

DESCRIPTION

INSTRUCTING COMPUTERS BY PHYSICALLY CONNECTED BUILDING BLOCKS / BRICKS FOR EDUCATION PURPOSES

BACKGROUND

In the contemporary digital landscape, educating children about programming is of paramount importance, fostering essential skills in problem-solving, critical thinking, and creativity. Introducing coding concepts early on not only equips children for the technology-driven future but also instills a proactive approach towards understanding and shaping the digital world. Motivating children to embark on this educational journey is key, and one effective solution lies in block programming.

By employing visual blocks or code blocks, block programming simplifies the learning process, making it intuitive, engaging, and enjoyable for young learners. This approach eliminates the need for complex syntax and embraces a user-friendly, drag-and-drop paradigm. These visual blocks represent programming constructs, such as conditions and loops, facilitating a deeper understanding of program flow without the complexities of traditional coding.

The simplicity of block programming, driven by a drag-and-drop interface, reduces initial barriers for learners. This design empowers users to focus on logic and problem-solving without being encumbered by the intricacies of typing out code. Immediate feedback within block programming environments enhances the learning experience, allowing users to see real-time results and reinforcing the connection between code creation and program behavior.

Block programming platforms are designed to be visually engaging, often incorporating elements of gamification. This not only demystifies programming but also instills a positive attitude towards this essential skill, particularly for younger audiences. The approach also includes error prevention, reducing the likelihood of syntactic errors and allowing learners to

concentrate on logical errors and problem-solving, gradually building their understanding of programming concepts.

Beyond programming skills, block programming cultivates computational thinking, developing skills like decomposition, pattern recognition, abstraction, and algorithmic thinking. This skill set provides valuable problem-solving abilities applicable across various disciplines. The versatility of block programming extends to interdisciplinary learning, enabling integration into diverse subjects, from science simulations to artistic animations and educational games in mathematics. In essence, block programming offers a visually intuitive and user-friendly path for beginners, laying a solid foundation for navigating more advanced programming challenges in the future.

DISCLOSURE OF THE INVENTION

PROBLEM STATEMENT

Previously only virtual block programming languages have been available, requiring individuals to use computers and engage in programming activities solely in a virtual environments. With this invention we provide a fundamentally new way of doing and learning programming by allowing one to build programs in the physical world leveraging the principles and benefits of block programming.

DEFINITION OF PHYSICAL BRICK PROGRAMMING

Physical Brick Programming refers to the technique where one can create programs in the physical world by connecting bricks capable of directing computers to perform specific tasks using block programming techniques.

DEFINITION OF PHYSICAL BRICK PROGRAMMING LANGUAGE (PBPL)

We introduce Physical Brick Programming Language (PBPL), which is a programming language that refers to the set of all the programs that can be created using Physical Brick Programming techniques. The physical blocks that we call bricks represent various instructions such as variables, conditions, loops and

functions. The connections between the bricks form a physical graph that we call a PBPL program.

ADVANTAGES OF PHYSICAL BRICK PROGRAMMING

Learning Physical Brick Programming offers several advantages, especially for beginners and children. It provides a visually intuitive and engaging way to grasp programming concepts without the complexities of syntax in a playful manner and without the need of a computer. Immediate feedback in real-time enhances the learning experience, and the attachment of bricks reduces entry barriers, fostering a focus on problem-solving and logic. Physical Brick Programming also serves as a stepping stone to traditional text-based languages, promoting a seamless transition to more advanced coding. Overall, it cultivates computational thinking, creativity, and a positive attitude toward programming.

DESCRIPTION OF THE DRAWINGS

FIG. 1: Architecture diagram of the Physical Brick Programming Environment

FIG. 2: A concrete example of a Physical Brick Programming Environment. The game PONG is being displayed on the screen that is controlled by a PBPL program. The PBPL program is defined by the connection of the bricks.

FIG. 3: Architecture diagram of the bricks that represent the programming elements of the PBPL language.

FIG. 4: Unassembled 401 and assembled (connected) 402 example assignment of the physical brick programming expression "X:=Y+3". The program flow goes from bottom to top.

FIG. 5: Unassembled 501 and assembled 502 example physical brick programming condition "if X < 0 then halt, else inc X". The program flow goes from bottom to top.

FIG. 6: An example for a PBPL loop "while X < 0 then sleep; print X" that demonstrates waiting until the variable brick X gets a non-negative value that gets printed once the execution flow gets out from the loop. The loop starts at the bottom brick 604. The execution goes into the right side of the conditional brick 602 if the value of variable brick X is negative and continues through the wire 603 to the beginning of the loop 604,

otherwise the execution flow exits the loop 601 and prints out the value of the variable brick X.

FIG. 7: An example PBPL program as a solution for playing the game PONG. The PBPL program controls the right paddle. In case of 701 the ball is below the center of the right paddle so the PBPL instructs the paddle to go down. In case of 702 the ball is above the center of the right panel so the PBPL instructs the paddle to go up. 704 summarizes the PBPL program on a flowchart visible on 703.

MODE FOR CARRYING OUT THE INVENTION

DESIGN OF THE MAIN BOARD, PHYSICAL BRICK PROGRAMMING ENVIRONMENT

First we present the design and architecture of the Physical Brick Programming Environment (PBP Environment). It is important to recognize that the scope of rights encompasses all modifications, equivalents, and alternatives related to the design and architecture of the PBP environment.

The PBP environment consists of a main device that reads and executes the PBPL program formed by the connection graph of the bricks. FIG. 1 shows the architecture of such an environment. The main device 101 contains 3 units: the main CPU / microcontroller 102, a memory (RAM) unit 104, a display screen 105 and an extra slot where one can connect bricks 103. The connections between the bricks then form the actual PBPL program that is executed by the main microcontroller 102. The bricks are communicating via electrical signals with a low voltage distributed to all of them originating from the main device board as together they form an electrical circuit.

An example of the environment is shown in FIG. 2. Here we captured the state of a running game (PONG). The main device 202 is a Printed Circuit Board (PCB) that has a microcontroller (CPU), memory and screen 203 integrated on the PCB. The bricks 201 are attached on the upper part of the PCB and together they form a set of instructions for interacting with the game.

The main microcontroller runs a software that controls each unit: memory, screen and bricks. It also executes the PBPL program that is formed by the connections of the bricks. The

PBPL program does not necessarily need to be stored in the memory, it can be interpreted brick by brick because it exists in the physical world.

DESIGN OF THE BRICKS

Here we present the architecture of the bricks. It is important to recognize that the scope of rights encompasses all modifications, equivalents, and alternatives related to the design and architecture of the bricks.

FIG. 3 shows an architecture diagram of a common brick. Each brick has a physical shell 304 that contains a PCB that has a microcontroller 305, one or more LEDs 306, one set of input female pins 307 or anti-studs and some sets of output pins 301, 302, 303 or studs. A brick could have zero, one or more output pins or studs. The communication and power supply occur via these input and output pins or studs. FIG. 4 and FIG. 5 show 2 examples on how the various types of bricks can be connected by their pins / studs.

The microcontroller of each brick contains the programming of the brick in its ROM. It stores the instruction type of the brick (e.g. variable X, conditional, jump etc.) as well as a program that handles the communication between its set of input and output pins / studs. The state of each brick changes dynamically depending on what instruction it gets from the main CPU unit 101. The main instructions are: "return brick type", "receive id", "send id", "next", "activate by id", "deactivate id". Instruction "return brick type" returns the programming element type of the brick (e.g. variable X, conditional, jump etc.). Instruction "receive id" receives an unique identifier number from the main CPU unit 101 and stores this number in the brick's memory. It also sets its state into active. When a brick is active, its LED 306 is turned ON indicating its active state and when it is deactivated its LED 306 is turned OFF. Instruction "send id" sends the identifier number stored in the brick's memory to the main CPU unit 101. Instruction "next" sends an activation signal for the brick's child brick and deactivates itself (in case of conditional brick, it sends to its children from left to right). The child of some brick A is

brick B where we connect the set of output pins / studs of brick A with the set of input female pins / anti-studs of brick B. In this context, we refer to brick A as the parent of brick B. Instruction "activate by id" activates the brick with the given identifier number only. Instruction "deactivate id" deactivates the brick with the given id.

Instruction types of the bricks are coming from the set of programming elements such as variables, operators, operations, numbers, functions, conditional, jump. The most basic bricks have only one set of input female pins / anti-studs and one set of output pins / studs. Bricks of this nature include variables, numbers, unary operators, binary operators, unary functions, assignment operators. FIG. 4 shows such bricks.

Conditional brick is a long brick that has one set of input female pins / anti-studs and 3 sets of output pins / studs so it can have up to 3 children bricks. Typically a binary operator can be connected to the middle set of output studs and the sides can have variables, numbers, or a set of expressions attached. The conditional expression is read and undergoes evaluation from left to right. FIG. 5 shows an example for a conditional brick denoted by "IF" on 502. Its children from left to right are variable brick "X", binary operator brick "<" and number brick "0". Its evaluation depends on the actual expression formed by its children read from left to right. If the evaluation results in the expression being FALSE, then the PBPL execution flow continues above the child brick of the condition's left child brick. If the evaluation results in the expression being TRUE, then the PBPL execution flow continues above the child brick of the condition's right child. In case of 502 if the expression "X < 0" is FALSE then the PBPL execution flow would continue above the variable brick "X", which is the brick "INC". Otherwise if the expression "X < 0" is TRUE, the PBPL execution flow continues above the number brick "0", but that has no children, so the PBPL execution flow stops.

Loops can be created using a conditional brick and a special "jump" brick that is the parent brick of the conditional brick and that is also connected with some of the children bricks or descendant bricks of the conditional brick. The long distance

connection can be achieved by connecting the set of output pins / studs of a brick with one end of a wire and the other end to a special "jump" brick that has two sets of input female pins / anti-studs designed for this purpose. See FIG. 6 for an example where if the evaluation results in the expression being TRUE, the flow of execution of the PBPL program continues towards 602 and jumps through the wire 603 to the parent of the conditional brick 604. If the evaluation results in the expression being FALSE, then, then the execution flow continues on the left side of the conditional brick and ends in 601.

For communicating between the bricks we use electrical signals of low voltage, radio or other electromagnetic signals. The communication and power supply happens between the set of input and output pins or studs of the bricks.

READING AND EXECUTING THE PBPL PROGRAM

The main microcontroller 102 on FIG. 1 communicates with the bricks 103 through electrical signals. The approach that the invention uses for getting the next brick in the PBPL execution flow works the following way. Each brick has at least one parent: another brick or the main device and can have zero, one or multiple children bricks. If one brick has zero children bricks, it means that there are no instructions coming after that brick. If a brick has one child brick then the next brick in the PBPL execution flow is going to be its child brick. If a brick has multiple children bricks e.g. a conditional brick that has 3 children bricks, left, middle and right bricks, the next brick in the PBPL execution flow depends on the evaluation result of the expression of the condition as we defined in the previous chapter. Getting the next brick in the PBPL execution flow can be achieved by sending the command "next" from the main microcontroller 102 to all the bricks via an open channel as only the active brick is going to execute the command "next".

The connection (physical) graph that forms the PBPL program is defined from from the ground upwards. This graph is discovered by a graph traversal algorithm. While discovering the graph the PBPL program gets executed brick by brick. The first brick that is physically connected to the PCB 101 is the first instruction

of the PBPL program. First it gets activated by the main microcontroller 102. Then the main microcontroller 102 sends the command "return brick type" that returns the instruction type of the first brick. The main microcontroller 102 generates an identifier number and sends it to the brick that receives it by its "receive id" instruction. The main microcontroller 102 also executes the instruction that was returned from the brick as the first step of the PBPL program. Then the traversal continues recursively. Depending on the type of the brick, it can have zero one or multiple children. If at any point of the traversal a brick has zero children, the traversal ends there. Let's assume that some active brick has one child. In that case the "next" command activates its child brick and the traversal continues there. If the traversal hits a conditional brick then it first evaluates its children as the expression they form and based on the evaluation results it continues either on the child brick of the left child brick or the child brick of the right child brick. If the traversal hits a brick by jumping or looping back that was already discovered then there is no new identifier number generated and no new brick is added to the graph. But the PBPL execution flow continues from that point until it reaches a termination state or a brick that has no children.

FIG. 7 shows a concrete example of a PBP environment and PBPL program. The program is essentially an infinite loop that tries to catch a ball with a paddle visible on 701 and 702. 703 is the PBPL program controlling the paddle either upwards or downwards based on the position of the paddle compared to the position of the ball. The brick jmp jumps back to the beginning of the PBPL program, to the conditional block. The corresponding flowchart is visible on 704.

CLAIMS

1. Physical Brick Programming facilitates a gamified programming environment by incorporating block programming concepts in the physical world. The invention utilizes innovative physical building blocks / bricks to simplify programming tasks, providing immediate feedback and fostering a positive attitude towards computational thinking, particularly targeted for use in educational software and interactive learning platforms.
2. Bricks mentioned in Claim 1 represent instructions for a computer and can be connected together physically. These sequences of connections define computer programs that are capable of doing any tasks. The transmission between the blocks occurs through electrical signals of low voltage, radio frequencies, or other forms of electromagnetic waves.