Collaborative Coding in a Robotic Visual Language

Farah Yasser Dawoud, Ahmed Adel and Nada Sharaf
Media Engineering and Technology, German University in Cairo, Third Settelment, Cairo, Egypt

Keywords: Visual Programming, Google Blockly, Collaborative Programming, Real-time Collaboration, Lego Mindstorms EV3 Programming, Block-based Programming.

Abstract: This paper presents a platform, that is a block-based visual programming environment. Students can collaboratively interact using a flexible and versatile definition of visual programming code and interface created using Google Blockly, JavaScript, and Firebase. The web application is designed to allow students to program the Lego Mindstorms EV3. An experiment was conducted to investigate the effect of real-time collaboration on students working on the block-based visual programming web application.

1 INTRODUCTION

Visual programming along with robotic programming have both been evolving rapidly and are proven to have a positive impact on students. They allow them to enhance problem-solving skills and develop different solutions to the same problem. (Danahy et al., 2014). Collaborative tools and technologies facilitate teamwork leading to better results, greater innovation, and higher productivity. After all, collaborative tools cannot be just confined to better sharing of knowledge; collaboration literally means working together, and collaborative tools can improve the speed and effectiveness of people's efforts. Some popular examples of real-time collaborative tools are Google docs, Google sheets, Google Slides, eRoom, and SamePage. Some programming platforms also allow this such as CodeR (Kurniawan et al., 2015) and Overleaf.

The world is going towards a technological revolution, anything that saves time and effort would be very much preferred and appreciated. Visual programming has originally supported individual work where only one person works on a program at a time. However, based on previous studies, collaborative coding has been proven to be more efficient and shows that students working collaboratively outperformed individual programmers (Nosek, 1998). To the best of our knowledge, there exist no real-time collaborative block-based platforms. The aim of the work in this paper is to introduce a methodology to enable real-time collaboration in block-based programming settings. The methodology was applied in a platform to control and simulate the Lego EV3 Robot.

This aim of the work to test the effect of collaborative coding on students by allowing some of the participants to test the application collaboratively through accomplishing a task and others are given the same task to accomplish individually. Aspects including user's engagement, mental load, performance, effort, and frustration levels are analysed for both cases through surveys to investigate the effect of working collaboratively on students.

2 VISUAL PROGRAMMING

Visual Programming reduces the need to write lond textual programs. Program thus have multiple dimensions not only the text (Burnett and McIntyre, 1995). The idea is that with visual constructs, it could have a better learning curve and be more appealing.

For example, with a robotic visual programming language, users can program the robot without writing a textual code and without significant knowledge of any programming language. Examples include Make Block[1] and Open Roberta [2] for programming different hardware systems using blocks.

Visual programming languages have different several classes including block-based, diagrammatic, and iconic visual languages (Myers, 1990).

[1]http://editor.makeblock.com/ide.html
[2]https://lab.open-roberta.org/

Dawoud, F., Adel, A. and Sharaf, N.
Collaborative Coding in a Robotic Visual Language.
DOI: 10.5220/0010455501510156
In *Proceedings of the 13th International Conference on Computer Supported Education (CSEDU 2021) - Volume 1*, pages 151-156
ISBN: 978-989-758-502-9

2.1 Block-based Programming

Block-based languages are a class of visual programming languages where the code constrcuts are blocks that should be connected to each other.

In this case, the programmer has to drag the needed blocks and drop them in the editor and connect them together shown in Figure 1. Blocks usually contain visual hints that eliminate the possibility of having wrong connections or programs.

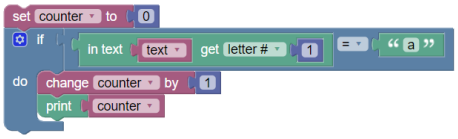

Figure 1: Block Based Program Example.

2.2 Google Blockly

In the search for the best visual editor, and after experimenting with many Visual Programming Languages, it was concluded that Google Blockly[3] was the most preferable option. It offers the most suited flexibility to the need.

Users would consider Blockly as a fun way to build programs. However, on the other side it provides developers with a means to build their own visual blocks and language.

Blockly also allows users to export their programs into well-known languages including Python and Javascript.

Custom blocks were created to satisfy our application's needs and this was achieved through 'Block Factory' provided by Blockly as shown in Figure 2.

The custom blocks that our web application required were mainly the blocks responsible for controlling and programming the Lego Mindstorm EV3 along with some other basic blocks. Lego Mindstorm is a programmable robot based on Lego building blocks (Valk, 2014).

3 COLLABORATIVE BLOCK-BASED PROGRAMMING

While collaborative approaches have been shown to be highly beneficial, implementing real-time collaborative coding feature in the application is not a straight-forward task and requires overcoming several technical challenges.

[3]https://blockly.games/

One challenge of a successful real-time collaborative system is avoiding interference between users by synchronizing workspace changes across all parties. The application's synchronization has to be reliable enough to guarantee constant user access, ensure the right order of applying edits from other users, and avoiding code duplication so that there is constant awareness of the state of the workspaces being edited. Code integration and compilation while others might still be editing source code as well are complicated by the presence of collaborators which necessitates handling syntax errors from edits made by other users. These technical challenges have various solutions that often force trade-offs between ease of implementation and usability.

To be able to track changes of the user's workspace in our application, the following scheme is followed:

1. Track and listen to the stream of events
2. Save them
3. Deliver and fetch by other parties
4. Apply them on other parties' workspaces

3.1 Block Events

Editing code in real-time allows multiple programmers to do changes in the code. Such changes should be automatically reflected for all programmers/collaborators with access to the workspace (Kurniawan et al., 2015). In our application, these changes can vary from being a deletion, creation, update of a block, moving a block across the workspace (dragging and releasing), and connecting it to other blocks or even making a UI change for the arrangement of blocks. All these changes should be tracked and applied to other parties and are all considered '**events**'.

Every change on the user's workspace triggers a certain event where these events fully describe the before and after state of each change. For example, the MOVE event has properties that describe the type of event, workspace ID, block ID, and old coordinates (position) of the block and the new ones to determine where it was moved.

But how do we listen to these events? Blockly workspaces have an *addChangeListener* and *removeChangeListener* methods that are used to listen to the events stream. The event listener works by taking a function ("func" in the example below) as a parameter. This function is executed whenever anything in the workspace changes (i.e whenever an event is triggered).

addChangeListener(func)

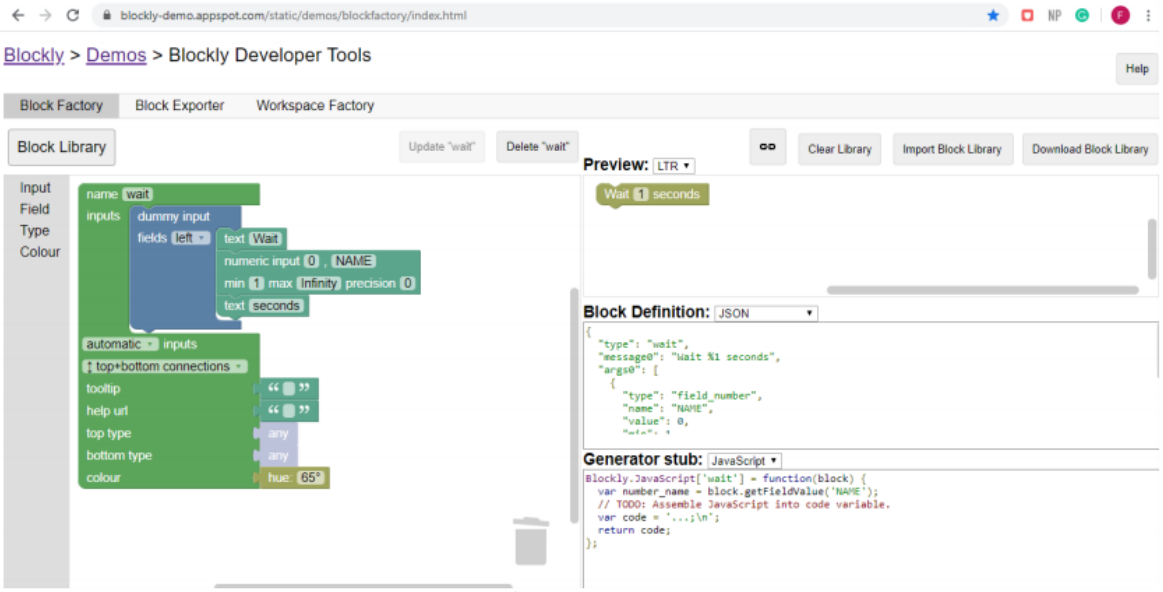

Figure 2: An example of a basic 'wait' custom block created by Blockly factory.

To be able to perform the remaining steps, there has to be a connection and communication method between users and their workspaces to allow real-time data exchange which was achieved through Firebase.

3.2 Firebase

Firebase (Moroney, 2017) was used to implement the real-time collaboration in the application. Firebase provides Cloud Firestore that can store document-structures data. In addition, it is able to synchronize corresponding applications in case of any data change. This consequently means that the application is always listening to the database.

To continue with the Four main steps mentioned above, the code below shows the Listener method and the instructions that need to execute when a change is detected.

```
function listener(change) {
 let workspace = Blockly.getMainWorkspace();

 //converting the event to json then String

 let change_js = change.toJson();
 let change_js_string = JSON.stringify(change_js);

 //adding the event
 //to the database as a string

 var number2 = String(number);
 firebase.firestore()
    .collection('events' + number2).add({
        workspace: workspace.id,
        event: change_js_string,
        timestamp: firebase.firestore.
        FieldValue.serverTimestamp()
    })

 //updating the code of the whole
 //workspace and sending it to simulation

 var code = Blockly.JavaScript.
 workspaceToCode(workspace);
 localStorage.setItem("code", code);
 }
```

We start by saving the event to the database. The event or change is initially of type 'Blockly event'. The Blockly event is first converted to JSON (JavaScript Object Notation), which is a minimal, readable format for structuring data. The JSON object is then converted to a String. Figure 3 shows the example document in Firestore collection.

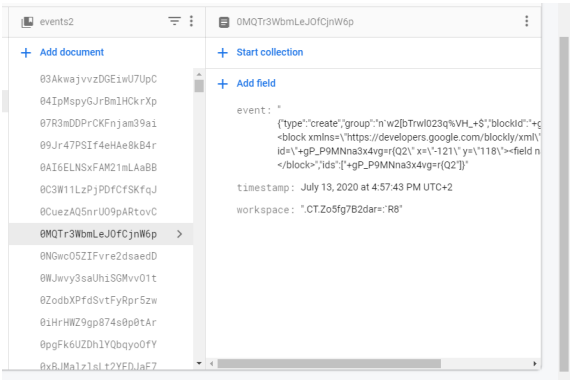

Figure 3: Example event document in Firestore collection.

Once the event is triggered by any of the users, it is fired automatically by the listener function to be saved in the Events collection in Firestore. This ensures achieving step 1 and 2 (Listening to the stream of events and Saving events). However, the application needs to make sure that all workspaces are synchronized with the required state of awareness.

3.2.1 Real-time Updates

This section discusses the etailed steps to achieve the required synchronization. As shown in the *realtimeEvents()* function below, we need to fetch the document, convert the event field from String to JSON then back to a Blockly event. This event can then be applied to other workspaces using blockly_event.*run(true)* command. Cloud Firestore was used for to keep the data synchronized accross the client applications through the realtime listeners.

The *onSnapshot()* method is executed whenever there is a change in the data.

To explain the process with an example, if two users are working collaboratively and user1 created a block, the event is detected by Blockly's event listener and is saved to Firestore by adding a new document of the new event to the Events collection. A document has been added to the collection and a change is now detected by the realtime updates, thus the *onSnapshot()* function is subsequently executed and the event is now applied to other users.

```
function realtimeEvents() {

 let mainWorkspace = Blockly.getMainWorkspace();

    firebase.firestore()
        .collection('events' + String(number))
        .orderBy("timestamp", "asc")
        .onSnapshot(function (querySnapshot) {
querySnapshot.docChanges()
        .forEach(function (change) {

 let workspace = change.doc.data().workspace;
 if (workspace !== mainWorkspace.id) {
    Blockly.Events.disable()

 let event = JSON.parse(change.doc.data().event);
 const blockly_event = Blockly.Events.
 fromJson(event, mainWorkspace)

 try {
    blockly_event.run(true);
    } catch (error) { console.log(error); }

 Blockly.Events.enable()

        } }) })}
```

4 EXPERIMENTAL DESIGN

This section describes in more details the experiment,its design and results. The main purpose was to test the effect of real-time collaboration. In other words, the aim was to test whether it enhances the process, makes it easier, faster, more efficient, more interesting with a less mental demand or not.

4.1 Participants

The experiment was conducted on 10 participants, 7 males and 3 females. Their ages ranged from 9 to 14. All of the participants were children who have previous knowledge and were familiar with block-based programming applications such as Scratch, or have previously been programming the EV3 using the Lego Mindstorms EV3 Home edition. None of them had a history of any health problem.

4.2 Procedure

All of the participants had their experiment on the same conditions and surroundings to be able to evaluate correctly only for the effect of one aspect that is collaboration. They were all given the same set of algorithm to solve written in pseudo-code in which they are required to implement using our block-based application. Some of them were asked to perform the task completely on their own (individually). Others, on the other hand, were asked to finish the task together using the real-time collaboration feature. The participants were then asked to fill a post-experiment survey with their feedback and experience. The survey is divided into three parts; System Usability Scale (SUS) (Brooke, 1996), User Engagement Survey (Pearce et al., 2005; O'Brien et al., 2018; O'Brien and Toms, 2010), and NASA Task Load Index (TLX) workload survey (Hart and Staveland, 1988).

4.3 Statistical Results

Data collected from the post-experience questionnaire of all participants were analysed to calculate the results.

Starting with the first section of the survey, the System usability score provides a useful measure for the usability of a product (Bangor et al., 2008). Final scores for the SUS range from 0 to 100. Higher scores indicate better usabilities.

The mean SUS score of all participants was calculated to be **89.25**.

The total workload scores were analysed as well. Final scores for the Nasa TLX workload survey can

range from 0 to 100. In this case, a higher score indicates s higher workload on the student while performing the task (Colle and Reid, 1998). The average Workload score of students who worked individually was 33.75 while the average of those who used the real-time collaboration feature was 9.4 as illustrated in Table 1.

Table 1: Mean overall workload on students with respect to their group.

Group	Mean overall workload
Collaborative	9.4
Individually	33.8

Results for four different aspects; performance level, effort level, mental demand, and frustration level were analysed for each participant. Mean scores are shown in Table 2 according to their group and final scores can range from 0 to 10.

Table 2: Mean scores of Performance, Frustration, Effort, and Mental Demand on students with respect to their group.

Dimension	Group	Mean score
Performance Level	Collaborative	10.00
Performance Level	Individually	9.00
Frustration Level	Collaborative	1.67
Frustration Level	Individually	4.00
Effort Level	Collaborative	2.67
Effort Level	Individually	6.50
Mental Demand	Collaborative	1.17
Mental Demand	Individually	4.25

Data collected was also analysed to calculate the results of the User Engagement Scale (UES). The UES is a measure of the engagement or depth of investmenent. (O'Brien and Toms, 2010). UES-SF which is a Short form of the UES was used in the post-experiment survey. The final UES score can have values ranging from 0 to 5. The mean UES score for each group is shown in Table 3.

Table 3: Mean UES Score with respect to the students' group.

Group	Mean UES Score
Collaborative	4.3
Individually	2.9

4.3.1 Correlation

There is a **significant negative correlation** between students working collaboratively and the mental demand on them while performing a task with Spearman's correlation coefficient of -0.913, Kendall's tau coefficient of -0.840, and Pearson's coefficient of -0.930.

There is a **significant negative correlation** between collaborative work and effort done in the task with Spearman's correlation coefficient of -0.799, Kendall's tau coefficient of -0.719, and Pearson's coefficient of -0.757.

There is a **significant positive correlation** between collaborative work and performance level in accomplishing the task with Spearman's correlation coefficient of 0.612, Kendall's tau coefficient of 0.612, and Pearson's coefficient of 0.612 as well.

There is a **significant negative correlation** between collaborative work and frustration level in accomplishing the task with Spearman's correlation coefficient of -0.761, Kendall's tau coefficient of -0.722, and Pearson's coefficient of -0.764.

There is a **significant negative correlation** between collaborative work and total workload on participants while accomplishing the task with Spearman's correlation coefficient of -0.853, Kendall's tau coefficient of -0.730, and Pearson's coefficient of -0.891.

There is **no significant correlation** between collaboration and system usability. Spearman's correlation coefficient is 0.434, Kendall's tau coefficient of 0.387, and Pearson's coefficient of 0.5.

There is a **significant positive correlation** between collaborative work and user engagement while accomplishing the task with Spearman's correlation coefficient of 0.775, Kendall's tau coefficient of 0.705, and Pearson's coefficient of 0.747.

Table 4 shows a summary table for all factors with Spearman's, Kendall's, and Pearson's correlation coefficient values along with the p-value (significance value) of Pearson's correlation.

5 CONCLUSIONS AND FUTURE WORK

This study was performed to investigate the effect of real-time collaborative programming on the process by investigating several aspects including effort, performance, engagement, and mental workload. An experiment was conducted on a representative sample of 10 normal children (7 males and 3 females) who

Table 4: Correlation coefficient and significance values for all factors.

Factor	Spearman's	Kendall's	Pearson's	p-value	Significant correlation
User Engagement	0.775**	0.705*	0.747**	.006	YES
System Usability	0.434	0.387	0.500	.042	NO
Total Workload	-0.853**	-0.730**	-0.891**	.000	YES
Performance Level	0.612*	0.612*	0.612*	.030	YES
Frustration Level	-0.761**	-0.722*	-0.764**	.005	YES
Effort Level	-0.799**	-0.719**	-0.757**	.006	YES
Mental Demand	-0.913**	-0.840**	-0.930**	.000	YES

**. Correlation is significant at the 0.01 level

*. Correlation is significant at the 0.05 level

are familiar with block-based visual programming applications. They were given a task to accomplish in a set time interval. Results showed users working collaboratively expressed higher levels of confidence about their work, engagement to the task, less levels of frustration, mental demand, and total workload, and higher performance level while accomplishing the task than programmers who worked and performed the task alone. This was concluded by calculating the SUS, NASA TLX workload, and engagement scores then calculating the correlation and regression.

In the future, working on a larger sample size would help in getting more reliable and accurate results. It would be better to a large number of participants for each age and gender. In addition, making sure all participants have approximately the same years of experience in using block-based applications.

REFERENCES

Bangor, A., Kortum, P. T., and Miller, J. T. (2008). An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6):574–594.

Brooke, J. (1996). Sus: a "quick and dirty'usability. *Usability evaluation in industry*, 189.

Burnett, M. M. and McIntyre, D. W. (1995). Visual programming. *COMPUTER-LOS ALAMITOS-*, 28:14–14.

Colle, H. A. and Reid, G. B. (1998). Context effects in subjective mental workload ratings. *Human factors*, 40(4):591–600.

Danahy, E., Wang, E., Brockman, J., Carberry, A., Shapiro, B., and Rogers, C. B. (2014). Lego-based robotics in higher education: 15 years of student creativity. *International Journal of Advanced Robotic Systems*, 11(2):27.

Hart, S. G. and Staveland, L. E. (1988). Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier.

Kurniawan, A., Soesanto, C., and Wijaya, J. E. C. (2015). Coder: Real-time code editor application for collaborative programming. *Procedia Computer Science*, 59:510–519.

Moroney, L. (2017). The firebase realtime database. In *The Definitive Guide to Firebase*, pages 51–71. Springer.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing*, 1(1):97–123.

Nosek, J. T. (1998). The case for collaborative programming. *Communications of the ACM*, 41(3):105–108.

O'Brien, H. L., Cairns, P., and Hall, M. (2018). A practical approach to measuring user engagement with the refined user engagement scale (ues) and new ues short form. *International Journal of Human-Computer Studies*, 112:28–39.

O'Brien, H. L. and Toms, E. G. (2010). The development and evaluation of a survey to measure user engagement. *Journal of the American Society for Information Science and Technology*, 61(1):50–69.

Pearce, J. M., Ainley, M., and Howard, S. (2005). The ebb and flow of online learning. *Computers in human behavior*, 21(5):745–771.

Valk, L. (2014). *Lego mindstorms Ev3 Discovery Book: A beginner's guide to building and programming robots*. No Starch Press.