Exploring Minecraft as a Conduit for Increasing Interest in Programming

Christopher Zorn
University of Central Florida
4000 Central Florida Blvd
Orlando, FL 32816, USA
czorn@knights.ucf.edu

Chadwick Wingrave
University of Central Florida
4000 Central Florida Blvd
Orlando, FL 32816, USA
cwingrav@eecs.ucf.edu

Emiko Charbonneau
University of Central Florida
4000 Central Florida Blvd
Orlando, FL 32816, USA
miko@cs.ucf.edu

Joseph J. LaViola Jr.
University of Central Florida
4000 Central Florida Blvd
Orlando, FL 32816, USA
jjl@eecs.ucf.edu

ABSTRACT
We present an investigation of how Minecraft can be used to promote interest in computer programming. To facilitate this exploration, we developed CodeBlocks, a block-based programming language used to control a virtual robot that navigates, senses, and interacts within the game. We modeled it after several successful graphical languages for programming education and performed a study with non-programmers to evaluate its ability to improve perceptions of programming and teach non-programmers to program. A survey of current Minecraft players was conducted to identify interest in the plugin. We found support for our main hypothesis that the programming interest of non-programmers improved as a result of using CodeBlocks. The plugin has been publicly released to the Minecraft modding community and is available to play on our public server.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**] Computer and Information Science Education – Computer science education; K.8.0 [**Personal Computing**]: General - Games;

General Terms
Design, Human Factors, Languages

Keywords
Game-based learning; Minecraft; programming education

1. INTRODUCTION
In the past decade, the number of students interested in computer science has been decreasing [6]. Students are rarely exposed to programming in elementary and middle school and often lack engaging learning experiences. The potential for collaborative constructivist learning, using games in an informal learning approach to address learner involvement, is promising, but often elusive. Learners quickly recognize learning games for what they typically are: shallow, with poor stories, bad gameplay, and artistically lacking scenery.

Figure 1. We extended the commercial video game Minecraft with our robot programming language, CodeBlocks. It was used to teach programming concepts and generate interest in programming.

To address this problem, we explored how Minecraft can be used as a means to improve interest in computer programming. Minecraft [18], an open world style game and VGA 2011 independent game of the year awardee, is a gaming phenomenon which paradoxically attracts the attention of mainstream gamers with: no plot, no story, no goal, simplistic combat and pixelated graphics. Yet, it sold more than 3.5 million copies before it was even released. The game's main mechanic is creative play, but also incorporates collaboration, exploration, and adventure. In its simplest form, it consists of pixelated 1-meter cubes (see Figure 1), but also includes tool crafting, resource gathering, survival mode, multiplayer servers, farming, livestock, and even programmable Boolean logic and mechanical motion. The emphasis on personal progression and the presence of supernormal stimuli help make Minecraft captivating [2]. It also has an active modding community, exchanging plugins and extending the game in new creative ways.

To improve interest and educate non-programmers, we looked to existing successful programming education tools for inspiration. We found several aspects that were common among the tools: simplified language syntax, a graphical interface, and a focus on controlling the behavior of in-game entities. Most notably, our plugin, CodeBlocks, is modeled after Scratch [17] and StarLogoBlocks [3].

CodeBlocks is a plugin for Minecraft that creates new ways for players to enjoy the game; either through the experience of programming a robot, the ability to automate in-game tasks or by solving challenging puzzles. With CodeBlocks, we can capitalize on the amount of time players spend in Minecraft in order to expose a larger group of players to programming concepts and stimulate interest in computer programming. It has a small but functional instruction set and a structural syntax that mimics Minecraft’s block-based gameplay. Players can quickly create powerful programs to automate common tasks, while indirectly learning the basics of programming.

2. RELATED WORK

2.1 Computer Programming Education

There have been several approaches through the years to improve the learning of computer programming with games. The idea being that the excitement and “play” involved in the game keeps the learner in the educational process. Many programming education games use languages with simplified syntax or visual representations to reduce the learning curve. Additionally, they often emphasize the use of manipulating in-game entities to achieve specific goals. Although there has been a lot of work in programming education, to the best of our knowledge, this is the first time Minecraft has been used to do so.

To combat user frustration of syntax errors, particularly with novice programmers, most educational programming languages are visual and allow users to drag-and-drop actions and constructs to create programs [1, 3, 5, 15, 17, 24, 27]. This reinforces program structure and keeps the program in a runnable state, allowing learners to test and debug all of the changes they make with immediate feedback [22, 23].

Most educational programming systems and games focus on defining the behavior of in-game objects, such as people or animals; the use of animated, relatable entities in scenarios with meaningful goals can improve learner engagement [14]. Some systems allow learners to create sequences of actions for a robot to perform in both creative contexts [8, 13] and puzzle contexts [1, 30] where the robot interacts and senses its environment. Other systems allow the manipulation of many and diverse entities within a scene. Users can create stories [5, 22], games [16, 29], or both [17] by defining rules for entity interaction.

CodeBlocks aims to incorporate the best aspects of these systems. Its block-based language is based upon the visual style and program control flow of [3, 17], but represented as 3D blocks to coincide with Minecraft’s play style. CodeBlocks can be used for creative purposes like [5, 13] by encouraging learners to programmatically design elaborate structures. Alternatively, learners can use CodeBlocks to overcome challenges they face, as in [1, 16], such as mining resources within Minecraft. While CodeBlocks pulls from these different approaches, two major features of CodeBlocks are: 1) *that it supplements and is motivated by current gameplay;* 2) *and that millions of users already play Minecraft, greatly reducing its barriers to entry*.

2.2 Game-based Learning

Games for learning are not new and have been used as a medium for teaching many topics, including programming concepts, in a way that engages learners [21, 26]. Today, students are proficient consumers of visual and digital content. They prefer to learn with inductive reasoning, and learn best with smaller, more frequent exposure to educational content [10]. Digital games can complement these dispositions when they are properly constructed. Digital games should be motivating and emphasize constructivist learning environments, where gameplay is “experiential, active, problem-based, and collaborative” [31]. Certain components in the game are necessary for inspiring motivation: [20]

1. the learning context should encourage curiosity;
2. learning goals should align with player interests;
3. learners should be rewarded with correct effort.

Additionally, learning environments should provide players with balanced challenges that keep players engaged while they work on tasks. Immersive games improve flow and lead to natural stealth learning, which aids in knowledge transfer and active learning [20].

CodeBlocks and Minecraft embody many of the desired characteristics of game-based learning environments. Its gameplay is immersive and generates a sense of ownership; players approach the game more as a tool for creative expression, and less as a traditional game [9]. CodeBlocks extends these characteristics, providing players with an educational tool that aids in game progression and rewards players for using it. Additionally, most existing education games are developed by academics, resulting in effective learning tools, but poorly engaging games [10]. By modifying a popular commercial game, CodeBlocks avoids this problem.

In addition, Minecraft is an open-ended game where players are free to express their creativity. Placing constructivist learning into the game immediately allows them freedom to explore ideas, solve them, and learn from the process. For this reason, Minecraft has been used to inspire players to be creative [32], defining inspiration as three (motivation, knowledge, environment) of the six (plus intelligence, personality, thinking styles) creativity resources of the Investment Theory of Creativity [28].

3. CODEBLOCKS

CodeBlocks is a robot control programming language in Minecraft implemented as a freely available plugin for Minecraft servers (Available from [33]). To fit with Minecraft's play style, programs are written by creating sequences of blocks, with each block representing a robot instruction. Players can place and destroy instruction blocks much like they would when they create structures within the game. This differentiates it from another, simultaneously developed Minecraft plugin ComputerCraft [4], which embeds a lua-based programming language in-game, to control a robot. This laudable plugin requires more programming expertise to use, though demonstrates the desired power that creative players wish to wield.

In CodeBlocks, where typical programming games have separate programming modes, such as an IDE or text editor, CodeBlocks players never leave the game setting, remaining immersed. A program’s entry point is specified by a sign and it names the program or function (see Figure 2). Blocks are placed in a line and provide different instructions to the robot, for example to move forward, turn right, or sense the block in front. For clarity, a custom texture pack is used to make blocks appear with identifiable markings or text that describes the block's purpose. Functions are defined adjacent to the program definition and are also named using a labeled sign. Then, to call a function, the function call block is placed in the sequence of the program with a sign indicating the called function. Branching is achieved with a sensing block and a sign specifying the block type for which the robot is checking. In this way, we have created robots that dig for minerals, traverse mazes and solve puzzle challenges.

Figure 2
robot to move forward, turn right and pick up a block. The block-

3.1 Piloting and Design Decisions

We piloted the resulting system on users from our research team and our Minecraft server's play-testing group. They were asked to build simple programs while we watched and were told to speak aloud their ideas. Overall, they liked playing with the blocks, which was encouraging. They identified a few remaining difficulties that were resolved in iterative changes. We created a custom texture pack that made each block recognizable as to how it functioned (see Figure 3). Remembering the branching direction was problematic too, so when a sensing block was placed, we added a green block on the ground to show the positive branch direction and a red block to show the negative branch direction.

A few other design considerations were made. We were concerned with the ability for players to trace program execution. We tried different approaches and settled on stacking a block on top of each instruction when it was being executed. As the execution traversed the main program and recursed down functions, players could watch each step by following the moving, extra block. To slow down or speed up execution for debugging or other purposes, a player could place a sign on top of the start block to set the speed of execution. It defaulted to one block per second.

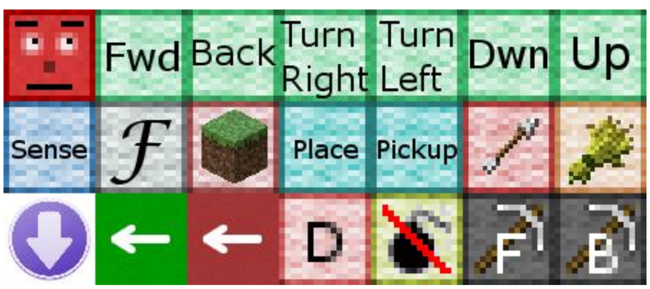

Figure 3. The texture pack demonstrates the programming statement of each block. Row 1: Robot, Move Forward, Move Backward, Rotate 90° Left, Rotate 90° Right, Move Vertically Down, Move Vertically Up; Row 2: Sense, Function Call, Build, Place Block, Pickup Block, Shoot Arrow, Harvest; Row 3: Current Action Indicator, True Branch Marker, False Branch Marker, Destroy, Defuse, Mine in Front, Mine Below

Although this design worked well in testing, we needed to allow CodeBlocks to be used by any Minecraft player on any server. We made programs be defined by a name on a sign, and allowed players to spawn a robot remotely, not just next to the program blocks. So, users placed a robot at any desired location and issued a command to start execution of a program. Second, we allowed users to name functions with a sign, so programs can call functions belonging to other programs, allowing for function reuse and collaboration. The defining name for each function was split into two parts: the program name it belonged to and the name of the function. For example, two functions of a program named Tower were Tower.createWall and Tower.createStaircase.

4. EXPERIMENT

We designed an experiment to explore three things: could non-programmers use CodeBlocks to create programs to solve puzzles, do interactions with CodeBlocks improve perception of programming, and does the method of program creation affect their interest.

To determine the effectiveness of the block approach to programming, as well as the sufficiency of the representation, we developed a second interface to CodeBlocks, more familiar to programmers. This text interface allowed users to load and save programs written in a text version of the CodeBlocks language (see Figure 4). This was done via a webpage that wrote a file the plugin would load when a player ran their program. The text representation was a one to one relationship of block to function.

We were concerned with the following participant perceptions and their change in the course of this intervention:

- overall programming interest,
- perceived programming difficulty,
- perceived programming usefulness, and
- programming enjoyment,

and explored the following interface conditions:

- *Block* – participants placed blocks in Minecraft to program the robot,
- *Text* – participants typed text into a web page to program the robot.

Our three hypotheses are listed in Table 1.

Table 1. We have three experiment hypotheses, centered on Co - to computer and robot programming.

H1	Computer and robot programming appreciation will increase with use of CodeBlocks.
H2	The Block group will have more appreciation than the Text group.
H3	The participant's learning style will affect the change in appreciation's magnitude.

Because CodeBlocks is based on successful educational programming languages and Minecraft is immersive and engaging [9], we expected participant perceptions to improve regardless of the program creation method (H1). Between the two groups, we expected Block participants to have a larger improvement because they don't leave the game environment and building structures with blocks is more exciting than entering text into a webpage (H2). Additionally, we expected that the learning style of the participants would impact their feelings towards programming, with active and visual learners changing the most (H3). We used 7-point Likert surveys to measure the feelings of the participants before and after the intervention (see Table 3). During the experiment, participants completed the Index of Learning Styles [11, 12], which we used to group like learners in four categories (see Table 2) during analysis.

Table 2.
looked for differences in appreciation between participants based on their learning style.

Category 1	Active: 11, Reflective 19
Category 2	Sensing: 21, Intuitive: 9
Category 3	Visual: 24, Verbal: 6
Category 4	Sequential: 20, Global: 10

4.1 Participants and Apparatus

Thirty participants (15 male and 15 female, ages 18-51) from the University of Central Florida were recruited. All participants were psychology students who were required to participate in research studies for course credit. The study was advertised on an internal system, which was used to organize the experiment sessions. Only one of the participants had previous knowledge of Minecraft. None of the participants had any prior experience with programming. We were able to balance the two condition groups with 15 in each; however, because each group had an odd number of participants we were unable to balance the genders within the groups. The Block group had 8 females and 7 males, and the Text group had 7 females and 8 males. Participants used a dual-core desktop PC with an NVIDIA GeForce GTX 470 graphics card and 50 inch Samsung DLP 3D HDTV. They were seated approximately three feet from the display with Minecraft playing in full-screen mode.

4.2 Experimental Procedure and Design

We conducted a single session, between-subject intervention where participants were individually taught how to use the system and then solved puzzles by creating programs. During the intervention, we introduced participants to the system, taught them how to create programs, let them solve puzzles independently, and challenged them to create a program related to bubble sort, a common algorithm taught to new programmers. At various points in the intervention, we measured their perceptions using questionnaires.

Table 3. Participants indicated their prior interest in programming using a 7-point scale. This was used to test all hypotheses.

	Pre-Test Assessment Questions
Q1	I am interested in computer programming
Q2	I think computer programming is too difficult for me to learn
Q3	It is useful to know how to program computers
Q4	Computer programming sounds fun
Q5	I am interested in robot programming
Q6	I think robot programming is too difficult for me to learn
Q7	It is useful to know how to program robots
Q8	Robot programming sounds fun

During piloting, we found that non-programmers were lacking sufficient knowledge of computer and robot programming to accurately indicate their perceptions. So, at the study start, the moderator briefly talked with the participants about computer and robot programming so they could better respond to a pre-test questionnaire. The moderator explained what a program is and why they are useful. The moderator then discussed robot programming with the participant and how it differed from computer programming, specifically how robot programming is often tailored for interaction between a robot and its environment. The participants then took a pre-test containing 7-point Likert scale statements (see Table 3) to gauge their prior interest in programming. Then, participants completed the 40-question Index of Learning Styles [11, 12], which we later used to determine whether the learning style of the participant affected the outcome of the intervention.

Table 4. Participants were given a tutorial on the basic syntax of CodeBlocks. This enabled them to use CodeBlocks to solve puzzle challenges.

	Tutorial Section
1	Demonstrate how to define a program
2	Demonstrate how to create a simple 3-instruction program
3	Describe branching and how it is used
4	Complete a partial program with a branch statement
5	Demonstrate how to define functions and when to use them; participant completes a partial program with a function.
6	Demonstrate how to create a recursive function and when to use it

Next, participants were guided through a 6-part tutorial (see Table 4) that explained how to create programs with the system. They learned about the functionality of each of the blocks and the different ways the robot could interact with Minecraft's environment. They were briefly taught the programming concepts of functions, branching and recursion. The tutorial took approximately 20 minutes.

Table 5. To become more familiar with creating programs in CodeBlocks, participants completed 4 learning puzzles. The solutions to the puzzles required participants to use all of the program constructs available.

	Learning Puzzles	Solution Descriptions
1	Move the robot to a specified location	Requires a sequence of five simple actions
2	Rotate the robot and destroy the specified blocks	Longer sequence of instructions with varied actions
3	Of the blocks surrounding the robot, destroy the Dirt blocks and defuse the TNT blocks. The target blocks are randomly placed.	Define a function containing one branch statement and calling the function four times
4	Move the robot to a specified location while defusing TNT obstacles in its path. The destination and the location of the obstacles are randomized.	Requires a recursive function with a branch statement

Next, participants completed four learning puzzles. These puzzles increased in difficulty to challenge their understanding of programming (see Table 5). They worked independently on the solutions; however, the moderator answered questions as needed so that the participants were ultimately successful. A solution for learning puzzle 3 is show in Figure 4.

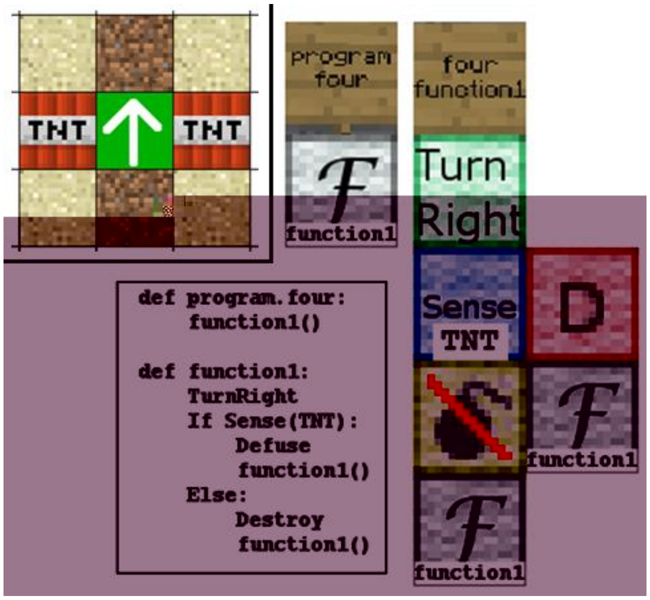

Figure 4. A diagram representation of learning puzzle 3 (see Table 5) is shown (top left). Participants were required to create a program that destroys Dirt blocks (brown) and defused TNT blocks. (Right) Solutions for the same puzzle are shown with the block version on the bottom right, and the text version on the bottom left.

After the four learning puzzles, participants selected one of four larger challenge puzzles and attempted a solution. This was done to challenge participants and to determine if they were capable of completing a difficult puzzle.

- Destroy specific blocks in a pattern
- Destroy all blocks in a given area
- Maze traversal
- Collect specific blocks along a path

In the last puzzle, the grand challenge puzzle, participants were pushed to their limits so we could observe how they would respond to a real computer programming problem. The grand challenge puzzle was to implement a modified bubble sort algorithm, which we broke into three parts for them to solve as individual functions. Given an array of blocks consisting of two colors (blue and yellow), participants were required to sort the blue blocks to the right and yellow to the left. The first part had participants sort a pair of adjacent blocks, swapping their position if they were out of order. By moving the robot right and recursively calling this function, one block could be pushed to the end. The second part moved the robot along the array until it found a blue block. Combining the two functions allowed the robot to move a blue block from the middle of the array to the end. Lastly, participants defined a third part that moved the robot from the end of the array to the beginning. By repeating the process of finding the leftmost blue block, pushing it to the end and returning to the beginning, participants were able to group all of the blue blocks at the end of the array.

Once the participants had successfully created a program that sorted the blocks, it was explained to them that the algorithm they had described in Minecraft was similar to an algorithm that would have been explored in a traditional programming environment, specifically bubble sort. They were then shown bubble sort, and it was explained to them how the two algorithms were similar and what would be needed to transform their CodeBlocks algorithm to the Bubble Sort algorithm (i.e. a change in the comparison function). Participants then completed a post-test questionnaire, identical to the pre-test questions. They also completed a subsection of the Intrinsic Motivation Inventory (IMI) [7] which we used to measure motivation and interest. The IMI contained sections for measuring interest, perceived competence and effort.

5. RESULTS

To test H1, we performed a Wilcoxon signed ranked test to determine whether the perceptions of participants changed as a result of their interaction with the system (see Table 6). As can be seen, most perceptions did positively change as a result of the CodeBlocks intervention, especially for computer programming.

Table 6. After using CodeBlocks, participants had changes in their perceptions of computer programming. We see that CodeBlocks is quite successful in changing perceptions, especially in Computer programming. (For *Condition Type* below, C=Computer Programming, R=Robot Programming, B=Block Group and T=Text Group) (* indicates $p < 0.05$; ** indicates $p < 0.01$)

Condition Type	Measure	Mean Before	Mean After	Z - Score
CB	Interest	4.20	5.33*	-2.859
CB	Difficulty	4.27	2.87*	-3.086
CB	Enjoyment	4.60	5.33**	-2.299
CB	Usefulness	5.33	6.07**	-2.006
CT	Interest	3.93	5.27*	-2.829
CT	Difficulty	4.53	3.07**	-2.100
CT	Enjoyment	4.47	5.40**	-2.360
CT	Usefulness	6.27	6.00**	-2.000
RB	Interest	4.67	5.13	-1.469
RB	Difficulty	4.73	2.87*	-2.748
RB	Enjoyment	5.07	5.47	-0.997
RB	Usefulness	4.93	5.67**	-2.050
RT	Interest	4.47	5.07**	-2.309
RT	Difficulty	4.87	3.07**	-3.090
RT	Enjoyment	5.27	5.67	-1.387
RT	Usefulness	5.67	6.00	-1.633

For H2, although many of the changes in perception were individually significant for a participant, we did not find any significance between participants of the two interface groups (Text and Block). To determine whether there were any significant differences in prior perceptions of programming between the two groups, we performed a Mann-Whitney test. We found that the Text group believed prior to the intervention that robot programming was more useful than the Block group ($Z = -2.157$, $p < 0.05$).

For H3, we used a Mann-Whitney test to determine whether learning styles affected programming appreciation, but found no significance between styles. Additionally, there was no significance in the IMI results between different learning styles or between different interface groups.

Our results indicate that hypothesis H1 held, while H2 and H3 did not. We observed a significant improvement in perceptions towards both computer and robot programming in most of the categories; however, neither the program creation method nor the learning styles of the participants had a significant influence on the magnitude of the improvements.

6. MINECRAFT COMMUNITY SURVEY

To identify interest in CodeBlocks among typical Minecraft players, we created a survey based upon our study questionnaires and advertised it online in several forums and listservs dedicated to Minecraft. We received 43 responses in just over a week's time. The survey had four sections: demographics, computer and programming familiarity, video game interest and aspects of Minecraft they found interesting. For most of the survey, a 7-point Likert scale was used to assess agreement with a statement, with 1 meaning 'not very' and 7 meaning 'very'.

The results indicate that the Minecraft players who are active in forums and listservs were active video game players with an interest in programming. Most respondents were between 18-20 years old, and all but one was male. Their educational levels varied, most likely because of the age groups. Respondents were very comfortable using a computer, with 39 participants answering 7 and 3 answering 6. There was also a high level of interest in programming (mean 5.95) and enjoyment of programming (mean 5.87) however the number of people currently programming in their jobs was lower (mean 4.39), probably because many respondents are still in school. The number having previously studied computer science was low, with 22 having no experience, 15 some courses either in high school or college, and 2 being self-taught. The high interest, combined with the age and education details, indicates that Minecraft is a game that appeals to a younger generation that is not yet in place to work in computers, but is definitely interested. Most of them had familiarity with some languages, especially web-based ones. HTML, Java, PHP, and C/C++/C# were the most commonly known languages in that order. Regarding video game interest, they varied greatly with a mean of 26.33 hours and median of 15 hours played per week. Participants felt overall very active (mean 6.2) and very comfortable (mean 6.76) playing games. Finally, their interests in Minecraft's creative, survival, social and challenging gameplay aspects resulted in all aspects having fairly high means. Creative was the highest (6.31), then challenging (5.63), social (5.23) and survival (5.11). Level of enjoyment overall was rated highly, with a mean of 5.89 and most people considered themselves active players (mean 4.89). We also asked several Yes-No questions about their involvement in Minecraft. As expected, they were heavily involved: 26 answered that they configured and ran their own server, 23 were admins and 15 developed custom plugins.

We feel these results indicated a potentially strong interest in CodeBlocks and computer programming in active players of Minecraft. Because of Minecraft's ability to foster creativity and its existing use of programmable Boolean logic, its players seem open to more programmatic control that CodeBlocks offers, as we were expecting.

7. CODEBLOCKS IN THE WILD

In addition to the experiment and survey, the CodeBlocks plugin has been freely released for download by players and server admins through a popular plugin repository, Bukkit (bukkit.org). We have also used the plugin in a day-long STEM education summer camp to middle school students to interest them in computer science.

7.1 Public Servers

We released and tracked CodeBlock's use by other Minecraft servers to determine what they used it for. We wanted to gauge Minecraft player interest, their use of the plugin, whether they could use it effectively and their opinions. It was released for 66 days. It was downloaded 319 times, and installed on 6 online servers. 27 distinct programs were created, and robots executed 133 programs in total. An example of a program created by a Minecraft player can be seen in Figure 5. On our website, users made comments such as "nice work", one stating "Absolutely Brilliant! Minecraft needs more original mods like this." One of the users mentioned that they had previous programming experience with Java, Squeak [8] and Scratch [17].

Figure 5. A program created by a player using CodeBlocks in a public Server. The program mines resources, a task often done by hand in Minecraft.

7.2 Middle School Summer Camp

We were involved in a STEM education summer camp at our University, where we demonstrated CodeBlocks and our server (our Minds of Chimera server is designed to support creative play) to Middle School students. Students participated in groups of approximately 25, with 10-15 of each group exploring CodeBlocks with the remainder exploring the server. We had 40 minutes with each group, which we used to teach the students how to use CodeBlocks and to help them create their own programs.

Student ability varied: some were avid Minecraft players and others uninterested in video games, but most students were able to follow along with a brief tutorial given by the author, with two other graduate students assisting. The tutorial taught them to create a simple program with a few instructions. On completion, students were allowed to play independently. Most students extended the tutorial program by adding additional instructions; however, some students used functionality that went beyond the tutorial, such as functions and branching. One student even defined recursive functions that allowed her to programmatically build a four-walled tower (see Figure 6).

At the end of each session, we gave the students a brief questionnaire, which asked them to describe what they liked and disliked about the system and to indicate on a 1 to 7 scale how

8. Ducasse, Stéphane. Squeak: Learn programming with robots. Apress, 2005.
9. Duncan, S. Minecraft, Beyond Construction and Survival.Well Played Journal, 1(1), 2011.
10. Eck, V. R. (2006). Digital game-based learning: It's not just the digital natives who are restless. Educause Review 41, 16.
11. Felder, R.M., and Soloman, B.A., Index of Learning Styles, http://www4.ncsu.edu/unity/lockers/ users/f/felder/public/ILSpage.html
12. Felder, R M and Silverman, L K. 1988. Learning and teaching styles in engineering education. Engineering Education, 78: 674–81
13. Howe, J. and O'Shea, T. 1978. Learning mathematics through LOGO. SIGCUE Outlook 12, 1,,p. 2-11.
14 Lee, M.J. and Ko, A.J. "Investigating the Role of Purposeful Goals on Novices' Engagement in a Programming Game", IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2012.
15 Light-Bot. http://armorgames.com/play/2205/light-bot, 2012
16. MacLaurin, B. 2011 The design of Kodu: A Tiny Visual Programming Language for Children on the XboX 360. SIGPLAN-SIGACT symposium on Principles of progr. languages, POPL, p. 241-245.
17. John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The Scratch Programming Language and Environment. Trans. Comput. Educ. 10, 4, Article 16 (November 2010), 15 pages.
18. Minecraft - http://minecraft.net, 2012
19. Pane, J. and Myers, B. 1996. Usability Issues in the Design of Novice Programming Systems, Carnegie Mellon University, School of Computer Science Technical Report CMU-CS-96-132, August, 85 pages.
20. Paras, B., and Bizzocchi, J. (2005). Game, motivation, and effective learning: An integrated model for educational game design. In Design (Citeseer), pp. 1-7.
21. Prensky, M. 2003. Digital game-based learning. Computer Entertain. 1, 1, p. 21.
22. Powers, K., Ecott, S. and Hirshfield, L. 2007. Through the looking glass: teaching CS0 with Alice. SIGCSE Bull. 39, 1 (March 2007), p. 213-217.
23. Powers, K., Gross, P., Cooper, S., McNally, M., Goldman, K., Proulx, V., and Carlisle, M. 2006. Tools for teaching introductory programming: what works?. In Proceedings of the 37th SIGCSE technical symposium on Computer science education (SIGCSE '06).
24. Repenning, Alexander. "AgentSheets®: An interactive simulation environment with end-user programmable agents." Interaction (2000).
25. Sandford, R., and Ulicsak, M., Facer, K. and Rudd, T. (2006) Teaching with Games: Using commercial off the-shelf computer games in formal education. Bristol. Futurelab.
26. Schaffer, D. W., Squire, K. D., Halverson, R., & Gee, J. P. 2004. Video games and the future of learning. *Phi Delta appan*, 87(2), 104-11.
27. Seals, Cheryl, et al. "Fun learning stagecast creator: an exercise in minimalism and collaboration." Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on. IEEE, 2002.
28. Sternberg, R. J. (ed). 1999. Handbook of Creativity, Cambridge University Press.
29. Stolee, K.T. 2011. Expressing Computer Science Concepts Through Kodu Game Lab. Computer science education, SIGCSE, p. 99-104.
30. Untch, R. H. 2011. Teaching Programming Using the Karel the Robot Paradigm Realized with a Conventional Language. Accessed December 10, 2011.
31. Whitton, N. 2007. Motivation and computer game based learning. Proc. of ASCILITE, 1063-1067.
32. Wingrave, C., Norton, J., Ross, C., Ochoa, N., Veazanchin, S., Charbonneau, E. and LaViola, J. 2012. Inspiring Creative Constructivist Play. CHI WIP.
33. CodeBlocks plugin. http://dev.bukkit.org/server-mods/codeblocks/