Idea Builder: Motivating Idea Generation and Planning through Storyboarding

ABSTRACT

In computing classrooms, building an open-ended programming project engages students in the process of designing and implementing an idea of their own choice. An explicit planning process has been shown to help students build more complex and ambitious open-ended projects. However, novices encounter difficulties to explore and express ideas during planning. We present Idea Builder, a storyboarding-based planning system to help novices use drawings and images to express their ideas. Idea Builder connects code to plans by showing students animations of code examples (called "planning examples") when selecting assets, and connects plans to code by automatically converts a student's storyboard to starter code. Our case studies with two high school coding workshops found that Idea Builder helped students self-perceived as feeling creative and expressive during planning; planning examples encouraged students to plan a mechanic and use a code example more effectively; converting storyboards to starter code were generally perceived useful and time-saving to students.

ACM Reference Format:

. 2023. Idea Builder: Motivating Idea Generation and Planning through Storyboarding. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In computing classrooms, building an open-ended programming project engages students in the process of designing and implementing an idea of their own choice [8]. It motivates students by placing learning in a real-world context, and encourages students to pursue solutions in the form of self-driven activities, including generating and debating ideas, designing solutions, and implementing their plans [2]. As a result, open-ended programming projects are widely-used in introductory programming classrooms (e.g., [5, 6, 8]). Additionally, many popular novice programming environments, such as Scratch [5], Snap*!* [6] and Alice [4], are widely used to build such open-ended projects in introductory programming classrooms, which lowers the barriers to build complex projects, and help students to build visual, interactive artifacts that they could interact with, increasing their interest in computing [9].

Planning, including generating, choosing ideas, and designing a step-by-step solution, is the first step during open-ended project-making [16]. An explicit planning process, where students write down the key features and functionalities in their programs, has been shown to help students build more complex and ambitious projects [7]. However, prior work has found that novices frequently experience difficulties to explore and explain their ideas when they primarily use text to write down their ideas during planning, and prefer drawing their ideas on a slide than listing their ideas in texts [12], showing that text-focused planning can feel less inspiring and expressive for novice programmers.

In this work, we present Idea Builder, a system that supports novices to design and plan an open-ended project through storyboarding. The Idea Builder system includes two components: 1) the "storyboarding" component, which helps novices to use drawings and images to indicate motions, interactions, and mechanics; and 2) the "connection" component, which connects code to plans by showing students animations of code examples (called "planning examples") when selecting assets, and connects plans to code by automatically converts a student's storyboard to starter code.

Our evaluations of students' experience include two parts. 1) We qualitatively evaluated students' experience using only the "storyboarding" component during a high school summer internship program, where students used the system with only the storyboarding feature, to plan for an open-ended programming project, and we conducted interviews with students afterward to understand their experience. 2) We also evaluated students' experience using the "connection" component by conducting a within-subject comparison study to investigate whether giving students example assets with planning examples increases students' example use rate. Our evaluations found that Idea Builder helped students self-perceived as feeling creative and expressive during planning; planning examples encouraged students to plan a mechanic and use a code example more effectively; converting storyboards to starter code was generally perceived as useful and time-saving to students.

This work has the following contributions: 1) The introduction of Idea Builder, a digital storyboarding system that connects planning to students' Snap*!* programming experience. 2) A qualitative student study, which shows that students ... 3) A quantitative & qualitative study, which shows that assets with gif examples inspired students to plan and use these features during open-ended programming.

2 RELATED WORK

We first discuss related works on planning and its impact on novices' programming experience, as well as challenges novices encounter during planning. We next discuss storyboarding and examples for open-ended programming, which inspired our design of Idea Builder.

Planning. Many prior works show that an explicit planning process, where students write down the step-by-step plan for their programming project, is helpful towards students' programming outcomes [7, 11]. For example, **(author?)** investigated the impact

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

of using planning sheets on 155 middle school students from an introductory Scratch programming course, and found that the students who used planning sheets to write down their plans prior to programming completed significantly more objectives, and built more complex programming projects, compared to those who did not have access to planning sheets [7]. **(author?)** conducted an analysis on undergraduate students' open-ended programming, by comparing their planning data extracted from a text-based planning application (called "PlanIT") [14], and compared students' planning data with features they have completed in their projects. The results show that the number of features students listed in their plans is significantly correlated with the number of features achieved in their final programming artifact, showing that students who build more complex plans also tends to produce more ambitious programming projects.

However, novice students also encounter many challenges when planning for open-ended programming projects [17, 19], as they frequently lack of skills to express a programming feature they want to build, and may encounter challenges to explore new ideas [19]. For example, **(author?)** conducted a three-year study on a enrichment program that guides African-American middle school students to iteratively design (using a design document), and build complex, open-ended projects for social change, and found that students experienced struggles in expressing game ideas, and articulating user actions and related game functionalities [17], which may be due to the limits in traditional, text-based planning.

Storyboarding. Prior works on personal narrative and storytelling show the potential for storyboarding to help learners become more expressive and creative when expressing their ideas [15]. For example, the theory of embodied cognition [21] suggests that drawing and imagery connect human thinking with their real-world experience [13], and leads to richer, more creative storytelling experience [3, 15, 18]. **(author?)** conducted a qualitative study on how 17 6-13-year-old children make use of The Telling Board, a storyboarding tool designed for children to use drawings and images to build their own stories, and found that children were highly engaged in the storyboarding process, and felt creative and proud of their artifacts [15]. One study investigated the use of storyboard-based planning in computing classrooms – **(author?)** conducted a 2-day qualitative case study to investigate 5 pairs of students' planning experience during open-ended project-making, comparing text-based planning to storyboard-based planning using Google Slides. The students expressed feeling more creative when using storyboards, and preferred using storyboards to using texts to plan their projects [12]. This work shows the potential of using storyboards to support students' planning experience during open-ended project-making.

Examples for Open-Ended Programming. Novice programmers encounter many challenges when designing and building an open-ended programming project. For example, **(author?)** conducted an interview and log data analysis on undergraduate students working on an open-ended programming project, and found that they encounter barriers to exploring and deciding features they want to build in their programs, and experience difficulties to explain a feature they want to build [19]. However, prior work on examples to support open-ended programming focus only on the "programming" experience. For example, Example Guru is a tool that provides on-demand examples based on students' code states when building open-ended projects in Looking Glass [10]. A study conducted by **(author?)** found that students preferred examples over static documentation, and build projects with more advanced APIs after integrating them from code examples. This shows the potential of using examples to support *programming*. However, while some work discussed students' need for exploration during the *planning* phase of open-ended programming, and the potential for using examples to support idea exploration during planning [12], no prior work evaluated the impact of having access to planning examples (i.e., examples provided during project planning), on students' planning and programming experience.

3 THE IDEA BUILDER SYSTEM

3.1 An Example Usage Scenario

Ada is making a plan for an open-ended project they need to build in Snap*!*. We explain how Ada may use the "storyboarding" and "connecting" component in Idea Builder to explore and express their ideas.

3.1.1 The "Storyboarding" Component: The storyboarding component mainly features

Step 1: Define Actors.

They can start making plans by defining the actors and backgrounds in their game. Idea Builder include an library of assets for students to search and integrate into their list of assets. An actor maps to a sprite in Snap*!*, and can include multiple costumes. Ada can add costumes by searching for a different image, or doing image editing (e.g., by flipping it or adding another icons on it) inside Idea Builder. Later during storyboarding, Ada can click on an actor or a background to copy it to her storyboards. In addition to actors and backdrops, Ada may also use events in Idea Builder, indicating what may triggers the actions. There are 6 pre-defined events, which students can directly click to move to their storyboards: when green flag clicked; when key pressed; when mouse clicked; when mouse hovered; when mouse released; and when time goes by.

Good design thing: 1) allowing students to search, saving their time.

Step 2: Make Storyboards. Ada can use the actors and backgrounds she has defined to make storyboards. Ada can use a storyboard to express one specific feature they are interested in building, such as a fruit drops to the ground, or a goat jumps over the platforms. Within each storyboard, students can make several frames to indicate how a mechanic works in action. For example, Figure **??** shows that after an user clicks on the "play" button, the flowers will start to fall from the trees. Ada can draw the flower's falling motion by clicking on the "Add motion" button, and drawing a trajectory of the flower falling from the tree. While making the storyboards, Ada can also add descriptions of the storyboard on the right section named "Storyboard Notes".

Good design thing: support both drawing and writing the motion thing connects to programming experience.

3.1.2 The "Connecting" Component: The "connecting" component of Idea Builder includes two parts: 1) It connects programs to plans by using gifs to generate example animations of a potential

feature for an actor, called "planning examples" (Figure **??**). 2) It connects plans to programs by synthesizing students' storyboards to a starter code, which they can directly import into Snap*!*. We discuss the two parts below. **Planning Examples.** The goal of including planning examples is to use animations to help students explore what is possible in Snap*!*, and link those animations to actors that they could directly use in their plans. Shown in Figure **??**, when selecting actors, Ada can click on the "âĺĄ" sign to explore a gallery of example actors. Under each example actors, they can view an gif animation of the actor, which shows a potential usage of the actor. For example, a star actor can sometimes be used as many clones of the star sprite changing brightness repetitively over time. **Synthesizing Starter Code.** To help students transition easily from planning to programming, a student may click on the "download code" button, which they could use to download an xml file, as the starter code of their Snap*!* program. The code synthesizing feature use human-authored synthesizing rules, which interprets properties such as actors' positions, clones, movements, events, and conversations, and generate code that can reproduce the longest storyboard in a student's plan. As shown in Figure **??**-b, the generated example is sometimes of less quality than a curated, human-authored example. For example, instead of inferring that the trees are at random positions, the generated example used repeated cloning statements to locate trees to the precise locations. The goal of using these auto-generated starter code is to help students get started with all actors and costumes already loaded from their plans, and some starter code that brings actors to the correct size and locations, and therefore to save students' time.

3.2 Design Goals

The design of Idea Builder follows the 7 design principles proposed by Green [1] on building extensible, inter-connected planning systems that supports introductory programmers. The "storyboading" component of Idea Builder follows the first four principles, which aim for students to express a variety of ideas and designs in the system, to iterate over different ideas, and to build designs that align with their programming tasks [1]. The "connecting" component of Idea Builder follows the last three principles, which aims to help students generate different ideas to directly connect their plans to their programming task.

1) *Tailored plans*: the system should constrain the design experience to the affordances of the programming environment. Idea Builder helps to constrain the design experience to the affordances of Snap*!* by mapping multiple functionalities of design to the functionalities of programming. For example, the actors and states in Idea Builder maps to the sprites and costumes in Snap*!*; the events in Idea Builder maps to a variety of control/event blocks in Snap*!*; the "add motion" functionality in Idea Builder maps to movements in Snap*!*; and the "add speech bubble" functionality in Idea Builder maps to the "say" blocks in Snap*!*. 2) *Refactoring support*: **(author?)** proposed that design systems should allow rapid prototyping and iterating on plans [1]. Idea Builder allows easy iteration and refactoring of designs by allowing students to quickly copy, modify, and delete different states, actors, backdrops and frames. 3) *Set of relationships*: allowing the expression of a wide range of features/mechanics/object relationships. To allow students to express relationships, such as how actors interact, how different scenes in their project transition to each other, Idea Builder makes use of the Project-Storyboard-Frame hierarchy, meaning, inside each project, there may be multiple storyboards, each represents a single mechanic. Each storyboards makes use of multiple frames, each indicates a scene at a specific time frame. 4) *Extensibility*: allowing users to extend and customize their own designs. To allow students to extend and customize their own designs, Idea Builder not only allows students to choose their own assets (i.e., by uploading and searching), but also makes use of multiple features, such as motions and actor states, for students to creatively express ideas they want their programs to achieve. 5) *Reverse engineering*: being able to generate elements for planning from code. The planning examples translates potential features students may build into animations, that students can view, and select the relevant actor into their programs. The planning examples help students to explore and foresee different options in Snap*!*, to explore different ideas and understand what may be possible to achieve. 6) *Code generation*: being able to generate code from plan. The code generation feature helps students to convert their plans in Idea Builder to a starter code, including actors, sprites' starter positions, and basic interactions. This allows students to quickly transition from planning to programming, without spending repetitive efforts or feel stuck at getting started. 7) *Run-time interactions*: the system should allow users to interact with the generated code at run time. As students may download the starter code at any time during planning, and experiment with it by importing it into Snap*!*, the system allows the students to test, modify and interact with the starter code both during and after planning.

4 STUDENT STUDIES

4.1 Study 1: Understanding students' storyboarding experience

We conducted the first qualitative to understand how students perceive their storyboarding experience. The study took place in a high school internship program. The participants were high school students working on building xxx for teachers. There are xxx participants, xxx (demographics). During the study, xxx (study procedure). After the study, we conducted interviews with the students, asking ... .

We next used thematic analysis to analyze the transcribed interview data ...

4.2 Study 2: Investigating the impact of planning examples and code generation on students' planning and programming experience.

The goal of the second study is to investigate the impact of the "connecting" component on students' planning and programming experience. Specifically, we investigate 1) whether showing students example usage of an actor helps students become more interested in using the actor in their programs; and 2) how students self-perceived their experience using the starter code generated from idea builder.

- population: We recruited xx participants from local high schools, for a full-day programming workshop. The workshop is hosted repetitively for 3 Saturdays during 3 consecutive weekends. Students can choose any day to attend the workshop. There are xx females and xx males. There are xxx white, ... Participants reported different levels of prior CS experience: xxx, xxx, xxx.

- procedure: Each of the Saturdays follow the same procedure. Students first complete a 2-hour Snap*!* introduction, where they follow the instructor to build a game in Snap*!*, while learning basic programming concepts and Snap*!* APIs: clones,

- data collected: - analysis method:

REFERENCES

[1] C. Alphonce and B. Martin. Green: a pedagogically customizable round-tripping uml class diagram eclipse plug-in. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 115–119, 2005.

[2] P. C. Blumenfeld, E. Soloway, R. W. Marx, J. S. Krajcik, M. Guzdial, and A. Palincsar. Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational psychologist*, 26(3-4):369–398, 1991.

[3] S. L. Chu and F. Quek. The effects of visual contextual structures on children's imagination in story authoring interfaces. In *Proceedings of the 2014 conference on Interaction design and children*, pages 329–332, 2014.

[4] S. Cooper, W. Dann, and R. Pausch. Alice: a 3-d tool for introductory programming concepts. In *Journal of Computing Sciences in Colleges*, volume 15, pages 107–116. Consortium for Computing Sciences in Colleges, 2000.

[5] D. Franklin, D. Weintrop, J. Palmer, M. Coenraad, M. Cobian, K. Beck, A. Rasmussen, S. Krause, M. White, M. Anaya, et al. Scratch encore: The design and pilot of a culturally-relevant intermediate scratch curriculum. In *Proceedings of the 51st ACM technical symposium on computer science education*, pages 794–800, 2020.

[6] D. Garcia, B. Harvey, and T. Barnes. The beauty and joy of computing. *ACM Inroads*, 6(4):71–79, 2015.

[7] D. Gonzalez-Maldonado, A. Pugnali, J. Tsan, D. Eatinger, D. Franklin, and D. Weintrop. Investigating the use of planning sheets in young learnersâĂŹ open-ended scratch projects. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 247–263, 2022.

[8] S. Grover, S. Basu, and P. Schank. What we can learn about student learning from open-ended programming projects in middle school computer science. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, SIGCSE '18, page 999âĂŞ1004, New York, NY, USA, 2018. Association for Computing Machinery.

[9] M. Guzdial. Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, 8(6):1–165, 2015.

[10] M. Ichinco, W. Y. Hnin, and C. L. Kelleher. Suggesting api usage to novice programmers with the example guru. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1105–1117, 2017.

[11] W. Jin, A. Corbett, W. Lloyd, L. Baumstark, and C. Rolka. Evaluation of guided-planning and assisted-coding with task relevant dynamic hinting. In *International Conference on Intelligent Tutoring Systems*, pages 318–328. Springer, 2014.

[12] A. Limke, A. Milliken, V. Cateté, I. Gransbury, A. Isvik, T. Price, C. Martens, and T. Barnes. Case studies on the use of storyboarding by novice programmers. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*, pages 318–324, 2022.

[13] D. McNeill. *Gesture and thought*. University of Chicago press, 2008.

[14] A. Milliken, W. Wang, V. Cateté, S. Martin, N. Gomes, Y. Dong, R. Harred, A. Isvik, T. Barnes, T. Price, and C. Martens. Planit! a new integrated tool to help novices design for open-ended projects. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, SIGCSE '21, page 232âĂŞ238, New York, NY, USA, 2021. Association for Computing Machinery.

[15] D. Powell, P. Gyory, R. Roque, and A. Bruns. The telling board: An interactive storyboarding tool for children. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*, IDC '18, page 575âĂŞ580, New York, NY, USA, 2018. Association for Computing Machinery.

[16] A. Saad and S. Zainudin. A review of project-based learning (pbl) and computational thinking (ct) in teaching and learning. *Learning and Motivation*, 78:101802, 2022.

[17] J. O. Thomas, Y. Rankin, R. Minor, and L. Sun. Exploring the difficulties african-american middle school girls face enacting computational algorithmic thinking over three years while designing games for social change. *Computer Supported Cooperative Work (CSCW)*, 26(4-6):389–421, 2017.

[18] S. Valguarnera. Eppics: Enhanced personalised picture stories. In *Interaction Design and Children*, pages 620–623, 2021.

[19] W. Wang, A. Kwatra, J. Skripchuk, N. Gomes, A. Milliken, C. Martens, T. Barnes, and T. Price. Novices' learning barriers when using code examples in open-ended programming. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE '21, pages 394–400, New York, NY, USA, 2021. Association for Computing Machinery.

[20] W. Wang, A. Le Meur, M. Bobbadi, B. Akram, T. Barnes, C. Martens, and T. Price. Exploring design choices to support novices' example use during creative open-ended programming. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, pages 619–625, 2022.

[21] M. Wilson. Six views of embodied cognition. *Psychonomic bulletin & review*, 9(4):625–636, 2002.