Code Park: A New 3D Code Visualization Tool and IDE

2017

Pooya Khaloo
University of Central Florida

CODE PARK: A NEW 3D CODE VISUALIZATION TOOL AND IDE

by

POOYA KHALOO
B.S. University of Tabriz, 2014

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2017

ABSTRACT

We introduce Code Park, a novel tool for visualizing codebases in a 3D game-like environment. Code Park aims to improve a programmer's understanding of an existing codebase in a manner that is both engaging and fun to be appealing especially for novice users such as students. It achieves these goals by laying out the codebase in a 3D park-like environment. Each class in the codebase is represented as a 3D room-like structure. Constituent parts of the class (variable, member functions, *etc.*) are laid out on the walls, resembling a syntax-aware "wallpaper". The users can interact with the codebase using an overview, and a first-person viewer mode. They also can edit, compile and run code in this environment. We conducted three user studies to evaluate Code Park's usability and suitability for organizing an existing project. Our results indicate that Code Park is easy to get familiar with and significantly helps in code understanding. Further, the users unanimously believed that Code Park was an engaging tool to work with.

Dedicated to my beloved wife and my parents, for their never ending support the entire way.

ACKNOWLEDGMENTS

Many thanks to my faculty advisor Dr. Joseph LaViola. for his guidance and directions along my graduate studies. Additionally, I greatly appreciate the members of my committee, Dr. Hassan Foroosh, and Dr. Gary Leavens. Furthermore I would like to thank David Bettner for his financial support and his great feedbacks along the way.

Finally special thanks to the members of UCFs Interactive Systems and User Experience Lab, especially Mehran Maghoumi and Eugene Taranta.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Integrated development environments (IDE) for writing code with programming languages have been around for decades. Newcomers to the field of software development quickly discover that a large number of tools that they will spend the majority of time working with are either text-based or graph-based. The common observation in almost all programming tools today is that they are inherently 2D. This may not pose a significant issue for an expert developer when writing code. However, learning new language or even understanding an existing code base is often a huge undertaking for a novice programmer [6]. This often leads to combing through hundreds of files that are scattered in usually of directories to understand how somebody else's code works. Code understanding becomes even more important when a programmer wants to debug an existing piece of software. The user's understanding usually comes from studying and reading the code. However, this form of understanding can quickly become tedious and boring, especially discouraging beginners from trying to learn by reading code. Our hypothesis is that this issue can be mitigated with code visualizers that display constituent parts of a code base in a more clear and coherent manner.

Code visualization techniques assist developers in gaining insights into a codebase that may otherwise be difficult or impossible to acquire via the use of a traditional text editor. Over the years, various techniques have been proposed to address different issues. For instance, SeeSoft [7] maps each line of code to an auxiliary UI bar such that the color of the bar represents a property of the code, *e.g.* red rows could represent recently changed lines. Code Bubbles [1] organizes source into interrelated editable fragments illustrated as bubbles within the user interface, and CodeCity [3] uses a 3D city-like structure to visualize the relative complexity of modules within a codebase. Each of these tools attempt to show code in a new form that make learning and understanding code easier and more efficient. However, among all of the techniques that have been explored, little attention has been given to how source code can be visualized in order to help developers become familiar with and learn a *new* codebase.

The task of learning a new codebase is fraught with many issues, three of which include memorization, cognitive load, and engagement [8, 9, 10]. Imagine a new hire set to study a proprietary project that is a complex hierarchical conglomeration of thousands of source files. Indeed memorizing the structure of such a system is a difficult chore, not to mention mentally exhausting as well as potentially tedious and likely boring . To address these issues, we introduce Code Park, a new 3D code visualization tool that aims to improve a programmer's understanding of an existing codebase in a manner that is both engaging and fun. Code Park organizes source code into a 3D scene in order to take advantage of human's spatial memory capabilities [11] and help one better understand and remember the architecture. By extending into 3D space, Code Park is also able to provide an exciting game-like environment, thereby encouraging engagement and subverting boredom. Code Park also supports two unique points of view: exo-centric and ego-centric, which allows one to examine the codebase at different granularities.

Our design goals with Code Park are threefold. We aimed to create a code visualization tool that helps user in becoming familiar with and memorizing an existing codebase. This tool must be easy to work with and must show information in a form which reduces the user's cognitive load. Finally, the tool must be engaging and enjoyable to work with. We designed Code Park in three iterative step. In each step we evaluated our design by conducting a user study. Based on the result of the previous user study we evolved our design and created a new environment or added new features.

Specifically the following are the contributions of our exploratory work:

1. Creating a fun, engaging and unconventional tool for improved code understanding.

2. Gauging user interest and examining the usability of such tool.

3. Displaying the source code itself in a 3D environment (rather than a metaphorical representation) and facilitate “intimate” interaction with the code via an ego-centric mode.

4. Editing, compiling and debugging code in 3D environment.

Reader’s Guide

In this section a brief explanation of each chapter is given.

Chapter 2 - A review of relevant literature in the field of code visualization is presented.

Chapter 3 - Covers first iteration of Code Park that called Code House and how it was created.

Chapter 4 - Presents an evaluation and discussion on Code House.

Chapter 5 - Talks about the design and implementation process of the second iteration of Code Park: Code Visualizer.

Chapter 6 - Discuss the user study and its results on Code Park.

Chapter 7 - Presents the third and the last version of Code Park: A 3D IDE.

Chapter 8 - Covers an evaluation and its result’s discussion of Code Park: A 3D IDE.

Chapter 9 - Limitation and plans for future work are laid out

Chapter 10 - Concluding remarks for the presented work are presented.

Appendix A - IRB Approval Letters

Appendix B - All the questionnaires and assignment that was used in evaluations.

CHAPTER 2: RELATED WORK

There is a body of work available in the literature focusing on software visualization. These efforts can mostly be put into two categories. The ones which perform 2D visualization and the ones that perform 3D visualization. SeeSoft [7], one of the earliest visualization metaphors, allows one to analyze up to 50,000 lines of code by mapping each line of code into a thin row. Marcus *et al.* [12] added a new dimension to SeeSoft to support an abstraction mechanism to achieve better representation of higher dimensional data. Goldberg and Robson introduced SmallTalk, one of the first visualized development environment [13].

Recently, there have been more work focusing on 2D visualization. Code Bubbles [1] suggested a collection of editable fragments that represent functions in a class (see Figure 2.1). Code Gestalt [14] used tag overlay and thematic relations. Lanza and Ducasse [15] proposed categorizing classes and their internal objects into blocks called Blue Prints. Gutwenger *et al.* [16] proposed an approach for improved aesthetic properties of UML diagrams when visualizing hierarchical and non-hierarchical relations. Radfelder and Gogolla [17] extended UMLs by adding third and fourth dimension of data in a way that they can show both dynamic and static aspect of diagram in a single view. Balzer *et al.* [18] introduced hierarchy-based visualization for software metrics using Voroni Treemaps. Additionally, Holten [19] used both hierarchical and non-hierarchical data to visualize adjacency relations in a software. Hawes *et al.* [20] presented CodeSurveyor a spatial visualization technique that aims to support code comprehension in large codebases by allowing developers to view large-scale software at all levels of abstraction. The common observation among these efforts is that they are all based on 2D environments and were mostly suitable for expert users.

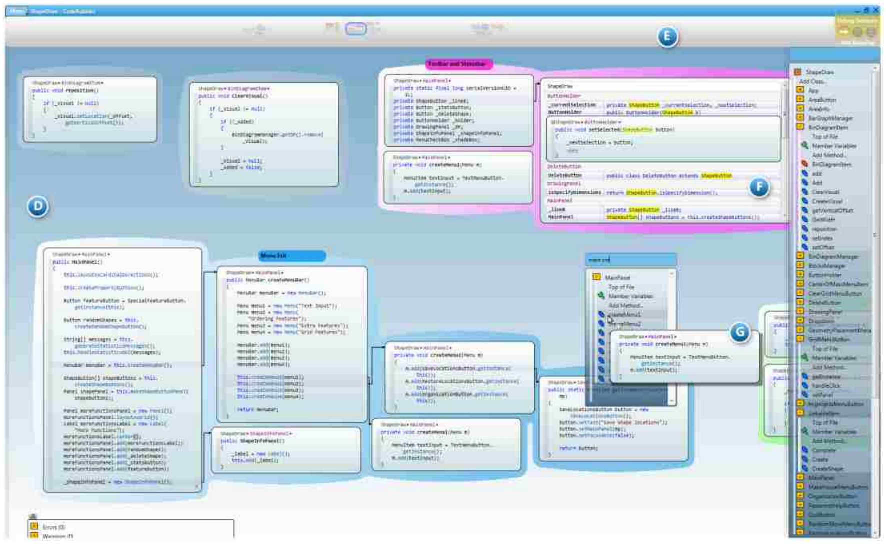
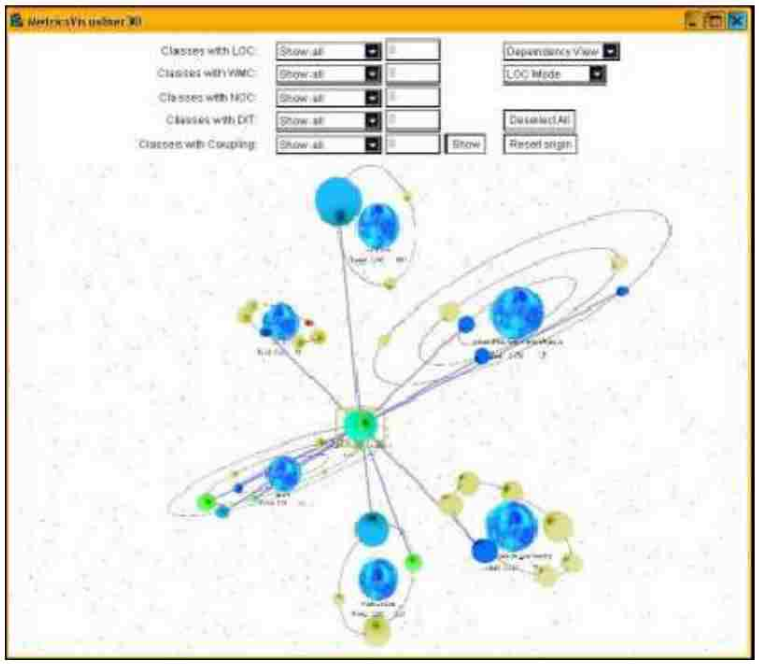


Figure 2.1: Code Bubble: suggested a collection of editable fragments that represent functions in a class [1].

Prefer 3D over 2D?

Remembering code structure will result in faster development so it is an essential part of being a programmer. Specifically, 3D environments tap into the spatial memory of the user and help with memorizing the position of objects [11]. These objects could be classes or methods. There are also studies which provide evidence that spatial aptitude is a strong predictor of performance with computer-based user interfaces. For instance Cockburn and McKenzie [21] have shown that 3D interfaces that leverage the human's spatial memory result in better performance even though some of their subjects believed that 3D interfaces are less efficient. Robertson *et al.* [22] have also shown that spatial memory does in fact play a role in 3D virtual environments. As a result, there are a variety of work available in the literature that are focused on 3D software visualization.


MetricsVisualiser 3D

Classes with LOC: Show all 0

Classes with WMC: Show all 0

Classes with NOC: Show all 0

Classes with DIT: Show all 0

Classes with Coupling: Show all 0

Dependency View

LOC Mode

Figure 2.2: A Solar System Metaphor for 3D Visualisation of Object Oriented Software Metrics [2].

A number of researchres have attempted to solve the problem of understanding code structure. Graham *et al.* [2] suggested a solar system metaphor, in which each planet represented a Java class and the orbits showed various inheritance levels (See Figure 2.2). Balzer *et al.* [23] presented the static structure and the relation of object-oriented programs using 3D blocks in a 2D landscape model. Bonyuet *et al.* [24] suggested a 3D software platform for visualization and manipulation of complex and large softwares called Code Mapping. Code Mapping was color coded for function, call position and experimental interaction data. Greevy *et al.* [25] worked on visualizing executive trace of software in 3D. They visualized dynamic behaviors of execution traces in terms of object creations and interactions.

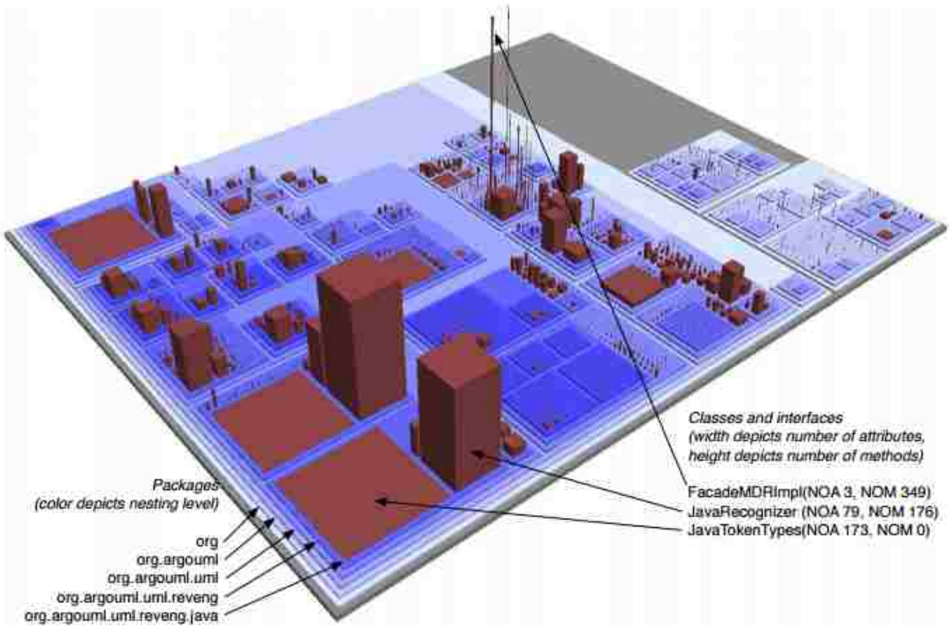
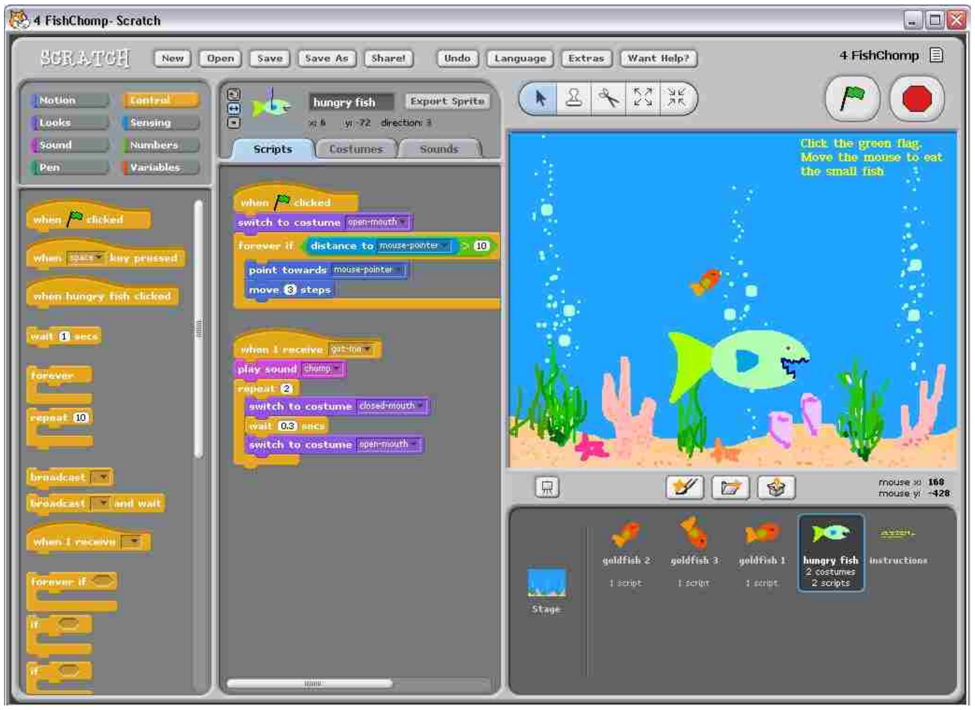

Classes and interfaces
(width depicts number of attributes,
height depicts number of methods)
FacadeMDRImpl(NOA 3, NOM 349)
JavaRecognizer (NOA 79, NOM 176)
JavaTokenTypes(NOA 173, NOM 0)
Packages
(color depicts nesting level)
org
org.argouml
org.argouml.uml
org.argouml.uml.reveng
org.argouml.uml.reveng.java

Figure 2.3: Code City: focus on the visualization of large and complex codebases using city structures [3].

Among these, the city metaphor is one of the most populer ideas. CodeCity [3] is an example of a city metaphor. CodeCity focus on the visualization of large and complex codebases using city structures where the number of methods in each class represented the width of the buildings and the number of attributes represented their height (see Figure 2.3). Panas *et al.* [26] used Vizz3D[1] to visualize communication between stakeholders and developers in the process of software implementation. Alam and Dugerdil [27] introduced the Evospaces visualization tool in which they represent files and classes with buildings similar to CodeCity. Additionally, they showed the relations between the classes using solid pipes. There are other works that leverage such visualization metaphors such as [28] and [29]. The common property of these tools is that most of them only showed the name of the classes in their environment which is not instrumental for learning purposes and again, they are

[1]`http://vizz3d.sourceforge.net/`

mainly targeting experienced developers.



```
when clicked
switch to costume open-mouth
forever if distance to mouse-pointer > 10
    point towards mouse-pointer
    move 8 steps

when I receive got-fish
play sound chomp
repeat 2
    switch to costume closed-mouth
    wait 0.3 secs
    switch to costume open-mouth
```

Figure 2.4: Scratch: designed to be fun and likable [4].

What about being engaging?

The fun and engaging aspect of programming tools, especially for teaching purposes, has gained attention in recent years. ToonTalk [30] is a general purpose programming system that animated source code in a video game like environment. It was created as a self teaching programming system which was suitable for children to use. Alice by Cooper *et al.* [31] is another tool that emphasize this aspect. Alice is a 3D interactive animation environment tool that aims to encourage student engagement by creating a game-like environment. Resnik *et al.* [4] introduced a novel development environment called Scratch that appeals to people who believe their skills are not on par with experienced developers. They designed their tool in a way that was fun and likable. They also made it suitable for the youth learners (see Figure 2.4). Parsons and Haden [32] introduced an interactive tool that helps novice programmers to learn programming principles in an entertaining puzzle like format where each puzzle's solution is a complete sample of well written code. AlgoPath by Perrin *et al.* [5] is another project focused on teaching algorithm through a 3D game like environment. In AlgoPath each variable is represented by a figure carrying a backpack and the sequence of instructions is represented by stone path (see Figure 2.5). Zorn *et al.* [33] examined the game Minecraft as a means of increasing interest in programming. They introduced CodeBlocks a block based programming language in Minecraft environment.

There are many other works that are focused on software visualization. We direct the reader to comprehensive surveys of different methods available in the work of Teyseyre and Campo [34] and also Caserta and Olivier [35].

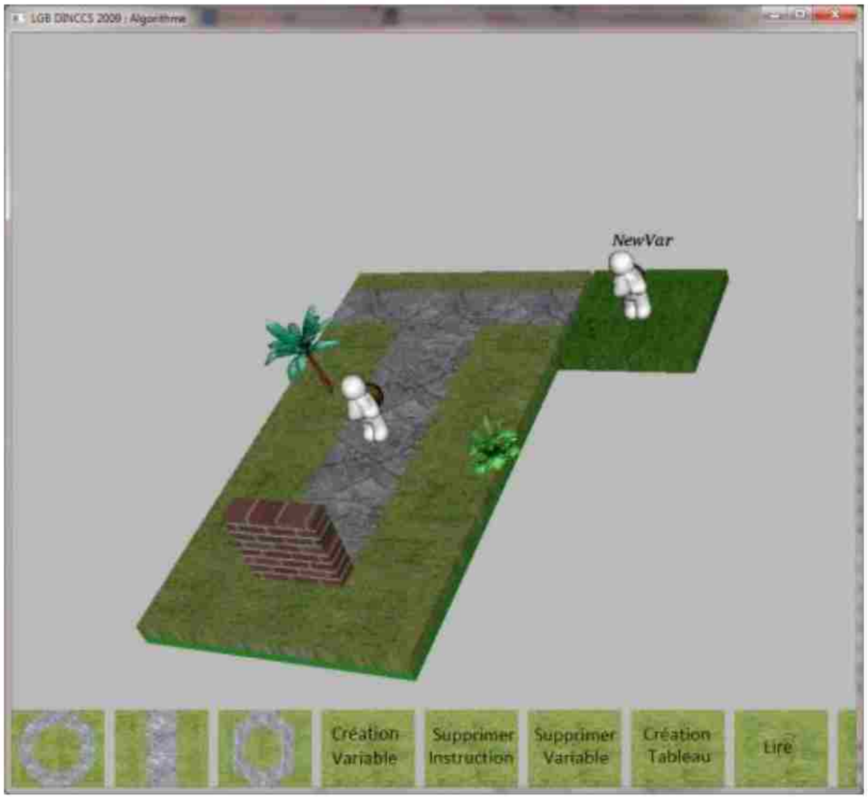


Figure 2.5: AlgoPath: A New Way of Learning Algorithmic [5].

One thing that sets Code Park apart from current tools in the literature is that it is designed to be suitable for both beginner and experienced developers alike. Saito *et al.* [36] examined the learning effects between a visual and a text-based environment on teaching programming to beginners. Their results deemed the visual environment as a more suitable option for teaching beginners. Also, Code Park offers code interaction from an exo-centric and an ego-centric perspective, combining the benefits of both interaction modalities. The addition of the ego-centric view is the key distinction between the current work and CodeCity [3], as this additional view makes examining the code at different granularities possible.

CHAPTER 3: CODE HOUSE

Our very first design was a 3D house environment (see Figure 3.1). The code of each class laid out (like wallpaper) on the walls of different rooms such as shown in Figure 3.3a. The house was a fixed model and was not procedurally generated. We called this version Code House. The users would explicitly define where each code wallpaper was placed. The wallpapers were syntax aware. Users could explore the environment in a first-person view mode. The users had a cross-hair in the middle of the screen and could walk up to a wall and inspect the code that was attached to the wall. We filled out each room with different sets of furniture and painted walls with different color to differentiate rooms from each other (such as bedroom, living room, kitchen, etc) and helped users remembering each room and the files they were attached in that room (see Figure 3.2).

Figure 3.1: The house's exterior.

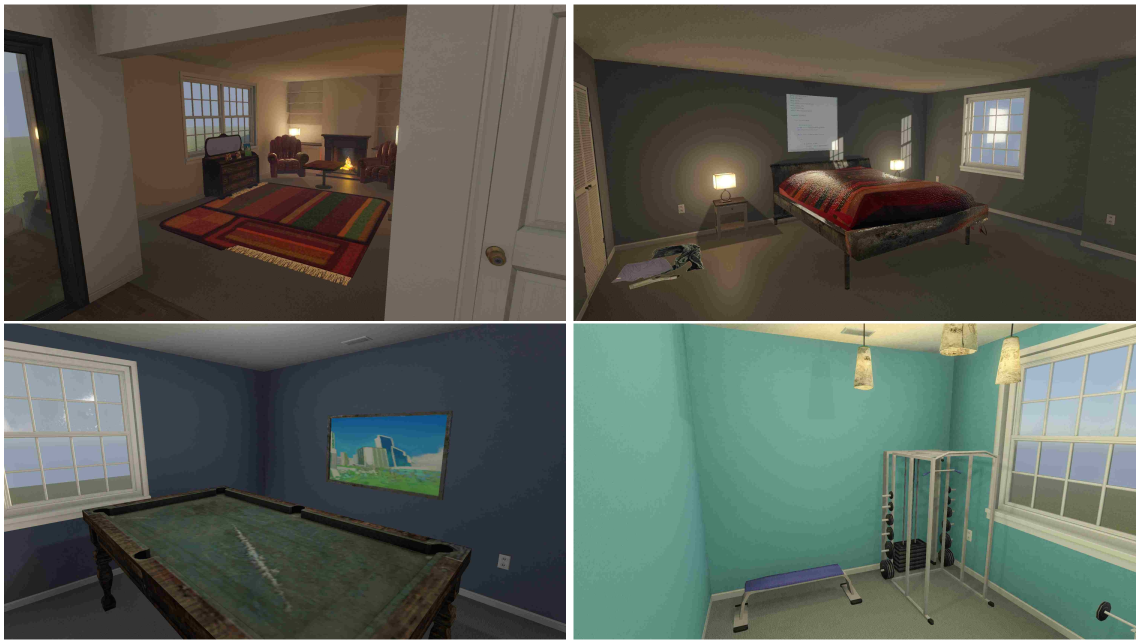


Figure 3.2: Different rooms with different sets of ferniture and different wall color to help user make this rooms apart.

The reason for starting this way was to get some insight about the users' ability in using their spatial memory and determining if they could remember the location of each part of the code base in the house. Essentially, we were interested in answering one question: can the users remember where they placed class *X* in this code-filled house, similar to the way they remember where they left their (say) toothbrush in real life?

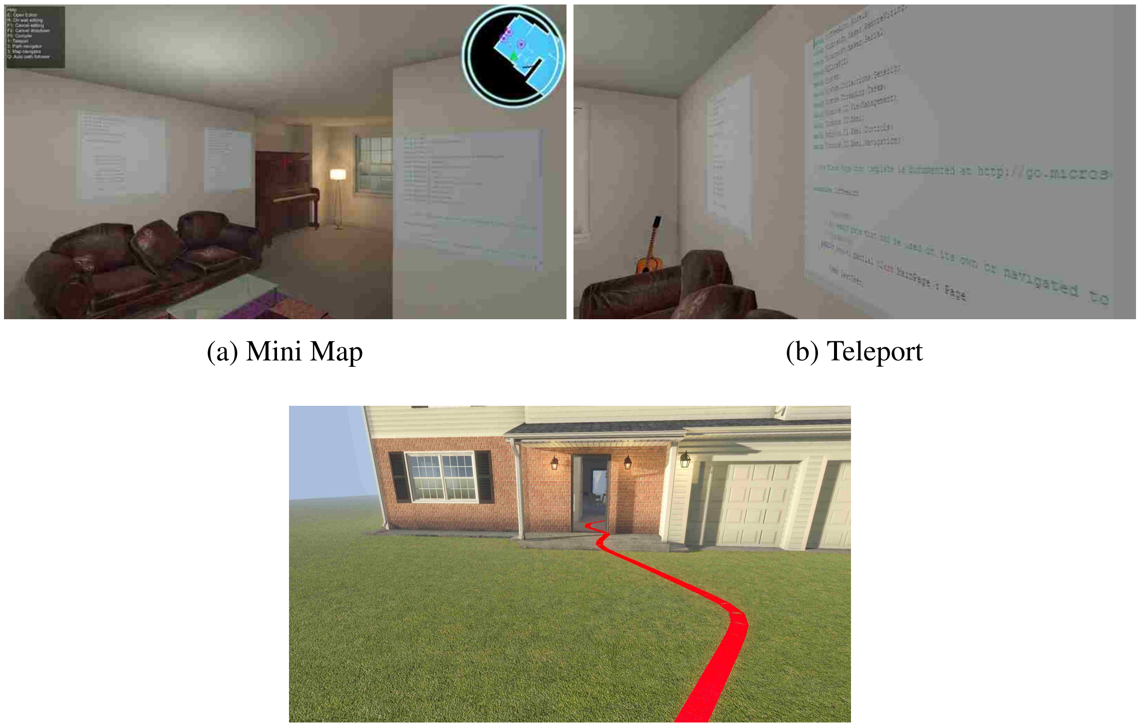
To answer this question we designed four different navigation methods. Users can select each of these four methods by pressing a button assigned to that method. After that they should chose their destination from a list of classes that already attached to walls. These four methods are:

Mini Map: Using a mini map that shows the location of each code canvas in the house. With showing a blueprint of the environment to users it helps users to reach their location and have an awareness of the environment around them (see Figure 3.3a).

Follow Path: Create a path that leads users to the location of selected code canvases. We used A* algorithm to create shortest path from user to the destination. The users need to follow this path through the house and like the mini map, it helps users to reach their destination with having awareness from the environment they are in (see Figure 3.3c).

Follow Path Automated: It is like the previous method but automated. After this method and the destination is selected it is automatically moves user on the path toward the selected code canvas. With this method, users can look around while automatically transporting to their destination and get more detailed information from the objects surrounded them.

Teleport: Instantly transfer users to the location of each code canvases. It changes the position of the user in the house and set it in front of the selected destination. It also changes the view point of the user to always look at the selected code after being teleported. The advantage of this method is that it takes no time to reach the destination but users are going to lost their awareness of the environment and how they reached their destination (see Figure 3.3b).


(a) Mini Map

(b) Teleport

(c) Follow Path

Figure 3.3: Different type of transport in Code House.

CHAPTER 4: CODE HOUSE EVALUATION

As mentioned we created the Code House prototype to answer these questions: can users remember where they put their specific code in the house and which navigation methods is better to help them remembering? We ran a small user study to evaluate Code House and answer to those questions.

Participants and Experiment Design

We recruited 5 participants (all male ranging in age from 22 to 28) from the Interactive Systems and User Experience (ISUE) lab at University of Central Florida. We asked each participant to bring one of their own project that was written with C#. We did so because this version of Code House only supports C# language. Each participant was asked to attach 8 classes from their project to the walls in different rooms. After they took a 30 minutes break they were asked to reach two of the classes with one of the transportation methods and then took another break for 2 hours. At the end, they were asked if they can remember the location of each class and reach them without any navigation method. They also were given a post questionnaire to ranked each method using 1 to 10 scale.

Result and Discussion

The goal of our study was to find out if the 3D environment helps users to remember the position of objects (in our case code canvases) they placed in the house. Out of 5 participants, three of them remembered the location of all the classes. The other two forgot the location of classes that they reached with Teleport method. This result shows that the 3D environment itself plays the main role in remembering objects using spatial memory but the efficient navigation method also has an affect on memory. It shows that how you reach your destination and how much data you acquire from your surrounding are going to help you to remember the path to the destination.

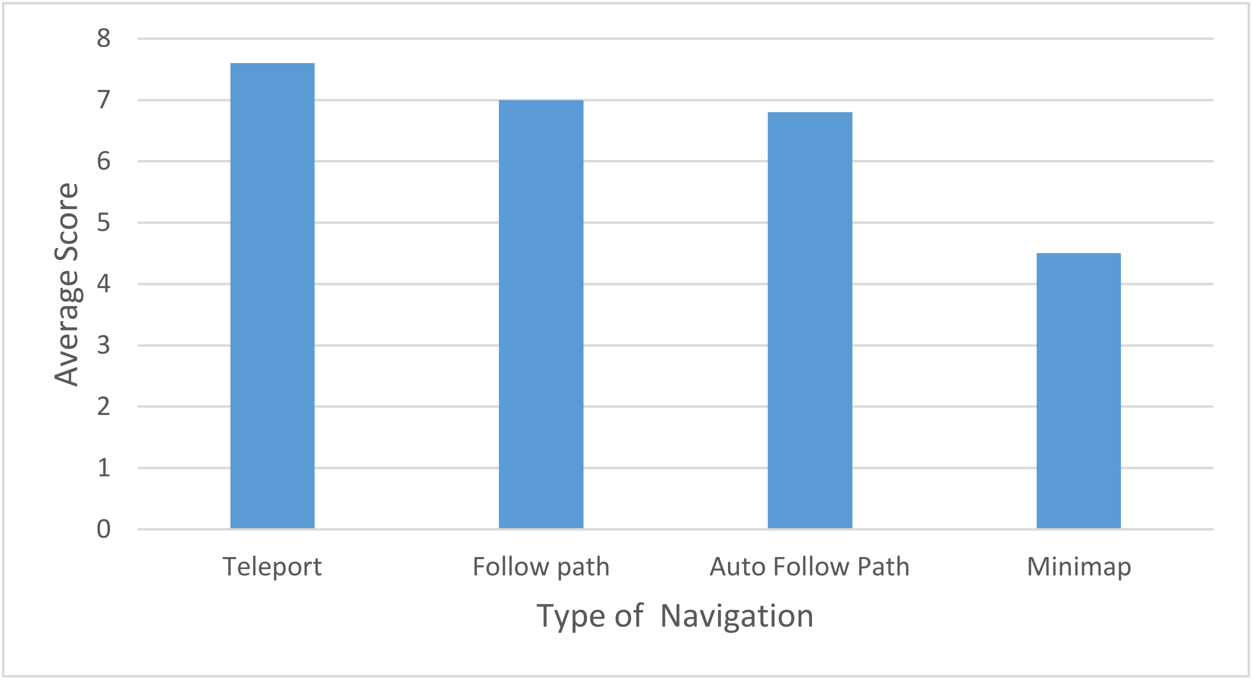


Figure 4.1: Mean Score of different navigation type.

Our second observation shows that Teleport was the most popular method of navigation among the others (see Figure 4.1). The possible explanation for this observation could be that Teleport transfers users to their destination faster compare to the other methods and because of that they like it more. The result also shows that they didn't like Mini Map compared to the other navigation methods. The possible explanation for this observation could be that with Mini Map they needed to constantly look at the top right corner of the screen and figured out where they were in the house and where the desired class was which took longer to reach to their desired destination and made them frustrated.

Our initial design quickly provided a positive answer to our main question: can the users remember where they placed class *X* in 3D environment? We also realized that looking at and interacting with the code with such a modality was both appealing and fun according to opinions of two participants of ours. However, this design was neither easy to use, nor was it helping with code understanding. Manually placing code in different parts of the house quickly became tedious and there was a lack of a logical mapping between the code files and the location they were placed inside the house: what would placing a *Queue* class inside the kitchen really imply and how are these two concepts even related? Further, a fixed house model would not be able to indefinitely accommodate today's ever expanding code bases. These problems and the concern to find some ways to solve them led us to change our design which led to Code Park.

CHAPTER 5: CODE PARK - 3D CODE VISUALIZATION

In our second iteration, we shifted our focus from an entire house to individual rooms. In this iteration, we implemented procedurally generated code rooms. This way, we could eliminate the need for manual code placement and could support large code bases. The rooms were placed on a ground filled with a grassy texture, which resembled a park. The rooms were grouped together based on the code's project file structure: files inside the same directory resulted in adjacent rooms and each group of rooms were labeled by the directory name (see Figure 5.1). This was done in order to avoid confusion for large code bases that had many files.

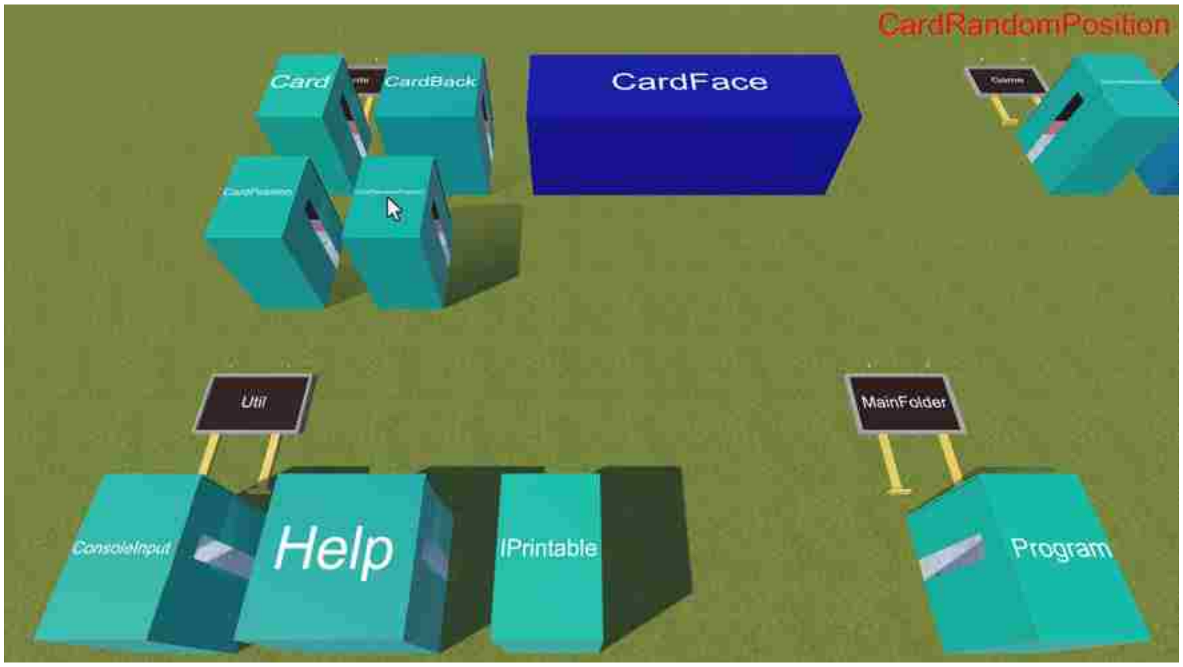
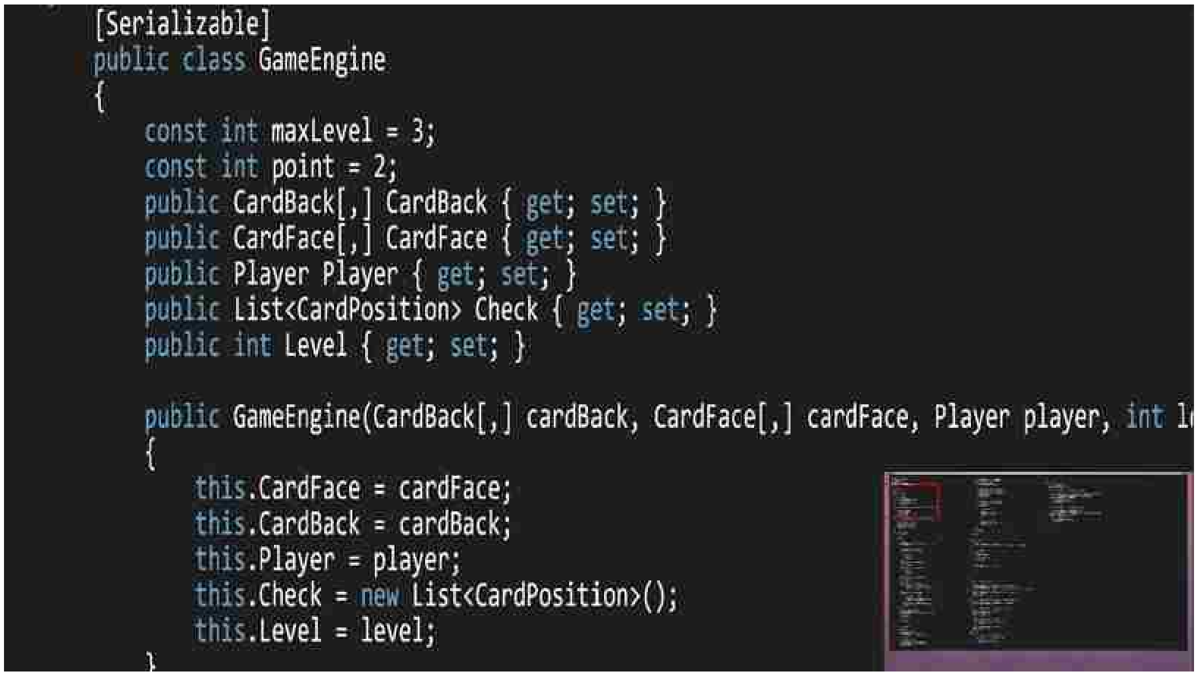


Figure 5.1: God's view with tooltip on top-right.

The size and the color of each room was proportional to the size of the class they contained (larger classes appeared as larger rooms that had a darker color on their exteriors). Each room had syntax-aware wallpapers that had a color theme matching Microsoft Visual Studio's dark theme (dark background and gray text with colored language keywords, comments and strings literals that was created with RTF (Rich Text Format) – see Figure 5.2). The users could explore the environment in first-person mode. The users could click on the wallpaper with the cross-hair to transition into the code viewing mode. This would place the camera orthogonal to the wallpaper, allowing the users to read the code and scroll it (using the mouse wheel) similar to a text editor.



```
[Serializable]
public class GameEngine
{
    const int maxLevel = 3;
    const int point = 2;
    public CardBack[,] CardBack { get; set; }
    public CardFace[,] CardFace { get; set; }
    public Player Player { get; set; }
    public List<CardPosition> Check { get; set; }
    public int Level { get; set; }

    public GameEngine(CardBack[,] cardBack, CardFace[,] cardFace, Player player, int l
    {
        this.CardFace = cardFace;
        this.CardBack = cardBack;
        this.Player = player;
        this.Check = new List<CardPosition>();
        this.Level = level;
    }
```

Figure 5.2: Code viewing mode.

One problem we noticed with this means of interaction was that the users were only able to inspect the code base one part at a time, limiting their ability to quickly jump between each part of the code base. As such, this iteration was later refined and was augmented by a *god's view* mode (see Figure 5.1). In god's view mode, the user have a top-down view of all the rooms placed on the ground. This way, the users can first get an overview of all the different pieces of the code base and then study it in more detail in the first-person mode. In god's view, the name of each class is etched on the roof of each room. This immediately helps the users know which class they are looking at. In this view, hovering the cursor over each room pops a tool-tip that shows the name of the class in a larger font at the top right corner of the screen (see Figure 5.1). The users can transition to the first-person view by clicking one of the code rooms and can go back to god's view by pressing a keyboard shortcut. The camera's transition to and from god's view is animated. This animation is not only visually appealing, but is also crucial to maintain the users' sense of spatial awareness of the 3D environment [37].

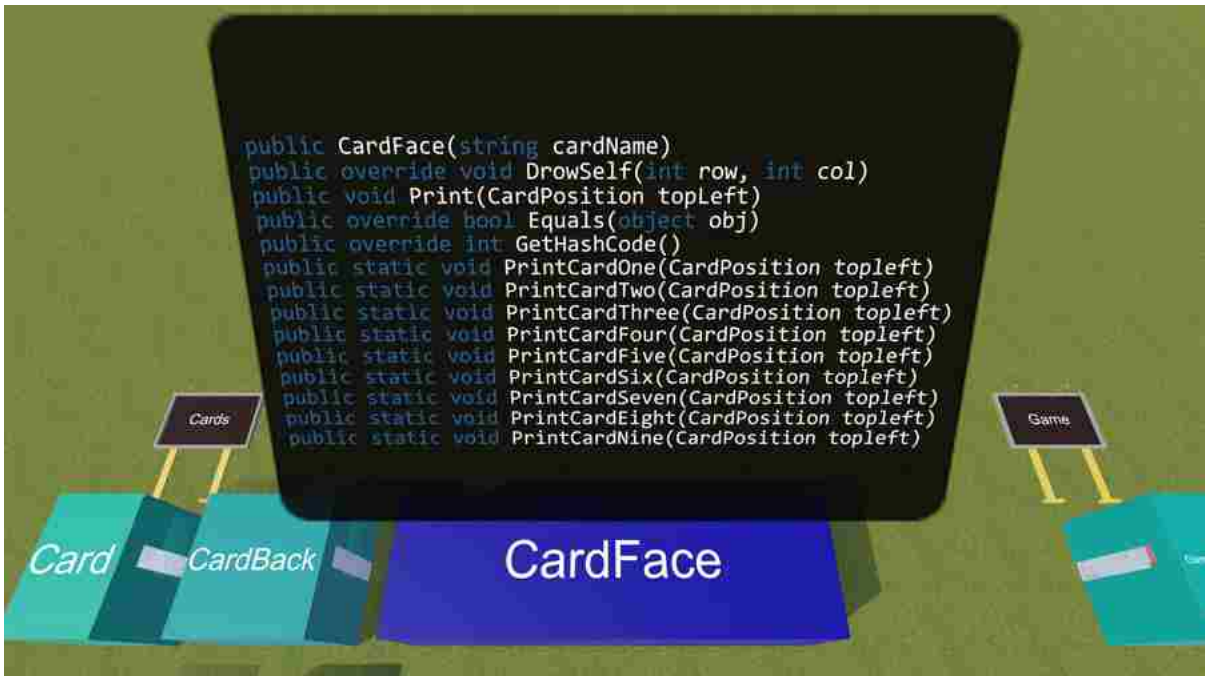

```
public CardFace(string cardName)
public override void DrowSelf(int row, int col)
public void Print(CardPosition topLeft)
public override bool Equals(object obj)
public override int GetHashCode()
public static void PrintCardOne(CardPosition topleft)
public static void PrintCardTwo(CardPosition topleft)
public static void PrintCardThree(CardPosition topleft)
public static void PrintCardFour(CardPosition topleft)
public static void PrintCardFive(CardPosition topleft)
public static void PrintCardSix(CardPosition topleft)
public static void PrintCardSeven(CardPosition topleft)
public static void PrintCardEight(CardPosition topleft)
public static void PrintCardNine(CardPosition topleft)
```

Figure 5.3: Class method overview tooltip.

We improved this mode further by adding syntax parsing features so as to be able to provide class-level details (such as the list of all member functions) to the user. We used .NET Compiler

Platform Roslyn[1] for the syntax parsing. We needed to create a third party application in order to use Roslyn because it only supports .Net 4 and above while the Unity that we built Code Park with only supports .Net 3.5 and below. We used socket programming to send data back and forth between these two applications.

In god's view, right-clicking on a room shows a tool-tip balloon that provides a list of the methods defined in the class corresponding to the room (see Figure 5.3). This allows the users to quickly glean useful information about each class. This overview is also available on one of the wallpapers inside each room as shown in Figure 5.4.

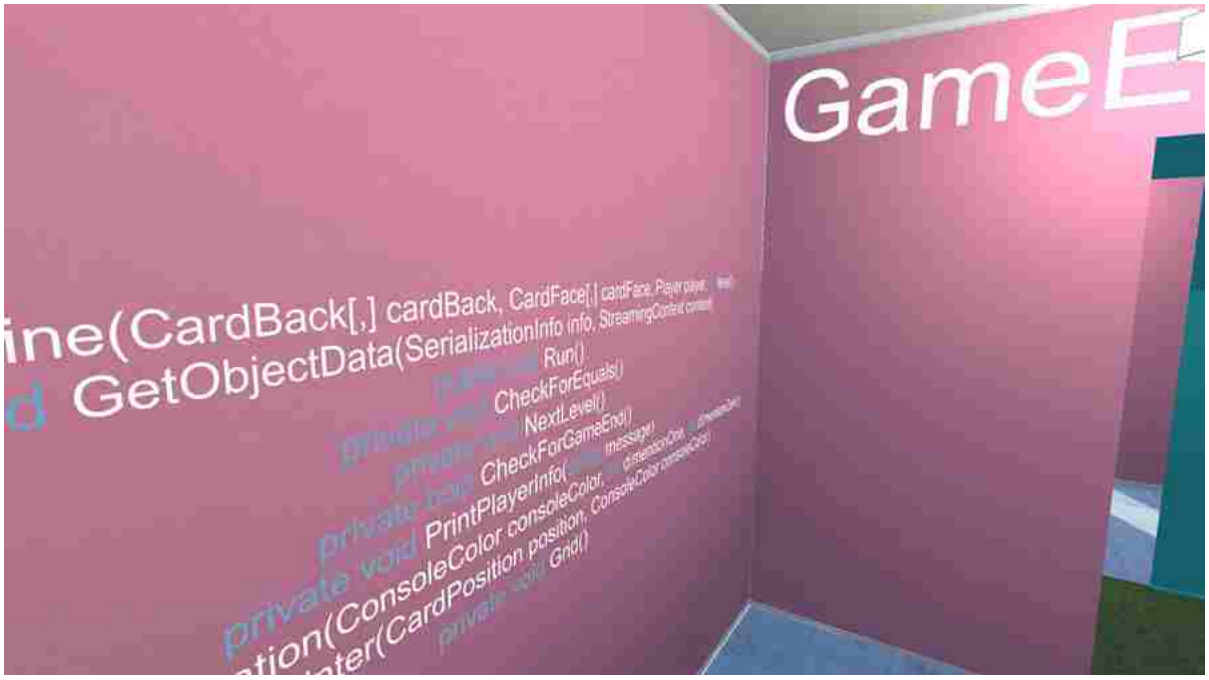

GameE
ine(CardBack[,] cardBack, CardFace[,] cardFace, Player player,
d GetObjectData(SerializationInfo info, StreamingContext context
Run()
CheckForEquals()
NextLevel()
CheckForGameEnd()
PrintPlayerInfo(
private void
ation(ConsoleColor consoleColor,
nter(CardPosition position, ConsoleColor consoleColor
private void Grid()

Figure 5.4: Class method overview wallpaper.

By supporting syntax parsing, we implemented a commonly used feature in most IDEs, namely

[1]`https://github.com/dotnet/roslyn`

go-to definition. This feature allows the programmers to quickly jump to the location inside a file where a user-defined type or a variable is defined for the first time. To do so, we parsed all the project when Code Park starts using Roslyn and created semantic and syntax trees. These trees give us all the necessary information about the location of each variable and where they are defined.

In Code Park, the users can click on a variable, function or a user-defined type with their cross-hair and jump to the location where the clicked item is defined for the first time. The jump from a room to another room is done using two consecutive animated camera transitions: from the current room to god's view and from god's view to the room containing the definition[2]. After the transition is finished, the definition of interest is signified by a blinking highlight to focus the user's attention and also to indicate the completion of the task. In addition to being visually appealing, these transitions maintain the users awareness of the 3D environment.

[2]Unless the definition is inside the same room in which case the camera is transitioned to the definition directly.

CHAPTER 6: CODE PARK EVALUATION

As mentioned in the Introduction, we designed Code Park to achieve three goals: ease of use, help with code understanding, and making code understanding fun and engaging. To evaluate these goals and gauge the usability of our system, we designed and performed a user study. In this version, Code Park only supports C# projects. Therefore, we decided to compare Code Park with one of the most prominently used IDEs for C#, namely Microsoft Visual Studio. Given a few programming-related tasks, we are interested in studying the effects that using Code Park has and also determine how Code Park helps with code understanding.

Participants and Equipment

We recruited 28 participants (22 males and 6 female ranging in age from 18 to 31 with a mean age of 22.8). Our requirements for participants were that they should be familiar with the C# language and also have prior experience with Visual Studio. Each participant was compensated with $10 for their efforts. Each participant was given a pre-questionnaire containing some demographic questions as well as some questions asking about their experience in developing C# applications. After that, a short C# skill test was administered to validate their responses in the pre-questionnaire. The skill test contained five multiple-choice questions with varying difficulties selected from a pool of coding interview questions[1]. All the questionnaires and assignment can be found in the Appendix.

After the skill test, each participant was given a set of tasks to perform using each tool (Visual Studio or Code Park) and filled out a post-questionnaire detailing their experience with each tool. Prior to performing the tasks on a specific tool, the participants were given a quick tutorial of the

[1]`https://www.interviewmocha.com/tests/c-sharp-coding-test-basic`

tool in question and were given a few minutes to work and get comfortable with it.

At the end of the study, the participants were asked to fill out a post-study questionnaire to share their overall experience. The duration of the user study ranged from 60 minutes to 90 minutes depending on how fast each participants progressed towards their assigned tasks.

Our setup consisted of a 50-inch Sony Bravia TV used as the computer screen (see Figure 6.1). The users used Visual Studio 2013 Community Edition and Unity3D v5.4.0 on a machine running Microsoft® Windows 10 64-bit equipped with 16.0 GB of RAM, Intel® Core^{TM} i7-4790 processor with 4 cores running at 3.60 GHz and NVIDIA GeForce GTX 970 graphics processor.

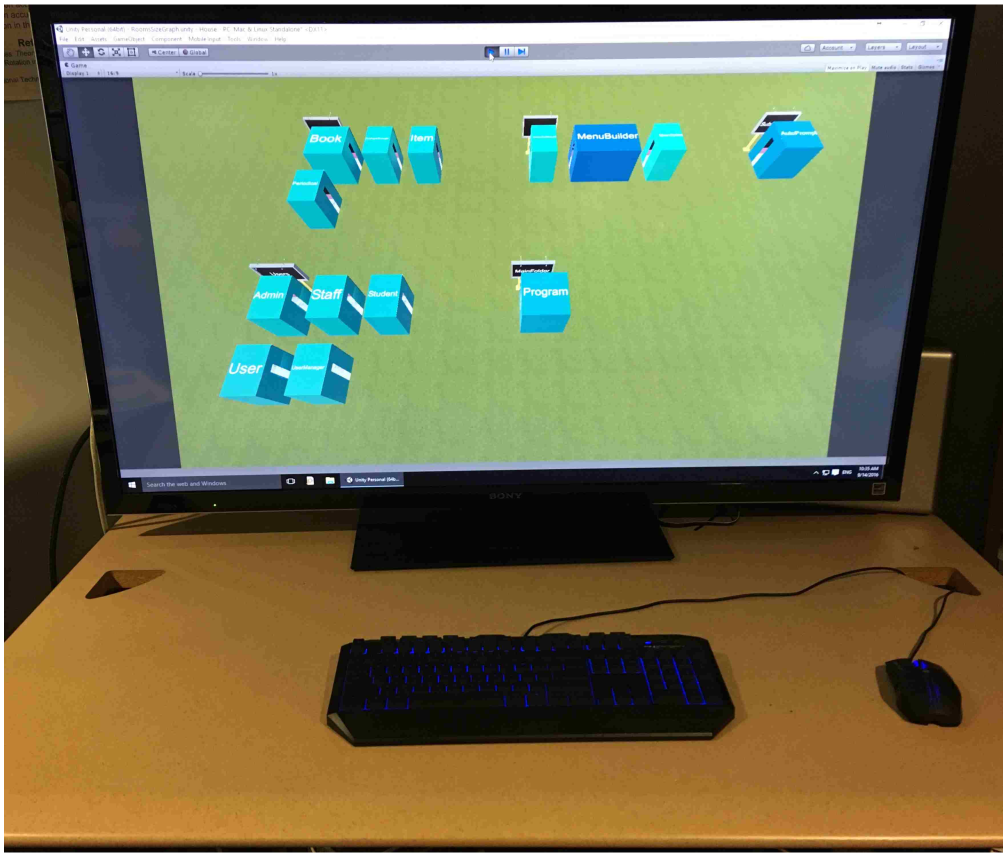


Figure 6.1: The user study setup.

Table 6.1: Comparison of the two code bases used in the user studies: Library Manger (LM) and Memory Game (MG). *LoC* is lines of code reported by the line counting program *cloc* .[2]

Code Base	**No. Classes**	**LoC**	**LoC (largest class)**
LM	14	977	237
MG	16	1753	791

Experiment Design and Procedure

When comparing Visual Studio with Code Park, there are a few considerations involved in order to design a sound experiment that allows a fair comparison between the two tools. First, Visual Studio has been in development for many years and most C# developers work with Visual Studio frequently. As a result, it could be the case that the users who use Visual Studio frequently are biased towards Visual Studio, because they have had more time to work and get comfortable with it. Accordingly, devising a fair between-subject design would be difficult.

Second, focusing on a purely within-subject design presents other complications. It is important to avoid any unwanted learning effects in a within-subject design when the user is working with both tools. Given a particular code base and a set of tasks, if a user performs those tasks in Visual Studio and then switches to Code Park to perform the same set of tasks, chances are that those tasks are performed much faster the second time. This is because the user will have learned the code base's structure the first time and can leverage that knowledge in Code Park. To avoid this learning effect, the user should be given two different code bases to work with. However, care must be taken when selecting the two code bases, as they should be relatively similar in structure and the degree of difficulty.

[2]`http://cloc.sourceforge.net/`

Table 6.2: Different experiment groups. Group names are only for tracking the number of participants in each permutation of the study.

Group Name	First Tool	First Code Base	Second Tool	Second Code Base
A	Visual Studio	LM	Code Park	MG
B	Code Park	MG	Visual Studio	LM
C	Visual Studio	MG	Code Park	LM
D	Code Park	LM	Visual Studio	MG

Having two code bases may pose another problem. Even if the two code bases are specifically chosen to be similar, minor differences between the two could affect the results. Moreover, studying and learning someone else's code can quickly become tedious and the users can become fatigued after using the first tool, affecting their performance in the second tool. Therefore, it is imperative to mitigate any of these unwanted effects in the study.

Facing with all these considerations, we opted to use a mixed-effects design for our study to benefit from both of the design modes. In our experiments, each participant used two different code bases with both tools. Several code bases were considered at first and after a few pilot runs, two code bases were carefully selected. These selected code bases shared similar structures and properties. One code base is a console-based library catalog system called Library Manager (LM). The other code base is a console-based card matching game called Memory Game (MG). The summary of these code bases are shown in Table 6.1. Note that even though MG contains more lines of code, some of its largest classes contain mostly duplicate code that handle drawing the shape of each playing card on the console window. Our assumption with the choice of code bases is that the two code bases are not significantly different and would not bias our results[3].

[3]This assumption will later be examined in the Discussion section.

Table 6.3: Participant tasks. Each task was timed. We used these measurements for our quantitative analysis.

	Task
T1	Find a valid username to login into the program.
T2	Find an abstract class in the code base.
T3	Determine the relationship between classes *A* and *B*.
T4	Find a desgined bug that causes a program crash.
T5	Pinpoint a reasonable location in the code for adding the necessary logic to support feature *X*.

To avoid the unwanted effects discussed previously, we permuted the order of tools and code bases across participants. As a result, each participant started the experiment with either Visual Studio or Code Park. Also their first tasks were performed on either LM or MG. The possible permutations divided our participants into four groups detailed in Table 6.2. By recruiting 28 participants, we had 7 participants per group. We randomly assigned a participant to a group. This results in a balanced mixed-effects design in which all possibilities and orders are considered.

On each code base, the participants are asked to perform five tasks, each with a varying degree of difficulty. These tasks are presented in Table 6.3. Among these, some tasks force the participant to explore the code base, whereas other tasks are directly related to a participant's understanding of the code base, the program structure and the logic behind it. Tasks T1 and T2 are similar for both code bases. Task T3 asks about the object-oriented relationship between classes *A* and *B*. In LM, this relationship is inheritance and in MG this relationship is having a common parent class (*A* and *B* are siblings)[4]. Prior to being tasked with T4, the participants are asked to try out a particular feature of the program. Upon the trial, the program crashes prematurely and no output is produced. The participants are told that the reason for this behavior is a simple intentional bug and are tasked with

[4]The selected classes were the same for all participants.

finding the likely location of the bug. For task T5, the participants are asked to imagine a scenario where somebody asks them about their approach for adding a new feature to each program. In LM, they are asked to add additional menu options for specific types of users. In MG, they are asked to modify the scoring system to penalize the player for each mistake.

We should note that none of these tasks involve writing any code. This was necessary because in this version, Code Park did not incorporate a code editor. Also, we are primarily interested in determining the effects of Code Park on code understanding. The participants are responsible for showing a suitable location in the logic to add these features. There are multiple valid locations for each task in each code base and the participants are free to select any of the valid locations. When performing any of these tasks, the users are explicitly told that they are not allowed to use the debugging features of Visual Studio, nor the search functionality for finding a specific type or class. They are, however, permitted to use VS's built-in *go-to definition* feature by holding down the control key and clicking with the mouse on a user-defined type. After performing the tasks, the participants are given a few questionnaires to fill out.

Metrics

For quantitative data, we recorded the time each participant took to perform each task. The participants were not told that their performance was being timed, in order to avoid stressing them.

The qualitative measurement was performed using the post-task and post-study questionnaires. After performing the tasks with each tool, the participants were given a post-task questionnaire that asked them to share their experience about the tool they worked with. These questions were the same for both tools and are shown in Table 6.4. The responses to these questions were measured on a 7-point Likert scale (1 = the most negative response, 7 = the most positive response).

Table 6.4: Post-task questionnaire. Participants answered these questions on a 7-point Likert scale after finishing their tasks with both tools. We used this data for our qualitative analysis.

	Post Task Questionnaire
Q1	I found it easy to work with Code Park/Visual Studio.
Q2	I found it easy to become familiar with Code Park/Visual Studio.
Q3	Code Park/Visual Studio helps me become familiar with code base's structure.
Q4	It was easy to navigate through the code with Code Park/Visual Studio.
Q5	It was easy to find the definition of some variable with Code Park/Visual Studio.
Q6	How much did you like Code Park/Visual Studio?
Q7	How did you feel when using the tool?
Q8	It was easy to find what I wanted in the code using Code Park/Visual Studio.

Upon the completion of all tasks, the participants were given a post-study questionnaire to measure their preferences of both tools from different aspects. This questionnaire is detailed in Table 6.5. The participants were required to select either Visual Studio or Code Park in their responses to each question.

Results

As mentioned in the previous section, we recorded quantitative as well as qualitative data. To analyze these data, we divided all of our participants into two equally sized groups based on their experience in C# and software development. We leveraged the results of the C# skill-test as well as the self-declared answers to determine the skill level of each participants. The skill-test questions were weighted twice as much in order to reduce potential over- or undervaluation of the self-declared responses.

Table 6.5: Post-study questionnaire. Participants answered these questions after finished all their tasks with both tools. The answer to each question is either Code Park or Visual Studio. We used this data for qualitative analysis.

	Post Study Questionnaire
SQ1	Which tool is more comfortable to use?
SQ2	Which tool is more likable?
SQ3	Which tool is more natural?
SQ4	Which tool is easier to use?
SQ5	Which tool is more fun to use?
SQ6	Which tool is more frustrating?
SQ7	Which tool helps you more in remembering the code base structure?
SQ8	Which tool do you prefer for learning a code base?
SQ9	Which tool do you prefer for a code base you are already familiar with for additional work?
SQ10	Which tool do you prefer for finding a particular class/variable?
SQ11	Which tool do you prefer for tracking down a bug?
SQ12	Overall, which tool is better?

As a result, our experiments have three factors: tool, code base and experience. The tool factor has two levels: Code Park or Visual Studio, the code base factor has two levels: LM or MG and the experience factor has two levels: beginner or expert.

Quantitative Results

Our quantitative results are based on the time a participant took to complete an assigned task. When validating ANOVA assumptions, we found that most group response variables failed the Shapiro-Wilk normality tests. Since our factorial design contains 3 factors (Code base$\times$Tool$\times$Experience) with two levels each, we decided to use the Aligned Rank Transform (ART) [38] to make our data suitable for analysis with 3-way ANOVA [39]. The analysis results are presented in Table 6.6.

Table 6.6: ANOVA results on the quantitative data (time to task completion). Statistically significant results (95% confidence) are highlighted in gray. The first column (TT) is task time. Code base is abbreviated as *CB*. Experience level is abbreviated as *Exp*.

TT	**CB**		**Tool**		**Exp.**		**Tool×CB**		**CB×Exp.**		**Tool×Exp.**		**Tool×CB×Exp.**	
	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value
T1	3.92	$= 0.06$	12.0	< 0.05	7.53	< 0.05	0.78	$= 0.39$	0.49	$= 0.49$	0.19	$= 0.66$	0.10	$= 0.75$
T2	1.18	$= 0.29$	0.32	$= 0.58$	3.71	$= 0.07$	0.46	$= 0.50$	0.63	$= 0.44$	0.04	$= 0.84$	0.46	$= 0.51$
T3	45.2	< 0.05	15.9	< 0.05	5.70	< 0.05	1.02	$= 0.32$	0.58	$= 0.46$	6.01	< 0.05	0.00	$= 0.95$
T4	0.70	$= 0.41$	0.03	$= 0.86$	3.56	$= 0.07$	0.15	$= 0.70$	0.11	$= 0.74$	0.10	$= 0.76$	0.71	$= 0.41$
T5	7.71	< 0.05	41.9	< 0.05	0.08	$= 0.78$	3.93	$= 0.06$	1.35	$= 0.26$	0.32	$= 0.58$	0.05	$= 0.82$

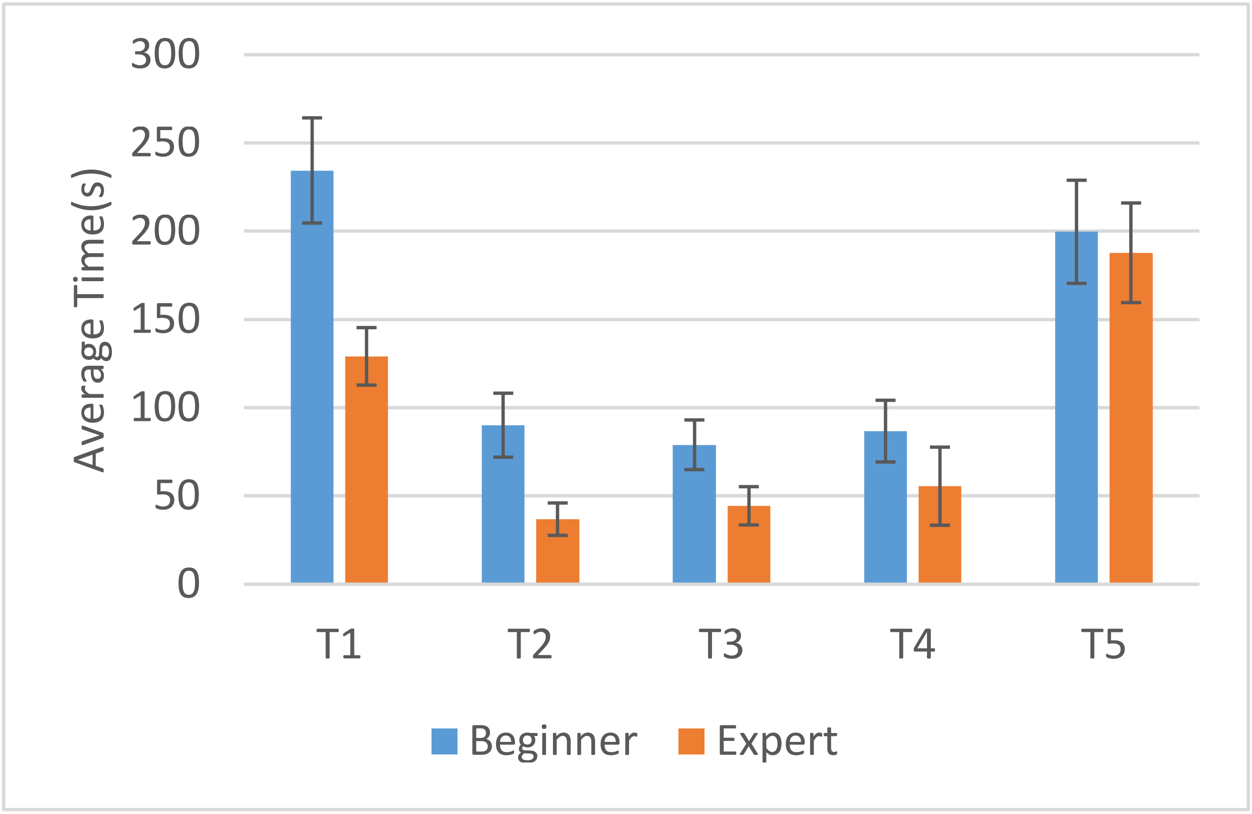


Figure 6.2: Mean time to task completion by experience level.

Our tests were not powerful enough to detect significant main effects or interactions in T2 and T4. For T1, the participants who used Visual Studio completed the task significantly faster than those who used Code Park ($F_{1,24} = 12.00,\ p < 0.05$) (see Figure 6.3)[5]. The average task time was 136.82 seconds (σ = 135.48) and 226.71 seconds (σ = 124.15) for Visual Studio and Code Park respectively. Also the participants who were experts finished the task significantly faster than the beginners ($F_{1,24} = 7.53,\ p < 0.05$). The average task time was 129.10 seconds (σ = 86.11) and 234.42 seconds (σ = 157.63) for experts and beginners respectively (see Figure 6.2).

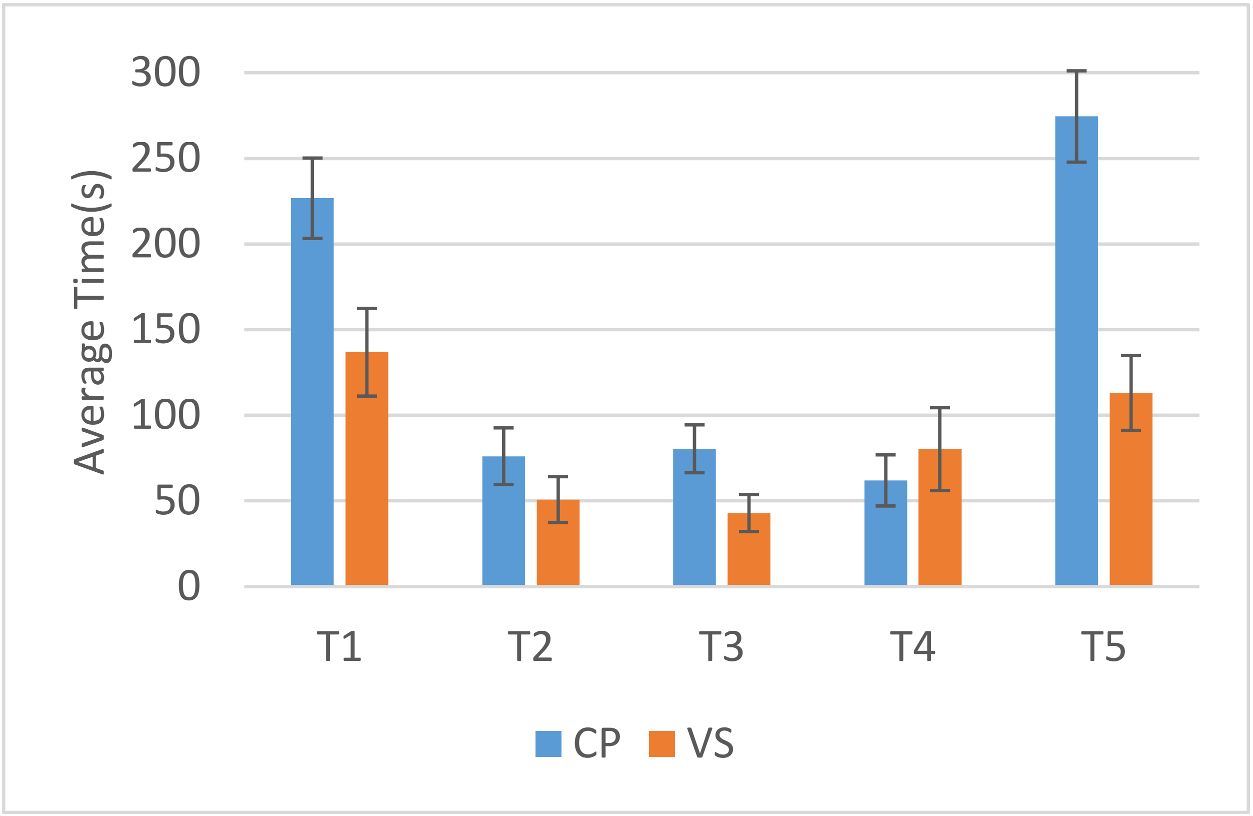


Figure 6.3: Mean time to task completion based on the tool used.

[5]All error bars are standard error values.

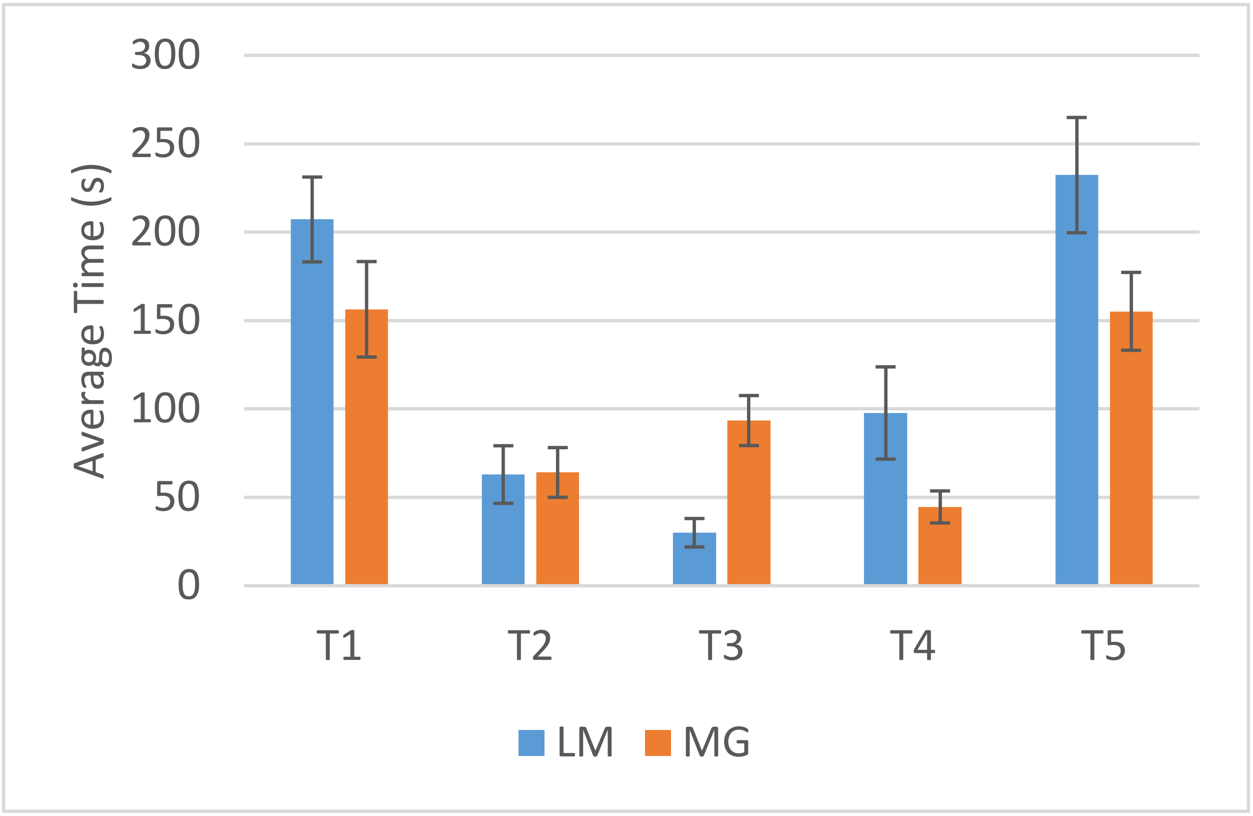


Figure 6.4: Mean time to task completion based on the code base.

In T3, statistics were similar to T1 for both tool and experience. The participants who used Visual Studio completed the task significantly faster than those who used Code Park ($F_{1,24} = 15.9$, $p < 0.05$). The average task time was 42.96 seconds (σ = 57.05) and 80.46 seconds (σ = 73.92) for Visual Studio and Code Park respectively. Also the participants who were experts finished the task significantly faster than the beginners ($F_{1,24} = 5.70$, $p < 0.05$). The average task time was 44.42 seconds (σ = 57.31) and 79.00 seconds (σ = 74.43) for experts and beginners respectively. In addition we find two more significant differences in T3. The participants finished the task significantly faster with LM compared to MG ($F_{1,24} = 45.20$, $p < 0.05$). The average task time was 30.00 seconds (σ = 42.49) with LM and was 93.42 seconds (σ = 74.87) with MG (see

Figure 6.4). The other significance shows the interaction effect[6] between tool and experience ($F_{1,24} = 6.01,\ p < 0.05,\ \eta_p^2 = 0.20$ (L)). The average task time for experts who used Visual Studio was 38.21 seconds (σ = 60.86) and was 50.64 seconds (σ = 52.81) for those who used Code Park. The average task time for beginners who used Visual Studio was 47.71 seconds (σ = 52.54) and was 110.29 seconds (σ = 79.76) for those who used Code Park.

In T5 the participants who worked with MG finished the task significantly faster that the ones who worked with LM ($F_{1,24} = 7.71,\ p < 0.05$). The average task time was 232.25 seconds (σ = 172.43) for LM and was 155.21 seconds (σ = 116.54) for MG. Also the participants who used Visual Studio have done it significantly faster than those who used Code Park ($F_{1,24} = 41.91,\ p < 0.05$). The average task time was 113.03 seconds (σ = 115.61) for Visual Studio and was 274.42 seconds (σ = 141.03) for Code Park.

Qualitative Results

Our qualitative results are comprised of two parts. The first part consists of the responses of the participants to the Likert-scale questions in the post-task questionnaire. The second part consists of the responses of the participants to the questions in the post-study questionnaire.

Post-Task Questionnaire

The responses to the Likert-scale questions in our post-task questionnaire failed the Shapiro-Wilk normality tests. As such, similar to our quantitative results, we used ART [38] to make our data suitable for analysis with 3-way ANOVA [39]. The summary of our data analyses is shown in Table 6.7. Average ratings for the eight post-task questions are shown in Figure 6.5.

[6]The effect sizes are reported as η_p^2 and described as large (L), medium (M) or small (S).

Table 6.7: ANOVA results on the qualitative data (post-task questionnaire). Statistically significant results (95% confidence) are highlighted in gray. Code base is abbreviated as *CB*. Experience level is abbreviated as *Exp*.

Question	CB		Tool		Exp.		Tool×CB		CB×Exp.		Tool×Exp.		Tool×CB×Exp.	
	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value	$F_{1,24}$	p-value
Q1	1.00	$= 0.33$	0.94	$= 0.34$	1.94	$= 0.18$	2.35	$= 0.14$	0.55	$= 0.47$	9.10	< 0.05	0.06	$= 0.81$
Q2	0.91	$= 0.35$	7.98	< 0.05	7.81	< 0.05	2.45	$= 0.13$	0.61	$= 0.44$	0.00	$= 0.98$	0.00	$= 0.98$
Q3	0.02	$= 0.89$	14.2	< 0.05	2.35	$= 0.14$	0.70	$= 0.41$	0.57	$= 0.46$	0.04	$= 0.84$	0.08	$= 0.77$
Q4	0.10	$= 0.75$	2.10	$= 0.16$	8.18	< 0.05	0.34	$= 0.56$	0.12	$= 0.74$	0.81	$= 0.38$	0.38	$= 0.54$
Q5	0.97	$= 0.33$	0.16	$= 0.69$	1.07	$= 0.31$	4.44	< 0.05	0.02	$= 0.90$	0.23	$= 0.64$	0.16	$= 0.69$
Q6	0.38	$= 0.54$	0.28	$= 0.60$	3.35	$= 0.08$	0.14	$= 0.71$	0.00	$= 0.97$	0.09	$= 0.77$	0.07	$= 0.80$
Q7	0.09	$= 0.77$	1.10	$= 0.30$	11.3	< 0.05	0.43	$= 0.52$	0.05	$= 0.82$	0.10	$= 0.75$	0.28	$= 0.60$
Q8	0.11	$= 0.74$	0.27	$= 0.61$	3.04	$= 0.09$	0.27	$= 0.61$	0.67	$= 0.42$	0.03	$= 0.86$	0.00	$= 0.95$

Table 6.8: Chi-squared analysis on the post-study responses. Statistically significant results (95% confidence) are highlighted. The numbers in Visual Studio and Code Park columns represent the total times each tool was selected by the participant in response to a question. The winner in each significant category is highlighted.

Question	Visual Studio	Code Park	Chi-squared Test	
SQ1	21	7	$\chi^2(1,\ N = 28)\ = 7.00$	$p < 0.05$
SQ2	8	20	$\chi^2(1,\ N = 28)\ = 5.14$	$p < 0.05$
SQ3	16	12	$\chi^2(1,\ N = 28)\ = 0.57$	$p = 0.45$
SQ4	17	11	$\chi^2(1,\ N = 28)\ = 1.29$	$p = 0.26$
SQ5	0	28	$\chi^2(1,\ N = 28)\ = 28.00$	$p < 0.05$
SQ6	11	13	$\chi^2(1,\ N = 24)\ = 0.17$	$p = 0.68$
SQ7	5	23	$\chi^2(1,\ N = 28)\ = 11.57$	$p < 0.05$
SQ8	7	21	$\chi^2(1,\ N = 28)\ = 7.00$	$p < 0.05$
SQ9	19	9	$\chi^2(1,\ N = 28)\ = 3.57$	$p = 0.06$
SQ10	15	13	$\chi^2(1,\ N = 28)\ = 0.14$	$p = 0.71$
SQ11	19	9	$\chi^2(1,\ N = 28)\ = 3.57$	$p = 0.06$
SQ12	16	11	$\chi^2(1,\ N = 27)\ = 0.93$	$p = 0.34$

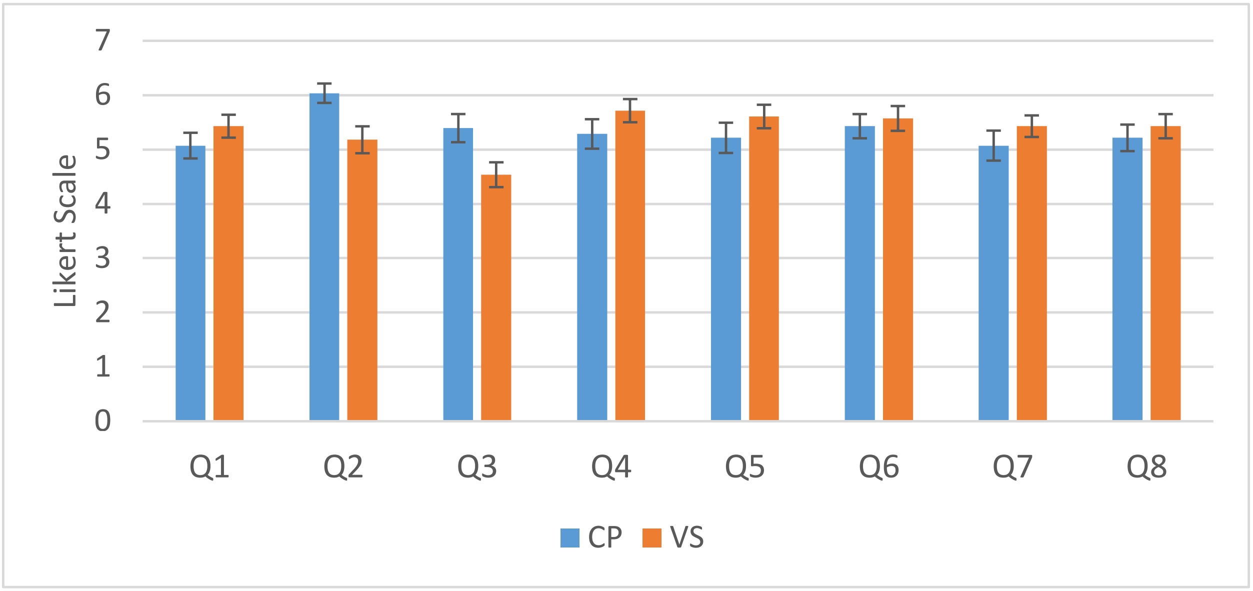


Figure 6.5: Mean responses to the post-task questionnaire for Code Park and Visual Studio.

Our tests did not detect significant main effects caused by the code base on the participants' responses. However, the tool had a significant difference on the responses to the questions, namely Q2 and Q3. Results show that the participants found it easier to become familiar with Code Park compared to Visual Studio ($F_{1,24} = 7.98,\ p < 0.05$). They also found Code Park to be more helpful in becoming familiar with code base's structure compared to Visual Studio ($F_{1,24} = 14.21,\ p < 0.05$). The significant differences based on experience is not related to our study. Also there was an interaction effect between tool and experience on the responses to Q1 ($F_{1,24} = 9.10,\ p < 0.05,\ \eta_p^2 = 0.27$ (L)). There was also another interaction effect between tool and code base on the responses to Q5 ($F_{1,24} = 4.44,\ p < 0.05,\ \eta_p^2 = 0.37$ (L)).

Post-Study Questionnaire

The post-study questionnaire requires the participant to pick one tool for each question (see Table 6.5). Since each question had only two choices, we used Chi-squared test to analyze the results. The summary of our data analysis is shown in Table 6.8. We noticed that a few participants left some of the questions unanswered.

The results indicate that there was a significant difference in being comfortable with using Visual Studio compared to Code Park ($\chi^2(1,\ N = 28)\ \ = 7.00,\ p < 0.05$). Also Code Park was significantly more likable than Visual Studio ($\chi^2(1,\ N = 28)\ \ = 5.14,\ p < 0.05$). Every single participant believed that Code Park was more fun to use than Visual Studio ($\chi^2(1,\ N = 28)\ \ = 28.00,\ p < 0.05$). Code Park also helped the participants in remembering code base's structure significantly more than Visual Studio ($\chi^2(1,\ N = 28)\ \ = 11.57,\ p < 0.05$). For learning a code base the participants preferred Code Park significantly more than Visual Studio ($\chi^2(1,\ N = 28)\ \ = 7.00,\ p < 0.05$). In other questions there was no significant deference between Code Park and Visual Studio.

Result and Discussion

The goal of our user study was to determine the degree to which we achieved our design goals with Code Park. Specifically, we were interested in determining how easy it is to get familiar with Code Park, how much it helps in understanding a code base, how the users feel about working with it and how it compares against a traditional IDE such as Visual Studio. Our results can be discussed from various aspects.

On the tool level and from a qualitative aspect, it is evident that the participants found Code Park significantly easier than Visual Studio to get familiar with, even though all participants had prior familiarity and experience with Visual Studio. The participants also found Code Park to be significantly more beneficial in becoming familiar with a code base compared to Visual Studio. Both of these results were obtained regardless of the participant's experience, *i.e.* both experts and beginners found Code Park to be superior than Visual Studio in these two categories. Referring to the post-study questionnaire, we see that the participants believed Code Park to be more likable compared to Visual Studio (see Table 6.5). Further, they believed that Code Park not only helped in learning the code structure but also helped them in remembering the code they studied and also finding the code elements that they were looking for. The interesting observation here is that all 28 participants unanimously believed that Code Park was more enjoyable which indicates that we achieved our goal of designing an engaging tool. Note that the results in all these categories were statistically significant. These results are further corroborated by the written feedback that our participants provided. Here we provide some of them:

Participant 1: *"I enjoyed a lot as it was the first time I was viewing the code in 3D environment."*

Participant 4: *"It is nice visual representation of the code."*

Participant 5: *"It is a very nice and visual way for people to conceptualize object oriented program due to the class association with houses."*

Participant 6: *"It is much easier to understand the overall code structure."*

Participant 10: *"It is interactive and keep me engaged."*

Participant 13: *"UI is good."*

Participant 17: *"It is very friendly and easy."*

Participant 19: *"We could easily access classes and methods."*

Participant 20: *"Easy to look thorough classes/methods and how different classes/methods interact."*

Participant 22: *“It is easy to navigate with interactive interface.”*

Participant 23: *“You don’t go through the whole code to find something.”*

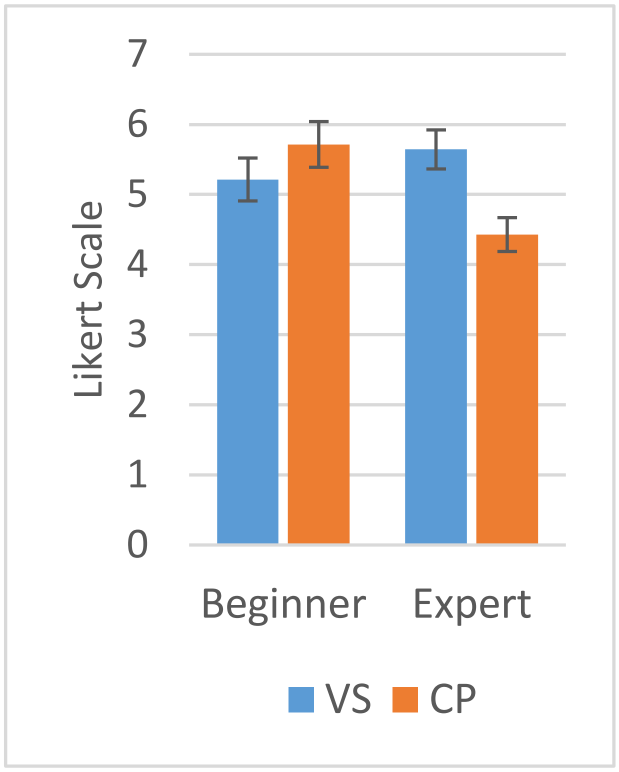
Participant 26: *“The use of spatial representation was well used.”*

Participant 27: *“Generally easier [than Visual Studio] to understand the structure of the code”*

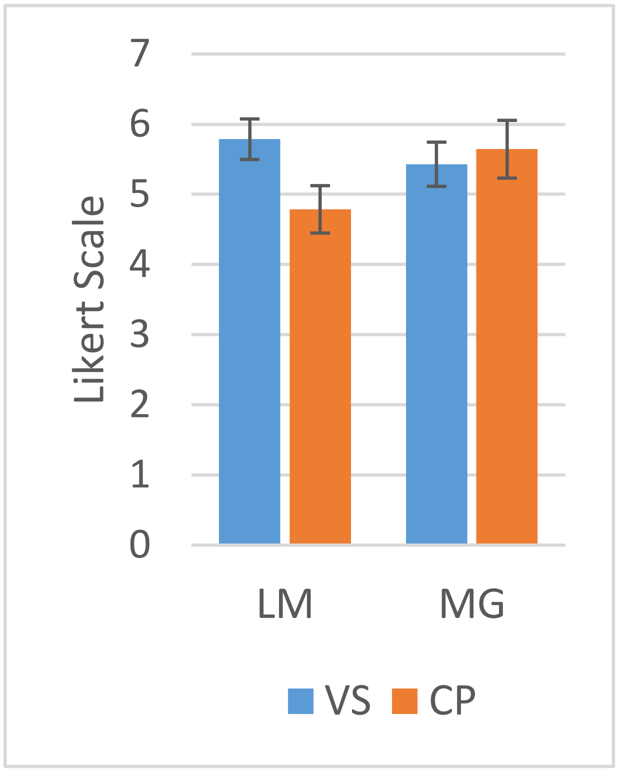
In the remaining qualitative categories, there was no significant difference observed in users’ responses between Code Park and Visual Studio. This is again interesting because Visual Studio has been in development since 1997 and has gone through many iterations and refinements, whereas Code Park has only been in development for about 6 months.

Our results indicate an interaction between the tool and the participants’ experience levels in responses to Q1. A closer look at the responses reveals that the beginners found Code Park to be easier to work with. Conversely, experts found Visual Studio to be easier (see Figure 6.6a). This coincides with what one would expect: the prior experience of the experts with Visual Studio gives them an edge in performing the assigned task. As a result, those participants found Visual Studio easier to work with compared to Code Park. When asked about their opinion, one user described Code Park as *”an amusement park for beginners”*. Another user told us: *”With more polish (smoother experience), I think there is promise for Code Park (especially for new programmers)”*.

We observe an interaction effect between tool and the code base in the responses to Q5. The mean responses for this question are detailed for each code base in Figure 6.6b. From LM to MG, we see an increase in the mean response values for Code Park, but a decrease in the responses for Visual Studio. One possible explanation for the small superiority of Code Park over to Visual Studio on MG is that MG contains more code with larger classes (see Table 6.1). It could be that this larger size and the existence of more clutter give Code Park a slight edge when finding the definition of



(a) Q1



(b) Q5

Figure 6.6: (a) Difference of averages between Visual Studio and Code Park grouped by user experience for Q1 responses. (b) Difference of averages between Visual Studio and Code Park grouped by code base for Q5 responses.

variables. This could potentially lead to the conclusion that Code Park is a more favorable tool for larger projects. However, this conclusion is premature and we believe a more detailed study is warranted to examine this interaction in more details.

On the tool level and from a quantitative point of view, it is evident that the choice of tool had a significant effect on time to completion of T1, T3 and T5. In all these cases, participants took significantly less time to complete their tasks with Visual Studio compared to Code Park. There are several possible explanations for this observations.

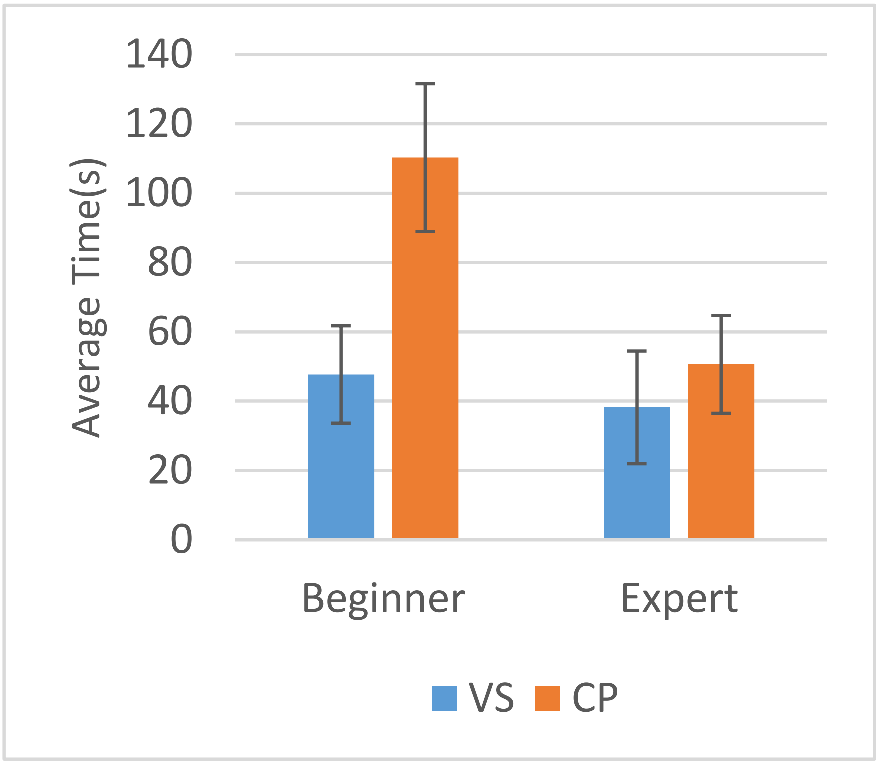


Figure 6.7: The average time spent on completing T3 based on each tool and the participants' experience level.

The first is the existence of transition animations in Code Park. As discussed in previous chapter, the animations were necessary to preserve the user's sense of environmental awareness. At any instance of time, the length of these animations depend on the camera's relative position to each code room and also the size of the code base. Nevertheless, every animation was at least 1.5 seconds long. Considering a hypothetical situation where a user is in god's view and wants to switch to first-person mode to inspect a code room and then go back to god's view, the transition animations take about 3 seconds. It takes a significantly less amount of time to open two files consecutively in Visual Studio.

Another possible explanation for observing faster task completion times with Visual Studio is that, Code Park provides more interaction capabilities compared to Visual Studio. Also 3D interactions are inherently slower compared to 2D interactions because of the added degree of freedom. When a user is inside a code room in Code Park, they have the freedom of walking around, looking at the wallpapers or clicking on the wallpapers to inspect the code more closely. All these interactions take time. We also gained some insight about this issue from a different perspective. Often times, after we assigned a task to a participant, we noticed that some participants started to "play" with the interface or wander about aimlessly for a few seconds: due to the game-like environment of Code Park, they occasionally walked around the room and inspected the visual aspects of the environment, or would make a verbal note about something in the environment that was irrelevant to the assigned task (such as how realistic the reflection effect was on the floor tiles, or how the grass texture looked unrealistic).

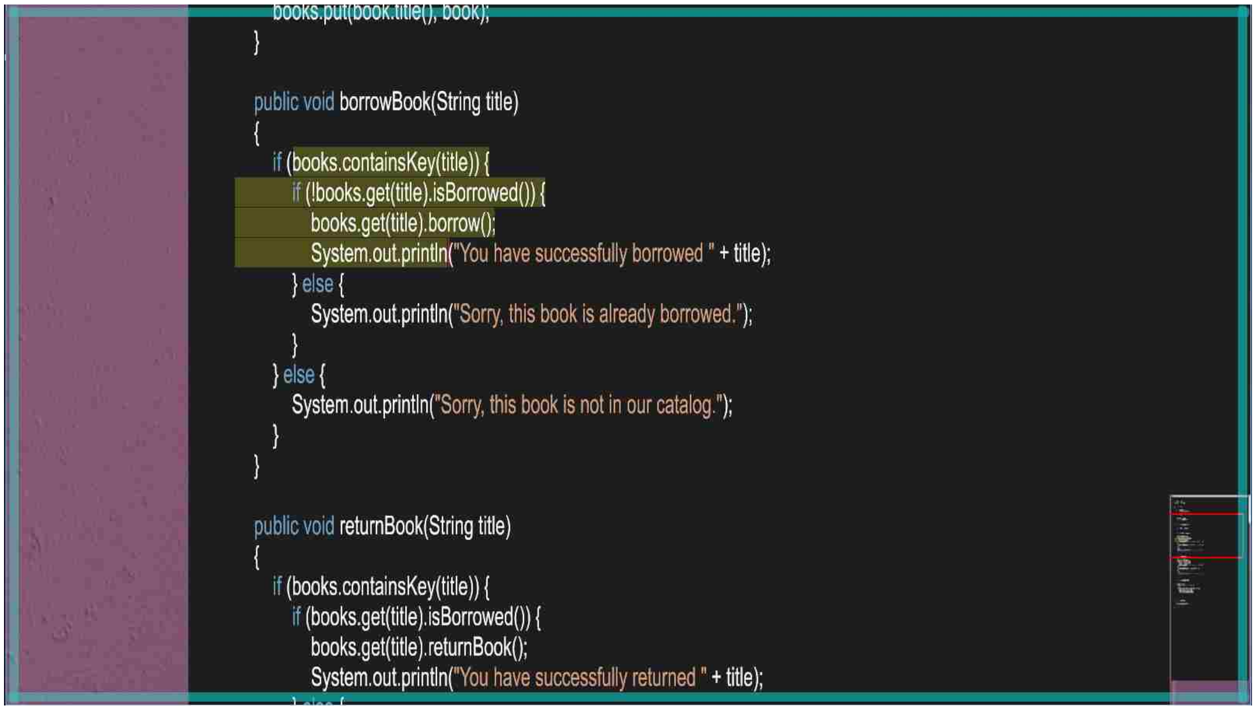
Focusing on the quantitative results based on the experience level of the participants, reveals other findings. As one would expect, and regardless of the tool, beginners generally took more time to finish their tasks compared to experts. The other result of interest is the observed interaction effect between the tool and the experience of the participants. Figure 6.7 presents the average time spent on T3 based on the each tool and the participants' experience level. The increase in the average time spent by the beginners when they switched from Visual Studio to Code Park was much more than this increase for experts. Without a more detailed study, it is difficult to draw any concrete conclusions. Informally, we think a possible explanation could be that it takes more time for the beginners to realize they do not know how to tackle T3. As a result, they take their time and explore the code further with Code Park, hoping to find a clue that aids them in completing the task.

Considering the code base aspect of our study and focusing on the quantitative results, we see that the choice of code base only affected the time spent for completing T3 and T5. This observation can be explained by noting the difference in the sizes of the two code bases. As detailed in Table 6.1, MG is larger and more cluttered than LM. For T5, where the participants are asked to go through the code, it generally took them longer to browse through MG compared to LM. Other than task completion time, the choice of code base did not significantly affect the qualitative results. This bolsters our initial assumption that the two code bases were very similar and the choice of the code base, would not significantly affect our results.

CHAPTER 7: CODE PARK V2 - 3D IDE

In this iteration, we focused on adding two main features that most of the participants from the previous study asked for. The two features were editing code and compiling code within the Code Park environment. With these two features, Code Park became more and more like a real and usable IDE. We also added support for Java because it is one of the most popular languages based on the tutorials that are searched for it on the web[1]. Users also can create a new project from scratch and start to work on it with Code Park. This feature also supports adding a new class and placing this class in a desirable location. They also can replace or even delete an existing class. All the user's project data (all the classes and their location in Code Park) will save in a file so the users can continue working on their project later.

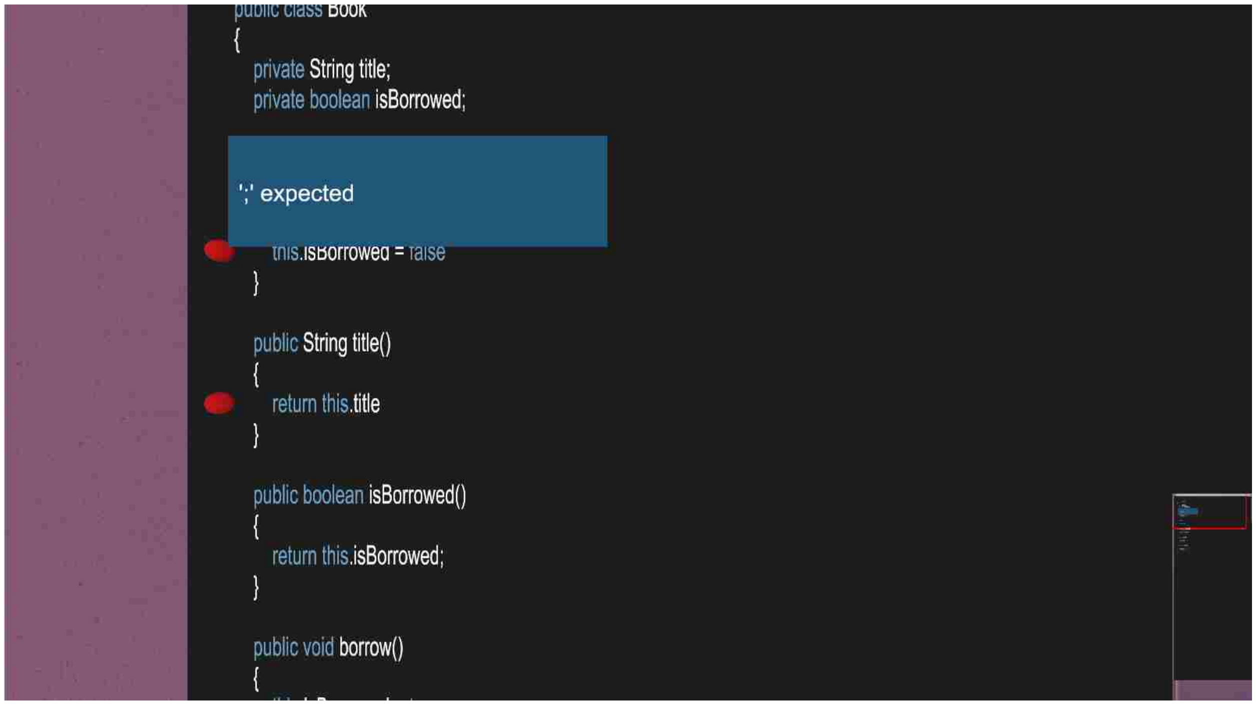
[1]`http://pypl.github.io/PYPL.html`



```
        books.put(book.title(), book);
    }

    public void borrowBook(String title)
    {
        if (books.containsKey(title)) {
            if (!books.get(title).isBorrowed()) {
                books.get(title).borrow();
                System.out.println("You have successfully borrowed " + title);
            } else {
                System.out.println("Sorry, this book is already borrowed.");
            }
        } else {
            System.out.println("Sorry, this book is not in our catalog.");
        }
    }

    public void returnBook(String title)
    {
        if (books.containsKey(title)) {
            if (books.get(title).isBorrowed()) {
                books.get(title).returnBook();
                System.out.println("You have successfully returned " + title);
```

Figure 7.1: Editing code on wall with all the basic editor functionality. The blue border helps users to know that they are in editing mode.

Users can edit a class by going to the code viewing mode and then start editing the code. We added a blue border around the screen in editing mode in order to help users understand that they are in a different mode. Editing mode supports all the main functionality of the basic editor such as selecting a part of text and copy, cut or pasting it (See Figure 7.1). All these functionalities implemented to work in the 3D environment and on the exact same code canvas that user reached. In this way users can maintain their spatial awareness and remember the position of the function that they wrote.

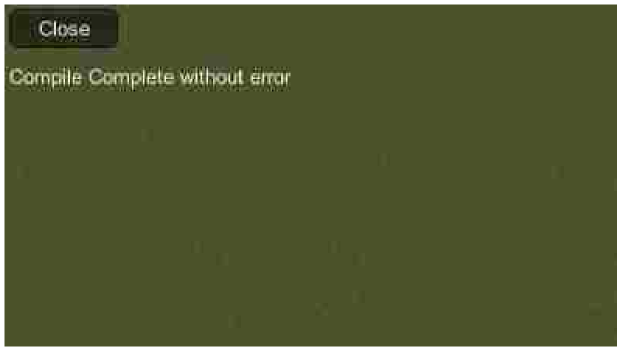


```
public class Book
{
    private String title;
    private boolean isBorrowed;

';' expected
        this.isBorrowed = false
    }

    public String title()
    {
        return this.title
    }

    public boolean isBorrowed()
    {
        return this.isBorrowed;
    }

    public void borrow()
    {
```

Figure 7.2: Showing error on code canvas. Users can hover the mouse over the red dot and more detail about the bug in that line will be showed.

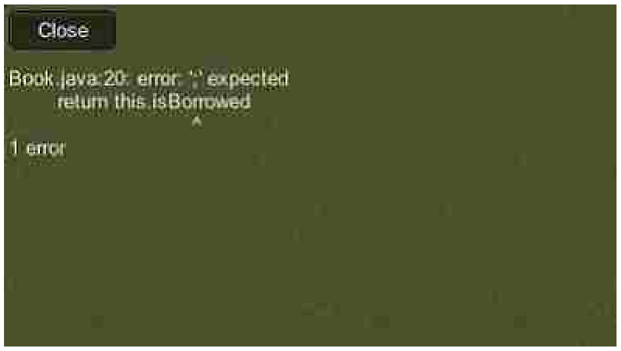
It is an essential part of every IDEs to have the ability to compile and run a code base. We added this ability as a built-in feature which helps users to compile and debug their own code without leaving the Code Park environment. In this version of Code Park we only support console application programs so we implemented a terminal within the Code Park to show the output of compiler and output of the developed program itself (see Figure 7.3). To do so we used Java JDK installed on users' machine. We sent all the compile instructions to the Microsoft Windows Command Prompt and read the output from it. Then we parsed the output to find out if the user's application compiled without errors or have not.

Based on the output' parse result it could have two different outcomes: the code, complied without any error and the terminal showed the successful note (see Figure 7.3a). The other outcome could happen when the code has some errors. In that case the terminal showed the output of the compiler (see Figure 7.3b). After we parsed the output, in the case of containing some errors we can determine the exact location of each errors in the code. With showing the error on the code canvas, users could find and resolve the bug much faster and easier like in the most common IDEs (see Figure 7.2). If program compiled without any error users can run their code and see the output of their application on the terminal (see Figure 7.3c).



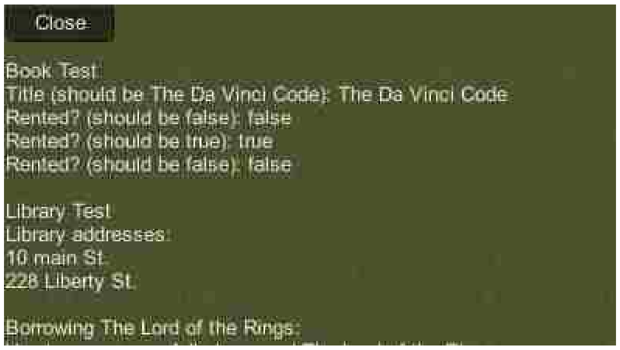
```
Compile Complete without error
```

(a) Compiled without error.



```
Book.java:20: error: ';' expected
        return this.isBorrowed
                              ^
1 error
```

(b) Compiled with errors



```
Book Test:
Title (should be The Da Vinci Code): The Da Vinci Code
Rented? (should be false): false
Rented? (should be true): true
Rented? (should be false): false

Library Test:
Library addresses:
10 main St.
228 Liberty St.

Borrowing The Lord of the Rings:
```

(c) Compiled program output

Figure 7.3: Compiler output

CHAPTER 8: CODE PARK V2 EVALUATION

Evaluation: First Part

As mentioned in previous chapter in this version of Code Park we focus on adding two main functionally that most common IDEs have: editing and compiling code. To evaluate these features and gauge the usability of our system, we designed and performed a new user study with two parts. We decided to use Java in our study therefore we recruited a bigger variety of users because it is one of the languages that was taught to freshmen students in Computer Science. We gave an assignment to the participants and asked them to work on it within a week on their own laptop. We were interested in studying the effects of regular developing with the Code Park.

Participants and Experience Design

We recruited 9 participants (8 males and 1 female ranging in age from 18 to 29 with a mean age of 22.8). Our requirements for participants were that they should be familiar with the Java language and also brought their own laptop. Each participant was compensated with $20 for their efforts ($5 for the first session and $15 for the second session). Like the previous study each participant was given a pre-questionnaire containing some demographic questions as well as some questions asking about their experience in developing Java applications.

After completing the pre-questionnaire, the participants were given a quick tutorial of Code Park and given a brief explanation of each feature of Code Park. Then the Code Park was installed on their laptop and an assignment was given to them to work on it within a week. They were asked for to implement two classes called Library and Book with requested functions. More details can be found in Appendix.

After a week the participants came back with the assignment completed. They were asked to complete a post-study questionnaire to share their overall experience. The responses to these questions were measured on a 7-point Likert scale (1 = the most negative response, 7 = the most positive response). This questionnaire is detailed in Table 8.1.

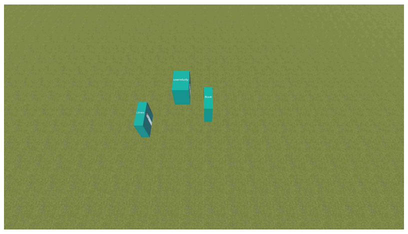
Table 8.1: Post-Study questionnaire. Participants answered these questions on a 7-point Likert scale. We used this data for our qualitative analysis.

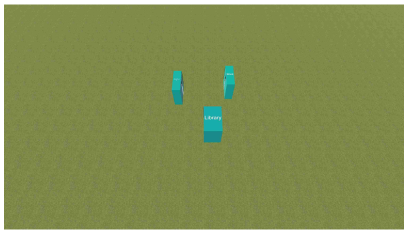
	Post Study Questionnaire
Q1	I found it easy how to use Code Park.
Q2	I found it easy how to learn using Code Park.
Q3	I think that I would like to use Code Park frequently.
Q4	I found the various functions in Code Park were well integrated.
Q5	It was easy to navigate through the code with Code Park.
Q6	It was easy to write code in Code Park.
Q7	It was easy to work on a project with Code Park.
Q8	How much did you like the Code Park?
Q9	How did you feel when using the interface?
Q10	It was easy to find what I wanted in the code using Code Park.

Results and Discussion

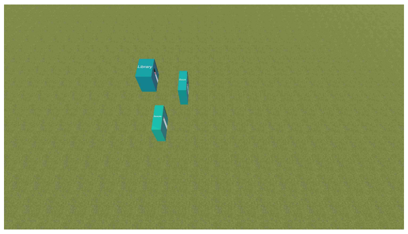
All the participants finished the assignment in the time frame that we asked which was a week. In our post-study questionnaire, we asked approximately how long did it take to finished the assignment? The mean time based on 9 participants response was approximately one and a half hour. After participants gave us their completed assignment we captured a screenshot to see how they placed their classes in Code Park environment.

As it shown in Figure 8.1 some of the participants finished the assignment with three classes while the others have done it with only two. That is because some of them created a separate class as the main class while the others wrote the main function in one of the two classes they created before. The other observation shows that the ones that used three classes, placed them that resemble a triangle shape. Also, the others with two classes positioned theirs in a horizontal manner instead of vertical. The possible explanation for this observation could be that the feature of the brain which tends to organize objects around it in a meaningful shapes.

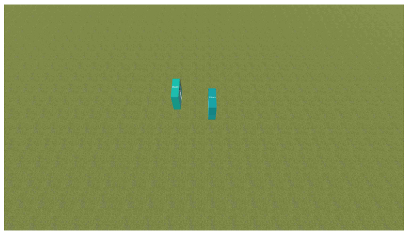

(a) Participant 1

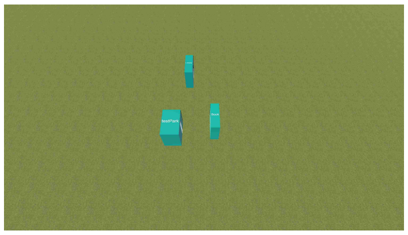


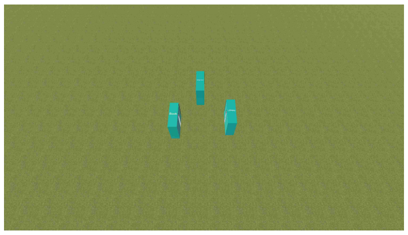
(b) Participant 2

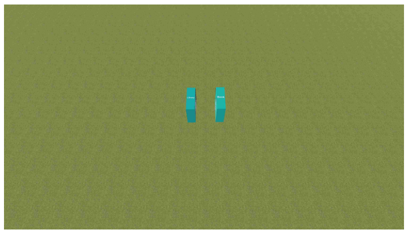


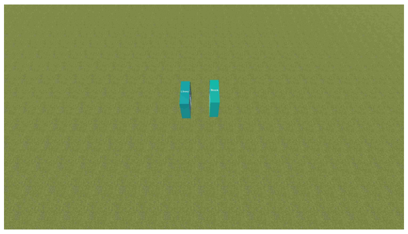
(c) Participant 3


(d) Participant 4


(e) Participant 5


(f) Participant 6


(g) Participant 7


(h) Participant 8

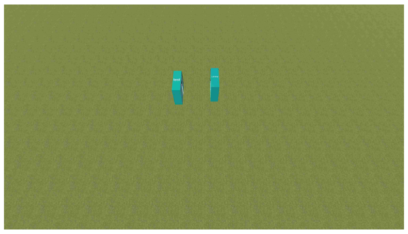

(i) Participant 9

Figure 8.1: Screenshots of participants completed assignment.

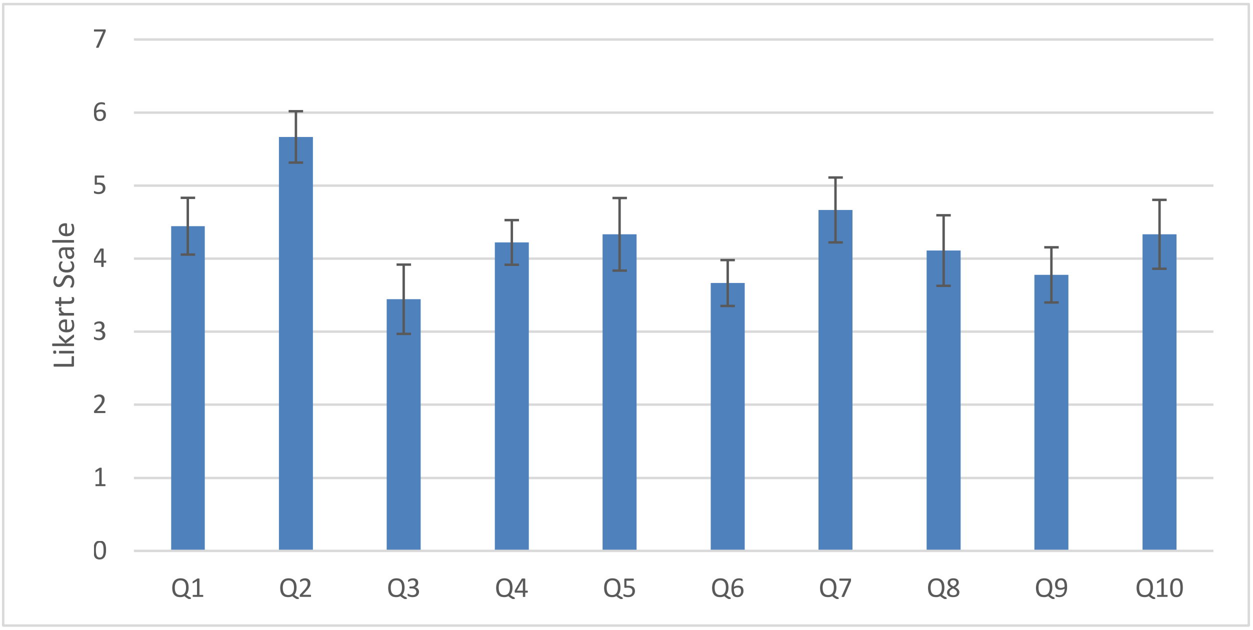


Figure 8.2: Mean responses to the post-study questionnaire.

The average rating for post-study questionnaire's questions are shown in Figure 8.2. This result can be discussed from various aspect. It shows that except Q3, Q6 and Q9 all the other answers are beyond the average scale. It is evident that the participants found easy how to learn using the Code Park (Q2), how to use the Code Park (Q1) and find what they wanted in the Code Park (Q10). Q7 also shows that it was easy for participants to work on a project with Code Park. Q4 indicates that participants found all the features and functions was well integrated into Code Park. Also, they believed that it was easy to navigate through code (Q5). Q8 also shows that how much they like Code Park.

A possible explanation for why the participants didn't want to use Code Park frequently (Q3) or why they didn't find writing code in Code Park easy (Q6) could be the 3D environment itself. All of the participants used to write code on a 2D text-based environment for their daily usage. Comparing to Code Park with all the animations for the transition between classes, 2D text-based environment is much faster and maybe that is the cause of the participants' frustration (Q9). Another explanation for these rating could be because we installed Code Park on their system they taught that this is the complete and fully functional software and any small bug make them frustrated. Also, lack of some features such as autocomplete or the option to see multiple files at the same time may lead to these result.

Generally, our results indicate that we achieved our goals to create an environment that is fun and engaging that users also can work on a project from scratch to the end and edit or compile a code base. The environment that based on some participants comments *"which you can easily swap between the class and see the code overview"* or *"easy to navigate through the classes which are cool"*. One of the other participants said that *"Code Park has wonderful UI and it was fun to use it"*.

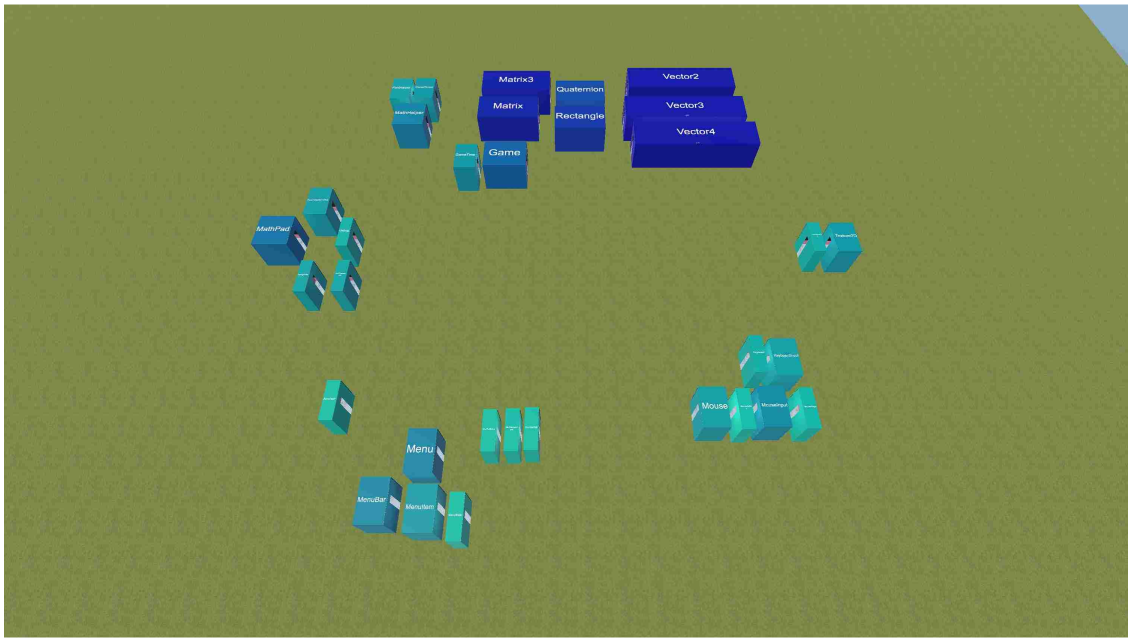
Evaluation: Second Part

As we mentioned we ran a user study with two parts. In the second part we were curious about the suitability of Code Park in the task of organizing an existing project. Therefore the users are tasked with organizing an existing project in Code Park in any way they saw fit. Each participant was tasked with placing all 33 classes of an existing project in CP's environment in a manner that they saw reasonable. As a result, they were free to organize the classes in any way they preferred. We separated our participants into two groups of 5 and 4.

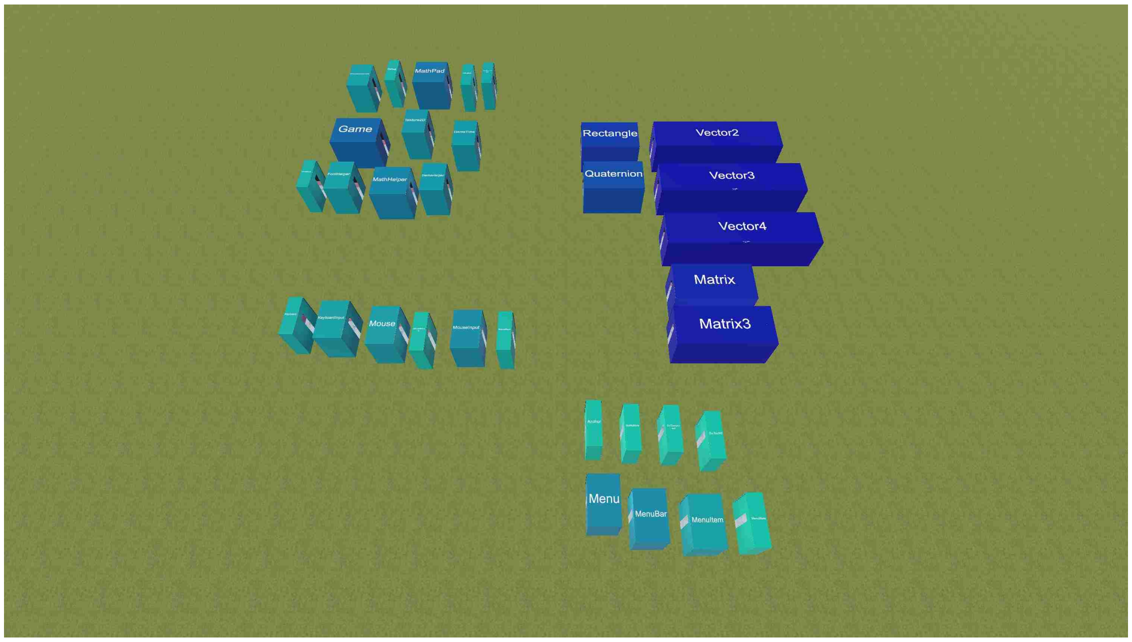
The classes in the project that was given to the first group were already organized into directories based on their relation (e.g. classes that handled user input were all inside of a directory called "input"). Conversely, the classes in the project that was given to the second group were not organized in any particular manner (i.e. all classes were inside the same directory). The goal of such separation was to observe whether grouping the classes based on their inherent relationships would affect users' decisions. At the end, they were asked to explain why they organize the classes like they did. All the questionnaires and assignment can be found in the Appendix.

Result and Discussion

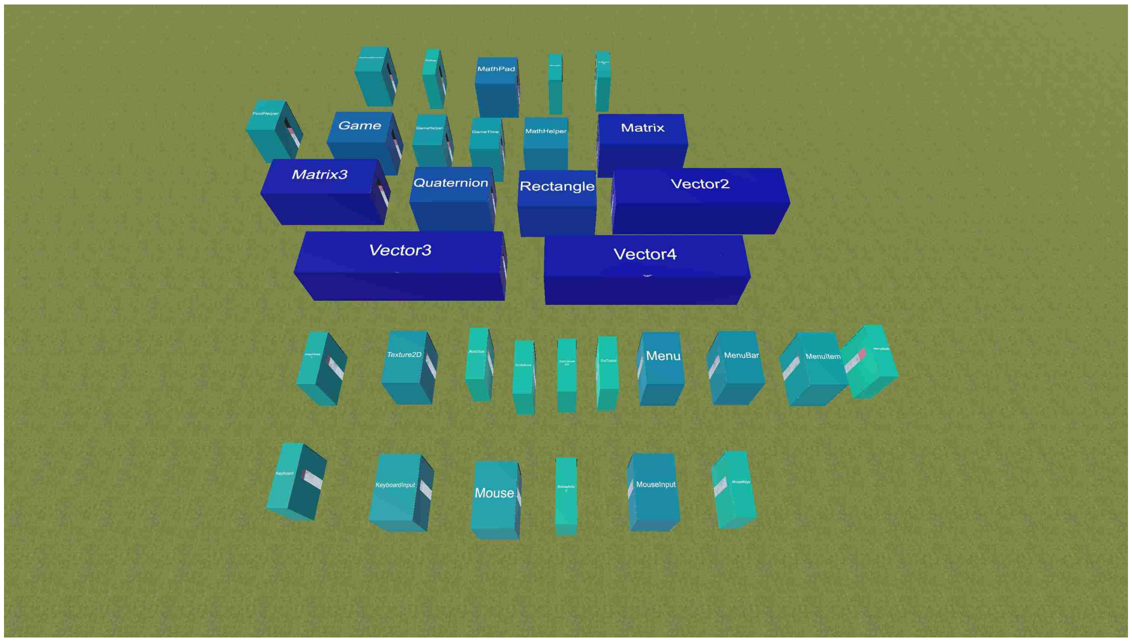
The second part of our study was to ask the participants to organize an existing project consist of 33 classes in Code Park environment. We asked them to arrange this classes that make them feel confident about the location of each class and make it easier for them to remember and access to each file. We took a screenshot from each participants arrangement and they are shown in Figure 8.3.



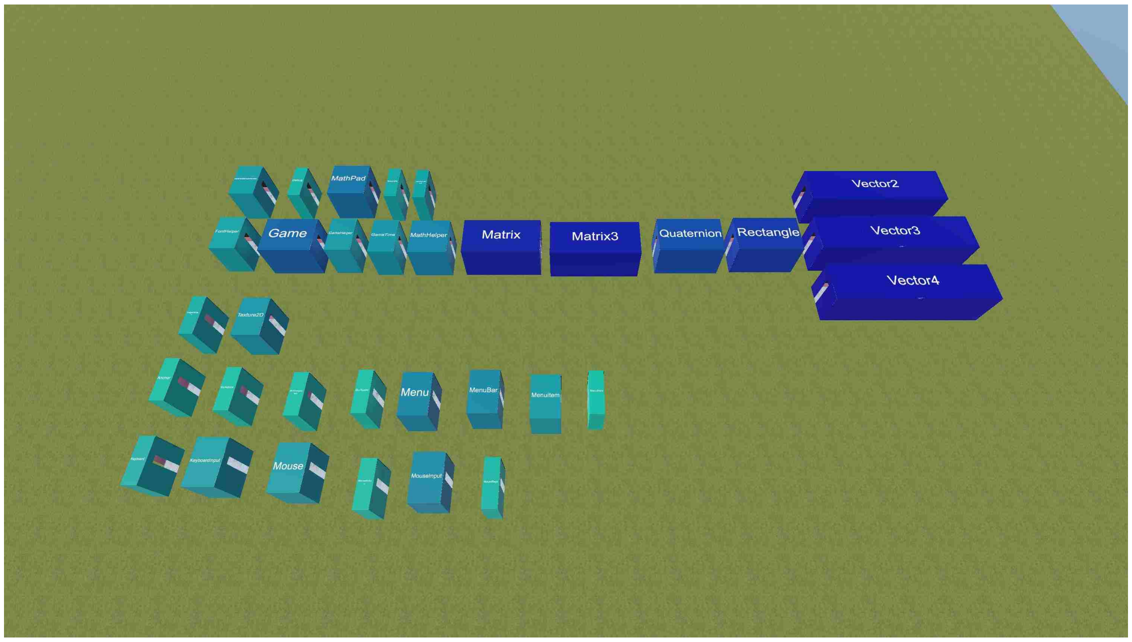
(a) Participant 1



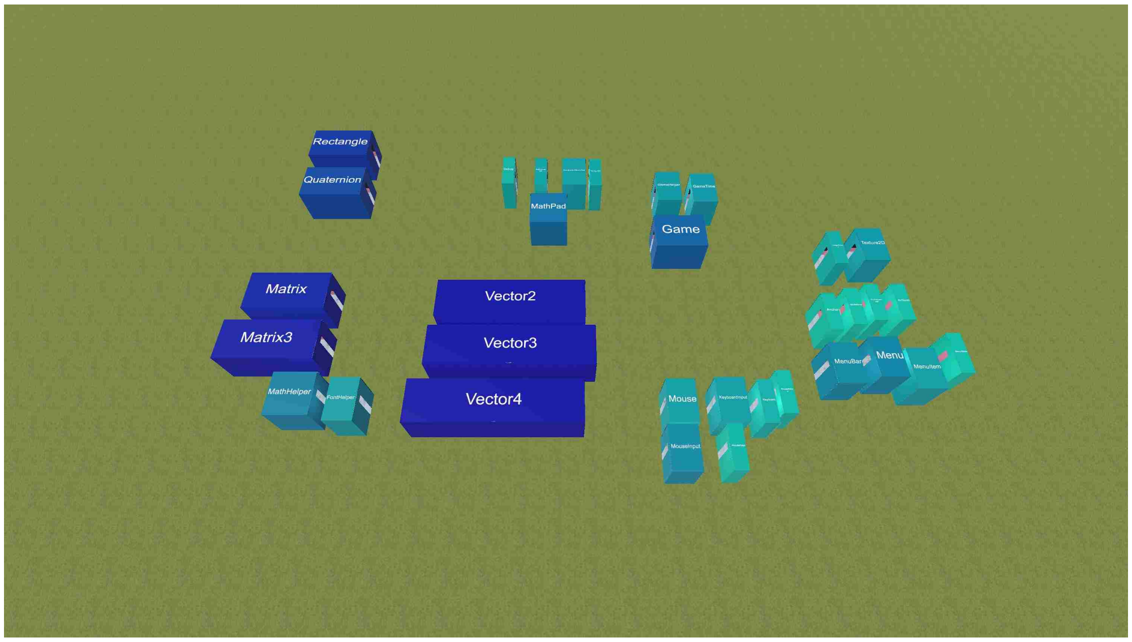
(b) Participant 2



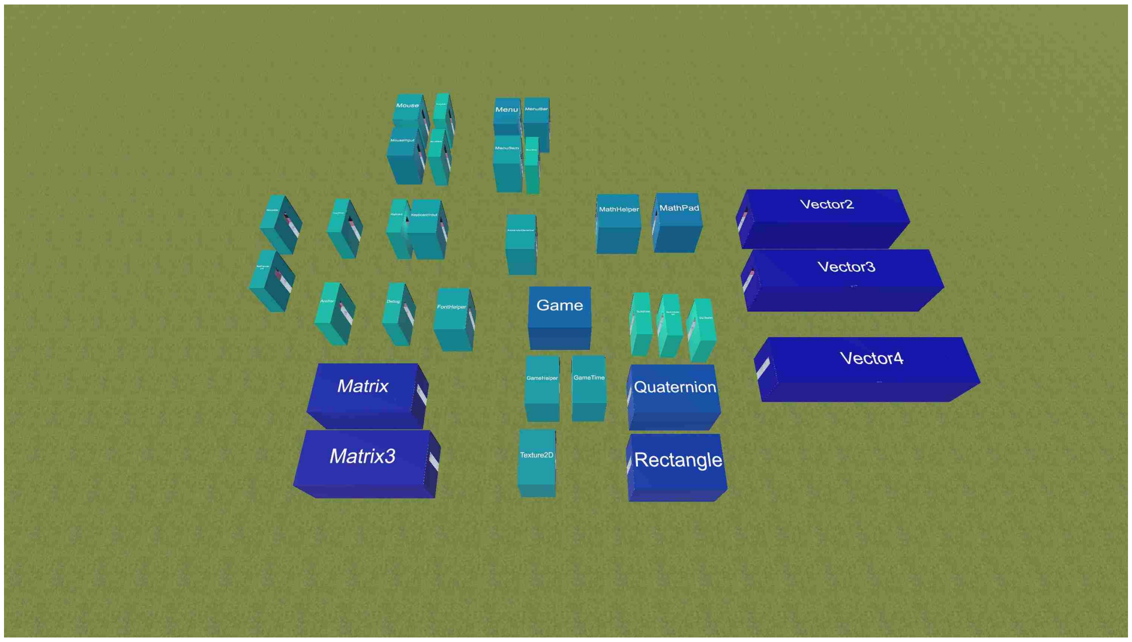
(c) Participant 3



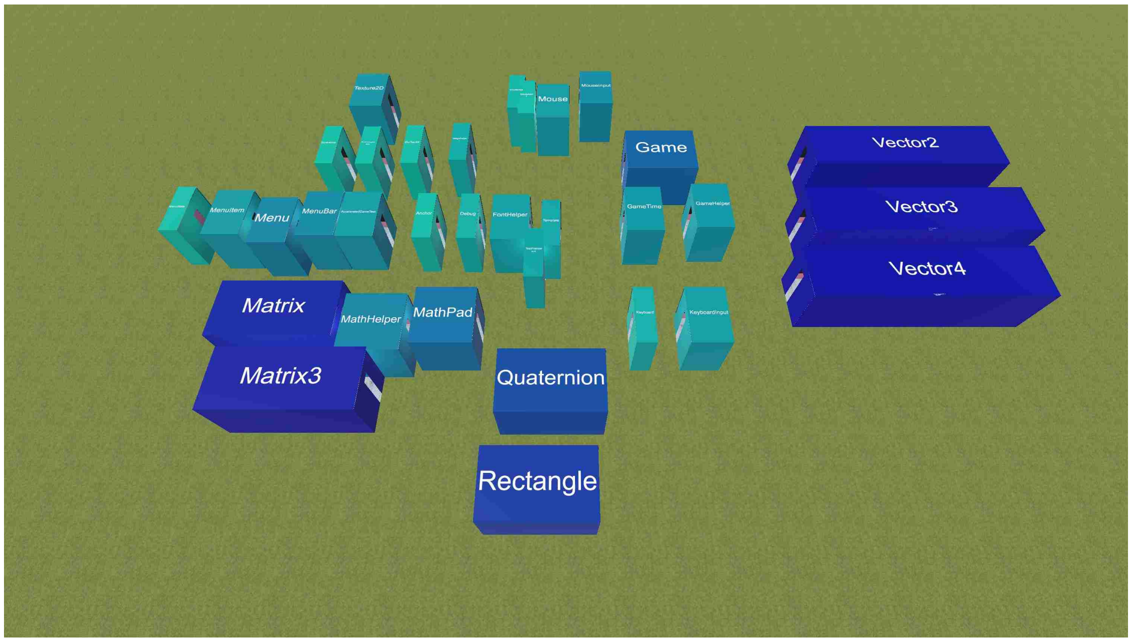
(d) Participant 4



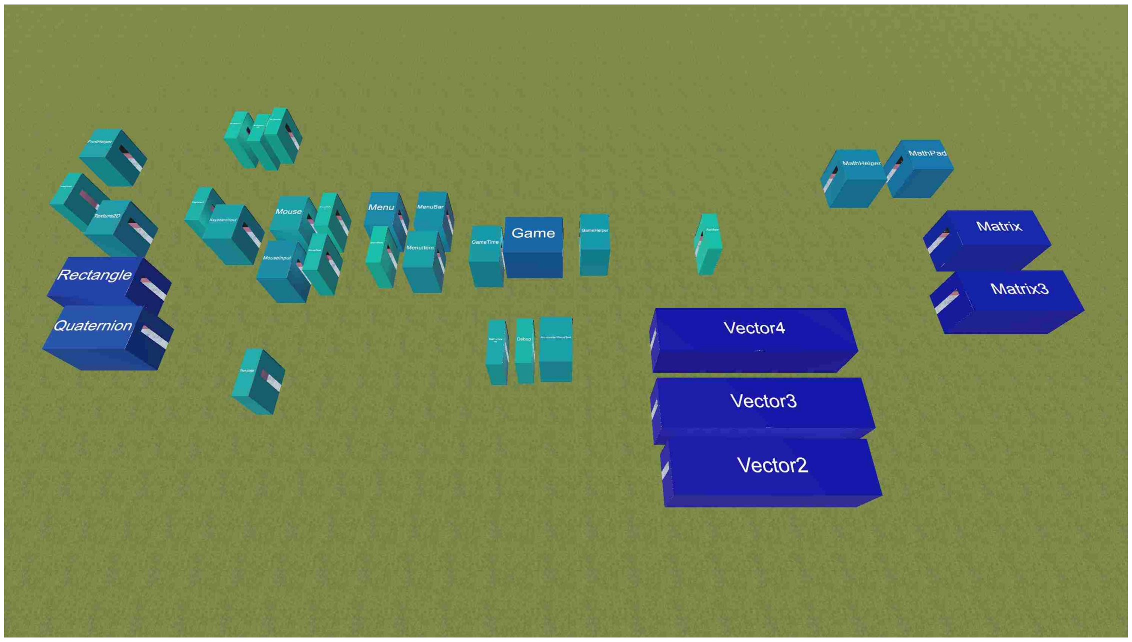
(e) Participant 5



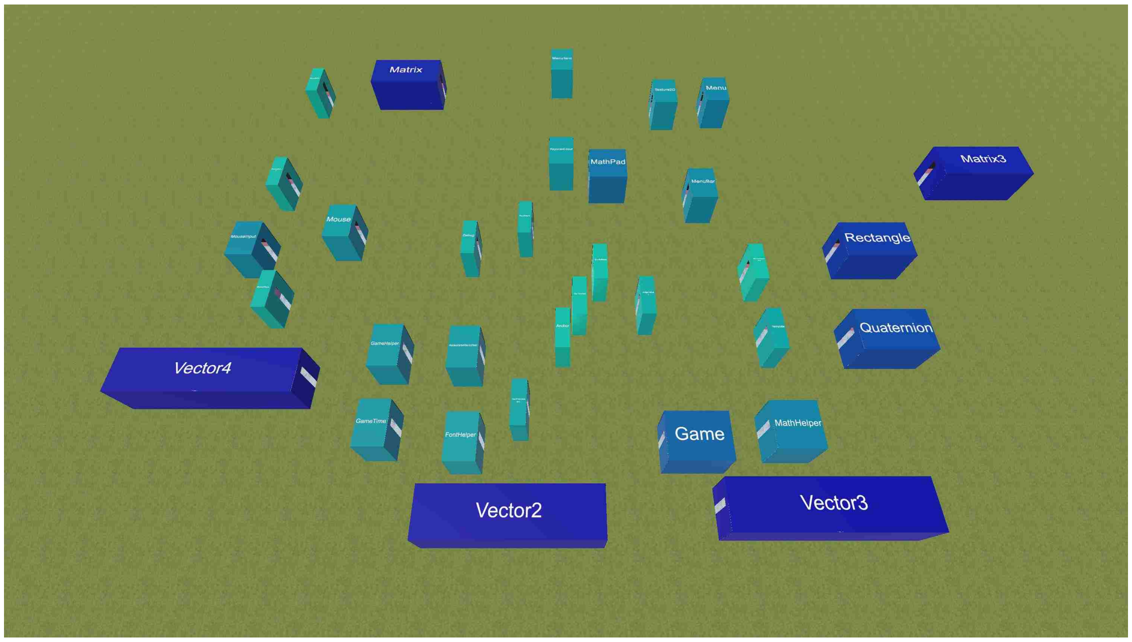
(f) Participant 6



(g) Participant 7



(h) Participant 8



(i) Participant 9

Figure 8.3: Screenshots of participants project arrangement.

As evident in Figure 8.3a to 8.3e, the organization performed by the participants mostly followed the directory-based organization of the project that was given to them. Some participants chose to place the contents of each directory in a separate line while others chose to spatially group them into a group of adjacent blocks. We asked the participants about their reasoning for such arrangements and obtained the following responses:

Participant 1: *"Folders were arranged spatially in groups. Classes that appeared related by name were sub-grouped."*

Participant 2: *"[I kept] directories grouped together."*

Participant 3: *"I just arranged classes of a particular folder in each row."*

Participant 4: *"The classes were arranged alphabetically for each folder and I arranged the classes in the same folder in the same line."*

Participant 5: *"I tried to group the related class together based on the usefulness and field."*

Figure 8.3f to 8.3i depict the results obtained from the second group of participants. When asked about their reasoning for their decisions, the following responses were obtained:

Participant 6: *"When arranging the classes my first concern was to group similar classes together. After considering which groups existed, I tried to come up with a hierarchy based on the classes size. So I put bigger classes on the side and all the smaller ones in the middle."*

Participant 7: *"I tried to place the rooms in the chunk of similar classes. My priority was to place them in such a way that they are easy to find again."*

Participant 8: *"I grouped the rooms based on their classes' name."*

Participant 9: *"Big models together. Smaller ones in the middle so I can find them easier."*

The results show a possible relation between the user's cognitive understanding of the codebase and their decisions in organizing building block of the project when working with CP. As shown in Figure 8.3, .

In cases where the project had an inherent organization by means of directories, the users mostly followed that same organization when working with CP. However, if the project lacked an inherent organization, the users' decisions were guided either by the size of each class or the semantic relationship of those classes. The users mostly elected to organize similar parts of the project in the close proximity of each other. This is inline with the results observed by Abbes *et al.* [40]. Abbes *et al.* reported that the increase in spatial dispersion of objects results in more difficulty in processing and attentional allocation.

CHAPTER 9: LIMITATION AND FUTURE WORK

Limitation

There are a few notable limitations associated with the design of our usability study and CP in general. First, we realize that our comparison with VS in second user study could potentially bias the results. This is due to the fact that most C# developers have experience with VS and their prior familiarity with it could affect their responses. Second, comparing VS which is fast and responsive to an interface that has animations and is slower may not result in a completely fair assessment. Some participants also pointed this out and felt CP was generally slower than VS and that first-person *"walking speed was slow"* and the interface needed *"more polishing"*.

Future Work

First, the whole environment needs to be improved; we believe with more refinements, it can improve the user's experience and make the interface more fluid and natural. We also plan to incorporate more functionality into Code Park such as auto-complete text or supporting inheritance (showing related classes with an arrow or different color), mainly because these two were the most requested features by our user studies' participants.

Implementing Code Park as a plugin for Microsoft Visual Studio is also worth exploring. In this case we can combine the power of Visual Studio features such as text editor with Code Park 3D code visualization features.

Furthermore we can change the layout of Code Park and put the classes in 3D space with 3DOF (Degree Of Freedom). In that case we move from 2.5D in current version (3D classes only have

2DOF) to completely 3D environment. Then we can compare these two versions and study the affect of adding new dimension in organizing classes on Code Park usability.

In the third user study it may be worth considering giving participants a more complicated project to work on. It will reveal what the limitation of our design might be and where we should improve Code Park.

Additionally, performing a more thorough study involving more participants is worth considering. As such, a possible means of studying would be to observe how Code Park would affect the learning of a group of novice programmers in (say) a semester-long course similar to the work of Saito *et al.* [36] or Buchanan and Laviola [41]. It would be interesting to perform a similar study on Code Park and evaluate its effectiveness on programming.

Other possible avenues for future work are using and evaluating Code Park in different environments, such as virtual reality (VR) and augmented reality (AR). The first-person view mode that Code Park currently employs makes it suitable for adaptation to VR. Further, in addition to viewing and studying code structure, the programmers would like to be able to write code. As such, it is sensible to design an AR system that employs Code Park to aid in code understanding and development.

CHAPTER 10: CONCLUSIONS

We presented Code Park, a 3D code visualization tool designed for improving code understanding. Code Park lays out an existing code base in a 3D environment and allows the user to explore, edit and compile the code in two modalities, a god-view mode and a first-person view mode.

Through 3 different user studies, we evaluated our design and the usability of our software. We improved our design in each iteration based on the result of the previous user study. The analysis of our results demonstrated the benefits of Code Park as a viable tool for understanding an existing code base. The results also show that Code Park is reliable for working on a project from scratch to the end. Our participants found Code Park to be easy to learn, instrumental in understanding as well as remembering the structure of a code base, and enjoyable to use.

Finally, users feedback to the application has shown that there is still a lot to do, that is very valuable to be explored. Adding new features and also new long user study with more participants would be essential.

APPENDIX A: IRB APPROVAL LETTERS

University of Central Florida Institutional Review Board
Office of Research & Commercialization
12201 Research Parkway, Suite 501
Orlando, Florida 32826-3246
Telephone: 407-823-2901 or 407-882-2276
www.research.ucf.edu/compliance/irb.html

Approval of Human Research

From: **UCF Institutional Review Board #1**
FWA00000351, IRB00001138

To: **Pooya Khaloo**
Date: **September 29, 2016**

Dear Researcher:

On 09/29/2016 the IRB approved the following human participant research until 09/28/2017 inclusive:

Type of Review:	Submission Response for UCF Initial Review Submission Form Expedited Review
Project Title:	Comparing CodeHouse (New metaphor in IDEs) to Visual Studio
Investigator:	Pooya Khaloo
IRB Number:	SBE-16-12513
Funding Agency:	
Grant Title:	
Research ID:	N/A

The scientific merit of the research was considered during the IRB review. The Continuing Review Application must be submitted 30days prior to the expiration date for studies that were previously expedited, and 60 days prior to the expiration date for research that was previously reviewed at a convened meeting. Do not make changes to the study (i.e., protocol, methodology, consent form, personnel, site, etc.) before obtaining IRB approval. A Modification Form **cannot** be used to extend the approval period of a study. All forms may be completed and submitted online at https://iris.research.ucf.edu .

If continuing review approval is not granted before the expiration date of 09/28/2017, approval of this research expires on that date. When you have completed your research, please submit a Study Closure request in iRIS so that IRB records will be accurate.

Use of the approved, stamped consent document(s) is required. The new form supersedes all previous versions, which are now invalid for further use. Only approved investigators (or other approved key study personnel) may solicit consent for research participation. Participants or their representatives must receive a copy of the consent form(s).

All data, including signed consent forms if applicable, must be retained and secured per protocol for a minimum of five years (six if HIPAA applies) past the completion of this research. Any links to the identification of participants should be maintained and secured per protocol. Additional requirements may be imposed by your funding agency, your department, or other entities. Access to data is limited to authorized individuals listed as key study personnel.

In the conduct of this research, you are responsible to follow the requirements of the Investigator Manual.

On behalf of Sophia Dziegielewski, Ph.D., L.C.S.W., UCF IRB Chair, this letter is signed by:

[illegible]

Signature applied by Patria Davis on 09/29/2016 03:53:03 PM EDT

IRB Coordinator

University of Central Florida Institutional Review Board
Office of Research & Commercialization
12201 Research Parkway, Suite 501
Orlando, Florida 32826-3246
Telephone: 407-823-2901 or 407-882-2276
www.research.ucf.edu/compliance/irb.html

Approval of Human Research

From: **UCF Institutional Review Board #1**
FWA00000351, IRB00001138

To: **Pooya Khaloo**

Date: **April 21, 2017**

Dear Researcher:

On 04/21/2017 the IRB approved the following human participant research until 04/20/2018 inclusive:

Type of Review:	UCF Initial Review Submission Form Expedited Review
Project Title:	Evaluating Code Park a new 3D IDE
Investigator:	Pooya Khaloo
IRB Number:	SBE-17-13013
Funding Agency:	
Grant Title:	
Research ID:	N/A

The scientific merit of the research was considered during the IRB review. The Continuing Review Application must be submitted 30days prior to the expiration date for studies that were previously expedited, and 60 days prior to the expiration date for research that was previously reviewed at a convened meeting. Do not make changes to the study (i.e., protocol, methodology, consent form, personnel, site, etc.) before obtaining IRB approval. A Modification Form **cannot** be used to extend the approval period of a study. All forms may be completed and submitted online at https://iris.research.ucf.edu .

If continuing review approval is not granted before the expiration date of 04/20/2018, approval of this research expires on that date. When you have completed your research, please submit a Study Closure request in iRIS so that IRB records will be accurate.

Use of the approved, stamped consent document(s) is required. The new form supersedes all previous versions, which are now invalid for further use. Only approved investigators (or other approved key study personnel) may solicit consent for research participation. Participants or their representatives must receive a copy of the consent form(s).

All data, including signed consent forms if applicable, must be retained and secured per protocol for a minimum of five years (six if HIPAA applies) past the completion of this research. Any links to the identification of participants should be maintained and secured per protocol. Additional requirements may be imposed by your funding agency, your department, or other entities. Access to data is limited to authorized individuals listed as key study personnel.

In the conduct of this research, you are responsible to follow the requirements of the Investigator Manual.

On behalf of Sophia Dziegielewski, Ph.D., L.C.S.W., UCF IRB Chair, this letter is signed by:

[illegible]

Signature applied by Gillian Amy Mary Morien on 04/21/2017 12:37:11 PM EDT

IRB Coordinator

APPENDIX B: QUESTIONNAIRES AND TASKS USED IN USER STUDIES

Code Park V1 Study: Pre Questionnaire

Please complete this survey.

Age: _______________

Major: ___________________________________

Gender:

Male Female

1- Academic Standing:

Freshman Sophomore Junior Senior Graduate N/A

2- How many years you have experience as a developer?

Less Than a year 1 – 2 years 2 – 5 years More than 5 years N/A

3- Rate yourself as a developer

1	2	3	4	5	6	7
Novice			Intermediate			Expert

4- Rate your experience with C# language

1	2	3	4	5	6	7
Novice			Intermediate			Expert

5- Rate your experience with Visual Studio IDE

1	2	3	4	5	6	7
Novice			Intermediate			Expert

6- Rate your interest in computer programming

1	2	3	4	5	6	7
Not Interested			Normal			Very Interested

7- I think computer programming is too difficult for me to learn

1	2	3	4	5	6	7
Very Difficult			Fair			Very Easy

8- Generally, it is hard for me to remember a program's structure

1	2	3	4	5	6	7
Very Hard			Fair			Very Easy

9- What was the largest code you ever developed based on lines of code and the number of classes? (Describe it)

10- Have you ever collaborated on a project after the project had started? Did you need to understand the code first and then continue working on it? (Describe it)

Question 1:
Which of the following statements is CORRECT about the C#.NET code snippet given below?

```
 namespace IMConsoleApplication
{
   class Sample
   {
      public int index;
      public int[] arr = new int[10];

      public void fun(int i, int val)
      {
         arr[i] = val;
      }
   }

   class MyProgram
   {
      static void Main(string[] args)
      {
         Sample s = new Sample();
         s.index = 20;
         Sample.fun(1, 5);
         s.fun(1, 5);
      }
   }
}
```

Options:

- s.index = 20 will report an error since index is public.
- The call s.fun(1, 5) will work correctly.
- Sample.fun(1, 5) will set a value 5 in arr[1].
- The call Sample.fun(1, 5) cannot work since fun() is not a static function.
- arr being a data member, we cannot declare it as public.

Question 2:
Blueberries cost more than strawberries.
Blueberries cost less than raspberries.
Raspberries cost more than both strawberries and blueberries.

If the first two statements are true, the third statement is:
Options:

- true
- false
- uncertain

Question 3:
What will happen if the value of nNum is 2 and the following code is compiled and executed?

```
switch(nNum)
{
        case 1:
                Console.WriteLine("nNum = 1");
                break;
        case 2:
                Console.WriteLine("nNum = 2");
        default:
                Console.WriteLine("nNum is not 1 nor 2");
        break;
}
```

Options:

- The code will output: nNum = 2
- The code will output: nNum is not 1 nor 2
- The code will output: nNum = 1
- The code will output: nNum = 2 \n nNum is not 1 nor 2

Question 4:
Consider the following code:

```
    string s1 = "Old Value";
    string s2 = s1;
    s1 = "New Value";
    Console.WriteLine(s2);
```

What will be the output printed?

Options:

- "New Value" because string is reference type
- "Old Value" because string is value type
- "New Value" because string is value type
- "Old Value" because string is reference type
- "Old Value" because string is reference type which is treated as a special case by assignment operator

Question 5:

If a class called Point is present in namespace n1 as well as in namespace n2, then which of the following is the CORRECT way to use the Point class?

Options:

-
  ```
  using n1;
  using n2;
  namespace IMConsoleApplication
  {
          class MyProgram
          {
                  static void Main(string[] args)
                  {
                          n1.Point x = new n1.Point();
                          x.fun();
                          n2.Point y = new n2.Point();
                          y.fun();
                  }
          }
  }
  ```

-
  ```
  namespace IMConsoleApplication
  {
          class MyProgram
          {
                  static void Main(string[] args)
                  {
                          import n1;
                          Point x = new Point();
                          x.fun();
                          import n2;
                          Point y = new Point();
                          y.fun();
                  }
          }
  }
  ```

-
  ```
  import n1;
  import n2;
  namespace IMConsoleApplication
  {
          class MyProgram
          {
                  static void Main(string[] args)
                  {
                          n1.Point x = new n1.Point();
                          x.fun();
                          n2.Point y = new n2.Point();
                          y.fun();
                  }
          }
  }
  ```

-
  ```
  namespace IMConsoleApplication
  {
          class MyProgram
          {
                  static void Main(string[] args)
                  {
                          using n1;
                          Point x = new Point();
                          x.fun();
                          using n2;
                          Point y = new Point();
                          y.fun();
                  }
          }
  }
  ```

Code Park V1 Study: Post Task Questionnaire

Visual Studio

Please answer all questions below. Read the directions before answering any question.

The following questions use a ranking scale from 1 – 7.

I found it easy to work with Visual Studio

1 Hard to use	2	3	4 Fair	5	6	7 Easy to use

I found it easy to become familiar with Visual Studio.

1 Hard to learn	2	3	4 Fair	5	6	7 Easy to learn

Visual Studio helps me become familiar with code base's structure.

1 Not Helping	2	3	4 Fair	5	6	7 Helping a lot

It was easy to navigate through the code with Visual Studio.

1 Hard	2	3	4 Fair	5	6	7 Easy

It was easy to find the definition of some variable/method with Visual Studio.

1 Hard	2	3	4 Fair	5	6	7 Easy

How much did you like the Visual Studio?

1 Not much	2	3	4 Fair	5	6	7 Very much

How did you feel when using the interface?

1 Frustrated	2	3	4 Fair	5	6	7 Enjoyed

Post Experiment Questionnaire (VS) Participant: _______

It was easy to find what I wanted in the code using Visual Studio.

1	2	3	4	5	6	7
Hard			Fair			Easy

List the most positive aspects of the user interface.

List the most negative aspects of the user interface.

Additional comment

Code Park V1 Study: Post Task Questionnaire

Code Park

Please answer all questions below. Read the directions before answering any question.

The following questions use a ranking scale from 1 – 7.

I found it easy to work with Code Park

1	2	3	4	5	6	7
Hard to use			Fair			Easy to use

I found it easy to become familiar with Code Park.

1	2	3	4	5	6	7
Hard to learn			Fair			Easy to learn

Code Park helps me become familiar with code base's structure.

1	2	3	4	5	6	7
Not Helping			Fair			Helping a lot

It was easy to navigate through the code with Code Park.

1	2	3	4	5	6	7
Hard			Fair			Easy

It was easy to find definition of some variable/method with Code Park.

1	2	3	4	5	6	7
Hard			Fair			Easy

How much did you like the Code Park?

1	2	3	4	5	6	7
Not much			Fair			Very much

How did you feel when using the interface?

1	2	3	4	5	6	7
Frustrated			Fair			Enjoyed

Post Experiment Questionnaire (CP) Participant: _______

It was easy to find what I wanted in the code using Code Park.

1	2	3	4	5	6	7
Hard			Fair			Easy

List the most positive aspects of the user interface.

List the most negative aspects of the user interface.

Is there anything you would like to see the Code Park do that is not currently supported?

Additional comment

Task List for Memory Game

1. Examine the game and find the user to login

2. Find an abstract class in the code base.

3. What is the relationship between the "CardBack" class and the "CardFace" class?

4. Try to load a game without looking at the stack trace, pinpoint the place the issue is occurring.

5. Imagine you are tasked with adding a feature to the game so that every time the user fails to correctly match a card, one point is deducted from the player’s total score. Can you pinpoint the place to add the necessary logic to the code to support this?

6. Can you remember position of _______ Class/method?

Task List for Library Manager

1. By examining the code base, try to find valid username and password to log into the system!

2. Create a username for yourself

3. Find an abstract class in the code base.

4. What is the relationship between the "Admin" class and the "User" class?

5. Log in with the "admin" user and try deleting your username. Without looking at the stack trace, pinpoint the place the issue is occurring.

6. Imagine you are tasked with adding some extra functionalities to the users of type “Staff”. For instance, we are interested in allowing staff members to edit book information. Can you pinpoint the place to add the necessary logic to the code to support this?

7. Can you remember position of _______ Class/method?

Code Park V1 Study: Post Study Questionnaire

Please answer all questions below. Read the directions before answering any question.

Select one of the interfaces for each question.

Which interface is more comfortable to use?

Code Park	Visual Studio

Which interface is more likeable?

Code Park	Visual Studio

Which interface is more natural?

Code Park	Visual Studio

Which interface is easier to use?

Code Park	Visual Studio

Which interface is more fun to use?

Code Park	Visual Studio

Which interface is more frustrating?

Code Park	Visual Studio

Which interface helps you more in remembering code base structure?

Code Park	Visual Studio

For learning a code base I would prefer to use _____

Code Park	Visual Studio

Once I am already familiar with a code base, I would preferred to use _____ for additional work with the code base.

Code Park	Visual Studio

Post Questionnaire Overall Participant: _______

For learning the structure of code base I think I preferred to use _____

Code Park	Visual Studio

For finding a particular class/variable I would preferred to use _____

Code Park	Visual Studio

For tracking down a bug I would preferred to use _____

Code Park	Visual Studio

Overall which interface is better?

Code Park	Visual Studio

Additional comment

Code Park V2 Study: Pre Questionnaire

Please complete this survey.

Age: _______________

Major: __________________________________

Gender:

Male Female

Academic Standing:

Freshman Sophomore Junior Senior Graduate N/A

How many years you have experience as a developer?

Less Than a year 1 – 2 years 2 – 5 years More than 5 years N/A

Rate yourself as a developer

1	2	3	4	5	6	7
Novice			Intermediate			Expert

Rate your experience with Java language

1	2	3	4	5	6	7
Novice			Intermediate			Expert

Rate your experience with any Java IDEs (Netbeans, Eclipse, …)

1	2	3	4	5	6	7
Novice			Intermediate			Expert

Rate your interest in computer programming

1	2	3	4	5	6	7
Not Interested			Normal			Very Interested

Generally, it is hard for me to remember a program's structure

1	2	3	4	5	6	7
Very Hard			Fair			Very Easy

What was the largest code you ever developed based on lines of code and the number of classes? (Describe it)

Have you ever collaborated on a project after the project had started? Did you need to understand the code first and then continue working on it? (Describe it)

Code Park Assignment

The libraries of SmallTownX need a new electronic rental system, and it is up to you to build it. SmallTownX has two libraries. Each library offers many books to rent. Customers can print the list of available books, borrow, and return books.

Problem:

You need to develop two classes called Library and Book.

Book:

- The Book library needs to have a constructor that get the name of the book and save it in the variable within the class.
- A Boolean value that shows if the book is borrowed or not.
- Two methods named as borrowed and returned with appropriate functionality.
- A method to check if a book is borrowed or not.
- A method to get the title of the book.

For test you need to have a functionality in your main method to create this output:

```
Title (should be The Da Vinci Code): The Da Vinci Code

Rented? (should be false): false

Rented? (should be true): true

Rented? (should be false): false
```

Library:

The Library needs to have a constructor to get the address of the library.

- An `addBook` method that get a Book object as an argument and add it to the library.
- A `parintAddress` method to print address of the library.
- A `borrowBook` method that borrow a book with its name.
- A `returnBook` method that return a book with its name.
- A `printAvailableBook` method that print all the available books in the library.

For test first you need to create two libraries and add four books to them.

```
Library firstLibrary = new Library("10 Main St.");

Library secondLibrary = new Library("228 Liberty St.");

// Add four books to the first library

firstLibrary.addBook(new Book("The Da Vinci Code"));

firstLibrary.addBook(new Book("Le Petit Prince"));

firstLibrary.addBook(new Book("A Tale of Two Cities"));

firstLibrary.addBook(new Book("The Lord of the Rings"));
```

Then have a functionality in your main method to create this output:

```
Library addresses:
10 Main St.
228 Liberty St.

Borrowing The Lord of the Rings:
You successfully borrowed The Lord of the Rings
Sorry, this book is already borrowed.
Sorry, this book is not in our catalog.

Books available in the first library:
The Da Vinci Code
Le Petit Prince
A Tale of Two Cities

Books available in the second library:
No book in catalog

Returning The Lord of the Rings:
You successfully returned The Lord of the Rings

Books available in the first library:
The Da Vinci Code
Le Petit Prince
A Tale of Two Cities
The Lord of the Rings
```

Code Park V2 Study: Post Task Questionnaire

Why do you arranged the rooms like this? What was your priority? What was you criterion?

Post Experiment Questionnaire Participant: _______

Code Park V2 Study: Post Study Questionnaire

Please answer all questions below. Read the directions before answering any question.

The following questions use a ranking scale from 1 – 7.

I found it easy how to use CodePark.

1	2	3	4	5	6	7
Hard to use			Fair			Easy to use

I found it easy how to learn using CodePark.

1	2	3	4	5	6	7
Hard to learn			Fair			Easy to learn

I think that I would like to use CodePark frequently.

1	2	3	4	5	6	7
Strongly disagree			Fair			Strongly Agree

I found the various functions in CodePark were well integrated.

1	2	3	4	5	6	7
Strongly disagree			Fair			Strongly Agree

It was easy to navigate through the code with CodePark.

1	2	3	4	5	6	7
Hard			Fair			Easy

It was easy to write code in CodePark.

1	2	3	4	5	6	7
Hard			Fair			Easy

It was easy to work on a project with CodePark.

1	2	3	4	5	6	7
Hard			Fair			Easy

How much did you like the CodePark?

1	2	3	4	5	6	7
Not much			Fair			Very much

Post Experiment Questionnaire Participant: _______

How did you feel when using the interface?

1	2	3	4	5	6	7
Frustrated			Fair			Enjoyed

It was easy to find what I wanted in the code using CodePark.

1	2	3	4	5	6	7
Hard			Fair			Easy

How long did it take to finish the assignment? (Approximately in hour)

List the most positive aspects of the user interface.

List the most negative aspects of the user interface.

Is there anything you would like to see the CodePark do that is not currently supported?

Additional comment

LIST OF REFERENCES

[1] A. Bragdon, S. P. Reiss, R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr., “Code bubbles: Rethinking the user interface paradigm of integrated development environments,” in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE ’10, (New York, NY, USA), pp. 455–464, ACM, 2010.

[2] H. Graham, H. Y. Yang, and R. Berrigan, “A solar system metaphor for 3d visualisation of object oriented software metrics,” in *Proceedings of the 2004 Australasian Symposium on Information Visualisation - Volume 35*, APVis ’04, (Darlinghurst, Australia, Australia), pp. 53–59, Australian Computer Society, Inc., 2004.

[3] R. Wettel and M. Lanza, “Visually localizing design problems with disharmony maps,” in *Proceedings of the 4th ACM Symposium on Software Visualization*, SoftVis ’08, (New York, NY, USA), pp. 155–164, ACM, 2008.

[4] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai, “Scratch: Programming for all,” *Commun. ACM*, vol. 52, pp. 60–67, Nov. 2009.

[5] E. Perrin, S. Linck, and F. Danesi, “Algopath: A new way of learning algorithmic,” in *in The Fifth International Conference on Advances in Computer-Human Interactions*, 2012.

[6] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, “A study of the difficulties of novice programmers,” in *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, ITiCSE ’05, (New York, NY, USA), pp. 14–18, ACM, 2005.

[7] S. C. Eick, J. L. Steffen, and E. E. Sumner, “Seesoft-a tool for visualizing line oriented software statistics,” *IEEE Transactions on Software Engineering*, vol. 18, pp. 957–968, Nov 1992.

[8] A. D. Baddeley, *Human memory: Theory and practice*. Psychology Press, 1997.

[9] J. Sweller, “Cognitive load theory, learning difficulty, and instructional design,” *Learning and Instruction*, vol. 4, no. 4, pp. 295 – 312, 1994.

[10] R. M. Carini, G. D. Kuh, and S. P. Klein, “Student engagement and student learning: Testing the linkages*,” *Research in Higher Education*, vol. 47, no. 1, pp. 1–32, 2006.

[11] N. Burgess, E. A. Maguire, and J. O'Keefe, “The human hippocampus and spatial and episodic memory,” *Neuron*, vol. 35, no. 4, pp. 625 – 641, 2002.

[12] A. Marcus, L. Feng, and J. I. Maletic, “3d representations for software visualization,” in *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, (New York, NY, USA), pp. 27–ff, ACM, 2003.

[13] A. Goldberg and D. Robson, *Smalltalk-80: The Language and Its Implementation*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1983.

[14] C. Kurtz, “Code gestalt: A software visualization tool for human beings,” in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, (New York, NY, USA), pp. 929–934, ACM, 2011.

[15] M. Lanza and S. Ducasse, “A categorization of classes based on the visualization of their internal structure: The class blueprint,” *SIGPLAN Not.*, vol. 36, pp. 300–311, Oct. 2001.

[16] C. Gutwenger, M. Jünger, K. Klein, J. Kupke, S. Leipert, and P. Mutzel, “A new approach for visualizing uml class diagrams,” in *Proceedings of the 2003 ACM Symposium on Software Visualization*, SoftVis '03, (New York, NY, USA), pp. 179–188, ACM, 2003.

[17] O. Radfelder and M. Gogolla, "On better understanding uml diagrams through interactive three-dimensional visualization and animation," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, (New York, NY, USA), pp. 292–295, ACM, 2000.

[18] M. Balzer, O. Deussen, and C. Lewerentz, "Voronoi treemaps for the visualization of software metrics," in *Proceedings of the 2005 ACM Symposium on Software Visualization*, SoftVis '05, (New York, NY, USA), pp. 165–172, ACM, 2005.

[19] D. Holten, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, pp. 741–748, Sept 2006.

[20] N. Hawes, S. Marshall, and C. Anslow, "Codesurveyor: Mapping large-scale software to aid in code comprehension," in *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*, pp. 96–105, Sept 2015.

[21] A. Cockburn and B. McKenzie, "Evaluating the effectiveness of spatial memory in 2d and 3d physical and virtual environments," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02, (New York, NY, USA), pp. 203–210, ACM, 2002.

[22] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich, "Data mountain: Using spatial memory for document management," in *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST '98, (New York, NY, USA), pp. 153–162, ACM, 1998.

[23] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz, "Software landscapes : Visualizing the structure of large software systems," in *Joint Eurographics - IEEE TCVG Symposium on Visualization (VisSym)*, 2004.

[24] D. Bonyuet, M. Ma, and K. Jaffrey, “3d visualization for software development,” in *Web Services, 2004. Proceedings. IEEE International Conference on*, pp. 708–715, July 2004.

[25] O. Greevy, M. Lanza, and C. Wysseier, “Visualizing feature interaction in 3-d,” in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, pp. 1–6, 2005.

[26] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc, “Communicating software architecture using a unified single-view visualization,” in *12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007)*, pp. 217–228, July 2007.

[27] S. Alam and P. Dugerdil, “Evospaces visualization tool: Exploring software architecture in 3d,” in *14th Working Conference on Reverse Engineering (WCRE 2007)*, pp. 269–270, Oct 2007.

[28] C. Knight and M. Munro, “Virtual but visible software,” in *Information Visualization, 2000. Proceedings. IEEE International Conference on*, pp. 198–205, 2000.

[29] G. Langelier, H. Sahraoui, and P. Poulin, “Visualization-based analysis of quality for large-scale software systems,” in *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE ’05, (New York, NY, USA), pp. 214–223, ACM, 2005.

[30] K. Kahn, “Toontalk tman animated programming environment for children,” *Journal of Visual Languages & Computing*, vol. 7, no. 2, pp. 197–217, 1996.

[31] S. Cooper, W. Dann, and R. Pausch, “Alice: A 3-d tool for introductory programming concepts,” in *Proceedings of the Fifth Annual CCSC Northeastern Conference on The Journal of Computing in Small Colleges*, CCSC ’00, (USA), pp. 107–116, Consortium for Computing Sciences in Colleges, 2000.

[32] D. Parsons and P. Haden, “Parson’s programming puzzles: A fun and effective learning tool for first programming courses,” in *Proceedings of the 8th Australasian Conference on Computing*

Education - Volume 52, ACE '06, (Darlinghurst, Australia, Australia), pp. 157–163, Australian Computer Society, Inc., 2006.

[33] C. Zorn, C. A. Wingrave, E. Charbonneau, and J. J. LaViola Jr, "Exploring minecraft as a conduit for increasing interest in programming," in *Proceedings of the International Conference on the Foundations of Digital Games 2013 (FDG 2013)*, pp. 352–359, 2013.

[34] A. R. Teyseyre and M. R. Campo, "An overview of 3d software visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 87–105, Jan 2009.

[35] P. Caserta and O. Zendra, "Visualization of the static aspects of software: A survey," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, pp. 913–933, July 2011.

[36] D. Saito, H. Washizaki, and Y. Fukazawa, "Influence of the programming environment on programming education," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '16, (New York, NY, USA), pp. 354–354, ACM, 2016.

[37] D. A. Bowman, E. T. Davis, L. F. Hodges, and A. N. Badre, "Maintaining spatial orientation during travel in an immersive virtual environment," *Presence: Teleoper. Virtual Environ.*, vol. 8, pp. 618–631, Dec. 1999.

[38] J. O. Wobbrock, L. Findlater, D. Gergle, and J. J. Higgins, "The aligned rank transform for nonparametric factorial analyses using only anova procedures," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, (New York, NY, USA), pp. 143–146, ACM, 2011.

[39] J. O. Wobbrock, "Practical statistics for human-computer interaction: An independent study combining statistics theory and tool know-how," in *Annual workshop of the Human-Computer Interaction Consortium (HCIC'11)*, 2011.

[40] A. B. Abbes, E. Gavault, and T. Ripoll, “The effect of spatial organization of targets and distractors on the capacity to selectively memorize objects in visual short-term memory,” *Advances in cognitive psychology*, vol. 10, no. 3, p. 90, 2014.

[41] S. Buchanan and J. J. Laviola, Jr., “Cstutor: A sketch-based tool for visualizing data structures,” *Trans. Comput. Educ.*, vol. 14, pp. 3:1–3:28, Mar. 2014.