Virtual Reality Project Report
Hello VWorld, Group 1

Cristiana Antunes, Francisco Nicolau, Manuel Correia and Miguel Valério

Abstract—Programming is becoming more and more of a necessary skill for all kinds of careers, as such, the age of contact with these concepts is trending down and a greater percentage of people are expected to learn how to code. This leads to a greater necessity for an effective introduction that will keep the users engaged and interested in learning more. Although some efforts to create interactive experiences that teach this skill have been made, very few try to use *gamification* or even leverage the benefits of emergent technology, such as Virtual Reality, to prevent the feelings of disinterest and demotivation often felt by beginners with such complex topics. To mitigate these issues we present Hello VWorld, a VR puzzle game, designed to be the first contact with programming concepts for total beginners. VR allows us to create an environment with great potential for learning, being able to keep the user immersed and focused all the while providing great accessibility through it's natural and familiar motion controls. The goal of Hello VWorld is to teach the fundamental thought processes required and motivate further learning ultimately providing a smooth introduction to this area of knowledge.

Index Terms—Virtual Reality, Didactic Entertainment, Interaction Techniques

1 Introduction

1.1 Motivation

The Virtual Reality space presents an opportunity for augmented learning experiences as it allows for the use of the three-dimensionality to better convey complicated concepts. Programming is a skill which involves several complicated concepts and it has been shown learning it benefits from more interactive mediums. Physical toys in particular, appear popular with younger audiences but are expensive and ultimately limited by the number of components included. In a VR environment a middle ground may be achieved, where the lack of physicality can take away some benefits but still be a boon.

- *Cristiana Antunes, ist186398.*
- *Francisco Nicolau, ist186419.*
- *Manuel Correia, ist18470.*
- *Miguel Valério, ist18483.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

1.2 Problem

How can we make a more interactive learning tool to better retain student interest and focus, when learning programming?

1.3 Hypothesis

The properties of VR environments facilitate the learning of some skills by promoting engagement and focus, along with removing limitations of physical circumstances.

1.4 Objectives

This work lays out the system, a VR game, created by us to build an environment that facilitates the learning of programming. With this in mind, we endeavored to examine works on pseudo-programming, from 2D virtual environments to physical games and continuing to projects similar to our own. All of which informed our design choices which we sought to lay out here.

1.5 Outline

This report begins with an introduction that presents our goals of teaching programming in a Virtual Reality setting. Following this, the second section focuses on works of both pseudo-programming and virtual reality which we analised to inform our own work, which in turn is we explain over sections 3 and 4. After these, section 5 is a brief discussion of the work we did, complemented with what work should in the future be carried out to continue on this topic, following which are our conclusions. To end, after our references, an appendix can be found going over the steps to add content to the systems we created.

2 Related Work

When elaborating the concept for this project we knew that we wanted to tackle the topic of learning in an immersive environment, something that is unique to VR and that has been shown to have great benefits. To better understand this problem and draw some inspiration we looked at several papers, researched existing solutions and identified what we believed to be the key obstacles in the way of progress for this specific application. Moving forward we will analyze some of these papers and make observations with the goal of maximizing engagement, our application's ability to attract and keep the users' interest, and effectiveness, the success rate with which it can in fact transmit the intended knowledge.

We will begin by taking a look at existing solutions in the same space. Since our project aims to teach programming fundamentals and thought processes, we looked to other efforts with the same goal. For many years now interactive software, known as Didactic Video-games, has existed with a very similar goal, to impart certain information or rational understanding in young children by keeping them engaged and motivated, using the same methods traditional video-games do (See Figure 1). Even when we look closely at the very specific knowledge area that we are interested we find that some of these game already exist that attempt to teach their young player base some of the basic underlying concepts of engineering.

When it comes to programming the same can also be said, Visual scripting languages like Scratch [6] and Block-based programming puzzle-games like Blockly [2] are designed to introduce complete beginners to the concept of coding, serving as the very first step in the complex and often times overwhelming world of computer programming. Additionally physical toys [3] with very similar architecture also exist to bring this learning experience to an even younger audience. It is not surprising that tools like this are becoming more and more relevant as this skill is also becoming one of the most sought after for a high percentage of college courses and job positions. It's only natural there is a great necessity to introduce programming in a fun and engaging way. The most significant missed opportunity with these applications is that with current day video-games becoming more and more visually impressive and attention demanding, simple static interfaces don't really have a great impact anymore. Physical applications are a good alternative because they leverage the physicality of the pieces to generate more interesting interactions but it becomes especially limiting precisely because it relies on a finite number of physical artifacts.

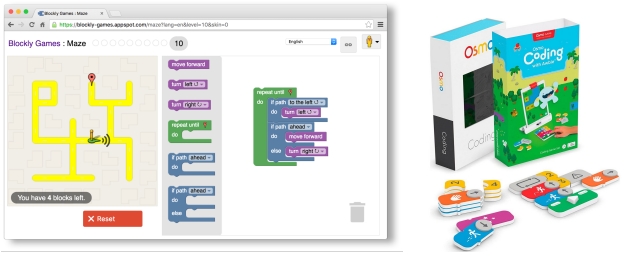

Fig. 1. Blockly [2] and Osmo [3], software and physical Didactic games.

Virtual Reality is notorious for it's ability to immerse users in a virtual world, removing outside distractions, providing a sense of presence, and allowing for the use of new and creative interaction techniques. These inherent qualities are what sets VR apart as a very promising platform for Didactic Gaming and Learning in general [1].

One such example of this is presented by Clare Wood [8]. In Civil Engineering University courses, students are encouraged to explore many spatial designs solutions for any given problem. Traditionally this is done with the help of regular 3D CAD software, however by utilizing Virtual Reality as a tool for to-scale visualization, studies found that users could identify structural problems much more easily and had generally a better and more productive design experience. Presence was found to be especially important as the study revealed that users that felt more present, also felt more engaged, created better solutions and had overall better performance.

Some papers have already attempted to leverage VR with the goal of creating an immersive and interesting experience for players to learn programming in a more effortless way. We will now analyze two such applications.

In a project called VR-OCKS [7], R.J. Segura et al. created a system inspired by visual languages such as Scratch or Kodu (See Figure 2) that works by proposing to the user the resolution of simple puzzles in a 3D environment. The main features include controlling a character path with code-blocks and different levels with linear difficulty progression. This project relied heavily on physical interactions such as moving blocks to code and has a target audience of children around the age of 12, due to the constraint of commercial HMDs having a limited interpupillary distance range.

The authors observed that the controls were easy to pick up, as such the game had a relatively quick on-boarding even for inexperienced users. However, tests also showed that users enjoyed coming up with a solution but got bored waiting to for the result to confirm if they were correct, this highlights the importance of building and maintaining a high level of engagement at all times.

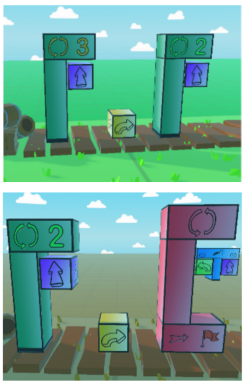

Fig. 2. VR-OCKS [7] programming, similar to block-based languages.

In the second paper, entitled VWorld, the authors designed a VR system with the goal to boost children's creativity and computational thinking skill. In this game, players could switch between two modes (See Figure 3). Creator Mode, where they had access to a Mini-map, and could Drag&Drop objects into the world, and Programmer Mode, where they could create linear instruction sequences to give said objects behavior (e.g: barking wolf, moving tree, etc).

The most relevant factors about this project are that it takes advantage of VR to drive engagement in a very intuitive and familiar way, similar to a children's playground, also it featured a customized non-linear learning experience, meaning that each player could approach the game in whatever way they deemed interesting with only very open-ended goals to drive their motivation. Finally it was aimed at young teens, (ages 13 to 15) for similar reasons as the previous paper.

Through testing, the researchers observed that users could complete the simple tasks given with relative ease, which provided a good learning environment. Users would play with the pieces even without being

given any goal, demonstrating their interest for playing with the system itself. The biggest drawback seemed to be that the simple programming system wasn't flexible enough to teach more than simple instruction sequencing.

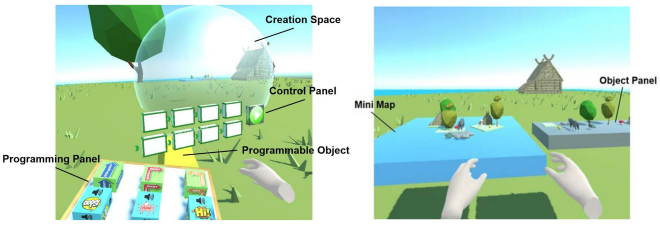

Fig. 3. VWorld [5], Programmer and Creator Mode.

From the previous two papers we can draw a few conclusions:

Virtual Reality Benefits

- 3D immersive environments are effective at keeping a user engaged and focused.
- Interaction/Object manipulation is easy to pick up since it feels natural to the user.
- Allows for the creation of experiences that wouldn't otherwise be possible.

Learning Progression

- Linear progression (ordered levels) are good to give users a goal which motivates learning and keeps them focused.
- Non-linear progression (sandbox) allows users to customize the experience to their own learning pace making it more accessible.

Target Users

- People above the age of 13 (recommended by HMD manufacturers due to IPD).
- Users with no previous knowledge of programming concepts.

Evaluation Strategy

- Questionnaires to gauge user reception and engagement.
  - Explore the ease of use of the interfaces and interaction techniques.
  - Determine difficulty progression curve adequacy.
  - Rate the level of enjoyment and sense of knowledge gained.
- Gather Performance Metrics: time to complete tasks, number of tries, number of registered inputs etc. To evaluate progress throughout the experience.

Finally, while searching for interaction techniques that would lend themselves to fun and engaging data/logic manipulation, an activity required for programming, we also analyzed the paper entitled ViBlock by Miteki Ishikawa [4], which propose a novel interface in VR that creates enjoyable experiences through natural and playful interactions with block-mapped content.

This framework can be used for many different applications but due to the context of our research topic we will be focusing particularly on one specific use case where the blocks are used to make a interactive children's game (See Figure 4).

In this example a block represents multiple type of data, a 3D model of an animal's habitat, an animated 3D model of the animal itself, a description of the animal and a representative sound. Then the user has access to a wall. This wall tiles blocks around the user, when users are looking at a block it becomes larger and dynamically rearranges the neighbor blocks, this way users can see all the blocks by looking

Fig. 4. ViBlock [4], Interactive Children's game application.

around with the HMD. Another important part of this framework are the connectors. Connectors manage the content of the block, with each region representing a different mode so the different type of data stored on the block can be displayed.

The main contribution of this paper is that this framework was shown to encourage users to interact with digital content in VR by allowing novel and playful content presentations and explorations beyond real-world experience. These interaction methods can be used for many other applications, and we will be drawing inspiration from them when designing our own metaphors and game mechanics.

3 System Design

Based on all the information gathered from the analyzed literature, we opted by creating a minimalist environment in which the user would be able to explore programming concepts through the virtual manipulation of coding blocks. Based on VR-OCKS [7] we decided to integrate a gaming component into our system, which consisted of several puzzles that the user must solve through the programming blocks, thus encouraging the user to explore these concepts in a more goal-oriented task.

The system is organized in levels, being that in each the user is faced with a puzzle that, once solved, will allow access to the next. The difficulty of the system accompanies this progression, given that the first puzzle can be solved through simple instructions while the last puzzle requires more complex instructions, e.g. loop or conditional instructions.

The puzzles consist of a chamber inside which there is a robotic character and diverse mechanics that must be activated in order to complete the puzzle. To do this, the user must build a program to control the robot in order to activate the diverse mechanics that can take multiple forms, in order to ultimately achieve the level's goal - opening the room's door so that the user can go to the next level.

In this section we will disclose our initial planning regarding the iterative development of the system followed by the actual iterative development of it, including the differences between the planned milestones and the actual developed work.

3.1 Milestone Planning

After deciding how we wanted to tackle the programming interface, we decided to structure the different milestones that would be developed throughout the semester. Those were:

- **Milestone 1:**
  - Implementation of preliminary environment and character;
  - Enabling basic VR interactions;
  - Implementation of simple blocks that control the character;
- **Milestone 2:**
  - Addition of new programming block(s);
  - Designing of puzzles;
  - Implementation of the design in the VR environment;

- Interaction between game entities;

- **Milestone 3:**
  - Conducting User Tests;
  - User testing followed by questionnaires with users to ascertain engagement, fun, ease-of-use and future interest in programming;

However, some of these goals were modified to meet the teacher's useful feedback or due to other external conditions.

3.2 Iterative Work

3.2.1 Milestone 1

Given that on this iteration phase we still did not have any feedback, the work done did not differed from what was planned. Therefore, on the two first weeks we focused on creating the basis for our system.

We started by focusing on setting a simulator through which we could test our system without using the Oculus HMD, and on adapting our Unity project in order to function as a virtual reality environment, e.g. making objects grababble. After this we created a minimalist environment where the user was given several grabbable coding blocks placed on a table. With these, he should create a coding program on the designated area which consisted on a dais connected to a computer that, when turned on, would enable a zero gravity field in which the blocks could be placed freely so that the user could organize them as he wished.

In terms of how we designed our "programming language", we opted by limiting the code through a START and END blocks, so that the user could more easily see the logical order of the program built. In order to ease the code "construction", as seen in Figure 5, we opted by enabling a physical linkage between the different blocks through a visible cable so that the user can easily see what blocks are connected.

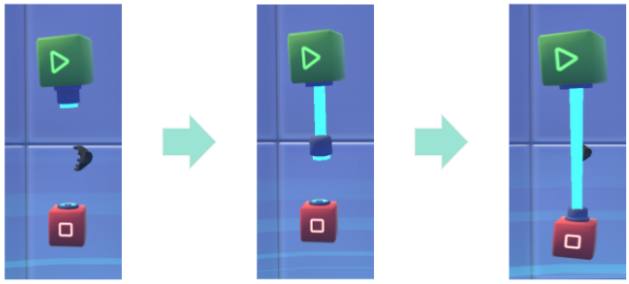

Fig. 5. Example of how to connect two blocks

Regarding the connection bewteen the code "constructed" and the robot which would exectue this, we decided to use a metaphor associated to a physical element, a floppy disk, that the user had to insert on the computer in order to save the built program by pressing the save button placed on the computer. After this, the user would have to remove the floppy disk from the computer and place it into the robot, so that this could receive the user's code, which would be executed when the user pressed the robot's "play button".

3.2.2 Milestone 2

Given the feedback received on the last milestone we opted by changing our priorities regarding the second milestone, and instead of designing multiple puzzles we opted by designing a simple puzzle and adding more mechanics that enabled run-time interaction between the user and the program the robot was executing.

Therefore, on this milestone we implemented the puzzle chamber inside which the robot would execute the program and implemented some mechanics that the user could activate by pressing some pressure tiles. Additionally we also added a specific area in which the user could summon the robot, so it could be easily accessible. Given that in this milestone the puzzle chamber was already developed, after inserting into the floppy disk into the robot, the user had to place it in a specific entry tile, through which it would enter the puzzle chamber.

In terms of coding blocks, in this milestone we added conditional instructions that enabled the user to evaluate the robot's different sensors. In order to keep a progression regarding the complexity of the code the user had to execute, we also designed two isolated puzzles for this milestone - one that could be solved without resorting to conditional statements and another in which, due to the limitations of the coding blocks provided, the user would have to use a conditional block to complete the puzzle. It is important to mention that, on this implementation phase, the developed puzzles were still isolated from each other, meaning that there was not a "in-game transition" between these.

3.2.3 Milestone 3

Given the current pandemic situation we decided to completely change our plans regarding this milestone. Therefore we believed it would be better to develop more programming concepts and consequently designing new puzzles instead of conducting user tests.

In order to integrate all the feedback we had receive throughout the previous milestones, we decided to change several aspects of our system. First, and in order to maximize the virtual reality component of our system, we changed the way the user can access the different available coding blocks. Thus, we replaced the table where these used to appear by a screen with which the user could interact through Ray-Cast technology. By selecting a block on this screen this would spawn in a designated area and the user could use it or erase it by throwing it in the bin. We opted by integrating this dynamic approach instead of the previous, to also help declutter the space given that the user could progressively spawn the needed available blocks instead of having all of them displayed in a surface. Besides, and to help the user understand what part of the program was being executed by the robot, we also added an highlighting system that highlighted the surface of the block being executed at the moment, as seen on Figure 6.

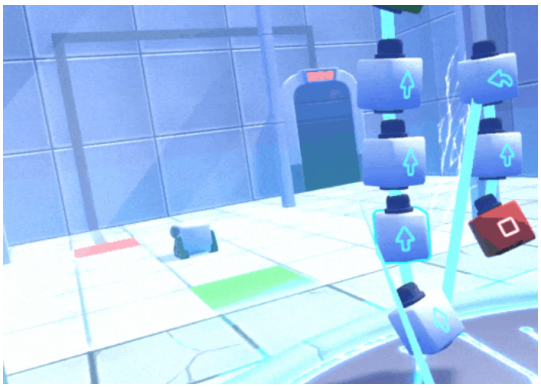

Fig. 6. Highlighting system being applyed

Following the same thought line and to ease even more the usability of the system by the user, we also decided to remove the floppy disk metaphor, given that this implied unnecessary movements. Therefore we decided to implement a more updated approach regarding how the code information is passed to the robot by limiting this to a unique computer's button that when pressed would "save" the code by automatically transmitting it into the robot which would automatically begin executing the code. It is also relevant to point out that another change we implemented was placing the robot always inside the puzzle chamber, contrary to the previous approach we had taken which consisted of having the user to insert the robot into the puzzle chamber.

On this milestone we also present a more complex programming concept - loops. With these programming blocks the user was enabled to constantly check the robot's sensors which enables the completion of some puzzles. In order to progress the complexity imposed on the user we decided to implement a level which the user could solve without

these type of blocks, given that he had enough blocks to do so, followed by a very similar level in which the user had few available blocks which would make him use the new programming block in order to complete the level.

Apart from this, we also implemented other mechanics that enabled run-time interaction from the user besides buttons. In this milestone we decided to take advantage of the Ray-casting technology used to interact with the screen to also use it to activate certain objects inside the puzzle chamber. The way these were design enabled the user to activate these from a distance an, in a way, forced this interaction to happen in the run-time, given that the mechanic activated would result in the disappearance of boxes that blocked the robot's trajectory which would reappear once the mechanics got deactivated, which disables the user from activating these in the begging of the level.

3.3 Polishing

The last iteration of our work consisted in polishing some aspects that we believed would improve the level of immersion offered by the system. Therefore, and once more following the useful feedback provided by the teachers, we decided to refine our system by adding background music alongside with audio feedback to all the mechanic interactions between the user and the system's components, e.g. when a button is pressed. Additionally, and in order to improve the game component of our system, we connected all the puzzles implemented throughout the semester in a progressive order, so that when the user solves a puzzle and consequently, the room's door open he can walk through this to move to the next level, having no alternative to progress in the game except by completing the puzzles proposed with the blocks available on each level, given that these differ from level to level.

4 System Implementation

Our project has a main system, responsible for the sudo-programming and robot behavior, and supporting systems, which facilitate the interactions with programming blocks and the creation of puzzles to challenge the user.

4.1 Block Programming

Making a sudo-programming system is not trivial as the goal is to simplify the relevant concepts to make learning easier for the user, but still accurately represent real programming to not invalidate that learning.

A core concern when implementing our system was the physicality of the interactions, as it was our hope that they would aid in building user understanding of programming. A program is a sequence of instructions and so, we needed to represent each segment of the sequence, the instructions, and how they related to each other, the sequence. For the instructions we chose blocks and, for their relationships, plugs and sockets.

Each programming block is an object composed of:

- An icon to identify it;
- One or more plugs to connect it to the following instructions;
- Zero to one sockets, for previous blocks to connect with it;
- A set of conditional sockets, for the conditional blocks (If and While);
- A Monobehaviour script to dictate its functioning.

Each of the block's scripts has a common ancestor, the Programming Block script which establishes the fundamental aspects of the blocks. From ascertaining which block comes next, utilizing plugs, to parsing it from its current form to a Program Node, to later be executed. This branch of the system's only responsibility is to maintain program structure.

In normal programming there is an inherent sequence to programs. By virtue of being expressed through language, a representation in 2-dimensional space, there is little ambiguity to what comes first. On the other hand, when in a 3D environment this becomes a problem and our solution was to explicitly connect instructions using plugs and sockets. These are relatively simple components with scripts that handle the connection that they established between their respective blocks, as well as how they interact with the player. Given the VR context of our project, these were a feature that benefited from work already done on grabbing objects.

Most of the player interaction with the virtual world is handled with the help of the XR Interaction Tool Kit. The most basic uses of the tools included within our project are controlling the virtual hands or grabbing objects. However, we went a step further and utilized for interactions between objects, like the aforementioned sockets and plugs, which needed to physically connect. To achieve all we required, the tools provided had to be expanded upon. We added functionality for keeping objects' relative position to the hand when grabbed, when they originally would snap to its center. Moreover, we needed to be able to disconnect plugs and sockets at will, to that end we implemented code to correctly separate interactors and interactables in the context of the plugin.

Following the example of many programming languages, we built our system such that the user created program, would be turned into executable code through a parser. Much like reserved words in normal programming, each of our blocks can be made executable by its respective Code Node. To reiterate, the Programming Blocks represented only the program structure, then the Code Nodes represent the functionality. When pressing the 'Execute Button' on the computer console in the room, the player triggers the parsing process, which starts with finding the 'Start Block'. From there, as the first block is merely an indicator, the parser moves on the convert each block to its node counter part, each with a reference to the next one. Using an If Block as an example, the parser first build its condition from the conditional blocks in the sockets and then moves on to the parsing of the instructions within the If's code block. Only after all this does the parser continue through the programming block sequence.

The program originated from the parsing process is a sequence of Code Nodes. Each of these Nodes has an Execute method, responsible for carrying out its functionality, be it querying the robot's sensors to evaluate the condition of an If or While or merely controlling the robot's actuators. They also have auxiliary functions, for example, to control the lighting up of the respective physical block to indicate to the user that it is being executed.

In turn, the robot receives the linked-list described and iterates through it by executing each Node. After the execution is complete, the robot asks the current Node what follows as it is the Node's responsibility to control the flow of the program. It is within the Code Node class that it is determined if after a While loop is completed, the robot is to return to the While's code block or continue through the program, for instance. The robot itself has two main subsystems besides the program execution, the sensors and the actuators. In the Code Nodes all robot specific implementation is abstracted and the robot has methods which receive instructions mainly in the form of Enum values to signal which sensor to access or action to enact.

4.2 Auxiliary Systems

Alongside the system described above, our project also required the implementation of auxiliary systems and individual components to achieve our goal.

With the intent of teaching basic programming concepts, the systems which allows for program construction is the main focus. However, in order to teach we also needed to present users with tasks so that in carrying them out, they would gain understanding. For that purpose, we created several elements for puzzle creation, namely: pressure plates (plates placed on the ground, which can trigger some other mechanism), laser receptors (objects which would trigger some other mechanism upon having a laser pointed at them) and others.

Beyond these singular puzzle elements, we implemented a system for the delivery of program blocks to the player. Part of our puzzles are the resources available to users, by limiting which programming blocks were available and their respective amounts, the same puzzle could

teach different lessons. With this in mind and also considering ease-of-use, we built a system which lets players select from a pre-determined inventory of blocks, what they require and spawns the requested block by the programming field. This system makes accessing the resources fast and avoids clutter, as blocks can be returned to the inventory when not needed anymore.

5 Discussion and Future Work

When compared to the existing literature, our system presents several benefits, mainly related to its interactivity and immersion. Besides enabling run-time interactions between the user and the system's components during the code execution by the robot, our system offers higher presence given that it takes better advantage of the three dimensions associated with virtual reality.

Additionally, the system presented in this report aims to be a powerful tool that can motivate people to learn how to code. Given the relative simplicity of the puzzles offered by our system, this mainly targets people with no previous background in programming. Therefore, its primary goal is to create a sense of curiosity in these people so that they can further investigate this skill on their own. By offering them an engaging and game-like system, we intend to increase their will to learn, thus making this system a boosting tool instead of a learning one. We believe that, by integrating a gaming component into our system this becomes more interesting to the user specially given that this component's benefits are augmented by the system's use of Virtual Reality which offers a compelling immersive environment ideal for the correct stimulation and engagement of the user.

Even though we are pleased with the resulting system, we believe that it should be evaluated by user tests, which was not possible to execute given the current pandemic situation we live in. Therefore our future work consists in conducting these user tests, where it would be proven if the user's will to learn how to code increased with the usage of our system. Additionally, our future work also includes expanding the programming concepts and puzzles offered by our system, which can be done easily by following the guide on the appendix.

6 Conclusion

After all the research reported, it is possible to conclude that, by using Virtual Reality, the players' senses are more stimulated, which produces a more real experience. However, nothing is more real than reality itself, but that can also become a constraint, as seen in Section 2. In the context of teaching or boosting the will to learn, physical media constitutes a limit because the exercises are limited to the pieces players possess. Games like the one presented, *Osmo*, suffer from this problem.

To evade this problem, Virtual Reality is one of the best solutions. However, nothing is perfect, and games, such as *VR-OCKS* and *VWorld*, also have their problems, which this turn, are associated to them being implemented in VR. As explained before, both of them try to teach the player basic concepts of programming, but they fail in keeping the players interested after creating their solutions.

Hello VWorld tries to succeed where the games used as inspiration failed by, as described in Section 3, incorporating mechanics where the players interact with the game's elements during their program's execution. The game can be considered successful since it achieves the pre-determined goals, such as being an immersive environment where players can acquire the taste of programming a robot. Another relevant aspect is that players feel a bigger sense of achievement by interacting directly with the programming blocks since they believe they control the robot's behavior with their own hands.

Finally, in terms of being a tool for boosting the players' will to learn how to program, Hello VWorld provides an excellent platform that can easily be expanded to new programming concepts by creating new puzzles and new mechanics.

References

[1] M. Bricken. Virtual reality learning environments: Potentials and challenges. *SIGGRAPH Comput. Graph.*, 25(3):178–184, July 1991. doi: 10.1145/126640.126657

[2] Google. Blockly games.

[3] T. P. Inc. Osmo – award-winning educational games system for ipad.

[4] M. Ishikawa, T. Hagiwara, K. Takashima, and Y. Kitamura. Viblock: block-shaped content manipulation in vr. pp. 1–2, 11 2016. doi: 10.1145/2996376.2996391

[5] Q. Jin, Y. Liu, Y. Yuan, L. Yarosh, and E. S. Rosenberg. Vworld: An immersive vr system for learning programming. In *Proceedings of the 2020 ACM Interaction Design and Children Conference: Extended Abstracts*, IDC '20, p. 235–240. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3397617.3397843

[6] M. M. L. Lifelong Kindergarten Group. Scratch - imagine, program, share.

[7] R. Segura, F. Pino, C. Ogáyar, and A. Rueda. Vr-ocks: A virtual reality game for learning the basic concepts of programming. *Computer Applications in Engineering Education*, 28:31–40, 01 2020. doi: 10.1002/cae.22172

[8] C. Wood. *Virtual Reality for enhanced teaching of conceptual design development*, pp. 43–47. IM Publications Open, 05 2019. doi: 10.1255/vrar2018.ch5

7 Appendix

The systems we have implemented show a lot of promise. However, given our time constraints, the full extent of their capabilities is not explored. Within this appendix, you can find instructions on how to both use the systems as they are to add new content, such as levels, and also expand them.

7.1 Adding a new level

7.1.1 Base Level

When opening our project within Unity 3D, you can find all project assets in the Assets folder, presented in the "Project" editor window. Within this folder, all assets are sub-divided into their respective categories. There are many ways to structure a game when developing in Unity, our approach was to create a 'Scene' for each.

To begin the creation of a new level, we prepared an asset to function as a base. Within the 'Scenes' folder you will find several sub-folders, the one we are looking for is the 'Puzzles' folder. The 'Puzzles' folder contains all scene assets corresponding to the levels we have developed and the 'LevelBase' scene asset. This is the aforementioned base from which additional levels can easily be created. It already has all systems and respective hierarchies set up and requires only that content be added. This scene should never be changed in order to always have it available for future development, instead duplicate it and give the copy the name you wish.[1]

7.1.2 Resources

With a puzzle design in mind, there are two main areas to implement. In this guide, we will first go over the resources made available to the player for programming. To define what resources are available in a level, look to the "Hierarchy" editor window and look at the "Room" object's children objects. Among these children there is a "Workspace" object, this is responsible for all programming related components. For resources we want to look at "Block Dispenser", which is a child of the "Computer Screen" object, which you should have seen when looking at the children of "Workspace".

Now, select the "Block Dispenser" object and look at the "Inspector" editor window, within it are displayed all of the object's components. The component we need to look at is the script named "Block Manager", all others can be collapsed for easier editing. In this script the two important attributes are "Available Programming Blocks" and "Available Condition Blocks", both of which are lists. First, when opening either, you will see a field with an integer which corresponds to the number of different block types you want to have available to the player (not individual blocks). In "Available Programming Blocks" you should see it is already set to 1 and there is an example entry. This entry is labeled "Element 0" and, if you expand it, you can see the attributes to define what block it is:

- "Prefab" in which you must select a prefab from those present in the "Programming Blocks" folder of the "Prefabs" folder;
- "Icon" should be the image corresponding to the block;
- "Available" is the number of blocks of this type to make available.

The next field is only relevant if, as it says in the "Inspector", the selected prefab is "InstructionBlock". This is used to choose which variant we want to make available from those listed in the relevant Enum field. The last field, "Stored", should be ignored.

"Available Conditional Blocks" follows the same structure as "Available Programming Blocks", with the only difference being it has three fields that need only have a correct value selected, if the selected prefab matches the one mentioned on the "Inspector"'s tooltip. Furthermore, all of the relevant prefabs for this list can be found in the "Conditional Blocks" folder within the "Prefabs" folder.

[1] Though by right-clicking, as you might instinctively do, there is no option to duplicate an asset, select the one you want and press Ctrl-D.

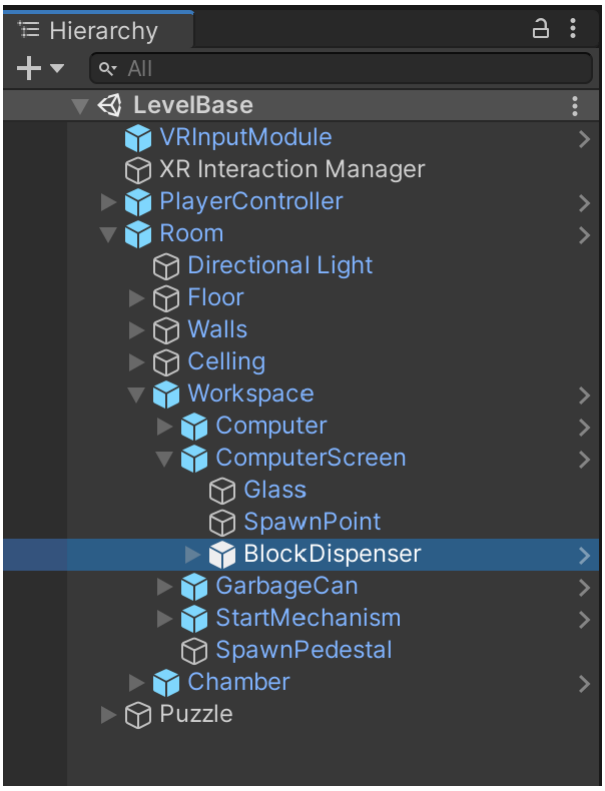

Fig. 7. Location of the Block Dispenser within the project's hierarchy.

7.1.3 Puzzle Chamber

As we have designed our teaching game, the objective of any level is ultimately to open the door of the chamber and continue on to the next. To have flexibility in designing our puzzles we created "Line Connectors", these function as cables and are meant to carry a signal for opening the door from the several possible sources. To set them up, you need only place one in the scene and, in the "Inspector", set another object as its 'input', the prospective source of the signal. These essentially work by creating a chain of objects with the component "Input Connector", culminating in an "Output Connector", in this case the door of the chamber. The most important thing is to make sure all objects in the chain, from its input to its output, are properly linked through their attributes in the inspector. Simple logic gates, such as "and"s or "or"s are also available.

Continuing, we have the objects we made to have as triggers to that signal for the door. These include the "Pressure Tile", which starts sending a signal whenever the robot is on top of it, or the "Laser Sensor" activated by shinning the laser the player has available on it. Examples of the versatility of these components can be found throughout the available levels.

Finally, we have the "Robot Boxes", these were created to be obstacles for the robot, something for the player to work around. They are very simple, but a good example of the uses for the "Detectable" script which is what enables the robot's sensors to pick up objects within the room. This script must be added as a component to anything the designer wishes the robot can detect to be in front of it.