The Design of Co-Robotic Games for Computer Science Education

Ross Higashi*
National Robotics Engineering Center, Carnegie Mellon University
rhigashi@cmu.edu

Erik Harpstead
Human-Computer Interaction Institute, Carnegie Mellon University
harpstead@cmu.edu

Jaemarie Solyst
Human-Computer Interaction Institute, Carnegie Mellon University
jsolyst@andrew.cmu.edu

Jonaya Kemper
National Robotics Engineering Center, Carnegie Mellon University
jkemper@andrew.cmu.edu

Judith O. Uchidiuno
National Robotics Engineering Center, Carnegie Mellon University
jio@andrew.cmu.edu

Jessica Hammer
Human-Computer Interaction Institute, Carnegie Mellon University
hammerj@andrew.cmu.edu

ABSTRACT

Digital games featuring programmable agents are popular tools for teaching coding and computational thinking skills. However, today's games perpetuate an arguably obsolete relationship between programmable agents and human operators. Borrowing from the field of human-robotics interaction, we argue that collaborative robots, or cobots, are a better model for thinking about computational agents, working directly with humans rather than in place of or at arm's length from them. In this paper, we describe an initial design inquiry into the design of "cobot games", programmable agent scenarios in which players program an in-game ally to assist them in accomplishing gameplay objectives. We detail three questions that emerged out of this exploration, our present thinking on them, and plans for deepening inquiry into cobot game design moving forward.

CCS CONCEPTS

• **Applied computing** → Education; Interactive learning environments; • **Social and professional topics** → Professional topics; Computing education.

KEYWORDS

Games-based learning, Co-robotic games, Cobots, Robotics education

ACM Reference Format:
Ross Higashi*, Erik Harpstead, Jaemarie Solyst, Jonaya Kemper, Judith O. Uchidiuno, and Jessica Hammer. 2021. The Design of Co-Robotic Games for Computer Science Education. In *Extended Abstracts of the 2021 Annual Symposium on Computer-Human Interaction in Play (CHI PLAY '21), October 18–21, 2021, Virtual Event, Austria.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3450337.3483472

CHI PLAY '21, October 18–21, 2021, Virtual Event, Austria

ACM ISBN 978-1-4503-8356-1/21/10.
https://doi.org/10.1145/3450337.3483472

1 INTRODUCTION AND BACKGROUND

In the 21st century, computational knowledge and skills are power. K-12 school districts have increasingly recognized the importance of computing education, with many now making it a required topic across grades (e.g., Chicago Public Schools). Digital games and game-based experiences have a long track record as flagship CS and STEM learning experiences. Some of the earliest tools for teaching what we might now call computational thinking were playful and digital. Logo [6] served as a programming language for youth to playfully learn mathematics and computing concepts together by giving instructions to a turtle to draw shapes. Its descendant Scratch [16] is a cornerstone of programming education today. Hour of Code [17] and Code.org [18], among the most popular platforms for computing education, make heavy use of playful programming scenarios, such as programming virtual characters to dance to child-friendly hit songs. More recognizably "game"-shaped experiences like CodeCombat [19] leverage video game tropes (here, adventuring and fighting) to encourage the player to explore and fight in a fantastical world by programming a protagonist character.

Many of these experiences center around programmable agents. This is a sensible choice: an on-screen agent provides immediate visual feedback on the effects of the player's code [12]. However, in our review of relevant literature and online curriculum, we noted that programmable agents in coding games always seem to maintain the same fundamental relationship between the player/programmer and the programmed agent. Programmers write code, and onscreen agents run it to accomplish game objectives with no further interaction with the player. The relationship is always unidirectional and arm's length. Once the player gives the rabbit instructions in Google's Coding for Carrots [20], their exchange is done; the agent simply executes the directions, and the player wins or loses on that basis. Likewise Apple's Swift Playgrounds [21], Cartoon Network's Glitch Fixers [22], and so on. We propose that the time has come to re-examine this relationship.

1.1 Co-robotics

For context, we turn to the field of robotics for both analogical and practical reasons. One of the primary justifications for the importance of teaching computer science skills is their relevance to the job market. The rapid growth of automation will force as many as 375 million workers globally (14% of the workforce) to transition to new occupations and learn new skills [10]. In the U.S., 73 million

existing jobs will be eliminated, and an estimated 60% of those that remain could have up to 30% of their activities automated.

However, the robots of the coming generation will not look like those that replaced factory labor in the 20th and early 21st centuries. Instead, co-robotics applications in which robots work alongside humans in a meaningful, intuitive, and safe manner are now recognized by industry [7] and academia [14] as a primary direction for the growth of robotics technologies. Cooperative manufacturing robots work side-by-side with human workers, with tasks allocated based on whether they are more easily solved by human or machine intelligence. Outside of industry, socially assistive robots reduce isolation in seniors [1], support the learning of social behaviors for children with autism spectrum disorder [4], provide students with additional one-on-one educational opportunities at school [3], and support the needs of special needs populations [9]. Domestic service robots perform household tasks well with minimal intervention, and without any expectation of maintenance by a robot technician.

Co-robotic capabilities fall along a sliding scale of sophistication. While individual frameworks vary based on field, they generally take into account two critical factors: (1) whether the cobots share physical space with humans, and (2) how intertwined their respective tasks are. Michaelis, et al. [13] describe four steps along this scale, ranging from "No Interaction" where the robot is totally separated from humans by physical fencing, to "Collaborative" where the robot and human work in the same space at the same time while the robot learns from and adjusts to the human's behavior. El Zaatari, et al. [15] consider different manufacturing scenarios, ranging from "independent", where humans and robots never touch the same parts; or "simultaneous", where the cobot maneuvers around the human while working in a shared space; to "sequential", where one hands off work to the other; and ultimately highly interactive "supportive" workflows where the cobot and human perform mutually dependent tasks at the same time, e.g., the operator fastening screws that the cobot holds in place.

While there is imperfect agreement about the precise tiering of co-robotic capabilities, it is no coincidence that non-interactive scenarios are universally placed at the bottom. Such scenarios represent the past state of robotics, especially in manufacturing, where mass automation was the great industrial advance of the 1980s and 1990s. Robots were programmed once and left to run their tasks efficiently and separately from human workers. "Hands-off automation" is also a perfect – and unflattering – analogy for the human-robot interaction paradigm baked into nearly every programmable agent game in use today. If one of the goals of computing education is to build skills and understandings relevant to the job market of today, then it is time to consider modernizing the human-robotic interaction relationship embedded in computing education products. But how?

2 DESIGN EXPLORATION

Our work aims to address the gap in design knowledge around how cobots should be integrated into programming games. Since little is known about the design of cobot games at present, our first step was to explore the bounds of the design space. We launched two simultaneous design probes with the objective of creating coherent, functional cobot games via different design methods. The first probe used a traditional expert-driven game design approach in which multidisciplinary teams of game design and subject matter experts convened to ideate and produce games using conventional methods, including prototype user testing. The second probe used a participatory collaborative design (codesign) approach in which participants from the intended learner population were recruited to an 18-week afterschool game design program where they gave design input, direction, and feedback. This codesign activity also doubled as data collection for a broader research project, e.g., around prior programming experience and preconceptions of cobots.

2.1 Design Exploration 1: Expert-Driven Design

Our team of game design and robotics education experts employed conventional methods to rapidly generate, prototype, and consider a broad array of cobot game concepts. We call products of this approach Expert-Designed Games (EDGs).

2.1.1 First Iteration. Our expert-driven design approach began with members of our team brainstorming responses to the prompt "The Robot is a(n) ____ to the Player". This process generated 381 unique relationships, which were then grouped using an affinity clustering approach [11]. This produced categories of relationships such as "The Robot is a Teammate to the Player", which contained relationships like magician's assistant and band member, and "The Robot is a Protector to the Player", which contained relationships like shield and lifeguard.

The team then held an internal game jam to generate initial design concepts using these relationships. Members of our team were paired off, and each pair was randomly assigned a set of three design prompts consisting of a cobotic relationship and two gameplay verbs, one that the player would do and one that the cobot would do. Pairs were given approximately half an hour to design a game pitch for each of their prompts. Pitches were shared, remixed, and downselected through group scoring over several rounds. The game jam produced dozens of concepts, three of which were selected for production as technical and conceptual prototypes. Two were ultimately produced. These jam prototypes explored intentionally different visions of gameplay, cobot roles, and coding interactions.

2.1.2 Second Iteration. With the experience and technical foundation gained from the game jam iteration, we began a second development cycle toward a full-sized EDG. As before, the cycle began with brainstorming. Members of the team individually generated and pitched 10 game ideas with the goal of creating a transformational game [5] that would facilitate identity transformation from non-programmer to programmer, impart programming knowledge, and support socio-emotional learning. Pitches were analyzed and inductively coded [2] using a set of themes generated during discussion and agreed upon by all team members: mechanics, gameplay themes, transformations, playspace, barriers overcome, aesthetic, programming type, game genre, and cobot-player relationship ("The Robot is a(n) ____ to the Player"). Promising codes within each dimension (e.g., Aesthetic = Hopepunk) were voted on by group members and used to construct a second round of remixed pitches.

Ultimately, one concept from this second round was selected for full development.

2.2 Design Exploration 2: Participatory Codesign

In parallel with the expert-driven exploration, we developed a second cobot game through a participatory codesign process with youth at site-based afterschool programs. Codesign allows for incorporation of valuable and diverse insights from childrens' knowledge of childhood, understanding of their interests and enjoyment, domain knowledge, etc., in ways that adult designers are unlikely to replicate.

2.2.1 Codesign Process. Our research team partnered with a site-based afterschool club network located in the mid-Atlantic region of the United States, with around a dozen clubhouses spread throughout demographically diverse areas of the city. For our codesigned game, we worked with three urban and suburban clubhouses. A total of 36 students ages 6-10 participated in our program: 12 girls and 24 boys. Students were from diverse racial backgrounds including Caucasian (58%), African American (28%), and middle eastern and Latin American descent (14%).

HCI researchers with extensive adult-child codesign experience met students at each site for one hour weekly, either online or in-person per each site's COVID-19 protocols at the time. A second member of the research team took notes remotely. Sessions typically included snacks, an ice breaker question of the day, planned codesigned activities, and students playing video games from a curated selection to give them a more diverse game playing experience.

Each session's codesign activity prompted and collected student ideas, beliefs, or feedback on relevant concepts, such as what cobots might look like or do, what gameplay features students felt were most important, or how a prototype build might be improved. Ideas and suggestions were elaborated on by robotics and game design experts to ensure playability and congruence with subject matter, then integrated into subsequent pitches or builds on a weekly basis. Altogether, we engaged in 52 hours of codesign across the three sites. Written consent was obtained from the families of each attending student, and our research was approved by our institution's Institutional Review Board (IRB).

3 PRELIMINARY RESULTS AND DISCUSSION

Five unique game designs were produced to the playable prototype stage. Comparison across the designs and the experience of having produced them pointed to three focal areas in which we felt we had made progress, either answering or raising important questions.

3.1 How should cobots be integrated with the in-game narrative?

In all of our prototypes, the cobot was presented as diegetically ubiquitous. That is, inhabitants of the game world always treated cobots as if they were commonplace in those worlds. This design choice was deliberate, so that players would feel that even if programming the cobot was difficult, they would ultimately be able to succeed. In one game, an overarching narrative is established in which the player is coming of age aboard a multi-generational spaceship. Receiving, using, and learning to program the cobot are presented as expected activities tied to the player's coming of age. In a second game where much of the narrative is left implicit, players' families simply have cobots as pets, and programming them is implied to be a normal and expected behavior through inclusion as an unremarkable part of the character selection process. Diegetic ubiquity is sensible from a contextual perspective – the robot's role in the world, its available primitive commands, and others' reactions to it all follow internally consistent context cues.

However, the assumption that diagetic ubiquity is superior to having the cobot stand out as unique, is testworthy. It could indeed be more powerful to introduce the cobot's programmability and complementary gameplay as high-profile focal aspects that emphasize the importance and value of collaborative robotic arrangements.

3.2 What kinds of tasks should cobots be given in cobot games?

A large number of possible cobot-to-player relationships were proposed, suggesting that in principle, nearly any task characterization is possible. However, in practice, three of the five prototypes coalesced around cobots whose functionality was strongly aligned with the player's own existing goal structure, while the two that did not – i.e., that left the player to try and figure out what the cobot was for – suffered for it. Ultimately, both final prototypes directly used the verb structure of the player's character to define the verbs available to the cobot.

In the final prototype of the Expert-Designed Game pathway, for instance, the player is tasked to clear out boxes from a cargo bay aboard a large spacecraft. Gameplay starts with a tutorial explaining the keyboard controls (WASD+E) to pick up, move, and drop boxes by hand. The cobot is then introduced, along with point-and-click commands that allow it to perform Pick Up, Move, and Drop actions – the same as the player has just done manually. Programming is then introduced, featuring the same Pick Up, Move, and Drop commands but in the context of a block-based programming environment (Figure 1). This sequence cues the player toward an understanding of the cobot as a programmable agent capable of automating the player's own tasks using familiar command verbs in an identical context.

The second (codesign) exploration produced *Super Slime Battle*, a Halloween-themed base defense game in which players must stop waves of enemies from reaching the player's base (a pile of candy). Defeated enemies drop powerups that the player or cobot can collect by walking over them. The cobot's verbs, like the player's, stemmed naturally from its ability to move around the map, attack enemies, or pick up powerups. The implicit alignment of verbs served a larger purpose in *Super Slime Battle*. Simple commands with obvious mappings to in-game actions maintained a low cognitive load cost for reading, interpreting, and modifying code. This low friction was instrumental in lowering the barrier to code modification as part of a Use-Modify-Create [8] strategy described in Section 3.3.

In contrast, one of the early game jam prototypes we developed had an opaque set of commands and usage goals for the cobot. In *iSuffer (So You Don't Have To)*, the player's objective was to get through a maze shrouded in darkness and filled with enemies. The

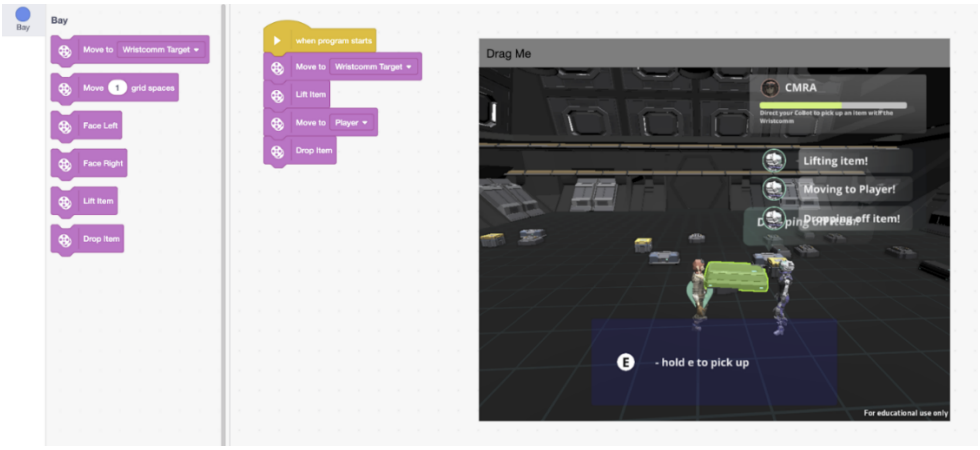

Figure 1: A screenshot of the box-moving game's programming view.

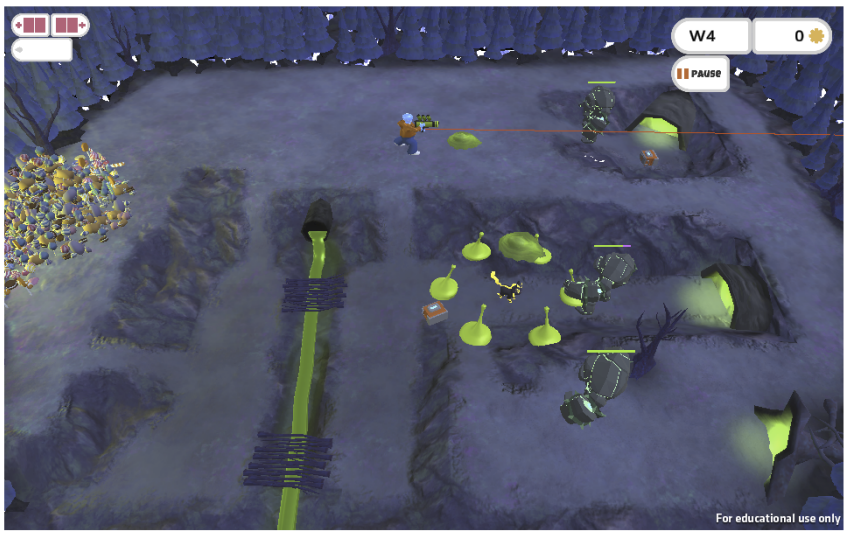

Figure 2: Super Slime Battle in progress. The player avatar (blue, top) is fighting to defend the candy pile base (left) from an enemy in the top right, while the cobot (cat, center right) uses its programmed attack to fight off two enemies at a crossroad.

cobot allowed the player to see by acting as a light source, but it could also be used as a distraction, as enemies would chase it instead of the player. This second use–having the cobot act as bait and frequently die–was almost totally hidden from the player at first and felt like an exploit when discovered. We suspect this owed in part to the goal dissonance between the player's character (who must live) and the cobot being sent to die intentionally.

Ultimately, the alignment of cobot verbs to player goals and verbs makes sense, as any game needs to ensure that players can comprehend the cobot as a mechanic. Thus, our preliminary conclusion is that cobot tasks will always "inherit" domains of action from the game worlds they inhabit, but that player understanding may benefit when cobot goals and verbs overlap the player's own.

3.3 How can games get players to engage with cobot programming?

The two main prototypes took distinctly different approaches toward getting players to program the cobot. The final prototype of the Expert-Designed Game used a didactic tutorial approach, which included mandatory programming steps. It included forcing functions such as automatically fullscreening the programming view at certain points in the tutorial, which shrinks the view of the player's surroundings down to a small picture-in-picture window. Even allowing for the presence of interface bugs, this approach was immensely unsuccessful. Few players wrote meaningful programs afterward, many did not even complete the required on-screen

steps, and a substantial number of playtesters ignored the programming window in its entirety and continued to play in the tiny picture-in-picture window for the remainder of the session.

The Codesigned Game, on the other hand, adopted a high-level framing that cobot program editing and selection would be treated as a step analogous to character selection, which we knew to be popular based on codesign feedback. This was supported by a Use-Modify-Create progression [8] embedded in the game's replay cycle. As players played (and lost, and replayed) the game, over 60% eventually tried editing the cobot code as a way of improving their performance in the next round.

In addition, players' cobots came seeded with deliberately inadequate default program options to encourage modification. We observed some students emphatically adding, e.g., a dozen "Attack" commands in a row to their program, echoing a codesign demand they had made earlier that "spam click" weapons be included in the game. We interpret this similarity as signifying the beginning of personal ownership and investment in upgrading the cobot's program in their own way.

We do not yet have strong evidence to conclude that a gameloop-embedded Use-Modify-Create design will always result in superior engagement compared to tutorialized instruction, but our preliminary results suggest that the former does work and is neither confusing nor off-putting to players. In short, it is an excellent candidate for future use.

4 LIMITATIONS AND FUTURE WORK

Our exploratory work to date has begun to suggest answers to some questions about the design of educational cobot games while raising new questions and uncovering new avenues for design. There is a need to collect outcomes data from an unbiased sample of players, so that the impressions and effects of gameplay can be separated from the effects of codesign participation. This effort is underway through the development of public online versions of the games with enhanced surveying and telemetry capability as well as creating new cobotic games with additional codesign cohorts, which will be released in the Fall of 2021.

One particularly important aspect we are also exploring is the reception of the games by minoritized youth in low-status settings, as well as the alternative cobot games they codesign. The cohort that codesigned the first CDG described in this paper is decidedly different from the students we are now working with at rural and majority-minority afterschool clubs. Not only have these students expressed a different relationship to STEM, but also different expectations and social pressures around video games. When asked to playtest the EDG and CDG prototypes, they described them as "trash" and "weak", confirming that these (or other yet unidentified) differences have major implications for the design of acceptable cobot games for these crucial populations. By extension, cobot games may also be of interest to older (even adult) non-programmer groups, who may similarly be put off by aesthetic, thematic, or mechanics choices designed in collaboration with 6-10 year olds.

A related observation that may provide some hope is that even within sites, there is tremendous diversity in which games individual students want and enjoy. Even though nearly all students report liking a handful of popular games at first, observation of their free play reveals that individuals gravitate toward a wide variety of games in different genres. This means that if we could design and produce enough games that appeal to different sub-audiences, we may succeed in reaching even those players who disliked our first few prototypes. It also broaches a broader topic worth exploring within cobot game design, which is whether there are genres that are more, less, or completely unsuitable for cobots. We are investigating this to a limited extent now through our second-round codesign efforts and will put the challenge before groups of talented game design students through our Fall courses.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1906753.

REFERENCES

[1] Ionut Anghel, Tudor Cioara, Dorin Moldovan, Marcel Antal, Claudia Daniela Pop, Ioan Salomie, Cristina Bianca Pop, and Viorica Rozina Chifu. 2020. Smart Environments and Social Robots for Age-Friendly Integrated Care Services. International Journal of Environmental Research and Public Health 17, 11. https://doi.org/10.3390/ijerph17113801

[2] Virginia Braun and Victoria Clarke. 2012. Thematic analysis. In APA handbook of research methods in psychology, Vol 2: Research designs: Quantitative, qualitative, neuropsychological, and biological. American Psychological Association, Washington, 57–71. https://doi.org/10.1037/13620-004

[3] Lavonda N. Brown and Ayanna M. Howard. 2014. The positive effects of verbal encouragement in mathematics education using a social robot. ISEC 2014 - 4th IEEE Integrated STEM Education Conference: 0–4. https://doi.org/10.1109/ISECon.2014.6891009

[4] John-John Cabibihan, Hifza Javed, Marcelo Ang, and Sharifah Mariam Aljunied. 2013. Why Robots? A Survey on the Roles and Benefits of Social Robots in the Therapy of Children with Autism. International Journal of Social Robotics 5, 4: 593–618. https://doi.org/10.1007/s12369-013-0202-2

[5] Sabrina Culyba. 2018. The Transformational Framework: A process tool for the development of Transformational games. Carnegie Mellon University.

[6] Wallace Feurzeig, Seymour Papert, Marjorie Bloom, Richard Grant, and Cynthia Solomon. 1970. Programming-languages as a conceptual framework for teaching mathematics. ACM SIGCUE Outlook 4, 2: 13–17.

[7] R. Colin Johnson. 2012. "Co-robots" help boost human productivity. Electronic Engineering Times, 1626: 24–28.

[8] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. ACM Inroads 2, 1: 32–37. https://doi.org/10.1145/1929887.1929902

[9] Nadia Magnenat-Thalmann and Zhijun Zhang. 2014. Assistive social robots for people with special needs. In *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, 1374–1380. https://doi.org/10.1109/IC3I.2014.7019828

[10] James Manyika, Susan Lund, Michael Chui, Jacques Bughin, Jonathan Woetzel, Parul Batra, Ryan Ko, and Saurabh Sanghvi. 2017. Jobs lost, jobs gained: Workforce transitions in a time of automation. McKinsey Global Institute 150.

[11] Bella Martin and Bruce Hanington. 2012. Uniersal Methods of Design. Rockport Publishers.

[12] Orni Meerbaum-Salant, Michal Armoni, and Mordechai (Moti) Ben-Ari. 2010. Learning computer science concepts with scratch. In *Proceedings of the Sixth international workshop on Computing education research - ICER '10*, 69. https://doi.org/10.1145/1839594.1839607

[13] Joseph E. Michaelis, Amanda Siebert-Evenstone, David Williamson Shaffer, and Bilge Mutlu. 2020. Collaborative or Simply Uncaged? Understanding Human-Cobot Interactions in Automation. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–12. https://doi.org/10.1145/3313831.3376547

[14] Brian Scassellati and Katherine M. Tsui. 2016. Co-Robots: Humans and Robots Operating as Partners. In *Handbook of Science and Technology Convergence*. Springer International Publishing, Cham, 427–439. https://doi.org/10.1007/978-3-319-07052-0_27

[15] Shirine El Zaatari, Mohamed Marei, Weidong Li, and Zahid Usman. 2019. Cobot programming for collaborative industrial tasks: An overview. Robotics and Autonomous Systems 116: 162–180. https://doi.org/10.1016/j.robot.2019.03.003

[16] Scratch. Retrieved from https://scratch.mit.edu/

[17] Hour of Code. Retrieved from https://code.org/learn

[18] Code.org. Retrieved from https://code.org/
[19] CodeCombat. Retrieved from https://codecombat.com/play
[20] Coding for Carrots. Retrieved from https://www.google.com/doodles/celebrating-50-years-of-kids-coding
[21] Swift Playgrounds. Retrieved from https://www.apple.com/swift/playgrounds/
[22] Glitch Fixers. Retrieved from https://www.cartoonnetwork.com/games/powerpuff-girls/glitch-fixers/index.html