CODENECT: VISUAL PROGRAMMING SOFTWARE FOR LEARNING FUNDAMENTALS OF PROGRAMMING

An Undergraduate Thesis
Submitted to the Faculty of the
College of Engineering and Information Technology
Cavite State University
Indang, Cavite

In partial fulfilment
of the requirements for the degree
Bachelor of Science in Information Technology

Lim-it, Brandon B.,
Punay, Jaykel O.
May 2021

TABLE OF CONTENTS

BIOGRAPHICAL DATA

Brandon Blanker Lim-it, born November 02, 1997 in Dasmarinas, Cavite. He is the third child of four children to Mr. Bobby Sy Lim-it and Mrs. Rosie Blanker Lim-it. He is currently living in Brgy. Manggahan, General Trias, Cavite. He is a christian and a church worker. He spends his time programming, developing video games, and helping others learn programming through tutoring. His main sport is the game of chess.

He finished his elementary education at Manggahan Elementary School in 2010. He finished his secondary education at Governor Ferrer's Memorial National High School - Special Science Class in 2014.

He obtained his bachelor degree in Information Technology in 2021 in Cavite State University - Main, Indang Campus.

BIOGRAPHICAL DATA

Jaykel Ortega Punay was born on the 11th day of January year 1987 in San Francisco Province of Quezon. The 2nd child of Michael and Loida Punay. A car enthusiast and a sporty person. A good husband and a responsible father to his only one child. He always looks on the brighter side of every person and bad situations and he always moves forward despite of trials and struggles. He is positive minded person.

He graduated his primary education at Tadlac Elementary School, Los Banos, Laguna and secondary education at Batong Malake National High School, Los Banos, Laguna. He then transfered to Cavite and work in a company for ten years before deciding to pursue a college degree.

He obtained his Bachelor Degree in Information Technology in 2021 from Cavite State University - Indang, Cavite.

ACKNOWLEDGMENT

The researchers express their gratitude and earnest appreciation to the following people who inspired, helped, and contributed to the development and completion of this study:

First and foremost, to God and the Lord and Savior Jesus Christ for the boundless grace, knowledge, wisdom, and strength He bestowed continually. Our source of strength and hope every time, especially in times we feel like giving up.

Mr. James Angelo V. Aves, our thesis adviser, for the advises and supervision in the documentation and development of this study.

Ms. Anabelle J. Almarez, our technical critic, for her critics and guidance in the documentation and development of this study.

Ms. Gladys G. Perey, unit research coordinator, for her commitment and management of the time and schedule for thesis.

Mr. Marlon R. Pereña, department chairman, for his kindness in inspiring and assisting the students.

Family, relatives, and church family, for the moral, spiritual, and financial support. Our inspiration and motivation that keep us pursuing our dreams.

To the people and fellow programmers in the Löve Discord community, for their help and guidance during the development of the software.

This study is sincerely dedicated to all of them and more.

THE RESEARCHERS

ABSTRACT

LIM-IT, BRANDON B. and PUNAY, JAYKEL O. CODENECT: VISUAL PROGRAMMING SOFTWARE FOR LEARNING FUNDAMENTALS OF PROGRAMMING. Undergraduate Thesis Manuscript. Bachelor of Science in Information Technology. Cavite State University, Indang, Cavite. June 2021. Adviser: Mr. James Angelo V. Aves

The CodeNect: Visual Programming Software For Learning Fundamentals of Programming was developed for aiding beginners in the field of programming get a better grasp and understanding about its fundamentals.

The software is composed of seven modules which are the Input/Output, Visual Nodes, Transpiler, Filesystem, Simulation, Debug, and Assessment modules.

The V-Model was used as methodology for the development of the software which has the following phases: requirements, system design, architecture design, module design, implementation and coding, and testing.

The visual programming software was evaluated using the ISO 9126 standards with the criteria for quality including functionality, reliability, usability, maintanability, efficiency, and portability. The software passed the criteria for evaluation and met all the requirements and objectives having an overall mean of 4.50.

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDIX FIGURES

Appendix

Figure **Page**

LIST OF APPENDICES

Appendix **Page**

DEVELOPMENT OF CODENECT: VISUAL PROGRAMMING SOFTWARE FOR LEARNING FUNDAMENTALS OF PROGRAMMING

Lim-it, Brandon B.,
Punay, Jaykel O.

An undergraduate thesis manuscript submitted to the faculty of the Department of Information Technology, College of Engineering and Information Technology, Cavite State University, Indang, Cavite, in partial fulfillment of the requirements for the degree of Bachelor of Science in Information Technology. Pre-pared under the supervision of Prof. James Angelo V. Aves

INTRODUCTION

As the innovation in technology is continuously making its progress in improving the quality of life. With this nature of technology comes the essential need for programming skill as core profiency. The competition in the field that developers and programmers alike strive for is becoming harder to get into due to high standards and requirements. One of the requirements for a programmer and developer is to have expertise in technical skills that include multiple programming languages (Tsai, Yang, and Chang, 2015). Without the proper knowledge and understanding in programming in its fundamental level and depth, one will find it difficult to adapt to the constantly shifting world of computer.

Improving learning without prolonging the time allotted in each academic year needs to focus on enhancing the properties of the software that are both utilized as teaching and learning tools by the instructor and the student. A system that is implemented using modern tools, industry standard design, and functionality that focuses in simplicity, readability, and learning experience. Modern technology increases the rate of knowledge acquisition and absorption through its usage and implementation in education (Raja and Nagasubramani, 2018). The advancement in technology greatly contributes to education as it enables convenience in communication and presentation of knowledge and information almost instantaneously (Anggrawan, Ibrahim, M., and Satria, 2016). A software that prioritize functionality over unnecessary features to ensure that the user is not overloaded

with information in the screen that is unnecessary. Users perceive numerous features in a product to be useful and engaging but such can result in fatigue (Thompson, Hamilton, and Rust, 2005).

Software with the necessary tools and functionalities oriented towards learning purposes and is also designed to and packaged with coding exercises and problems which range from beginner, intermediate, to advance difficulties is not popular and lacking in availability. Features that are carefully selected and designed towards showing and comparing various solutions that are working in the context that they meet the requirements and output and are technically correct, but not all will meet the standard when it comes to better quality which is the advantages in skills acquired when mastering the fundamentals of programming. This approach in problem solving allows learners to develop logical and critical thinking through the application of the theory of variation, wherein some aspects that are critical must vary while other aspects stay constant (Cheng, 2016). This is effective in the domain of programming as even a slight change in data amounts to a change in effect and output.

Programming is a skill wherein it focuses in the connection of logic rather than memorizing information as that of in other domain, the curve in starting to learn it is steeper compared to actually applying it in real works and mastering it. Mastering a programming language is not an easy task, but in general all share common concepts. Learning by heart these core concepts and fundamental knowledge will help programmers to easily learn and master any existing or new programming language through the reiteration of principles that all programming languages are built and modeled upon. For the design and decision that go behind the creation of new programming languages are reevaluation of existing studies, syntax, semantics, and inspired by widely used and accepted languages (Chang Boon Lee, 2011). For anyone who is new to programming, the topics can be a daunting and intimidating task. Failure in familiarization and application in the early academic years and progressing to the next period wherein advance subjects are covered bereaves the overall learning of the student.

Visual programming enables learning through interaction and manipulation of graphical elements which can provide more feedback through the use of color, size, and icons. The abstract and high-level concepts are represented visually that can facilitate learners by variable observation, logic flow tracing, and debugging skills (Tsai et al., 2015). The use

of visual programming language has been effective in facilitating and assisting beginners in programming as visual programming lessens the time needed to learn new topics and concepts in advance subjects. There are fewer difficulties in learning as well as visualization provides higher cognitive levels of understanding for most concepts (Armoni, Meerbaum-Salant, and Ben-Ari, 2015).

Statement of the Problem

The fundamental concepts of programming are essential basics that are necessary for programmers to master. Concepts such as syntax, semantics, variables, data types, data structures, logic, conditionals, loops, algorithm, and memory are key to easily understanding and getting better at programming as it is a discipline (Prahofer, Hurnaus, Wirth, and Mossenbock, 2007). Programming is a skill which can be boring, intimidating, and unrelated to daily activities and experience. Students are lacking in understanding of the execution of a program (Tan, Ting, and Ling, 2009). Programming education requires the assistance of technology itself through software in improving the quality of learning. The traditional method of pure lecture is nowadays complimented with the application of software. But most tools are not beginner-friendly and are cluttered with features that present confusion and steep learning curve in familiarity and mastery that diminish the learning experience (Tsukamoto et al., 2016).

The assessment of the respondents under the courses with programming subjects (See Appendix Figure 1) shows that students (See Appendix Figure 13) are not familiar and not well versed on fundamental concepts and find it difficult to understand (See Appendix Figure 12 and 15).

Basic concepts such as loops, memory management, and functions are easily understood individually, but combining them into a program has confused students (See Appendix Figure 16). Respondents obtained low score in the assessment questions (See Appendix Figure 13).

Survey shows that 76% of students use text-based editors in their laboratory classes such as Notepad++, DevC++, and TurboC/C++, while 24% use professional and modern editors (See Appendix Figure 9). These software are general programming tools and are not oriented for assisting beginners in learning fundamental concepts of programming (See Appendix Figure 17).

Objectives of the Study

The general objective of the study is to develop a CodeNect: Visual Programming Software that will help in learning the fundamentals of programming.

Specifically, this study seeks:

1. Identify the concepts learners find difficult to understand through conducted survey.
2. Analyze the problems through data gathering.
3. Design the system with the following modules:
    (a) Visual Nodes Module handles the core elements and building blocks in the software for writing logic and code.
    (b) Filesystem Module handles the creation, modification, reading, and deletion of files.
    (c) Input and Output Module is the interface for user actions such as mouse events and key events and what is displayed to the screen for the user.
    (d) Debug Module lints the visual code for errors and warnings before the running the code. It also captures errors and warnings during runtime and report it to the user.
    (e) Simulation Module for compiling and running the visual program to a command line program.
    (f) Transpiler Module that will convert the visual logic to the C programming language.
    (g) Assessment Module that will evaluate the knowledge and learning of the users by providing basic and common coding exercises. Exercises can be imported, shared, and distributed through simple package files.
4. Develop the system as designed using the C++ programming language, GLFW and OpenGL renderers, TinyC Compiler as runtime compiler and runner, and DearImGui and ImNodes as user-interface libraries.
5. Test and improve the usability and functionality of the software using metrics.
6. Evaluate the acceptability of the software using the ISO 9126.

Significance of the Study

The result of the study is of great benefits for the following:

The software helps in the education and improvement in the knowledge, skills, understanding, and expertise of the students and learners about programming. Thus, allowing them to compete and increasing the opportunities for their careers.

The software provides assistance for teachers and instructors to teach and demonstrate programming concepts through visualization. This aids in relieving workload, stress, and maximizing lessons each class time.

The software benefits educational institutions like university for computer laboratory classes by providing a free software oriented for the purpose of learning.

The software provides learning experience for the developers and researchers in preparation for software development career.

This study serves as a guide and reference in the field of software development and education for future researchers.

Conceptual Framework of the Study

The conceptual framework (refer to Figure 1) represents the relationship and flow of the concept of developing a visual programming software for learning the fundamentals of programming.

It shows the order of actions required by the study to achieve the desired output following the design of the context diagram (refer to Figure 27)

The inputs has the following requirements for the development of the study. Knowledge requirements include programming, programming languages, parsing, evaluation, C++ programming language, user-interface design, user-experience design, data flow diagram, and context diagram. Software requirement include Linux 5.4 - Manjaro as operating system and distribution, Vim as text and code editor, terminal, OpenGL and GLFW library for graphics and inputs, TinyC Compiler for running the transpiled code at runtime, and the C++ programming language for development. Microsoft Windows 7 and above, C++ Runtime libraries, and the C++ programming language for deployment. Hardware requirement are machine with at at least Intel Core 2 Duo at 1.4 GHz, 2 GB of RAM, and 80 GB HDD storage for development. At least Intel Core 2 Duo at 1.4 GHz, 2 GB of RAM, and at least 1 GB HDD storage space for deployment.

The process to be followed and used for the development of the software is the V model. The process model involves the requirements of gathering necessary data and information from respondents through conducted survey. High-level design of the implementation of the requirements of its technical usage for system design. Design of the relationships and dependencies of the modules, creation of diagrams, and selection of technology to be used. Preparation of the design and test method for each module. Followed by the coding of the modules and the software. The evaluation is based on the ISO 9126 , specifically the product quality evaluation to assure the functionality, efficiency, usability, portability, maintainability, portability, and reliability of the software.

The impact expected from the study is the availability of a software designed for learning to program and its fundamentals for learners, students, and educators. Another impact is the improvements in the understanding, skills, and academic performances of the students in the course of programming.

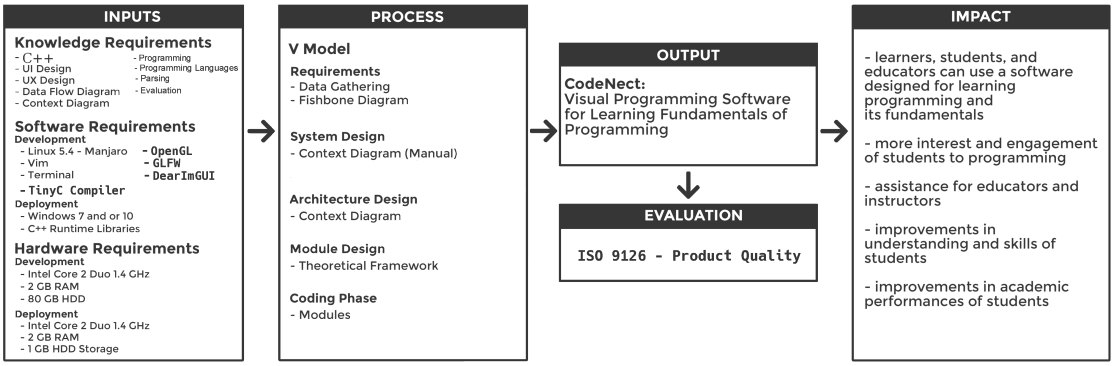


Figure 1. Conceptual Framework of proposed CodeNect: Visual Programming Software for Learning Fundamentals of Programming

Time and Place of the Study

The study started from the approval of the title on the month of February of year 2020. The development was completed on the month of May of year 2021. The necessary data were gathered through survey and research at the Cavite State University and other information were researched and obtained from the Internet.

Scope and Limitations of the Study

The study focuses on the development of a CodeNect: Visual Programming Software for learning the fundamentals of programming. The software will prioritize simple and

basic functionalities over numerous features for the purpose of learning and education. The software is designed for beginner programmers who can be students or non-students. The software can also serve as teaching and supplementary aid tool by instructors for their class. The software has no field of access as anyone with the program can use it. No requirements of Internet connection or account.

The software is to be developed in Linux operating system using the C++ programming language, Vim as code editor, terminal for building, OpenGL and GLFW framework for rendering and input. The source is to be released as open-source with appropriate license to improve contributions. Since the software is stand-alone desktop program, there will be no account management. The software is only accessible by a single user such as the student or instructor. The software fully works in offline mode.

The software is designed with seven core modules: visual nodes module, filesystem module, input and output module, debug module, simulation module, transpiler module, and assessment module.

Visual Nodes Module

Nodes are graphical elements that serve as the building blocks of the software. Nodes can be used as a variable, logic, and conditionals. The properties of the node are position, size, and type. The fields of the node that are visible to the users, which can be modified, are name, and value(s). Each node has input socket(s) and output pins which are used for the flow of logic and redirection of data. The visibility of the sockets and pins of a node is dependent on its type. For example, nodes that are constant variables will only enable the output socket as it is read-only, while regular variable nodes allow for both sockets. Nodes are connected to one another through the use of wires. This relation of nodes is called the node graph. The flow of logic is easily determined using the wires with directional arrows signifying the direction of the logic.

Filesystem Module

This module serves as the interface between the software and the user's machine for handling files. The module have four main functionalities, creation, modification, reading of files, and backup. One feature of the software that benefits from this module is the

importation of exercises from package format file which allows for more learning materials that greatly increases the possible usage of the software.

When a user starts a new project, a template project structure with base files are created by the module and is saved into the user's machine. The modules assure that the project is stored with proper permission and in safe location. Modification such as addition or deletion is also handled by the module. The reading of files and project functionality takes into consideration the validity and safeness of the file and handles if the file is corrupted. The backup functionality regularly makes a backup of file in case of emergency such as program crash or user-side accident.

Input and Output Module

This module captures user input events such as key press, mouse movement, mouse click, and so on. The module is responsible for processing and responding events and performing actions based on the event. This ensures that the interaction between the user and the software provides rich experience in terms of usability and learning.

This module handles output to the user. File, displays, views, and screens are examples of the possible types of output. The module manages everything that is rendered to the screen for the user to see such as the elements, the visual graph, a combination of the visual nodes connected to each other, the assessment files or reports, and the simulation view.

Debug Module

This module will linter and give feedback and indication to the user whenever there is an attempt to perform an action that is faulty in logic. For example, the red color means error or danger while the yellow color means warning. The color based feedback and highlight is used in combination with useful messages giving more detailed information regarding the fault. These are placed accordingly to the source of the fault, whether in the node or in the wire. Runtime errors or warnings during the simulation stage is propagated to the user with detailed information and explanations about the probable cause of error and displays tips in debugging and fixing the program.

Simulation Module

The process of simulation involves the compiling, building, and running the visual code is executed by this module. The compilation stage involves going from the main node which is the entry point of the program followed by the importing of libraries and packages (depending on the target language). After that is the declaration of variables and methods and will continue to parse and convert the visual code to its equivalent source code.

The compilation stage is followed by the building, also known as linking stage. During this stage, the compiled source code is linked with the necessary libraries required by the target programming language. Examples of this are Dynamic Linking Libraries (.dll) and Shared Objects (.so) files.

After the compilation and building stage is the simulation or execution stage. This executes the program and run it with additional features enabled to allow a more dynamic and intuitive interaction between the user and the program.

Transpiler Module

This module transpiles the visual code made by the user into source code in target programming language. This module tests that the transpiled source code compiles and runs correctly as well. This allows for learners to see and compare their work into the C programming language which is part of the education curriculum. This serves as helper for their transition from learning the fundamentals of programming into its application towards programming languages that are more robust, high quality, and industry-standard. The module will walk through each nodes provided by the Visual Nodes Module and will parse each to create a tree structure that will be read and converted by the module into the selected programming language by the user. Since this module is reponsible for the conversion of each visual data into their corresponding textual data, this module also evaluates the parsed tree structure in order to check and interpret expressions in programming.

Assessment Module

The functionality of providing exercises designed for the learning of topics and concepts in programming and evaluation of the results are handled by the Assessment Module. The possible types of exercise range from output-based program to writing an

algorithm that has memory and time limitation and complexity. The module can compare the submitted code output with the expected output of the assessment line by line and show to the user which lines are correct or incorrect.

The software is limited to simulating text-based or command/terminal prompts as the priority is learning the fundamentals of programming. The software does not also compete as an Integrated Development Environment (IDE). The software does not provide networking functionalities such as connection to the internet to send or fetch data. There is also no access level or account management for the user.

Definition of Terms

Algorithm is a set of instructions designed to perform a specific task. Assessments provide exercises for understanding algorithm as a fundamental concept.

Bug is unwanted behavior caused by faulty logic. Learning fundamental concepts reduces proneness to bugs.

Building is linking compiled code with libraries to make an executable. This is a key step in the simulation of the system.

CodeNect combination of the word "Code" which is synonymous to programming, and "Nect" which is taken from "Connect", based on the connection of multiple nodes to create a program.

Compilation is turning human-readable language to machine language. This is the initial step in the simulation of the system.

Conditionals are statements or expressions comparing logic in programming. Assessments provide exercises for understanding conditionals as a basic and fundamental concept.

Data is information digitally stored in and processed by computer. Data are stored in and represented by visual nodes.

Data Structure is grouping and storage of data efficiently in memory. Assessments provide exercises for understanding data structures as a fundamental concept.

Data Types are attributes to determine the size and type of data. Visual nodes represent data types by color for categorization. Assessments provide exercises for understanding data types as a fundamental concept.

Debugging is the process of finding and resolving of bugs in a program. The system provides interactive and visual way in learning debugging.

Dynamic Linking Libraries contain functions and other information used by a Windows program. These are necessary for the system to be stand-alone.

Exercise is a simple coding problem statement with expected output. An assessment is in the form exercise.

Graphical User Interface is the interface of interactive graphical elements. The system provides clean and minimal GUI for easier usage and learning.

Integrated Development Editor provides features for easier text programming. This has features not necessary for the purpose of learning and as such the system is not an IDE.

Loops are statements or expressions that repeat a sequence of code. Assessments provide exercises for understanding loops as a fundamental concept.

Memory is data storage of computer allotted for a program to use. Assessments provide exercises for understanding memory as a fundamental concept.

Module is a component in software that provides specific functionalities. The system is composed of multiple modules that serve and provide the core.

Nodes are visual elements that contains data and can be linked with other nodes.

Programming is the writing of code for instructing computers what to do. The system prioritizes users in learning quality programming.

Programming Language is a human-readable language that a programmer uses. The system provides other programming languages for output.

Runtime Error is an error that occurs when the program is running. Learning fundamentals of programming will reduce the occurences of runtime error.

Semantics is the evaluation of syntax and tokens of a programming language. Learners are not restricted by learning semantics in visual programming language.

Software is a program or collection of instructions operating the computer.

Socket is the input/output slot for connecting nodes to form a graph.

Syntax is a set of rules that defines the structure of symbols. Its version in visual programming are nodes and connectors.

Technology is the application of knowledge in a particular area or field. The system applies technology to provide richer learning experience.

Text-based Programming is the use of texts to write a program. This is also an output by the visual code when transpiled.

Terminal is an interface that accepts input or command in text form. Users can interact and test their program through the terminal.

Transpilation is the conversion of code to other programming language. The system integrates this to facilitate learning and easier adoption to real programming languages.

User-Interface is the layer that the user controls and interacts with. This spans the concepts of text user-interface and graphical user-interface.

Variable a named reference that holds a value in memory for the user to use. Assessments provide exercises for understanding variable as a fundamental concept.

REVIEW OF RELATED LITERATURE

This chapter discusses the collected literature and studies that contribute to attain the objectives of this study after thorough and in-depth search done by the researchers. This presents the theoretical and conceptual study to fully understand the research to be developed.

C++

C++, a programming language created by Bjarne Stroustrup as a superset of the C programming language, is a compiled, statically-type, performant, and feature-rich language used and is the first choice almost anywhere from embedded machine to large system ("A Brief Description of C++", 2021).

The C++ programming language provides powerful and flexible capabilities for abstraction. It is designed to be high-level and efficient as close to machine language as possible (Stroustrup, 1999).

C++ is the choice used and in combination with in Unreal Engine's Blueprints Visual Scripting System as provides safety due to its type system, security, flexibility, modularity, and efficiency (Chu and Zaman, 2021).

Visual Programming Language

Conveniently, explaining what a program does leads to the usage of graphical representation of the control flow, connections, shapes, and more elements. This could also be applied for programming and learning it. Visual programming languages enable users to achieve the same concept (Craft.ai, 2015).

Aside from programming logic, visual programming languages are also used in a wide variety of applications and has corresponding types. Some of these are the following:

The drag-and-drop type of visual programming language uses blocks as elements that can fit into other blocks for composition, similar to a jigsaw puzzle piece. A study that compared drag-and-drop visual programming to text-based programming concluded that respondents were confident in their knowledge and skill in performing simple and basic command with visual language programming, but found it harder to express what they want in drag-and-drop for complex problems. This suffice to using visual programming as first steps in basic programming before proceeding to complex concepts (DiSalvo, 2014).

Flowchart-inspired type visual programming languages provide basic and limited capabilities for programming. The common usage for this is evaluation of the conditionals and flow of the program. It mainly uses arrows and boxes with simple value. Examples of this are Flowgorithm, Raptor, and WebML.

Dataflow type visual programming language commonly used in professional applications moreso for designers than programmers. With this format, there is a wide selection of available capabilites as each block represents a function or procedure which can store and output multiple values through lines or wires. Examples of this are Unreal Blueprint and CryEngine Flow Graph.

The Finite-state Machines (FSM) type uses basic shapes and connections only. This is commonly used for animation and states to visualize the transition from one block to another. Example of this is NodeCanvas.

Behavior Trees type is similar to Finite-state Machines but is more complex and allows for multiple states to be triggered depending on the parameters that match the current state. This is mostly used for complex animation in the game industry. Examples of this are NodeCanvas and Craft.ai.

Event-based type of visual programming language is the simplest and most akin to the traditional text-based programming languages. The simplest illustration to define this is to write a programming code in text form and then assign a graphical element or picture of each keyword. For example, the picture for the keyword "for" will be a looping arrow. Examples of this are Construct, IFTTT, and Kodu.

Visual Programming Software

Visual programming is commonly built-in or provided by industry-size software for allowing non-programmers to also perform complex logic and controls without the need to learn traditional text-based programming languages. This is widely popular in the game development field as desginers and artists can create visual effects through the use of visual programming language.

The development of software that support visual programming for novice programmers such as Scratch and Snap! results in learners learning to code without the need for grammer correctness as needed in traditional text-based programming languages (Bau, Gray, Kelleher, Sheldon, and Turbak, 2017). But Scrach and Snap! has their own programming languages which the visual programming side exports to. This increases the effort

and time required to transition into learning popular programming languages like C, Python, and Java. There are environments which allow the visual code to output in C language but it can not execute, one needs to copy and paste the output to another editor to run it (Abe, Fukawa, and Tanaka, 2019).

For Java, there exist numerous visual programming software such as Symantec Cafe, Visual J++, and Visual Age for Java. The core feature of the software is to enable end-users to manipulate elements of the interface in their natural graphical representation. The software allow for editing the packages, classes, methods, and variables of the available elements. But the following require basic understanding and or experience already with programming, particularly in Java, to make full use of the software (Prokhorov and Kosarev, 1999).

AgentSheets is one of the early pioneers of the concept of visual programming. This visual programming software use block-based type of graphical elements similar to a jigsaw puzzle piece. The elements can be dragged and dropped onto another to create composition for the logic of the program. Since most of the elements are visually represented, this allows for comprehension as a block or group of blocks can explain itself in terms of its purpose, control, and logic. Another key concept of visual programming expressed in AgentSheets is easy sharing of blocks with one another instead of looking at plain texts (Repenning, 2017).

End-User Programming Approaches

The types of programmer range from professional, novice, and end-user. Professional programmers are whole main work is to develop or maintain a code base. Novice programmers can be thought of as professional programmers under training. End-user programmers are those that program but programming is not their main function or career. Another case for comparing the types of programmers are their interest and knowledge when it comes to programming itself. Professional and novice programmers has more in-depth understanding about the processes involved in programming and are capable of programming using traditional semantic and text-based code whereas end-user programmers do not. The following are the various approaches to programming for end-users.

Preferences Programming

This is provided by applications to allow the users to modify the behaviors and

visual appearances of the application itself. These are predefined options in the form of checkbox, radio button, or dropdown menu that the user can interact with to suit their preferences.

Programming by Demonstration

This programming approach uses a system for recording user inputs for future playback. This allows users to work in a general way to program the system what to accomplish by showing the actual actions. This approach is tightly rule-based to enforce the smooth replaying of actions (Harrison, 2004).

Spreadsheet Programming

This approach focuses on the requirements of mathematical knowledge and skills in building formuals and models in the form of functions. Since this approach is visual-based as the spreadsheets constantly indicate the result of calculations for errors (Abraham, Burnett, and Erwig, 2009).

Script Programming

This approach uses scripting languages as opposed to full programming language. A scripting language can be a subset of a programming language or embedded language. These languages are tiny and are generally designed to be used by people whose main domain is not programming. This application can range from game design, music generator, video effects, and prototyping (Ousterhout, 1998).

Gulf of Execution and Evaluation

This book features a study regarding the usage of things. The learning phase that occurs during usage has two gulfs, the gulf of execution wherein the user figures out and attempts how it works, and the gulf of evaluation wherein the user observes and comprehends the results of usage. This understanding in the part of the user is applicable in designing a system where the goal is to assist and improve learning. The gulf presents cases wherein users failing to use a simple object results in blaming one's self and users failing to learn a complex object results in forfeit in further learning. In reality, the fault is not solely on the user, but from the designer and the design of the object.

The study recommends that the design of the system should bridge the gulf by developing it to be accessible and understandable relative to the expectations of the users either through concise information or feedback per step on the behaviors (Norman, 2013).

Difficulty in Learning Programming

Different studies prove that many students has a poor learning in programming in the midst of their programming education. Through observation they found out that a lot of students are unable to read and write code effectively. There are few teachers who claimed that their students are able to meet the standards of programming by graduation however it was admitted that many programming graduates are still unable to program (Carter and Jenkins, 1999).

An average student does not make much progress in an introductory programming course (Robins, Rountree, and Rountree, 2003). Most of the students struggled to get past in learning language features and never had a chance to learn higher programming skills and problem-solving strategies in programming (Linn and Dalbey, 1985). Several working groups have looked into the skill levels of the student at the end of CS1 courses in the past decades. These frequent studies has been beneficial as they prove the mismatch between programming education and the actual programming skill gained (McCracken et al., 2001). Programming teachers believe that one must learn how to read code before learning how to write a code. However, programming students give more attention to reading compared to writing codes. Some says that writing a code is much easier than reading (Lister et al., 2009).

A study that measured students ability to trace through a given programs execution to follow-up McCraken's investigation. Multiple-choice questionnaire was given to CS1 graduating students around the world. The questions required the students to predict the values of variables at given points of execution and to complete short programs by inserting a line of code chosen from several given alternatives. The result was disappointing across the board as it shows that many students are unable to trace (Lister, 2004).

Another study found that novices were unable to mentally trace interactions within the system they were themselves designing (Adelson and Soloway, 1985). Another study reports that an inability to trace code linearly as a major theme of novice difficulties (Kaczmarczyk, Petrick, East, and Herman, 2010). The analyses of quiz questions indicate that many students fail to understand statement sequencing to the extent that they cannot grasp a simple three-line swap of variable values (Corney, Lister, and Teague, 2011).

Software Visualization

Software visualization tools have different types for different use cases. The fol-

lowing are broad classifications of the visualization tools: program visualization, algorithm visualization, and visual programming.

Program visualization is used to determine the runtime behavior of a program and visualize it for the user to see and inspect the information. This is commonly used for debugging programs such as showing of virtual memory and CPU usage.

Algorithm visualization is used to visually show the each step in the process of running an algorithm. An example of this is sorting algorithm wherein elements that represent a value to be sorted are in every iteration selected, compared, and sorted. This is to show a high level of abstraction for learning and understanding the concept of an algorithm.

Visual Programming is similar to program visualization but is distinct enough to be set as a different classification. Compared to other visualization type, visual programming allows interaction with the visualization rather than the visualization can be interacted. This is used as a mean to program visually as opposed to visually see the program.

Teaching and learning programming through visualization is a pedagogically sound approach. The nature of a program is that the code is static but during runtime it is dynamic. The dynamic aspect is difficult to learn at first especially for novice programmers as they need to form a mental model of the processes involves based on logic and set of theories (Sorva, 2012).

Visual Learning

Information is retained more in memory through visual formats. Visual information can be presented in various formats such as images, diagrams, graphs, video, and simulations. This approach in learning helps the instructors to convey their lesson better and clearer while the students develop visual thinking skills. This skill is the comprehension of association of data such as concepts, theories, and ideas into graphical elements like imagery and diagram (Raiyn, 2020).

Visual learning can be improved more through the addition of interaction using visual interactive tools. This is beneficial in many domains that require logical thinking and skill such as programming. Interaction and visualization at the initial level of coding motivates the interests and engagement of learners even at the young age. This approach has been very effective for the Scratch programming environment as they adapted to adding visualization and media content creation to programming activities which are trends in the culture of youth. Learning through exploration and sharing to peers, this motivated young

people to focus less on direct instruction that other programming languages provide (Maloney, Resnick, Rusk, Silverman, and Eastmond, 2010).

Having a physical design, blueprint, or a diagram that serves as guide for the product or machine to be made has been the traditional method for manufacturing complex and expensive things. The same principle applies in programming. Programmers manually input code from their brain which can be called as mental model to task the computer into doing a complex routine. But this is a challenge for many reasons such as other people does not inherently have the same mental model regarding the solution and structure. Another reason is that the level of familiarity and expertise to a particular tool or environment used is not the same for all programmers. So intead of from one's mental model to code, it should suffice to create and visualize the model itself before jumping into directly generating the code. This allows for coordination between multiple programmers as they have the same guide for the solution and concept. This also applies for novice in programming to learn that visualization before coding is a discipline one must come to understand and put into practice. This application of visuals into learning and execution could be of great benefit to reduce the complexity, effort, and time consumption (Ottosson and Zaslavskyi, 2019).

Persistence in a Game-Based Learning Environment

Persistence has been a great challenge in online learning environments. The motivation to finish a chalenge may have a negative efffect on learning at specific levels and hence on learning in general. Gaming and interactivity have been suggested as important features in increasing persistence in online learning. Persistence, the determination of learners to complete a learning process and obtain their goals despite challenges, has been determined to be a very useful and valuable skill when solving problems (DiCerbo, 2016). The researchers studied microlevel persistence in the context of acquiring computational thinking, which is the thought process of solving problems through abstraction (Israel-Fishelson and Hershkovitz, 2020).

In game-based learning environments, learners often receive immediate response or feedback regarding the solution they have submitted. Incorrect solutions have a different indicator as opposed to correct solutions. Instead of accepting the solution as is if correct, there will still be a feedback to inform that though it is correct, there can be improvements and better way to do it. To achieve such an understanding, the learners require metacognitive knowledge and beliefs together with metacognitive skills of monitoring and control

(Guven and Cabakcor, 2013).

ISO 9126 Software Engineering - Product Quality

The ISO 9126 Software Engineering - Product Quality is a a set of standards for software testing internationally recognized and approved. Its goal is to address well known biases of human that has an effect on the delivery and perception of a software project. This standard defines the following for usage with software development lifecycle: maintanability, efficiency, portability, reliability, functionality, and usability of the software product (Sukoco, Agus and Marzuki and Cucus, Ahmad, 2012).

Functionality The functionality subset of quality has the following attributes: suitability, accuracy, interoperability, security, and functionality compliance. These functions are provided to satisfy and meet the needs.

Reliability The reliability subset of quality has the following attributes: maturity, fault tolerance, recoverability, and reliability compliance. It focuses on the capability of the product in maintaining its performance and quality under light or stress conditions for a period of time.

Usability The usability subset of quality has the following attributes: understandability, learnability, operability, attractiveness, and usability compliance. These adheres to the difficulty and ease of an individual that must exert and face when using and operating the product.

Efficiency The efficiency subset of quality has the following attributes: time behaviour, resource utilization, and efficiency compliance. These attributes refer to the relation between the performance of the software and the amount of resources used in conditions.

Maintainability The maintanability subset of quality has the following attributes: analyzability, changeability, stability, testability, and maintanability compliance. These refer to the effort required to make adjustments and modifications to the product.

Portability The portability subset of quality has the following attributes: adaptability, installability, co-existence, replaceability, and portability compliance. These characterize the ability of the product to be used or ported from across multiple environments.

User Interface and User Experience

A user interface (UI) refers to a system and a user interacting with each other through commands or techniques to operate the system, input data, and use the contents.

This ranges from systems such as computers, mobile devices, games, to application programs and content usage. On the other hand, user experience (UX) refers to the overall experience of the user. This includes the perception, reaction and behavior that the user may feel and think in direct or indirect usage of the system, product, content or services. It is a concept that is widely applied not only in software and hardware development but also in services, products, processes, society and culture. Both UI and UX is an interface through which a person can interact with a system or application in a computer and communication environment, which is classified into a software and hardware interface. Software interface is represented by the user interface while hardware interface is categorized into a plug or an interface card connecting the computer and its peripheral devices. UXs has four key axes which are needs, expectations, attributes and capabilities. Hence, it identifies the problem with the need of the users, applies motivation and manage the expectations of the users (Joo, 2017).

V-Model

The V-model process is a software development process which is an extension of the traditional waterfall model. The model shows the relationships between each of the different phases in the life cycle of the development process, each with an associated testing phase. After the linear top to bottom phases are complete, the process proceeds to bottom to top phases for testing and to complete the model cycle. The V-model is a well-structured method in which each phase is implemented following the documentation provided previously. The primary focus and purpose of this model is to improve the efficiency of development and to ensure the effectiveness by following the relationship of each phase with its associated testing phase (Rook, 1986).

The traditional V-model is composed of the following phases:

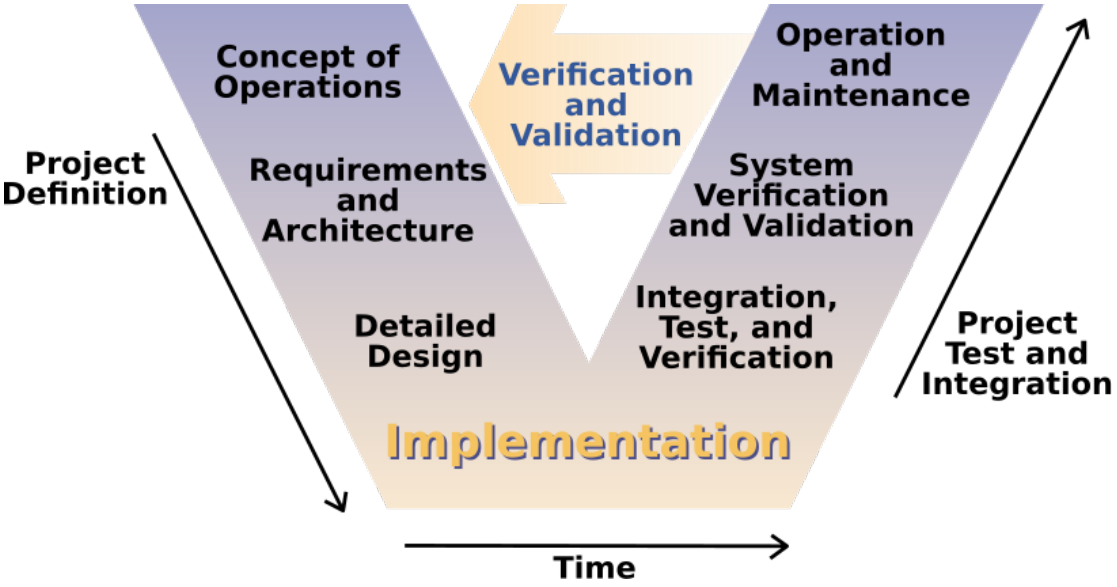


Figure 2. V-Model

Requirements. Involves the gathering of data, analysis of the gathered data, and preparation of the system requirements in defining the scope and features of the project. This stage also involves the documentation of the requirements.

System Design. The documents created in the previous stage will be used to generate more specific and technical documents and designs about the software to be developed for this stage. The documents outline the components, modules, and high-level guidelines for business logic.

Architecture Design. The technical designs from the System Design stage will be used to generate specifications with lower level of technical details about the software and its modules. The technology stack is also selected in this stage. During this stage, the tests are also prepared for future use.

Module Design. In this stage, low-level designs are developed from the high-level designs generated from previous stages. This will include specifications regarding each individual module, component, interface, and so on. Unit tests are also prepared in this stage.

Implementation and Coding. This stage the implementation through programming starts. It starts with coding the each module individually with unit tests along applied following the designs made in the initial stages of the development life cycle. Integration of the modules to the system is done afterward and is run through system tests.

Testings. Tests are further applied to the system such as unit testing, integration

testing, system testing, and acceptance testing. Passing all these tests will be considered as the verification and validation of the project.

Context Diagram

Context diagram is a simple diagram that shows the source systems contributing data to a system as well as the major user constituents and downstream information systems that it supports. This diagram is so simple that it makes it perfect for agile requirement management. This diagram also called Level 0 data flows diagrams because if one were to put arrows on the connections between sources and targets, the diagram could serve as the cover sheet of a data flow diagram packet that many analysts prepare for traditionally managed projects. This diagram greatly reduce project risk because they are easy for a teams business partners to understand (Hughes, 2016).

Data Flow Diagram

A data flow diagram illustrates the processes, data stores, and external entities in a business or other system and the connecting data flows. It is a graphical representation of the flow of data through information system. DFD was first proposed by Larry Constantine, the original developer of structured design in 1970s. It is a primary artifact and is required to be created for every system in a structured approach. It provides a different abstraction level that is useful in system designing because of its hierarchical structure. It shows data flow from external into the system and shows how the data moved from one process to another. There are four symbols for a data flow diagram: 1.) Squares or Ovals which represent external entities. It can be a person or a group of people outside the control of the system being modeled. It shows where information comes from and where it goes. 2.) Circles or Rounded Rectangles shapes represent processes within the system. They show a part of the system that transforms inputs to outputs. The name of the process in the symbols usually explains what the process does so that it is generally used with the verb-object phase. 3.) Arrows represents how the data flows. It can be electronic data or physical items or both. The name of the arrows represents the meaning of the packet that flows along. It also shows direction to indicate whether data or items are moving out or into a process. The last symbol is 4.) Open-Ended Rectangles which represents data stores, including both electronic stores and physical stores. Data stores might be used for accumulating data for a long or short period of times (Aleryani, 2016).

Gantt Chart

Gantt chart is a classic tool in project management. It is one of the most known and widely used planning and management tool in projects in different domains. The principles for the development and design of Gantt chart are time-focused, objective, deterministic, analytic, accountable, and sequential (Geraldi and Lechter, 2012).

Time-focus as projects have a target time for the either the completion or progress milestone. Each task should be well coordinated in time and work as a crucial part in project management.

Objective as projects must have ground in reality for the objectives to be met in a realistic and feasible manner.

Deterministic as each task should be properly defined, studied, and defined. This ensures that there should be no uncertainty in the objective and method of the tasks.

Analytical as projects are the sum of different and subset of tasks. A project must be analyzed very well and divided properly into smaller tasks. This should take into consideration the execution and scope of each task.

Accountable as a project gets divided into smaller parts, a project is also divided and assigned to different person. Each person should be accountable for the progress and completion of the task assigned.

Sequential as in the management of a project, there is always a sequence or order required for further tasks to be started by waiting for the completion of other tasks it depend upon. This sequence fit as a timeline analogy.

Ishikawa Diagram

Ishikawa diagram is also called the fishbone diagram and cause-and-effect diagram in that it is a graphical technique in the shape of a fish skeleton used to numerous causes of a phenomenon. It is commonly used to identify and analyze causes and its complex relation to each other that contribute to the specific problem.

A finding of a study about the use of Ishikawa diagram as an appropriate theoretical framework for representing visually and analyzing technology of complex factors of major improvements and innovations over the course of history and time. This graphical representation tool presents a simple and clear the order and relation of the causes and roots of a problem addressed by the change in technology (Coccia, 2017).

Six Learning Barriers in End-User Programming Systems

The researchers identified the following aspects prone to false assumptions that include fundamental and basic concepts in programming as barriers to learning programming. These barriers closely related to the concept of interfaces of a programming environment such as the constructs of the language itself and the availability of libraries, features, and syntax that can be used to achieve desired procedures (Ko, Myers, and Aung, 2004).

Design barriers are internal difficulties of a problem in programming. Solutions to problems which are difficult to visualize affect the learning experience and may lead to false assumptions and confusions.

Selection barriers occur when learners know what to do but are unable to identify which of the available tools and features in the programming interface is to correct.

Coordination barriers are difficulties in using libraries provided by the programming environment in compliment with another. Learner may know how to solve individual and simple tasks but fails to combine the approaches to solve complex problems.

Use barriers are inherent to users new to the environment. The unfamiliarity to the interface hinders its usage due to the lack of information and guide.

Understanding barriers involve the obscurity of the processes the programming interface do that are hidden to the users. This occurs when learners fail to evaluate and undestand the behavior of the program relative to their expectations.

Information barriers are difficulties in obtaining information about the internal workings of the interface. This occurs when the environment provides no method for the users to test their hypothesis regarding the behaviors of the environment.

The barriers explicitly defined are closely related to each other. The effect of having difficulties in overcoming a barrier affects the learning of another barrier.

Visual Programming vs. Text-based Programming

Programming is seen to be as the career of the future as technology continues to scale up and improve. For this reason many countries are already promoting and implementing programming subjects to their curriculum in primary education (Williams, Alafghani, Daley, Gregory, and Rydzewski, 2015). Most of these formal subjects use a visual programming language for teaching instead of the traditional text-based programming language.

It would seem to be appropriate and more productive to teach text-based language as it is the standard and the type of programming language used for the development of

softwares and applications in the industry and the real world. However, in considering the comparison of the engagement, interest, and actual learning of the students at the introductory level, visual programming language is more suited and ideal as results showed that the motivation of the subjects that use visual programming language improved (Tsukamoto et al., 2016).

Studies also show that using a visual programming language like Scratch to teach students programming and transitioning to real programming language (text-based) shows a marginal improvements to their understanding of computer science concepts (Armoni et al., 2015). Visual-based environment as compared to textual environment also displays positive results in terms of the interest and motivation of the learners to pursue programming (Saito, Washizaki, and Fukazawa, 2017).

Novice programmers take longer time to learn programming because of many constraints that are needed to be learnt first such as the syntax and semantics of a programming language. Common errors and difficulties in text-based programming languages include type corrections, misspellings, typographical errors, and grammatical or syntax errors. Another factor is that students whose native language is not English find it harder to type because the keywords in most programming languages are in English. Even the tools used, text editors or integrated development editors (IDE) are hard to operate and use (Liu, Wu, and Dong, 2010).

Prototype of Visual Programming Environment for C Language Novice Programmer

The C programming language is often the first programming language taught and learned in higher education. This is also the case for the author's institution, the Kanagawa Institute of Technology - Department of Information Engineering. The C language classes are thrice a week in the first year. The students use Visual Studio for programming. In text-based languages like C, students must memorize tokens or keywords such as data type and syntax in general. These are difficulties for beginners. Even a single typographical error in text-based programming can lead to undesirable messages like compilation errors.

The research and development of a programming support system for beginner programmers has attained an advanced stage. One of which is the block-based visual programming language environment such as Scratch or Snap!. Block-based programming environments allow student to edit the program by visually selecting blocks and combining them together. They can create programs even without the need to memorize keywords or

to understand the syntax in formulating the grammar for the semantics. Scratch and Snap! use their own programming languages which are not suitable for learning other popular languages like C, C++, or Java.

With these factors considered, in this investigation the researchers have developed a visual programming environment for the C language with the goal of lowering the barriers between starting in programming and learning of the C programming language. This programming environment is a Web application that has functions to edit a C language program and to execute and step through the program and trace the changes in the variables (Abe et al., 2019).

On the Design of a Generic Visual Programming Environment

Visual programming languages are commonly embedded and coupled within environments that are visually interactive. The identification of visual programming language is associated with its environment. Therefore, the creation of a visual programming language is tightly integrated with the creation of its environment. The Requirements of a visual programming environment include graphical elements with relationships graphically shown as each element is connected with another element. Making an algorithm must be done graphically as well through editing elements and creating connections between related data. Each element contains underlying data structures that can be complex as there are more information stored and used such as the visual representation, logical connection, domain knowledge, and more. Parsing group of elements, or diagrams, are difficult in that a parsing algorithm must be employed for handling such case.

A generic visual programming environment can be represented as a group of textual and visual specification tools. Designing a visual programming environment requires consideration for the semantics and syntax of the language as well as the visual interface. To handle with ease the maintenance, modification, and reuse of a visual programming environment, modules are needed to be specified clearly and with their interactions with one another (Zhang and Zhang, n.d.).

Environment pi J for Visual Programming in Java

There is a wide known visual tools for programming in Java such as Symantec Cafe, Visual J++ by Microsoft, and Visual Age for Java by IBM. The main feature of the tools listed is the possibility to modify the graphical elements of the user interface. The

tools support representation in graphic forms of program structure for packages, classes, methods, and properties. The tools apply the visual representation to view, modify, and debug the programs. There is a difficulty to consider that is related to using the tools listed, and that is it is necessary for the users to have relatively high knowledge in programming, particularly in the Java programming language.

Pi J is a programming environment for developing professional software basing on the concepts of object-oriented programming. Pi J can operate with two kinds of file format and it allows opening and saving of text file in plain Java. It also allows opening and saving of file as structured format that the tool can read. The tool supports most of the features provided by the Java programming language (Prokhorov and Kosarev, 1999).

HASKEU: An editor to support visual and textual programming in tandem

This paper shows the effectiveness and usefulness of combining textual and visual system where a change between both system updates both interfaces. The application of textual and visual systems in combination with one another allows users and learners of programming to easily develop programs. Focusing on the visual representation but at the same time seeing the textual representation. This enables them to understand the effect of a change in both formats. Learning through this will make it easier for learners to transition into more advance and complex concepts. HASKEU was developed in a research project to assist end-user functional programming for Haskell programs (Alam and Bush, 2016).

Learning programming is most times a time-consuming and frustrating task. Writing and testing programs after the skills are learned can also be a laborious endeavor. A textual program is one where the program presents a set of commands in textual form for function and variable definition. Textual programming tests our analytical, logical, and verbal thinking abilities. Visual programming tests our non-verbal thinking ability as it uses meaningful graphical representations. Visual representations assist learning and retention in memory and may provide an incentive to learn programming without language barriers.

In 1990, a committee of functional programmers created a well-developed and powerful functional programming language called Haskell. Functional languages are based on the lambda calculus. These mean that the programs have no concept of state. Functional programs are pure functions which take input and produce an output. In recent years, there is an increase in the usage of functional programming but one thing to be considered is that functional programming can be overwhelming to learn as it is very different to other more

common type of language such as declarative or imperative programming.

HASKEU is a prototype programming and development environment for the Haskell programming language. This programming system was developed to support both textual and visual programming. The HCI (HumanComputer Interaction) techniques are used extensively by the design of HASKEU. The HCI techniques include rules of design for UI, data display, icon, and direct manipulation.

The Scratch Programming Language and Environment

The Scratch is visual programming language and environment that allows users to create media-rich projects such as games and interactive media. Scratch is an application that is used to create projects provided with media and scripts. Assets like images and audio can be imported or created directly within the application through the built-in paint and audio recorder tools. Any of these things, that a text-based programming environment can, is possible through the use of colorful command blocks to control graphical objects called sprites set in a background called the stage (Maloney et al., 2010).

Users learn Scratch as they use it, experimenting commands from the provided palette or exploring projects from existing projects. Scratch was designed to allow scripting, provide immediate feedback for the execution of scripts, and making the execution and data visible to motivate users for such self-directed learning. The following are the design followed by Scratch:

Single-Window user interface. The user interface of Scratch makes navigation easier as it only uses a single window with multiple panes to ensure that key components are always visible.

Liveness and Tinkerability. One of the key features of Scratch is that it is always live. There is no compilation phase. The program listens to each interaction between the user and the interface and process it accordingly to provide smoother experience and immediate feedback to the users.

Making Execution Visible. Scratch provides feedback visually to show the execution of scripts. There is an indicator around the script element for users to easily identify and follow it. This feature helps users understand the flow of the program such as when it is triggered and the duration of the script.

No Error Messages. When people play with blocks such as LEGO, there is no error message encountered. Blocks fit together only in certain ways, and it is easier to

get things right than wrong. The shapes of the block suggest the possible position and orientation for it to properly fit and experimentation and experience teach what works and what does not.

Making Data Concrete. In most programming languages, variables are abstract and harder to understand. Scratch turns variables into concrete objects that the user sees and manipulates, through tinkering and observation making programming easier to understand.

Minimizing the Command Set. Scratch aims to minimize the number of command blocks while still allowing a wide range of project types. One might point out that flexibility, convenience of the programmers, and more features are better than a small set of commands. In Scratch, every command consumes screen space in the command palettes, so there is more cost to increasing the set of commands. Addition of more commands requires additional categories or forcing the user to navigate and scroll down to see all the commands available within a given group.

CodeMonkey

CodeMonkey is a learning environment for developing computational thinking aimed primarily at elementary and secondary school students. This learning environment follows the context of game-based and challenge-based learning methodology. CodeMonkey is different than common block-based programming approach like Scratch in that students are required to input code at the initial phase of the game. Nevertheless, no prior programming knowledge is required. Learners need to aid a monkey, the main character, in catching bananas while facing various challenges and obstacles. There are multiple worlds, each with theme and set of challenges, progressing from one world to another is only possible if the learner completes the previous world and its challenges. Users may submit a solution to a challenge, then the system automatically checks it and responds with immediate feedback based on the submission. Feedback includes stars which act as rewards to further motivate the learner into solving more challenges (Israel-Fishelson and Hershkovitz, 2020).

METHODOLOGY

This chapter shows the materials and methods that the researchers used for the development of CodeNect: Visual Programming Software for Learning Fundamentals of Programming.

Materials

For the development of CodeNect: Visual Programming Software for Learning Fundamentals of Programming, the researchers used the following specifications:

For the software requirements, the following materials were used in the development of the software: Linux 5.4 Kernel with Manjaro distrubution as Operating System, Terminal for running commands, Vim for text and code editor, OpenGL for rendering, GLFW for windowing and input, DearImGUI for immediate mode graphical user interface (GUI), TinyC Compiler for running transpiled code at runtime, and C++ as programming language. The following are used for the deployment of the software: Microsoft Windows 7 and above, C++ Runtime libraries, and the C++ programming language.

For the hardware requirements, the following materials are used in the development of the software: Laptop with 2 GB of RAM (Random Access Memory), processor of Intel Core 2 Duo (1.4 GHz), and storage of 80 GB HDD (Hard Disk Drive). The following materials are used for the deployment of the software: at least Intel Core 2 Duo at 1.4 GHz, 2 GB of RAM, and 1 GB HDD of storage.

Method

The researchers used the V-Model methodology of Software Development Life Cycle (SDLC) for the proposed software to be developed. The V-Model methodology is a linear development methodology that focuses and follows a strict and incremental steps of stages. The initial phases are generally focused on planning and designing the system, the next phases are focused on implementation and actual programming. After that, the model will go in upwards direction for testing and verification of the project. The development of the software follows the timeline (See Appendix Figure 28).

The V-Model figure shows the following stages:

Requirements.

In this stage the researchers conducted a survey to gather data from students, instructors, and learners in the field of technology and under the course with programming subjects such as Bachelor of Science in Information Technology, Bachelor of Science in Computer Science, and Bachelor of Science in Computer Engineering as respondents. The data gathered (See Appendix Figure 1) are evaluated and assessed to determine the knowledge and understanding of the respondents in regards to the fundamentals of programming, experience and feedback on traditional text-based tools and software. Problems are identified and the Ishikawa Diagrams are constructed (See Appendix Figure 15, 16, and 17).

System Design. The researchers assessed the gathered data and study the information in order to construct a context diagram representing the manual way that the study will solve (See Appendix Figure 26). The schedule of the development and the allotted time for each is task is planned through the Gantt chart (See Appendix Figure 28).

Architecture Design.

The researchers in this stage designed and developed specifications that served as the blueprint of the software. The libraries, packages, tools, and more are finalized and prepared for later use (See Appendix Figure 27).

Module Design.

The researchers identified and defined the scopes and specific features of each module and how each is integrated along the system to work with other modules and components to ensure that each module is decoupled and can be tested without dependency in other module (See Appendix Figure 29).

Implementation and Coding.

The researchers started to program each module in the C++ programming language using Vim as the primary text editor. Compiling and running the software will be done by running a command in the terminal. The rendering backend is the OpenGL and the GLFW for the windowing and input framework. Each functionality of each module was tested using unit tests by running the test every functionality that is implemented. After each module passes the associated test, all is coupled and integrated to a single system. The end product of this phase is the CodeNext: Visual Programming Software for Learning Fundamentals of Programming.

Testing.

The researchers applied tests in the software that was developed. Further testings are done. Bugs, errors, and misbehaviors were handled and fixed. This test ensured that the functionality, efficiency, usability, portability, and reliability of the software meets the standard.

Evaluation.

The evaluation to be used is the ISO 9126 - Product Quality. The evaluators tested the software thoroughly to specifically evaluate their experience and their learning using the standard metrics. The technical and non-technical evaluators assessed the functionality, reliability, usability, maintainability, portability, efficiency, and user-friendliness of the software.

RESULTS AND DISCUSSION

System Description

The study entitled "CodeNect: Visual Programming Software for Learning Fundamentals of Programming" was developed with the purpose of aiding beginners in learning the fundamentals of programming in the field of technology and to serve as an intermediary software in helping beginners get familiarized with programming.

The researchers surveyed (See Appendix A) students taking courses with programming subjects in the collegiate level. The identified problems from the survey results show that students find it difficult to write code and solve problems using programming. The software alleviates the cognitive overload by providing visual elements in the form of nodes and connections as opposed to traditional text-based style of programming.

The software was developed and completed at the College of Engineering and Information Technology in Cavite State University - Main, Indang Campus. The related resources such as studies and researches were gathered for valuable information from the Internet and libraries for the benefit of further improving this study.

The methodology used for the development of the software is the V-Model. The researchers followed the phases of the V-Model accordingly as requirements, system design, architecture design, module design, implementation and coding, and testing.

The features designed and implemented as core functionalities and modules of the software adhere to the problems and requirements identified. Each of the components meet the needs in learning of the end-users towards programming. The seven modules of the software are: input/output module, visual nodes module, transpiler module, filesystem module, simulation module, debug module, and assessment module (See Appendix Figure 29).

The software was developed using the C++ programming language with the usage of GLFW and OpenGL for rendering and other multiple open-source libraries for the Linux and Windows desktop platforms. TinyC Compiler is used for compiling and running the transpiled C code at runtime. Adobe Photoshop and Aseprite were used for the creation of media such as icons and logo. The evaluation method for the system is ISO 9126 (see Appendix C).

The researchers gathered and reviewed the results of the form for feedbacks for further evaluation of the effectiveness and functionality of the system (see Appendix E).

System Overview

The software CodeNect is a stand-alone desktop application that needs no process or installation. The software can be run immediately after being downloaded as all its dependencies are bundled along the software, giving a smooth experience for the end users.

Figure 3 shows the logo of CodeNect.

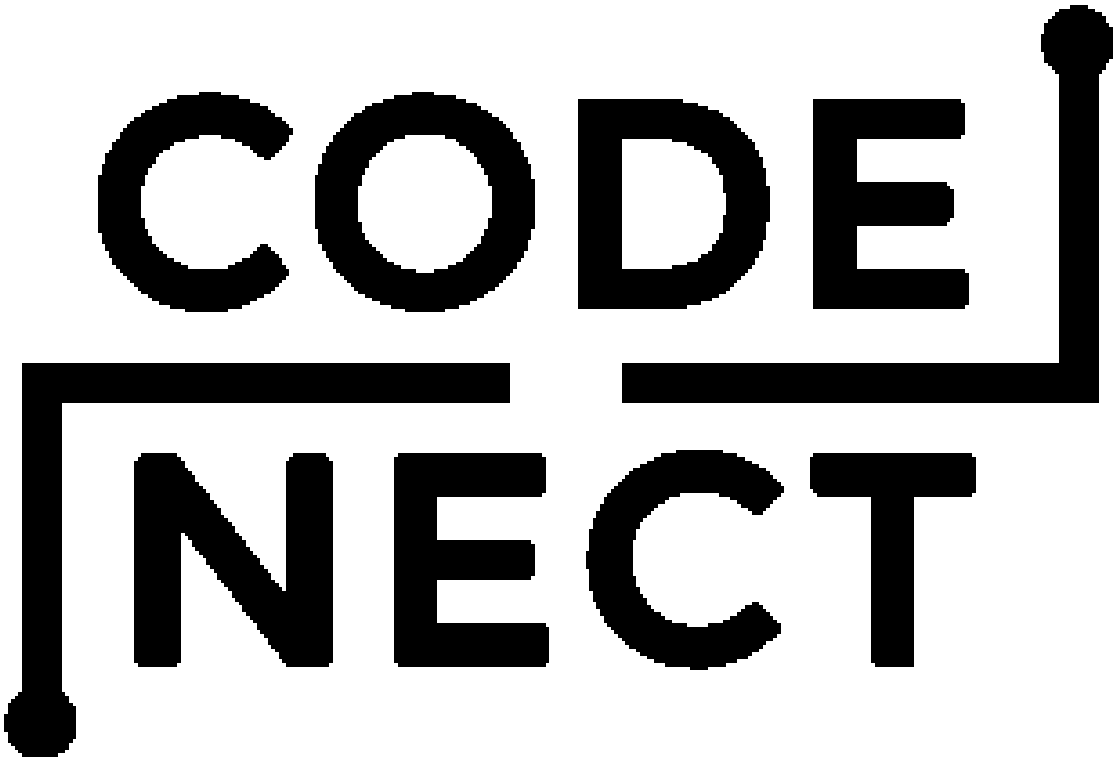


Figure 3. Logo of CodeNect

Figure 4 shows the home and welcome screen of CodeNect upon opening it. The home screen shows basic actions on opening the Sidebar, Command Palette, Terminal, and the Inspector.

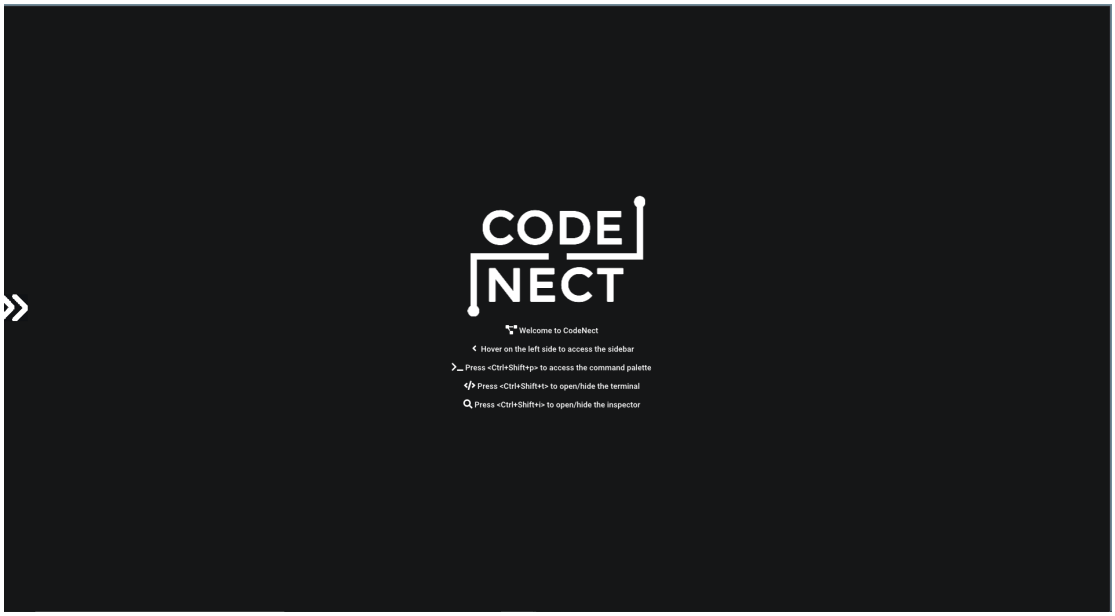


Figure 4. Screenshot of Home screen of CodeNect

Figure 5 shows the node interface that serves as the workspace for the users to code visually by creating and managing nodes and connections. The figure shows an example of a basic visual program.

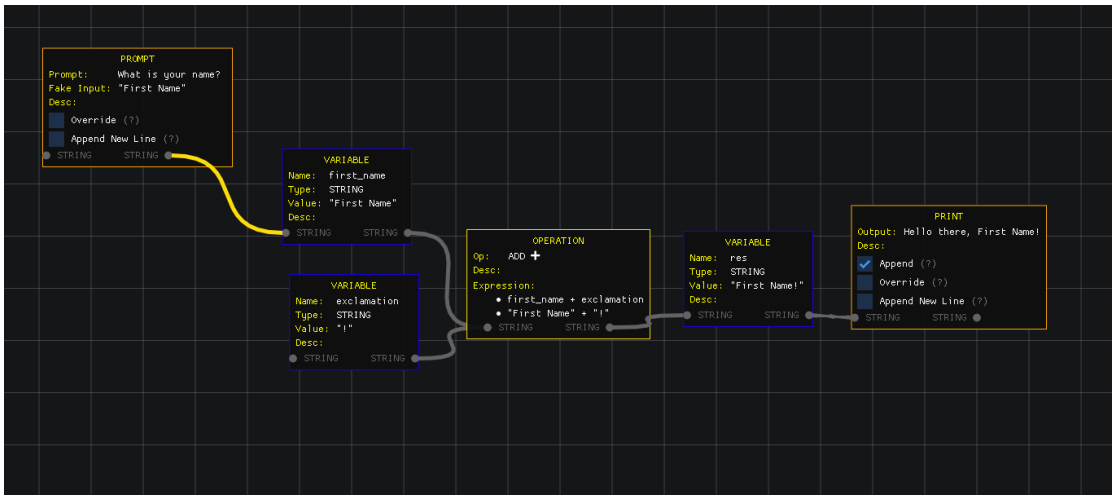


Figure 5. Screenshot of Node Interface of CodeNect

Figure 6 shows the node interface context menu that lists all the possible kind of nodes, arranged by category, that the user can create. The context menu can be opened by right-clicking anywhere on an empty space on the node interface.

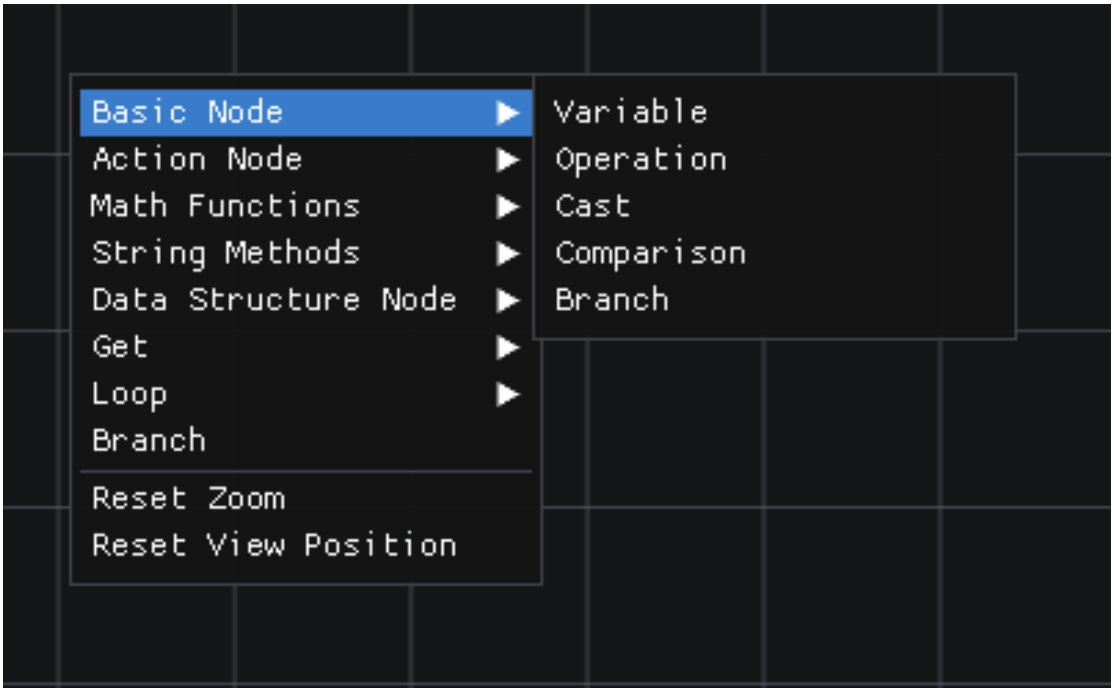


Figure 6. Screenshot of Node Interface Context Menu in CodeNect

Figure 7 shows a window for creating a node of kind variable with the base required data such as name, data type, value, and a conditional field for description or comment.

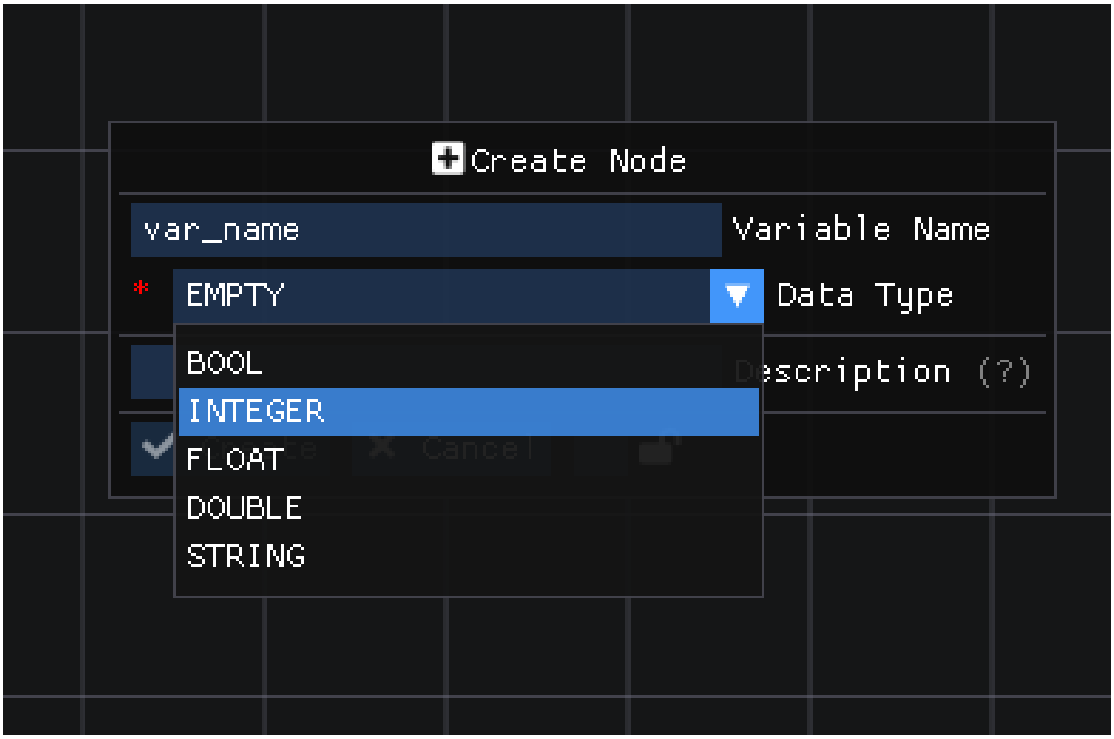


Figure 7. Screenshot of Node Creation Sample in CodeNect

Figure 8 shows the Sidebar menu where it can be toggled by the user by hovering the mouse to the left side of the screen. This allows user to choose options from buttons for

the following: project for creating, opening or closing of project; terminal for compiling, running, or opening the terminal; inspector for opening the inspector window; docs for opening the docs window for documentation and guides; assessments for opening the assessments window or visual coding exercises; settings for configuring the software; about for opening the about window; and help and support for checking the keyboard shortcuts, libraries used, details for help and support contacts.

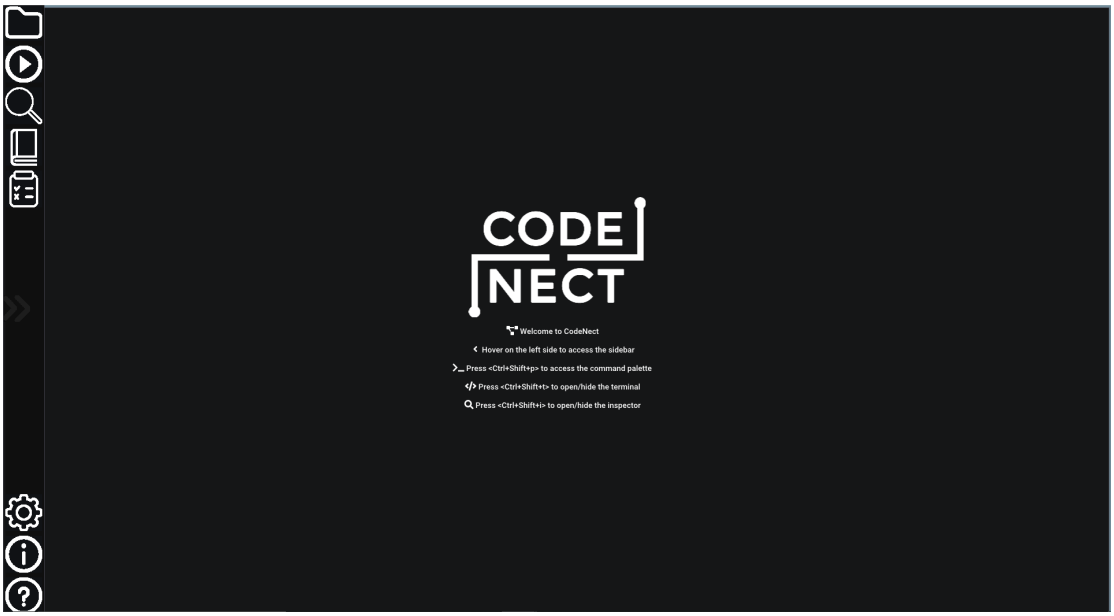


Figure 8. Screenshot of Sidebar Menu in CodeNect

Figure 9 shows the dialog window for creating a new CodeNect project. A project information can be filled with the path to the save location, title, and the name of the author.

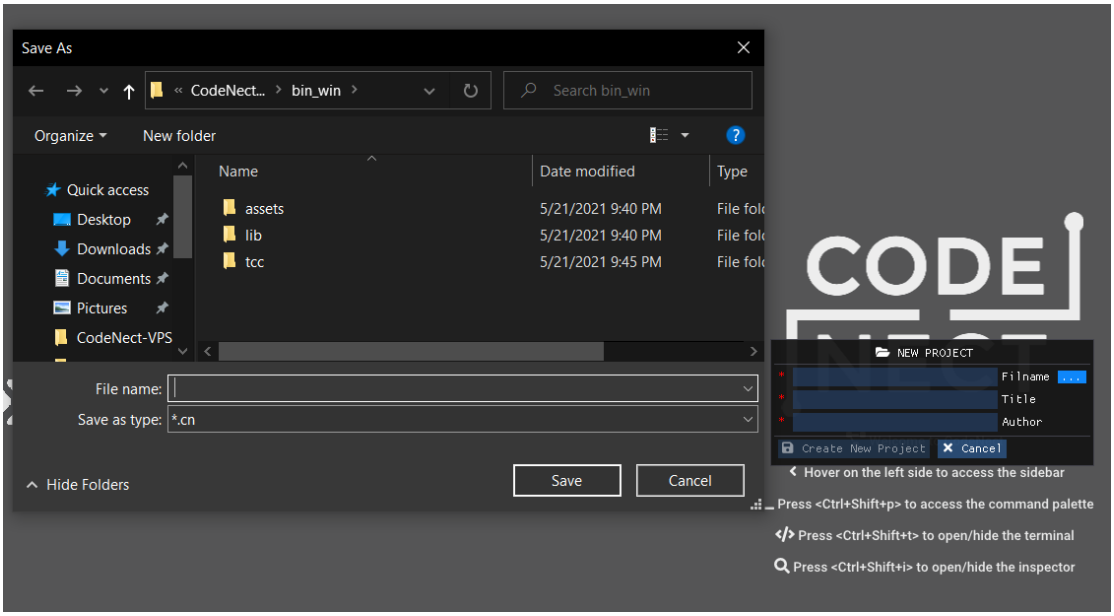


Figure 9. Screenshot of New Project Dialog in CodeNect

Figure 10 shows Terminal window can be toggled open by the keyboard shortcut **ctrl + shift + t**. The Terminal displays messages for the status of the visual code. The Terminal can display warning (yellow colored) and error (red colored) messages for debugging.

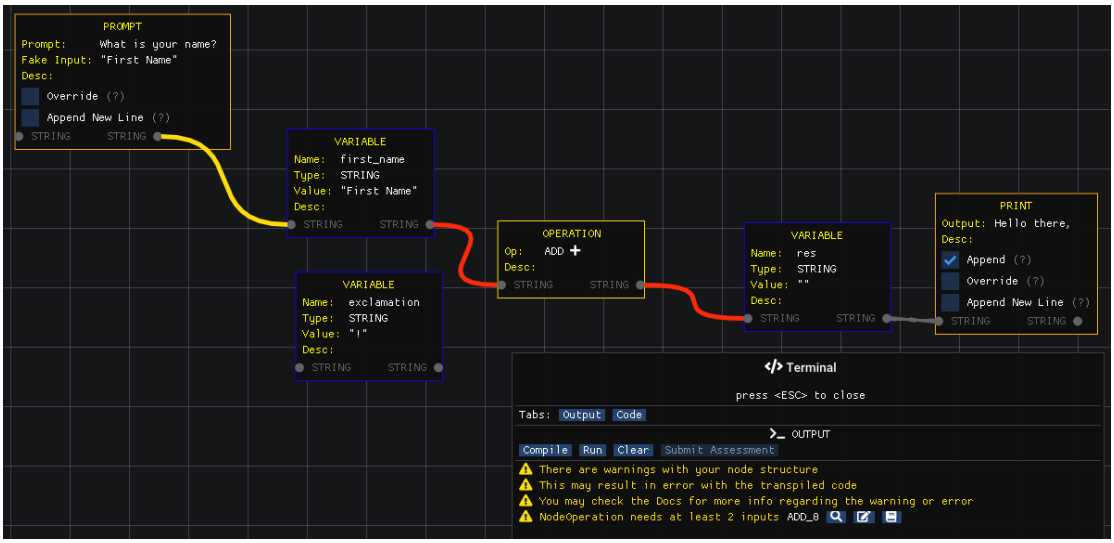


Figure 10. Screenshot of Terminal and Debug in CodeNect

Figure 11 shows code tab in the Terminal window that displays the transpiled visual code into the C programming language. The code can be copied into clipboard or save into a separate file.

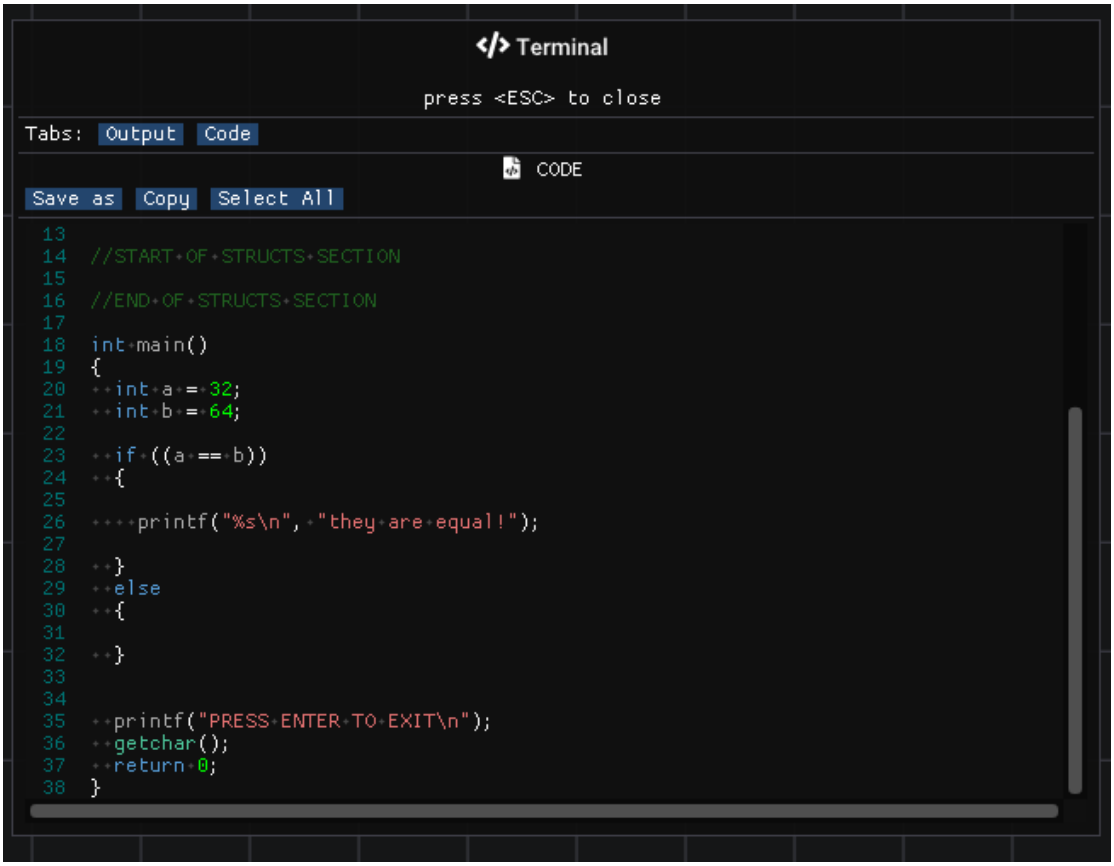


Figure 11. Screenshot of Transpiled Code Tab in Terminal

Figure 12 shows the transpiled code executed and running natively in the command prompt (for Windows) or terminal (for Linux).

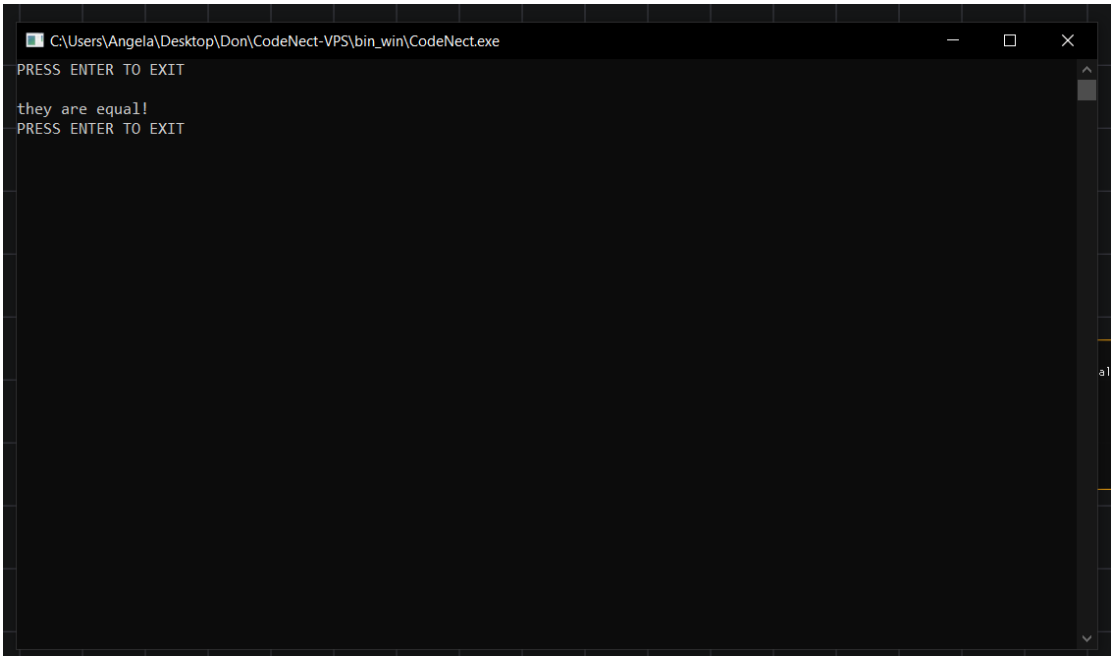


Figure 12. Screenshot of Running Transpiled Code

Figure 13 shows the Inspector window. This window allows users to easily see and browse through all the nodes within a project and inspect the attributes. The Inspector window also allows for rearranging the order of the nodes stored. The Inspector window can be toggled open by pressing the keyboard shortcut of **ctrl + shift + i**.

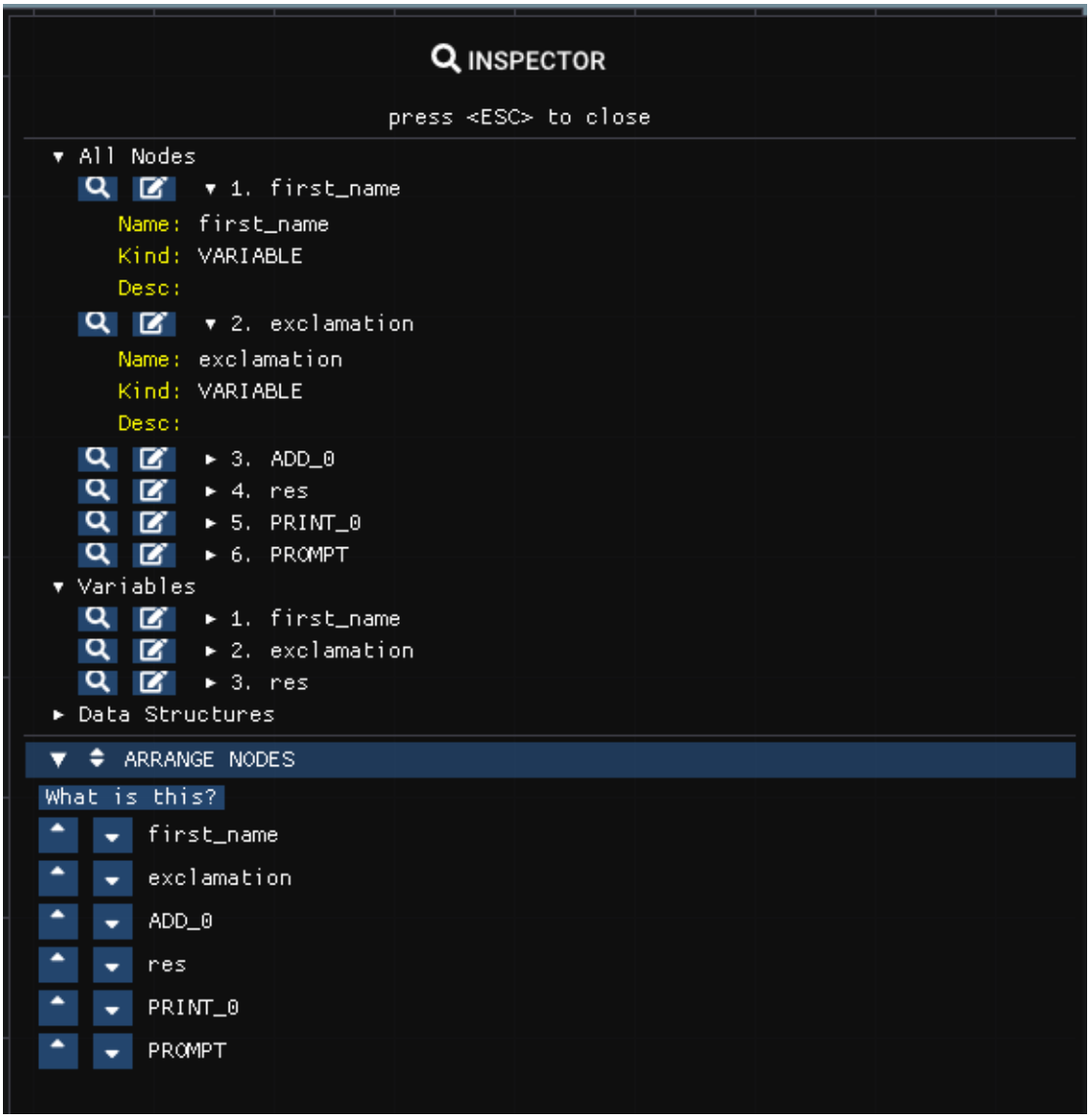


Figure 13. Screenshot of Inspector in CodeNect

Figure 14 shows the Docs window for accessing the different guides about particular topics in programming. This is accessible for users as help when debugging and understanding certain warnings or errors in their visual code.

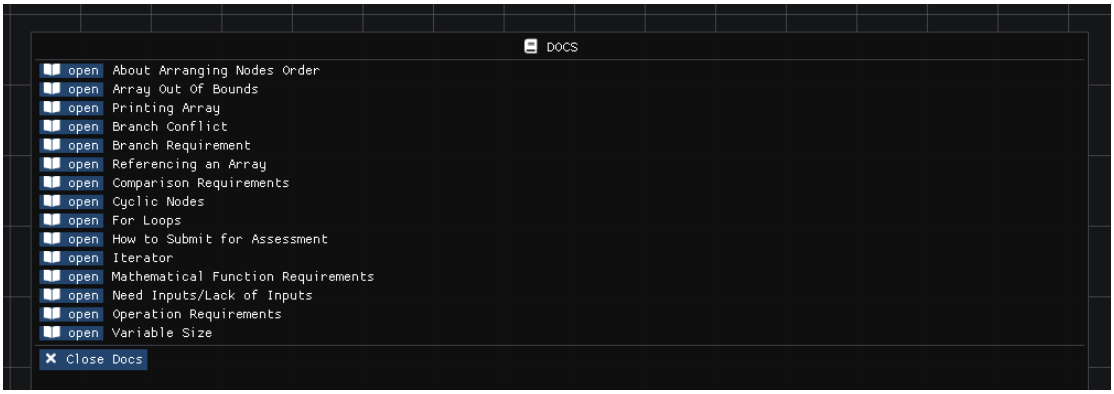


Figure 14. Screenshot of Docs in CodeNect

Figure 15 shows an example of a document in Docs window. This document provides explanation, code example, and key points about a particular topic.

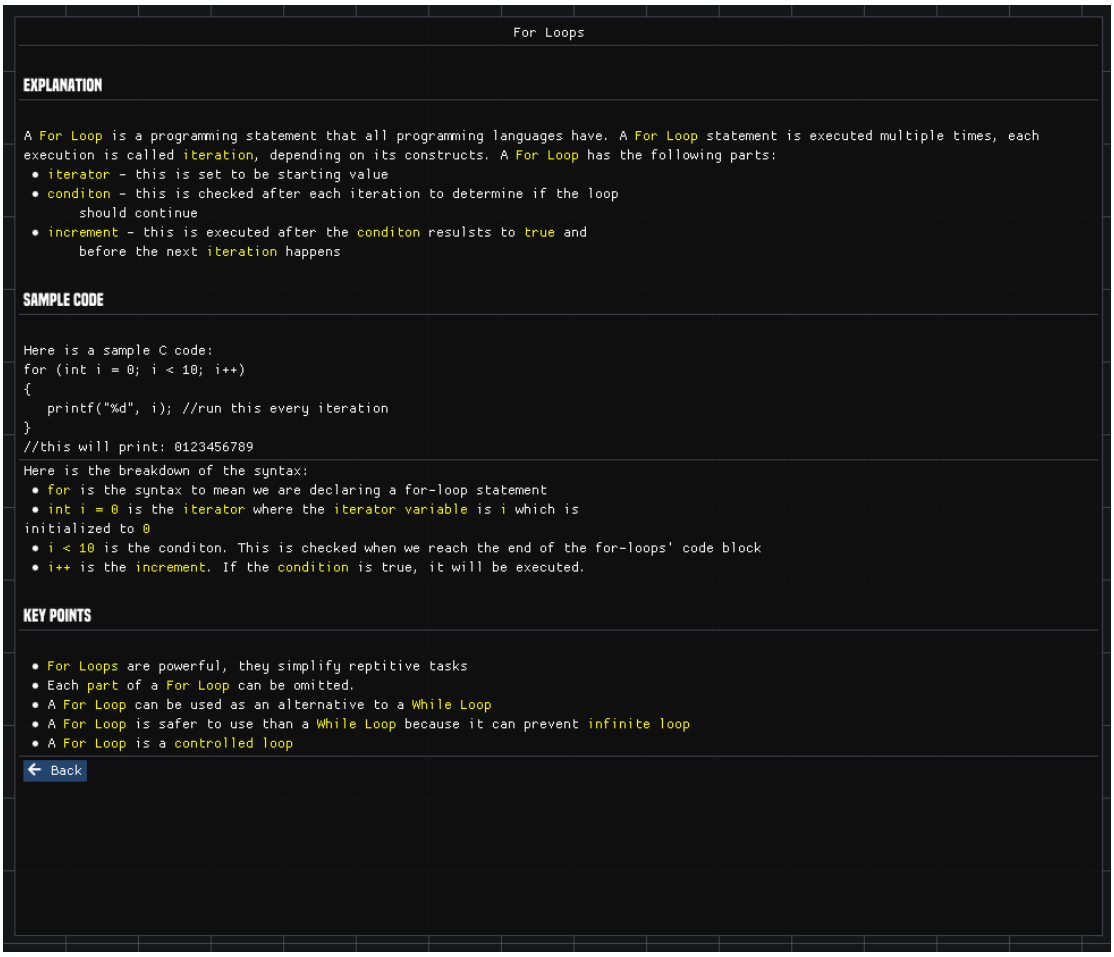


Figure 15. Screenshot of Sample Docs in CodeNect

Figure 16 shows the Assessments window for accessing the different built-in coding exercises for the users to try and solve. The Assessments window can be toggled open by the keyboard shortcut **ctrl + shift + a**.

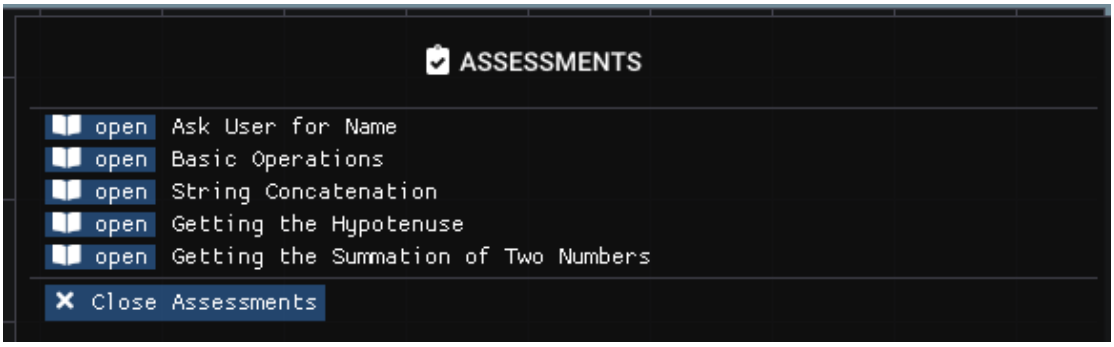


Figure 16. Screenshot of Assessments in CodeNect

Figure 17 shows an example coding exercise from the Assessments window. The assessment exercise shows step-by-step general instruction for what to do and shows the required output upon trying the assessment.

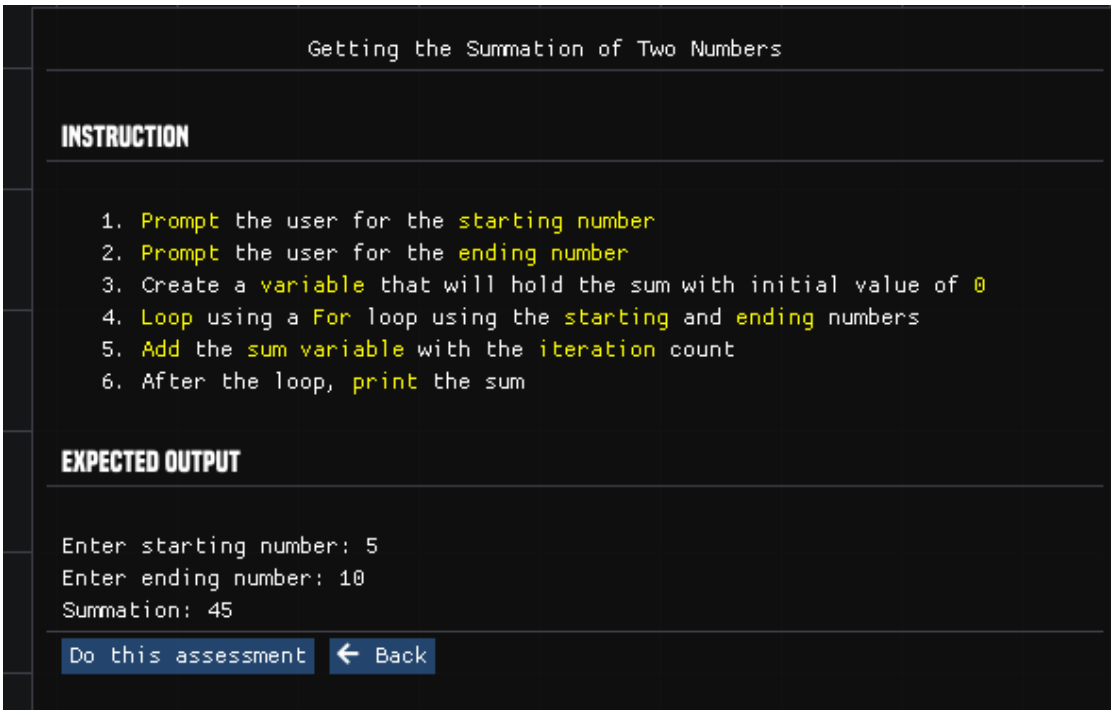


Figure 17. Screenshot of Sample Assessment in CodeNect

Figure 18 shows the output of a submitted visual code for an assessment problem. The output shows a score for each line that the user got correct. The user can also see the comparison line by line for easier checking of the result.

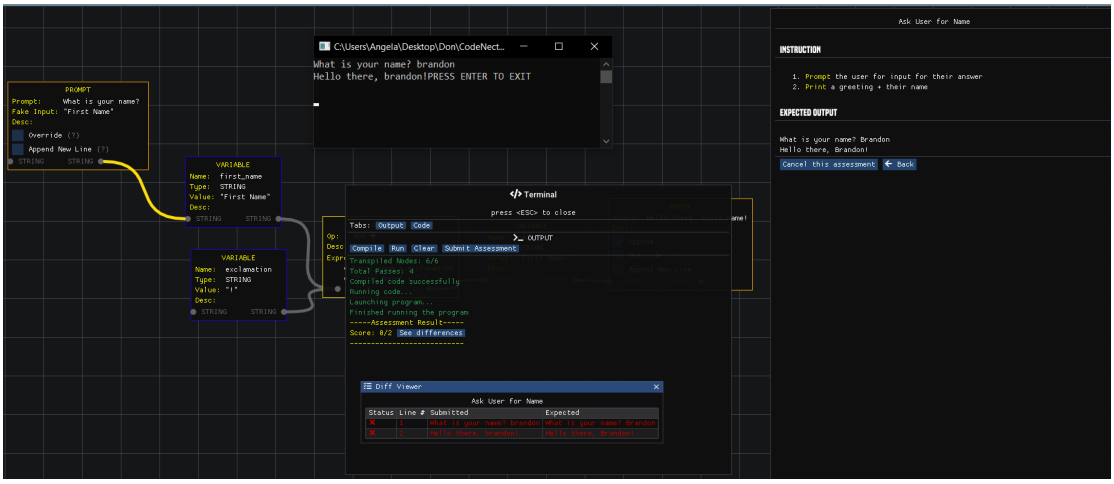


Figure 18. Screenshot of Sample Assessment Output in CodeNect

Figure 19 shows the Settings menu that allows the user to configure certain elements of the software such as: fullscreen, theme, font style and size, fade in/out duration for the sidebar, size of command palette, node text colors, and node frame padding.

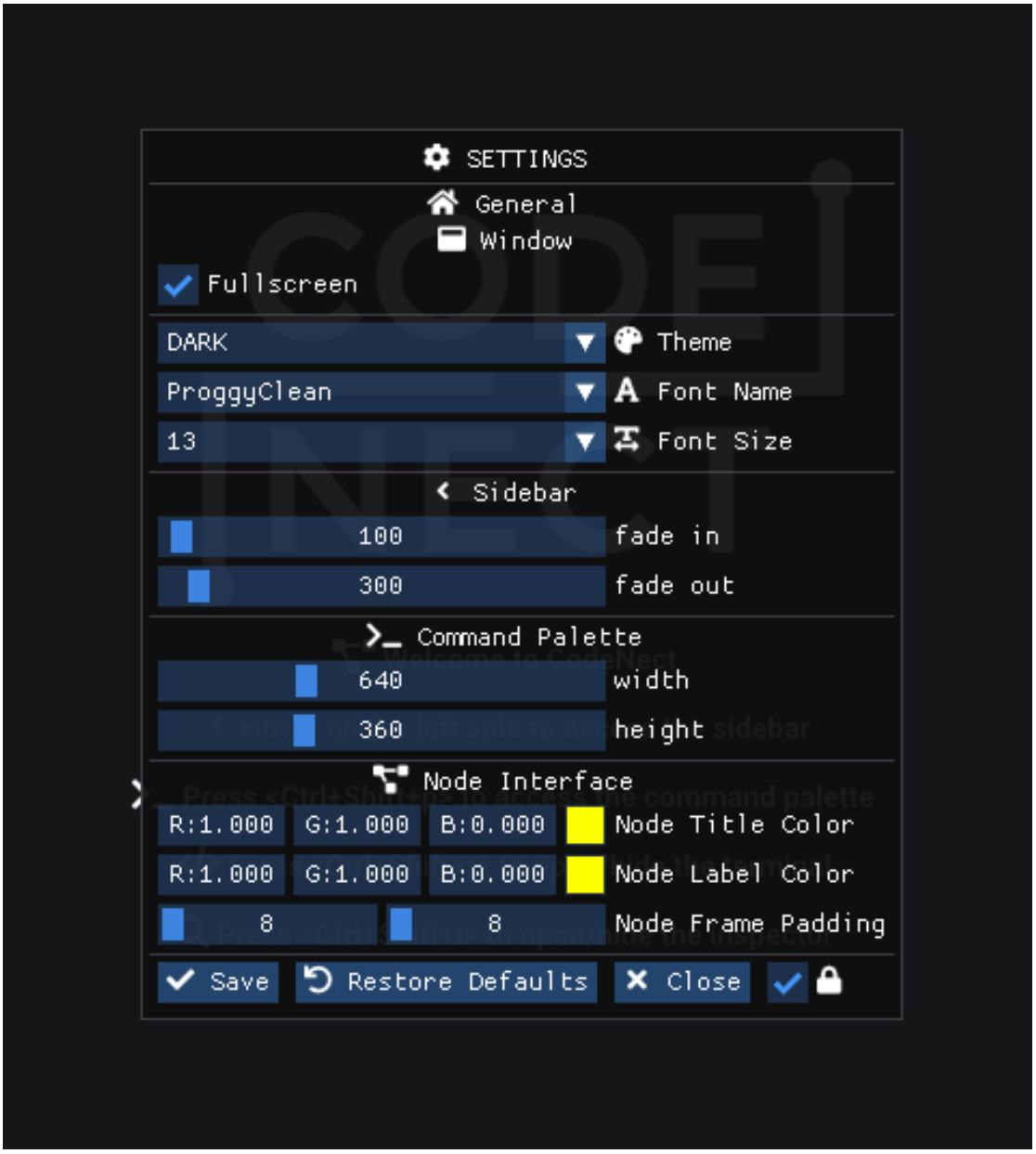


Figure 19. Screenshot of Settings Menu in CodeNect

Figure 20 shows the About window that shows information about the software such as the version number, developers and authors, credits, and special thanks.

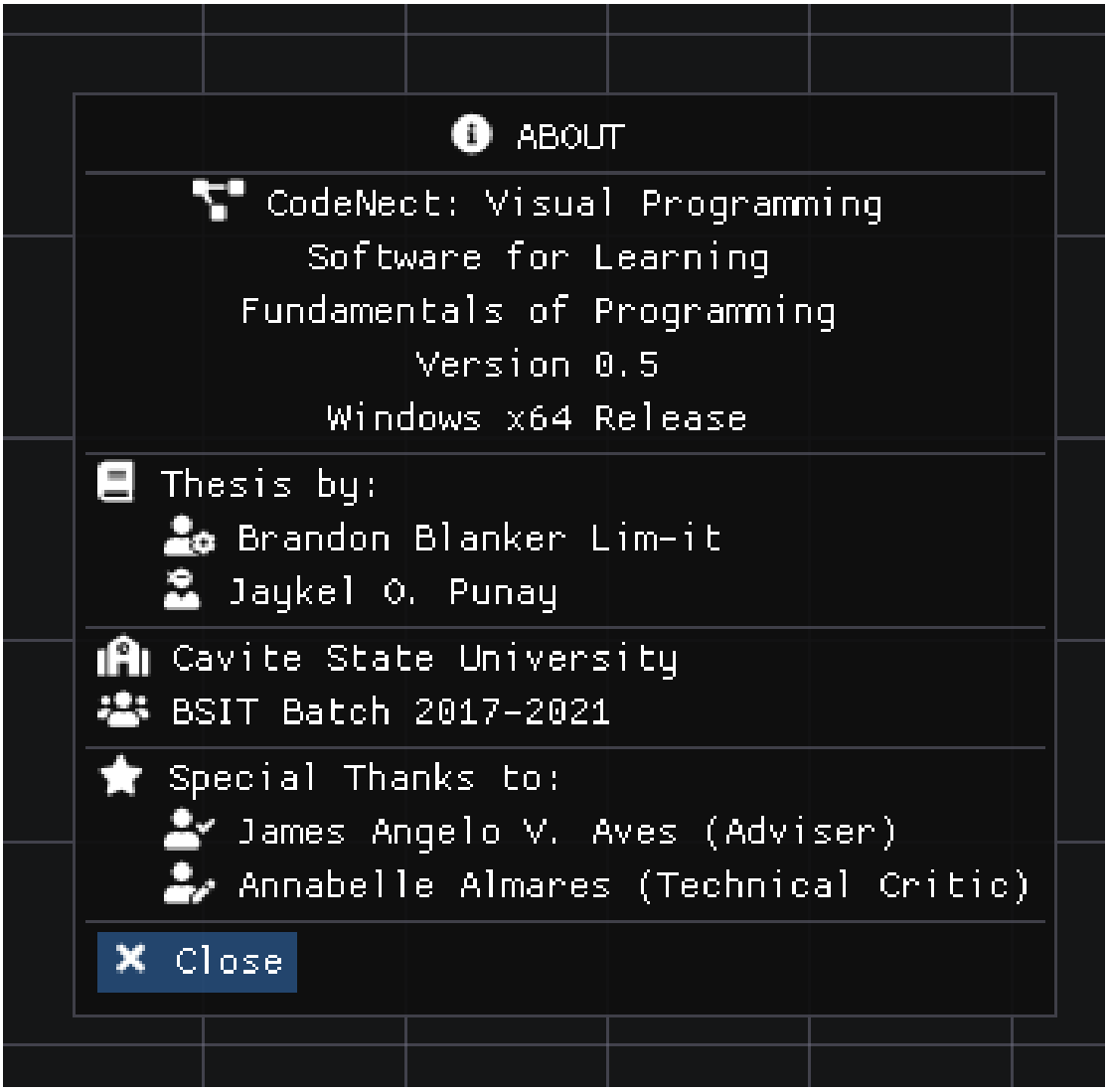


Figure 20. Screenshot of About Window in CodeNect

Figure 21 shows the Help and About window for listing brief information about the usage of the software. The window has the following category: commands or keyboard shortcuts (Figure 21), interfaces and color-coding, dictionary (Figure 22), libraries and tools used (Figure 23), and support and contacts (Figure 23).

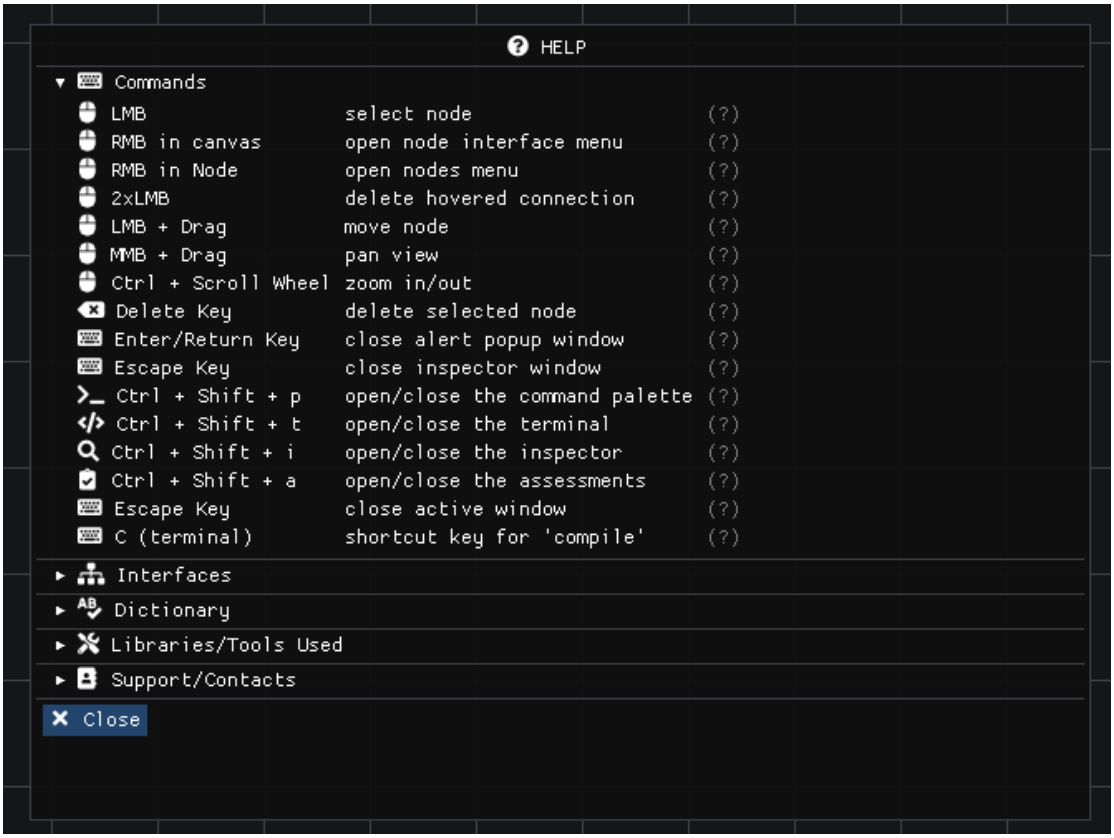


Figure 21. Screenshot of Keyboard Shortcuts Help in CodeNect

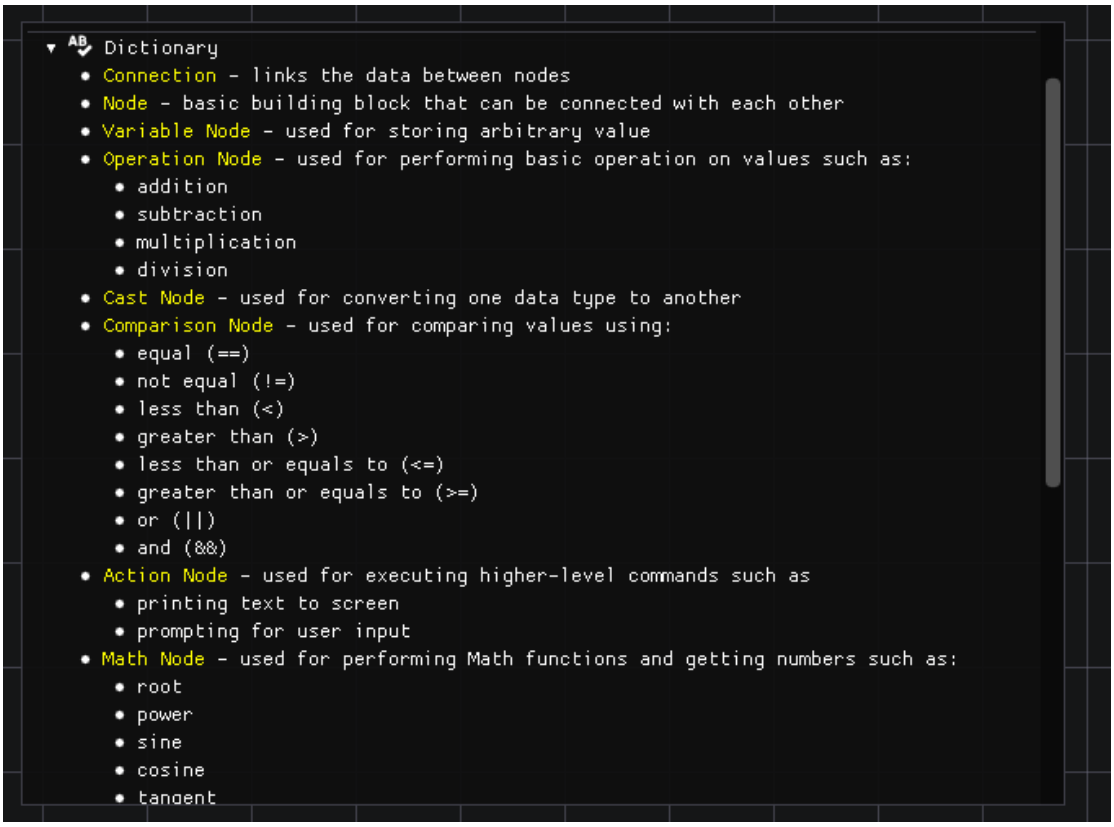


Figure 22. Screenshot of Dictionary Help in CodeNect

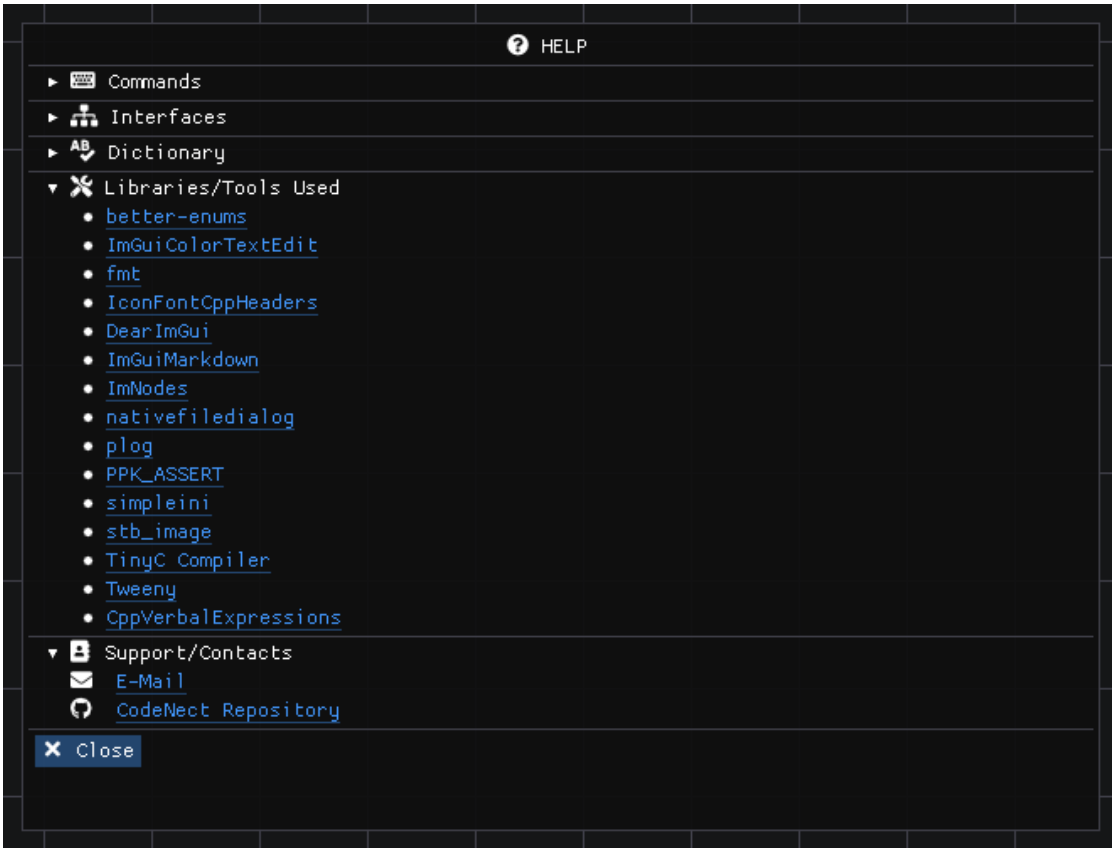


Figure 23. Screenshot of Libraries and Contacts Help in CodeNect

Software Evaluation

The functionalities of the software has been evaluated accordingly. The features and effectiveness of the software will be verified by ten college students taking programming subjects as the non-technical evaluators and another ten IT professionals as technical evaluators. Feedbacks and remarks are collected from the evaluators for further analysis by the researchers. The evaluators used a numerical rating with the interpretation shown in Table 1.

LIKERT SCALE	
RANGE	**INTEPRETATION**
4.21 - 5.00	Excellent
3.41 - 4.20	Very Good
2.61 - 3.40	Good
1.81 - 2.60	Fair
1.00 - 1.80	Poor

Table 1. Likert Scale for the System Evaluation

Technical Evaluation Results

The researchers gathered ten professionals in the field of Information Technology and Computer Science for the technical evaluation of the system on May 24 and 25 year 2021. The technical evaluators assessed the software according to the ISO 9126 metrics (See Appendix B) and provided feedback through the use of Google Form Sheet. The evaluation process was conducted online.

The rating used for the evaluation: 4.21 - 5.00 as Excellent which indicates that the software fully meets and far exceeds the most expectations and requirements. 3.41 - 4.20 as Very Good which indicates that the software fully meets and exceeds several expectations and requirements. 2.61 - 3.40 as Good which indicates that the system fully meets the requirements. 1.81 - 2.60 as Fair which indicates that the software lacks in meeting the expectations and requirements. 1.00 - 1.80 as Poor, which indicates that the software fails to meet the expectations and requirements.

The calculation of the evaluation results, Table 2 shows the functionality of the software was rated "excellent" (mean = 4.33 SD = 0.82 with criteria such as informative (mean = 4.00 SD = 1.05) accurate (mean = 4.40 SD = 0.70) and interoperability (mean = 4.60 SD = 0.70).

Table 2. Mean Score for the Functionality of the System (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Informative (The information is clear, concise and informative to the intended audience.)	4.00	1.05	very good
2. Accurate (The software provides accurate and correct data.)	4.40	0.70	excellent
3. Interoperability (The modules are interconnected to each other and functions as a whole.)	4.60	0.70	excellent
Average	4.33	0.82	excellent

The calculation of the evaluation results, Table 3 shows the reliability of the soft-

ware was rated "very good" (mean = 4.00 SD = 0.85) with criteria such as reliable (mean = 4.00, SD = 0.94) bug free (mean = 3.70 SD = 0.95) and standard equipment (mean = 4.30 SD = 0.67)

Table 3. Mean Score for the Reliability of the System (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Reliable (The software is reliable in normal use.)	4.00	0.94	very good
2. Bug free (Software is bug free.)	3.79	0.95	good
3. Standard Equipment (The system uses standard equipment that is reliable, widely available and applicable to a variety of users.)	4.30	0.67	excellent
Average	4.00	0.85	very good

The calculation of the evaluation results, Table 4 shows the usability of the software was rated "very good" (mean = 4.10 SD = 0.83) with criteria such as understandability (mean = 4.00, SD = 1.05) operability (mean = 4.20 SD = 0.63) and learnability (mean = 4.20 SD = 1.03).

Table 4. Mean Score for the Usability of the System (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Understandability (The software is easy to understand.)	4.00	1.05	very good
2. Operability (The software is easily operated by the intended user.)	4.20	0.63	very good
3. Learnability (The program is attractive and interesting; it motivates users to continue using the program.)	4.20	1.03	very good
Average	4.10	0.83	very good

The calculation of the evaluation results, Table 5 shows the effeciency of the software was rated "excellent" (mean = 4.40 SD = 0.77) with criteria such as special equipment (mean = 4.40, SD = 0.70), storage (mean = 4.40, SD = 0.84), and detection (mean = 4.40, SD = 0.84).

Table 5. Mean Score for the Efficiency of the System (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Special equipment (If the program requires special equipment, the requirements are minimal and clearly stated by the developer.)	4.40	0.70	excellent
2. Storage (The program doesnt consume large amount of memory that can slow down the processing of the system.)	4.40	0.84	excellent
3. Detection (The program can easily identify the cause of failure within the software.)	4.40	0.84	excellent
Average	4.40	0.77	excellent

The calculation of the evaluation results, Table 6 shows the maintability of the software was rated "excellent" (mean = 4.3, SD = 0.75) with criteria such as function (mean = 4.4, SD = 0.82), process (mean = 4.3, SD = 0.82), and test (mean = 4.3, SD = 0.67).

Table 6. Mean Score for the Maintainability of the System (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Function (The effort required to change the system functions is minimal.)	4.30	0.82	excellent

2. Process (The program is stable that if when something is changed, it will not affect the processing of the system.)	4.30	0.82	excellent
3. Test (The effort needed to test the system is minimal.)	4.30	0.67	excellent
Average	4.30	0.75	excellent

The calculation of the evaluation results, Table 7 shows the portability of the software was rated "excellent" (mean = 4.55 SD = 0.76) with criteria such as installation (mean = 4.70 SD = 0.67) and adaptability (mean = 4.4, SD = 0.84).

Table 7. Mean Score for the Portability of the System (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Installation (The effort required to install the system is minimal.)	4.70	0.67	excellent
2. Adaptability (The system has the ability to adapt to new specifications or operating environments.)	4.40	0.84	excellent
Average	4.55	0.76	excellent

The calculation of the evaluation results, Table 8 shows the user-friendliness of the software was rated "excellent" (mean = 4.43 SD = 0.67) with criteria such as clarity (mean = 4.1, SD = 0.88), objectivity of contents (mean = 4.9, SD = 0.32), and typographical accuracy (mean = 4.3, SD = 0.82).

Table 8. Mean Score for the User-Friendliness of the System (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Clarity of controls (Information about controls are understandable and available to the users.)	4.10	0.88	very good
2. Objectivity of contents (The language is non-discriminatory. Content is free from race, ethnic, gender, age and other stereotypes.)	4.90	0.32	excellent
3. Typographical Accuracy (The content is free from spelling and grammatical errors.)	4.30	0.82	excellent
Average	4.43	0.67	excellent

The calculation of the overall evaluation results, Table 9 shows the software was rated "excellent" (mean = 4.30, SD = 0.78) with criteria such as functionality (mean = 4.33 SD = 0.82), reliability of contents (mean = 4, SD = 0.85), usability of contents (mean = 4.1, SD = 0.83), effeciency of contents (mean = 4.4, SD = 0.77), maintability of contents (mean = 4.3, SD = 0.75), portability of contents (mean = 4.55, SD = 0.76), and user-friendliness (mean = 4.43, SD = 0.67).

Table 9. Overall Technical Evaluation Assessment of the Software (Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
Functionality	4.33	0.82	excellent
Reliability	4.00	0.85	very good
Usability	4.10	0.83	very good
Efficiency	4.40	0.77	excellent
Maintainability	4.30	0.75	excellent
Portability	4.55	0.76	excellent
User-friendliness	4.43	0.67	excellent

Average	4.30	0.78	excellent

Non-Technical Evaluation Results

The researchers gathered twelve students taking courses with programming subjects such as Information Technology and Computer Science for the non-technical evaluation of the system on May 24 and 25 year 2021. The non-technical evaluators assessed the software according to the ISO 9126 metrics (See Appendix B) and provided feedback through the use of Google Form Sheet. The evaluation process was conducted online.

The rating used for the evaluation: 4.21 - 5.00 as Excellent which indicates that the software fully meets and far exceeds the most expectations and requirements. 3.41 - 4.20 as Very Good which indicates that the software fully meets and exceeds several expectations and requirements. 2.61 - 3.40 as Good which indicates that the system fully meets the requirements. 1.81 - 2.60 as Fair which indicates that the software lacks in meeting the expectations and requirements. 1.00 - 1.80 as Poor, which indicates that the software fails to meet the expectations and requirements.

The calculation of the evaluation results, Table 10 shows the functionality of the software was rated "excellent" (mean = 4.69 SD = 0.52 with criteria such as informative (mean = 4.67 SD = 1.05) accurate (mean = 4.83 SD = 0.39) and interoperability (mean = 4.48 SD = 0.67).

Table 10. Mean Score for the Functionality of the System (Non-Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Informative (The information is clear, concise and informative to the intended audience.)	4.67	0.49	excellent
2. Accurate (The software provides accurate and correct data.)	4.83	0.39	excellent
3. Interoperability (The modules are interconnected to each other and functions as a whole.)	4.58	0.67	excellent
Average	4.69	0.52	excellent

The calculation of the evaluation results, Table 11 shows the reliability of the software was rated "very good" (mean = 4.41 SD = 0.67).

Table 11. Mean Score for the Reliability of the System (Non-Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Reliable (The software is reliable in normal use.)	4.41	0.67	excellent

The calculation of the evaluation results, Table 12 shows the usability of the software was rated "very good" (mean = 4.37 SD = 0.67) with criteria such as understandability (mean = 4.41, SD = 0.67), and learnability (mean = 4.33 SD = 0.78).

Table 12. Mean Score for the Usability of the System (Non-Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Understandability (The software is easy to understand.)	4.41	0.67	excellent
2. Learnability (The software is easily operated by the intended user.)	4.33	0.78	excellent
Average	4.37	0.73	excellent

The calculation of the evaluation results, Table 13 shows the user-friendliness of the software was rated "excellent" (mean = 4.50 SD = 0.6) with criteria such as clarity (mean = 4.25, SD = 0.62), objectivity of contents (mean = 4.83, SD = 0.39), and typographical accuracy (mean = 4.3, SD = 0.82).

Table 13. Mean Score for the User-Friendliness of the System (Non-Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
1. Clarity of controls (Information about controls are understandable and available to the users.)	4.25	0.62	excellent

2. Objectivity of contents (The language is non-discriminatory. Content is free from race, ethnic, gender, age and other stereotypes.)	4.83	0.39	excellent
3. Typographical Accuracy (The content is free from spelling and grammatical errors.)	4.3	0.82	excellent
Average	4.50	0.60	excellent

The calculation of the overall evaluation results, Table 14 shows the software was rated "excellent" (mean = 4.50, SD = 0.62) with criteria such as functionality (mean = 4.69 SD = 0.49), reliability of contents (mean = 4.41, SD = 0.67), usability of contents (mean = 4.37, SD = 0.73), and user-friendliness (mean = 4.50, SD = 0.60).

Table 14. Overall Technical Evaluation Assessment of the Software (Non-Technical)

INDICATOR	MEAN	STANDARD DEVIATION	INTERPRETATION
Functionality	4.69	0.49	excellent
Reliability	4.41	0.67	excellent
Usability	4.37	0.73	excellent
User-friendliness	4.50	0.60	excellent
Average	4.50	0.62	excellent

SUMMARY, CONCLUSION AND RECOMMENDATIONS

This chapter presents the the overview of the study, the presentation of the results, and the recommendations derived from the analysis and interpretation of the findings.

Summary

The study is the development of a visual programming software for aiding beginners in the field of programming get a better grasp and understanding about its fundamentals. The software "CodeNect: Visual Programming Software for Learning Fundamentals of Programming" was developed with the purpose of aiding beginners in learning the fundamentals of programming in the field of technology and to serve as an intermediary software in helping beginners get familiarized with programming.

The gathering of data needed for the studies was achieved through the conduction of survey, review of existing related studies and journals, and research on the Internet.

The software is composed of seven modules which are the Input/Output, Visual Nodes, Transpiler, Filesystem, Simulation, Debug, and Assessment modules. The software was developed using the C++ programming language and the usage of multiple open-source libraries for graphics and tools. TinyC Compiler was used for compiling and running the transpiled C code at runtime. Adobe Photoshop and Aseprite were used for the icons and logo.

The V-Model was used as methodology for the development of the software which has the following phases: requirements, system design, architecture design, module design, implementation and coding, and testing.

The software was evaluated of its features and functionalities using the ISO 9126. The remarks and verification by twelve students taking programming subjects as non-technical evaluators and another ten IT professionals as technical evaluators. The results of the evaluation shows that the software is "excellent" in overall both for the technical and non-technical evaluation.

Conclusion

The visual programming software is a stand-alone executable that can run without installation in Linux and Windows platforms. The problems the study aims to solve were identified and represented visually though Ishikawa Diagrams and Context Diagrams. The

software consists of seven modules: Input/Output module, Visual Nodes module, Transpiler module, Filesystem module, Simulation module, Debug module, and Assessment module. The software was evaluated using ISO 9126 and analyzed statistically.

The results of the technical evaluation of the study show that the visual programming software, CodeNect, is excellent in meeting most of the expectations and requirements except for the usability and reliability field as visual programming is relatively uncommon and rarely used so the evaluators had trouble figuring out the environment and the software itself.

The results of the non-technical evaluation of the study show that the visual programming software, CodeNect, is excellent in meeting and exceeding the expectations and requirements for non-technical evaluators.

The software serves also as an alternative approach to learning programming by providing visual elements compared to traditional text-based programming. With that said, the software, or visual programming in itself, does not try to compete with the power and effectiveness of using and learning text-based programming. Students are encouraged to continue learning programming after understanding the basics and fundamentals of programming.

The visual programming software was evaluated using the ISO 9126 standards with the criteria for quality including functionality, reliability, usability, maintanability, efficiency, and portability. The software passed the criteria for evaluation and met all the requirements and objectives having an overall mean of 4.30.

Recommendations

The researchers recommend the following:

1. Add more target programming language for the visual code to be transpiled
2. Increase the number of available nodes that can be used
3. Expand the number of assessment exercises
4. Expand the number of documents for guidelines and solutions

REFERENCES

A brief description of c++. (2021). *cplusplus.com*. Retrieved from https://www.cplusplus.com/info/description/

Abe, K., Fukawa, Y., & Tanaka, T. (2019). Prototype of visual programming environment for c language novice programmer.

Abraham, R., Burnett, M., & Erwig, M. (2009). Spreadsheet programming.

Adelson, B., & Soloway, E. (1985). The role of domain expenence in software design.

Alam, A., & Bush, V. (2016). Haskeu: An editor to support visual and textual programming in tandem.

Aleryani, A. (2016). Comparative study between data flow diagram and use case diagram.

Anggrawan, A., Ibrahim, N., M., S., & Satria, C. (2016). Influence of blended learning on learning result of algorithm and programming.

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From scratch to real programming.

Bau, D., Gray, J., Kelleher, C., Sheldon, J., & Turbak, F. (2017). Learnable programming.

Carter, J., & Jenkins, T. (1999). Gender and programming: What's going on?

Chang Boon Lee, P. (2011). Toward intuitive programming languages.

Cheng, E. (2016). Learning through the variation theory: A case study.

Chu, E., & Zaman, L. (2021). Exploring alternatives with unreal engines blueprints visual scripting system. *Entertainment Computing*.

Coccia, M. (2017). The fishbone diagram to identify, systematize and analyze the sources of general purpose technologies.

Corney, M., Lister, R., & Teague, D. (2011). Early relational reasoning and the novice programmer: Swapping as the "hello world" of relational reasoning.

Craft.ai, R. D. (2015). The maturity of visual programming. Retrieved from https://www.craft.ai/blog/the-maturity-of-visual-programming

DiCerbo, K. E. (2016). Assessment of task persistence. *Handbook of Research on Technology Tools for Real-World Skill Development*.

DiSalvo, B. (2014). Graphical qualities of educational technology: Using drag-and-drop and text-based programs for introductory computer science.

Geraldi, J., & Lechter, T. (2012). Gantt charts revisited.

Guven, B., & Cabakcor, B. O. (2013). Factors influencing mathematical problem-solving achievement of seventh grade turkish students. *Learning and Individual Differences*.

Harrison, W. (2004). From the editor: The dangers of end-user programming.

Hughes, R. (2016). Artifacts for the enterprise requirements value chain.

Israel-Fishelson, R., & Hershkovitz, A. (2020). Persistence in a game-based learning environment: The case of elementary school students learning computational thinking. *Journal of Educational Computing Research*.

Joo, H. (2017). A study on understanding of ui and ux, and understanding of design according to user interface change.

Kaczmarczyk, L. C., Petrick, E. R., East, J. P., & Herman, G. L. (2010). Identifying student misconceptions of programming.

Ko, A. J., Myers, B. A., & Aung, H. H. (2004). Six learning barriers in end-user programming systems.

Linn, M. C., & Dalbey, J. (1985). Cognitive consequences of programming instruction: Instruction, access, and ability.

Lister, R. (2004). Teaching java first: Experiments with a pigs-early pedagogy.

Lister, R., Clear, T., Simon, Bouvier, D. J., Carter, P., Eckerdal, A., ... Robbins, P. e. a. (2009). Naturally occurring data as research instrument.

Liu, Y., Wu, L., & Dong, X. (2010). Research on controls-based visual programming.

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B.-D., ... Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year cs students.

Norman, D. (2013). *The design of everyday things*.

Ottosson, S., & Zaslavskyi, V. (2019). Visualize what to be coded before programming.

Ousterhout, J. (1998). Scripting: Higher level programming for the 21st century.

Prahofer, H., Hurnaus, D., Wirth, C., & Mossenbock, H. (2007). The domain-specific language monaco and its visual interactive programming environment.

Prokhorov, V., & Kosarev, V. (1999). Environment pij for visual programming in java.

Raiyn, J. (2020). The role of visual learning in improving students' high-order thinking skills.

Raja, R., & Nagasubramani, P. C. (2018). Impact of modern technology in education.

Repenning, A. (2017). Moving beyond syntax: Lessons from 20 years of blocks programming in agentsheets.

Robins, A., Rountree, J., & Rountree, N. (2003). Learning and teaching programming: A review and discussion.

Rook, P. (1986). Controlling software projects.

Saito, D., Washizaki, H., & Fukazawa, Y. (2017). Analysis of the learning effects between text-based and visual-based beginner programming environments.

Sorva, J. (2012). Visual program simulation in introductory programming education.

Stroustrup, B. (1999). An overview of the c++programming language. *An Overview of the C++Programming Language*.

Sukoco, Agus and Marzuki and Cucus, Ahmad. (2012). Concept of quality measurement system software based on standard ISO 9126 and ISO 19011.

Tan, P., Ting, C., & Ling, S. (2009). Learning difficulties in programming courses: Undergraduates' perspective and perception.

Thompson, D. V., Hamilton, R. W., & Rust, R. T. (2005). Feature fatigue: When product capabilities become too much of a good thing.

Tsai, C.-Y., Yang, Y.-F., & Chang, C.-K. (2015). Cognitive load comparison of traditional and distributed pair programming on visual programming language.

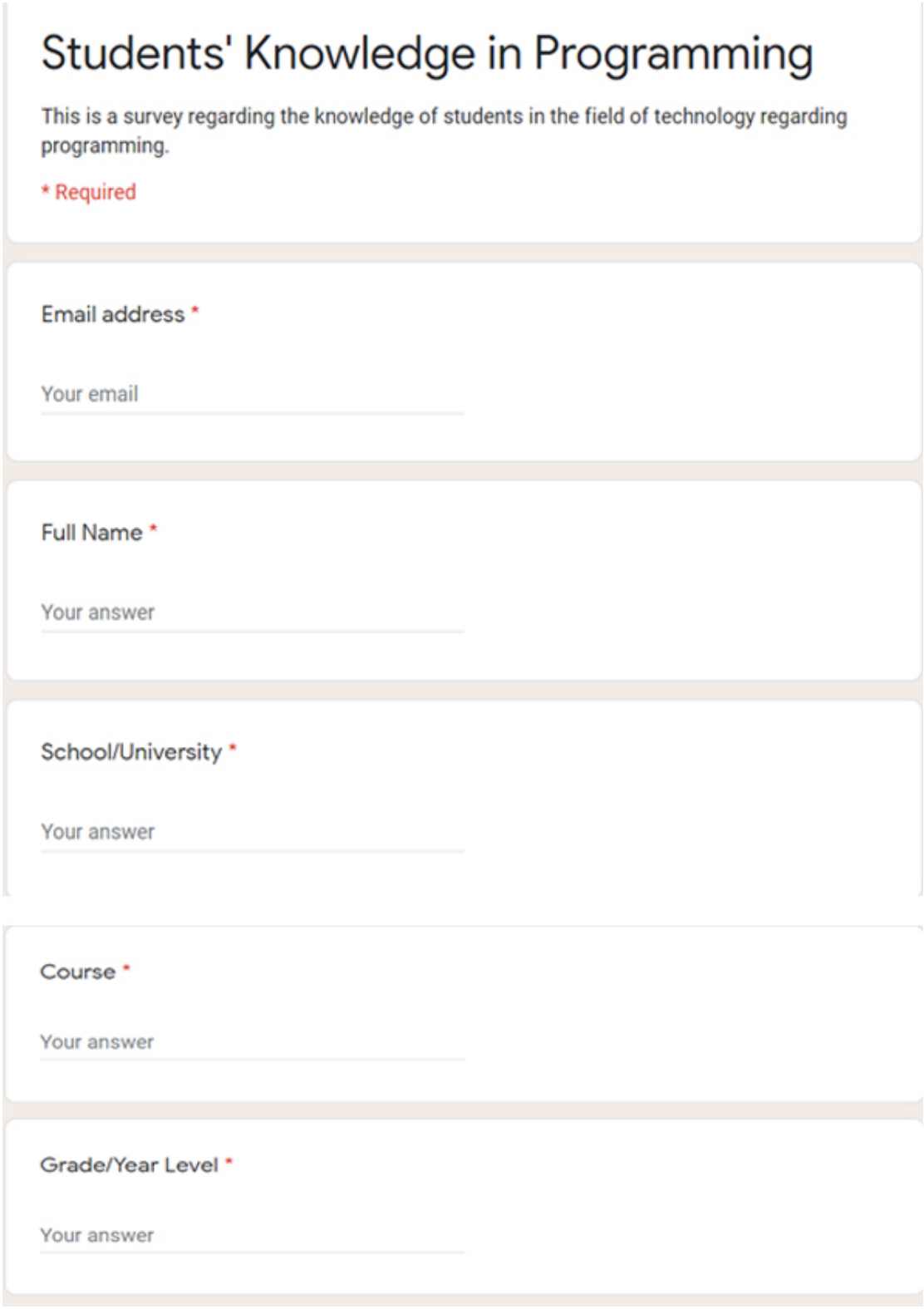
Tsukamoto, H., Takemura, Y., Oomori, Y., Ikeda, I., Nagumo, H., Monden, A., & Matsumoto, K.-i. (2016). Textual vs. visual programming languages in programming education for primary schoolchildren.

Williams, C., Alafghani, E., Daley, A., Gregory, K., & Rydzewski, M. (2015). Teaching programming concepts to elementary students.

Zhang, D.-Q., & Zhang, K. (n.d.). On the design of a generic visual programming environment.

APPENDICES FIGURES

Appendix A. Survey and Assessment Form



Students' Knowledge in Programming

This is a survey regarding the knowledge of students in the field of technology regarding programming.

* Required

Email address *

Your email

Full Name *

Your answer

School/University *

Your answer

Course *

Your answer

Grade/Year Level *

Your answer

What field in technology do you desire to work in? *

- [ ] Game Development
- [ ] Website Development
- [ ] Database Administrator
- [ ] Software Engineer
- [ ] Other: _____

Do you know how to program before you entered college level? *

- ◯ Yes
- ◯ No

Did you know that the course you've taken has programming subjects? *

- ◯ Yes
- ◯ No

How many years have you been programming? *

Your answer

What are the programming languages you are comfortable with? *

- [ ] C/C++
- [ ] Java
- [ ] C#
- [ ] Python
- [ ] Javascript
- [ ] Lua
- [ ] Other: _____

In what method of education do you learn the best/most? *

- [ ] Classroom setting
- [ ] Watching video tutorials
- [ ] Reading tutorials
- [ ] Modifying someone else's code
- [ ] One-on-one sessions
- [ ] Other: _____

What do you find hard to learn about programming ? *

- [ ] Concepts
- [ ] Syntax
- [ ] Terminologies
- [ ] Code editor/Integrated Development Environment (IDE)
- [ ] Data structures
- [ ] Algorithm
- [ ] Data types
- [ ] Conditionals
- [ ] Other: _____

Which alternative method do you prefer to use in learning programming? *

- ◯ Visual Interactive Environment

- ◯ Block-based programming
- ◯ Other:

Rate your knowledge about Data Types *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Memory *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Debugging *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Algorithm *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Basic Quiz

To fully evaluate your knowledge about programming, please answer the following if you can.

Rate your knowledge about Data Types *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Memory *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Debugging *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Rate your knowledge about Algorithm *

	1	2	3	4	5	
lowest	○	○	○	○	○	highest

Basic Quiz

To fully evaluate your knowledge about programming, please answer the following if you can.

Please choose the correct data type for the following value * 5 points

	Double or Float	Integer	Character	String	Boolean
10.0f	○	○	○	○	○
20	○	○	○	○	○
'c'	○	○	○	○	○
"Hello World"	○	○	○	○	○
true	○	○	○	○	○

Consider the code below

```
int a = 10;
int b = 20;
```

evaluate: (a < b) * 1 point

- ○ true
- ○ false

evaluate: (a != b) * 1 point

- ○ true
- ○ false

evaluate: (! (a < b)) * 1 point

- ◯ true
- ◯ false

evaluate: (a == (b - 10)) * 1 point

- ◯ true
- ◯ false

Consider the code below:

```
1   int a = 20;
2   int *b = &a;
3   (*b)++;
```

What is the value of b in line number 2? * 1 point

- ◯ 20
- ◯ 21
- ◯ a

What is the value of b in line number 3? * 1 point

- ◯ 20
- ◯ 21
- ◯ 22

What is the final value of a? * 1 point

- ◯ 20
- ◯ 21
- ◯ 22
- ◯ b

Consider the code below:

```
int c[] = {1, 2, 3, 4, 5};
```

What is the value of c[1]? * 1 point

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 0

What is the value of c[5]? *

- ◯ 5
- ◯ Out of bounds
- ◯ 6
- ◯ 0

Consider the code below:

```
int a = 5;
int b = 7;

if (a < b)
   a -= b;
else
   b += a;

a++;
b++;
```

What is the final value of a? * 1 point

- ◯ 0
- ◯ -2
- ◯ -1
- ◯ 1
- ◯ 2

What is the final value of b? * 1 point

- ◯ 8
- ◯ 7
- ◯ 6
- ◯ 9

Consider the code below:

```
int a = 5;
int b = 20;

for (int i = a; i < b; i += 2)
{
  a++;
  b--;
}
```

What is the final value of a? * 1 point

Your answer

What is the final value of b? * 1 point

Your answer

Laboratory Assessment

Do you have different instructors/teachers for laboratory and lecture? *

- ◯ Yes
- ◯ No

What software do you use for programming during your laboratory subject? *

- ☐ TurboC\C++
- ☐ DevC++
- ☐ Netbeans
- ☐ Eclipse
- ☐ VS Code
- ☐ Codeblocks
- ☐ Notepad
- ☐ Other:

Do you find the softwares used in your laboratory for programming difficult to use? *

- ◯ Yes
- ◯ No

What are the things you find difficult in using softwares in your laboratory for programming? *

- ☐ Hard to compile, build, and run
- ☐ Messy/Cluttered interface - too many elements are in the screen
- ☐ Error/warning messages are unhelpful/complicated
- ☐ Old/Outdated design
- ☐ Other:

CodeNect: Visual Programming Software for Learning Fundamentals of Programming Survey Form

This is survey is for gathering of data for the development of a visual programming software for learning programming fundamentals.

This is a form about using text-based editors in programming. Examples are:
* Notepad++
* TurboC/C++
* Sublime Text
* DevC++
* Visual Code
* Codeblocks

* Required

Email address *

Your email

Full Name *

Your answer

School Name *

Your answer

Course *

- ◯ Bachelor of Science in Information Technology
- ◯ Bachelor of Science in Computer Science
- ◯ Bachelor of Science in Computer Engineering
- ◯ Other: _____

Grade/Year Level *

- ◯ 1st year
- ◯ 2nd year
- ◯ 3rd year
- ◯ 4th year
- ◯ Grade 11
- ◯ Grade 12

What text-based editing tool do you use for programming? *

- ☐ CodeBlocks
- ☐ Sublime Text
- ☐ Notepad++
- ☐ Visual Code
- ☐ DevC++
- ☐ TurboC/C++
- ☐ Other: _____

How did you learn how to use a text-based editor? *

- [ ] Instructor's demo
- [ ] Written lesson/lecture
- [ ] Self-learned
- [ ] Video tutorial
- [ ] Official documentation
- [ ] Experimentation/Playing around
- [ ] Other: _____

Rate your learning experience with using a text-based editor *

	1	2	3	4	5	
worst	○	○	○	○	○	best

How much do you prefer to use text-based editors for programming? *

	1	2	3	4	5	
not preferred	○	○	○	○	○	preferred

Rate the design/appearance of the text-based editors you use *

	1	2	3	4	5	
worst	○	○	○	○	○	best

Rate the debugging experience with using text-based editors. *

	1	2	3	4	5	
worst	○	○	○	○	○	best

Rate the coding experience with using text-based editors. *

	1	2	3	4	5	
worst	○	○	○	○	○	best

How likely do you use the tools/features available in the text-based editors? *

	1	2	3	4	5	
not likely	○	○	○	○	○	most likely

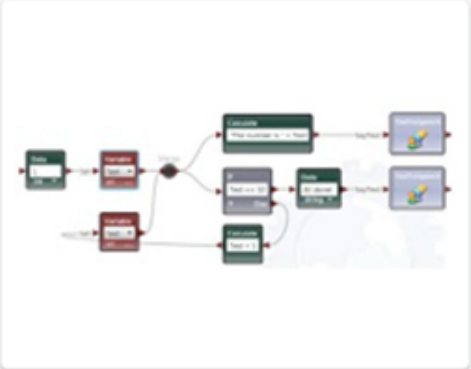
How much do you understand the tools/features in the text-based editors? *

	1	2	3	4	5	
do not understand at all	○	○	○	○	○	absolutely understand

Which do you prefer more to do programming? *

- ◯ Using keyboard to type to code
- ◯ Using mouse to interact/select to code
- ◯ Other: _____

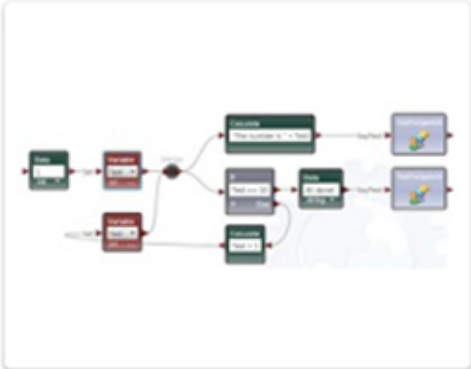
Which of the following do you prefer more as an alternative way to program and learn? *



◯ Node-based

◯ Block-based

Which of the two do you find easier to learn and use? *



◯ Node-based

◯ Block-based

Appendix B. Evaluation Form

Republic of the Philippines
CAVITE STATE UNIVERSITY
Don Severino delas Alas Campus
Indang, Cavite

COLLEGE OF ENGINEERING AND INFORMATION TECHNOLOGY
Department of Information Technology

CODENECT: VISUAL PROGRAMMING SOFTWARE FOR LEARNING FUNDAMENTALS OF PROGRAMMING

Proponents: Lim-it, Brandon B.
Punay, Jaykel O.

Instruction: Please kindly evaluate the software material by putting a checkmark (/) under the corresponding numerical rating. Use the legend as your guide.

Name (optional): _____ Course: _____ Year Level:_____ Date: _____

Legend: **5 –** Excellent **4** – Very Good **3** – Good **2** – Fair **1** – Poor

FUNCTIONALITY	**5**	**4**	**3**	**2**	**1**
The information is clear, concise and informative to the intended audience.					
The software provides accurate and correct data.					
The modules are interconnected to each other and functions as a whole.					
RELIABILITY					
The software is reliable in normal use.					
USABILITY					
The software is easy to understand					
The program is attractive and interesting; it motivates users to continue using the program and exploring career options.					
USER-FRIENDLINESS					
Information about controls are understandable and available to the users.					
The language is non-discriminatory. Content is free from race, ethnic, gender, age and other stereotypes.					
The content is free from spelling and grammatical errors.					

Suggestions:

_____
_____
_____

_____
Signature

Figure 24. Non-Technical Evaluation Form

Republic of the Philippines
CAVITE STATE UNIVERSITY
Don Severino delas Alas Campus
Indang, Cavite

COLLEGE OF ENGINEERING AND INFORMATION TECHNOLOGY
Department of Information Technology

CODENECT: VISUAL PROGRAMMING SOFTWARE FOR LEARNING FUNDAMENTALS OF PROGRAMMING

Proponents: Lim-it, Brandon B.
Punay, Jaykel O.

Instruction: Please kindly evaluate the software material by putting a checkmark (/) under the corresponding numerical rating. Use the legend as your guide.

Name (optional): _____ Date: _____

Legend: **5 –** Excellent **4** – Very Good **3** – Good **2** – Fair **1** – Poor

FUNCTIONALITY	**5**	**4**	**3**	**2**	**1**
The information is clear, concise and informative to the intended audience.					
The software provides accurate and correct data.					
The modules are interconnected to each other and functions as a whole.					
RELIABILITY					
The software is reliable in normal use.					
Software is bug free.					
The system uses standard equipment that is reliable, widely available and applicable to a variety of users.					
USABILITY					
The software is easy to understand					
The software is easily operated by the intended user.					
The program is attractive and interesting; it motivates users to continue using the program and exploring career options.					

Figure 25. Technical Evaluation Form

EFFICIENCY	5	4	3	2	1
If the program requires special equipment, the requirements are minimal and clearly stated by the developer.					
The program doesn't consume large amount of memory that can slow down the processing of the system.					
The program can easily identify the cause of failure within the software.					
MAINTAINABILITY					
The effort required to change the system functions is minimal.					
The program is stable that if when something is changed, it will not affect the processing of the system.					
The effort needed to test the system is minimal.					
PORTABILITY					
The effort required to install the system is minimal.					
The system has the ability to adapt to new specifications or operating environments.					
USER-FRIENDLINESS					
Information about controls are understandable and available to the users.					
The language is non-discriminatory. Content is free from race, ethnic, gender, age and other stereotypes.					
The content is free from spelling and grammatical errors.					

Suggestions:

_____

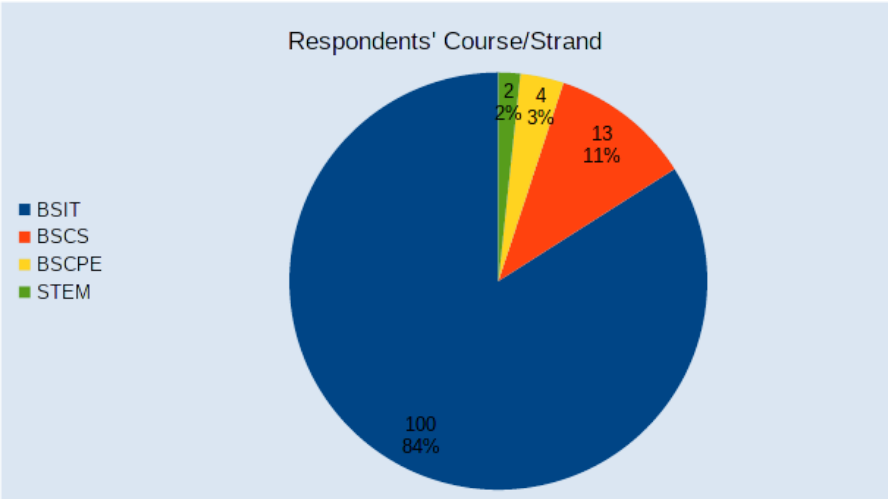
_____

_____

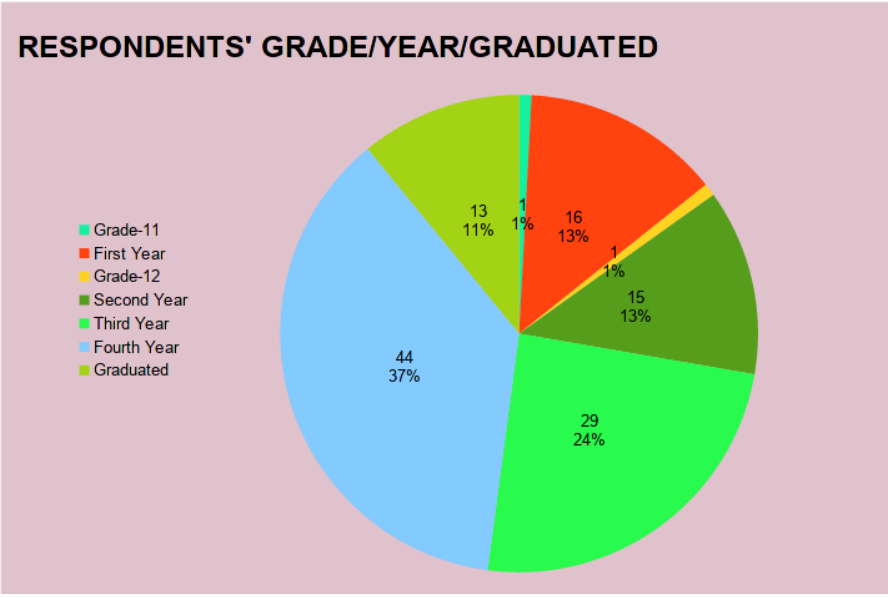
_____

_____

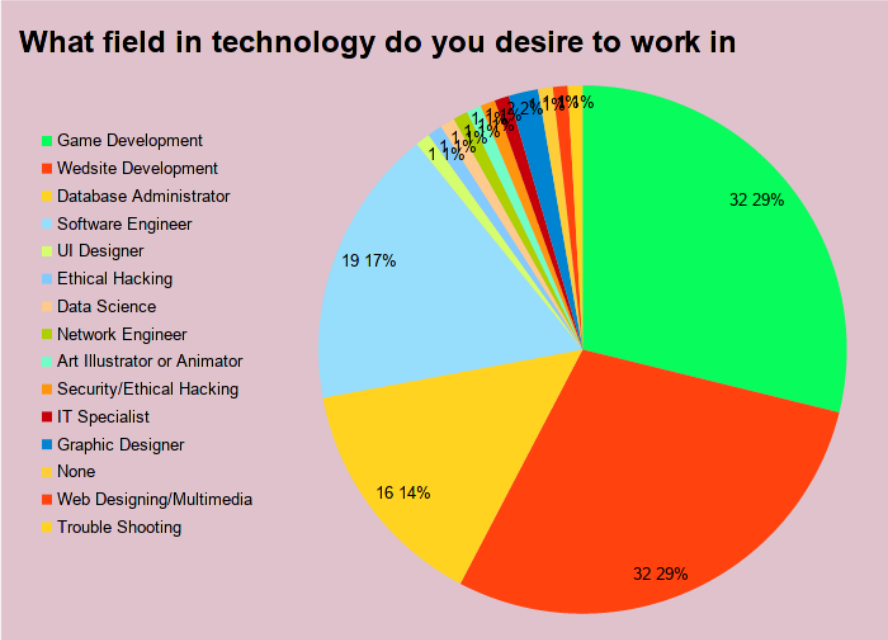
Signature



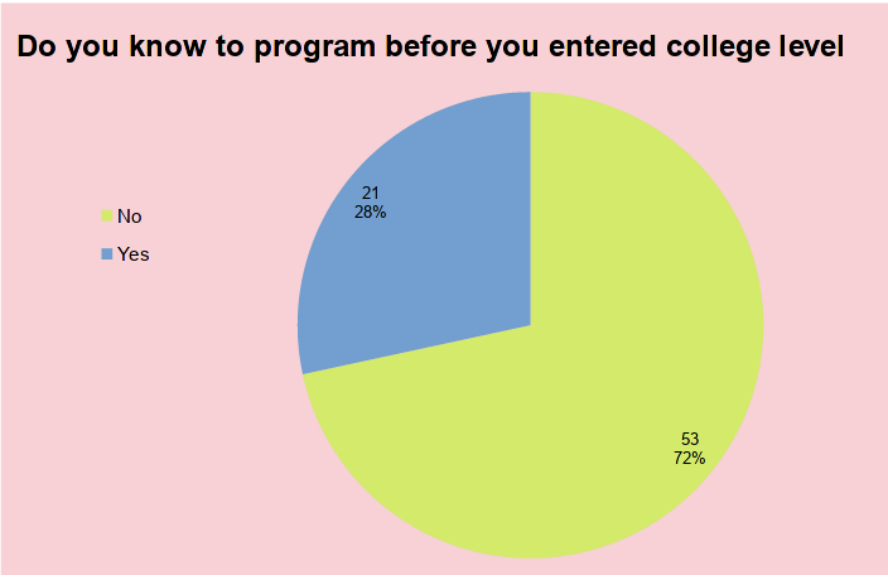
1. Graphical representation of the course and strand of the respondents



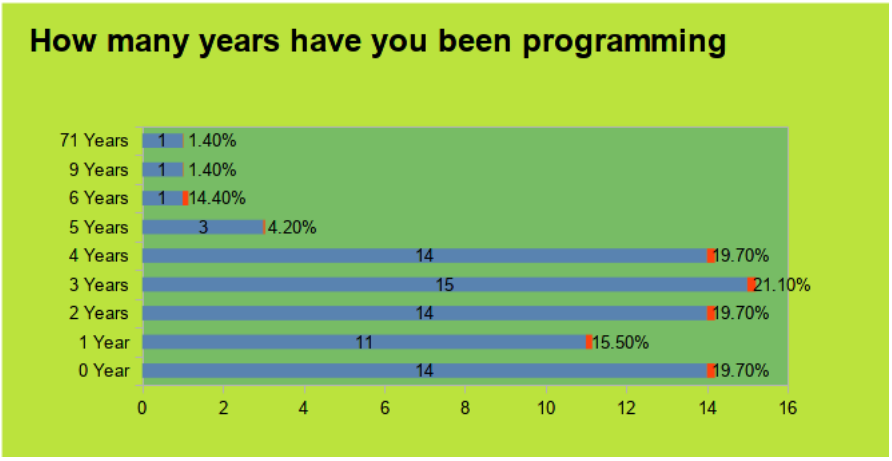
2. Graphical representation of the grade/year level of the respondents



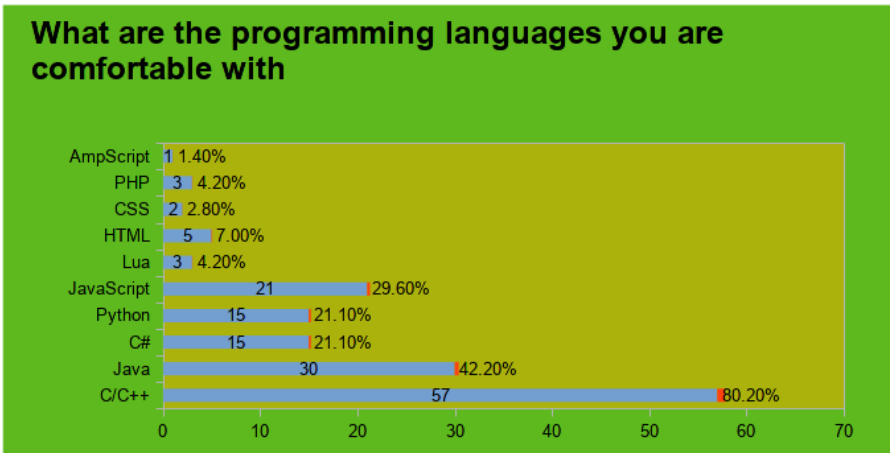
3. Graphical representation of the desired technology field of the respondents



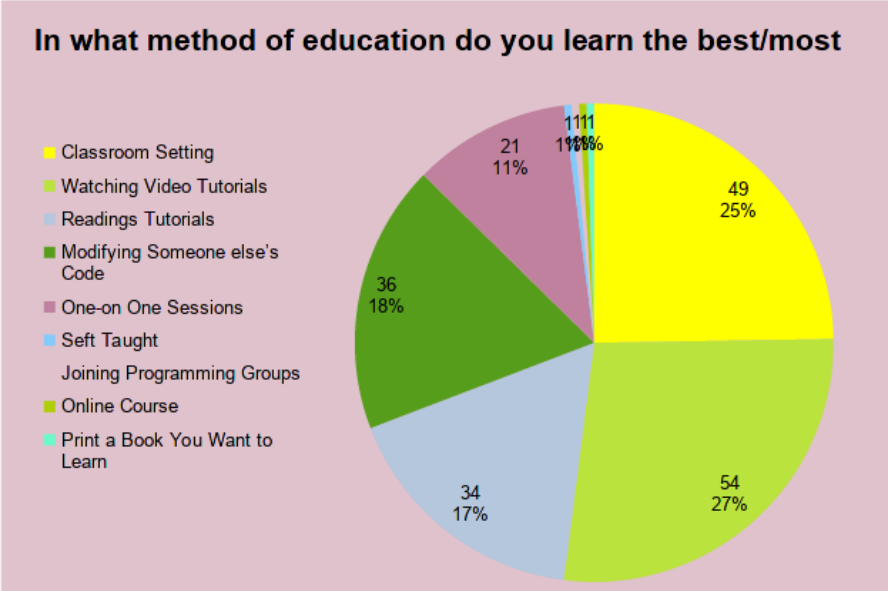
4. Graphical representation of knowing programming pre-college of the respondents



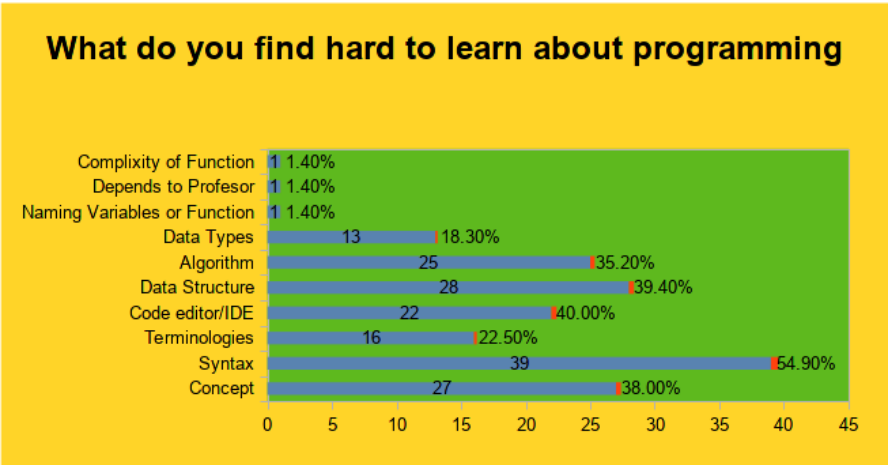
5. Graphical representation of the years programming of the respondents



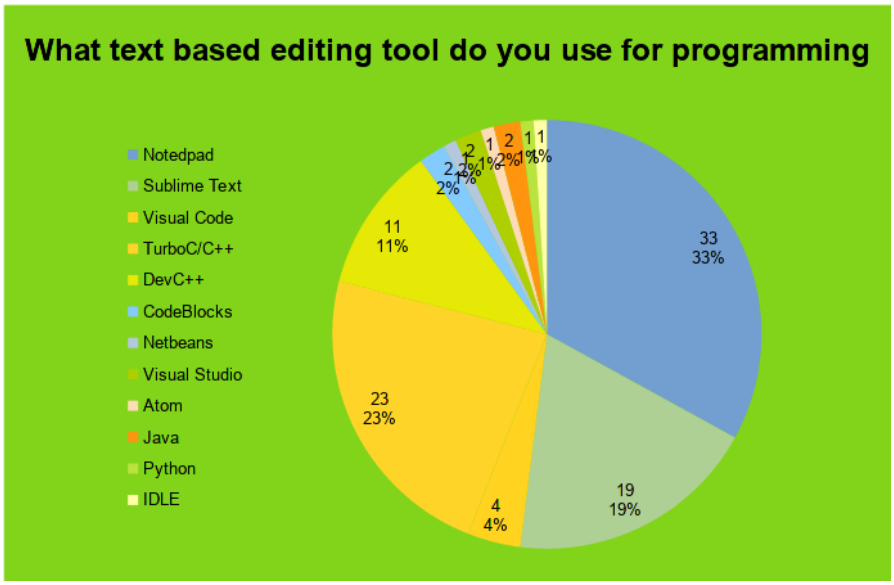
6. Graphical representation of the languages comfortable with of the respondents



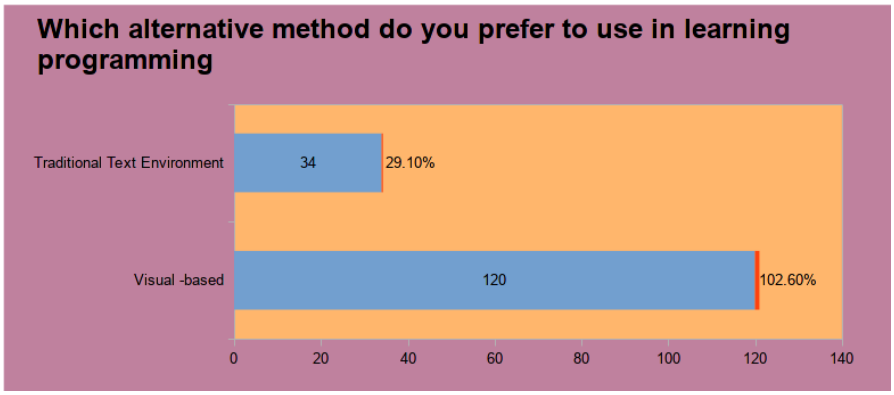
7. Graphical representation of the best learning method of the respondents



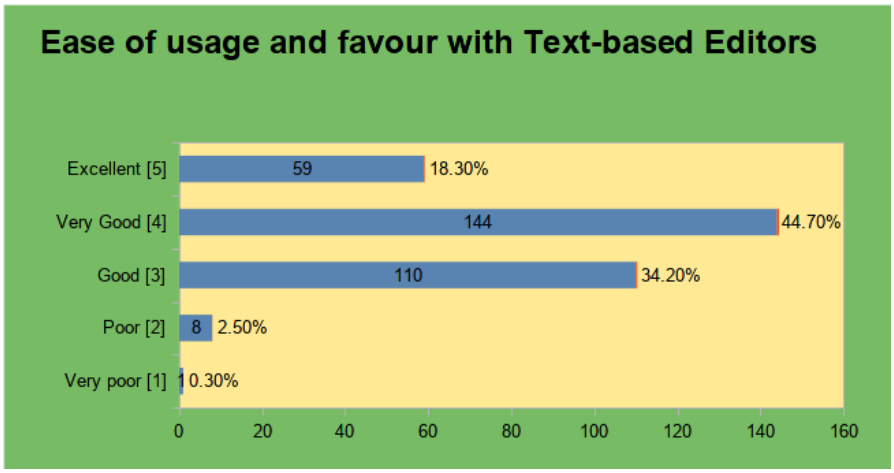
8. Graphical representation of the hard to learn in programming of the respondents



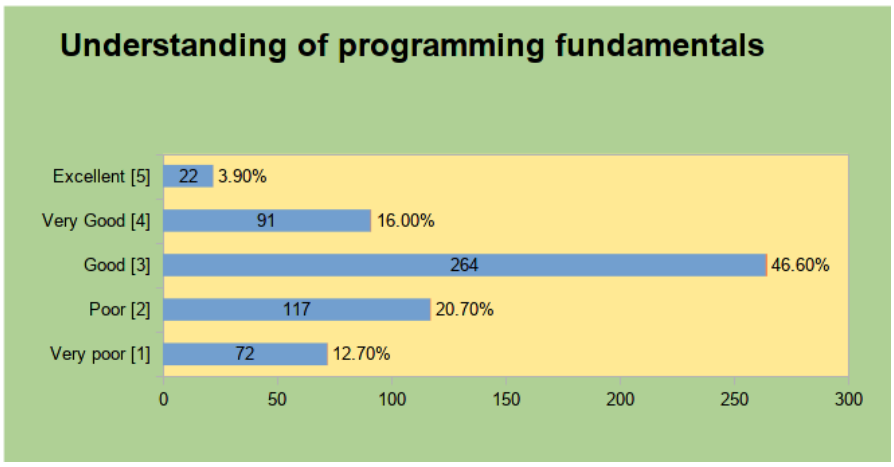
9. Graphical representation of the text-editing tool used of the respondents



10. Graphical representation of the preferred alternative learning method of the respondents



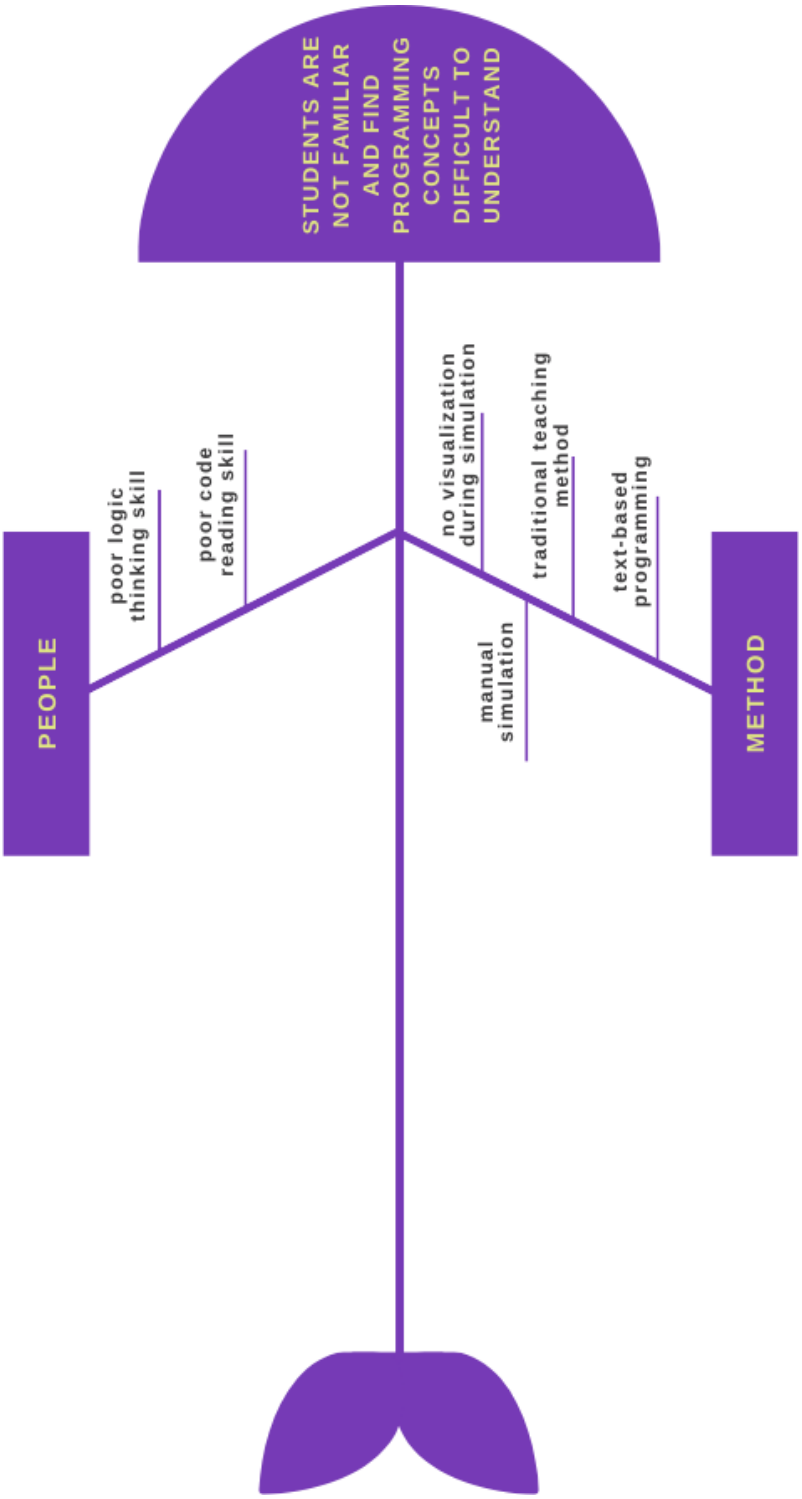
11. Graphical representation of the text editors' usage ease of the respondents



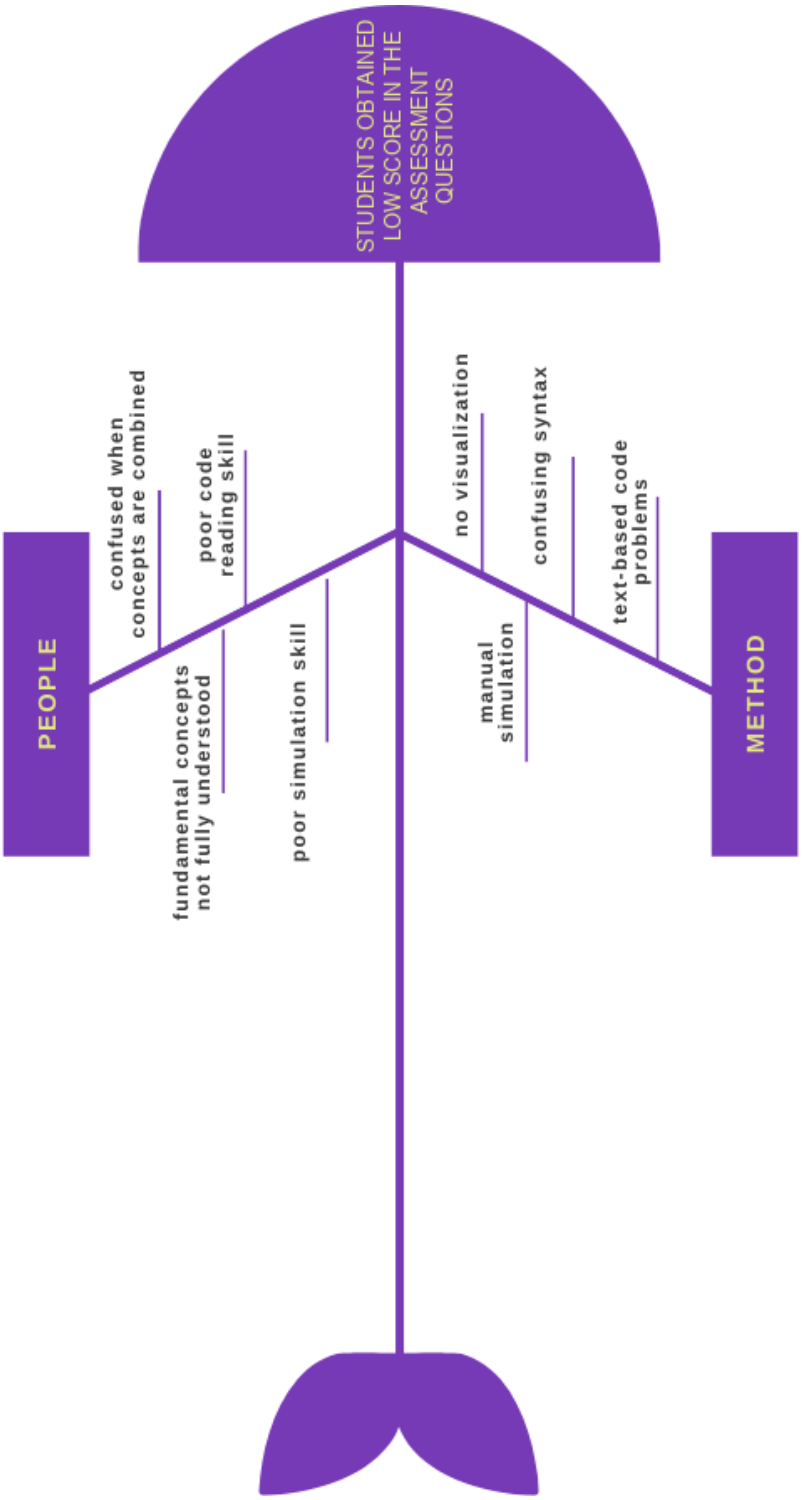
12. Graphical representation of the programming fundamentals understanding of the respondents

13. Graphical representation of the programming fundamentals assessment of the respondents

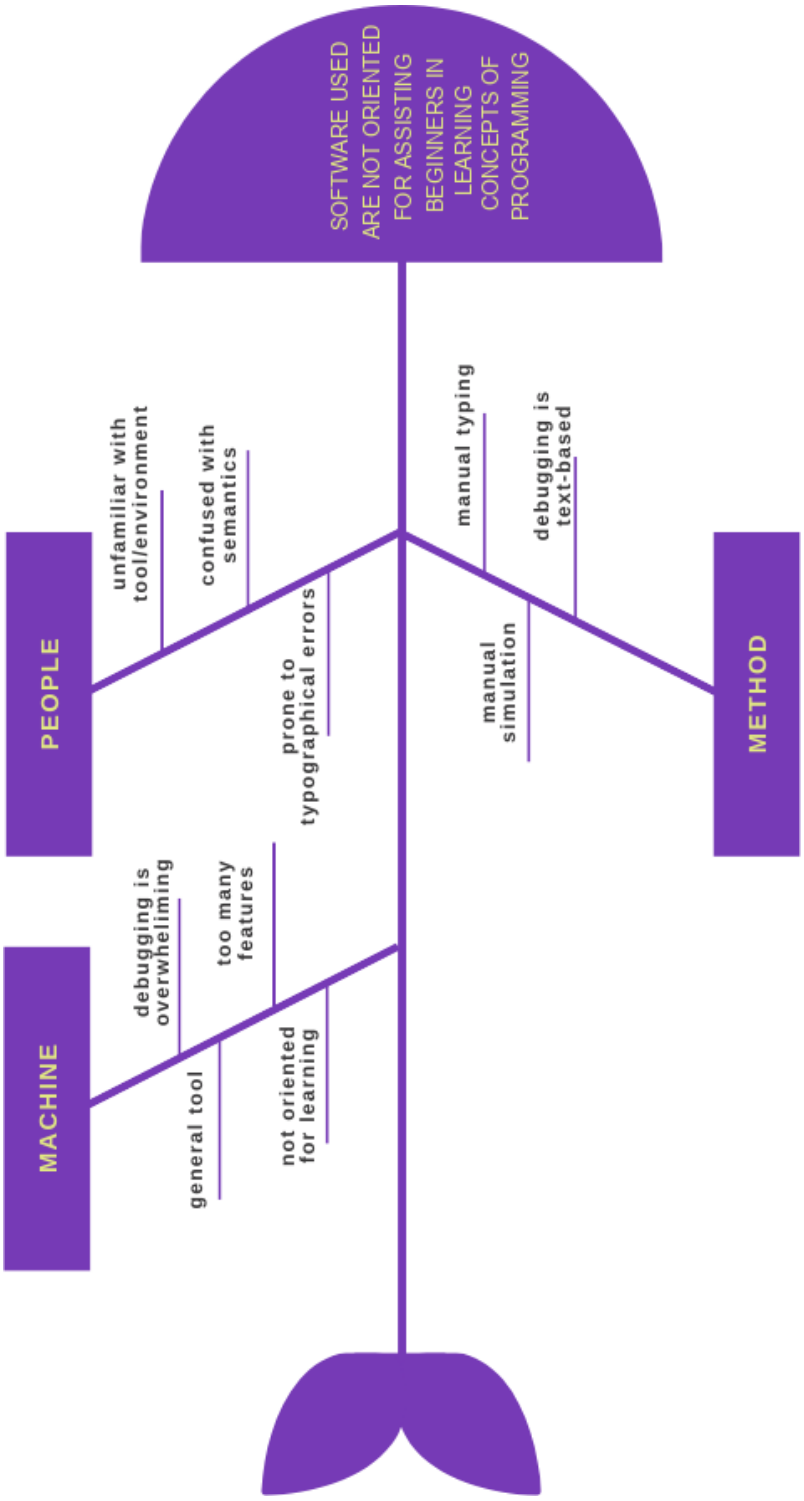
14. Ishikawa Diagram



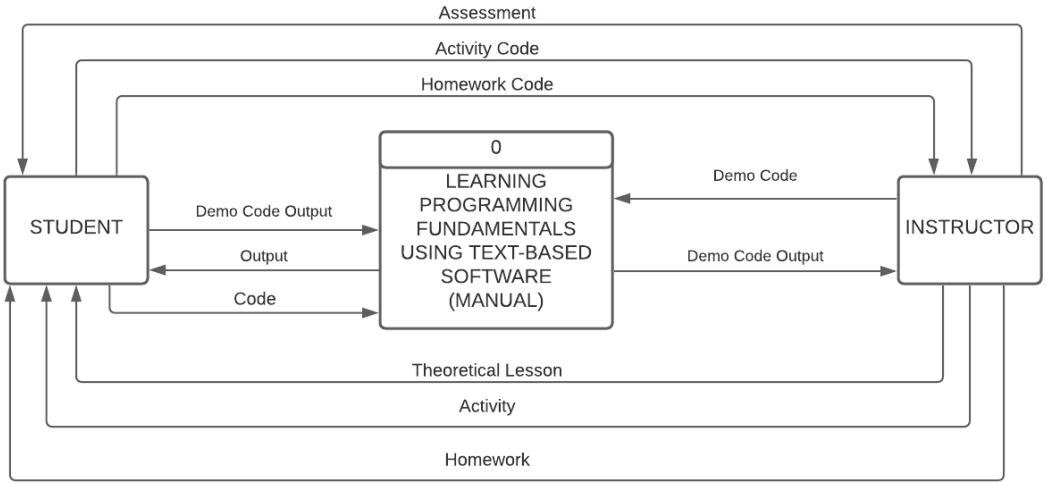
15. Fishbone diagram of the students not familiar and find programming concepts difficult to understand



16. Fishbone diagram of the students incorrect answer to programming assessment

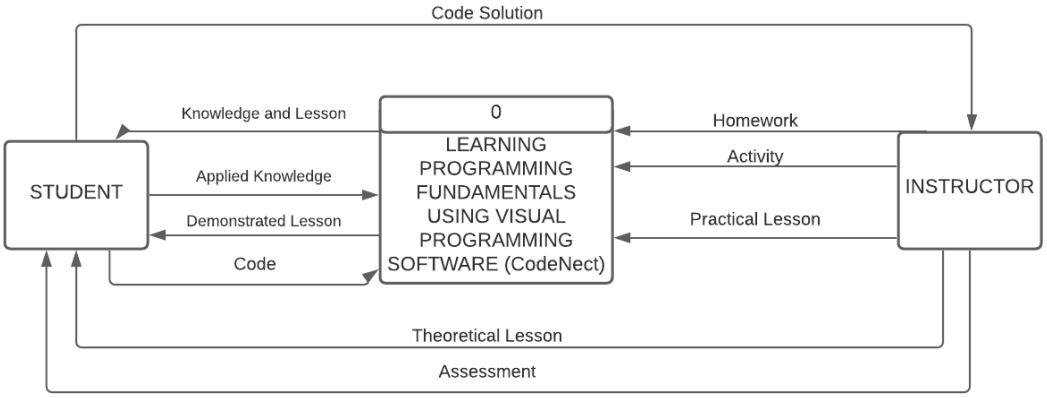


17. Fishbone diagram of the tool for programming not effective for learning



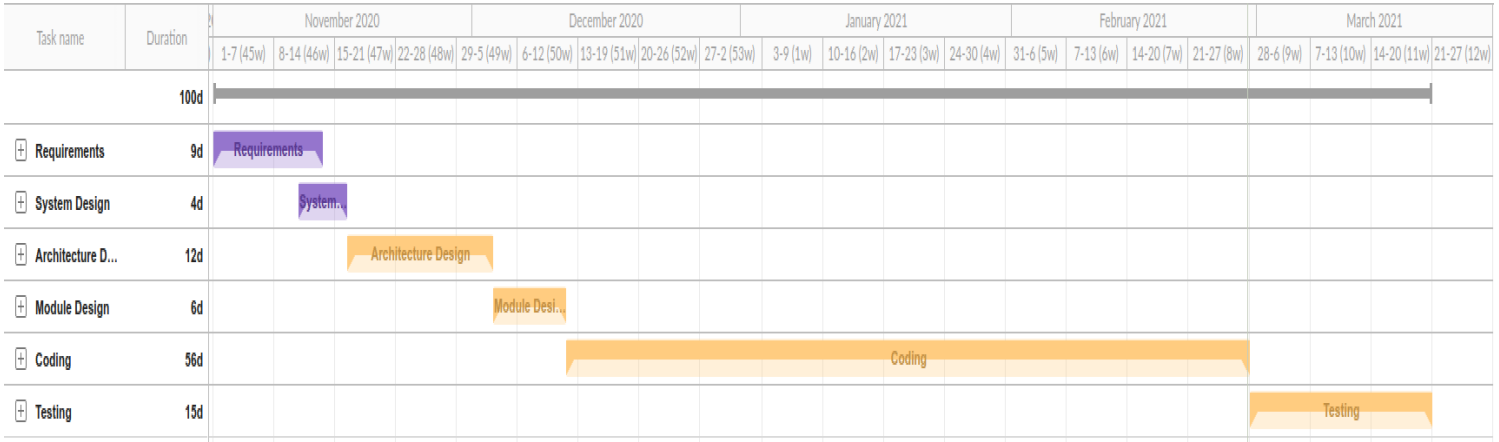
Appendix Figure 26

18. Context Diagram of Existing System

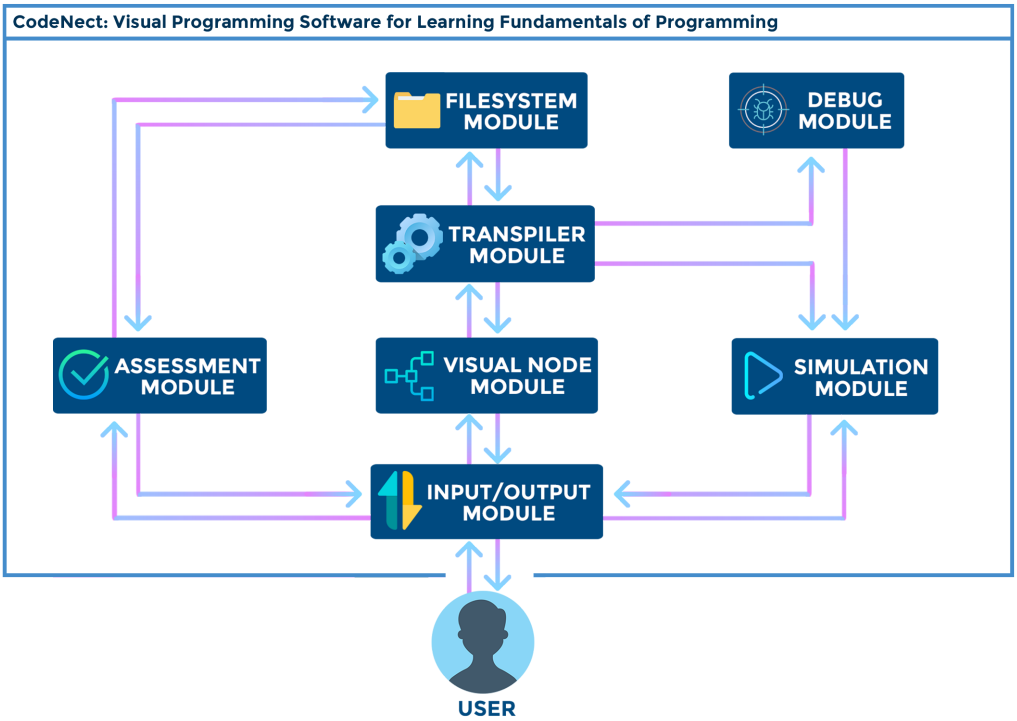


Appendix Figure 27

19. Context Diagram of Proposed System



Appendix Figure 28
20. Gantt Chart of the Development of CodeNect



Appendix Figure 29

21. Theoretical Framework of CodeNect: Visual Programming Software for Learning Fundamentals of Programming

Appendix C. Unit Testing

UNIT TESTING SHEET

Proponents: LIM-IT, BRANDON B., PUNAY, JAYKEL O.

Date: March 31, April 20, and May 3, 2021

FUNCTIONALITY	PASSED/FAILED	REMARKS
Visual Nodes Module		
1. Create Nodes	PASSED	OK
2. Edit Nodes	PASSED	OK
3. Delete Nodes	PASSED	OK
4. Connect Nodes	PASSED	OK
5. Process Logic	PASSED	OK
Filesystem Module		
1. Create Project	PASSED	OK
2. Open/Load Project	PASSED	OK
3. Save Project	PASSED	OK
4. Linux platform compatibility	PASSED	OK
5. Windows platform compatibility	PASSED	OK
Input/Output Module		
1. Keyboard compatibility	PASSED	OK
2. Mouse Compatibility	PASSED	OK
3. User interaction	PASSED	OK
4. Feedback/Response to User	PASSED	OK
Debug Module		
1. Detect warning/error	PASSED	OK
2. Identify cause of warning/error	PASSED	OK
3. Find cause of warning/error	PASSED	OK
4. Display warning/error	PASSED	OK
Simulation Module		
1. Forward iterate control	PASSED	OK
2. Backward iterate control	PASSED	OK

3. Automatic iteration control	PASSED	OK
4. Timer for iteration control	PASSED	OK
5. Stop/Reset iteration control	PASSED	OK
Transpiler Module		
1. Transpile visual code to C language	PASSED	OK
2. Compile transpiled code	PASSED	OK
3. Run compiled code	PASSED	OK
4. Linux compatibility	PASSED	OK
5. Windows compatibility	PASSED	OK
Assessment Module		
1. Display assessment	PASSED	OK
2. Submit code for assessment	PASSED	OK
3. Compare submitted with expected output	PASSED	OK
4. Calculate score	PASSED	OK
5. Show mistakes	PASSED	OK

Table 15. Unit Testing

Appendix D. Integration Testing

INTEGRATION TESTING SHEET

Proponents: LIM-IT, BRANDON B., PUNAY, JAYKEL O.

Date: May 21, 2021

MODULE	FUNCTIONALITY	REMARKS
Visual Nodes Module	1. Create Nodes	OK
	2. Edit Nodes	OK
	3. Delete Nodes	OK
	4. Connect Nodes	OK
	5. Process Logic	OK
Filesystem Module	1. Create Project	OK
	2. Open/Load Project	OK
	3. Save Project	OK
	4. Linux platform compatibility	OK
	5. Windows platform compatibility	OK
Input/Output Module	1. Keyboard compatibility	OK
	2. Mouse Compatibility	OK
	3. User interaction	OK
	4. Feedback/Response to User	OK
Debug Module	1. Detect warning/error	OK
	2. Identify cause of warning/error	OK
	3. Find cause of warning/error	OK
	4. Display warning/error	OK
Simulation Module	1. Forward iterate control	OK
	2. Backward iterate control	OK
	3. Automatic iteration control	OK
	4. Timer for iteration control	OK
	5. Stop/Reset iteration control	OK
Transpiler Module	1. Transpile visual code to C language	OK
	2. Compile transpiled code	OK
	3. Run compiled code	OK

	4. Linux compatibility	OK
	5. Windows compatibility	OK
Assessment Module	1. Display assessment	OK
	2. Submit code for assessment	OK
	3. Compare submitted with expected output	OK
	4. Calculate score	OK
	5. Show mistakes	OK

Table 16. Integration Testing

Appendix E. Profile of Respondents

Respondents	**Number**
Students (BSIT/BSCS)	12
IT Professionals	10
Total	**22**

Appendix F. Frequence Distribution Table

Technical

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The information is clear, concise, and informative to the intended audience.	4	40%	3	30%	2	20%	1	10%	10	100%
The software provides accurate and correct data.	5	50%	4	40%	1	10%			10	100%
The modules areinterconnected with each other and functions as a whole.	7	70%	2	20%	1	10%			10	100%

Table 18. Frequency Distribution of Scores on Functionality (Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The software is reliable in normal use	3	30%	5	50%	1	10%	1	10%	10	100%
Software is bug free.	2	20%	4	40%	3	30%	1	10%	10	100%
The system uses standard equipment that is reliable, widely available and applicable to a variety of users.	4	40%	5	50%	1	10%			10	100%

Table 19. Frequency Distribution of Scores on Reliability (Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The software is easy to understand.	4	40%	3	30%	2	20%	1	10%	10	100%
The software is easily operated by the intended user.	3	30%	6	60%	1	10%			10	100%
The program is attractive and interesting; it motivates users to continue using the program and exploring career options.	5	50%	3	30%	1	10%	1	10%	10	100%

Table 20. Frequency Distribution of Scores on Usability (Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
If the program requires special equipment, the requirements are minimal and clearly stated by the developer.	5	50%	4	40%	1	10%			10	100%
The program doesnt consume large amount of memory that can slow down the processing of the system.	6	60%	2	20%	2	20%			10	100%
The program can easily identify the cause of failure within the software.	6	60%	2	20%	2	20%			10	100%

Table 21. Frequency Distribution of Scores on Efficiency (Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The effort required to change the system functions is minimal.	5	50%	3	30%	2	20%			10	100%
The program is stable that if when something is changed, it will not affect the processing of the system.	5	50%	3	30%	2	20%			10	100%
The effort needed to test the system is minimal.	4	40%	5	50%	1	10%			10	100%

Table 22. Frequency Distribution of Scores on Maintainability (Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The effort required to install the system is minimal.	8	80%	1	10%	1	10%			10	100%
The system has the ability to adapt to new specifications or operating environments.	6	60%	2	20%	2	20%			10	100%

Table 23. Frequency Distribution of Scores on Portability (Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
Information about controls are understandable and available to the users.	4	40%	3	30%	3	30%			10	100%
The language is non-discriminatory. Content is free from race, ethnic, gender, age and other stereotypes.	9	90%	1	10%					10	100%
The content is free from spelling and grammatical errors.	5	50%	3	30%	2	20%			10	100%

Table 24. Frequency Distribution of Scores on User-Friendliness (Technical)

Non-Technical

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The information is clear, concise, and informative to the intended audience.	8	66.67%	4	33.33%					12	100%
The software provides accurate and correct data.	10	83.33%	2	16.67%					12	100%
The modules are interconnected with each other and functions as a whole.	8	66.67%	3	25%	1	8.33%			12	100%

Table 25. Frequency Distribution of Scores on Functionality (Non-Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The software is reliable in normal	6	50%	5	41.67%	1	8.33%			12	100%

Table 26. Frequency Distribution of Scores on Reliability (Non-Technical)

Indicator	Excellent		Very Good		Good		Fair		Total	
	f	%	f	%	f	%	f	%	f	%
The software is easy to understand.	6	50%	5	41.67%	1	8.33%			12	100%

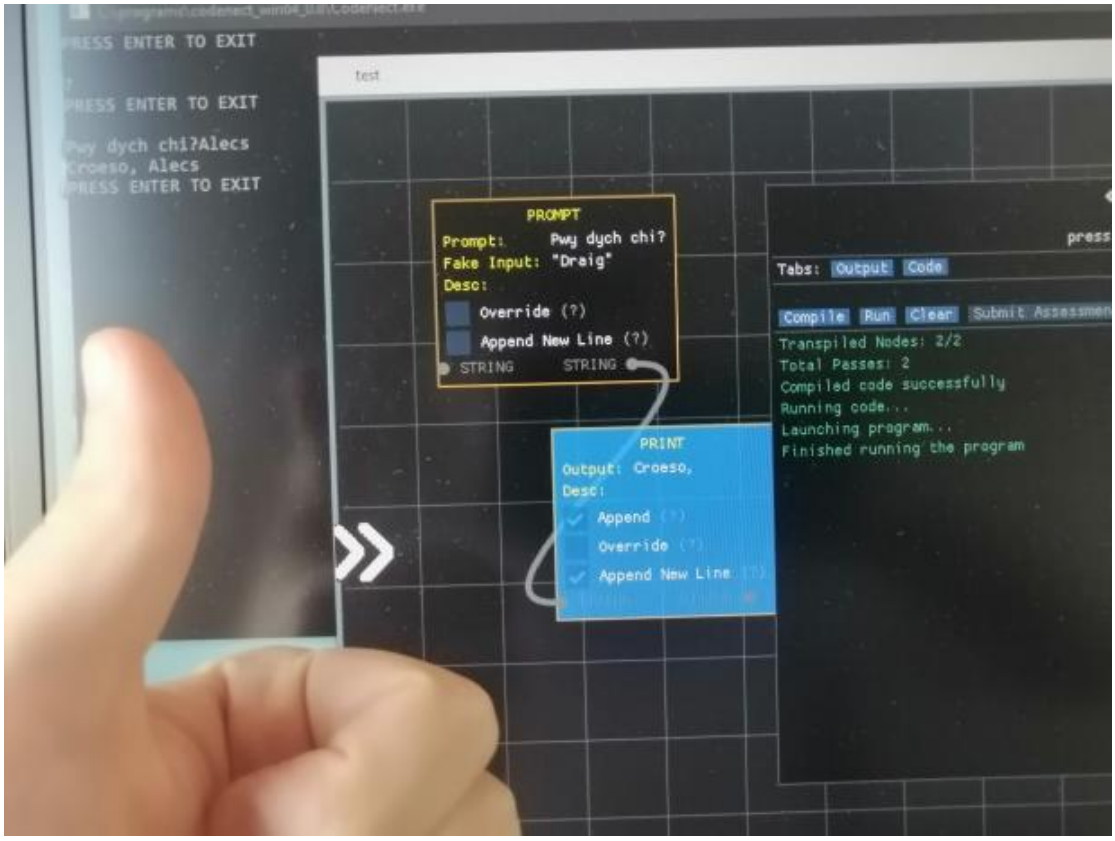
The program is attractive and interesting; it motivates users to continue using the program and exploring career options.	6	50%	4	33.33%	2	16.67			12	100%

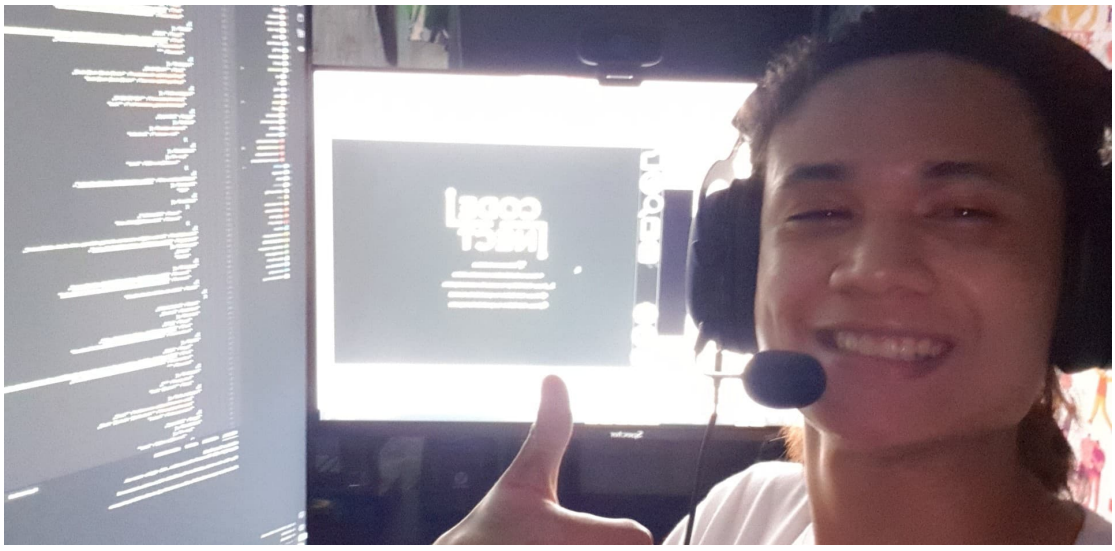
Table 27. Frequency Distribution of Scores on Usability (Non-Technical)

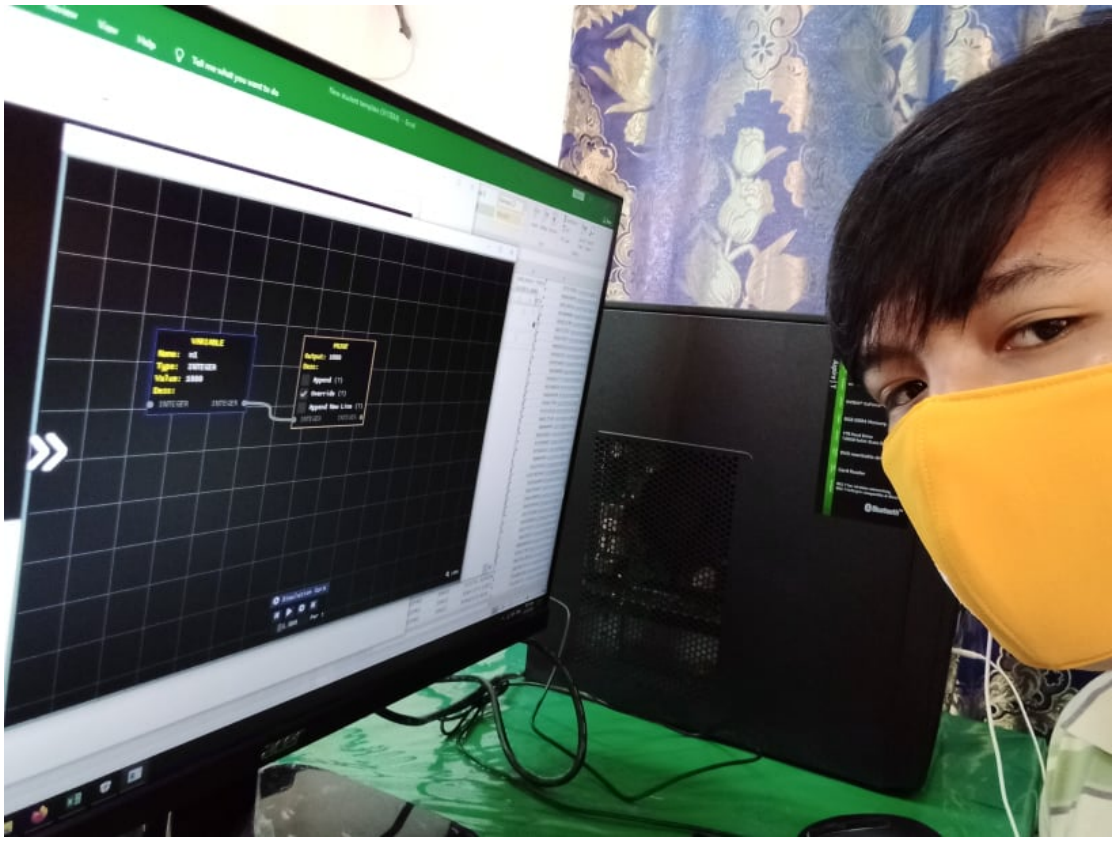
Indicator	**Excellent**		**Very Good**		**Good**		**Fair**		**Total**	
	f	**%**	**f**	**%**	**f**	**%**	**f**	**%**	**f**	**%**
Information about control is understandable and available to the users.	4	33.33%	7	58.33%	1	8.33%			12	100%
The language is non-discriminatory. Content is free from race, ethnic, gender, age and other stereotypes.	10	83.33%	2	16.67%					12	100%
The content is free from spelling and grammatical errors.	5	41.67%	3	25%	2	16.67%			12	100%

Table 28. Frequency Distribution of Scores on User-Friendliness (Non-Technical)

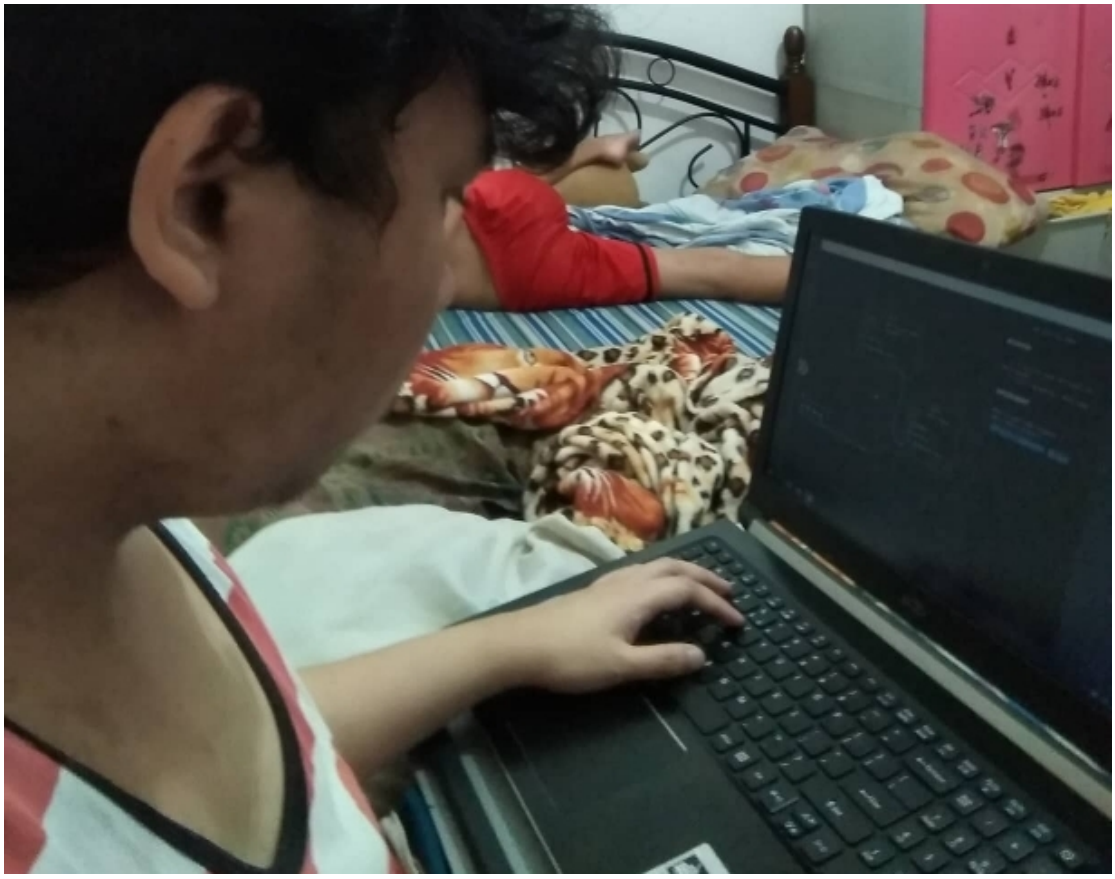
Appendix G. Software Evaluation

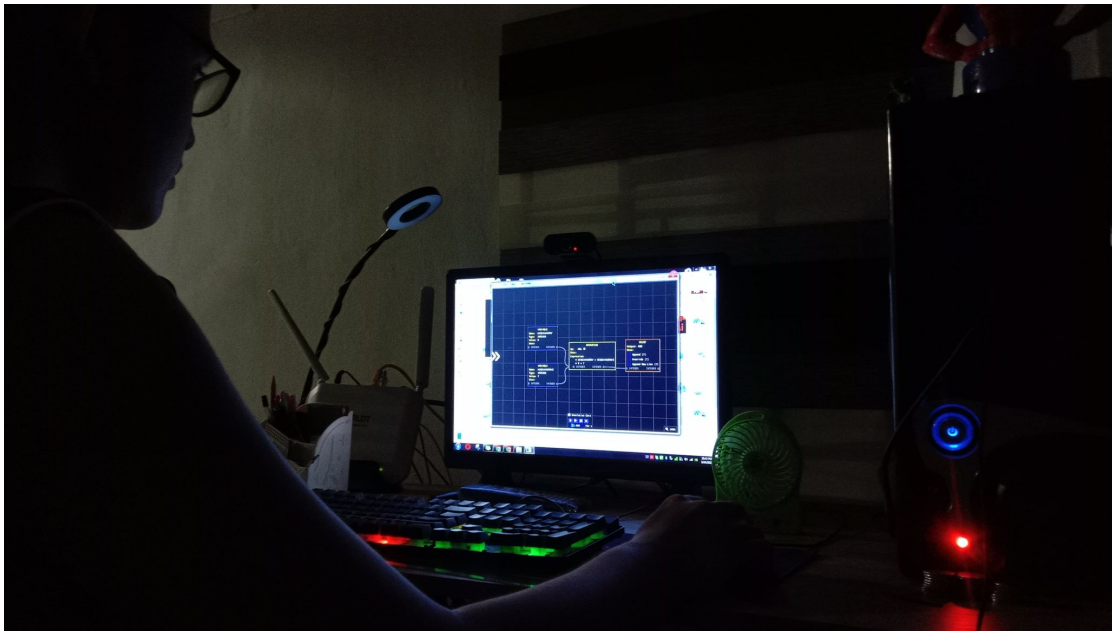

PRESS ENTER TO EXIT
PRESS ENTER TO EXIT
Pwy dych chi?Alecs
Croeso, Alecs
PRESS ENTER TO EXIT
test
PROMPT
Prompt: Pwy dych chi?
Fake Input: "Draig"
Desc:
Override (?)
Append New Line (?)
STRING STRING
PRINT
Output: Croeso,
Desc:
Append (?)
Override (?)
Append New Line (?)
Tabs: Output Code
Compile Run Clear Submit Assessment
Transpiled Nodes: 2/2
Total Passes: 2
Compiled code successfully
Running code...
Launching program...
Finished running the program

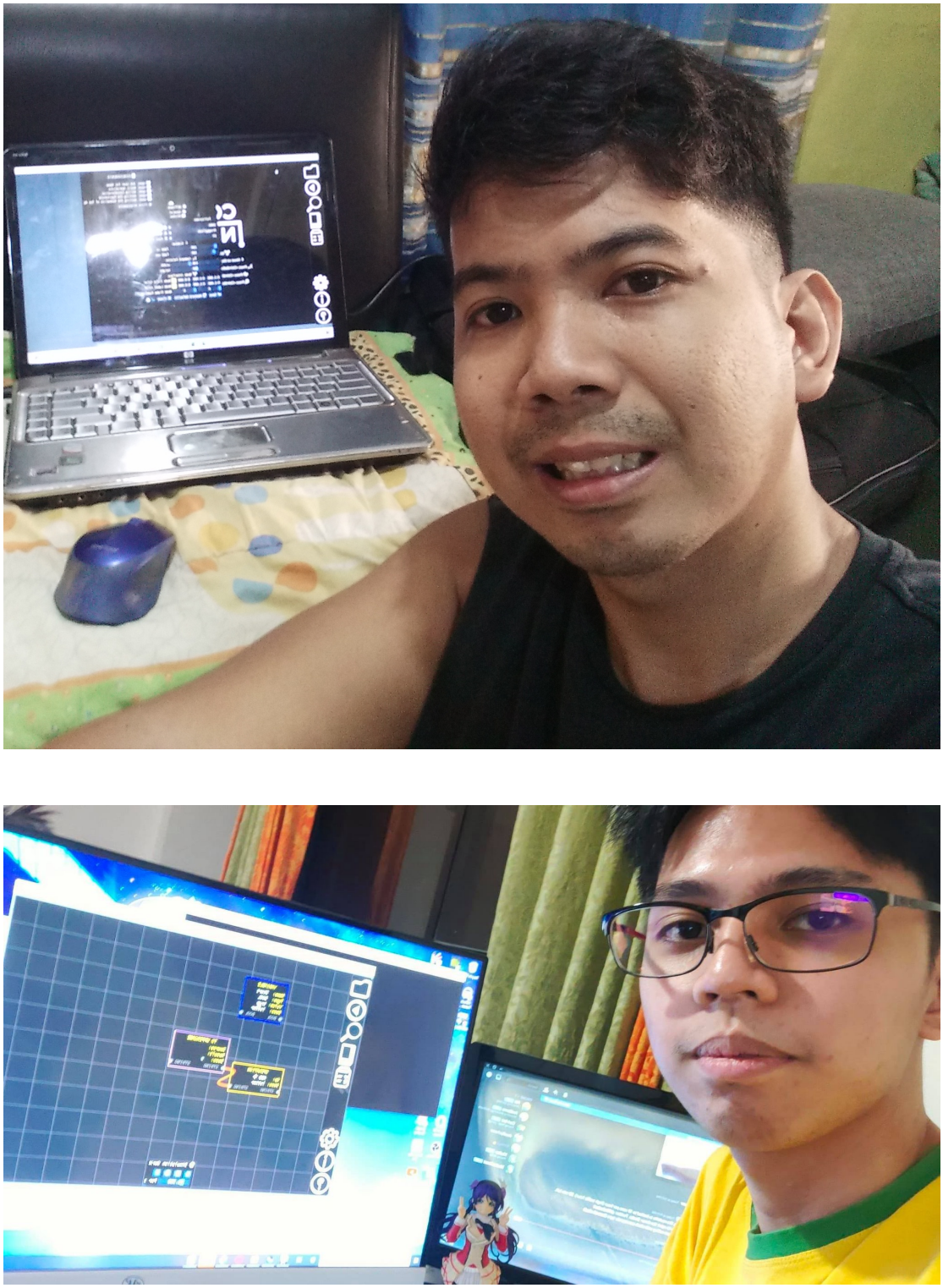




Appendix Figure 30. IT/CS Professionals Software Evaluation Pictures







Appendix Figure 31. IT/CS Students Software Evaluation Pictures

Appendix H. Relevant Source Code

```
#include "core/app.hpp"
#include "core/config.hpp"
#include "core/defines.hpp"
#include "core/font.hpp"
#include "core/project.hpp"
#include "modules/filesystem.hpp"
#include "modules/simulation.hpp"
#include "modules/transpiler.hpp"
#include "ui/about.hpp"
#include "ui/alert.hpp"
#include "ui/assessments.hpp"
#include "ui/command_palette.hpp"
#include "ui/diff_viewer.hpp"
#include "ui/docs.hpp"
#include "ui/help.hpp"
#include "ui/inspector.hpp"
#include "ui/node_interface.hpp"
#include "ui/settings.hpp"
#include "ui/sidebar.hpp"
#include "ui/sidebar_handler.hpp"
#include "ui/sidebar_indicator.hpp"
#include "ui/simulation_control.hpp"
#include "ui/terminal.hpp"
#include "ui/zoom.hpp"

int main(int argv, char** args)
{
 CodeNect::App app;
 app.init();

 if (CodeNect::Config::init() != RES_SUCCESS) return -1;

 if (app.init_app() != RES_SUCCESS) return -1;

 CodeNect::Font::init();

 //Command Palette
 if (CodeNect::CommandPalette::init() != RES_SUCCESS) return -1;
 if (CodeNect::Alert::init() != RES_SUCCESS) return -1;

 CodeNect::App::register_commands();
 CodeNect::Project::register_commands();
 CodeNect::Settings::register_commands();
 CodeNect::About::register_commands();
 CodeNect::Help::register_commands();

 if (CodeNect::Docs::init() != RES_SUCCESS) return -1;
 CodeNect::Docs::register_commands();

 //Inspector
 if (CodeNect::Inspector::init() != RES_SUCCESS) return -1;
 CodeNect::Inspector::register_commands();

 //DiffViewer
 if (CodeNect::DiffViewer::init() != RES_SUCCESS) return -1;

 //Assessments
 if (CodeNect::AssessmentsUI::init() != RES_SUCCESS) return -1;
 CodeNect::AssessmentsUI::register_commands();

 //Zoom
 if (CodeNect::Zoom::init() != RES_SUCCESS) return -1;
 CodeNect::Zoom::register_commands();

 //SimulationControl
 if (CodeNect::SimulationControl::init() != RES_SUCCESS) return -1;
```

```
CodeNect::SimulationControl::register_commands();

//Sidebar
CodeNect::Sidebar sidebar;
CodeNect::SidebarIndicator sidebar_indicator;
CodeNect::SidebarHandler sidebar_handler;

if (sidebar.init() != RES_SUCCESS) return -1;
if (sidebar_indicator.init() != RES_SUCCESS) return -1;
sidebar_handler.init(&sidebar, &sidebar_indicator);

//Interfaces
if (CodeNect::NodeInterface::init() != RES_SUCCESS) return -1;

//Transpiler
if (CodeNect::Transpiler::init() != RES_SUCCESS) return -1;
CodeNect::Transpiler::register_commands();

//Terminal
if (CodeNect::Terminal::init() != RES_SUCCESS) return -1;
CodeNect::Terminal::register_commands();

ImGuiIO* imgui_io = &ImGui::GetIO();

if (argv > 1)
{
 const char* filepath = args[1];
 if (std::filesystem::exists(filepath))
  CodeNect::Project::open(filepath);
}

PLOGI << "Initialization is complete";

while(!glfwWindowShouldClose(CodeNect::App::window))
{
 float dt = imgui_io->DeltaTime;

 glfwPollEvents();

 //update
 sidebar_handler.update(dt);
 CodeNect::NodeInterface::update(dt);
 CodeNect::Simulation::update(dt);

 app.render_start();

 //draw other here
 sidebar.draw();
 sidebar_indicator.draw();
 CodeNect::NodeInterface::draw();
 CodeNect::Project::draw();
 CodeNect::CommandPalette::draw();
 CodeNect::Alert::draw();
 CodeNect::Inspector::draw();
 CodeNect::AssessmentsUI::draw();
 CodeNect::DiffViewer::draw();
 CodeNect::Zoom::draw();
 CodeNect::Terminal::draw();
 CodeNect::SimulationControl::draw();
 CodeNect::Docs::draw();

 app.render_end();
}

sidebar.shutdown();
CodeNect::Commands::shutdown();
CodeNect::Font::shutdown();
```

```
CodeNect::Config::shutdown();
CodeNect::Project::shutdown();
CodeNect::Transpiler::shutdown();
app.shutdown();

return 0;
}
```

```
#ifndef _ASSESSMENTS_HPP
#define _ASSESSMENTS_HPP

#include <vector>
#include <string>

namespace CodeNect
{
struct Assessment
{
 const char* title;
 std::string str_content;
 std::vector<std::string> v_expected;
 std::vector<std::string> v_submission;
};

struct AssessmentResult
{
 Assessment assessment;
 int score = 0;
 std::vector<int> v_lines_diff;
};

struct Assessments
{
 static std::vector<Assessment> v_list;
 static bool has_assessment;
 static Assessment current_assessment;
 static std::vector<AssessmentResult> v_results;

 Assessments() = delete;
 static void submit(std::vector<std::string>&);
 static int get_score(std::vector<std::string>&, std::vector<std::string>&, std::vector<int>&);
};
}

#endif //_ASSESSMENTS_HPP
```

```
#ifndef _DEBUGGER_HPP
#define _DEBUGGER_HPP

#include "node/node.hpp"
#include "core/message_info.hpp"

namespace CodeNect
{
struct Debugger
{
 static std::vector<MessageInfo> v_msg_info;

 static void add_message(const std::string&, OUTPUT_TYPE = OUTPUT_TYPE::WARNING, Node* = nullptr, DOC_ID = DOC_ID::EMPTY);
 static void clear(void);
};
}

#endif //_DEBUGGER_HPP
```

```
#ifndef _FILESYSTEM_HPP
#define _FILESYSTEM_HPP

#include <filesystem>
#include <string>
#include <vector>
#include <GLFW/glfw3.h>
#include "core/image.hpp"

namespace CodeNect::Filesystem
{
struct Paths
{
 static std::filesystem::path out_dir;
 static std::filesystem::path cur_path;
};

bool open_project_file(std::string& project_filepath);
bool open_filepath(std::string& filepath);
bool save_to_file(std::string& out_filepath, const char* ext, const std::string& content);
int load_texture_from_file(const char* filename, CodeNect::Image& image);
unsigned char* load_texture_from_file(const char* filename, GLFWimage& image);
std::vector<std::string> parse_stdout(std::string&);
}

#endif //_FILESYSTEM_HPP
```

```
#ifndef _INPUT_HPP
#define _INPUT_HPP

#include <functional>
#include <GLFW/glfw3.h>
#include <vector>

namespace CodeNect
{
typedef std::function<bool(int key, int scancode, int mods)> keycallback_t;

struct Input
{
 static std::vector<keycallback_t> v_keypresses;
 static std::vector<keycallback_t> v_keyreleases;

 Input() = delete;
 static void register_key_event(keycallback_t fn, bool press = true);
 static void key_callback(GLFWwindow* window, int key, int scancode, int action, int mods);
};
}

#endif //_INPUT_HPP
```

```
#ifndef _NODE_TO_CODE_HPP
#define _NODE_TO_CODE_HPP

#include "node/node_var.hpp"
#include "node/node_array.hpp"
#include "node/node_print.hpp"
#include "node/node_prompt.hpp"
#include "node/node_cast.hpp"
#include "node/node_op.hpp"
#include "node/node_math.hpp"
#include "node/node_cmp.hpp"
#include "node/node_size.hpp"
#include "node/node_array_access.hpp"
#include "node/node_branch.hpp"
#include "node/node_string.hpp"
#include "node/node_set.hpp"
#include "node/node_for.hpp"

namespace CodeNect::NodeToCode
{
std::string indent(void);
std::string slot_to_str(NODE_SLOT&);
std::string slot_to_spec(NODE_SLOT&);
std::string cmp_to_str(NODE_CMP&);
std::string to_array(NodeArray*);
std::string comment(Node*);

std::string ntc_var(NodeVariable*);
std::string ntc_cast(NodeCast*, bool, std::string&);
std::string ntc_array(NodeArray*);
std::string ntc_array_decls(NodeArray*);
std::string ntc_print(NodePrint*, bool = false);
std::string ntc_prompt(NodePrompt*);
std::string ntc_set(NodeSet*);
std::string ntc_op(NodeOperation*, bool, std::string&);
std::string ntc_math(NodeMath*, bool, std::string&);
std::string ntc_cmp(NodeComparison*, bool, std::string&);
std::string ntc_size(NodeSize*, bool, std::string&);
std::string ntc_array_access(NodeArrayAccess*, bool, std::string&);
std::string ntc_branch(NodeBranch*, bool);
std::string ntc_for(NodeFor*);
std::string ntc_string(NodeString*, bool, std::string&);
}

#endif //_NODE_TO_CODE_HPP
```

```
#ifndef _SIMULATION_HPP
#define _SIMULATION_HPP

#include <vector>
#include "node/node.hpp"
#include "node/node_for.hpp"
#include "core/timer.hpp"

namespace CodeNect
{
struct ForState
{
 NodeFor* m_node_for = nullptr;
 std::vector<Node*> m_nodes;
};

struct Simulation
{
 static std::vector<ForState> m_v_stack;
 static std::vector<Node*> m_v_tracker;
 static bool is_playing;
 static Timer* timer;

 Simulation() = delete;
 static void iterate(int dir);
 static void reset(void);
 static void play(float max_timer);

 static bool is_in_for(Node*);
 static void update(float dt);
 static std::vector<Node*> get_node_for_block(Node*);
 static void determine_node_for_block(std::vector<Node*>&);
};
}

#endif //_SIMULATION_HPP
```

```
#ifndef _TRANSPILER_HPP
#define _TRANSPILER_HPP

#include <functional>
#include <map>
#include <string>
#include <vector>
#include "libtcc.h"
#include "core/defines.hpp"
#include "node/node.hpp"
#include "node/node_branch.hpp"
#include "node/node_for.hpp"
#include "core/message_info.hpp"

namespace CodeNect
{
struct State
{
 NodeBranch* node_branch = nullptr;
 NodeFor* node_for = nullptr;
 bool is_in_else = false;
 std::vector<std::vector<Node*>> v_seq;
 std::vector<Node*> v_rest;
};

struct Transpiler
{
 static TCCState* tcc_state;
 static std::string output_code;
 static std::string runnable_code;
 static std::vector<MessageInfo> v_output;
 static std::vector<std::string> v_declarations;
 static std::map<std::string, bool> m_temp_names;
 static std::map<std::string, bool> m_declared;
 static std::map<std::string, std::string> m_array_init; //array_name, free_method_name
 static int level;
 static bool has_ran;
 static bool has_compiled;
 static std::string recent_temp;
 static int n_transpiled;
 static std::vector<State> v_states;
 static std::vector<NodeBranch*> v_finished_branches;
 static std::vector<std::string> v_printed;

 Transpiler() = delete;
 static int init(void);
 static void register_commands(void);
 static void handle_error(void* opaque, const char* msg);
 static void add_message(const std::string&, OUTPUT_TYPE = OUTPUT_TYPE::SUCCESS, Node* = nullptr, DOC_ID = DOC_ID::EMPTY);
 static std::string get_temp_name(const char* name);
 static std::pair<std::string, bool> get_temp_name(const char* name, bool reuse);
 static bool is_valid_decls(Node*);
 static void set_pre_entry(std::string& str_incl, std::string& str_structs, bool is_tcc);
 static void build_runnable_code(std::string& out, bool is_tcc);
 static void transpile_decls(std::vector<Node*>& v, std::string& output);
 static void transpile_decls_array(std::vector<Node*>& v, std::string& output);
 static Node* transpile(std::vector<Node*>& v, std::string& output, State* = nullptr);
 static void arrange_v(std::vector<Node*>&);
 static std::vector<Node*> get_sequence(Node*, State*);
 static std::vector<std::vector<Node*>> get_v_sequence(State*);
 static std::vector<Node*> get_rest(State*);
 static int compile(void);
 static int run(void);
 static void clear(void);
 static void save_file(void);
 static void shutdown(void);

};
}

#endif //_TRANSPILER_HPP
```