Autonomous Learning Intelligent Vehicles Engineering in a Programming Learning Application for Youth: ALIVE PLAY

Jihene Rezgui, Félix Jobin, Simon Beaulieu, Zarine Ardekani-Djoneidi
Laboratoire **R**echerche **I**nformatique **M**aisonneuve (LRIMa)

Montreal, Canada
jrezgui@cmaisonneuve.qc.ca

Abstract – The world of programming is taking a more and more important place in our lives, yet this field is a mystery for many people. Due to its appearing complexity, the majority of young students have many difficulties in learning programming and the tools for this purpose can be less attractive as they should. This paper seeks to raise this issue in learning programming applications and the lack of a transition between an easy block-based interface to a common programming language. It also proposes Autonomous Learning Intelligent Vehicles Engineering in a Programming Learning Application for Youth, called ALIVE PLAY to arouse the interest of young students by letting them interact with a physical vehicle. Additionally, it contains two distinct interfaces to suit the needs of all ages, including a new built-in programming language, named AliveScript that helps the student to make his way from block-based programming to true programming. Preliminary results demonstrate the effectiveness of ALIVE PLAY in terms of young students' interest, curiosity and satisfaction. In fact, they would want to explore more aspects related to the vehicle, and this is why we hope that this project reaches many schools in the near future.

Keywords: ALIVE PLAY, controlling physical vehicle, programming, block-based, AliveScript, WIFI.

I. Introduction

The recent pandemic has significantly changed the education field, forcing graduate schools to set aside traditional classrooms and prioritize online courses. Multiple logistic issues have arisen from this adaptation for most of the systems that were not already computerized. Teachers tried to modify their courses at their best, but the tools at their disposal are not always suited to their needs. On the other hand, specifically in the programming field, it can be hard for elementary and high schools to find the best approach to make students discover this domain, yet it has become the core of most systems of our society and needs more specialists than ever. Of course, many platforms are already available on the Internet, and some of them can really facilitate the journey of a beginner. However, as far as we are concerned, presently, there is no affordable application that tries to learn programming by using IoTs, such as connected vehicles, to enhance the student's enjoyment and curiosity. Our ALIVE PLAY project [1] aims to be the link between the two fields. It is a programming learning application addressed to young students, from elementary schools to high schools, connected to an IoT object, more precisely a vehicle from the ALIVE research project [2-4], that can do actions according to a simplified code written by the user. We want to offer students a simple and attractive way to discover the programming world without having to learn any traditional programming language, which could seem to be too complicated at this age. Instead, younger students can experiment their first program with the block-based interface while older and more experienced students have access to a more difficult challenge with the AliveScript interface, a new programming language specifically designed to be a transition step before real programming.

With our ALIVE PLAY project, we have achieved improvements on programming learning applications targeting youth in ways that are explained in the following section.

Our contributions in this paper can be summarized as follows: (1) We introduce a new JAVA application called "ALIVE PLAY" and its multiple functionalities; (2) We review related platforms that guided us in the development of our project and helped us understand the best ideas and the possible improvements; (3) We highlight the advantages of the improvements made with ALIVE PLAY and its IoT compatibility, compared to other platforms; (4) We present a new way to learn programming by controlling a physical ALIVE car with easy coding methods; (5) We develop block-based programming, suited for both elementary and high school students, and a new programming language, AliveScript, adapted specifically for high school students, and (6) we explore our main avenues for ALIVE PLAY's future, such as creating an online programming and AI course platform using the tools provided by this project and his IoT compatibility.

Section II gives a brief overview of related applications and compares them to our ALIVE PLAY project. In section III, we describe our whole application and the two programming methods we have developed. In section IV, we explain in detail the AliveScript language and its place in our application. Section V provides information about the physical car and the simulation implemented in the application. Section VI explores different possibilities to make improvements in the project, for instance creating an online programming course platform integrating this project. Section VII concludes the paper.

II. Related Works and Comparison

Several research projects [5-12] will be explained and compared to ours. We will be highlighting the differences between the two works and demonstrating their advantages and disadvantages.

A. Scratch [5] vs ALIVE PLAY

Scratch is a well-known project that focuses on learning the basics of programming and has been created by the Lifelong Kindergarten Group at the MIT Media Lab. Its interface allows the user to create a simple program by using blocks to facilitate the concepts of programming. Many challenges guide the young user to help him discover the multiple possibilities of this platform, from moving a 2D object to interact with physical objects.

Despite the fact that the interface of Scratch is more developed than ALIVE PLAY's block-based interface, the main difference between the two projects is that ALIVE PLAY offers a code-based interface using an simpler programming language, AliveScript, with the bloc-based interface, so that the student can make a better transition towards a more advanced programming language like Python and Java. In addition, having both interfaces allows us to reach more students from any age and skill level. Moreover, our project provides a physical car that was made to interact with these interfaces which is easier to connect than an external object from a different team.

B. Codecademy [6] vs ALIVE PLAY

Codecademy is a great education platform to learn and gain experience in programming by allowing the user to experiment in a code-oriented platform while learning the subjects. By following each step in their courses, the student will understand many basic concepts in programming as well as some of the most common programming languages.

The disadvantage of Codecademy is that it teaches the concepts of programming via a real language. In fact, it can be hard for a new learner to fully understand the concepts within the language because depending on the one the student wants to learn, they can have a various difficulty level. ALIVE PLAY proposes an easier approach where the language, AliveScript, is simplified, so that the user can have a better understanding of the subjects. In addition, our project includes a physical car, so the user can see a more concrete result of his program, which is often more encouraging than seeing numbers or letters in the console. Another advantage is the block-based interface that we offer to help younger users to learn the basics of programming, which, unfortunately, Codecademy does not have.

C. Other projects vs ALIVE PLAY

In our research for other projects like ours, we found that the amount of applications that are available to learn basic programming is large on the web. freeCodeCamp [7] and W3Schools [8] are really similar to Codecademy by their exercice-oriented interface; Pluralsight [9], One Month [10] and Udemy [11] dispense high quality courses but not for free; and Code.org [12] is highly addressed to young students with their interactive and captivating block-based interface, just like Scratch. However, we noticed that none of them had a physical car or any kind of connected object to appreciate the results of the user's program. In addition, our project provides both the block-based and the code-based interface with a simplified language. This versatility allows the ALIVE PLAY project to satisfy a larger group of people than the other applications.

III. ALIVE PLAY Project

Our project is an application of the ALIVE project in the educational field. It is a user-friendly interface where the student can learn basic concepts of programming by controlling one of the ALIVE physical cars with a Bluetooth connection or the car displayed on a built-in simulation. For now, the whole project and the AliveScript language are in French, because we want to aim at a local public at first, which, for instance, is the province of Quebec. However, we will surely make a translating option in the application to allow English-speaking students to easily use our platform. In this section, we will present the two interfaces that we have implemented, resulting in two ways of sending commands to the vehicle, with two distinct levels of difficulty (see **Fig. 1** below).

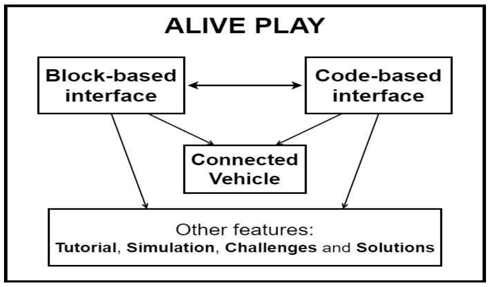


Fig. 1 The different aspects of ALIVE PLAY

A. Block-based Interface

Firstly, the block-based interface allows the student to make a clear and simple code by placing command blocks below each other. This interface is shown below in **Fig. 2**. When executing the code, the program will simply execute each command block, starting by the highest one and following the order of the blocks.

This interface seeks to arouse the interest of elementary school students who have never done programming before. With a simple way to learn the basics of programming, young students can explore this high-demanded field and have a better understanding of it.

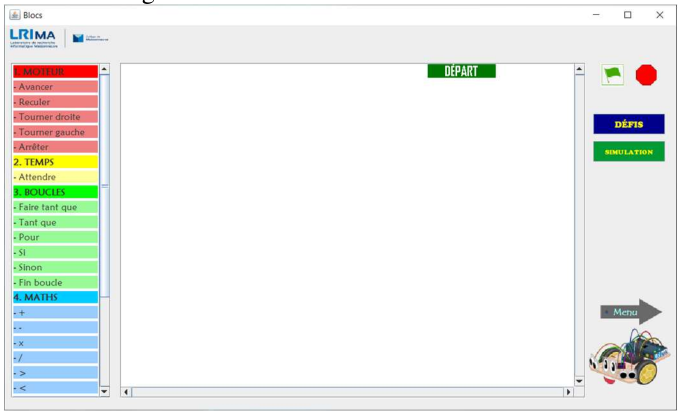


Fig. 2 Block-based interface

There are five categories of blocks. Firstly, the engine blocks allow the user to send instructions to the vehicle, whether to make it travel for a given time or turn towards a direction. In addition, the time block "Attendre" (wait), which is the second block category, makes the program stop for the given number of seconds. The third category, loops and conditions, contains blocks that surround other blocks. It is therefore easy to see which instructions are contained in the loop or the condition. The block languages supports the most common loops in programming languages such as *while*, *do/while* and *for*. It also allows conditional statements, which gives the opportunity to run specific blocks when a condition is met.

The fourth category, math blocks, allows the user to do math operations and comparisons, from additions to verify if a value is greater than another. It also contains the blocks "et" (and) and "ou" (or), which are used to link two booleans values. Finally, the block "Aléatoire" (random) generates a random number between the two given numbers. The variable blocks are the fifth category. They allow the user to create and manipulate variables. The variables' type is not specified in the block-based interface. Some blocks are shown in **Fig. 3** below.

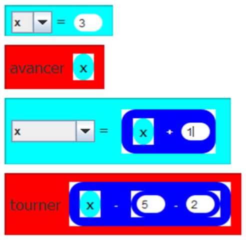
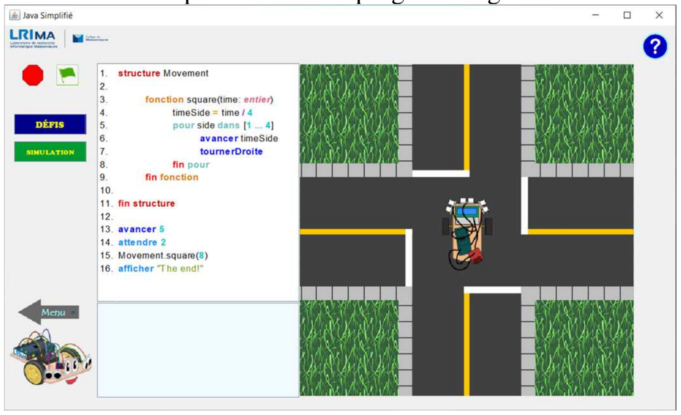


Fig. 3 Example of blocks in the block-based interface

There are several advantages to using a block-based interface for beginners. First, it makes a visual and therefore easily understandable code for someone who is not familiar with programming. While written code may seem austere and confusing, blocks are more welcoming and intuitive. The various shapes and colors make programming clear and attractive. In addition, it is faster to drag and drop blocks than typing code, especially if the user is not completely comfortable with a keyboard yet. Moreover, with the block list on the side, it is easy to have an overview of all available blocks. Finally, although some mistakes can still be made, large parts of programming mistakes are avoided, like spelling mistakes. Being able to produce a functioning code encourages beginners and gives them the confidence they need to later use the code-based interface.

B. Code-based Interface

Secondly, the code-based interface allows the student to type code directly in a text area (see **Fig. 4** below). This interface offers a much closer experience to real programming.



```
structure Movement

    fonction square(time: entier)
        timeSide = time / 4
        pour side dans [1 ... 4]
            avancer timeSide
            tournerDroite
        fin pour
    fin fonction

fin structure

avancer 5
attendre 2
Movement.square(8)
afficher "The end!"
```

Fig. 4 Code-based interface, with simulation visualised

The code-based interface is suitable for secondary school students and for those who, after trying the block-based interface, seek more challenges. It is harder to master, but it covers a wider range of programming concepts and it leaves more room for creativity. It allows the user to write programs which, because of their complexity, could not be created with blocks.

This interface offers more features than the block-based interface. In addition to the instructions found in the blocks, it allows the user to make comments, create functions, create structures and print text in a console. It also offers built-in methods. The features of AliveScript, the language used in the code-based interface, are further developed in section IV.

As the user types code, some words are automatically formatted in order to increase readability. Besides making the code pleasant to watch, the various colors make the role of each instruction understandable at a glance. This feature is a major asset in making programming attractive for youth.

The code-based interface has multiple advantages. It accustoms the student to the operating mode of real programming and it offers more possibilities. It also confronts the user with more potential errors, which leads to the development of error resolution strategies. It provides additional difficulty to keep the fastest learners interested. Finally, writing code that looks like real programming generates a feeling of satisfaction in the student.

IV. AliveScript Language

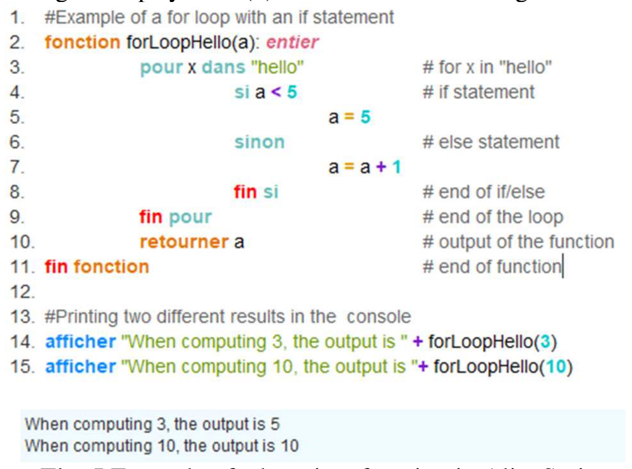
AliveScript is a new programming language specifically designed to be used in our code-based interface. It makes the bridge between block-based programming and true programming to allow a progressive and simple learning of key concepts in this field, like variables, conditions, and loops. Therefore, this programming language has been made to be as easy as possible for high-school students.

A. Core Features of AliveScript

The main difference between AliveScript and the other programming languages is the presence of built-in methods to move the physical vehicle as well as the simulated vehicle. These methods work the same way as the engine blocks in the block interface, but can also work with all features included in AliveScript, as they are part of the language.

B. Similarities to Other Programming Languages

AliveScript has many similarities to other programming languages, which allow the student to make a comprehensive transition between block-based programming and real programming. Because of the potential complexity of variable types (integers, floats, strings) we decided to make it dynamically typed, so the students do not have to worry about type declaration. The elementary mathematical operations are the same as many other languages, as they are already understandable for most people, even at a young age. It is also possible to code loops, functions, and conditional statements. On the other hand, loops, conditional blocks and functions have a similarity which is their ending. The keyword “fin” (end) followed by the name of the code block helps the student see which part of his code ends at a specific place, leading to a better understanding of the general programming syntax. The user can also comment his code by inserting a sharp symbol (#) before the commenting line.



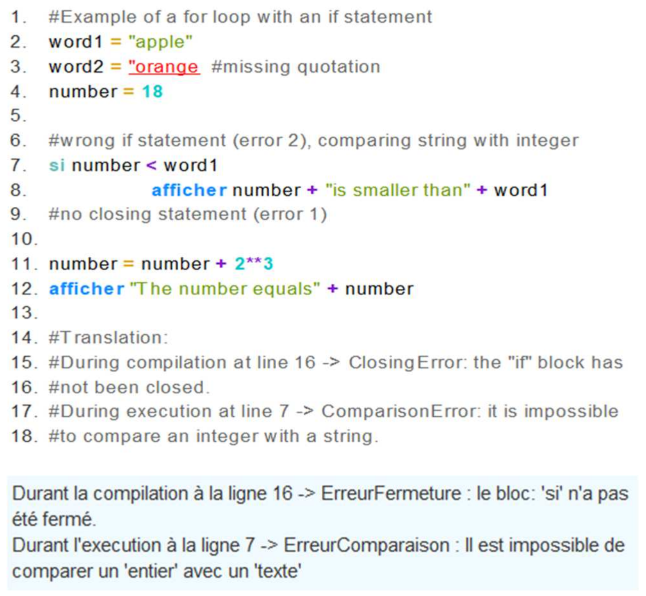
```
#Example of a for loop with an if statement
fonction forLoopHello(a): entier
        pour x dans "hello"                 # for x in "hello"
                si a < 5                    # if statement
                        a = 5
                sinon                       # else statement
                        a = a + 1
                fin si                      # end of if/else
        fin pour                            # end of the loop
        retourner a                         # output of the function
fin fonction                                # end of function

#Printing two different results in the  console
afficher "When computing 3, the output is " + forLoopHello(3)
afficher "When computing 10, the output is "+ forLoopHello(10)
```

```
When computing 3, the output is 5
When computing 10, the output is 10
```

Fig. 5 Example of a loop in a function in AliveScript

We made AliveScript in a way that loops and functions can be used in the most intuitive manner by a beginner user, but not far from common programming languages. For instance, a lot like Python, it is possible to loop through all the letters of a string variable (named “texte” (text) in AliveScript) as well as through elements of a list (see **Fig. 5** above). The use of functions has also similar features from Python such as the option to give or not a type and a default value to a parameter of a custom function. Those features have been chosen while keeping in mind that students need an easy programming language where the code is simple and easy to read.



```
1.  #Example of a for loop with an if statement
2.  word1 = "apple"
3.  word2 = "orange  #missing quotation
4.  number = 18
5.
6.  #wrong if statement (error 2), comparing string with integer
7.  si number < word1
8.          afficher number + "is smaller than" + word1
9.  #no closing statement (error 1)
10.
11. number = number + 2**3
12. afficher "The number equals" + number
13.
14. #Translation:
15. #During compilation at line 16 -> ClosingError: the "if" block has
16. #not been closed.
17. #During execution at line 7 -> ComparisonError: it is impossible
18. #to compare an integer with a string.
```

```
Durant la compilation à la ligne 16 -> ErreurFermeture : le bloc: 'si' n'a pas
été fermé.
Durant l'execution à la ligne 7 -> ErreurComparaison : Il est impossible de
comparer un 'entier' avec un 'texte'
```

Fig. 6 Example of error reports in an AliveScript code

AliveScript can also find and report errors in a program, as shown in the **Fig. 6** below. If the code made by the user with AliveScript has syntax errors in it, the application will highlight where the error is in red, so the user is aware of the problem. If an error occurs while compiling or executing the code, a message will be displayed in the application’s console where the user can know at which line has occurred the error and what exactly is the problem of that line. This allows the user to understand his mistakes and correct them easily.

C. Built-in Methods

Table. 1. Examples of built-in methods in AliveScript

Built-in method	Description
Math.sin(angle)	Returns the sinus of an angle, in degrees
Math.PI	Returns the constant π
entier(string)	Returns a string converted to integer if possible
tailleDe(string|list)	Returns the length of a string or a list
info(variable)	Returns info concerning the variable (type, value, etc.)

Finally, we added several built-in methods that can be easily used by students. They are additional tools to help the user create more complex programs in the AliveScript language. **Table. 1** displays some of the methods already implemented in the AliveScript language. Note that this list will continue to grow to give students more useful tools to let them go further in programming.

V. Connected Physical Vehicle and Built-in Simulation

To enhance the interest of young students into the programming field, we thought that showing them the result of their code in an amusing manner would be the best solution. It is in this state of mind that we tried to adapt the vehicle from the ALIVE project to suit the needs of students from elementary schools to high schools.

A. The Physical Vehicle

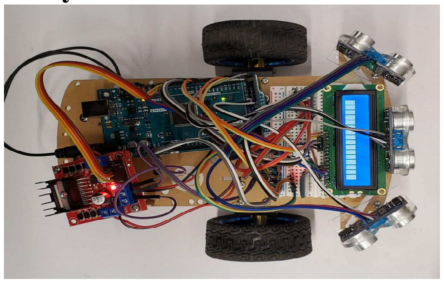


Fig. 7 ALIVE physical car

The ALIVE pedagogic project has been made to be compatible with the physical vehicle from the ALIVE research project, as shown in **Fig. 7**. The purpose of this car in our application is to execute the commands that will make it move towards a direction, given by the AliveScript program or the block-based program. This connection is made by a Bluetooth connection between the Bluetooth module of the car and the machine where the application is running. When the application executes the code written by the student, it sends a specific value every time it passes on one of the five vehicle methods, which makes the car execute the action ordered by the method. While running a code in the pedagogic ALIVE application, the vehicle simply listens to incoming values and reacts according to them until the end of the code.

This new way to learn programming can become very attractive and amusing for young students. In fact, having a vehicle that can react in accordance with the code they created by themselves could improve the students understanding of basic programming knowledge while they try to achieve the multiple challenges. Furthermore, the game-like aspect of the application could create enjoyment for students and encourage them to upgrade their code with the tools given by the application. In a time when programming is all around us, which makes the demand increase significantly and when this field can stay blurry and mysterious for a high school student, the education program should have the best tools to make our young but brilliant generation discover the world of programming and his infinite possibilities.

B. The Simulation

The pedagogic ALIVE project also has a built-in simulation. It becomes an alternative to the physical vehicle when the user does not own one and can be an additional visualisation of the vehicle responding to the code when the physical car is responding to it at the same time, as shown in **Fig. 8**.

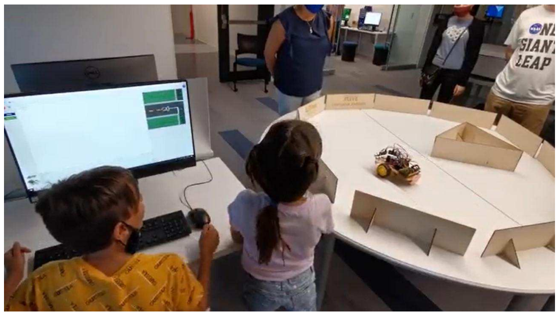


Fig. 8. ALIVE car responding to a program while the simulation is running

In its basic state, the simulation represents a classic four-ways intersection, with four two-ways roads that seem to continue beyond the window. As we can see in **Fig. 9**, in this setup, we have placed the representation of the car at the center of the intersection.

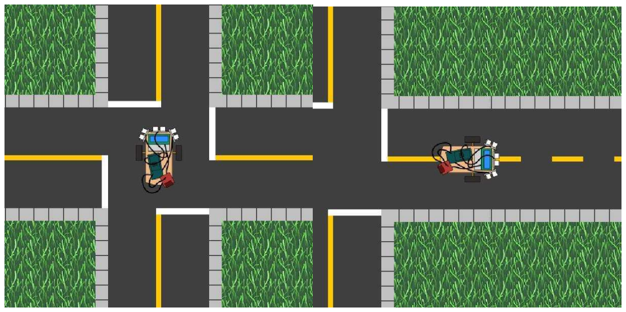


Fig. 9. Initial and moving setup of the simulation

When the user runs the application, the simulation follows the same vehicle methods in the code as the physical car. In the current version, the car cannot get out of the road as the simulated world adapts itself with the car's movements. This solution has been kept because its first objective is to be another visual support to see the progression of the car, along with the physical ALIVE car. This way, students can enjoy the application without owning the real vehicle.

C. The Results

To put our application to test, we have asked young students, who were complete beginners in programming, to use our project in order to make the physical vehicle and the simulated vehicle do specific movements. Because of the actual pandemic, we did not have the possibility to try ALIVE PLAY on a large group of students, but those who have tested it gave us a good idea of the potential success of our application.

We gave to a couple of elementary and high school students the task to make the physical vehicle move in specific ways. As shown in the **Fig. 10**, there were three different scenarios with distinct difficulties. We asked the students to do the tasks from the easiest, in green, to the hardest, in red, and we noted if they succeeded in the task or not, as well as their satisfaction regarding their accomplishment and the time required.

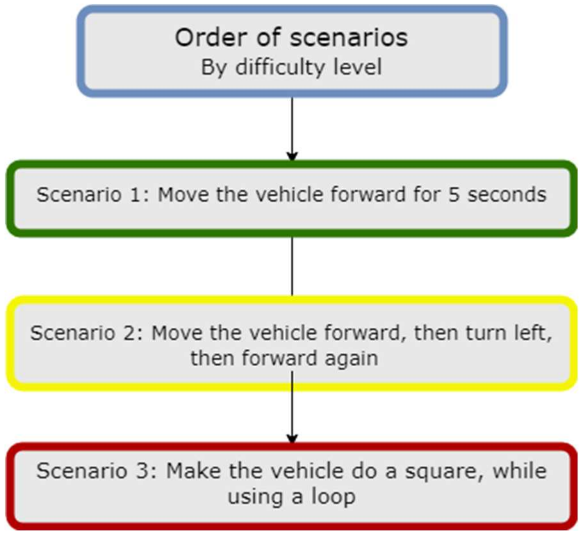

Order of scenarios
By difficulty level
Scenario 1: Move the vehicle forward for 5 seconds
Scenario 2: Move the vehicle forward, then turn left, then forward again
Scenario 3: Make the vehicle do a square, while using a loop

Fig. 10. The scenarios given to students

As we can see in **Table 2**, elementary school students have succeeded in all scenarios, except the third.

Table. 2. Statistics about scenarios on elementary and high school students (AliveScript interface)

Task	Difficulty	Education level	Success	User Satisfaction	Time required
Scenario 1	Easy	Elementary	Yes	High	30 s
		High school	Yes	High	15 s
Scenario 2	Medium	Elementary	Yes	High	1 min
		High school	Yes	High	1 min
Scenario 3	Hard	Elementary	No	High	+10 min
		High school	Yes	High	3 min

In fact, this last scenario contained loops, making it the hardest. It is therefore understandable that young students have encountered difficulties while trying to do this task. However, the students have manifested interest in doing more challenges. They enjoyed the experience and were amazed by the movement of the physical car. On the other hand, high school students have succeeded in all tasks, which demonstrates the effectiveness of our method. These older students can then try out more complex scenarios, which will allow them to learn more about programming.

VI. Software Upgrade Possibilities

We are aware that many upgrades can be made to make the project even more interesting for schools. As mentioned earlier in this paper, we are working on other features that will make the user learn the basics of programming and artificial intelligence while he can experiment the theory immediately by using the built-in interface of ALIVE PLAY. We will keep the possibility to choose between the block-based interface and the AliveScript interface, so the user can choose the one that better suits their needs. In addition, we will add a "sandbox" mode where the user can experiment with any of the two interfaces or both at the same time, and modify the simulation by adding obstacles in the simulated car's environment.

Finally, it could be very useful to add the possibility to use the vehicle's sensors. The ALIVE car is equipped with ultrasound sensors, which allow it to calculate the distance of the nearest obstacle in front of it. The use of these sensors in our application permits movements based on distances instead of time, offering a new way to command the car.

VII. Conclusion

To conclude, the ALIVE PLAY project proposed a new programming learning approach, which we compared to some other similar platforms. We proved that our project had better chances to reach the interest of youth by providing interfaces that can raise their attraction to programming. In addition, we are the only project that we found letting the user control a physical car by writing a simplified program in an accessible language. We aim to try out our project in local schools to see if the platform arouses the interest of teachers and students. However, we have explained possible improvements on the application to further develop the software and its functionalities, such as providing a less limited simulation to increase its functionalities.

ACKNOWLEDGMENT

We would like to thank Enric Soldevila, Mathis Laroche, Gabriel Landry and Éloi Vincent-Légaré, members of LRIMa for their valuable help in coding this application. This project would have never been as important as it is without their ideas and talent.

References

[1] F. Jobin, S. Beaulieu, G. Landry, Z. Ardekani-Djoneidi, É. Vincent-Légaré, M. Laroche and J. Rezgui on gitlab, https://gitlab.com/sim.beaulieu10/ alivepedagogique [last visited April 13th, 2021].

[2] J. Rezgui, É. Gagné and G. Blain "Autonomous Learning Intelligent Vehicles Engineering (ALIVE 1.0)", IEEE ISNCC 2020.

[3] J. Rezgui, É. Gagné, G. Blain, O. St-Pierre, M. Harvey and S. Charkaoui, "Open Source Platform for Extended Perception Using Communications and Machine Learning on a Small-Scale Vehicular Testbed", IEEE GIIS 2020.

[4] J. Rezgui, É. Gagné, G. Blain and O. St-Pierre, M. Harvey, "Platooning of Autonomous Vehicles with Artificial Intelligence V2I Communications and Navigation Algorithm", IEEE GIIS 2020.

[5] the Lifelong Kindergarten Group at the MIT Media Lab. https://scratch.mit.edu/ [last visited April 13th, 2021].

[6] Codecademy. https://www.codecademy.com/courses/welcome-to-codecademy/lessons/ [last visited April 13th, 2021]

[7] freeCodeCamp. Basic HTML and HTML5. Say Hello to HTML elements. https://www.freecodecamp.org/learn/ responsive-web-design/basic-html-and-html5/say-hello-to-html-elements.

[8] W3Schools. HTML Tutorial. https://www.w3schools.com/html/default.asp

[9] Pluralsight. Core Python: Getting Started.. https://www.pluralsight.com/courses/getting-started-python-core

[10] One Month. Learn Python. https://onemonth.com/ courses/python [last visited April 13th, 2021].

[11] Udemy. Learn Python Programming Masterclass. https://www.udemy.com/course/python-the-complete-python-developer-course/ [last visited April 13th, 2021].

[12] Code.org. Leçon 4: Programming with Angry Birds. https://studio.code.org/s/courseb-2020/stage/4/puzzle/2 [last visited April 13th, 2021].