP, Quartz Composer, vvvv)

Your answer

How often do you use diagrams or flowcharts in your line of work? *

- ☐ Never
- ☐ 1-2 times per month
- ☐ 3-7 times per month
- ☐ 1-2 times per week
- ☐ 3-7 times per week

Do you have experience with visual programming languages that are NOT node-based? (E.g., Block-based languages such as Scratch, Snap!, Alice) *

- ☐ Yes
- ☐ No

If you answered "Yes" to the previous question, list all the visual programming languages and tools that you have experience with.

Your answer

Do you have experience with generative design tools? (E.g., Processing, NodeBox, GEM-NI, GenerativeComponents, Grasshopper 3D, vvvv, Quartz Composer, Open Frameworks, Blender Sverchok) *

☐ Yes

☐ No

If you answered "Yes" to the previous question, list all the generative design tools you have experience with.

Your answer

BACK

NEXT

Never submit passwords through Google Forms.

Pre-Study Questionnaire (Part 2)

How many years of experience do you have with the Unity Engine? *

Your answer

In a typical month how many hours do you spend using the Unity Engine? *

- ☐ None
- ☐ 1-10 Hours
- ☐ 10-40 Hours
- ☐ 40-120 Hours
- ☐ 120+ Hours

How many years of experience do you have with the CryEngine? *

Your answer

In a typical month how many hours do you spend using the CryEngine? *

- ☐ None
- ☐ 1-10 Hours
- ☐ 10-40 Hours
- ☐ 40-120 Hours
- ☐ 120+ Hours

How many years of experience do you have with the Lumberyard Engine? *

Your answer

In a typical month how many hours do you spend using the Lumberyard Engine? *

- ☐ None
- ☐ 1-10 Hours
- ☐ 10-40 Hours
- ☐ 40-120 Hours
- ☐ 120+ Hours

How many years of experience do you have with the Blender Game Engine? *

Your answer

In a typical month how many hours do you spend using the Blender Game Engine? *

- ☐ None
- ☐ 1-10 Hours
- ☐ 10-40 Hours
- ☐ 40-120 Hours
- ☐ 120+ Hours

How many years of experience do you have with the Unreal Engine? *

Your answer

In a typical month how many hours do you spend using the Unreal Engine? *

- ☐ None
- ☐ 1-10 Hours
- ☐ 10-40 Hours
- ☐ 40-120 Hours
- ☐ 120+ Hours

What is your proficiency using Unreal Blueprints? *

- ☐ No Experience
- ☐ Beginner (E.g., I followed some tutorials online, never finished a game)
- ☐ Intermediate (E.g., I took INFR 4335 and created an Unreal Game for the final project)
- ☐ Expert (E.g., my proficiency is beyond what we were taught in INFR 4335. I am an indie developer who created a full game using Blueprints)

What is your proficiency using C++ in the Unreal Engine? *

- ☐ No Experience
- ☐ Beginner (E.g., I followed some tutorials online, never finished a game)
- ☐ Intermediate (E.g., I created a simple game using C++ in Unreal)
- ☐ Expert (E.g., I am an indie developer who created a full game using C++ in Unreal)

Appendix E

Post-Study Survey

Post Questionnaire

How would you rank the efficiency of using the Unreal Engine (by itself)? *

	1	2	3	4	5	6	7	
Poor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Excellent

How would you rank the ease of use of the Unreal Engine (by itself)? *

	1	2	3	4	5	6	7	
Poor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Excellent

How would you overall rank using the Unreal Engine (by itself)? *

	1	2	3	4	5	6	7	
Poor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Excellent

Rate how likely it is that you will use the Unreal Engine (by itself) for future projects? *

	1	2	3	4	5	6	7	
Very Unlikely	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Very Likely

How would you rank the efficiency of using the Unreal Engine with the additional tool? *

	1	2	3	4	5	6	7	
Poor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Excellent

How would you rank the ease of use of the Unreal Engine with the additional tool? *

	1	2	3	4	5	6	7	
Poor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Excellent

How would you overall rank using the Unreal Engine with the additional tool? *

	1	2	3	4	5	6	7	
Poor	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Excellent

Rate how likely it would be that you would use the Unreal Engine with the additional tool for future projects? *

	1	2	3	4	5	6	7	
Unlikely	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Likely

What was your preferred tool to use during the tasks? *

- ☐ The Unreal Engine by itself
- ☐ The Unreal Engine with the additional tool

Additional Comments

Your answer

BACK

SUBMIT

Never submit passwords through Google Forms.