Dragon Architect: Open Design Problems for Guided Learning in a Creative Computational Thinking Sandbox Game

Aaron Bauer
awb@cs.washington.edu
Paul G. Allen School of Computer Science and Engineering
University of Washington

Eric Butler
edbutler@cs.washington.edu
Paul G. Allen School of Computer Science and Engineering
University of Washington

Zoran Popović
zoran@cs.washington.edu
Paul G. Allen School of Computer Science and Engineering
University of Washington

ABSTRACT

Educational games have a potentially significant role to play in the increasing efforts to expand access to computer science education. *Computational thinking* is an area of particular interest, including the development of problem-solving strategies like divide and conquer. Existing games designed to teach computational thinking generally consist of either open-ended exploration with little direct guidance or a linear series of puzzles with lots of direct guidance, but little exploration. Educational research indicates that the most effective approach may be a hybrid of these two structures. We present *Dragon Architect*, an educational computational thinking game, and use it as context for a discussion of key open problems in the design of games to teach computational thinking. These problems include how to directly teach computational thinking strategies, how to achieve a balance between exploration and direct guidance, and how to incorporate engaging social features. We also discuss several important design challenges we have encountered during the design of *Dragon Architect*. We contend the problems we describe are relevant to anyone making educational games or systems that need to teach complex concepts and skills.

CCS CONCEPTS

•Applied computing → Interactive learning environments;

KEYWORDS

game-based learning; computational thinking; programming education

ACM Reference format:
Aaron Bauer, Eric Butler, and Zoran Popović. 2017. Dragon Architect: Open Design Problems for Guided
Learning in a Creative Computational Thinking Sandbox Game. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 7 pages.
DOI: 10.1145/3102071.3102106

FDG'17, Hyannis, MA, USA

978-1-4503-5319-9/17/08…$15.00
DOI: 10.1145/3102071.3102106

1 INTRODUCTION

The past few years have seen increasing efforts to broaden the reach of computer science education and make it available, and accessible, to more students in more places. Educational games have a potentially powerful and exciting role to play in this expansion. Game environments can provide an educational setting possessing many attributes known to support better learning outcomes, such as varied practice [11], immediate feedback [22], intrinsically motivating goals [14], and an engaging social context [5]. Well-designed, carefully scaffolded educational games that leverage these attributes could help address the current lack of wide-spread access to computer science education [18].

Existing educational programming environments and games employ myriad approaches to incorporating and teaching computational concepts [21, 34]. Despite this variation, the structure of these systems generally falls into one of two groups: (1) an open-ended setting with little to no direct guidance where players are intended to learn via exploration and from instructors or other members of the social community, or (2) a linear series of puzzles or exercises with substantial direct guidance, but little in the way of exploration of social interaction. Both structures have benefits, but the educational literature suggests that a hybrid structure, one that combines direct guidance with creative, player-driven exploration, may be the most effective [25, 28]. In this work, we explore some key open problems and design questions that arise when applying this hybrid structure to the computer science domain, using the context of the design of our educational game with this hybrid structure, *Dragon Architect*.

Computational thinking has received significant attention in the context of expanding access to computer science as a crucial area of conceptual knowledge. The literature contains numerous definitions of computational thinking, and though no consensus has been reached [13], the use of certain problem-solving strategies, or *computational practices* [4], has emerged as a central theme. These strategies include skills such as "being incremental and iterative" or "using abstraction and modularization." How to directly teach such skills in an educational game is an open problem, and we discuss this question and how it impacts the design of *Dragon Architect*.

Our contributions are as follows: we survey research on teaching computational thinking in software systems and describe the open problems in that space. We then survey and discuss related research problems in game-based learning. We are currently creating *Dragon Architect*, an educational, creative sandbox game designed to investigate these questions. We describe design trade-offs and decisions in *Dragon Architect*, focusing on a few unsolved design obstacles: how to combine open-ended sandbox games with direct guidance,

and how to encourage a social environment in a programming game. This paper focuses on the survey and discussion of open research problems in teaching computational thinking through games, and the design challenges and trade-offs encountered when tackling these problems, rather than our specific game per se. We believe the problems we describe are relevant to anyone making educational games or systems that need to teach complex concepts and skills.

2 GAME DESCRIPTION

Before we discuss open research problems and design challenges, we describe basic information about *Dragon Architect* to provide context for the discussion. In *Dragon Architect*, played in a web browser, players write code to control a dragon that builds 3D structures in a block world. Similar to other programming games, the user interface is separated into two parts: an area where the player assembles code and a visualization of the 3D environment their code affects (see Figure 2). The game uses the *three.js* library for the 3D environment and the Blockly drag-and-drop programming library for inputing code.

Players use a visual language made up of *code blocks* that snap together to form programs. The player can use these blocks to move the dragon in three dimensions and direct it to place and remove cubes of various colors. In addition to blocks that control the dragon directly, players can use definite loops and procedures (see Figure 1). Given that syntax can be an obstacle for those new to programming [35], we chose a visual language given the evidence they are helpful to novices [37].

As players progress through the game, they alternate between short sequences of puzzles (1–4 puzzles long) with a specific goal and restricted set of code blocks and an open-ended sandbox. The game begins with puzzles that introduce the idea of assembling and running code, as well as the code blocks for moving the dragon and placing cubes. After that, the player can creatively experiment and build in the sandbox and complete other puzzle sequences to make more code blocks available, switching between sandbox and puzzles at any time. In this way, the language the player uses to control the dragon gradually expands as the player advances.

The popularity and uniquely broad appeal of *Minecraft* (Mojang, 2011) motivated our use of a 3D grid world in which the player's programs could place cubes. *Minecraft* embodies many desirable attributes including creative, player-directed construction and exploration and a compelling social context delivered via a multiplayer mode. Our aim is to retain these attributes while introducing the additional structure necessary to best support learning. Using *Minecraft* as an inspiration also suggests natural future extensions to our game such as exploration, more complex interaction with the environment, and players working together in a shared world. Our playtests with *Dragon Architect* have supported the appeal of programming a dragon in a *Minecraft*-like world with younger players of all genders. Common sandbox activities have included making the dragon travel very long distances, building big and impressive towers, and spelling one's name out of cubes.

3 RESEARCH QUESTIONS

Our primary goal for *Dragon Architect* is to investigate questions in computer science education and game-based learning. Specifically,

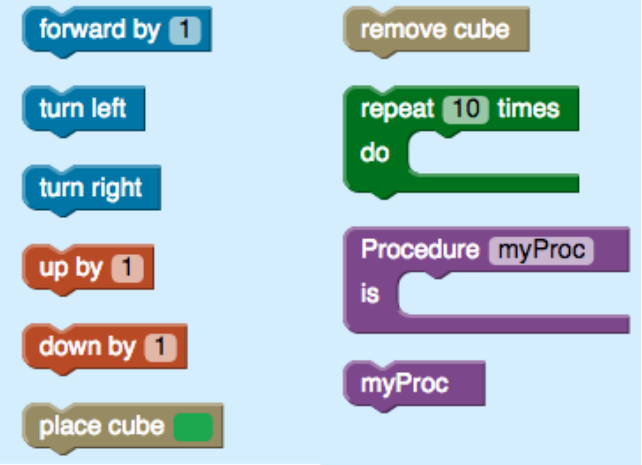

Figure 1: The programming elements available in *Dragon Architect*, which include moving the dragon, placing blocks, definite loops, and procedure definitions.

we are interested in exploring how to teach computational thinking skills and problem-solving strategies. Investigating these questions requires answers to larger research questions on how to effectively structure educational game. Open-ended games and systems typically perform poorly at teaching complex concepts without instructors or other outside help, but linear, direct-guidance-based games and systems don't offer the engaging creative and social experiences that can be found in more open-ended settings. In this section, we describe these research problems in detail along with related design trade-offs.

3.1 Teaching Computational Thinking

A major goal of our project is to explore methods of teaching computational thinking skills. Many have studied how to increase the presence and effectiveness of computational thinking in computer science education (and education in general) (e.g., Barr and Stephenson [2]), and others have developed games to teach these ideas (e.g., [20]). A recent review of the literature on teaching computational thinking found that additional research is needed [26].

Dragon Architect's design is structured to encourage and require use of computational thinking skills. Like many programming games, we force the player to automate tasks that they are used to performing manually (in this case, the construction of 3D block structures). As the player sets more sophisticated goals, modularity becomes important (e.g., putting the building of a wall into a procedure) to keep the visual programming feasible.

As we cannot expect players to learn such complex skills simply by playing in an environment that requires those skills [28], we must address how to directly teach computational thinking skills. A core component of computational thinking is the identification and application of problem-solving strategies. A great deal of recent education research suggests that "curricula can model such strategies for students" and that appropriate guidance, which in many cases consists of the capabilities afforded by a suitable computational environment, can "enable students to learn to use these

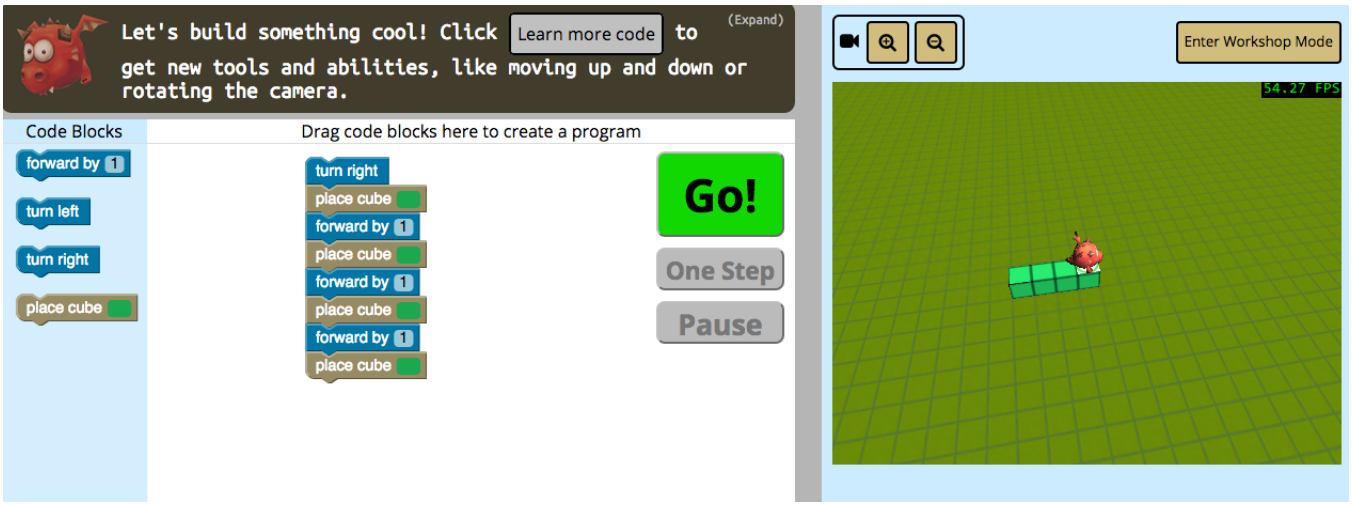

Figure 2: The player assembles code to control the dragon on the left side, and the game world is visualized on the right side. Only a few different code blocks are available to the player initially, and more are unlocked by completing additional puzzles.

strategies independently" [30]. Mayer and Wittrock call attention to the substantial evidence in the education literature for teaching what they call *domain-specific thinking skills* and *metacognitive skills* [29]. An example of the former is the ability to use a strategy like divide and conquer, and the latter would include knowing when and where to employ that strategy. In both cases, Mayer and Wittrock describe studies (for non-computer science domains) that have shown teaching these skills directly can improve learning and performance. It is an open question whether this can be applied to teaching computational thinking in a game.

One strategy we have focused on in *Dragon Architect* is *divide and conquer*. Our initial attempt to directly teach divide and conquer is to lead the player though a top-down deconstruction of building a large castle. The player is presented with a single code block that builds an entire castle, but discovers the construction has a number of flaws. The next several puzzles each decompose some part of the flawed program in order to give the player a chance to repair it. For example, to enable the player to give the castle the correct number of walls and towers, the castle code block is split into a tower block and a wall block that the player uses to write a corrected castle procedure, as shown Figure 3. This part of *Dragon Architect* needs to be expanded and refined before it can be evaluated, and the larger question merits further attention.

3.2 Structure of Learning Environment

3.2.1 Guided Discovery Learning. Teaching complex skills such as computational thinking or programming though interactive software is very challenging, and any effort to do so in a game naturally leads to research questions about which structures are most effective. For example, some games are structured as a sandbox where players discover the properties of the environment through largely unguided exploration (e.g., *Minecraft*, *SimCity* (Maxis, 1989)), while others provide a linear sequence of levels designed to teach the player the relevant information (e.g., *Portal* (Valve, 2007)).

In terms of learning theory, the former is known as *discovery learning*, in which learning takes place through exploration with the object of study. Discovery learning typically has a motivational advantage over a traditional teaching approach: directly exploring and interacting with something can be substantially more engaging than hearing someone talk about it. Furthermore, this allows learning domains to be grounded in an application of interest to the learner, creating meaning and intrinsic interest in developing the desired skills. While the pure form of this approach can work for simple learning domains, complex ones such as programming or computational thinking cannot be taught through exploration alone unless players have sufficient background knowledge [23].

The recommended practice is to use *guided discovery*, in which discovery learning is paired with some kind of external direct guidance [28]. For example, in games, players might consult wikis or ask friends to help them develop high-level skills. This approach of combining settings for discovery learning with guidance is a central part of *constructionism*, a learning theory that proposes students learn effectively by constructing things of social relevance in a social context [19]. *Scratch* [27], which enables users to create interactive digital media projects such as stories and games, is among the most popular of systems designed along constructionist principles. *Scratch*'s open-ended creative environment allows player to pursue meaningful creative projects, giving them motivation to learn the skills required to do so. However, the tool cannot effectively teach these skills in isolation; it is designed such that instructors or the social community help teach new users.

On the other hand, more structured systems can more effectively teach skills without external guidance by limiting player freedom. Linear games such as *Portal* introduce concepts in a deliberate ordering and pacing to allow players to develop the necessary skills. Methods of direct instruction, such as classroom lectures, also fall into this category. Online systems such as Code.org pair sequences of puzzles with videos to explain and teach the concepts.

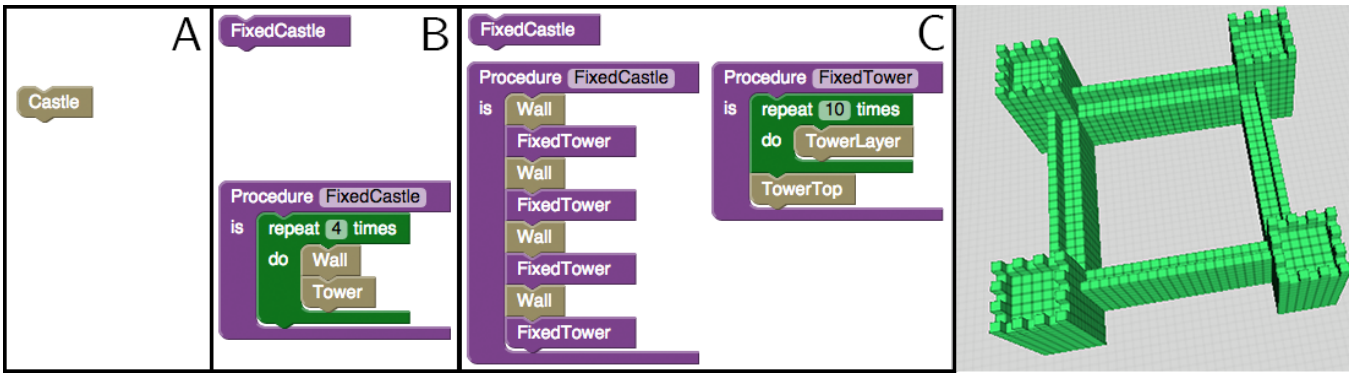

Figure 3: The code required by a progression of levels demonstrating the strategy of divide and conquer. In A, the player uses a single code block to build an entire castle. Then, in B, the player is given an empty `FixedCastle` procedure, which they must fill with the appropriate number of wall and tower blocks. Finally, in C, the player is given a completed `FixedCastle` procedure and must fill in the `FixedTower` procedure as shown. The final completed castle is shown on the right.

Other games teach players by doing something in between. Games in the Zelda series (e.g., *The Legend of Zelda: Ocarina of Time* (Nintendo, 1998)) contain a non-linear sequence of puzzles and teach new mechanics by requiring the player demonstrate understanding of a new mechanic before they are allowed to progress. Strategy games such as *Crusader Kings II* (Paradox Development Studio, 2012) offer a set of explicit tutorials the player may choose to go through before beginning more open-ended play.

Both types of systems (open-ended vs. highly-structured) have highly desirable benefits. The open research problem we are interested in is how to create a hybrid model that best combines benefits of both approaches, particularly in settings where we cannot rely on an instructor. Our initial approach joins an open sandbox with small sequences of puzzles that directly guide players. Our preliminary playtesting has shown this to be a promising approach, though this encounters several design challenges. We discuss the details and associated trade-offs in Section 4.1. The primary challenge with direct guidance in an open-ended game is providing guidance at the appropriate moment; effective games provide help on-demand and just-in-time [10].

3.2.2 Social Environments. Social environments can be an effective way to provide guided discovery learning. Social communities both create motivation to learn (to participate in the social group) and guidance (by getting help from the group). *Scratch* users, for example, have shared millions of projects and even formed online companies to tackle projects together [33]. Research using an online programming environment called MOOSE Crossing [6] found that a social context can support and motivate learning programming in a shared environment [5]. This work is supported by research in behavioral and social sciences that indicates sharing and collaboration can improve learning [3]. Open-world multiplayer games such as *Minecraft* and *World of Warcraft* (Blizzard, 2004) have been used as a virtual classroom environment in schools [12]. Players on the same *Minecraft* server are able to share their work, collaborate with others, and help directly guide others.

Dragon Architect incorporates some project-sharing features from *Scratch*: players can share what they build in the sandbox in a communal gallery and browse, view and download others' creations. Supporting closer collaboration activities, such as pair programming in *Dragon Architect*'s *Minecraft*-like world, could dramatically enhance the social element of our game. This presents several design challenges, however, which we describe in Section 4.2.

4 OPEN DESIGN PROBLEMS

Dragon Architect's design and research goals have led to several tensions and difficulties. In this section we discuss cases where the goals have presented unusual challenges, and relevant results from our preliminary playtesting.

4.1 Guided Learning in a Sandbox

In *Dragon Architect*, our design goal is to center the experience in an open sandbox (like *Minecraft*) but provide on-demand, just-in-time direct guidance for new tools and concepts in the form of small puzzle segments.

The ideal we are trying to emulate, like much educational software, is a personal human tutor. An effective tutor can allow the player to explore in the sandbox, and, once the player expresses the need for a tool or feature they do not yet possess, the tutor can guide them to the appropriate learning material. Knowing when to intervene and what the player is trying to do is a challenge. *Intelligent tutoring systems* accomplish this by tracking player knowledge and skill, which requires both a model of all conceptual knowledge and a rigid structure of problems with known solution strategies [24]. As we have neither of these, this is not an option.

Dragon Architect must strike a difficult balance: get the player to the sandbox where they can experiment as quickly and often as possible, but support the player in the gradual acquisition of new knowledge and skills. We do not wish to overwhelm new players with the full suite of features and tools upon start, so we lock many features behind puzzles until the player has a chance to learn about them in a guided, isolated setting. This is in tension with the desire to quickly allow a player access to the tools they need without jumping through hoops and challenges. Systems like *Scratch* are fully open from the start, and expect an expert to guide newcomers

through the system. We aim to avoid making learning in *Dragon Architect* substantially contingent on external support.

Empirically, players initially struggle with *Dragon Architect*'s hybrid structure, but enjoy it once it is explained to them. Very few players discover the puzzles without external guidance, a clear failure point if we wish the system to function without an instructor. In playtests, players would often ask if they game supports a feature (e.g., moving up and down, removing blocks), when such a feature was directly advertised in a list of available modules. After being informed of this, players began to understand the basic cycle between sandbox and puzzles and started checking for other skills they could learn through puzzles.

We have also found that creating in the sandbox is not necessarily the activity of choice for every player. Some choose to doodle in the sandbox, scattering cubes, often of different colors, in lines and clusters without trying to assemble anything in particular. Others exclusively seek out the game's puzzles, more interesting in solving those than working in the sandbox. Its hybrid structure allows *Dragon Architect* to appeal to players with either interest.

4.2 Collaboration in a Multiplayer Environment

Social environments are an important part of guiding discovery learning and a major goal of our project. The goal is to create a shared virtual environment in which players can assist each other and collaborate. This works naturally in multiplayer games such as *Minecraft*, where players on the same game server can explore and shape a persistent world together. This shared world has several benefits. It is easy for players to share their creations with others, but at the same time work relatively independently. Changes to the world are localized around the player, so one player does not interfere with those far away. At the same time, working closely together is as simple as walking to the same place in the world.

Such interactions could be invaluable for programming, supporting activities such as pair programming and mentoring. However, the persistent world model conflicts with other design considerations. It is very easy to accidentally (or intentionally) write programs that have non-localized effects, interfering with others in the shared world. Programming often relies on rerunning the same program over and over to iterate and find bugs, but this requires resetting the world state to a consistent start state. While such a reset or undo feature is an option, if a player writes code that places blocks all over another player's work, how should the game reset state in a consistent and understandable manner? This tension is even present in a single-player setting. Playtesters that wanted to construct complex objects, such as a town of buildings, often wanted to do so piecemeal. They write separate programs to build each part without wanting to reset the world to a static start state.

Our current approach does not sufficiently resolve this tension, and only addresses the single-player setting. The sandbox makes the effects of programs permanent by default, including any cubes placed and the dragon's position. To allow for experimentation, the player has the option to switch to *workshop mode*, which resets the world to its previous state each time the player runs a new program. The player can freely toggle between these two modes, testing out each program before committing to its results. In practice, players have a difficult time understanding the nuanced difference between the two modes, despite dramatic visual cues. The feature is empirically both difficult to discover and difficult to understand even once explained.

In contrast, *Dragon Architect*'s gallery has successfully promoted a shared social context during playtesting. After players have a chance to get familiar with the game and begin sharing their creations in the gallery, it serves as a catalyst for players to talk to each other about what they are creating, go over to other players' computers to see how something was done, and remix or incorporate things posted to the gallery into their own work.

5 RELATED WORK

The development of tools designed to teach novices computational thinking dates back to systems such as Papert's LOGO [31]. Kelleher and Pausch review this extensive body of work and describe a taxonomy of these systems [21]. Within that taxonomy, *Dragon Architect* fits best as a teaching system that targets *structuring programs* and aims to provide learning support both through *social learning* and *providing a motivating context*. In this section, we summarize the current space of educational computational thinking tools. For a more comprehensive survey, see Kelleher and Pausch's review or Salleh et al.'s more recent review [34].

Both early tools like LOGO and recent tools like *Scratch* have an open-ended and creative approach. LOGO allowed players to create drawings by controlling a robot with a virtual pen. While LOGO was text-based, many modern examples use visual programming languages. *Alice* [7], like *Scratch*, focuses on storytelling, though it does so in a 3D animation context with more fine-grained control than Scratch offers. Other systems such as *AgentSheets* [32] let users create simulations and games. *AgentSheets* models its world as agents on a grid, and players can program the behavior of each type of agent, conditioning behavior on the contents of adjacent grid cells. There are also educational programming games that have their players tackle open-ended challenges. *CodeSpells* [8] is a game in which players write Java code to cast spells that control their environment. In *RoboBuilder* [36], players program robots to battle against enemies. *BlockStudio* [1] lets users create rule-based games and simulations using programming by demonstration.

Another type of computational thinking educational system is instead arranged as a linear sequence of problems or puzzles. Step-by-step lessons are available from Khan Academy and Codecademy, in which users program in an industry programming language such as JavaScript or Java, sometimes with accompanying video. Code.org provides sequences of videos and puzzles where users control characters from popular games like Rovio's *Angry Birds* or movies like Disney's *Frozen* with drag-and-drop programming. Using programming by demonstration to seamlessly integrate gameplay and educational content, *The Orb Game* [9] teaches players algorithms for various list operations. *BOTS*, a programming puzzle game, has been used to study user-generated educational content in terms of submission requirements [15] and level editor design [16].

Games can be educational tools by design, or be adapted to educational purposes. *Minecraft*'s popularity and its ability to engage children have prompted several efforts to use it as an educational tool. *Minecraft Education Edition* (Mojang, 2017) is an official variant

of *Minecraft* with a number of classroom-centric features. Teachers have used it in lessons on math, English language arts, computer science, and other subjects. Zorn et al. created *CodeBlocks*, a plugin for Minecraft that allows players to program a robot within the game, and they found that *CodeBlocks* increased non-programmers' interest in programming [38]. Another avenue is to teach programming by having students create modifications and plugins for *Minecraft*, through mods such as *ScriptCraft* [17].

6 CONCLUSION

Educational games have enormous potential to enhance wide-spread access to an engaging introduction to computational thinking. We have discussed crucial open questions related to effectively teaching computational thinking in games. Evidence from the educational literature argues for the direct teaching of problem-solving strategies such as divide and conquer, and we described a initial attempt to do so in our computational thinking game, *Dragon Architect*. The educational literature also highlights the need to combine open-ended exploration with sufficient structured guidance. In order to avoid requiring the presence of an instructor or mentor, an educational game like *Dragon Architect* must find a way to provide a flexible progression with both player-driven exploration and direct instruction. We addressed this thorny design problem by having players alternate at their own pace between experimenting in a sandbox and unlocking new features with short sequences of puzzles, but our preliminary playtests indicated players need additional scaffolding to easily understand this dynamic. Finally, education research makes clear the benefits of incorporating a social context into the learning environment. *Dragon Architect* provides players a way to share what they create and see what others have shared, but pushing further and supporting real-time collaboration in a shared world presents significant design challenges.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation under Grant No.: DRL-1639576 and Grant No.: DGE-1546510 and by the Oak Foundation under Grant No.: 16-644.

REFERENCES

[1] Rahul Banerjee, Jason Yip, Kung Jin Lee, and Zoran Popović. 2016. Empowering Children To Rapidly Author Games and Animations Without Writing Code. In *Proceedings of the The 15th International Conference on Interaction Design and Children*. ACM, 230–237.

[2] Valerie Barr and Chris Stephenson. 2011. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community? *ACM Inroads* 2, 1 (2011), 48–54.

[3] John D Bransford, Ann L Brown, Rodney R Cocking, et al. 2000. How people learn. (2000), 219-220 pages.

[4] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*.

[5] Amy Bruckman. 2000. Situated support for learning: Storm's weekend with Rachael. *The Journal of the Learning Sciences* 9, 3 (2000), 329–372.

[6] Amy Susan Bruckman. 1997. *MOOSE Crossing: Construction, community, and learning in a networked virtual world for kids*. Ph.D. Dissertation. MIT.

[7] Stephen Cooper, Wanda Dann, and Randy Pausch. 2000. Alice: a 3-D tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, Vol. 15. Consortium for Computing Sciences in Colleges, 107–116.

[8] Sarah Esper, Stephen R Foster, and William G Griswold. 2013. On the nature of fires and how to spark them when you're not there. In *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 305–310.

[9] Stephen R Foster, Sorin Lerner, and William G Griswold. 2015. Seamless Integration of Coding and Gameplay: Writing Code Without Knowing it.. In *FDG*.

[10] James Paul Gee. 2003. What video games have to teach us about learning and literacy. *Computers in Entertainment (CIE)* 1, 1 (2003), 20–20.

[11] Douglas A Gentile and J Ronald Gentile. 2008. Violent video games as exemplary teachers: A conceptual analysis. *Journal of Youth and Adolescence* 37, 2 (2008).

[12] Lucas Gillispie. 2014. From Students to Heroes: Unlocking Students' Potential Through Games. In *Procedings of the 9th International Conference on the Foundations of Digital Games*. Keynote Presentation.

[13] Shuchi Grover and Roy Pea. 2013. Computational Thinking in K–12 A Review of the State of the Field. *Educational Researcher* 42, 1 (2013), 38–43.

[14] MP Jacob Habgood and Shaaron E Ainsworth. 2011. Motivating children to learn effectively: Exploring the value of intrinsic integration in educational games. *The Journal of the Learning Sciences* 20, 2 (2011), 169–206.

[15] Andrew Hicks, Veronica Cateté, and Tiffany Barnes. 2014. Part of the Game: Changing Level Creation to Identify and Filter Low Quality User-Generated Levels. In *9th International Conference on the Foundations of Digital Games*.

[16] Drew Hicks, Zhongxiu Liu, and Tiffany Barnes. 2016. Measuring Gameplay Affordances of User-Generated Content in an Educational Game. In *9th International Conference on Educational Data Mining*.

[17] Walter Higgins. 2016. ScriptCraft. http://scriptcraftjs.org/. (2016). Accessed: 2017-02-19.

[18] Gallup Inc. and Google. 2015. Searching for Computer Science: Access and Barriers in U.S. K-12 Education. (2015). https://services.google.com/fh/files/misc/searching-for-computer-science_report.pdf

[19] Yasmin B. Kafai. 2006. Constructionism. In *Cambridge Handbook of the Learning Sciences*, R. Keith Sawyer (Ed.). Cambridge University Press, 35–46.

[20] Cagin Kazimoglu, Mary Kiernan, Liz Bacon, and Lachlan Mackinnon. 2012. A serious game for developing computational thinking and learning introductory computer programming. *Procedia-Social and Behavioral Sciences* 47 (2012).

[21] Caitlin Kelleher and Randy Pausch. 2005. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys (CSUR)* 37, 2 (2005), 83–137.

[22] John Kirriemuir. 2002. The relevance of video games and gaming consoles to the Higher and Further Education learning experience. *JISC Techwatch report* (2002).

[23] Paul A Kirschner, John Sweller, and Richard E Clark. 2006. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational psychologist* 41, 2 (2006), 75–86.

[24] Kenneth R. Koedinger and Albert Corbett. 2006. Cognitive Tutors: Technology Bringing Learning Sciences to the Classroom. In *Cambridge Handbook of the Learning Sciences*, R. Keith Sawyer (Ed.). Cambridge University Press, 61–77.

[25] Detlev Leutner. 1993. Guided discovery learning with computer-based simulation games: Effects of adaptive and non-adaptive instructional support. *Learning and Instruction* 3, 2 (1993), 113–132.

[26] Sze Yee Lye and Joyce Hwee Ling Koh. 2014. Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior* 41 (2014), 51–61.

[27] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. 2010. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)* 10, 4 (2010), 16.

[28] Richard E Mayer. 2004. Should there be a three-strikes rule against pure discovery learning? *American Psychologist* 59, 1 (2004), 14.

[29] Richard E Mayer and Merlin C Wittrock. 1996. Problem Solving Transfer. In *Handbook of educational psychology*, David C Berliner and Robert C Calfee (Eds.). Routledge.

[30] National Research Council. 2010. *Report of a Workshop on the Scope and Nature of Computational Thinking*. National Academies Press.

[31] Seymour Papert. 1980. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., New York, NY, USA.

[32] Alexander Repenning, Andri Ioannidou, and John Zola. 2000. AgentSheets: End-user programmable simulations. *Journal of Artificial Societies and Social Simulation* 3, 3 (2000), 351–358.

[33] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. 2009. Scratch: programming for all. *Commun. ACM* (2009).

[34] Syahanim Mohd Salleh, Zarina Shukur, and Hairulliza Mohamad Judi. 2013. Analysis of Research in Programming Teaching Tools: An Initial Review. *Procedia-Social and Behavioral Sciences* 103 (2013), 127–135.

[35] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *Trans. Comput. Educ.* 13, 4, Article 19 (Nov. 2013), 40 pages. https://doi.org/10.1145/2534973

[36] David Weintrop and Uri Wilensky. 2013. Robobuilder: a computational thinking game. In *SIGCSE*. 736.

[37] Kirsten N. Whitley. 1997. Visual programming languages and the empirical evidence for and against. *Journal of Visual Languages & Computing* 8, 1 (1997).

[38] Christopher Zorn, Chadwick A Wingrave, Emiko Charbonneau, and Joseph J LaViola Jr. 2013. Exploring Minecraft as a conduit for increasing interest in programming.. In *FDG*. Citeseer, 352–359.