

BBVPL: A Block-Based Visual Programming Language Built on Google's Blockly



Ashfaq Ahmad¹, Muhammad Idrees², Muhammad Arif Butt², Hafiz Muhammad Danish³

¹Department of Computer Science, College of CS & IT, Jazan University, Saudi Arabia

²Department of Data Science, University of the Punjab, Lahore, Pakistan

³Department of Computer Science, University of Lahore, Pakistan

ABSTRACT

The paper presents a Block-based Visual Programming Language (BBVPL) built on an open-source Google's Blockly Framework. BBVPL inherits Blockly's alive features as well as provides new functionalities. Although, the existing Visual Programming Languages (VPLs) such as Flowgorithm, Raptor, Flint, Larp, Snap, Scratch, Kodu, Blockly, and many more, provide a graphical computer programming interface for learning and educational purposes. Some of them are used for kid's games and robotic applications development and others for general-purpose programming. But they lack fundamental programming capabilities mostly being used to teach basic programming concepts. Therefore, the feature enhancement approach has used, and new iconic vocabulary and grammar created by Blockly's Block-factory module to develop a new language (BBVPL). In this paper, BBVPLs programming features have introduced to teach computer programming skills, especially for beginners in computer science. The BBVPL provides an intuitive visual drag-drop program development interface and a user-friendly console for input-output (I/O). It offers essential programming features, including Object Orientation, Modular Support, Conditions (If-Else), Loops (For, While), Exceptions (Throw, catch, Finally), and File Handling. However, the BBVPL can translate Visual Program into Textual Program that will execute, and results will be shown on the console screen. To accomplish this, some experimental visual programs and their translated textual codes are also part of our paper, indicating the smooth working of our tool. The integration of user's roles and rights management that will provide proper user access to save their programs on the server, debugging, and multithreading features will be the parts of BBVPL in the future.

Keywords: Textual Programming Languages (TPLs), Visual Programming Languages (VPLs), Open Source, Web-based, Interactive input-output (I/O), Exceptions, Debugging, Multithreading, Block-based Programming, Visual Programming and user interface

1 INTRODUCTION

The computer programs can be developed either using Textual Programming Languages (TPLs) or Visual Programming Languages (VPLs). In TPLs, a computer program is a sequence of textual directives that might be operation fields, operand fields, name fields, and comment

fields. Java, Php, Python, etc. are a few examples of TPLs [1]. On the other side, VPLs provide a visual interface that facilitates the users by manipulating program elements graphically [2]. VPLs use visual syntax that represents terminals in the form of pictures [3]. Software developers solely need to drag-drop visual elements [4] on a canvas to create, modify, and design pictorial program structure. A lot of VPLs are accessible to educational and professional programming purposes. Flowgorithm, Raptor, Visual Logic, Scratch, Blockly, Kodu, Microsoft VPL, EToys, TouchDevelop, GameSalad, Open Roberta and Lego Mindstorms are a few examples of Visual Programming Languages [5]. A programming language, either a VPL or a TPL should provide competence to create arrays, custom modules, the user's defined data types and file handling [6]. It should provide an interactive input-output (I/O) interface, exception handling, multithreading, and debugging capabilities. The surviving VPLs lack fundamental programming features discussed earlier [7]. So, the time is to propose an all-in-one VPL that can provide an interactive user I/O with proper file handling. A VPL that can manage object orientation, lists of data, customized modules, exceptions and debugging capability to grip bugs and errors.

Google's Blockly [8], [9], an open-source framework, is provided with visual block-based icons and a drag-drop programming environment. A computer programmer selects desired visual blocks and connects them at the provided platform according to the Blockly's syntax and program's logic. Blockly also permits the transformation of visual programs into corresponding textual codes, e.g., JavaScript, Php, Python, Dart, and Lua [10]. It executes visual programs and empowers to design a new Visual Programming Language (VPLs). Existing Blockly provides various features including arrays creation, conditions, loops, custom modules, and input-output (I/O). Still, some critical programming concepts are missing, just as users defined data types, exception handling, file handling competency [11], debugging, and multithreading support.

In this paper, the work has been done on a Block-based Visual Programming Language (BBVPL), a VPL to teach fundamental computer programming skills, that is stemming from Google's Blockly framework. The Blockly's block factory [12] module assists in developing BBVPL's iconic vocabulary and grammar. BBVPL provides Blockly's contemporary features as well as newly created functionalities. It includes object orientation and exception handling power

with proper availability of try-catch blocks. It has all-inclusive functionalities related to files, including open file, close file, read line, read char, write line, seekp, seekg, tellp, and tellg along with others. Although input-output (I/O) feature is available in Blockly, shown in Figure 1, it is a discomfort to watch a lot of pop-ups and alerts on screen for I/O. BBVPL also enables an interactive console-based I/O facility as well.

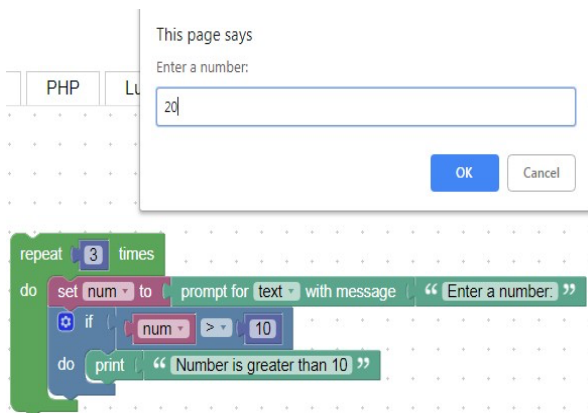


Figure 1: Blockly's sample program

2 RELATED WORK

Ivan Sutherland's groundbreaking Sketchpad system designed in 1963 at MIT, was the first application to create 2D graphics and led to the invention of VPLs [13]. A pioneered human-computer interactive software awarded with Turing Award in 1988, provided the medium of line, mechanical shapes, electric circuits drawing and the like. The domain of man-machine communication was built, where a developer used a light pen and push buttons to draw, move and more of the same. Sketchpad eliminated typed written statement method used in the past, to converse computer. In 1965, Ivan Sutherland's Brother William contributed by creating a Visual Dataflow Language (VDL) [14]. He used Sketchpad on the TX-2 computer at MIT, the first graphics application, to develop VDL that provided the services of creation, debugging, and execution of data flow diagrams. David Canfield Smith's published his Ph.D. work in 1975. "Pygmalion: A Creative Programming Environment" is an application that provided us icon-based programming interface. A programmer could use visual icons and connections to create and link small visual objects [15].

Flowchart Interpreter System (FLINT) was developed to avoid syntax problems and helped to enhance the solving skills of student's issues as an introductory programming course [16]. It provides students, a top-down environment to design algorithms as flowcharts and iconic interface in which buttons enable to add and remove child nodes in flowchart structure instead of drag-drop visual elements. A student begins by developing a structure chart and then creates many significant steps in it that can be implemented separately. It facilitates with input, output, if-conditions, loops, variables, and modules. Step by step program execution (debugging) by highlighting the current statement is the key feature that makes variables and programs logic observable [17].

In 2004, the first release of Raptor published, in which Martin Carlisle designed a flowchart-based application for students to visualize algorithm graphically. It was not only used for writing and execution of a computer program but also

used to teach programming concepts and provided sequential, selection, and repetitive structures for computer programming. Hooshyar and *et. al.*, in their paper [18], emphasize that Raptor and similar visual programming tools enhance the understanding of novice programmers. Raptor as a CS0 [19] course, provides arrays, object orientation, modular support, file input/output, and debugging services but lack of exception handling features and threads creation. The primary purpose of Raptor was to enhance solving skills of student's problems. Its stable release of 2015 is freely available [20].

Larp, a software which was officially released in 2004, to teach structured programming and algorithms, is using pseudo-code and flowcharts [21]. Rapid prototyping of algorithms is the principal purpose of Larp programming language [22]. It has the potential to transform textual algorithms (pseudo code) into flowcharts to make them more understandable [23]. Object orientation, arrays creation variables, if-conditions, structures, modular support, debugging, and data storage features, with a simple user-friendly working environment, makes it more practical especially for teaching purposes. Latest stable version of the application released in 2008, is freely available.

Scratch, a free scholastic programming language developed by Lifelong Kindergarten Group at Massachusetts Institute of Technology (MIT) in 2005, is based on the idea of blocks. In that, the user has to drag and drop programming blocks from a block palate and connect them like a jigsaw puzzle [24]. Scratch is also using as an Integrated Development Environment (IDE) for children of aged between 9 to 11. Its block programming technique makes it easy to write program code and to learn problem-solving skills for schoolchildren [25]. It has different tools to create interactive stories, kid's games, simulations, animations arts and many more using block-based programming [26]. Built-in paint and sound editor are the parts of it, but unable to provide object orientation, file I/O, and exception handling amenities. Its stable version was released in 2013 [27].

Microsoft VPL [28], a robotic application development tool was initially released in 2006, in which a program is a sequence of activities and each of them is connected to perform a single task. A Microsoft VPL data flow is represented as adjacent activity blocks of inputs and outputs that can be connected with other activity blocks. It has the competency of creating arrays, lists, if-else conditions, variables, customized activities, debugging, and multitasking support, but lack of object orientation, file I/O, and exceptions. Students, enthusiasts, web developers, and professional programmers are the audience of Microsoft VPL [29].

Michael Agustin, Dan Treiman and Tan Tran in 2007 founded GameSalad, a visual drag and drop development tool based on games. The software enables professional programmers as well as non-programmers to create two-dimensional (2D) games for social networks like Facebook, Android, and IOS devices (iPhone/iPad) [30]. It allows consumers to design and develop their distinctive games and offers the ability to collaborate with other users. GameSalad used innovative technology and provided a professional-level of artificial intelligence using complex algorithms. It is an open-source platform, and freely available to download [31].

Stencyl was published in 2011 for kid's game development [32]. It can manage to create 2D games for the web and mobile devices. Aside from icon palate contains resources and logic to create an actor and sets its behavior. A user may define backgrounds, tiles, and sounds, to plan more attractive games [33]. It provides intuitive tool-set and best drag-drop interface to accelerate workflow. Stencyl supports comprehensive deployment platforms including IOS (iPhone/iPad), Android, Windows, Mac, and Linux. Still, there are some deficiencies in the above-defined characteristics, essential for the completeness of a VPL [34].

In 2012, Google developed Blockly, an open-source VPL similar to scratch. A user may create a program using Blockly provided web-interface and he is capable of executing as well as translating it into JavaScript, Php, Dart, XML, and Python [10]. A drag-drop environment with a connection point attached to each block enables to chain together on screen. Blockly is neither a full application nor a language that is ready to use for end-users. It is a platform where users can work with an already built-in set of vocabulary (visual icons or blocks) as well as can create their visual blocks and can associate code with each of them accordingly. Blockly provides arrays, loops, modular support, conditions, and others. Still, it surpluses to enable interactive user I/O interface. It shows Java Script's alert pop-ups to represent output of the program instead of a proper black screen console. Moreover, exception handling, multithreading, debugging, structures, object orientation, and file I/O services are missing segments, that are necessary for the completeness of a language [11].

Lego produced a hardware-software platform Lego Mindstorms EV3 in 2013, for the development of a robotic application for Lego building blocks. Robot development is the primary step using Lego bricks, motors, wheels, sensors, etc. Then a user has to download EV3 [35] application for PC/MAC or Tablet to write a program. The software facilitates with a toolbar, full of functional blocks and drag-drop capability that enables a developer to join different activates. A programmer can bring life into a robot by controlling motors and sensors programmatically [36].

Visual Logic, another intuitive graphical flowchart-based VPL, developed to teach students diagrammatically. A graphical interface that provides essential programming concepts including variables, arrays, pre-and-post test loops, if-else conditions, procedures, debugging, and enhanced I/O facilities from console and text files as well. It uses minimal syntax and clarifies the program logic and concepts rather than focusing on program syntax. But its major drawback is operating system dependency, it is functional only for windows OS [37]. It has the proficiency in executing flowcharts but object orientation, exception handling, and multitasking features are not developed in it. The latest version was launched in 2014 [38].

Flowgorithm, designed by Devin Cook was a free application, first appeared in 2014. It is a graphical tool that allows developers to write and execute a program with the help of flowcharts [39]. Classical flowchart symbols appear in a toolbox; a user solely needs to drag and drop them instead of writing the bulk of instructions. It provides an easy way to write a computer program that can execute directly as well as can translate into other source languages like C#, C++, Python,

Perl, Java, etc. The existing features of the Flowgorithm are array creation, modular support, graphical variable window watch, multilingual support, loop, flexible expressions, recursion, and understandable interactive output [40].

3 BLOCKLY'S EXISTING FEATURES

Google's Blockly Library based on JavaScript is an open-source framework that activates the development of new Visual Programming Languages (VPLs). Blockly also provides a drag-drop platform to develop a visual computer program by using its built-in visual icons. It can generate textual code in different languages including JavaScript, Php, Python, Lua, and Dart. It can be used to create custom generators of other TPLs as well [11]. It provides a toolbar, including the "Logic" group, which holds conditional blocks, e.g., If-do, If-Else, Else-If, etc. Blockly implemented transitive connections [41], in which block makes connections according to the designed language grammar or syntax and a sound produces that indicates the creation of a correct block couple. The "C" shaped block in Blockly enables loops, e.g., "For Loop", "While Loop", and "For Each Loop". It allows putting repetition statement blocks inside the "C" shaped block.

"Math" group contains mathematical visual blocks to calculate random numbers, square root and to perform addition, subtraction, division, and similar operations. Trigonometric functions, e.g., sine, cosine, and tangent are also part of this collection. It has the proficiency to create variables, arrays, and lists. Blockly provides the user's input-output (I/O) capability using windows alerts and prompts. In Blockly's toolbar, the "Text" group contains a "print" block connects with a "variable" or a "string" block to display values, while the "prompt" block prompts a message and stores inputted value of a variable shown in Figure 1.

Blockly's "Functions" group provides modular functionality. Again the "C" shaped visual "do something" block allows us to create functions and return values. It does also permit the user to put related visual statements in- side. Blockly provides a drag-drop platform as well as a textual language generator framework that translates the visual code to TPL's, e.g., JavaScript, Lua, Dart, Php, Python, and XML. It can also execute translated JavaScript code [42] by clicking the rightmost "Run" button as well. Blockly also provides pop-up-based input-output (I/O) facilities. Its imperative visual icons are shown in Appendix A.

4 METHODOLOGY

It's all starts with an open-source Google's Blockly framework that is not only a visual programming platform but also a library that allows developers to create their visual languages [8]. Block-based Visual Programming Language (BBVPL) is implemented by using the Blockly framework, so it presents interface similar to Blockly but with more programming features including structures, exceptions, files, and interactive console integration. Blockly's block factory module [12] was used that enabled us to create an iconic vocabulary and language syntax for BBVPL. Then the newly created connectable visual icons became part of BBVPL. The three-step program execution life cycle shown in Figure 2, in which the first step is to connect visual blocks to create a visual

program. Secondly, the visual program translates into JavaScript textual code, and in the third step, translated code executes, which is further communicating with the console application for input-output (I/O) purpose.

BBVPL comes up with a console gadget for interactive user input-output (I/O). It is an open-source console template that is modified according to BBVPL's requirement. The visual block program, translated into JavaScript executes and communicates with black console screen to print outputs and to deal with inputs. Figure 2 is screening the complete BBVPL's program execution life cycle. Then the software-testing phase started, in which complex program scenarios tried to execute and corrected reported bugs. Finally, the complete web application BBVPL deployed on a server to provide user access.

Step 1: Development of a Visual Program

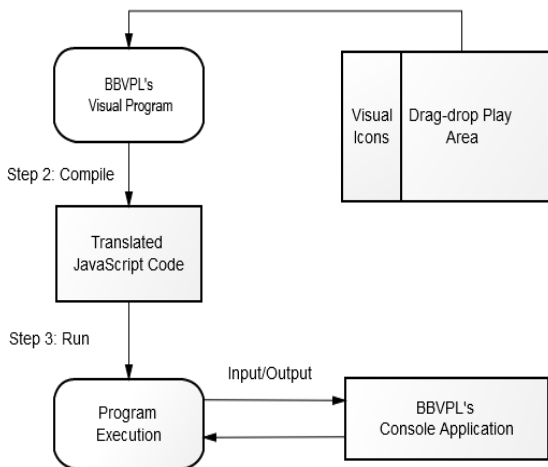


Figure 2: BBVPL's program execution life cycle

5 BLOCK-BASED VISUAL PROGRAMMING LANGUAGE (BBVPL) FEATURES

BBVPL provides the same Blockly's programming atmosphere with its existing programming proficiency's such as arrays creation, conditions, loops, modular support, etc. as well as presents newly created features that are necessary to teach fundamental programming skills. BBVPL provides console application to communicate with users. It comes up with users defined data types, exception handling facilities to catch and throw exceptions. It offers file handling simulation to deal with permanent data storage. BBVPL's freshly developed features are discussed below in detail.

5.1 Console Application

A console application [43] is a text-only computer interface that provides a more straightforward way to communicate with users either to display outputs or to take inputs, similar to the modern Textual Programming Languages (TPLs) such as C#, Java, Python, etc. Block-based Visual Programming Language (BBVPL) implemented an integrated console application. Although Blockly enables the user's input-output (I/O) using alerts and pop-ups as shown in Figure 1, but continuous alerts and prompts on the screen present an irritating atmosphere for Blockly. So BBVPL offers an interactive console, a black color screen to view the computer

program's output and to deal with the desired input. The console application is the HTML web page based on JavaScript. Figure 3 is presenting a sample BBVPL's program that requests the user to input two numbers, takes the sum of them, and prints a message on the console if the sum is greater than 10. Figure 4, presents the console's output and preferred input as per the program's logic.

```

    repeat 3 times
    do
        print "Enter first number:"
        Input num1
        print "Enter second number:"
        Input num2
        set sum to num1 + num2
        if sum >= 10
        do
            print "sum is greater than 10"
        else
            print sum
        end
    end
    
```

Figure 3: Input Output (I/O) console's related BBVPL sample program

```

    Enter first number:
    > 1
    Enter second number:
    > 2
    3
    Enter first number:
    > 10
    Enter second number:
    > 12
    sum is greater than 10
    Enter first number:
    > 12
    Enter second number:
    > 9
    sum is greater than 10
    
```

Figure 4: BBVPL's integrated console sample output

5.2 Structures

A structure is a composite data type that can contain a list of variables in an association using a continuous block of memory [44]. It is a technology that can aid software engineering and can provide object models better fit with actual problems [45]. Using structures, a developer can create a composite type as a collection of data items, e.g., an object of a student with the variable name, age, contact, and blood group. From this perspective, many programming languages facilitate object-orientation support that is necessary to learn basic programming concepts. So, the structure or user-defined data type construction capability belongs to the BBVPL, invites software developers to define their desired objects. Figure 5 presents selected "Structures" choice, which is further screening drag-able visual blocks. Table 1 is describing structures related to visual blocks, their names, and symbol images. A visual block named as "Create structure", creates a new user-defined data type, and provides an input field to write structures name and a clickable button to include the

structure’s elements. When a developer clicks on a blue button, a new pop-up opens to a drag-drop “var” block that connects with the “struct” block to comprise new structure fields, shown in Figure 5.

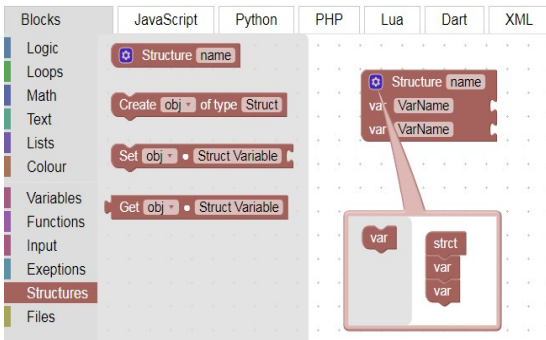


Figure 5: Selected “Structures” group

The “Create structure variable” block creates an object; a program developer needs to put an object name and structure type. “Set variable value” and “Get variable value” blocks work the same as setter and getter, select an object name, and looked-for filed name to set or get object values. A “Student” data type is created with required fields, e.g., student name, roll number, and degree information. Then a “Student” type object “student1” created and assigned values of its attributes using set blocks. After that, print the object’s values using get blocks according to the sample program’s logic in Figure 6. The BBVPL’s sample program translated into textual code shown in Figure 7.

Table 1: List of “Structures” related blocks

Sr.	Block name	Image
1	Create structure	
2	Create structure variable	
3	Set variable value	
4	Get variable value	

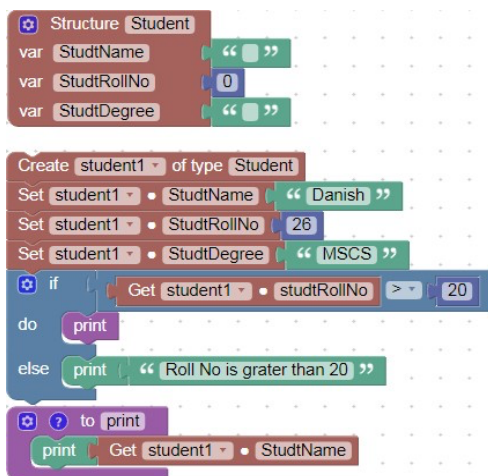


Figure 6: Structures related BBVPL’s sample program

```

var student1, obj;
function print()
{
    ConsoleDisplay(student1.StudtName);
}
structfunction Student ()
{
    this.StudtName = '';
    this.StudtRollNo = 0;
    this.StudtDegree = '';
}
student1 = new Student ();
student1.StudtName = 'Danish' ;
student1.StudtRollNo = 26 ;
student1.StudtDegree = 'MSCS' ;
if ((student1.studtRollNo) > 20)
{
    print();
} else
{
    ConsoleDisplay('Roll No is grater than 20');
}
    
```

Figure 7: Structures related visual program’s textual code

5.3 Exceptions

A program module must be reliable and fault-tolerant that can manage different failures in a wide verity of circumstances instead of crashing the application, and it is only possible using the exception-handling capability of a programming language [46]. The exception-handling was developed in Lisp in the 1960s and 1970s. It included both resumption semantics and termination semantics, which provides a mechanism that not only supports error detection but also redirects towards error handling service routines [47]. The anomalous situations require special processing, and developers need to handle it. For this purpose, the block-based Visual Programming Language (BBVPL) accelerates the exception-handling aptitude that plays a vital role dealing with irregular processing and to execute the program in a flow.

BBVPL has introduced a new option in its toolbar, named as “Exceptions”. It provides a “Try” block and a “Throw” block. The blue color button on the “Try” block enables us to include the “Catch” and the “Finally” blocks as much as a programmer need using the drag-drop environment, shown in Figure 8. The “Try” block holds the actual block instructions chosen to execute, while the “Catch” block catches exceptions and accomplishes further alternative instructions. Table 2 is showing exceptions related to visual blocks. “Try” and “Catch” block discussed earlier, while the “Throw” block throws exceptions of various types. “Try Catch”, “Try Finally” and “Try Catch Finally” blocks are also available for software developers.

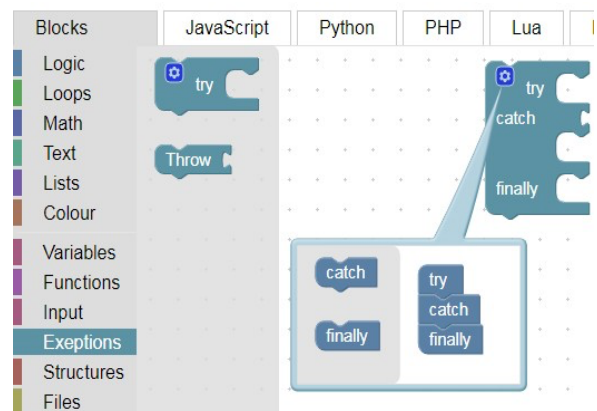




Figure 8: Selected “Exceptions” group

Table 2: List of “Exceptions” related blocks

Sr.	Block name	Image
1	Try	
2	Throw	
3	Try Catch	
4	Try Finally	
5	Try Catch Finally	

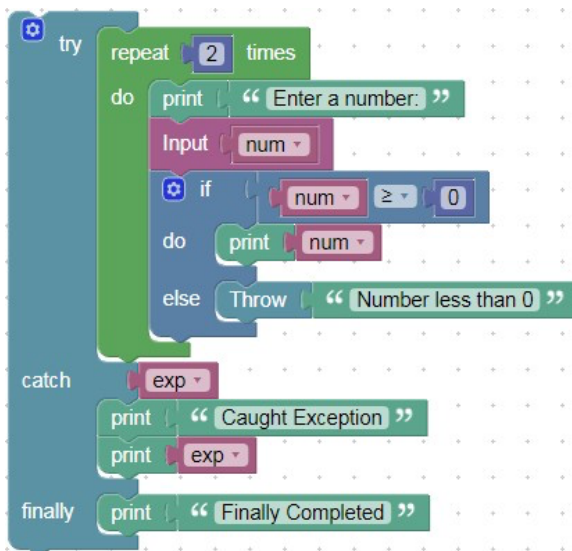


Figure 9: Exceptions related BBVPL’s sample program

A BBVPL’s sample program has presented in Figure 9, in which “try” block executes two iterations of “for loop” and asks a number. If an inputted number is greater than 10, then it prints, otherwise throws an exception message. “Catch” block catches exceptions and prints it. In the end, “finally” block prints “Finally Completed” message. Figure 10 is showing the translated textual code.

```

var exp, num;
try {
  for (var count = 0; count < 2; count++)
  {
    ConsoleDisplay('Enter a number:');
    input(num);
    if (num >= 0)
    {
      ConsoleDisplay(num);
    } else
    {
      throw 'Number less than 0';
    }
  }
}
catch (exp)
{
  ConsoleDisplay('Caught Exception');
  ConsoleDisplay(exp);
}
finally
{
  ConsoleDisplay('Finally Completed');
}
    
```

Figure 10: Exceptions related visual program’s textual code

5.4 Files

A file is a collection of bytes, that keeps data stored permanently on the secondary storage disk. Text files contain alphabet and symbols, while binary files consist of 0’s and 1’s [48]. The four basic operations related to files are opening, closing, reading and writing a file. The programming languages provide different built-in functions in this regard, e.g., fopen(), fclose(), fgets(), fprintf() etc. File handling is the distinctive facility of Block-based Visual Programming Language (BBVPL) that empowers a software developer to read and write contents on file. A simulation-based file system has implemented due to the limitation of JavaScript (unavailability of client-side file handling). Once a developer finishes his work, the complete data collected from the file object, stores on a server location permanently.

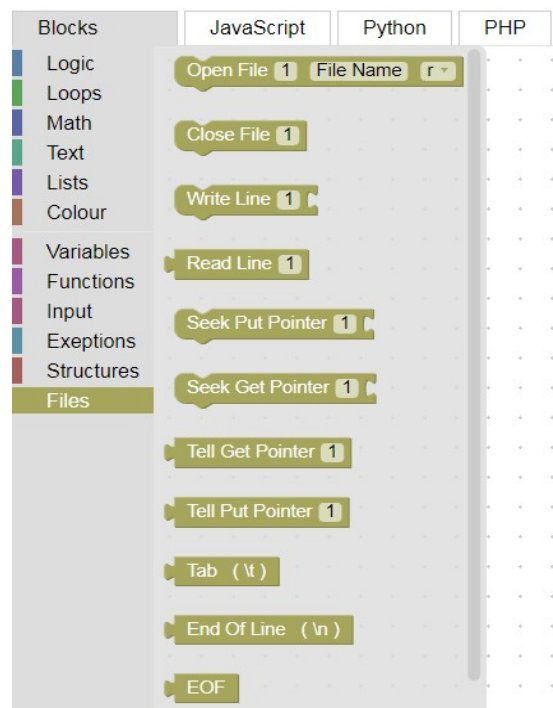


Figure 11: Selected “Files” group

BBVPL allows clicking “Files” option of drag-drop

related visual blocks shown in Figure 11. Table 3 presents all file’s related visual blocks, and the first one in the serial is an “Open File” block made of two inputs and one drop-down field. The placeholder “1” is representing the file descriptor, and its value should be unique, while the placeholder “File Name” represents the file name. The right drop-down field represents file mode, which is already filled with different file creation modes like r, r+, w, w+, a, a+, etc.

Table 3: List of “Files” related blocks

Sr.	Block name	Image
1	Open File	
2	Close File	
3	Write Line	
4	Read Char	
5	Read Line	
6	Seek Put Pointer	
7	Seek Get Pointer	
8	Tell Put Pointer	
9	Tell Get Pointer	
10	Back Space	
11	New Line	
12	Tab	
13	End of File	

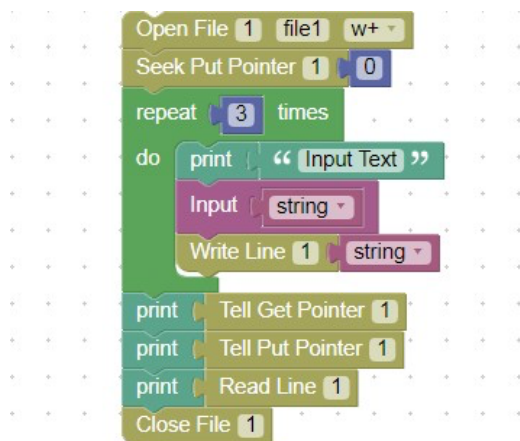


Figure 12: File’s related BBVPL’s sample program

The second block in Table 3 is “Close File” closes related file by putting file descriptor number in the provided placeholder. “Write Line” block to write characters, variables and string values to the desired file. For file reading purposes, BBVPL provides a single character reader and line by line file’s content reading ability using its “Read Char” and “Read Line” blocks accordingly. “Seek Put Pointer” and “Seek Get Pointer” blocks, to demand an integer value to set the pointing

index consequently. On the other side, “Tell Put Pointer” and “Tell Get Pointer” blocks return “Get” and “Put” pointer indexes. BBVPL also competence to deal with special characters, for instance, backspaces, new line, tab, and end of the line. Table 3 is screening “Files” related blocks discussed earlier.

```

var string;
var file1 = new file("file1","w+");
file1.seekp(0);
for (var count = 0; count < 3; count++)
{
    ConsoleDisplay('Input Text');
    input(string);
    file1.fileWrite(string);
}
ConsoleDisplay(file1.tellg());
ConsoleDisplay(file1.tellp());
ConsoleDisplay(file1.fileRead());
file1.fileClose();
    
```

Figure 13: File’s related visual program’s textual code

Figure 12 is viewing a sample visual program in which the “File Open” block opens a file named as “file1” in “w+” mode. The “Seek Put Pointer” is setting the “Put” pointer to 0th index and a loop iterates three times, takes an input string and writes inputted contents into “file1”. The next block instructions print “Put” and “Get” pointing indexes. “Read-Line” block is reading the file’s character until the end of the line (n) character. The textual executable JavaScript code is representing in Figure 13. Finally, the file closed using the “Close File” icon.

6 CONCLUSION AND FUTURE WORK





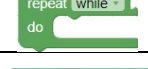


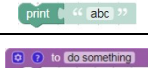
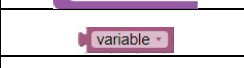

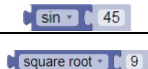





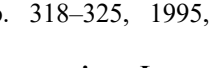
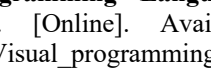
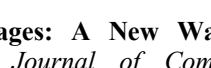
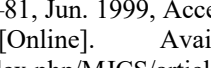
A lot of Visual Programming Languages (VPLs) have been developed for learning and educational purposes and the rest for games and robotic applications development. The existing VPLs lack many fundamental programming capabilities. Teaching programming fundamentals, requires to use a language either a VPL or TPL that can manage arrays or lists creation, modular support, exceptions, file handling, conditions, loops, multithreading, object-orientation, and debugging support. In this paper, a Block-based Visual Programming Language (BBVPL) has been proposed and implemented, based on Google’s Blockly Library that provides a visual drag-drop programming environment to teach basic programming concepts. BBVPL helps to learn vital programming skills, which were discussed earlier. In the future, it will also provide single-step program execution (debugging) and multithreading facilities as well. The user’s roles and rights management will be part of BBVPL in the future.

APPENDIX

Appendix A tabulates the Visual Icons/Features already available in Blockly.

A. Blockly's Existing Visual Icons

Table 4: List of “Blockly’s” related blocks

Sr.	Block name	Image
1	If do	
2	If Else if Else	
3	Counter Loop	
4	For each Loop	
5	While Loop	
6	Is Empty	
7	String Length	
8	Change Case	
9	Prompt For	
10	Print	
11	Function	
12	Variable	
13	Set Variable Value	
14	Change Variable	
15	Trigonometric Functions	
16	Square Root	
17	Return Remainder	
18	Random Integer	
19	Sort	
20	Search in List	

REFERENCES

1. M. Erwig and B. Meyer, “Heterogeneous visual languages - integrating visual and textual programming,” *IEEE Symposium on Visual Languages, Proceedings*, pp. 318–325, 1995, doi: 10.1109/VL.1995.520825.
2. wikipedia.org, “Visual Programming Language.” Accessed: Jul. 28, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Visual_programming_language
3. R. Ahmad, “Visual Languages: A New Way of Programming,” *Malaysian Journal of Computer Science*, vol. 12, no. 1, pp. 76–81, Jun. 1999, Accessed: Jul. 28, 2021. [Online]. Available: <https://ejournal.um.edu.my/index.php/MJCS/article/view/5775>
4. K. N. Whitley, “Visual programming languages and the empirical evidence for and against,” *J. Vis. Lang. Comput.*, vol. 8, no. 1, pp. 109–142, 1997.
5. M. K. Kourouma, “Capabilities and Features of Raptor, Visual Logic, and Flowgorithm for Program Logic and Design,” 2016.
6. P. B. Hansen, “Distributed processes: A concurrent programming concept,” *Communications of the ACM*, vol. 21, no. 11, pp. 934–941, 1978.
7. T. R. G. Green, M. Petre, and others, “Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework,” *Journal of visual languages and computing*, vol. 7, no. 2, pp. 131–174, 1996.
8. E. Tilley and J. Gray, “Dronely: A Visual Block Programming Language for the Control of Drones,” in *Proceedings of the SouthEast Conference*, 2017, pp. 208–211.
9. N. Fraser and others, “Blockly: A visual programming editor,” URL: <https://code.google.com/p/blockly>, 2013.
10. “Google’s Blockly visual programming environment.” Accessed: Jul. 28, 2021. [Online]. Available: <https://www.csee.umbc.edu/2012/06/googles-blockly-visual-programming-environment>
11. E. Pasternak, R. Fenichel, and A. N. Marshall, “Tips for creating a block language with blockly,” in *Blocks and Beyond Workshop (B&B), 2017 IEEE*, 2017, pp. 21–24.
12. “Custom Blocks Blockly Google Developer.” Accessed: Aug. 05, 2018. [Online]. Available: <https://developers.google.com/blockly/guides/create-custom-blocks/overview>
13. I. E. Sutherland, “Sketch pad a man-machine graphical communication system,” in *Proceedings of the SHARE design automation workshop*, 1964, pp. 6–329.
14. C. BurdeaGrigore and P. Coiffet, *Virtual reality technology*. London: Wiley-Interscience, 1994.
15. M. Boshernitsan and M. S. Downes, *Visual programming languages: A survey*. Citeseer, 2004.
16. T. Crews and U. Ziegler, “The flowchart interpreter for introductory programming courses,” in *Frontiers in Education Conference, 1998. FIE’98. 28th Annual*, 1998, vol. 1, pp. 307–312.
17. G. Atanasova and P. Hristova, “Flowchart interpreter: An environment for software animation representation,” in *Proceedings of the 4th International Conference on Computer Systems and Technologies*, 2003, pp. 453–458.
18. D. Hooshyar *et al.*, “A Flowchart-based Multi-Agent System for Assisting Novice Programmers with Problem Solving Activities,” *Malaysian Journal of Computer Science*, vol. 28, no. 2, pp. 132–151, Jun. 2015, Accessed: Jul. 28, 2021. [Online]. Available: <https://ejournal.um.edu.my/index.php/MJCS/article/view/6859>
19. Hooshyar D., Ahmad R.B., Yousefi M., Yusop F.D., and Horng S.-J., “A flowchart-based intelligent tutoring system for improving problem-solving skills of novice programmers,” *Journal of Computer Assisted Learning*, vol. 31, no. 4, pp. 345–361, Aug. 2015, doi: 10.1111/JCAL.12099.
20. M. Thompson, “Evaluating the Use of Flowchart-based RAPTOR Programming in CS0,” 2012.

21. E. Perrin, S. Linck, and F. Danesi, “**AlgoPath: A new way of learning algorithmic.**” 2012.
22. W. Q. Burke, *Coding & composition: Youth storytelling with Scratch programming.* University of Pennsylvania, 2012.
23. “**Microsoft Word - LarpEnV3x.doc**”, Accessed: Aug. 06, 2018. [Online]. Available: <http://www.marcolavoie.ca/larp/en/default.html>
24. J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, “**The scratch programming language and environment,**” *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 16, 2010.
25. N. Zamin, H. Ab Rahim, K. S. Savita, E. Bhattacharyya, M. Zaffar, and S. N. KatijahMohd Jamil, “**Learning Block Programming using Scratch among School Children in Malaysia and Australia: An Exploratory Study,**” *2018 4th International Conference on Computer and Information Sciences: Revolutionising Digital Landscape for Sustainable Smart Society, ICCOINS 2018 - Proceedings*, Oct. 2018, doi: 10.1109/ICCOINS.2018.8510586.
26. A. Reppenng, “**Agentsheets: a tool for building domain-oriented visual programming environments,**” in *Proceedings of the INTERACT’93 and CHI’93 conference on Human factors in computing systems*, 1993, pp. 142–143.
27. Scratch Wiki, “**Scratch.**” <https://en.scratch-wiki.info/wiki/Scratch> (accessed Sep. 06, 2018).
28. Y. Chen, Z. Du, and M. Garcia-Acosta, “**Robot as a service in cloud computing,**” in *Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on*, 2010, pp. 151–158.
29. “**VPL Introduction Microsoft Docs.**” Accessed: Sep. 22, 2018. [Online]. Available: [https://docs.microsoft.com/en-us/previousversions/microsoft-robotics/bb483088\(v=msdn.10\)](https://docs.microsoft.com/en-us/previousversions/microsoft-robotics/bb483088(v=msdn.10))
30. S. Dekhane and X. Xu, “**Engaging students in computing using GameSalad: a pilot study,**” *Journal of Computing Sciences in Colleges*, vol. 28, no. 2, pp. 117–123, 2012.
31. “**Gamesalad gears up for iphone publishing system.**” Accessed: Aug. 22, 2018. [Online]. Available: <http://www.austinstartup.com/2009/09/gamesalad-gears-up-for-iphone-publishing-system/>
32. J. Liu *et al.*, “**Making games a snap with Stencyl: a summer computing workshop for K-12 teachers,**” in *Proceedings of the 45th ACM technical symposium on Computer science education*, 2014, pp. 169–174.
33. wikipedia.org, “**Stencyl.**” Accessed: Jul. 28, 2021. [Online]. Available: <https://en.wikipedia.org/wiki/Stencyl>
34. “**Stencyl: Make iPhone, iPad, Andoid, Windows, Mac, Flash and Html5 games without code.**” Accessed: Jul. 15, 2018. [Online]. Available: <http://www.stencyl.com/features/>
35. “**About EV3- Mindstorms LEGO.com.**” Accessed: Jul. 15, 2018. [Online]. Available: <https://www.lego.com/en-us/mindstorms>
36. J. G. P. Francisco, A. M. Rees, J. Hughes, J. Vermeersch, I. Jormanainen, and T. Toivonen, “**A survey of resources for introducing coding into schools,**” *ACM International Conference Proceeding Series*, vol. 02-04-November-2016, pp. 19–26, Nov. 2016, doi: 10.1145/3012430.3012491.
37. G. Cooper, “**Using Visual Logic with Pseudocode to Teach an Introductory Programming Course,**” *proceedings of WorldComp 2014*, 2014.
38. “**Visual Logic.**” Accessed: Jul. 28, 2021. [Online]. Available: <https://www.visuallogic.org/>
39. G. Shobaki, “**Computer Science Colloquium Series since 1997 Topics, Speakers, and Abstracts during 2010-2015,**” 2015.
40. “**Flowgorithm Flowchart Programming Language.**” <http://www.flowgorithm.org/> (accessed Jul. 28, 2021).
41. N. Fraser, “**Ten things we’ve learned from Blockly,**” in *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*, 2015, pp. 49–50.
42. “**Introduction to Blockly.**” <https://developers.google.com/blockly/guides/overview> (accessed Jul. 28, 2021).
43. wikipedia.org, “**Console Application.**” Accessed: Jul. 28, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Console_application
44. Wikipedia.org, “**Struct (C programming language).**” Accessed: Jul. 27, 2021. [Online]. Available: [https://en.wikipedia.org/wiki/Struct_\(C_programming_language\)](https://en.wikipedia.org/wiki/Struct_(C_programming_language))
45. G. Kiczales *et al.*, “**Aspect-oriented programming,**” in *European conference on object-oriented programming*, 1997, pp. 220–242.
46. B. H. Liskov and A. Snyder, “**Exception handling in CLU,**” *IEEE transactions on software engineering*, no. 6, pp. 546–558, 1979.
47. Wikipedia.org, “**Exception handling.**” Accessed: Jul. 27, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Exception_handling
48. “**File Handling in C language with inbuilt functions.**” <https://fresh2refresh.com/cprogramming/c-file-handling/> (accessed Jul. 27, 2021).