How families design and program games: a qualitative analysis of a 4-week online in-home study

Stefania Druga
University of Washington
Seattle, Washington, USA
st3f@uw.edu

Thomas Ball
Microsoft Research
Redmond, Washington, USA
tball@microsoft.com

Amy J. Ko
University of Washington
Seattle, Washington, USA
ajko@uw.edu

Figure 1: Overview of TileCode Platform [4] for game programming on web application and a handheld arcade device.

ABSTRACT

Prior work has broadly explored empowering children to learn to program by making video games. However, such work has rarely considered the role of families in this learning, leaving many open questions about how inter-generational collaborations might support and constrain learning. To investigate these opportunities, we conducted a family-based study of TileCode, a rule-based programming platform for video-game programming, and scaffolded a 4-week series of game programming activities with 19 children (9 to 14 years old) and 16 parents. Using a joint media engagement lens to analyze family knowledge and programming strategies, we found: 1) families demonstrated many dynamic collaboration patterns distinct from pair programming and other collaboration models, 2) parents played a unique role in scaffolding and guiding more complex designs and programming tasks, 3) families found it challenging to start their games from scratch but benefited greatly from having programming patterns for particular game behaviors. These findings suggest the need for game programming platforms to design around the unique kinds of collaboration in inter-generational domain-specific programming.

CCS CONCEPTS

• **Human-centered computing** → **Usability testing**; **User centered design**; **Interface design prototyping**; *Field studies*; *Sound-based input / output*; **Human computer interaction (HCI)**; User studies; • **Social and professional topics** → **Children**.

KEYWORDS

Domain Specific Languages, Children, Families, Computational literacy

ACM Reference Format:
Stefania Druga, Thomas Ball, and Amy J. Ko. 2022. How families design and program games: a qualitative analysis of a 4-week online in-home study. In *Interaction Design and Children (IDC '21), June 24–30, 2022, Lisbon, Portugal.* ACM, New York, NY, USA, 16 pages. https://doi.org/xxxxx

1 INTRODUCTION

Youth are spending more time playing computer, and video games, with 97% of children ages 12-17 playing computer, web, portable, or console games [55]. This engagement is encouraged by the growth and evolution of hardware platforms supporting gaming, an increase in the number and types of computer and video games, and significant growth in the participation in online gaming communities such as Minecraft, Roblox or Fortnite (141, 120 and 80 million monthly active players) [87].

Given the pervasive influence of computer and video games on youth culture [17, 49], many educators, designers and scholars have taken an interest in how some of the motivating aspects of video games might be harnessed to facilitate learning [86]. Previous studies have explored games as narrative and digital literacy practice spaces [35, 46, 88]; as creative, artistic expression and storytelling engines [1, 11, 34]; as ways for engaging with cultural and historical heritage [63] and developing mathematical inquiry [48] and scientific learning [16, 43]; or as avenues for critical learning [50] and civic engagement [80].

One promising approach has been to use video games as an avenue for teaching children how to program [49]. Platforms such as

Conference'17, July 2017, Washington, DC, USA
2022. ACM ISBN 978-1-4503-8452-0/21/06…$15.00
https://doi.org/xxxxx

Scratch [78], Snap [42], Cognimates [23] and MakeCode [3] provide children with procedural visual programming languages [14] that they can use to create games. Prior work underlines the benefits of designing domain-specific languages (DSLs) for game programming for non-experts, providing programming constructs that are closer to the natural language youth use to express game mechanics [70]. Moreover, recent studies show that programming different game genres (e.g., action, storytelling) can impact childrens' programming styles differently [93].

An early platform for building interactive simulations and games was Agentsheets [74]. In Agentsheets, the programmer defines the look and behavior of domain-specific building blocks called agents. Elementary school students with no programming background have used AgentSheets to create interactive simulations in a variety of disciplines, including computer science, environmental design, fine art, robotics, music, history, and biology [9, 75]. More recent initiatives propose DSLs for youth game programming; platforms such as BlockStudio use dedicated game icons, and programming by demonstration [6], CodeSpells uses functional programming and a 3D game environment to immerse novices in code through embodiment [30]. Finally, Gamechangineer uses natural language processing to generate game templates based on childrens' text descriptions [44].

Most prior work on game programming has explored learning contexts where youth create games alone or in groups in a formal or informal learning context. For example, studies of Alice in after-school programs have found that when girls engage in programming game-based stories, as opposed to just engaging in programming games, their short-term motivation to persist is greater [51]. Studies of middle school youth creating games with Scratch and wearables revealed how mixed media intersected with multiple dimensions of student interest [95]. Numerous studies have considered these phenomena in the context of pair programming amongst youth (e.g., [56] highlight the intricate role of social conflict in mediating successful learning).

While this prior work reveals the rich opportunities and tradeoffs of learning through making interactive games, there is a critical context that it has yet to explore: families. The inter-generational structure of families is critical to understand for numerous reasons. Prior work, for example, has found that parents, peers, and caregivers can play a dynamic role in youth learning, sometimes acting as facilitators or guides [8] and other times as learners, leading youth to see themselves as experts [24, 64]. Families can also be a bridge between formal learning at school and informal student-driven learning outside of school [66].

Prior work has also demonstrated that parental experience in technology fields plays a significant role in how they support their children's learning [22]. To address this, family-oriented programs using design-based activities, like Family Creative Learning (FCL) [81, 82], tried to support families lacking "preparatory privilege" [61] to get involved with their children's creative coding activities. Family-oriented coding programs are of particular importance as studies on family use and perception of coding showed that parents' main concern was that they would not be able to help their children due to their limited programming knowledge [100]. To better support parents to overcome their anxiety with programming [2], designers have explored text-free programming platforms, finding that families can successfully create together [5, 36]. Further understanding game programming in family contexts may reveal new opportunities for linking youth interests in games with interest-driven programming [18], family relationships [68], and formal computing education [7].

While prior work shows the feasibility of family game programming, it has not investigated the specific relational dynamics of parents and children programming together. Understanding the forms of collaboration that emerge in these settings is essential for designing tools, platforms, and scaffolding that support this collaboration. Joint use of media by parents and children has been broadly examined through the lens of *Joint Media Engagement* (JME) [92], finding many types of mediated interaction (e.g., cognitive, physical, technical, affective) [31]. However, few studies have specifically examined joint media engagement around programming and the particular challenges of program comprehension. One of the closest studies to these [5] focused primarily on the extent to which JME occurred and not how it occurred.

Therefore, in this work, we asked: *How do families jointly engage in rule-based 2D video-game programming?* To answer this question, we ran a 4-week online study with 19 children (9 to 14 years old) and 16 parents. In a series of individual family sessions, family members designed and programmed their games using a rule-based programming environment called TileCode, which enables video-game programming on low-cost devices. We observed family interactions during learning activities and then analyzed family computational and collaboration practices, their game artifacts, and their ability to decompose and compose game ideas and examples into programming patterns.

Our findings revealed that families engaged in various game design and programming strategies and struggled to start their games from Scratch or identify how to modify existing complex projects. However, they benefited the most from using a vocabulary of programming patterns that expressed specific game behaviors. For example, we found that TileCode was helpful in terms of connecting game elements to game actions but proved to be more challenging when families were trying to create rules that would affect multiple game elements at once. These insights can inform the design of future family learning activities around video-game programming and inform future DSL design efforts that provide multiple ways to support the composition and de-composition of game programming with dedicated vocabularies of programming patterns.

2 METHOD

To analyze how families understand, plan and program their games together, we structured our study across four online sessions that engaged families in using the TileCode platform. These sessions provided sufficient scaffolding to help families make meaningful progress while letting us observe a diversity of rich family interactions.

2.1 Participants

A total of 15 families, including 16 parents and 19 children, fully participated in our study. We primarily recruited families with at

least one child between the ages of 9-and 14 years old and encouraged participation by as many members of a family as possible. We recruited these families by posting an announcement on several family forums, social media groups, and family slack channels in North America. A total of 120 families applied to participate in the study, and we selected 19 families, trying to be as inclusive as possible along the following dimensions: family structure, ethnicity, geographical location, and socio-economic background. Of those 19 families, 15 attended all four sessions. The families unable to do so (due to extraordinary family circumstances or scheduling difficulties) were excluded from the final study analysis.

Parents' ages ranged from the low-thirties to mid-forties, with an average of 42 years. Fourteen of the parents were female, and 2 were male. Childrens' ages ranged from 7 to 14 years old, with an average of 10. Twelve of the children were female, and 7 were male. Of the 15 families, seven reported speaking only English, and three reported speaking English and Spanish (one also spoke Romanian). The remaining five families self-reported speaking (in aggregate) Thai, Indonesian, French, Mandarin, Cantonese, Hindi, Marathi, and English. Table 3 (in Results) shows a full list of family demographics and languages. Fourteen of the families were from seven states across the United States (CA, CO, IL, GA, NC, WA, and VA), and one was from Ontario, Canada. Most of the families self-reported familiarity with desktop/laptop computers, tablets, and smartphones, though one family reported familiarity only with tablets and smartphones (all families but one participated in our study using desktops/laptops, while the one family just mentioned using a tablet). Four families self-reported no programming experience, while seven reported using Scratch. Four other families mentioned Minecraft and Python.

2.2 Study Procedure

Our study had four sessions: 1) observe games (pre) and introduce the TileCode programming platform, 2) design and program a game, 3) create a game with patterns, and 4) observe games (post) and administer a quiz. All four sessions took place online using video teleconferencing software. We held four online sessions over four weeks with each family via video teleconferencing software and recorded the video, including any screen sharing. We designed each session to take place between 30 and 45 minutes.

Session 1: Observing (pre) initial game and TileCode walkthrough. In Session 1, we wanted to see how participants talked about video-game behavior (i.e., their vocabulary and how detailed a description they gave). We created three games using TileCode (replicas of Snake, Boulder, and Bejeweled) and recorded short videos of the gameplay before the sessions. We played each video in succession via screen sharing and prompted the participants to describe: how the games worked, game characters and events, and how they would change the games to make them more fun. We randomized the order in which we showed the videos. After this activity, we introduced the TileCode application to the families by screen sharing our web browser. We walked through the various parts of the application and modified a small example game using the rule editor (Figure 2). After this short overview, we asked participants to repeat our steps in their web browser. Each family picked one of the games from the videos and started changing it. (Prior work has demonstrated that modifying games offers several advantages over designing games from scratch [28] and is helpful for on-boarding novices into programming [26, 27] and can be used as a support for program understanding [37]).

Session 2: Designing and programming of games. Next, we introduced children to the TileCode platform (described below), so they could learn to program their video games. Then, we encouraged participants to use our "Game Design Activity" (described below) and design their games. After they finished designing their game on paper, the families started creating their game in TileCode. We also distributed a low-cost, battery-powered gaming handheld to each participating child so they could work with TileCode without needing access to the Internet or a computer with a web browser. Unfortunately, due to postal delays, only half of the participants received the handheld arcades in time for study sessions and got to choose if they wanted to create their game in the browser or on that device.

Session 3: Creating games with patterns. To support children in de- and re-composing games into game mechanics and corresponding programming patterns, we created a list of 10 game patterns (described in the next section). Then, we prompted participants to pick three patterns to use in a new game.

Session 4: Observing (post) other games and final quiz. In this final session, we repeated the game observation activity from the first session with three new game videos (replicas of Pacman, Space Invaders, and a side-scrolling video game), gathering families' descriptions of games. We also administered a short quiz to gauge families' understanding of TileCode programming.

2.3 Study Materials

This section describes the TileCode programming platform, the game videos, study activity printouts, and the quiz we used in our study to observe and evaluate how families engaged in video-game programming.

2.3.1 ***The TileCode Programming Platform***. TileCode is an existing system [4] designed to enable the creation of simple video games on low-cost gaming handheld devices. We summarize the system's essential features that are relevant to our study. Gaming handhelds have a small screen, a four-way direction pad (D-pad), and A/B buttons for selecting and deselecting. A secondary design goal was to allow the programming of these video games via a visual language of simple discrete rules with a minimum of text. Due to the limited input affordances, memory, and screen real estate of such gaming handhelds, both the TileCode user interface and the video games are restricted to a 2D grid of tiles, navigated using the D-pad. Figure 2(a) shows the "home screen" for creating a game, from which the user can select one of four main activities at the top (Map, Paint, Code, Play). The platform can run on various gaming handhelds, but it is also available as a web app.

A large part of video-game design is storytelling [51], which involves selecting and designing game characters and setting the initial scene which these characters occupy (also called a *map*). TileCode supports these activities in three ways. First, it provides a preset gallery of tile backgrounds and programmable sprites; the bottom of Figure 2(a) shows the four backgrounds and four sprites that the user has selected from the gallery. Second, selecting the

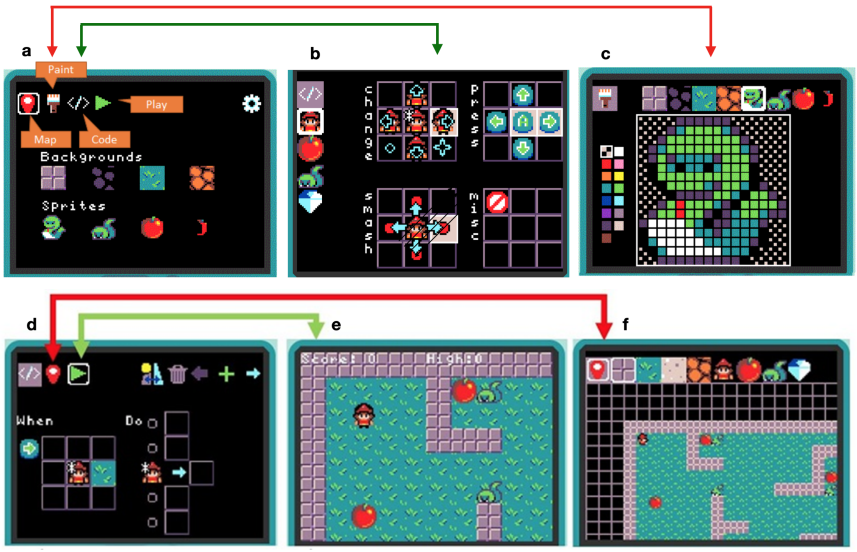

Figure 2: TileCode screens: (a) game home screen, (b) rules menu, (c) paint sprite editor, (d) rule editor, (e) gameplay (f) map editor.

map icon at the top left of the game home screen activates the "map screen" of Figure 2(f), allowing users to design the game map with tile backgrounds and sprites. Third, selecting the paint icon on the game's home screen activates the "paint screen" of Figure 2(c), where the user can change the artwork associated with the four backgrounds and sprites.

Programming Model. TileCode programming involves writing a set of rules that describe sprite behavior, much in the spirit of AgentSheets [74]. Figure 2(b) presents the possible rules for the player sprite, clustered by their type. Figure 2(d) displays a single-player sprite rule, which consists of a **When** pattern on the left-hand side and a **Do** action on the right-hand side. This rule reads as follows: when the user presses the right directional pad (D-pad) button and there is grass on the tile to the right of the player sprite, then send the player sprite a move-right command. TileCode rules execute in parallel over the game map. The user can code this rule by navigating to the tiles in the **When** and **Do** sections to create the pattern to match and the action to take. At the top left of the screen in Figure 2(d) are the familiar menu actions for running the game Figure 2(e) and modifying the map Figure 2(f). Other menu options to the right allow new rules and navigation.

TileCode can run on a variety of gaming handhelds, but it is also available as a web app. This portability was critical for our study (especially when we needed to provide a tutorial or example via screen sharing). The web app simulates the gaming handheld user interface and provides keyboard bindings for the D-pad and A/B buttons.

2.3.2 ***Game Videos.*** We created six games using TileCode and recorded short videos of the gameplay, each about 40-60 seconds long. The six games included: Snake (control a snake of increasing length without colliding into itself), Boulder (control a player to avoid falling boulders and collect diamonds), and Bejeweled (shuffle

Figure 3: Printout for Family Game Design Activity.

objects around so three objects of a kind can be connected) (videos shown during the first session) and Pacman (control a player to eat coins and avoid ghosts), Side Scroller (control a puppy to jump over snakes on a treadmill) and Space Invaders (control a ship to fire pellets at other ships and avoid their attacks) (videos shown during the last session). The videos did not include examples of rules and had no audio.

2.3.3 ***Game Design Activity.*** For this activity, we designed a printout paper sheet where both children and parents were to answer the following questions: "What are the game characters?", "What are their actions?", "What is on the game map?" and "What are the game rules?" We asked parents to print out the form in advance of the session and encouraged both children and parents to fill out the sheet. When designing this activity, we built on prior studies showing that unplugged programming activities [91] and physical game design sheets [45, 62] are effective ways for junior high school students to improve and develop their computational literacy skills. The printout sheet is included in described in Figure 3.

2.3.4 ***Game Patterns.*** Soloway and his colleagues provided evidence that both novice and expert programmers have schemas that match commonly used code patterns, which they termed *programming plans*. Programming plans are small program fragments that achieve a goal, like selecting values from a list that match specific criteria [85]. Other studies have shown that novice programmers use many code tracing and pattern recognition strategies [32]. Specific research on children's game programming showed that providing programming patterns and templates for different game types could facilitate computational literacy and expression [33, 44, 53, 93]. These approaches encouraged us to create a collection of ten game patterns that children could use as examples when devising their game plans and game behaviors. Our patterns provided examples of different game behaviors (e.g., throwing objects, animated motion). For each example, we showed an animation of the game's behavior in action. In addition, we showed images of the combination of game rules that could be used to achieve particular behavior.

2.3.5 ***"Guess the Rule" Quiz.*** After watching the videos, we asked the children to answer the "Guess the rule" quiz. We shared the quiz on our computer and asked children to talk aloud as they replied to questions. The quiz asked questions about three rules: a motion rule, a collision rule, and a trigger rule. For the first two rules (motion and collision), children had to pick the correct explanation (multiple choice). For the trigger rule, the children had to write down their explanations. The quiz is included in the appendix.

2.3.6 ***Handheld Arcade.*** We distributed a low-cost, battery-powered gaming handheld (a "Meowbit") to each participating family so they could work with TileCode without needing access to the Internet or a computer with a web browser. When the Meowbit runs TileCode, it can store eight games at a time. Families could copy their video games from the Meowbit to their computer and then send them to us, letting us inspect and test their games. Unfortunately, the ability to transfer games between the TileCode browser application and the Meowbit was not available at the time we ran the study. The device has four directional buttons and two selection buttons (select and back). In addition, it contains a 1.8' full-color screen, 6 x programmable buttons, 1 x buzzer, built-in light sensor, temperature sensor, and SD card slot (for external storage).

2.4 Data Collection and Analysis

Our data included pre-and post-perception game descriptions and video recordings of all sessions. One of the first sessions was not appropriately recorded, resulting in five minutes of video. The first session averaged 38 minutes in length (not including the five-minute outlier), while the second, third, and fourth sessions averaged 45, 40, and 47 minutes in length, respectively. We captured 43 hours of video, an average of about 170 minutes per family. The first author transcribed the videos for qualitative analyses and noted comments about children's body language and non-verbal interactions. The final corpus included 1,088 pages of transcripts (317,384 words). Once all transcriptions were completed, the first two authors reviewed half of the data independently, as described below.

2.4.1 ***Theoretical framework.*** Our qualitative analysis was informed by *Joint-Media Engagement* (JME) framework to guide how we analyzed how family members interacted with each other during the study sessions. JME explores "spontaneous and designed experiences of people using media together" [92]. Joint-media engagement between parents and children is linked to higher self-efficacy and expertise with computers [59], increased family connectedness [69], increased engagement and interest in computing [83], and improved creativity and thinking skills during pair-programming with parents [2]. Informed by this extensive prior work on JME in non-programming media, our JME analysis examined inter-participant interactions during game design and programming, analyzing interactions around specific programmatic and content aspects of the family's engagement with the platform.

Although not theoretical, our analysis was also informed by prior work that examines the scope of learners' use of programming constructs to characterize their program understanding [20, 52, 99]. This methodological frame aligns with JME because it uses particular programming language constructs as a focal point for interaction with a platform.

2.4.2 ***Analyzing games.*** To analyze interactions through a JME lens, the first two authors separately analyzed each transcript using a combination of etic codes developed from our theoretical frameworks and emic codes that emerged from the interviews themselves [65, 71]. Next, we listed all the program understanding, planning, and writing practices [85] specified in prior studies with novices learning how to code [98] and identified connections with a series of themes that emerged from our study. After we developed a final coding frame, all transcripts were coded by the first author. If new codes emerged, both authors discussed discrepancies in the analyses until they reached an agreement. We used this process to develop categories, which we then conceptualized into broad themes after further discussion [12]. Table 1 presents the final list of codes, their definitions, and their presence across the different study sessions.

Our resulting categories focused on interactions around three aspects of families interactions in TileCode. First, we analyzed *rule types*, aiming to provide a low-level picture of the kinds of rules within an artifact. Second, we analyzed *game types* to broadly characterize the genre of the entire game. Third, we analyzed *pattern types* to highlight the game mechanics and computational patterns (if any) in their work.

Rule types. To analyze structure, we analyzed the rules in each artifact. We looked for change, keypress, and collision rules created by each family. We ignored rules created with direct help from or by a researcher. We considered rules "complex" if they involved multiple kinds of tiles; otherwise, they were called "simple." A simple

			Study sessions			
	Code	**Definition**	**1**	**2**	**3**	**4**
Reading semantics	*Identify elements*	Recognize game characters or map components	x			x
	Identify events	Recognize game triggers or behaviors	x			x
	Connect elements to events	Determine connections between game elements and specific game events or player actions	x	x		x
	Identify game play	Recognize how a series of game events and actions compose a familiar game play	x	x	x	x
Reading templates	*Identify patterns*	Recognize a combination of rules that creates a known game mechanic	x		x	x
	Learn patterns	Learn to recognize a common combination of rules from a family member or researcher		x	x	x
	Plan pattern	Devise new game patterns while still observing existing game patterns		x	x	
	Unsure patterns	Not being sure if an observed combination of rules leads to a known game mechanic	x	x	x	
Writing semantics	*Plan goals*	Plan game goals and types of elements/map		x	x	
	Create element	Create or change a game character	x	x	x	
	Create map	Create or change the game map or specific tiles	x	x	x	
	Rules Changes	Modify, test and correct existing rules	x	x	x	x
	Rules Editing	Create a new rule and debug rule conflicts		x	x	x
Writing templates	*Plan mechanic*	Plan a specific game behavior or action		x	x	
	Implement pattern	Implement a specific game behavior or action		x	x	x
	Correct pattern	Correct the order or choice of rules for a pattern		x	x	x
	Debug combinations	Test and correct flow of multiple game actions			x	x
Family Joint Engagement	*Collaboration*	Children and parents support each other	x	x	x	x
	Parent prompt	Parent prompts a child with examples or questions	x	x	x	x
	Make it fun	Changes made to make the games more fun	x	x	x	x
	Game design	Discuss and plan various game designs		x	x	
	System Feedback	Provide feedback and ask for specific features		x		x

Table 1: List of final codes used for transcripts analysis, their definitions and presence across the different study sessions.

rule is "When the right arrow is pressed, move that sprite to the right." A complex rule example is "When the A button is pressed, create a new sprite." The latter rule involves multiple constraints because it is triggered by pressing the A button, leading to the creation of a new character. Complex rule examples include deleting one of the colliding game elements, generating new backgrounds or sprites upon collision, and collecting points. For every family, we tracked whether they had (or had not) created simple or complex rules of each of the three types (change, keypress, and collision).

Game types. To analyze families' design choices, we categorized the kinds of games created by each family. The first two authors clustered all games across the four sessions into three groups based on the overall game genre. These groups emerged through a joint inductive-deductive approach [65, 71]. We tracked whether they had created a game that belonged to each group for every family.

Pattern types. To provide another lens into computational complexity inside a game, we looked for design patterns that commonly occur in video games. We developed a set of patterns to look for within families' final games. For every family, we deductively tracked whether they had constructed an instance of a pattern from each category. We describe these patterns in Table 2.

In each category, we report the presence or absence of a category instead of counts; since our participants used TileCode for differing amounts of time, actual counts would not be comparable across families. However, creating a specific type of rule, game or pattern is categorical evidence of engaging with computation in TileCode since there are many ways to use it without creating rules at all (e.g., playing with existing games or just modifying game maps). The first two authors independently performed these analyses on all games from these sessions and compared their results. We resolved any mismatches by referring to the video and the saved game.

3 RESULTS

We now present an overall summary of our perceptions of families' experiences and then discuss our results relative to the following research question: *How do families jointly engage in rule-based 2D video-game programming?* Our qualitative analysis revealed that throughout the study, parents played a significant role in supporting children to engage in video games programming by jointly engaging in 1.) **Problem Formulation (Abstraction)** during session 1 on game observations and session 2 on game design; 2.) **Solution Expression (Automation)** during session 2 on game programming and session 3 on game patterns; 3.) **Execution& Evaluation (Analysis)** during session 4 when analyzing games anew and responding to "Guess the rule" quiz. The level to which each family jointly engaged in these practices varied. To illustrate

Game	Description
Static	No rules, map edited witch new tiles or sprites (i.e., tile art with colors)
Animated	With sprites that move on their own (i.e. ghosts that move around)
Interactive	At least one sprite is controlled by the player, either by pressing keys or by painting the map with other sprites.

Pattern	Description
Collectables	Collect items (i.e. food in Snake)
Obstacles	Avoid map elements or other sprites (i.e. avoid falling boulders in BoulderDash)
Firing	Players can create sprites on key press (i.e spaceship firing missiles)
Animation	Non-player sprites can move on their own (i.e. wheels on a conveyor belt)
Painting	Player edits map when moving (i.e. creating a pond trap for snakes)
Portal	Player can teleport from one tile to another on different regions of the map.
Win /Lose	Determine how to win or lose a game (i.e. collect all food to win)

Table 2: (a) Game Types, (b) Game Patterns.

this variation, we present several engagement tactics that emerged from our inductive analysis.

3.1 Family Joint Engagement in Game Programming

Concerning learning to program, the contexts in which families use coding environments matter. In particular, researchers have highlighted how co-engagement or joint media engagement (JME) [92] is supportive of family learning. For example, our study identified four main instances of joint family engagement when playing, designing, and programming games: **Collaboration, Prompting, Make it fun, Game design**.

We observed children and parents **collaborate** in many distinct ways: by having parents explain technical aspects of the programming platform, by debugging rules and pattern implementation together, or by building on each other's ideas for game designs. For example, the mom in F3 explained that the Princess sprite looked different on the gameplay screen versus the map editing screen because of the reduced resolution; the dad in F5 explained how the boulders in Boulder Dash fell in terms of the principle of gravity. In addition, parents collaborated with children when debugging by reminding them to test their rules or by proposing ways to correct game patterns:

> *"So now, if you go back to your game, you will see that you are not able to push the cat on the walls anymore. Because whenever the cat smashes into the wall, it has to stop."* — S., mom F1, referring to pushing object pattern. *"Oh, let me try that and see if it bounces."* — G., child F1, testing his mom's suggestion.

Prior studies that analyzed children's programming games alone showed that youth would often create more complicated code scripts (analogous to a set of rules in TileCode) before testing their programs. This would result in children having a harder time identifying where and why their code breaks [13, 94]. Even if the parents in our study did not always know how to create or correct a game rule, they supported their children by prompting them always to test their changes as they were making them. In this process, parents and children jointly engaged in a three stages model for game modding and programming, "Play-Fix-Create/Mod," which maps to

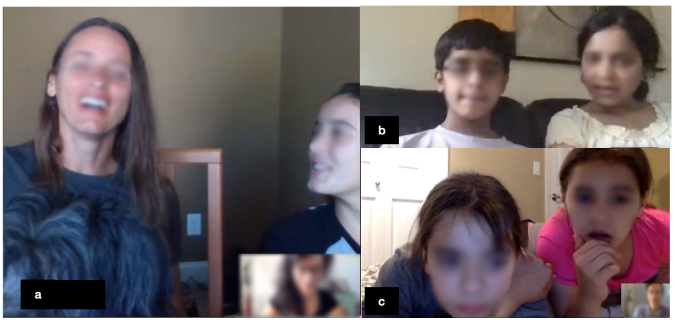

Figure 4: Examples of joint family engagement during the study: a) mom & daughter in F11 discussing game design; b) mom & son in F12 debugging their code; c) sisters in F3 watching game videos.

the stages of progressive engagement with computational thinking "Use-Modify-Create" observed in prior studies [54]. The parents in F1, F3, F8, F13, F15 **prompted** their children to elaborate on their game ideas and contributed their suggestions. Parents' prompts were also helpful when children did not know how to implement their games; parents would ask, "What should this character do?" after the child created a new game element and added it to the map. When children did not understand specific rules or game patterns, parents sometimes proposed useful metaphors and examples from real life to help them:

> *"I do not know how to do this portal thing."* — S., child F10, trying to adapt the portal pattern example. *"It is like, you know, Doctor Strange in the movie; he opens a hole, and he steps through it, and he appears somewhere else. So then the somewhere else is somewhere on the map."* — J., dad F10, explaining the portal pattern to his daughter.

Prior studies where children were designing and programming video games alone showed that many students did not start working and implementing their game ideas right away. In contrast, others started with one idea, abandoned it quickly, and moved on with a new one [**?**]. In our study, parents helped keep their children

on task and supported them to scaffold their ideas by encouraging them to pick one thing to try:

"I was trying to figure out what they can put on the cartwheel." — A., child F13, referring to his Scroller game. *"Oh yeah, exactly, so tell me what sprites you are putting on."* — R., mom F13, responding to her son.

Families also jointly engaged in **game design** by imagining how they could make the games from the videos more fun by proposing various modifications: add timers, add multiple screens and levels, increase the number of obstacles over time, collect game elements for re-using later, increase the number of sprites and portals over time, and play the game with multiple player sprites at once (e.g., two snakes).

"Yes, so basically we want to have two snakes. One is like I control, and I do not want it just moving around by itself." — C., age 14, F2, referring to a snake game.

The family interaction we observed during these brainstorming sessions mirrored the experiences of children pair-programming in non-family contexts [38] and remixing existing games in new creative ways [20].

3.2 Problem formulation: Post Game Descriptions & Game Design

During the first session, when asked to observe and describe games, families either focused on identifying game elements, identifying game events, or recognizing familiar game mechanics and different forms of playing the game. Participants could formulate more elaborate game descriptions when familiar with a particular game mechanic or type. For example, F14 recognized the similarity between the Bejeweled game shown in the study and the Candy Crush game and fully described game rules and logic.

When unfamiliar with a particular game, families only described the game's most salient aspects. For example, F9 and F4 identified how the player in Boulder Dash collected diamonds but did not recognize that it needed to avoid falling boulders. Likewise, F8 identified that the snake in the Snake game eats apples but did not observe that it ends if the snake touches its tail. During this session, parents primarily helped their children to engage in different levels of game reading by prompting them to describe the actions/interactions between the characters (support with abstraction) or by helping them to recognize similar game behavior or game mechanics from other familiar games (support with pattern recognition):

"You're trying to collect all the gems, then you have to try to avoid the boulders falling on you" — M., age 8, F10, describing the Boulder Dash video. *"And what happens if you have multiple boulders?"* — N., dad F10, helping his daughter to identify additional game events.

Prior work has demonstrated that children being able to engage in game "reading" supports their ability to write their games and learn essential programming concepts such as abstraction, decomposition, pattern recognition, and algorithm design [44, 57]. Moreover, Repenning et al. have shown that recognizing the pragmatics of programming rules and being able to comprehend programs in the context of specific situations is one of the main cognitive challenges in video-game programming for youth [76]. In this context, parental support in describing and decomposing game mechanics can help children overcome some of the common challenges when programming their games.

3.3 Solution Expression: From Game Concept to Code

In session two, we observed a clear pattern of family programming behavior. Participants started by formulating game goals and events and then designed the elements and map of the game; as they added elements to the map, they created rules to control game elements or make them interact with the map. What varied were the tactics that family members used when planning and programming their games. One primary source of variation was how families generated reading and writing semantics practices. When asked by researchers how they would make the game more fun or more challenging, families proposed changing game rules by either changing the speed of motion for sprites (F5, F10), adding more obstacles (F6), or adding a timer for specific actions (F7).

"Basically, we want to have two snakes. One that I control and one that is just moving around by itself." — C., age 14, F2, referring to his changes to the Snake game.

When families designed and planned their games using our "Game Design" sheet, children found it difficult to scaffold their complex game ideas (e.g., Minecraft (F11, F12) or Chess (F2)) into a set of game elements and rules that could be implemented in TileCode. Parents helped them adapt their ideas to match the platform's capabilities:

"I am thinking of the way the game will look; it will be like a bird's eye view. If you jumped or have anti-gravity boots, you could run on Mars or the Moon ." — R., age 12, F12, referring to his game design idea. *"I do not think you can do 3D in here. What if you add brown tiles to make it look like Mars."* — D., mom F12, helping her son to adapt his game design to TileCode.

Participants typically anchored their game idea around a strong narrative concept and then adjusted game mechanics based on the TileCode platform capabilities as they designed their sprites, edited the map, or created their first rules. The narrative and game mechanics sometimes emerged as participants edited the map or modified the existing sprites gallery. For example, K. (F9) started by creating a haunted house with spider webs on the game map. She later decided to use the map tiles that had spider webs to give clues to the ghosts in her game:

"If we are going to find a treasure, we have a pickup action." — C. & C., ages 11 & 12, F2 planning their game events. *"I am going to put some spider webs in my maze and use the orange sand in a few places. I do not know what it will do later, but it makes it a little scarier."* — K., ages 11 & 12, F2 planning their game events.

This is consistent with findings from prior work [45] that compared two-game programming platform versions, one with a rich narrative and one with a light narrative. It found that participants who programmed in the rich narrative version made fewer programming errors and were more engaged. Other participants, S. and M. (F2), were upset that the default gallery of sprites did not have enough female sprites. So they decided to change the default sprite into a female warrior with a hammer and created an entire game inspired by Minecraft around this character. Concerning rule

creation in their games, children found it easiest to start by creating rules that connected game elements to specific events. Parents often prompted the creation of a new rule when children would add a new element to the game map by asking what the purpose or action of the element is:

> *"Why are you putting a wall there? Do you want to block it?"* — S., mom, F1 asking her son about his most recent map changes. *"He is moving over the walls because we did not tell it he can go over the walls because there is no barrier. "* — G., age 7, F1 referring to the main sprite in his game. *"Oh wait, that is a half-eaten Apple. So whenever I am going to eat an Apple, we will see a snake wait; let me try to do it without dying."* — C., age 11, F2 referring to the Snake game.

This interaction is similar to prior studies which analyzed kids collaboratively creating and modifying video games, finding that students started by playing with preliminary game ideas and discussing what they would change in the game design iteratively as they were creating simple program actions [21, 37].

During session two, it was easier for families to create keypress and collision rules with direction connection either (1) between player actions and game elements actions (e.g., press right arrow moves sprite to the right) or (2) between two game elements on the map (e.g., when a snake touches an apple). On the other hand, creating change rules for game events such as continuous motion (e.g., making a sprite jump over an obstacle) was more challenging since they had to compose game behavior from various rules that kept track of the sprite's state at various moments during the motion on the map (e.g., when a sprite is left of another sprite move up):

> *"The trick is we do not have a way in TileCode to say here is the ghost, and the player is somewhere below me. I want another way to do things, so I have a notion of something it is somewhere below me as opposed to directly below me."* — C., age 14, F2, referring to her ghost animation pattern.

When participants began modifying the map to test a specific rule, they often got distracted and wanted to add more background elements or make more map changes (F1, F5, F9, F14, F15). Parents helped keep their children focused on the task at hand and suggested that the platform automatically add sprites to the map when a new rule for a sprite is created so users do not get distracted by editing the map.

Overall, participants engaged in a combination of game planning and "bricolage" when creating their games, similar to other studies where children were programming their games alone [47]. However, in our study, parents played a significant role in scaffolding the game planning when needed and in encouraging their children to code bricolage via iterative testing and collaborative debugging.

3.4 Solution Expression: From Game Mechanics to Programming Patterns

During the third session, families were introduced to a collection of seven-game patterns (see Table 2). While watching the different pattern animation examples, participants often identified and compared other similar game behaviors they observed in prior study sessions or when they played games on their own. For example, F8 compared the snake's random movement (non-player character movement pattern) with the ghost's movement in the Pac-Man game.

As families picked a collection of three patterns to build a game, we observed how families combined the same three patterns into different games. For example F13, F5 and F10 picked the Firing, Animation and Win/Lose patterns (see Table 2). F13 built a game with a cowboy that can sprinkle fairy dust over flowers (Firing), which then become apples that the cowboy collects to win the game (Win/Lose) while avoiding animated cacti (Animation)(see Figure 5.c). F5 built a racing game between two animals, a panda, controlled by a player (Firing), and a dog that runs by itself (Animation); whoever touches the finish line tiles first wins the race (Win/Lose)(see Figure 5.a). F10 made a game with a student and a teacher where the student is being chased by the teacher (Animation) and leaves a bubble gum trail behind (Firing). If the student touches the dome before the teacher, he wins the game (Win/Lose)(see Figure 5.b).

Overall, families picked various combinations of three patterns for their games in the third session, which resulted in a wide range of game types: interactive games, where a player is being controlled in different worlds (F1, F4, F6, F13); animation games, where a story is being played out (e.g., a haunted house with moving ghosts)(F5, F9, F15); and art games, where the goal is to paint the map either via sprite motion or by direct editing (F2, F8, F10, F14).

Plan mechanics. After using the patterns from our gallery of examples, some of the families (F1, F2, F6, F7, F11, F12, F15) went back to their original designed games and started adapting and combining learned patterns to create new game mechanics. For example, F2 created a modified version of a portal pattern to teleport the player from a green tile to a brown tile (see Figure 6.a); F7 made a bounce pattern to make the puppy move back when touching the wall tile (see Figure 6.b); F12 modified a firing pattern to create new puppy sprites on A press and made them move up (see Figure 6.c).

When creating their patterns, families first described the behavior they wanted to see in the game. They then planned and decided what type of rule combinations were appropriate to create this behavior (e.g., using a collision or changing a rule). Finally, they created one rule at a time, modified the map to test each rule, and corrected the rule or edited the map if needed. Some participants used the results of testing to build more complex models of the games' mechanics. For example, F14, F12, and F10 had a rule that painted the map and another that used newly painted tiles as a constraint (e.g., the teacher sprite could only walk on the bubble gum trail created by Student sprite F10 (see Figure 6.b). Other participants modified push sprites and portal patterns to make multiple game elements interact with each other:

> *"You could have something like a color that's trying to overtake the screen, then the projectiles can send it back like defending, but if you take the time, then you can push another color block in front of it, and then that will stop them."* — C., age 14, F2, planning to use Firing and Painting patterns to control multiple sprites.
> *"Check this out, all the coins become snakes, and they can kill us. Oh my God, this is hilarious."* — C., age 14, F2, referring to his Pacman modifications.

As their games became complex, families had to handle conflicting rules (e.g., a sprite being triggered to move and stop by different

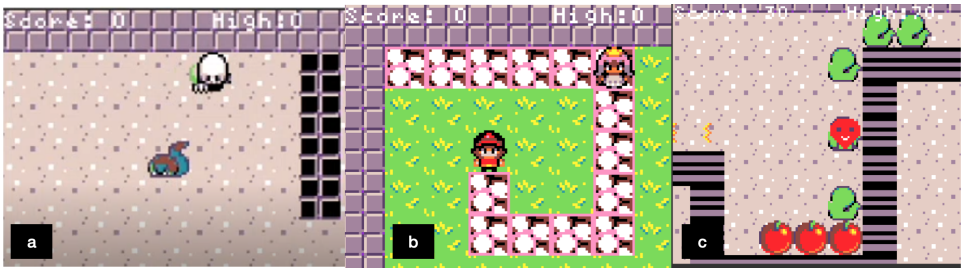

Figure 5: Examples of games families created using three patterns: a) race game between a panda a dog; b) a student being chased by the teacher on a bubble gum trail; and c) a cowboy sprinkling fairy dust over flowers to make them into collectable apples while avoiding animated cacti.

rules). In addition, it was harder for them to modify and correct their patterns when the number of rules increased, making it difficult to keep track of changes. Several families (F2, F4, F11, F15) asked for easier ways to connect multiple game elements by having "if/else" rules. The participants who struggled with animation patterns (e.g., random motion or jumping) also expressed the need for higher-order rules that could describe sprite behavior rather than composing them from a set of discrete rules:

"I would like to control multiple codes at once." — K., age 9, F8, referring to their jumping motion pattern.

3.5 Execution & Evaluation: Post Game Descriptions & Quiz

In their post-game descriptions, most families (except F14 and F10) formulated elaborate descriptions for each game, including the games they were not familiar with and saw for the first time. Unlike the pre-game descriptions, they explained game events and elements in detail and the conditions necessary to trigger these events (e.g., the shooter only moves left to right, and the space ships slide down when on black tiles in Space Invaders). In post descriptions, children also found it more accessible than pre-game ones to recognize player actions required to trigger game events (e.g., pressing the D-pad vs. the A button) and identify additional actions, such as the creation or deletion of sprites. Parents did not have to intervene with prompts or questions to help them describe the videos in this session.

"Or maybe there are ways to tell the system like generate ten snakes and make a move, and you would know where to place them and how to make them move. " — R., age 12, F12, sharing his feedback for future features.

Of 15 families, 12 participated in the "Guess the rule" quiz. All 12 families identified the correct explanation for the collision rule example; 9 families identified the correct explanation for the keypress motion rule example; 9 families correctly described the firing pellets pattern. The percentage of children guessing the program rules correctly in our final evaluation suggests a positive trend as compared with other studies where students had to identify program rules in scientific model simulations on their own, and only 14% identified all key rules [97].

3.6 Computational Analysis of Family Games

We now describe the results of our analysis of the rule vocabulary, the presence of specific game patterns, and the overall artifact type families used and created in their games. **Rule types.** All study families created multiple kinds of rules in TileCode (see "Rule Types" column in Table 3), which highlights several important points. First, our results indicate that every family created some new rules in example games, which meant that they had to engage with the system and modify the rules on screen. This is significant: if most families were unable to create new rules in example games, then the interface may have been poorly designed. Second, separating rules into simple (affecting one sprite or background tile) vs. complex (affecting multiple sprites or background tiles) is analogous to classifying code as modifying one object vs. modifying multiple objects. From a SOLO Hierarchy perspective, [58], the ability to create simple or complex rules can be seen as reflecting unistructural vs. multistructural understanding of TileCode's programming model. Simple rules create simple mechanics (like pressing a button to move a sprite), while complex rules can achieve more sophisticated game mechanics (e.g., pressing one button to create a new sprite next to a current sprite or "firing").

As the first three columns of Table 3 show, several families created complex rules of multiple types. For instance, siblings in F2 created a complicated maze game with four arrows controlling the motion of a diver sprite. They created complex press rules for each type of arrow that could be pressed, reflecting their multistructural understanding that an event (keypress) triggered on one action (move left) could modify another (diver).

Game types. Our analysis of game types revealed three genres of games our families created: **static, animated** and **interactive** (see Table 2). Table 3 ("Game Types") summarizes our classification of artifacts and Figure 7 shows examples of games for each game type.

Game patterns. Our analysis of game patterns revealed several practices. Almost all families (except three) used the *Win/Lose* pattern in their games; the next most popular were *Obstacles* (11 families) and *Animation* (10 families). Even when families picked the same patterns for their games, they used and adapted them in different ways to create very different mechanics (see Figure 5). Table 2 describes the patterns collection we provided in session three, while Figure 6 provides examples of families' new patterns.

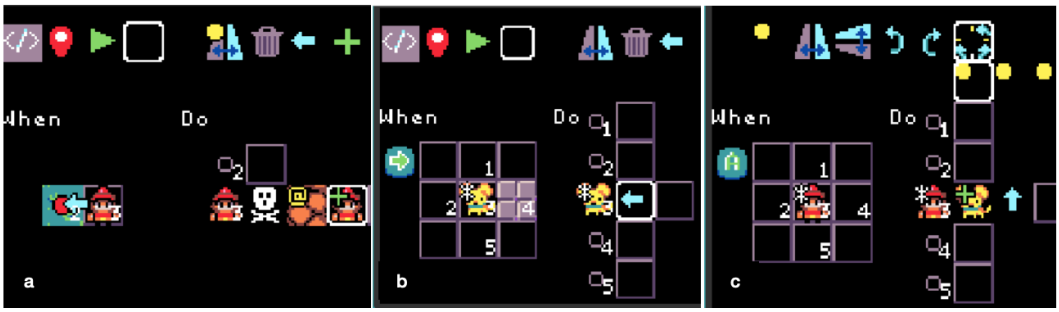

Figure 6: Examples of patterns created by families during the study: a) portal pattern which teleports the player from a green tile to a brown tile; b) bounce pattern which makes the puppy move back when touching wall tile; c) firing pattern which creates new puppy sprites on A press and makes them move up.

ID	Parents & Ages	FL	Children & Ages	Rule Types: Change (CH), Key (K) & Collision (C) *=complex			Games: Static, Animated, Interactive			Pattern Types: Collectible, Obstacle, Firing, Animation, Portal, Paint, Win					
				CH	K	C	S	A	I	C	O	F	A	P	Pa	W
F1	Mom, Dad (35)	Spanish	Boy (7)	✓	✓*	✓*	✓		✓	✓	✓			✓		✓
F2	Mom (45)	Thai	Boy (11) Girl (14)	✓*	✓*	✓*	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
F3	Mom (45)	None	Girls (10 & 12)	✓	✓	✓*	✓		✓	✓	✓		✓			
F4	Mom (40)	None	Boy (9) Girl (11)		✓	✓*	✓		✓		✓	✓				✓
F5	Dad (45)	None	Girls (7&9)	✓*	✓	✓		✓	✓	✓			✓			✓
F6	Mom (35)	Spanish	Girl (10)	✓	✓*	✓		✓	✓		✓	✓	✓			✓
F7	Mom (45)	Indonesian	Boy (10)	✓*	✓*	✓		✓	✓		✓	✓	✓			
F8	Mom (45)	French	Boy (9) Girl (10)		✓*	✓	✓		✓	✓			✓	✓		✓
F9	Mom (35)	Spanish	Girl (10)	✓	✓	✓	✓		✓				✓	✓		
F10	Dad (40)	Hindi	Girl (8)		✓	✓	✓		✓	✓					✓	✓
F11	Mom (35)	Romanian	Girl (14)	✓*	✓*	✓*		✓	✓	✓	✓		✓		✓	✓
F12	Mom (40)	Hindi	Boy (12)	✓	✓*	✓		✓	✓	✓	✓	✓	✓			✓
F13	Mom (40)	None	Boy (11)	✓	✓*	✓*		✓	✓	✓	✓		✓		✓	✓
F14	Mom (35)	Spanish	Girl (9)		✓	✓	✓		✓	✓	✓				✓	✓
F15	Mom (40)	None	Boy (8)	✓	✓*	✓*	✓		✓	✓	✓	✓		✓		✓

Table 3: Computational analysis of each family game.

4 DISCUSSION

Our research question was: *How do families jointly engage in rule-based 2D video-game programming?* Our results suggest that families engaged in video-game programming and planning in a highly non-linear way, switching between design, program understanding, program composition, program decomposition, and program testing. However, unlike many collaboration patterns found in models such as pair programming, families exhibited a dynamic shift in roles and power, with parents sometimes offering to scaffold, prompt, and guide like a teacher might, sometimes contributing as a peer in design ideation, and sometimes following their child's lead in expressing design ideas in code. The result of these complex collaboration patterns was that understanding the platform's programming language and game design patterns was emergent: families refined their understanding by evaluating rules they modified or created together and then imagining richer possibilities once they had learned what was possible. By the final sessions, both parents and children not only demonstrated a greater understanding of the platform's capabilities, but they were jointly able to demonstrate an understanding of complex programming patterns and to engage in game decomposition fully (during game descriptions), game design and program planning (during game design session), program planning with patterns (during game creation

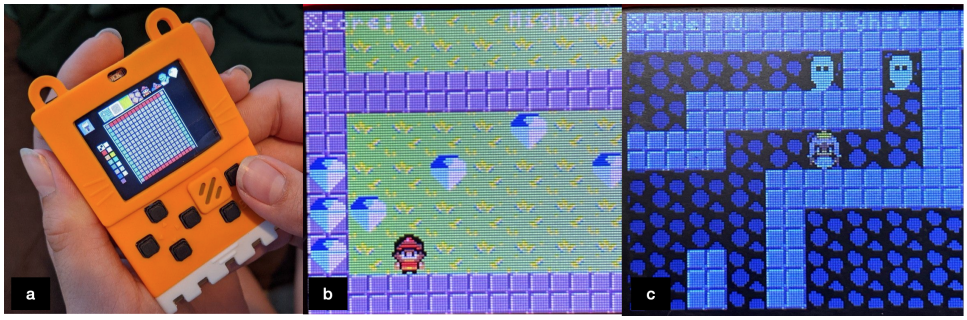

Figure 7: Examples of game types created by families during the study: a) static game displaying different colored tiles; b) animated game where the diamonds move automatically; c) interactive game where you can control a princess to explore a haunted house.

with patterns), and program tracing and explaining (during the "Guess rule game" quiz). These observations suggest that family learning in game programming is distinct from solitary or even peer collaborative learning along multiple dimensions.

Family Joint Game Programming. Our qualitative findings of families' video-game programming joint engagement suggest new interpretations of prior research on child programming practices. Whereas prior work has focused mainly on children's accounts of program understanding (e.g., how children learn to program games [25, 30, 48]), our investigation of family video-game programming from a joint-media engagement lens [89] suggests that children and parents support each other in significant ways to design and program games. These supports include problem formulation via game descriptions and design, solution expression via iterative game planning, programming and debugging, and solution execution and analysis via composition and decomposition of game mechanics to programming patterns. We found that our designs of different activities for game understanding, design, and programming let families with different perceptions, attitudes, and knowledge about games and programming engage in the following learning processes successfully: Activities in sessions 1 and 4 supported families to engage in discovering the programming logic and critical thinking behind popular video games. Activities in session 2 supported families in game design, composing object-oriented narratives, and engaging in iterative programming planning and debugging. Activities in session 3 supported families to develop systems thinking and reason logically and critically about complex scenarios by adapting and creating game patterns. By designing activities that allowed families to move in and across a repertoire of practices [40, 79] we supported multiple forms of participation [39, 67] and created the potential for authentic interactions and expansive learning [29] in the context of family video-game programming.

Our results suggest that engaging families in joint video-game programming can lead families to envision new ways for them to learn about and engage with this medium [68]. Moreover, it presents the possibility to use artifacts they are familiar with, such as video games and handheld arcades, and envisions sites of possibility for computational thinking and expression[67] within their individual and joint dispositions and repertoire of practices. Notably, newly acquired practices and skills led some families to consider making meaningful use of gaming devices in their homes and re-designing their interactions with them. These findings suggest that the family has the potential to act as a space for creative coding with games, where both children and parents can develop programming skills by combining family social contexts for learning (co-play) and their collective zone of proximal development [96].

Platform design choices. While our research questions were not specific to TileCode, the platform inevitably shaped what learning and interactions occurred. Moreover, observations about the specific design of TileCode might reveal implications for other game programming platforms. TileCode programming model supported families to engage in "Play-Fix-Create/Mod" programming flow [54] which enabled iteration and refinement of rules. It also supported graphical re-writes via the use of game patterns which families composition and decomposition of games. Prior studies have shown such graphical re-writes analogies support novices to better understand their code by avoiding the inheritor copy/past/reuse dilemma [72]. Moreover, the use of game patterns that can easily be modified in different game contexts supports learners to create their own logical rules analogies [76].

All rules in TileCode have an input ("When") and output ("Do") (see Figure 6). Typically, participants first created the rule to detect an input condition (e.g., when on green grass) and then tested the rule; they then decided on the output(e.g., destroy one of the other sprites, add points, win the game, lose the game). A TileCode rule editor feature that was helpful when participants were refining their rules input and output conditions was that tiles were numbered so participants could tell exactly where to add more conditions (e.g., add grass tile on tile number 2) (see Figure 6.a).

Families' use of the TileCode system also highlights several opportunities for improving the experience within this paradigm. For example, families struggled when creating continuous motion patterns where they had to keep track of sprites' state. Another issue was keeping track of rule conflicts and rule interactions (emergent behaviors) as the games became more complex. This suggests that future iterations of our platform should provide the option to write higher-level methods that can combine sets of low-level rules into procedures (e.g., if the condition is true, use rule 1 or else use rule 2). A similar solution was implemented successfully in systems like AgentSheets [75] and StarLogo [10]. Moreover, prior studies

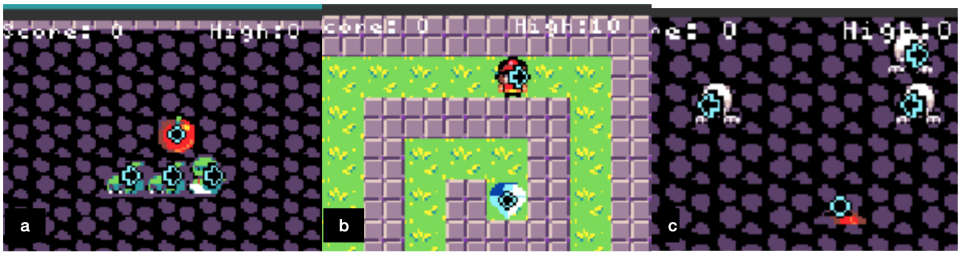

Figure 8: Examples of future debugger feature which can be used to indicate game sprites state: a) snake is moving left while the apple is stationary ; b) player is moving right while the diamond is stationary ; c) skulls are moving from left to right while descending and the space ship is stationary.

suggested opportunities to combine discrete rule creation, programming by demonstration, and direct manipulation techniques to facilitate game programming for youth [19, 77, 84]. However, implementing such functionality within the computing constraints of low-cost handheld devices remains an area of open research.

Other game programming paradigms might prevent difficulties in keeping track of rule interactions. For example, consider a game where the high-level scenario is for the user to control a cat chased by several dogs trying to catch it. In Kodu Game Lab [60, 90], the user can give a single rule, such as "If a dog sees the cat, move the dog towards the cat" to realize the behavior, which can be complex (the predicate "sees" depends on the distance between the sprites and whether a third sprite is between them, and the action "move towards" has similar behavioral complexity). On the other hand, in TileCode, families need to consider how to express the "chasing behavior" via a set of local rules based on exact matching of patterns. These differences between TileCode and other languages illustrate how the particular semantics of a platform can have an impact on what families can learn to express.

Future Work *Debugging support.* One potential new design feature for TileCode inspired by the findings of this study is to provide a debugger option that families could use when they want to keep track of sprites state as different rules are being executed. This can allow families to more easily identify the correct sequence of rules required to create specific animates (e.g., how should the move the head of the snake vs. the body of the snake in turns (see Figure 8.a). While we have used this feature internally while developing the TileCode Platform, we believe it could be beneficial for family members to use it when necessary. It would also be helpful to modify the rule editor to highlight rule conflicts. There is an opportunity on the web simulator to show both the gameplay and the rule editor in parallel and highlight rules as they are triggered in the game (e.g., show the rule for smashing when the snake eats the apple).

Controlled Automation. Children and parents also requested the option of auto-generating some of the game background elements based on their initial description of the game design. Prior work on the Gamechangineer platform showed how games could be generated based on natural language descriptions [44]. We believe there future iterations of TileCode on the web simulator or tablets could provide a digital version of our game design printout sheet (see Figure 3) and use the natural language descriptions to auto-generate [15] some of the game map elements based on users answers in the game design sheet. Similarly, it would be useful to automatically add sprites to the game map when a new rule for a sprite is created by building on prior work on ontology-driven generation of interactive games [73].

Progressive features disclosure. Prior work on advanced features disclosure for programming interfaces showed that children were able to use progressive systems such as Emile to create reasonably sophisticated programs and gained a qualitative understanding of kinematics in the process [41]. We want to explore further how to enable a progressive disclosure of features in TileCode that could account for changes from device to device (e.g., more features on tablet & laptop vs. arcade) and show new programming features based on the games complexity.

Limitations. It was impossible to observe every family interaction with every study activity systematically, nor did every family member speak in every family; it may be those family members who verbalized more reasoned differently than those who verbalized less. For the interactions we *could* observe, observing a child reason about a specific programming concept or game behavior did not necessarily indicate ground truth for their conceptions; for example, it may be the case that family members were reasoning in similar ways but were verbalizing their reasoning differently. We also did not have data for all game description questions from all sessions, nor did our study cover the many possible ways that culture, community, and collaboration might have shaped sense-making. Moreover, since our analysis was episodic rather than temporal, family programming strategies may have been highly variable within individual and group behavior. Our observations were also inevitably influenced by the specific activities, instruction, and scaffolding we designed. Future work should explore other forms of instruction and scaffolding and more diverse families to reveal other types of family learning through game programming.

Therefore, while a cautious interpretation of our results suggests that the families in our particular intervention demonstrated diverse video-game programming strategies and a shift toward increased computational literacy, other populations or a more granular assessment of individual parent and child knowledge could reveal new types of programming practices and different shifts in programming and game understanding.

Our specific activities could have predisposed families toward particular modes of collaboration; other forms of instruction and

scaffolding might have led to different interactions and learning outcomes. Therefore, our results are best scoped to creative, constructionist learning contexts and likely less relevant to more guided learning settings dominated by direct instruction.

5 CONCLUSION

Programming does not need to be a solitary activity, especially when introducing children to computing through the programming lens. We studied how family members can support each other as they describe, design, program, and debug video games. We saw positive engagement and advancement in various programming practices through a four-week observational study, where we introduced families to programming via low-cost device arcades using a domain-specific language. By helping to promote computational literacies for families, this work will better position us to respond to a future in which computation is embedded in families' everyday lives.

6 SELECTION AND PARTICIPATION OF CHILDREN

We recruited families by posting an announcement on several family forums, social media groups, and family slack channels in North America. A total of 120 families applied to participate in the study, and we selected 19 families, trying to be as inclusive as possible along the following dimensions: family structure, ethnicity, geographical location, and socio-economic background. Of those 19 families, 15 attended all four sessions. The families unable to participate in all sessions (due to extraordinary family circumstances or scheduling difficulties) were excluded from the final study analysis.

All parents and children over seven years old signed consent forms agreeing to participate in our study. The forms described the video and artifact data to be collected and how it would be analyzed.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1539179, 1703304, 1836813, 2031265, 2100296, 2122950, 2137834, 2137312, and unrestricted gifts from Microsoft, Adobe, and Google.

REFERENCES

[1] Espen J Aarseth. 1997. *Cybertext: Perspectives on ergodic literature.* JHU Press.
[2] Uzi Armon. 1997. Cooperative parent-child learning in a LEGO-Logo environment. *Retrieved January* 30 (1997), 2010.
[3] Thomas Ball, Abhijith Chatra, Peli de Halleux, Steve Hodges, Michał Moskal, and Jacqueline Russell. 2019. Microsoft MakeCode: embedded programming for education, in blocks and TypeScript. In *Proceedings of the 2019 ACM SIGPLAN Symposium on SPLASH-E.* 7–12.
[4] T Ball, S Kao, R Knoll, and D Zuniga. 2020. TileCode: Creation of Video Games on Gaming Handhelds. *Proceedings of the 33rd Annual ACM* (2020).
[5] Rahul Banerjee, Leanne Liu, Kiley Sobel, Caroline Pitt, Kung Jin Lee, Meng Wang, Sijin Chen, Lydia Davison, Jason C Yip, Amy J Ko, et al. 2018. Empowering families facing english literacy challenges to jointly engage in computer programming. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems.* 1–13.
[6] 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.* 230–237.
[7] Tiffany Barnes, Heather Richter, Eve Powell, Amanda Chaffin, and Alex Godwin. 2007. Game2Learn: building CS1 learning games for retention. In *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education.* 121–125.
[8] Brigid Barron, Caitlin Kennedy Martin, Lori Takeuchi, and Rachel Fithian. 2009. Parents as Learning Partners in the Development of Technological Fluency. *International Journal of Learning and Media* 1, 2 (May 2009), 55–77. https://doi.org/10.1162/ijlm.2009.0021
[9] Ashok R. Basawapatna, Kyu Han Koh, and Alexander Repenning. 2010. Using Scalable Game Design to Teach Computer Science from Middle School to Graduate School. In *Proceedings of the Fifteenth Annual Conference on Innovation and Technology in Computer Science Education* (Bilkent, Ankara, Turkey) *(ITiCSE '10).* Association for Computing Machinery, New York, NY, USA, 224–228. https://doi.org/10.1145/1822090.1822154
[10] Andrew Begel and Eric Klopfer. 2007. Starlogo TNG: An introduction to game development. *Journal of E-Learning* 53, 2007 (2007), 146.
[11] Ian Bogost. 2016. *Play anything: The pleasure of limits, the uses of boredom, and the secret of games.* Basic Books.
[12] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
[13] Neil CC Brown and Greg Wilson. 2018. Ten quick tips for teaching programming. *PLoS computational biology* 14, 4 (2018), e1006023.
[14] Margaret M Burnett and David W McIntyre. 1995. Visual programming. *COMPUTER-LOS ALAMITOS-* 28 (1995), 14–14.
[15] Brock Angus Campbell and Christoph Treude. 2017. NLP2Code: Code Snippet Content Assist via Natural Language Tasks. (Jan. 2017). arXiv:1701.05648 [cs.SE]
[16] Meng-Tzu Cheng, Jhih-Hao Chen, Sheng-Ju Chu, and Shin-Yen Chen. 2015. The use of serious games in science education: a review of selected empirical research from 2002 to 2013. *Journal of computers in education* 2, 3 (2015), 353–375.
[17] Thomas M Connolly, Elizabeth A Boyle, Ewan MacArthur, Thomas Hainey, and James M Boyle. 2012. A systematic literature review of empirical evidence on computer games and serious games. *Computers & education* 59, 2 (2012), 661–686.
[18] Kathryn Cunningham. 2020. Purpose-first Programming: A Programming Learning Approach for Learners who Care Most About What Code Achieves. In *Proceedings of the 2020 ACM Conference on International Computing Education Research.* 348–349.
[19] Allen Cypher and David Canfield Smith. 1995. KidSim: End user programming of simulations. In *Proceedings of the SIGCHI conference on Human factors in computing systems.* 27–34.
[20] Sayamindu Dasgupta, William Hale, Andrés Monroy-Hernández, and Benjamin Mako Hill. 2016. Remixing as a pathway to computational thinking. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing.* 1438–1449.
[21] Jill Denner, Linda Werner, and Eloy Ortiz. 2012. Computer games created by middle school girls: Can they be used to measure understanding of computer science concepts? *Computers & Education* 58, 1 (2012), 240–249.
[22] Betsy DiSalvo, Cecili Reid, and Parisa Khanipour Roshan. 2014. They can't find us: the search for informal CS education. In *Proceedings of the 45th ACM technical symposium on Computer science education.* 487–492.
[23] Stefania Druga. 2018. *Growing up with AI: Cognimates: from coding to teaching machines.* Ph.D. Dissertation. Massachusetts Institute of Technology.
[24] Stefania Druga, Fee Christoph, and Amy J. Ko. 2022. Family as a Third Space for AI Literacies: How do children and parents learn about AI together? *CHI '22: ACM Conference on Computer-Human Interaction* (2022).
[25] Stefania Druga and Amy J Ko. 2021. How do children's perceptions of machine intelligence change when training and coding smart programs?. In *Interaction Design and Children.* ACM, Athens Greece, 49–61. https://doi.org/10.1145/3459990.3460712
[26] Brianna Dym, Cole Rockwood, and Casey Fiesler. 2021. Learning Computing through Transformative Works: A Case Study of Game Modding. In *2021 Conference on Research in Equitable and Sustained Participation in Engineering, Computing, and Technology (RESPECT).* IEEE, 1–1.
[27] Magy Seif El-Nasr and Brian K Smith. 2006. Learning through game modding. *Computers in Entertainment (CIE)* 4, 1 (2006), 7–es.
[28] Francis Emmerson. 2004. Exploring the video game as a learning tool. *ERCIM news* 57 (2004), 30.
[29] Yrjö Engeström. 2015. *Learning by expanding.* Cambridge University Press.
[30] Sarah Esper, Stephen R Foster, and William G Griswold. 2013. CodeSpells: embodying the metaphor of wizardry for programming. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education.* 249–254.
[31] Carrie A Ewin, Andrea E Reupert, Louise A McLean, and Christopher J Ewin. 2021. The impact of joint media engagement on parent–child interactions: a systematic review. *Human Behavior and Emerging Technologies* 3, 2 (2021), 230–254.
[32] Sue Fitzgerald, Beth Simon, and Lynda Thomas. 2005. Strategies that students use to trace code: an analysis based in grounded theory. In *Proceedings of the*

first international workshop on Computing education research. 69–80.
[33] Diana Franklin, David Weintrop, Jennifer Palmer, Merijke Coenraad, Melissa Cobian, Kristan Beck, Andrew Rasmussen, Sue Krause, Max White, Marco Anaya, and Zachary Crenshaw. 2020. 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* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 794–800.
[34] Nate Garrelts. 2014. *Understanding Minecraft: essays on play, community and possibilities*. McFarland.
[35] James Paul Gee. 2007. *Good video games+ good learning: Collected essays on video games, learning, and literacy*. Peter Lang.
[36] Madhu Govind, Emily Relkin, and Marina Umaschi Bers. 2020. Engaging children and parents to code together using the ScratchJr app. *Visitor Studies* 23, 1 (2020), 46–65.
[37] Marianthi Grizioti and Chronis Kynigos. 2021. Children as players, modders, and creators of simulation games: a design for making sense of complex real-world problems. In *Proceedings of the 20th ACM Conference on Interaction Design and Children. ACM*.
[38] Shuchi Grover, Marie Bienkowski, Amir Tamrakar, Behjat Siddiquie, David Salter, and Ajay Divakaran. 2016. Multimodal analytics to study collaborative problem solving in pair programming. In *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge*. 516–517.
[39] Kris D Gutiérrez, P Zitlali Morales, and Danny C Martinez. 2009. Re-mediating literacy: Culture, difference, and learning for students from nondominant communities. *Review of research in education* 33, 1 (2009), 212–245.
[40] Kris D Gutiérrez and Barbara Rogoff. 2003. Cultural ways of learning: Individual traits or repertoires of practice. *Educational researcher* 32, 5 (2003), 19–25.
[41] Mark Guzdial. 1994. Software-realized scaffolding to facilitate programming for science learning. *Interactive learning environments* 4, 1 (1994), 001–044.
[42] Brian Harvey, Daniel D Garcia, Tiffany Barnes, Nathaniel Titterton, Daniel Armendariz, Luke Segars, Eugene Lemon, Sean Morris, and Josh Paley. 2013. Snap!(build your own blocks). In *Proceeding of the 44th ACM technical symposium on Computer science education*. 759–759.
[43] Christothea Herodotou. 2018. Mobile games and science learning: A comparative study of 4 and 5 years old playing the game Angry Birds. *British Journal of Educational Technology* 49, 1 (2018), 6–16.
[44] Michael S Hsiao. 2018. Automated Program Synthesis from Object-Oriented Natural Language for Computer Games.. In *CNL*. 71–74.
[45] Chaima Jemmali, Sara Bunian, Andrea Mambretti, and Magy Seif El-Nasr. 2018. Educational game design: an empirical study of the effects of narrative. In *Proceedings of the 13th international conference on the foundations of digital games*. 1–10.
[46] Henry Jenkins. 2004. Game design as narrative architecture. *Computer* 44, 3 (2004), 118–130.
[47] Yasmin B Kafai. 2012. Learning design by making games: Children's development of design strategies in the creation of a complex computational artifact. In *Constructionism in practice*. Routledge, 87–112.
[48] Yasmin B Kafai, ML Franke, Cynthia Carter Ching, and JC Shih. 1998. Game design as an interactive learning environment for fostering students' and teachers' mathematical inquiry. *International Journal of Computers for Mathematical Learning* 3, 2 (1998), 149–184.
[49] Yasmin Bettina Kafai, Yasmin B Kafai, and Yasmin B Kafai. 1995. *Minds in play: Computer game design as a context for children's learning*. Routledge.
[50] Juho Kahila, Teemu Valtonen, Matti Tedre, Kati Mäkitalo, and Olli Saarikoski. 2020. Children's experiences on learning the 21st-century skills with digital games. *Games and Culture* 15, 6 (2020), 685–706.
[51] Caitlin Kelleher, Randy Pausch, and Sara Kiesler. 2007. Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1455–1464.
[52] Kyu Han Koh, Ashok Basawapatna, Vicki Bennett, and Alexander Repenning. 2010. Towards the automatic recognition of computational thinking for adaptive visual language learning. In *2010 ieee symposium on visual languages and human-centric computing*. IEEE, 59–66.
[53] Chronis Kynigos et al. 2007. Half-baked logo microworlds as boundary objects in integrated design. *Informatics in Education-An International Journal* 6, 2 (2007), 335–359.
[54] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. 2011. Computational thinking for youth in practice. *Acm Inroads* 2, 1 (2011), 32–37.
[55] A Lenhart, J Kahne, E Middaugh, AR Macgill, C Evans, and J Vitak. 2008. Teens' gaming experiences are diverse and include significant social interaction and civic engagement. Pew Internet & American Life Project.
[56] Colleen M Lewis. 2011. Is pair programming more effective than other forms of collaboration for young students? *Computer Science Education* 21, 2 (2011), 105–134.
[57] Madeleine Lexén, Erik Ljungdahl, Hanna Rydholm, and Henning Sato von Rosen. 2020. Programming Arcade Games using Natural Language-Utilizing inherent language skills as a gentler introduction to Computational Thinking. (2020).
[58] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L Whalley, and Christine Prasad. 2006. Not seeing the forest for the trees: novice programmers and the SOLO taxonomy. *ACM SIGCSE Bulletin* 38, 3 (2006), 118–122.
[59] Sonia Livingstone, Leslie Haddon, Anke Görzig, and Kjartan Ólafsson. 2011. Risks and safety on the internet: the perspective of European children: full findings and policy implications from the EU Kids Online survey of 9-16 year olds and their parents in 25 countries. (2011).
[60] Matthew B MacLaurin. 2011. The design of Kodu: A tiny visual programming language for children on the Xbox 360. In *Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 241–246.
[61] Jane Margolis. 2010. *Stuck in the shallow end: Education, race, and computing*. MIT press.
[62] Ana Martins and Lia Oliveira. 2018. Educational video game design by 8th graders: Investigating processes and outcomes. In *European Conference on Games Based Learning*. Academic Conferences International Limited, 379–387.
[63] Cecile Meier, Jose Saorín, Alejandro Bonnet de León, and Alberto Guerrero Cobos. 2020. Using the Roblox Video Game Engine for Creating Virtual tours and Learning about the Sculptural Heritage. *International Journal of Emerging Technologies in Learning (iJET)* 15, 20 (2020), 268–280.
[64] Rebecca Michelson, Akeiylah DeWitt, Ria Nagar, Alexis Hiniker, Jason Yip, Sean A Munson, and Julie A Kientz. 2021. Parenting in a Pandemic: Juggling Multiple Roles and Managing Technology Use in Family Life During COVID-19 in the United States. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–39.
[65] Matthew B Miles and A Michael Huberman. 1984. Drawing valid meaning from qualitative data: Toward a shared craft. *Educational researcher* 13, 5 (1984), 20–30.
[66] Dana L Mitra. 2006. Youth as a bridge between home and school: Comparing student voice and parent involvement as strategies for change. *Education and Urban Society* 38, 4 (2006), 455–480.
[67] Luis C Moll and James B Greenberg. 1990. Creating zones of possibilities: Combining social contexts for instruction. *Vygotsky and education: Instructional implications and applications of sociohistorical psychology* (1990), 319–348.
[68] Geoff Musick, Guo Freeman, and Nathan J McNeese. 2021. Gaming as Family Time: Digital Game Co-play in Modern Parent-Child Relationships. *Proceedings of the ACM on Human-Computer Interaction* 5, CHI PLAY (2021), 1–25.
[69] Laura M Padilla-Walker, Sarah M Coyne, and Ashley M Fraser. 2012. Getting a high-speed family connection: Associations between family media use and family connection. *Family Relations* 61, 3 (2012), 426–440.
[70] J F Pane and B A Myers. 2001. Studying the language and structure in non-programmers' solutions to programming problems. *Int. J. Hum. Comput. Stud.* (2001).
[71] Michael Quinn Patton. 1990. *Qualitative evaluation and research methods*. SAGE Publications, inc.
[72] Corrina Perrone and Alexander Repenning. 1998. Graphical rewrite rule analogies: avoiding the inherit or copy and paste reuse dilemma. In *Proceedings. 1998 IEEE Symposium on Visual Languages (Cat. No. 98TB100254)*. IEEE, 40–46.
[73] Nenad Petrovic and Milorad Tosic. 2020. Ontology-Driven Generation of Interactive 3D Worlds. In *2020 19th International Symposium INFOTEH-JAHORINA (INFOTEH)*. IEEE, 1–5.
[74] Alex Repenning. 1993. Agentsheets: a tool for building domain-oriented visual programming environments. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. 142–143.
[75] Alexander Repenning. 2000. AgentSheets®: An interactive simulation environment with end-user programmable agents. *Interaction* (2000).
[76] Alexander Repenning. 2017. Moving Beyond Syntax: Lessons from 20 Years of Blocks Programing in AgentSheets. *J. Vis. Lang. Sentient Syst.* 3, 1 (2017), 68–91.
[77] Alexander Repenning and James Ambach. 1996. Tactile programming: A unified manipulation paradigm supporting program comprehension, composition and sharing. In *Proceedings 1996 IEEE Symposium on Visual Languages*. IEEE, 102–109.
[78] 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* 52, 11 (2009), 60–67.
[79] Barbara Rogoff, Behnosh Najafi, and Rebeca Mejía-Arauz. 2014. Constellations of cultural practices across generations: Indigenous American heritage and learning by observing and pitching in. *Human Development* 57, 2-3 (2014), 82–95.
[80] Ricarose Roque, Sayamindu Dasgupta, and Sasha Costanza-Chock. 2016. Children's civic engagement in the scratch online community. *Social Sciences* 5, 4 (2016), 55.
[81] Ricarose Roque, Karina Lin, and Richard Liuzzi. 2015. Engaging parents as creative learning partners in computing. *Exploring the Material Conditions of Learning* 2 (2015), 687–688.

[82] Ricarose Roque, Karina Lin, and Richard Liuzzi. 2016. "I'm Not Just a Mom": Parents Developing Multiple Roles in Creative Computing. Singapore: International Society of the Learning Sciences.

[83] Sandra D Simpkins, Pamela E Davis-Kean, and Jacquelynne S Eccles. 2005. Parents' socializing behavior and children's participation in math, science, and computer out-of-school activities. *Applied Developmental Science* 9, 1 (2005), 14–30.

[84] David Canfield Smith, Allen Cypher, and Larry Tesler. 2000. Programming by example: novice programming comes of age. *Commun. ACM* 43, 3 (2000), 75–81.

[85] Elliot M Soloway and Beverly Woolf. 1980. Problems, plans, and programs. *ACM SIGCSE Bulletin* 12, 1 (1980), 16–24.

[86] Kurt Squire and Henry Jenkins. 2003. Harnessing the power of games in education. *Insight* 3, 1 (2003), 5–33.

[87] Statista. 2021. Digital Media Report - Video Games. https://www.statista.com/study/39310/video-games/. (Accessed on 01/13/2022).

[88] Constance A Steinkuehler. 2006. Why game (culture) studies now? *Games and culture* 1, 1 (2006), 97–102.

[89] Reed Stevens and L. Takeuchi. 2011. *The New Coviewing: Designing for Learning through Joint Media Engagement.* The Joan Ganz Cooney Center.

[90] Kathryn T. Stolee and Teale Fristoe. 2011. Expressing computer science concepts through Kodu game lab. In *42nd ACM Technical Symposium on Computer science Education (SIGCSE)*. 99–104.

[91] Lihui Sun, Linlin Hu, and Danhua Zhou. 2021. Improving 7th-Graders' Computational Thinking Skills Through Unplugged Programming Activities: A Study on the Influence of Multiple Factors. *Thinking Skills and Creativity* (2021), 100926.

[92] Lori Takeuchi, Reed Stevens, et al. 2011. The new coviewing: Designing for learning through joint media engagement. In *New York, NY: The Joan Ganz Cooney Center at Sesame Workshop.*

[93] Giovanni Maria Troiano, Qinyu Chen, Ángela Vargas Alba, Gregorio Robles, Gillian Smith, Michael Cassidy, Eli Tucker-Raymond, Gillian Puttick, and Casper Harteveld. 2020. Exploring How Game Genre in Student-Designed Games Influences Computational Thinking Development. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–17.

[94] Giovanni Maria Troiano, Qinyu Chen, Ángela Vargas Alba, Gregorio Robles, Gillian Smith, Michael Cassidy, Eli Tucker-Raymond, Gillian Puttick, and Casper Harteveld. 2020. Exploring How Game Genre in Student-Designed Games Influences Computational Thinking Development. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–17.

[95] Veena Vasudevan, Yasmin Kafai, and Lei Yang. 2015. Make, wear, play: remix designs of wearable controllers for scratch games by middle school youth. In *Proceedings of the 14th international conference on interaction design and children*. 339–342.

[96] Lev Vygotsky. 1978. Interaction between learning and development. *Readings on the development of children* 23, 3 (1978), 34–41.

[97] Eliane S Wiese and Marcia C Linn. 2021. "It Must Include Rules" Middle School Students' Computational Thinking with Computer Models in Science. *ACM Transactions on Computer-Human Interaction (TOCHI)* 28, 2 (2021), 1–41.

[98] Benjamin Xie, Dastyni Loksa, Greg L Nelson, Matthew J Davidson, Dongsheng Dong, Harrison Kwik, Alex Hui Tan, Leanne Hwa, Min Li, and Amy J Ko. 2019. A theory of instruction for introductory programming skills. *Computer Science Education* 29, 2-3 (2019), 205–253.

[99] Seungwon Yang, Carlotta Domeniconi, Matt Revelle, Mack Sweeney, Ben U Gelman, Chris Beckley, and Aditya Johri. 2015. Uncovering trajectories of informal learning in large online communities of creators. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*. 131–140.

[100] Junnan Yu, Chenke Bai, and Ricarose Roque. 2020. Considering Parents in Coding Kit Design: Understanding Parents' Perspectives and Roles. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. ACM, Honolulu HI USA, 1–14. https://doi.org/10.1145/3313831.3376130