Idea Builder: Motivating Idea Generation and Planning for Open-Ended Programming Projects through Storyboarding

Wengran Wang*
North Carolina State University
Raleigh, USA

Ally Limke
North Carolina State University
Raleigh, USA

Mahesh Bobbadi
North Carolina State University
Raleigh, USA

Amy Isvik
North Carolina State University
Raleigh, USA

Veronica Catété
North Carolina State University
Raleigh, USA

Tiffany Barnes
North Carolina State University
Raleigh, USA

Thomas W. Price
North Carolina State University
Raleigh, USA

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 in exploring and creatively expressing ideas during planning. We present Idea Builder, a storyboarding-based planning system to help novices visually express their ideas. Idea Builder includes three features: 1) storyboards to help students express a variety of ideas that map easily to programming code, 2) animated example mechanics with example actors to help students explore the space of possible ideas supported by the programming environments, and 3) synthesized starter code to help students easily transition from planning to programming. Through two studies with high school coding workshops, we found that students self-reported as feeling creative and feeling easy to communicate ideas; having access to animated example mechanics of an actor help students to build those actors in their plans and projects; and that most students perceived the synthesized starter code from Idea Builder as helpful and time-saving.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; **User studies**; **Empirical studies in HCI**.

KEYWORDS

open-ended programming, planning, block-based programming, novice programming

*Currently at Apple Inc.

ACM Reference Format:
Wengran Wang, Ally Limke, Mahesh Bobbadi, Amy Isvik, Veronica Catété, Tiffany Barnes, and Thomas W. Price. 2024. Idea Builder: Motivating Idea Generation and Planning for Open-Ended Programming Projects through Storyboarding. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630872

1 INTRODUCTION

In computing classrooms, building an open-ended programming project engages students in designing and implementing an idea of their own choice [11]. It motivates students by placing learning in a real-world context and encourages students to take ownership of their work by generating ideas, designing solutions, and implementing their plans [2]. As a result, open-ended programming projects are widely-used in introductory programming classes (e.g., [7, 8, 11]). Many popular novice programming environments, such as Scratch [7], Snap*!* [8] and Alice [5], are designed to build such open-ended projects, which lowers the barriers to building complex projects, and help students to build visual, interactive artifacts, increasing their interest in computing [12].

Planning, involving idea generation and step-by-step design, is a crucial first step in creating open-ended projects [23]. Explicitly planning the key functionalities of their programs helps students develop more complex and ambitious projects [10]. Prior work shows a correlation between high-quality plans and high-quality projects [29]. However, beginners often struggle to create detailed plans, express their ideas clearly, and link their plans to later projects [25]. Novices find it challenging to explore and explain their ideas using traditional text-based methods for open-ended project planning, preferring visual communication through storyboards instead [19]. This highlights the necessity for teaching methods and tools that assist students in planning open-ended projects, particularly those that use visual approaches.

In this work, we present Idea Builder, a system that helps novices to design and plan an open-ended project through storyboarding. Idea Builder includes 3 key features. 1) It uses storyboards to help students express a variety of ideas that map easily to programming code. 2) It provides animated example mechanics, to help students explore the space of possible ideas supported by the programming environments. And 3) it generates an executable starter project after storyboarding to help students easily transition from planning to programming.

We deployed Idea Builder in two small-scale pilot evaluations: in a high school summer internship program ($n = 7$) and summer camp ($n = 20$). We present students' perceptions of each of the 3 core features of Idea Builder (storyboards, examples, starter projects), exploring how they helped students to plan and implement open-ended projects. We reflect on our experience as designers and educators using the tool.

The Idea Builder system offers the following novel contributions: 1) We propose a storyboard-based planning experience, which students self-reported to have helped them express and communicate ideas that connect to programming. 2) We provide animated examples during the planning process and provide evidence of its impact on students' planning and programming outcomes. 3) We apply program synthesis to the domain of block-based programming environments, and present evidence that the synthesized code was perceived by the students as useful and time-saving.

2 RELATED WORK

Idea Builder is designed for students who are planning an open-ended programming project in Snap*!*, a block-based programming environment, which enables novice programmers to build visual, interactive programming artifacts, such as games, apps, and stories [13]. The Snap*!* programming environment contains blocks for users to program instructions for "sprites", which are actors on the Snap*!* stage[1]. Snap*!* is commonly used to build open-ended projects in introductory programming classrooms [8], as it lowers the barriers for students to build complex games and apps, increasing their interests in computer science [12].

Our design and development of Idea Builder are motivated by related work on planning and its impact on novices' programming experience. Specifically, the design goals of Idea Builder are informed by 3 fields of research: storyboarding, supporting exploration for open-ended project-making, and systems for planning (closed-ended) problem solutions.

Planning. Soloway et al. describes project planning as a process where students 1) identify the key goals of a program they will write; 2) design methods to achieve those goals; and 3) specify objects or data items that compose the solution [24]. Prior works show that an explicit planning process, where students write down a step-by-step plan for their programming project, is beneficial for programming outcomes [10, 15]. For example, Gonzalez-Maldonado et al. investigated the impact of using planning sheets with 155 middle school students from an introductory Scratch programming course, and found that the students who used planning sheets completed significantly more objectives, and built more complex programming projects, compared to those who did not have access to planning sheets [10]. Wang et al. conducted an analysis on undergraduate students' open-ended programming, by comparing their planning data extracted from a text-based planning application, 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 tend to produce more ambitious programming projects.

However, novice students also encounter many challenges when planning for open-ended programming projects [25, 28], as they frequently lack the skills to express programming features they want to build, and may encounter challenges when exploring new ideas [28]. Thomas et al. conducted a three-year study on an 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. The results showed that students experienced struggles in expressing game ideas, user actions, and related game functionalities [25]. Some of these challenges may be due to the limits of traditional, text-based planning, and they suggest the need for better support planning support.

Storyboarding. Prior works on personal narrative and storytelling show the potential for storyboarding to help learners become more expressive and creative when creating and communicating their ideas [22]. The theory of embodied cognition [30] suggests that drawing and imagery connect human thinking with real-world experience [21], and lead to richer, more creative storytelling experiences [4, 22, 27]. Powell et al. conducted a qualitative study on how 17 6-13 year-old children made use of The Telling Board, a storyboarding tool in which images are drawn to build stories, and found that children were highly engaged in the process, and felt creative and proud of their artifacts [22]. As a prototyping method, storyboarding is also commonly used by UI designers to prototype system interfaces [26], for K-12 students to design video games [6], and for children to prototype interactive systems [9].

One study investigated the use of storyboard-based planning in computing classrooms – Limke et al. 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 text when planning their projects [19]. This work shows the potential of using storyboards to support students during open-ended project-making. However, Google Slides is not designed explicitly for planning programming projects, includes unrelated features that may distract students, and lacks any code-specific scaffolding.

Supporting exploration for open-ended project-making. Kery and Myers defined "exploratory programming" as an open-ended programming process, where programmers 1) make use of external resources (e.g., the web) to explore and generate ideas; and 2) use prototyping and experimentation to iterate through different ideas [16]. However, novice programmers encounter many challenges when finding and deciding ideas during open-ended programming [18, 24]. To help students explore ideas, some prior works provide a gallery of examples they could request on-demand during programming [28, 29]. However, as shown by Wang et al. in an interview study on novices' example use, many students do not feel motivated to use examples to explore different ideas once they start programming. Without leveraging support such as examples to understand what is possible to build in their projects, students can instead plan and build simple projects that they are familiar with,

[1]We will use "sprites" and "actors" interchangeably. Actors in Idea Builder map directly to sprites in Snap*!*.

without exploring and learning new programming concepts and APIs [18, 19]. These findings show students' needs to explore ideas during the *planning* phase of open-ended programming, which examples can potentially support. However, no prior work has evaluated the impact of having access to examples during planning, on students' planning and programming outcomes. Additionally, many systems exist to support closed-ended planning, e.g. through pseudocode (e.g., [18]), flowcharts (e.g., [14]), and UML diagrams (e.g., [1]), but these systems are less relevant for creative, open-ended novice projects.

3 THE IDEA BUILDER SYSTEM

3.1 Design Goals

We built Idea Builder with the following design goals:

- **DG1**: Help students creatively express a variety of ideas that map easily to programming code.
- **DG2**: Help students explore the space of possible ideas that are well-supported by their target programming environment.
- **DG3**: Help students easily transition from planning to programming.

We built Idea Builder[2] in a two-stage process. The first stage focused on the "storyboarding" feature for Idea Builder, which aimed to achieve DG1. The second stage focused on the "example" & "translation" feature, which aimed to achieve DG2 and DG3.

Feature 1: The "Storyboarding" Feature. This feature aims to allow students to quickly prototype ideas that are possible in Snap*!*, such as mechanics, movements, and interactions (DG1).

Step 1: Define Actors. To bridge between visual storytelling and planning for an interactive programming project, Idea Builder prompts users to define actors. A student can search and click on "use" to copy actors and backgrounds in a gallery of images (Figure 1 – 2), or upload images on their own. An actor maps to a sprite in Snap*!* (similar to an Object in object-oriented programming).

Step 2: Make Storyboards. Next, a student can build a storyboard to express a feature they are interested in building, such as "a fruit drops to the ground," or "a goat jumps over the platforms." Idea Builder allows making a wide variety of storyboards that will translate easily into Snap*!* programs through 4 key features:

1. Making frames. By clicking on an actor or a background users can copy it to a frame of their storyboard. Within each storyboard, several frames can be created to indicate how a mechanic works in action. For example, Figure 1 shows that after a user clicks on the green flag, the sticks will start to fall from the tree. This helps the student plan and communicate movement, interaction, and animation, which are central parts of interactive Snap programs.

2. Define events. Games and animations in Snap*!* commonly include user interactions, which consist of user-generated events. To express those events, Idea Builder provides an "events" category. There are 6 pre-defined events, which students can directly click to move to their storyboards (e.g., when the green flag is clicked; when the key is pressed; and when the mouse is clicked).

3. Draw motions. Users can draw a trajectory to describe actor movement. As an example, to describe actor movements, a student can add the sticks' falling trajectory by clicking on the "Add motion" button, and drawing a trajectory of the sticks falling from the tree (Figure 1).

4. Write textual descriptions. As prior work shows that students preferred to combine text and images to express their ideas [19], Idea Builder also allows students to add descriptions of each storyboard on the right section named "Storyboard Notes".

Feature 2: The "Example" Feature. The "example" feature aims to achieve DG2 – to help students explore what is possible in Snap*!*, so that they can plan interesting and ambitious features. To do that, Idea Builder links example animations to actors that they could directly apply to their plans. As shown in Figure 1, when selecting actors, one can click on the "⊕" sign to explore a gallery of example actors. The example actor gallery is sorted into four categories: Surrounding Actors, Main Actors, Tools/Accessories, and Buttons. Under each example actor, they can view an animation of an actor (called "example mechanic"), which shows the potential usage of the actor. For example, the top-right of Figure 1 shows a potential usage of three actors: snow, fish, and star. The animated example mechanic of the "star" actor is to create a moving background with many small, twinkling stars. Specifically, the example mechanics we presented were designed to 1) be easy to implement with the blocks available in Snap*!*; 2) encourage students to use the full variety of APIs that they had learned, including more advanced features like cloning.

Feature 3: The "Translation" Feature.

Moving from planning to programming (DG3), a student can click the "download code" button ("<>" on Figure 1, Reference 3). This button enables downloading an XML file as the starter code for their Snap*!* program. The provided code creates sprites corresponding to each actor, with appropriate costumes and starting positions. It also includes basic code for moving sprites as per events and movements from the storyboards. The goal of the starter code is not to do the students' programming for them, but rather to allow them to dive right into programming more interesting mechanics, rather than trivial setup code.

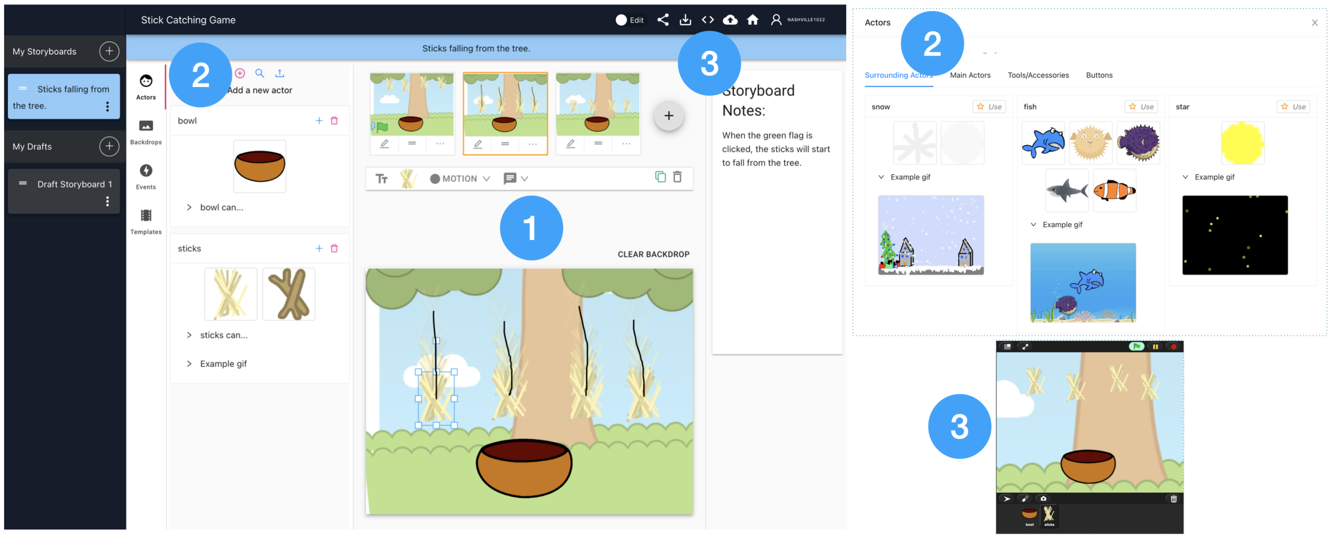
To translate a plan into students' starter code, Idea Builder uses human-authored synthesizing rules, which interpret properties such as actors' positions, clones, movements, events, and conversations, and generate code that can reproduce the storyboard with the most frames in a student's storyboards[3].

To generate starter code, Idea Builder calculates the changes between frames and uses them to infer actor movements, appearance alterations, and dialogues, translating these into Snap*!* code. It also utilizes the "events"[4] users indicated in the storyboard, translating them into corresponding "hat blocks" in Snap*!*. This process involves three steps: First, generate code that positions each actor as defined in the storyboard, with the same size and appearance as in the first frame of the storyboard. This is done using blocks such as "when green flag clicked", followed by "set size to", and "go to x:

[2]Idea Builder is an open-source software that can be downloaded at github.com/emmableu/Idea-Builder.

[3]As it is challenging to infer transition rules between multiple storyboards, Idea Builder only selects the longest storyboard, and try to reproduce the behavior only on the longest storyboard.

[4]Events are what triggers the code to run, such as "when green flag clicked", or "when space key pressed".



When the green flag is clicked, the sticks will start to fall from the tree.

Figure 1: The Idea Builder interface. Students can make storyboards that express their game mechanics and actor interactions (1). They can also click on the "⊕" button to view animated mechanics, for example, actors (2). Finally, they can click on the "< >" button to download an executable starter code for Snap*!*, synthesized from their storyboards (3).

_ y: _". If the user specifies multiple actor clones, the code generates clones at the indicated positions. For instance, the generated program in Figure 1 starts by placing 4 stick clones at the top of the stage, with space between them. Second, generate code for the actor to respond to specified events. For instance, if the user indicates the actor's movement with the right arrow key, generate code using the "when right arrow key pressed" block. Third, create movements that imitate the motions drawn by students. For example, in Figure 1, the generated program uses a "repeat 100" and "change y by -3" to simulate sticks gradually falling towards the ground.

As a limitation, the generated examples can be of lower quality compared to curated, human-authored examples. For instance, instead of using a loop structure to create the four sticks (seen in Figure 1 bottom-right), the generated example employs four separate statements for precise placement.

4 EVALUATIONS

We describe two small-scale deployments of Idea Builder: Deployment 1 investigates students' self-perceived experiences using the "storyboarding" feature in Idea Builder, while Deployment 2 includes all 3 features in a high school coding workshop, and investigates the usefulness of the "example" and "translation" features.

4.1 Deployment 1

The primary goals of Deployment 1 were to 1) assess whether Idea Builder's storyboarding is an effective planning medium for open-ended projects, as suggested by prior work [19] and; 2) understand *why* this might be the case (i.e., affordances of IdeaBuilder and storyboarding in general).

Participants and Procedure: We recruited participants from a high school internship program, where students learn Snap*!*, and build computing-infused programs for instructors. The study took place over 2 afternoons, during the COVID pandemic, and was held over Zoom. Each afternoon, students spent an hour using Idea Builder to plan for a new open-ended project and then spent one to two hours implementing their plans in Snap*!*. Additionally, on the afternoon of Day 2, between storyboarding and planning, students took 20 minutes to share their storyboards with their peers in breakout rooms, each room including 5 to 7 students. At the end of each day, we reached out to consented students (5 on Day 1 and 4 on Day 2) to invite them to a 10-minute 1-1 interview, where we asked students about their experience planning their projects and communicating ideas. Over the two days, a total of 7 students attended these interviews. 8 consented students[5] indicated their demographics in a pre-survey. The students were all Females, with 3 self-reported ethnicities as Biracial/Multiracial, 2 Asian, 2 White, and 1 Black. Students self-reported as having some (6) or very strong (2) programming skills.

Analysis: We next employed the thematic analysis method [3] to analyze the transcribed interview data. Two researchers first independently open-coded all 7 interview documents to collect all potential codes that described students' experience and perceptions with their planning, programming, and communication. The two researchers then merged their codes and sorted them into high-level themes by grouping codes that explain similar phenomena. Next, they organized these themes into a codebook and generated definitions of them. One researcher then used the codebook to review the original interview documents to confirm the themes and the definitions. Our final themes revealed two key activities students engage in when using Idea Builder: generating and communicating ideas.

[5]We did not retain students' names during interviews, so we report all consented students' demographics, instead of the 7 students who attended interviews.

Results (1): Generating ideas. When asked about how they came up with their ideas, some students mentioned that having access to a gallery of actors helped them generate ideas: "*We saw a dinosaur in there. Then we saw a unicorn, which is what kind of inspired my little fantasy storyboard things.*" (P2), explained that "scroll[ing] through the icons ... gave me some inspiration". One student added that the interface helped them get started ("*get the ball rolling*"), but afterward was not as necessary, since "*once you have one idea you can start building off of each other*" (P6). P6 also suggested that to help students generate "*more ideas about events that could happen in the game*", it would be helpful to include "*a list of common actions that happen*" in Snap*!*.

Results (2): Communicating ideas. Students noted that Idea Builder's visualizations helped with explaining and communicating ideas: 4 students mentioned that being able to visualize ideas was helpful for communication and that if they "*had just ... verbally explained it, [then others] wouldn't have understood what I was thinking*" (P5). Students appreciated that Idea Builder broke down a design project into storyboards and frames. Some noted similarities between the frames in Idea Builder on the Snap*!* stage, as "*different slides [i.e., frames] ... mimic what you would see on Snap! in different stages*"(P1), which *makes me feel like I have part of a project that's actually there*"(P3), and was "*a lot simpler than actually having to go into Snap! and do it*"(P1). Some discussed that the process of planning is useful to "*collect [their] ideas and organize them better*"(P5), and that "*being able to use multiple storyboards for different mechanics really helped my group member envision what I wanted for my game*". In addition, two students mentioned that combining storyboards and textual descriptions were useful when explaining ideas, as "*having the game mechanics written down next to the storyboard*"(P6) provides a "*side by side*" view, and makes it easier to express a "*clear idea of ... what we can do in our game*" (P1).

4.2 Deployment 2

In Deployment 2, we investigated the usefulness of the example mechanics, as well as the generated starter code.

Participants: We recruited 20 high school students for a full-day programming workshop. There were 6 female and 14 male students, including 7 White, 5 Asian, 6 Black or African American, 1 Caribbean Islander, and 1 Hispanic or Latino student. Reported prior CS experience follows: 5 had no programming experience, 10 had completed a few programming tutorials before, 2 had taken one course in programming, 2 had taken two courses in programming, and 1 had taken more than two courses in programming.

Procedure: In the morning, students completed a 2-hour Snap*!* introduction, where they followed the instructor to build a game in Snap*!*, while learning basic programming concepts and Snap*!* APIs: loops, conditionals, concurrency, cloning, and broadcast. In the afternoon, students completed a 20-minute storyboarding activity in Idea Builder. To prompt students to use example mechanics, students were required to select a minimum number of actors to use in their storyboards from various categories. Students then used 10 minutes to communicate their ideas with a peer. After planning, students downloaded starter code synthesized from Idea Builder and used it to build an open-ended project in Snap*!*. After programming, students were asked: "Was having access to the starter XML helpful for you to get started with programming?", where they provide a binary response (helpful/unhelpful) with an explanation.

Within-subject conditions: To compare the usefulness of example mechanics, we designed within-subject conditions, where, in each student's Idea Builder interface, a random half of the example actors were attached to an animated example mechanic using that actor, while another random half did not have animated example mechanics.

Analysis: We collected students' submissions of Idea Builder and final projects along with their design survey responses. To examine the impact of example mechanics on students' preferences for actors in their plans and programs, we first defined the "storyboard use rate" for an example actor. This rate represents the percentage of students who saved and used the actor in at least one of their storyboards. For each example actor, such as the star, we calculated the "storyboard use rate" among two groups: students who had access to the animated example mechanics of the star actor (animation group) and those who did not (no-animation group[6]). Likewise, we examined the "program use rate" among students who did not view animations for a specific actor and those who had access to animations for that actor. The "program use rate" refers to the percentage of students who programmed at least one feature different from the starter code translated from Idea Builder. This signifies meaningful engagement with programming the respective actor.

Next, to understand how students used the generated starter code from Idea Builder, we compared each student's starter code with their final submissions, and inspected the types of blocks students were more likely to keep in the starter code and actually integrate into their final project code. To do that, we first defined the "kept rate" of a block type (e.g. the "move" block) as the number of blocks of that type from the starter code that were kept in students' final code, divided by the number of times that block type appeared in the starter code. We ranked the "kept rate" of all blocks that appeared more than once in all student programs to analyze what blocks from the starter code were more/less preferred by the students.

Finding 2: Students who saw an animation of an example actor are more likely to use it in their storyboards and significantly more likely to use it in their programs.

We found that students who saw an animation of an example actor (the animation group) are somewhat more willing to use the example actors, compared to the no-animation group. For an average example actor, its "storyboard use rate" among the animation group is 16.8% (min = 0, max = 45.5%, std = 0.148), which is higher than the example actors' storyboard use rate among the no-animation group (mean = 13.0%, min = 0, max = 44.4%, std = 0.119). A paired t-test shows that this difference was not significant, with a small effect size ($p = 0.12$, Cohen's $d = 0.28$).

We also found that the animation group had significantly higher program use rate than the no-animation group – for an average example actor, its "program use rate" among the animation group is 9.17% (min = 0, max = 44.4%, std = 0.13), which is more than twice of the program use rate among the no-animation group (mean = 4.44%, min = 0, max = 22.2%, std = 0.06). This difference is significant, with a

[6]Unlike the "experiment" and "control" groups in controlled between-group experiments, the "animation" and "no-animation" groups differ for each actor. This is because, for each student, a *random* half of the actors will have example mechanics.

medium effect size (p = 0.037, d = 0.45). This suggests that presenting example actors with animated mechanics helped students to become more willing to integrate this actor into their plans and projects.

This shows presenting students with an animated example mechanic can potentially guide students towards planning and building features that are afforded by the programming environment and applying more challenging programming structures and APIs into their code.

Finding 3: Many students perceived translating plans to code as helpful. Our rating data shows that most students (90%, 18/20) rated that having access to the starter code was helpful, explaining that it "*transport my backgrounds and sprites over immediately instead of trying to recreate them.*", and that it "*saved their time*". Two students, on the other hand, rated that having access to the starter code was not helpful, explaining that "*I did not use any of the code that it generated as I could make the code that I actually needed myself, and the code was just the positions of the sprites in the presentation which was only a rough draft of the game.*".

By comparing each student's translated starter code to their submitted programs, we found that **most sprites from the starter code, and half of the code from the starter code were kept in students' final submissions**. Among the 20 students, the generated starter code includes an average of 49 blocks (std = 42.22), from an average of 8.25 sprites (std = 2.02). On the other hand, the 20 students' final submitted code includes an average of 87.3 blocks (std = 48.38), from an average of 8.8 sprites (std = 3.38). We found that students on average kept 50.1% of the starter code (std = 0.282), showing that about half of the starter code was helpful. Inspecting the kept rate, we found that **the synthesized hat blocks, appearance blocks, and motion blocks were the most likely to be kept.** We found that the blocks that had above average kept rate are: "when green flag clicked" (100%), "hide" (92.9%), "show" (77.0%), "set size to" (66.7%), "say _ for 2 seconds" (60%), "change y by" (55%), and "go to x:_, y:_" (52.2%). We also found that the 2 blocks that had the lowest kept rate are "create a clone of myself" (0), and "when I start as a clone" (7.55%). As discussed in Section 3, the synthesized cloning code are sometimes not clean or high quality (e.g., using multiple cloning statement rather than a "repeat" loop). Therefore, students may be less likely to keep these blocks, due to their lack of high quality.

5 DISCUSSION & LIMITATIONS

In this work, we presented Idea Builder, a storyboarding-based planning system to help novices visually express their ideas while having access to planning examples, and synthesizes programming starter code after storyboarding. Through two studies in high school coding workshops, we found that students discussed feeling creative and feeling easy to communicate ideas when using Idea Builder; having access to animated example mechanics of an actor helped students to plan and program those actors; and the synthesized starter code from Idea Builder was perceived by many students as useful and time-saving. The usage of Idea Builder can be extended to all types of visual, interactive programming environments (e.g., Scratch [20]), to help students generate ideas and communicate their plans. It can be used not only in workshops, but also in computing classrooms for open-ended programming project planning and implementation. Our work has the following teaching and research implications:

1. Students enjoy flexible modes of idea expression. Observations by instructors revealed diverse styles in students' storyboard creation. For instance, some students consolidated all potential game scenes into a single storyboard instead of using one for each game mechanic. Others opted for a single core frame but used lengthy text to describe the game. Additionally, some students viewed storyboards as actor presentations, placing one actor with their properties in each frame. Notably, all students heavily utilized the "motion" feature, indicating a strong need to convey movements in their plans. This highlights the necessity for tool designers to allow flexible expression of students' ideas.

2. Introducing examples early helps. We found that students were more willing to use example mechanics when they are prompted during during planning. One potential explanation is that students often face challenges in open-ended programming due to not knowing the possibilities [17, 28]. Offering feasible ideas during planning through example mechanics could guide students in exploring new programming APIs and concepts. By providing a feasible idea during planning, example mechanics have the potential to help direct students toward practicing some new programming APIs and knowledge. For example, instructors may author example mechanics to prompt students to practice cloning, such as stars with blinking effects.

3. Translating storyboards to code is challenging. Our work revealed several challenges in synthesizing code based on storyboards. Idea Builder's users built storyboards for planning and ideation, without knowing that this data is later used to provide starter code, and therefore, may not provide structured data that can be easily interpreted into code. However, we found that students still kept half of the starter code (50.1%) and the majority of the sprites, and rated the synthesized starter code as useful. This shows that synthesizing starter code can help students set up their program. However, it is unclear whether this might be helpful or detrimental to students' programming practice.

Our study has limitations: we relied on qualitative data from a small number of participants, limiting the generalizability of our findings to other student groups. Moreover, our assessment only considered students' self-assessments, and did not consider instructors' experiences and the learning impact of accessing code examples. It's also essential to further explore why some students didn't enjoy using Idea Builder and the translation feature, and how Idea Builder might have failed to meet their goals and needs. This exploration can guide future improvements to the system. Future work should explore the learning impact and instructor experiences of similar systems with a larger population, as well as students' ideation process by inspecting *how* students storyboard and how they connect their ideas to programming.

6 ACKNOWLEDGEMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1917885.

REFERENCES

[1] Carl Alphonce and Blake 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] Phyllis C Blumenfeld, Elliot Soloway, Ronald W Marx, Joseph S Krajcik, Mark Guzdial, and Annemarie Palincsar. Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational psychologist*, 26(3-4):369–398, 1991.

[3] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[4] Sharon Lynn Chu and Francis 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.

[5] Stephen Cooper, Wanda Dann, and Randy 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.

[6] Michael A Evans, Brett D Jones, and Sehmuz Akalin. Using video game design to motivate students. *Afterschool Matters*, 26:18–26, 2017.

[7] Diana Franklin, David Weintrop, Jennifer Palmer, Merijke Coenraad, Melissa Cobian, Kristan Beck, Andrew Rasmussen, Sue Krause, Max White, Marco 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.

[8] Dan Garcia, Brian Harvey, and Tiffany Barnes. The beauty and joy of computing. *ACM Inroads*, 6(4):71–79, 2015.

[9] Terrell Glenn, Ananya Ipsita, Caleb Carithers, Kylie Peppler, and Karthik Ramani. Storymakar: Bringing stories to life with an augmented reality & physical prototyping toolkit for youth. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2020.

[10] David Gonzalez-Maldonado, Alex Pugnali, Jennifer Tsan, Donna Eatinger, Diana Franklin, and David Weintrop. Investigating the use of planning sheets in young learners' open-ended scratch projects. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, pages 247–263, 2022.

[11] Shuchi Grover, Satabdi Basu, and Patricia 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 999–1004, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351034. doi: 10.1145/3159450.3159522. URL https://doi.org/10.1145/3159450.3159522.

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

[13] Brian Harvey, Daniel D Garcia, Tiffany Barnes, Nathaniel Titterton, Daniel Armendariz, Luke Segars, Eugene Lemon, Sean Morris, and Josh Paley. Snap!(build your own blocks). In *Proceeding of the 44th ACM technical symposium on Computer science education*, pages 759–759, 2013.

[14] Danial Hooshyar, Rodina Binti Ahmad, Moslem Yousefi, Moein Fathi, Shi-Jinn Horng, and Heuiseok Lim. Sits: A solution-based intelligent tutoring system for students' acquisition of problem-solving skills in computer programming. *Innovations in Education and Teaching International*, 55(3):325–335, 2018.

[15] Wei Jin, Albert Corbett, Will Lloyd, Lewis Baumstark, and Christine 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.

[16] Mary Beth Kery and Brad A Myers. Exploring exploratory programming. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 25–29. IEEE, 2017.

[17] Amy J Ko, Brad A Myers, and Htet Htet Aung. Six learning barriers in end-user programming systems. In *2004 IEEE Symposium on Visual Languages-Human Centric Computing*, pages 199–206. IEEE, 2004.

[18] H Chad Lane and Kurt VanLehn. A dialogue-based tutoring system for beginning programming. In *FLAIRS Conference*, pages 449–454, 2004.

[19] Ally Limke, Alexandra Milliken, Veronica Cateté, Isabella Gransbury, Amy Isvik, Thomas Price, Chris Martens, and Tiffany 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.

[20] John Maloney, Mitchel Resnick, Natalie Rusk, Brian Silverman, and Evelyn Eastmond. The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, 10(4):1–15, 2010.

[21] David McNeill. *Gesture and thought*. University of Chicago press, 2008.

[22] Denise Powell, Peter Gyory, Ricarose Roque, and Annie 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 575–580, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450351522. doi: 10.1145/3202185.3210778. URL https://doi.org/10.1145/3202185.3210778.

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

[24] Elliot Soloway, James Spohrer, and David Littman. E unum pluribus: Generating alternative designs. *Teaching and Learning Computer Programming*, pages 137–152, 1988.

[25] Jakita O Thomas, Yolanda Rankin, Rachelle Minor, and Li 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.

[26] Khai N Truong, Gillian R Hayes, and Gregory D Abowd. Storyboarding: an empirical determination of best practices and effective guidelines. In *Proceedings of the 6th conference on Designing Interactive systems*, pages 12–21, 2006.

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

[28] Wengran Wang, Archit Kwatra, James Skripchuk, Neeloy Gomes, Alexandra Milliken, Chris Martens, Tiffany Barnes, and Thomas 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. ISBN 9781450382144. doi: 10.1145/3430665.3456370. URL https://doi.org/10.1145/3430665.3456370.

[29] Wengran Wang, Audrey Le Meur, Mahesh Bobbadi, Bita Akram, Tiffany Barnes, Chris Martens, and Thomas 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.

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