

Social Addictive Gameful Engineering (SAGE):
A Game-based Learning and Assessment System for Computational Thinking

Jeff Bender

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Engineering Science
in the Fu Foundation School of Engineering
and Applied Science

COLUMBIA UNIVERSITY

2023

© 2023

Jeff Bender

All Rights Reserved

Abstract

Social Addictive Gameful Engineering (SAGE):

A Game-based Learning and Assessment System for Computational Thinking

Jeff Bender

At an unrivaled and enduring pace, computing has transformed the world, resulting in demand for a universal fourth foundation beyond reading, writing, and arithmetic: computational thinking (CT). Despite increasingly widespread acceptance of CT as a crucial competency for all, transforming education systems accordingly has proven complex. The principal hypothesis of this thesis is that we can improve the efficiency and efficacy of teaching and learning CT by building gameful learning and assessment systems on top of block-based programming environments. Additionally, we believe this can be accomplished at scale and cost conducive to accelerating CT dissemination for all.

After introducing the requirements, approach, and architecture, we present a solution named *Gameful Direct Instruction*. This involves embedding Parsons Programming Puzzles (PPPs) in Scratch, which is a block-based programming environment currently used prevalently in grades 6-8. PPPs encourage students to practice CT by assembling into correct order sets of mixed-up blocks that comprise samples of well-written code which focus on individual concepts. The structure provided by PPPs enable instructors to design games that steer learner attention

toward targeted learning goals through puzzle-solving play. Learners receive continuous automated feedback as they attempt to arrange programming constructs in correct order, leading to more efficient comprehension of core CT concepts than they might otherwise attain through less structured Scratch assignments. We measure this efficiency first via a pilot study conducted after the initial integration of PPPs with Scratch, and second after the addition of scaffolding enhancements in a study involving a larger adult general population.

We complement *Gameful Direct Instruction* with a solution named *Gameful Constructionism*. This involves integrating with Scratch implicit assessment functionality that facilitates constructionist video game (CVG) design and play. CVGs enable learner to explore CT using construction tools sufficiently expressive for personally meaningful gameplay. Instructors are enabled to guide learning by defining game objectives useful for implicit assessment, while affording learners the opportunity to take ownership of the experience and progress through the sequence of interest and motivation toward sustained engagement. When strategically arranged within a learning progression after PPP gameplay produces evidence of efficient comprehension, CVGs amplify the impact of direct instruction by providing the sculpted context in which learners can apply CT concepts more freely, thereby broadening and deepening understanding, and improving learning efficacy. We measure this efficacy in a study of the general adult population.

Since these approaches leverage low fidelity yet motivating gameful techniques, they facilitate the development of learning content at scale and cost supportive of widespread CT uptake. We conclude this thesis with a glance at future work that anticipates further progress in scalability via a solution named *Gameful Intelligent Tutoring*. This involves augmenting Scratch with Intelligent Tutoring System (ITS) functionality that offers across-activity next-game

recommendations, and within-activity just-in-time and on-demand hints. Since these data-driven methods operate without requiring knowledge engineering for each game designed, the instructor can evolve her role from one focused on knowledge transfer to one centered on supporting learning through the design of educational experiences, and we can accelerate the dissemination of CT at scale and reasonable cost while also advancing toward continuously differentiated instruction for each learner.

Table of Contents

List of Charts, Graphs, Illustrations	vii
List of Tables	viii
Acknowledgments.....	ix
Preface.....	1
Chapter 1: Introduction	2
1.1 Contributions.....	2
1.2 Background	3
1.3 Problem Statement	6
1.4 Requirements and Principles.....	7
1.4.1 System Requirements	7
1.4.2 Pedagogical Requirements	8
1.4.3 Principles	12
1.5 Approach.....	12
1.5.1 Gameful Direct Instruction.....	13
1.5.2 Gameful Constructionism.....	15
1.6 Limitations	16
1.6.1 CT Learning Platform.....	16
1.6.2 Game-based Learning.....	18
1.6.3 Threats to Validity	20
1.7 Architecture.....	21

1.7.1 Methodology.....	21
1.7.2 Concrete Architecture.....	23
1.7.3 Reference Architecture.....	26
1.7.4 Industry Architecture Comparison	27
1.8 Hypotheses.....	28
Chapter 2: Integrating Parsons Puzzles with Scratch	30
2.1 Motivation.....	30
2.2 Background.....	33
2.3 Implementation	35
2.4 Study Purpose	40
2.5 Formative Evaluation.....	41
2.5.1 Participants	41
2.5.2 CT Concept Engagement.....	42
2.6 Summative Evaluation	44
2.6.1 Study Design	44
2.6.2 Materials	48
2.6.3 Participants	49
2.7 Experimental Results	50
2.7.1 Data Collection & Processing	50
2.7.2 Cognitive Load.....	51
2.7.3 Performance.....	52
2.7.4 Efficiency	53

2.7.5 Motivation	54
2.7.6 Findings Summary.....	57
2.8 Conclusions.....	58
Chapter 3: Learning Computational Thinking Efficiently	60
3.1 Motivation.....	60
3.2 Related Work	62
3.2.1 Presentation	62
3.2.2 Distractors.....	63
3.2.3 Feedback.....	64
3.3 Implementation	66
3.4 Study Purpose	68
3.5 Formative Evaluation.....	69
3.6 Summative Evaluation	70
3.6.1 Study Design	70
3.6.2 Materials & Participants.....	72
3.6.3 Data Collection & Processing	72
3.7 Experimental Results	73
3.7.1 Cognitive Load.....	73
3.7.2 Performance.....	75
3.7.3 Efficiency	77
3.7.4 Motivation	79
3.7.5 Findings Summary.....	84

3.8 Conclusions.....	84
Chapter 4: Learning Computational Thinking Effectively	86
4.1 Motivation.....	86
4.2 Related Work	88
4.3 Implementation	90
4.4 Summative Evaluation	92
4.5 Experimental Results	93
4.5.1 Study Design	93
4.5.2 Participants	95
4.5.3 Data Collection & Processing	96
4.6 Analysis & Results.....	96
4.6.1 Cognitive Load.....	96
4.6.2 Performance.....	97
4.6.3 Efficiency	98
4.6.4 Motivation	99
4.6.5 Findings Summary.....	102
4.7 Conclusions.....	103
Chapter 5: Future Work	105
5.1 Gameful Intelligent Tutoring	105
5.1.1 Intelligent Tutoring Outer Loop.....	106
5.1.2 Intelligent Tutoring Inner Loop.....	108
5.2 Sustainable Interoperation	113

5.3 Educative Curriculum Materials	116
Chapter 6: Conclusions	118
6.1 Contributions.....	118
6.2 Findings Summary	119
6.3 Conclusion	121
References.....	122
Appendix A: Feasibility Study Materials	148
A.1 Teaching and Learning with Scratch: Survey	148
A.2 Teaching and Learning with Scratch: Semi-Structured Interview Protocol	156
A.3 Teaching and Learning with Scratch: Design Thinking Workshop.....	158
Appendix B: Field Study Protocol Materials.....	163
B.1 Example Study Guide.....	163
B.2 Learning System and CT Concept Video Tutorials	165
B.3 Demographics & Programming Behavior.....	165
B.3 Pretests and Posttests.....	174
B.4 Self-reported Cognitive Load.....	189
B.5 Programming Behavior, Motivation, and CT Perception	190
Appendix C: Objective Editor	198
C.1 Grammar.....	198
C.2 Terminal Details.....	199
Appendix D: Architecture Detail	205
Appendix E: Supplemental Related Work.....	207

E.1 Computational Thinking.....	207
E.2 Game-based Learning.....	209
E.3 Implicit Assessment.....	210

List of Charts, Graphs, Illustrations

Figure 1.1: Scratch active users and age distribution.	18
Figure 1.2: Sage concrete architecture.....	23
Figure 1.3: SAGE-RA: conceptual reference architecture.	26
Figure 1.4: Mapping SAGE-RA to Code.org®.	28
Figure 2.1: PPP palette and block configuration in design mode in Scratch.....	37
Figure 2.2: Puzzle play and assessment functionality integrated via PPP in Scratch.....	38
Figure 2.3: How a small sample of U.S. teachers perceive student engagement in CT concepts.	42
Figure 2.4: Summative evaluation protocol.....	44
Figure 2.5: Example <i>sequences</i> PPP solution and instructions.	48
Figure 2.6: Participant self-reported programming experience at the start of the study.....	49
Figure 2.7: Instructional efficiency (E) for each of the three conditions.....	54
Figure 3.1: PPP with selectively included palettes and blocks.....	65
Figure 3.2: Palette and block selective inclusion system.....	67
Figure 4.1: Examples of objective configuration and puzzle play experience.	92
Figure 5.1: Game recommendation process.....	107
Figure 5.2: Intelligent hinting data flow.	110
Figure 5.3: Hints delivered through palette highlighting and block shaking.....	112
Figure 5.4: Scratch Analyzer components and data flow.	114
Figure 5.5: Sample Scratch Extractor output.....	114
Figure 5.6: Hairball assessment results for two games.....	116

List of Tables

Table 2.1: Study protocol and measurements.	46
Table 2.2: Training and participant characteristics and differences across three study conditions.	47
Table 2.3: Within-subject attitude change. Positive shifts (p), negative shifts (n). *p<0.05, **p<0.01	56
Table 2.4: General summary of findings across conditions.....	58
Table 3.1: Study protocol and data collected.....	71
Table 3.2: Variation and participation across nine study conditions.....	71
Table 3.3: Selected within-subject programming attitude change. Positive shifts (p), negative shifts (n). *p<.05, **p<.01	80
Table 3.4: PPP element variation findings summary. *p<.05, **p<.01	84
Table 4.1: PPP scaffolding variation across 12 study conditions.	94
Table 4.2: Study protocol and data collected.....	95
Table 4.3: Within-subject attitude and CT perception change. *p<.05, **p<.01	101
Table 4.4: Summary of key findings.....	103
Table 6.1: Summary of findings across conditions in study described in Chapter 2.....	120
Table 6.2: PPP element variation findings summary in study described in Chapter 3. *p<.05, **p<.01	120
Table 6.3: Summary of key findings in study described in Chapter 4.....	120

Acknowledgments

Thank you to my adviser, Gail Kaiser, and family/friends for the support and patience.

This research was funded in part by NSF CCF-1161079, CCF-1302269, CNS-1563555, CNS-1842456 and CCF-1815494, and in part by DARPA / NIWC Pacific N66001-21-C-4018.

Preface

Computational thinking will influence everyone in every field of endeavor. This vision poses a new educational challenge for our society, especially for our children. – Jeannette Wing

When you have played a video game for a while, something magical happens to the texts associated with it. All of a sudden they seem lucid and clear and readable. – James Gee

[I]t would appear that both novice-friendly IDEs and educational games alike leave something to be desired from the standpoint of sparking lifelong passions for programming. The gamification of programming environments like Alice or Scratch might be one solution.

– Sarah Esper

We shouldn't "teacher-proof" curricula so that it can be tested; instead, we should create compelling materials that address teachers' needs and inspire them to teach creatively and effectively. – Kurt Squire

Chapter 1: Introduction

At an unrivaled and enduring pace, computing has transformed the world, resulting in demand for a universal fourth foundation beyond reading, writing, and arithmetic: computational thinking (CT) [1]. Through a deliberate weaving of gameful learning and assessment techniques derived from interdisciplinary research communities, opportunities arise to empower learners with CT proficiency. In this thesis, with an aim to help advance CT learning for all, we explore the opportunity for novice learners in school and beyond, and evaluate the efficiency and effectiveness of a novel learning and assessment system in the adult general population.

1.1 Contributions

We present two solutions, *Gameful Direct Instruction* and *Gameful Constructionism*. The first solution involves embedding Parsons Programming Puzzles (PPPs) in Scratch, which is a block-based programming environment currently used prevalently in grades 6-8 [2,3]. PPPs encourage students to practice CT by assembling into correct order sets of mixed-up blocks that comprise samples of well-written code which focus on individual concepts. The structure provided by PPPs enable instructors to design games that steer learner attention toward targeted learning goals through puzzle-solving play. The second solution involves integrating with Scratch implicit assessment functionality that facilitates constructionist video game design and play. CVGs enable learner to explore CT using construction tools sufficiently expressive for personally meaningful gameplay [4]. Instructors are enabled to guide learning by defining game objectives useful for implicit assessment, while affording learners the opportunity to take ownership of the experience and progress through the sequence of interest and motivation toward sustained engagement.

Our findings suggest:

1. Training with PPPs can induce lower extraneous cognitive load compared to training with PPPs with distractors, or via coding with limited constraint and feedback. Furthermore, PPPs with feedback and without distractors can induce lower overall cognitive load. Educators desiring to create periods of lower intensity in a learning progression can leverage consistent evidence across studies indicating PPPs with feedback offer cognitive load relief without sacrificing learning performance.
2. Training with PPPs both with and without distractors induce higher learning efficiency than does coding with limited constraint and feedback. Furthermore, evidence suggests PPPs with an isolated block palette and without distractors can lead to the highest learning efficiency. By fading correctness feedback during a learning progression, educators can provide an efficient transition path toward less-structured, open-ended learning.
3. Training via PPPs or PPPs with distractors increases learning motivation when compared to coding with limited constraint and feedback. Furthermore, PPPs with feedback induce higher motivation than those without. Educators can sustain high motivation in learning progressions by fading feedback and block presentation scaffolding, which incrementally affords learners increased agency.

1.2 Background

Researchers began studying how learners learn and experience programming in the late 1960s [5]. This was driven in part by industry's need both for programmers and for aptitude tests that could validate capabilities, as well as an initiative to understand the broader benefits of programming to the learner [6]. This latter initiative sparked substantial debate, as claims of

improved problem-solving ability and other thinking skills associated with the Logo introductory programming language [5], established in small-scale studies, were later questioned in the context of contradictory results produced in larger, longer-term studies in educational settings [7,8]. However, [9] later highlighted Logo's potential should proper pedagogy accompany learning, as learning transfer did occur when the transferable skill (e.g. debugging) was taught explicitly as part of the curriculum, instead of being expected to be learned as a side effect of experiencing Logo programming.

This debate unfolded during an era in which computer costs outclassed educational budgets, leading to slow adoption of computing education through the early 1990s. With the decrease in technology costs and increase in focus on science, technology, engineering, and mathematics (STEM) education since the mid-2000s, the demand for a technologically literate workforce and engaged citizenship has tremendously increased [10]. Though discussion of similar terms such as algorithmic thinking began in the 1960's, CT entered the computer science (CS) lexicon popularly in 2006 via a mobilizing ACM essay by Jeannette Wing [1]. Although a consensus definition remains elusive, CT describes how humans, not computers, think logically, algorithmically, procedurally, analytically, recursively, and creatively. This hybrid thinking helps to devise models and representations which enable problem solving, data transformation, and system automation. The primacy of conceptualization, not programming, of ideas, not artifacts, frees CS fundamentals from their constituent levees, allowing strong mental models to flood diverse disciplines both in the sciences and humanities, as well as in their applied counterparts, medicine, law, and journalism. Of course, computers retain their central role: "Abstractions are the 'mental' tools of computing. The power of our 'mental' tools is amplified by the power of our 'metal' tools. Computing is the automation of our abstractions" [11]. The

emphasis on semantics rather than syntax, however, engenders a common language in which to explore the possibilities of computing for all.

Despite increasingly widespread acceptance of CT as a crucial competency for the general populace, transforming education systems accordingly has proven complex. In response to a crisis in CS teacher certification and a deficit of K-12 student exposure to CS [12,13], governments are enacting policies that require CT in schools [14,15,16]. Due to the actuality that teachers and students frequently must learn CT simultaneously, however, innovative means of delivering this learning must be devised to affirm equality of access.

Some of the research fields most relevant for fostering this innovation are the learning sciences (LS), game-based learning (GBL), educational data mining (EDM), learning analytics (LA), and intelligent tutoring systems (ITS). LS blends cognitive science with CS and educational psychology, among others, with an interventionist aim to design and implement improved instructional environments. GBL situates defined learning goals in inviting virtual environments that encourage exploration and practice with principles in manners suitable for transfer to real life. EDM entails the automatic extraction of insight from data generated by learning activity, often with the goals of predicting future learning and analyzing the efficacy of educational support. LA, while similar to EDM in underlying technique, more commonly involves humans such as teachers and administrators in the analytic cycle. ITS research produces systems that deliver differentiated instruction to each student, using a variety of feedback mechanisms that aim to approximate the experience of one-to-one personalized tutoring for all.

Our approach harmonizes these fields for the purpose of teaching and learning CT by enriching existing block-based programming environments. Leveraging LS theory to underpin the pedagogical strategy, we introduce GBL tooling in a system named Social Addictive

Gameful Engineering (SAGE) that fosters student learning through gameplay and teacher learning through game design. The process and product data exhausted feeds LA instrumentation that implicitly assesses progress to provide learners real-time formative feedback that coaches toward CT mastery.

1.3 Problem Statement

Despite the lack of computing's uptake in schools in the 1980's and 1990's after initial enthusiasm, many now believe CT in K-12 is an idea whose time has come. Educators are requiring that all students study computing [17], attractive novice learning environments such as Scratch, Blockly, Alice, and many others have risen to prominence [18], informal learning organizations like Code.org, Girls Who Code, and Black Girls Code are available to many learners including those underserved by schools [19,20,21], and the Computer Science Teachers Association has both launched and established learning standards [22]. However, the newness of the discipline leads to deficits in preparedness.

To transform the idea of CT in K-12 into an implementable educational imperative, the CS community must equip schools, teachers, and students with novice learning environments that sustain engagement and promote durable learning. Although CS is inherently a dynamic field shaped by rapid technological change, its practitioners can contribute cogent definitions, economical authoring tools, efficient means of assessment, and age-appropriate pedagogical examples [23,24]. Schools need these resources to pervasively plug-in CT within curricula at all levels [25], and to establish professional development regiments for their teachers [26].

The need for teachers to learn alongside their students arises because state requirements for CS teacher certification remain disorganized, deeply flawed, and inconsistent in their formulation of CS [12]. Teachers need resources that demonstrate how to integrate CT concepts

with the content and pedagogical knowledge they use to teach their individual subjects. The CS community can contribute sensible orderings of concepts that sculpt developmentally appropriate CT learning progressions, demonstrate how these concepts operate across subjects, and influence the structure of systemic and sustainable CT platforms which embed ecosystems of accessible training materials [27,28]. Teachers need these resources to become computational thinkers capable of designing CT learning experiences [29].

Most importantly, students need to become captivantly immersed in CT. The learning system should not focus on programming at the expense of broader CT conceptualizations, but instead should invite students to develop a sense of agency as they innovate by imagining, playing, problem-solving, creating, sharing, and reflecting [30,31]. Gameful guided discovery can balance structure with agency to focus learning activity and provide students with stronger capability to explore design options than would be realizable by tinkering alone. Ultimately, integrating CT in classrooms involves fostering a computational culture, which requires adequate resources, pedagogy, and systems. The problem is that these school, teacher, and student needs have not been fully satisfied by existing solutions.

1.4 Requirements and Principles

To disseminate CT widely, a learning system must simultaneously address the needs of these diverse stakeholders. During formative evaluations and a literature review, discussed in Chapters 2, 3, and **Error! Reference source not found.**, we elicited and coalesced the following system requirements.

1.4.1 System Requirements

- 1 **Structured Play:** Focusing learner activity on achieving objectives which are satisfied by discovering CT concepts, practices, and perspectives, such as those outlined in [32], will

allow for efficient and effective teaching and learning. The solution should support game formats that promote both direct instruction and guided discovery.

- 2 **Implicit Assessment:** Assessing CT proficiency during gameplay informs learners, instructors, and the learning system, of progress in real-time. The solution should reduce grading burdens while also enabling quicker remediation of misconceptions and quicker advancement for learners who demonstrate mastery.
- 3 **Differentiated Instruction:** Each learner should receive instruction tailored to their level of mastery, with individualized coaching both across and within activities, in alignment with the ITS tradition of personalized feedback. As mastery develops, opportunities should increase for learners to influence their learning paths in alignment with the constructionist tradition of constructing personally meaningful artifacts.
- 4 **Learning Progressions:** Configurable and dynamic learning progressions aligned with customizable curricula allows for the inclusion of a variety of emerging CT standards [22,33].
- 5 **Economical Authoring:** Designing games and assessments must be economical and achievable for educational content developers. The solution should provide instructors coaching and CT learning opportunities throughout the design process.

1.4.2 Pedagogical Requirements

- 1 **Social Requirements:** Social learning aligns with the socio-cultural perspective in which individuals collectively construct knowledge through joint activities as they participate in cultural practices that shape cognition [34]. Social interaction within the learning environment thus emerges not only as a motivator for engaging with educational content, but also as a vehicle in which to enact collaborative skills consistent with CT practices [35].

The theories of social gameflow, computer supported collaborative learning (CSCL), and cognitive apprenticeship might best support social learning in a GBL system. Social gameflow involves instilling in players perceptions of interdependent goals and rewards, but also hinges upon a conception of games inseparable from contexts in which social relations and surrounding environments necessarily impact in-game and transferable learning [36]. CSCL emphasizes that learning occurs socially during shared knowledge construction and group cognition when reinforced by adaptive media which prompts, analyzes, and selectively responds [37]. Cognitive apprenticeship leverages more knowledgeable others or agents to illustrate the power of particular techniques applied in diverse settings, thereby bringing cognitive process into the open for observation and adaptive practice in which task complexity modulates so that component skills optimally integrate [38,39].

- 2 **Addictive Requirements:** In the video game industry, the term addictive refers to engaged, repeated play, and consonant rather than obsessive passion [40]; the term does not imply clinical addiction. Csíkszentmihályi's theory of flow might best support addictive learning in a GBL system. Characteristics of flow include: a clear sense of required action; intense concentration; continuous feedback; a sense of control; a capacity to act to overcome challenges; an accelerated distortion of time; and a sense of intrinsic reward [41]. While flow helps learners practice CT, studies indicate that fully immersive flow might inhibit metacognition, and therefore immersive flow provides better fit when aiming to proceduralize knowledge than when trying to acquire or reflect on it [42]. Furthermore, since individuals cannot infinitely sustain flow, addictive learning must extend beyond flow, and capture longer-term learner interest, motivation, and engagement [43]. Interest

describes a predisposition to reengage with content over time. A complex blend of environment, cognition, and affect comprise motivation. Engagement measures a learner's connection to the socio-emotional and cognitive aspects of the learning environment as evidenced by initiation of action, effort, and persistence in academic tasks. When utilized in a unified manner, these theories lead to the design of games that addict players to learning without presenting high-fidelity graphics and soundscapes; production values need not match commercial quality to challenge learners adaptively to the cusp of their capabilities [44].

- 3 **Gameful Requirements:** When CT concepts, practices, and perspectives meld within game contexts, the learning becomes gameful. Instead of overlaying the game cycle, CT content blends throughout the environment as an intrinsic component of gameplay [45]. Experiential learning theory emphasizes experiencing problem-solving strategies prior to deliberate cognition and reflective behavior during which abstract concepts subsequently develop [46]. Similarly, learning in activity affords groups opportunities to work with authentic objects and technological activity systems that enable close inspections before expanding toward broader goals and generalizations [47]. This active learning is supported by scaffolding, which provides structure by embedding guidance in context so that learners can perform expert-like tasks and develop understandings of CT and the relationship between its objectives and procedures [48]. Additionally, the theories of embodiment, goal-based scenarios, and story-centered curricula also support gameful learning. Embodiment guides learners from immersive action to structured reflection while making unconscious, tacit knowledge explicit [49]. Goal-based scenarios compartmentalize scaffolding within school-sized slices of class-time by establishing hierarchies of accomplishable missions

which elicit intrinsic motivation [50]. These hierarchies combine to form a story-centered curriculum, “a carefully designed apprenticeship-style learning experience in which the student encounters a planned sequence of real-world situations constructed to motivate the development and application of knowledge and skills in an integrated fashion” [51].

Purposeful narratives, therefore, provide the meaningful and motivating roles for learners as they face adaptive progressions of challenges that stretch their abilities.

- 4 **Engineering Requirements:** Exploring multiple routes while solving problems engenders engineering habits of mind and mental processes that require important aspects of CT [52]. This engineering becomes enjoyable in GBL since, after all, a game is “a problem-solving activity, approached with a playful attitude” [53]. Problem-based learning situates students in real-world contexts so that they can construct principled arguments while collaborating to solve complex, ill-structure problems [54]. This method enhances students’ capacities to transfer knowledge to new problems and achieve more coherent understandings than traditional instruction because it consistently activates prior knowledge and demands cumulative reasoning, theory building, and conceptual change. The theories that might best support an engineering approach when learning CT include problem-based learning (PBL), and a sub-theory of self-determination theory (SDT), cognitive evaluation theory (CET). CET promotes self-motivated and self-determined learning by highlighting the first two psychological nourishments of SDT’s triad: competence, autonomy, and relatedness [55]. Equifinality, manifested in in the form of multiple solution routes, offers students autonomy over their own actions as they achieve tasks within game contexts, resulting in experiences of competence and the sense of self-efficacy crucial for youth momentum in engineering disciplines [56,57,23].

1.4.3 Principles

Inspired by Gee’s approach in [58], we establish the following principles to steer the satisfaction of these requirements.

- 1 **S**ocial: *Nurturing Affinity Space Principle*. SAGE affiliates instructors with learners by facilitating the embedding of pedagogical content knowledge in the learning system, thereby fostering networked expertise.
- 2 **A**ddictive: *Regime of Competence Principle*. SAGE induces flow and cultivates engagement by presenting pleasantly frustrating challenges that stretch the competencies of each individual learner.
- 3 **G**ameful: *Situated Meaning Principle*. SAGE does not assemble CT concepts disconnected from gameplay, but instead envelopes learners in gameful environments in which concept exploration emerges as narratives unfold.
- 4 **E**ngineering: *Explicit Information On-Demand and Just-In-Time Principle*. SAGE explains directly only when learners eagerly seek or desire information and have accrued the experience and motivation imperative to consume that information lucidly.

1.5 Approach

Researchers and organizations have constructed numerous novice learning environments for programming during the last five decades [18]. While most weren’t designed with the aim of teaching CT, various CT curricula have leveraged these tools with moderate success. What has often made them effective is their low floors, high ceilings, and wide walls [59]. Low floors allow easy starts. High ceilings offer opportunities to construct increasingly complex projects over time. Wide walls support a diversity of project types so that students with varying interests and learning styles become engaged. To varying degrees, these environments also assist in

scaffolding learning, promote transfer to non-novice environments, and deliver toolsets by systemically equitable and sustainable mechanisms. However, since the late 1960's, Logo [5] and its numerous counterparts have failed to produce a general populace of computational thinkers. Toy-like programming languages might provide short bursts of enthusiasm, but correct solutions to problems at lower code levels do not necessarily indicate conceptual traces to CT [45]. Learners need a CT environment that catalyzes the transfer of knowledge from one concrete instantiation to another, that fosters abstract formulations over a continuum, and that elicits iterative refinement culminating in complex productions [60].

This thesis offers evidence that a game-based learning and assessment system equipped to present incrementally challenging experiences that maintain cognitive flow can help advance CT learning. We build this system modularly on top of popular existing block-based programming environments to utilize their strengths and amplify their impact. Specifically, we build upon Scratch, as well as Google Blockly [61], a highly extensible block-based toolset that simplifies the configuration of custom grammars. By adhering to the presented principles and leveraging the approaches of *Gameful Direct Instruction* and *Gameful Constructionism*, we satisfy the stated system and pedagogical requirements.

1.5.1 Gameful Direct Instruction

Since the 1960's, researchers have created novice learning environments intended to make programming more accessible [18], and recently, organizations have arisen to further democratize [23], evangelize [62], and capitalize on [63] computing education. For grades 6-8, probably no environment has reached more students than Scratch, the block-based programming environment first released in 2007 [64]. Its approximately one million monthly active users, and over 96 million registered users [65], enjoy an attractive UI that invites exploration, tinkering,

collaboration, and personally meaningful creation, in alignment with the principles of constructionism [66]. However, studies indicate Scratchers infrequently demonstrate skill increases over time [67], seldom reason about program state [68], misconceive loops, variables, and Booleans [69], and often adopt programming habits unaligned with accepted practice in CS [70]. Instead of problem-solving algorithmically, for example, they favor bottom-up tinkering with relevant blocks in a process called bricolage [66]. Without structured external guidance, students often engage in pure-discovery learning, which can lead to underwhelming learning outcomes, because when encountering novel information, students sometimes do not have sufficient prior knowledge to provide themselves with internal guidance that leads to the discovery of new knowledge [71].

By internalizing structure into the block-based programming environment, our *Gameful Direct Instruction* approach aims to enable learners to engage in efficient cycles of experimentation, remediation, and mastery. SAGE offers learners more efficient uptake of core CT concepts as they solve focused puzzles that provide formative guidance toward correct solutions. *Gameful Direct Instruction* extends existing research on PPPs in text-based languages by introducing the format within Scratch. This framework provisions instructors with tools to design puzzles, and provisions learners with an implicit assessment system that coaches toward mastery during gameplay.

Learning efficiency is a measurement regularly used in PPP research in the literature [72,73,74,75,76,77,78]. As discussed in Section 1.3, school, teacher, and student needs are unmet by extant offerings. The addition of CT as a learning outcome into curricula already overflowing with standardized test-driven requirements must be accomplished economically [79]. Consequently, deriving the core CT concept learning outcomes given inexpert teaching within

limited time might be perceived by all stakeholders as advantageous. In Section 2.2 we discuss the calculations used to quantify instructional and performance efficiency. Some of our key findings pertain to instructional efficiency; simply stated, if comparable performance gain is achieved on a transfer task with less training time, the learning is considered more efficient.

1.5.2 Gameful Constructionism

At MIT in the late 1960s, learning theorist and technologist Seymour Papert helped develop the Logo programming language that introduced turtle graphics to a generation coming-of-age alongside personal computers. Extending the constructivist theory of knowing evangelized by developmental psychologist Jean Piaget, Papert introduced constructionism, asserting not only that learning entails an active process in which people construct knowledge from their experiences in the world, as in constructivism, but also that this process proves particularly effective during the construction of personally meaningful artifacts [5,80]. In contrast to the traditional vision of schooling sometimes called instructionism, which involves a single speaker and many listeners who then need to take tests to prove they have listened adeptly [81], constructionism engenders a participatory culture in which learners build relationships between old and new knowledge during interactions with peers and mentors who provide formative feedback and contribute to the creation of socially relevant products suitable for summative assessment. Since CT both complements and combines mathematical and engineering thinking [1], two subjects rooted in the evolution of this learning theory, constructionism provides a valuable lens through which to analyze the needs for a system supportive of effective CT teaching and learning.

By integrating with block-based programming languages implicit assessment functionality, our *Gameful Constructionism* approach aims to enable learners to apply CT

concepts through cycles of guided discovery that broaden and deepen understanding. SAGE offers learners more effective uptake of CT by amplifying the impact of direct instruction through the provision of scaffolding functionality that enables the configuration of goals achievable through gameplay. *Gameful Constructionism* extends existing research on CVGs across subjects and environments by introducing the CVG format in Scratch and Blockly. Our framework provisions instructors with tools to design CT CVGs with objectives, and provisions instructors with formative feedback that guides gameplay through learning progressions that balance structure and agency.

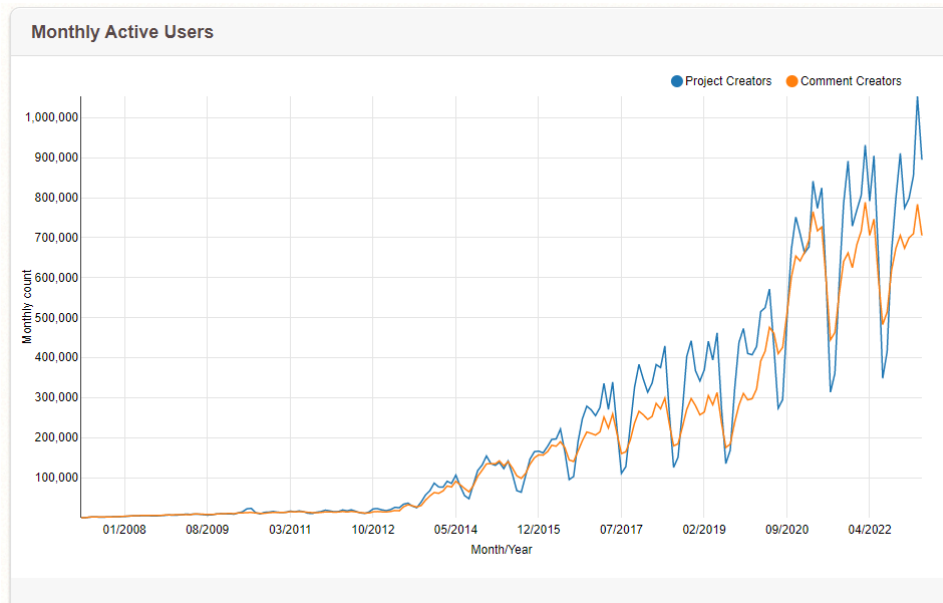
1.6 Limitations

The generalizability of this research is bound by the choice of the CT learning platform, Scratch, as the primary basis for this work, as well as by the PPP game-based learning approach and experimental context.

1.6.1 CT Learning Platform

Upon embarking on the research program, we considered several alternative novice learning environments applicable for CT, including Karel the Robot, StarLogo TNG, Alice, and Snap! [18,82], and later reviewed emergent options such as Microbit and Minecraft Education [83,84]. An analysis of each platform's capability to satisfy and support the system and pedagogical requirements documented in Section 1.4 resulted in the choice of Scratch. For example, Karel, first implemented in the 1970s, focuses more narrowly on teaching programming in a motivating context, with a development environment integrated with a simple simulation consisting of avenues, streets, walls, beepers, and robots. Its cousin platform, Light-Bot, drops the syntax of a specific language and instead leverages programming semantics to direct focus toward problem-solving within a CT framework including processes and

transformations, models and abstractions, patterns and algorithms, and tools and resources [85]. However, Karel and cousins primarily offer low floors for learning, without high ceilings or wide walls, resulting in limits in the breadth and depth of learning possible, and concerns about transference expressed by the developers: the link between the robot's movement and CT concepts might not consistently reach students. Similar evaluations of StarLogo TNG and Alice revealed high floors associated with the complexity of 3D environments as well as reduced presentation support in Alice without block-shape differentiation, resulting in a program-by-dropdown list paradigm. Scratch, with its low floors, wide walls, and high ceilings, as well as a robust community sharing and commenting upon each other's work, best offered a platform upon which we could research advancements in CT learning, especially since our initial research target focused on grade 6-8 students. Students in these grades are among the approximately one million active users most involved with the platform, as depicted in Figure 0.1 [65].



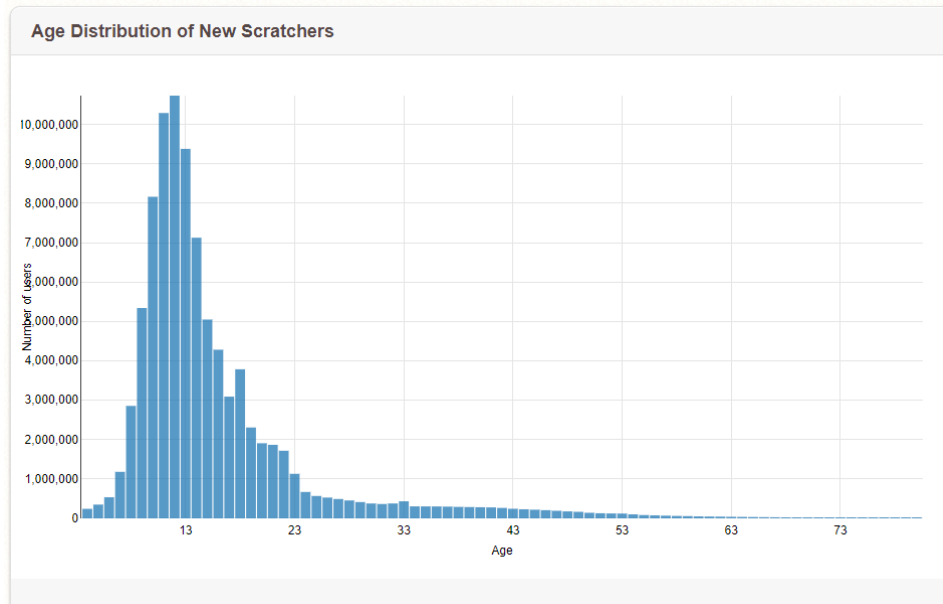


Figure 0.1: Scratch active users and age distribution.

1.6.2 Game-based Learning

To support an active CT learning strategy by emphasizing computational doing [25,86] we opted for a GBL approach, because GBL enticingly affords interactivity: a multi-way exchange with feedback and invitation to probe deeper [87]. CT's versatility as a glue that bridges domain-specific expertise and technical innovation further solicits GBL, since game content can support a broad diversity of subjects within existing curricula while providing students an opportunity to develop and apply CT synergistically [52]. The contextualized representations concretize CT's abstract concepts, resulting in both higher learning gains and increased engagement [88]. Furthermore, the multidisciplinary integration of CT helps students recognize appropriate use cases across domains, thereby instilling a more fluid kind of problem-solving than cultivated through siloed study [52]. By adaptively subdividing big problems into smaller ones solvable with the computational competence of each learner, games inculcate increased levels of self-efficacy. The associated elevated confidence inspires students to practice

communicating with CT within enjoyable social environments designed to encourage both collaborative problem-solving and iterative group reflection that enables not only the digestion of new learning, but also the collective construction of new knowledge [89].

Several literature reviews of the increasing volume of GBL research in recent years demonstrate the effectiveness with which games improve general learning [90,91,92,93,94], increase motivation [95,96,97], and improve performance [98,99,100,101], but close examinations of typical experimental designs reveal limited generalizability with results that apply only to the specific population under study [102]. The conclusions do not assert the efficacy of all games for all learners in all situations, and they reveal the risk of students gaming the system [103] by abusing feedback systems to achieve in-game goals without necessarily achieving the intended learning outcomes. More in-depth research focusing on learning outcomes and learning processes is encouraged in [104], which suggests a saturation point has been reached in comparing traditional education with GBL in the literature. The theoretical foundation from which GBL studies depart nonetheless has progressed enormously since video games' early years when edutainment gained prominence as “a sugarcoating of entertainment for the bitter medicine of education to become palatable” [105]. The interdisciplinary field of the learning sciences, which studies teaching and learning, most significantly has influenced the structure and robustness of GBL research, but several psychology subdomains, such as self-determination theory [106], also imprint modern educational game design. These bedrocks provide sturdy support for research that braids learning theory with ludology, the study of games [107]. To increase the generalizability of experimental results, however, researchers are evolving from the earlier orientation around coarse game instances and instead focus on granular game attributes that impact specific learning outcomes [108]. By determining the linkages between

instructional objectives and game attributes, we can better construct authoritative arguments based on empirical evidence to deepen the GBL knowledge base and advance CT learning. We discuss advantages of leveraging the constraints of the PPP game type for this purpose in Sections 2.1 and 2.2. Since this format facilitates focused and efficient practice on individual CT concepts, and educators can devise them economically, we chose PPPs over alternative game types like the adventure game approach presented with logic puzzle minigames in Zoombinis [109] or the open-ended sandboxes offered in Minecraft Education [84].

1.6.3 Threats to Validity

The work presented in this dissertation is based on these choices of Scratch and PPP as the primary learning environment and game types, and the results should be interpreted in that narrowed context. The results suggest a gaming approach might be useful, however, alternative approaches were not evaluated, and one cannot conclude that a different approach might not be better or that a gaming approach is the best way to teach and learn CT. Computing education might require a multitude of approaches including games, projects, tests, performance assessments, among others, and SAGE might be one system in the mix, rather than “one” scalable, cheap, automated solution.

While we evaluated CT concepts such as *sequences*, *conditionals*, and *loops*, we did not test *modularity*, *abstractions*, *composition/decomposition*, thereby excluding key CT concepts. Our work does not suggest the results extend to CT practices, such as debugging, or CT perspectives, such as the viewpoints formed by learners about the world around them [32]. The focus is on efficient and effective learning of a subset of cognitive CT, in part to better enable future learning of the creative expression and social engagement associated with situated CT, as well as the analytical approach to underlying values associated with critical CT [110]. Our

results are based on field studies involving adults rather than children in part because of the COVID-19 emergency that led to disruptive complexity in engaging teachers and their classes, as well as informal learning instructors and their students; the results need to be interpreted in that context. Furthermore, the evaluations focus primarily on efficiency, cognitive load, and motivation, though these are not all the measures one could have chosen. We did not evaluate alternatives such as “stickiness” in terms of testing whether a concept is retained n days/months/years later, nor did we measure “generalizability” in terms of being able to apply a CT concept in a completely different real-world setting.

1.7 Architecture

In a series of iterative proof-of-concept implementations and feasibility studies, we developed an implementation architecture suitable to advancing the teaching and learning of CT. In the following subsections we summarize the methodology used to document concrete and reference architectures, and then compare the result to an industry standard. This exercise enables explicit identification of fundamental learning system components and their interconnections with one another, while offering a potential basis for the standardization of interaction protocols. We first presented this work in 2019 at the 1st International Conference on Embedding Artificial Intelligence (AI) in Education Policy and Practice for Southeast Asia [111].

1.7.1 Methodology

Our methodology involves the use of two frameworks for the derivation [112] and classification [113] of a new reference architecture. We then validate the architecture against a widely used coding learning environment, Code.org® [62]. We selected Hassan and Holt’s framework because of its demonstrated applicability to derive a reference architecture from an

existing implementation through reverse engineering (this was the authors' approach in deriving a reference architecture for web servers, which was an emergent domain at the time the study was published). Similarly, we specified and validated the SAGE Reference Architecture (SAGE-RA) by abstracting and mapping components of existing concrete implementations. Here we share details of one of the mappings from SAGE-RA to a concrete architecture, SAGE-RA to Code.org[®], by reverse engineering a conceptual architecture for that implementation.

To establish a reference architecture, we first derive a concrete architecture for SAGE based on existing implementations. We then derive a conceptual architecture, based on a generalization of the concrete architecture, and plans for future development. From the conceptual architecture, we proposed a reference architecture, and validate it against another implementation in the domain, Code.org[®].

Code.org[®] is a nonprofit dedicated to expanding access to computer science in schools and increasing participation by women and underrepresented minorities. Code.org[®] provides a leading curriculum for K-12 CS in the largest school districts in the U.S. and also organizes the annual Hour of Code campaign which has engaged more than 15% of all students in the world [21]. We use Code.org[®]'s GitHub repository [114] to map the primary components of that application to components of SAGE-RA.

1.7.2 Concrete Architecture

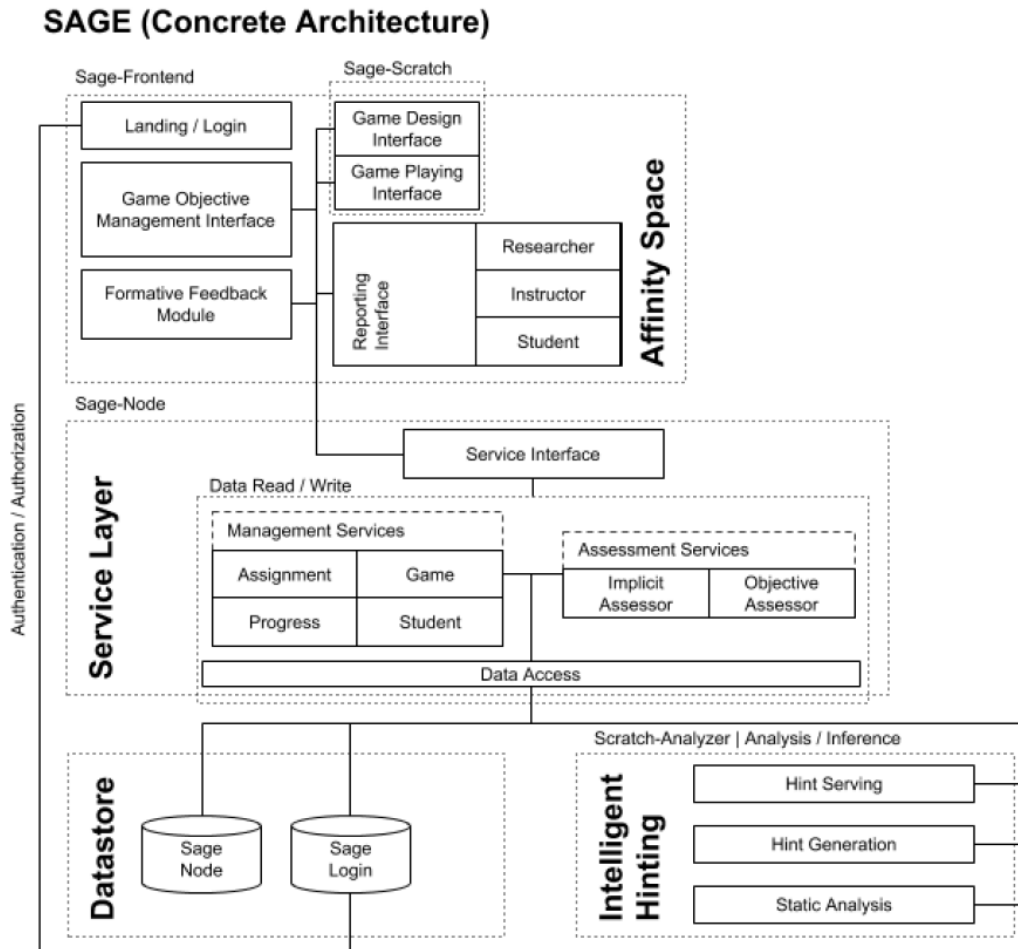


Figure 0.2: Sage concrete architecture.

We next present a short summary of components, groupings, and modules within the SAGE concrete architecture depicted in Figure 0.2. Additional diagrammatic detail is available in Appendix D.

- 1) SAGE Affinity Space: The Affinity Space, primarily through the Sage-Frontend grouping, is the presentation component of SAGE, implemented as a Node.js app using AngularJS, Bootstrap, and jQuery, that handles user management functions such as logins, registrations, and profile management. In addition, the Affinity Space also displays learning path presentations, and provides formative assessment through

the Formative Feedback module. When a student launches a learning session from the Game Objective Management Interface, the session occurs from within the Sage-Scratch grouping. The Affinity Space also supplies context-specific reporting for the three types of users: Researcher, Instructor, and Student. Moreover, from the end user perspective, the Affinity Space also functions as the interactive content distribution and engagement platform for these various types of users to share and consume other users' various types of content, in effect providing a virtual community of learners. Instructors may find the growing content, the archive of interactions, and the feedback-giving and feedback-receiving mechanism available through the Affinity Space useful in providing effective teaching.

- 2) SAGE Service Layer: The Service Layer is the back-end service component that handles and completes various requests from the Affinity Space. The Management Services grouping within this Service Layer processes creation, update, and deletion of assets related to Assignment, Game, Progress, and Student. The Assessment Services grouping executes the implicit assessments of the student's progress in achieving the learning goals, either via teacher-defined built-in rules, or via automated machine-learning generated results received from the Intelligent Hinting component.
- 3) SAGE Intelligent Hinting Layer: The Intelligent Hinting component assesses and generates formative feedback [115]. This includes evaluation of progress in achieving objectives through learning paths [116], which is fed back in the form of in-session messages, suggestions, and hints. As in-game assessment has been shown to improve learning outcomes for novice-programmers [117], a flexible assessment model is

necessary to ensure that a SAGE-RA compliant implementation can use a wide array of feedback and assessment models, in order to keep up with continuing research on efficacy of feedback types in game-based learning. To capture the student interaction data from the Affinity Space presentation layer, the Sage-scratch grouping regularly transmits game snapshots, represented as JSON, to Sage-node in the Service layer to be saved as a field to the database in the Datastore component. Each snapshot contains rich transactional and referential information such as timestamp, game object attributes (unique identifier, type, position, and coordinate), progress markers, and student association. These data are then used within the Scratch-Analyzer grouping where they undergo data wrangling and data cleansing processes, and become the basis for statistical analysis and machine learning processing. State transitions throughout the session are computed by tracking deltas between snapshots.

- 4) SAGE Datastore: The Datastore component functions as the database layer for the whole application, consisting of persistent data collection of users profiles, authentication information, games, quests, assessments, and feedback histories.

1.7.3 Reference Architecture

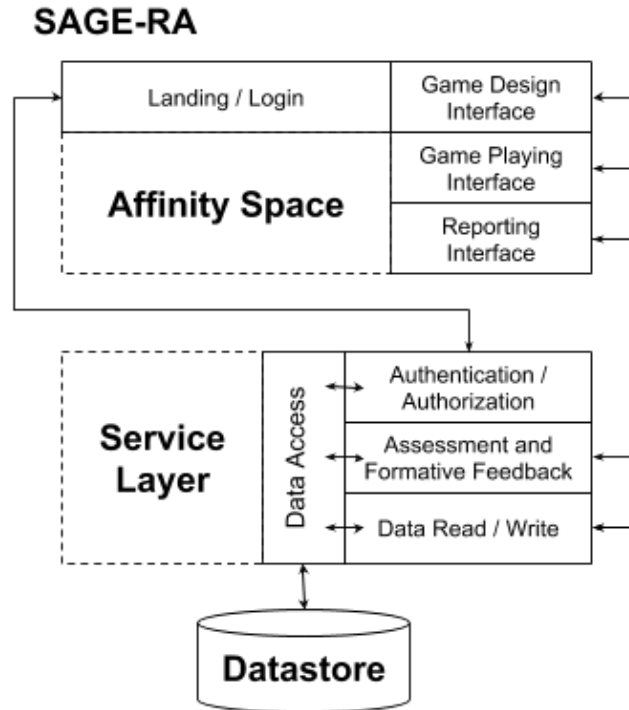


Figure 0.3: SAGE-RA: conceptual reference architecture.

The documentation and refinement of the concrete architecture facilitated the derivation of SAGE-RA conceptual reference architecture, shown in Figure 0.3. Since we aim for SAGE-RA to provide a guidepost for a wide array of concrete learning environment software architectures, we abstract and generalize the following key elements.

- 1) Client Independence: While SAGE-RA derives from several web-based learning environments, it has no explicit networking, client device, or web-specific dependencies. The architecture could be implemented as a standalone application, a scalable distributed learning environment, a classic web application, or in other application models.
- 2) Assessment Flexibility: While SAGE's concrete architecture includes a novel Formative Feedback module to implement feedback based on student interactions,

there is no prescriptive feedback model included in SAGE-RA. SAGE-RA does specify that feedback should be included and be reactive to assessment; the implementation of providing that feedback is up to the implementer. Grade or score-based feedback, hinting, or simple pass/fail feedback all fit within the confines of the architecture, which may allow for its application to systems strictly in the computer-based testing or assessment subdomain, such as certification tests.

- 3) Game Design Flexibility: SAGE-RA does not specify a set of game paradigms that can be designed in compliant systems. The architecture is as applicable to direct instruction [118] as constructionist [119] game design, and could as easily be applied to games that integrate both approaches [120], or to future game-based learning methodologies.

1.7.4 Industry Architecture Comparison

The primary components of Code.org[®]'s concrete architecture, according to an analysis of code in its GitHub repository [114], map clearly to the primary components of SAGE-RA, presented in Figure 0.4. Subcomponents, such as Code.org[®]'s Reporting Interface and Game Design Interface, exist within a Dashboard primary component, which in SAGE-RA would be considered a grouping or subcomponent within the Affinity Space primary component. Code.org[®]'s architecture additionally supports an Extensions component, which could be potentially useful for standardizing a strategy for feature expansion, though a similar component is not included in the SAGE-RA baseline. Though Code.org[®] was developed prior to the existence of a usable reference architecture for gameful CT learning in the literature, it maps well to the general reference architecture presented here. This suggests that such an architecture could be used for future gameful CT learning environment implementations.

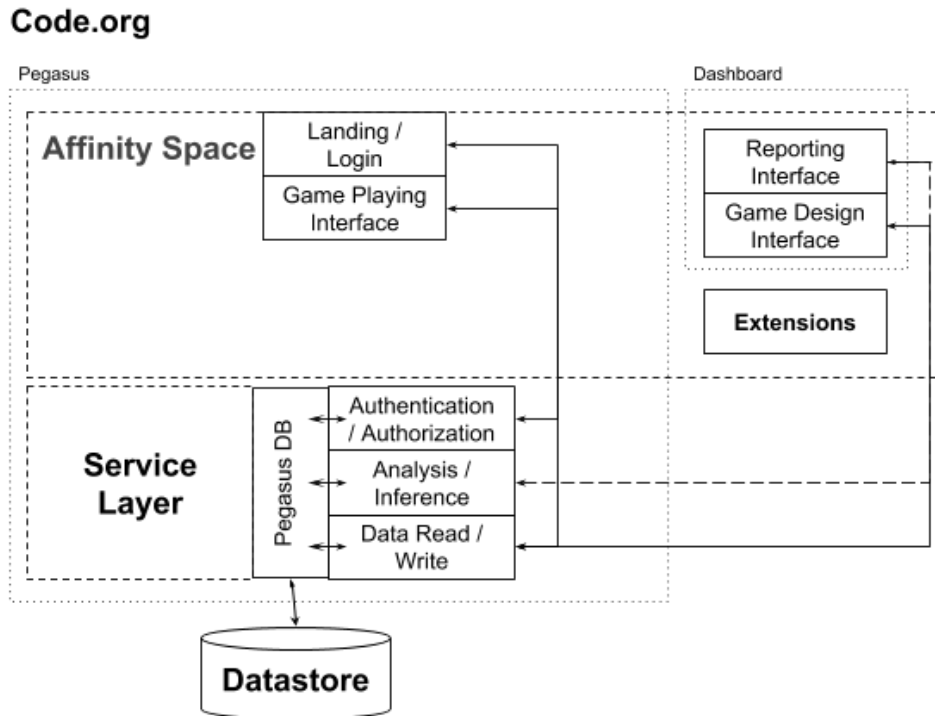


Figure 0.4: Mapping SAGE-RA to Code.org®.

1.8 Hypotheses

The refinement of the concrete architecture and derivation of the reference architecture, in support of the earlier discussed approaches employed to address the elicited system and pedagogical requirements, prepare us to design a series of experimental studies to measure the efficiency and effectiveness of learning CT. We hypothesize we can improve the efficiency and efficacy of teaching and learning CT by building gameful learning and assessment systems on top of block-based programming environments. This can be accomplished at scale and cost conducive to accelerating CT dissemination. To test these hypotheses, we implement and evaluate the following technologies.

- 1) Embedded puzzle authoring and gameplay in Scratch (*Gameful Direct Instruction*)
- 2) A game-objective system that enables implicit assessment (*Gameful Constructionism*)

In the next two chapters, we provide the context for and documentation of formative and summative assessments focused on *Gameful Direct Instruction*. We then present in Chapter 4 an experiment involving *Gameful Constructionism*, before sharing our recommendations for future work and concluding in Chapters 5 and 6.

Chapter 2: Integrating Parsons Puzzles with Scratch

2.1 Motivation

As earlier noted, in response to the crisis in CS teacher certification and limitations in K-12 student exposure to CS, governments are enacting policies requiring CT in schools. Additional argument [1,11] and evidence [121] provide rationale for ensuring children achieve CT competency during the formative cognitive and social development cycles throughout grades K-12. PPPs, which enable learners to practice CT by assembling into correct order sets of mixed-up programming constructs that comprise samples of well-written code focused on individual concepts, are one approach used to introduce CT efficiently [2,74].

These scaffolded program construction tasks facilitate learning the syntax and semantics underlying a CT concept. As the learner solves carefully designed single-solution puzzles, she arranges constructs from a curated assortment in a cycle of deliberate practice that exposes and addresses misconceptions [122,123]. Among the correct code fragments, she might find distractors which provoke cognitive conflict that reinforces learning [124]. The partial suboptimal path distractor type, for example, might tempt a learner toward faulty progress without enabling her to solve the problem fully, thereby triggering recognition of a misconception and productive backtracking toward the correct solution [72]. Researchers have hypothesized distractors might be beneficial in PPPs for reasons similar to those leading to their inclusion in multiple choice tests [2,125], such as the illumination of conceptual misunderstanding, flawed reasoning, or inconsistent reasoning. For example, in a puzzle that requires the execution of one of two alternative sequences, the instructor might include one `if`

block as a distractor, though only the also-included `if-else` block will enable the learner to solve the puzzle correctly.

Research indicates this structured approach to learning CT can lead to more efficient concept learning than alternatives such as learning by tutorial, or writing/fixing code [126,73,127]. To measure efficiency, researchers often leverage cognitive load theory (CLT) [128], which helps to distinguish between the complexity of the material, the instructional design, and the strategies used for knowledge construction. Since PPPs provide constrained problem spaces, they can induce lower cognitive load than that experienced when writing code with open-ended agency. Agency can be defined as a learner's ability to define and pursue learning goals [129]; its removal via structure in the form of rules and well-defined resources such as confined learning progressions limits learners in self-determining what they care about and how they will achieve their goals. Kirschner, Sweller, and Clark have well-argued the downsides of pure discovery learning and advocated for strong instructional guidance [71]. Aligned with Brennan's guidance to balance agency with structure [129], we advocate for increased structure early in the CT concept learning period as a means to enrich learning outcomes as agency increments.

In the current study, we seek evidence of the efficiency offered by constrained problem spaces by integrating PPPs into Scratch, a block-based environment initially designed for informal learning that invites exploration, collaboration, and knowledge construction through personally meaningful creation [3]. K-8 teachers use Scratch more than any other coding language internationally [130], resulting in an ecosystem with approximately one million monthly active users [65], and more research focus than any other environment in K-12 from 2012-2018 [131]. However, as noted in Chapter 1, historical findings indicate Scratchers

infrequently demonstrate skill increases over time [67], and in a recent study of 74,830 Scratch projects, 45% contained at least one bug pattern [132]. Instead of problem-solving algorithmically, Scratchers often engage in discovery learning through bricolage [66], which involves bottom-up tinkering that does not necessarily prove productive [133].

To balance this agency with structure as recommended in [129], and to encourage the development of desired habits when learning CT concepts without stifling learner creativity, researchers have designed external Scratch curricula [134,135], created introductory Scratch Microworlds with reduced functionality [136], and devised learning strategies based on the Use->Modify->Create pedagogy to scaffold instruction [137]. We extend this trend by integrating with Scratch PPPs with explicit goals that offer gameful scoring targets and per-block feedback to disincentivize trial-and-error behavior and steer learners toward correct solutions. We reason that if learners initially can internalize CT concepts efficiently via PPPs, they can better deepen their understanding in less-restrictive interest-driven projects such as those described in [138] that embrace Scratch's roots in constructionism [129]. This strategy would enable the development of learning progressions through the cognitive, situated, and critical framings of CT documented in [110], so that skill building leads to creative expression and participation in fostering equitable, ethical computing for all.

To test this reasoning, we ran a pilot study targeting adults, who comprise a general population that might not only benefit from learning CT, but who might most effectively mobilize the advancement of teaching and learning CT. The study separates participants into one of three training conditions: 1) PPPs; 2) PPPs with distractors (PPPDs); 3) programming with access to all blocks and without feedback (limited-constraint-feedback or LCF). Each successive condition offers the learner increasing agency by offering more block options from which to

construct puzzle solutions. Condition 3, with block-move correctness feedback and scoring removed, most closely resembles the code writing experience native to Scratch. We investigated the following research questions: **R1**) what are the effects on motivation and cognitive load when training occurs via: PPPs; PPPs with distractors (PPPDs); programming with access to all blocks and without feedback (limited-constraint-feedback or LCF)?; **R2**) what are the effects on learning efficiency for training via PPP, PPPD, and LCF? Although the 75-participant sample limits the number of statistically significant results, findings indicate: **F1**) participants self-report higher motivation when training via PPPs and PPPDs, and less extraneous cognitive load when training via PPPs than via PPPDs or via LCF; **F2**) participants training via PPPs and PPPDs experience increased learning efficiency compared with those training via LCF.

We first review the background and the required software development. We then document the study purpose, formative and summative evaluations, and results.

2.2 Background

Since PPPs emerged in the CS literature as a new form of program completion problem in 2006, the community has investigated their strengths and weaknesses. Strengths include: scaffolded support of syntax and semantics learning; solvers with prior experience perform better and need less time [72]; quicker grading and less grading variability than code writing problems [73]; easier detection of learning differences between students compared to code writing and code fixing problems [139]; a moderate correlation between PPP proficiency and code writing proficiency in an exam setting [140]; less completion time required than for code writing exercises with equivalent performance on transfer tasks [73,127]; higher enjoyment and less completion time required than for tutorial users with better performance on transfer tasks [141]; and a lack of significant differences in performance across gender. Weaknesses include:

constriction of puzzle-design surface to maintain single-solution structure (not strictly required, but commonly enforced to maintain strengths); the invitation of trial-and-error behavior in PPPs with excessive corrective feedback [142]; and a potential ceiling effect when feedback guides most learners to solve PPPs correctly, resulting in the need to evaluate learner process in addition to learner product when assessing [118].

The community also has explored differences in learning outcomes resulting from using different PPP elements. Evidence suggests that 2D puzzles, in which the student must not only correctly order programming constructs but also indent them correctly, are more difficult than 1D [143]. Similarly, PPPs that conceal the number of lines of code needed for each solution section and those that include distractors are more difficult, require more time to complete, and produce higher cognitive load during training than those that specify section sizes and those without distractors [144,126]. Learning differences continue to emerge when researchers vary these elements. For example, learners struggle more when distractors are randomly distributed among the correct code constructs than when they are paired with correct constructs [140].

To identify these strengths, weaknesses, and learning differences between PPP elements, researchers often evaluate cognitive load. According to CLT, the brain provides limited short-term memory and processing capability along with infinite long-term memory, and learning occurs via schema construction and elaboration that leads to automation. Construction ensues by combining new, single elements into one larger element, and elaboration follows by adding new elements to an existing, larger element. Through intensive practice, individuals can automate their processing of these larger elements so that they execute without controlled processing.

CLT helps distinguish characteristics of and between PPP systems by offering a framework with tools to measure the three types of cognitive load experienced: intrinsic,

extraneous, and germane. The total number of interacting elements perceived by the learner determines intrinsic load; the sometimes-impeding organization and presentation of the content determines extraneous load; and the instructional features necessary for schema construction determine the germane load. PPP designers aim to reduce extraneous load to free learners' capacity to contend with germane load when attempting to maximize learning efficiency. For example, the pairing of distractors with correct constructs might increase germane load by focusing student attention on the intended, misconception-revealing differences between two solution options, while also reducing extraneous load by eliminating the need to search for and identify the two relevant options amidst a random distribution of constructs.

To measure relative learning efficiency quantitatively across conditions, researchers calculate instructional and performance efficiency [145]. These calculations account for learners who compensate for an increase in mental load by committing more mental effort, thereby maintaining constant performance while load varies. The data recorded often include empirical estimates of mental effort during instruction (EI) and transfer (ET) tasks and the performance (P) on transfer tasks. The EI and P calculation measures the instructional efficiency of the learning process, while the ET and P calculation measures the performance efficiency of the learning outcome. For example, in a study that included interactive puzzles in the transfer phase, results indicate PPPs with randomly distributed distractors decrease performance efficiency [72]. In the study described here, we measure instructional efficiency with a focus on learning process economy.

2.3 Implementation

To investigate our research questions, we modified Scratch to facilitate the design, play, and assessment of PPPs. In alignment with the principles described in Subsection 1.4.3 and the

gamification strategy described in [146], in which the game elements were added to SQL-Tutor, and similar to recent iSnap integrations offering progress panels and adaptive messages [127,147], we augmented Scratch to influence the behavior of learners. As shown in Figure 0.1, we first established a design mode which enables content developers to assign points to individual blocks and select blocks for inclusion in a new PPP palette. Equipped with this functionality, instructors can assign higher point values to blocks relevant to the CT concept studied and can isolate in a single palette blocks pertinent to the puzzle.

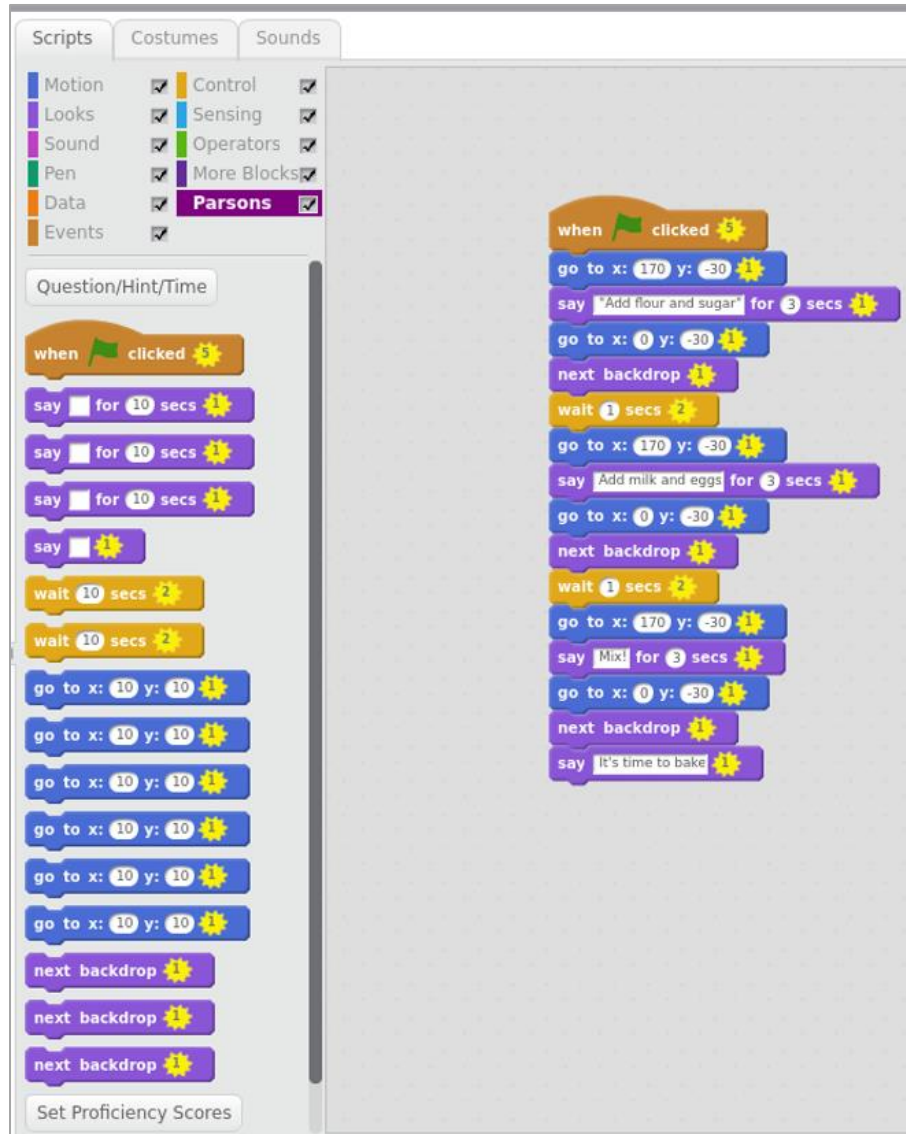


Figure 0.1: PPP palette and block configuration in design mode in Scratch.

As presented in Figure 0.2, we next established a play mode which enables students to load PPPs in a manner that displays the designed animated elements in the Scratch stage, but none of the blocks in the scripts pane authored as the solution. To facilitate the generation of feedback, we add an assessment system that includes a gameful scoring algorithm intended to encourage deliberate practice and discourage trial-and-error behavior. Our early development attempts involved calculating the Manhattan distance of each block placed from its correct position and multiplying that by the points assigned to each block and the length of the sequence,

combined with subtractions for special cases such as multiple disconnected sequences of blocks in the learner solution. During testing, however, this strategy proved insufficient, as scores could confusingly decrease when a block placed incorrectly in a long sequence was moved to the correct place in a shorter sequence. The longest common subsequence feedback algorithm described in [124] ultimately inspired our final approach; ours differs in that we leverage block points, use them and subsequence length as multipliers, and sum the multiples from all subsequences matching the single correct solution while also deducting for incorrectness in absolute position. The closer the participant is to the solution, the higher the score.

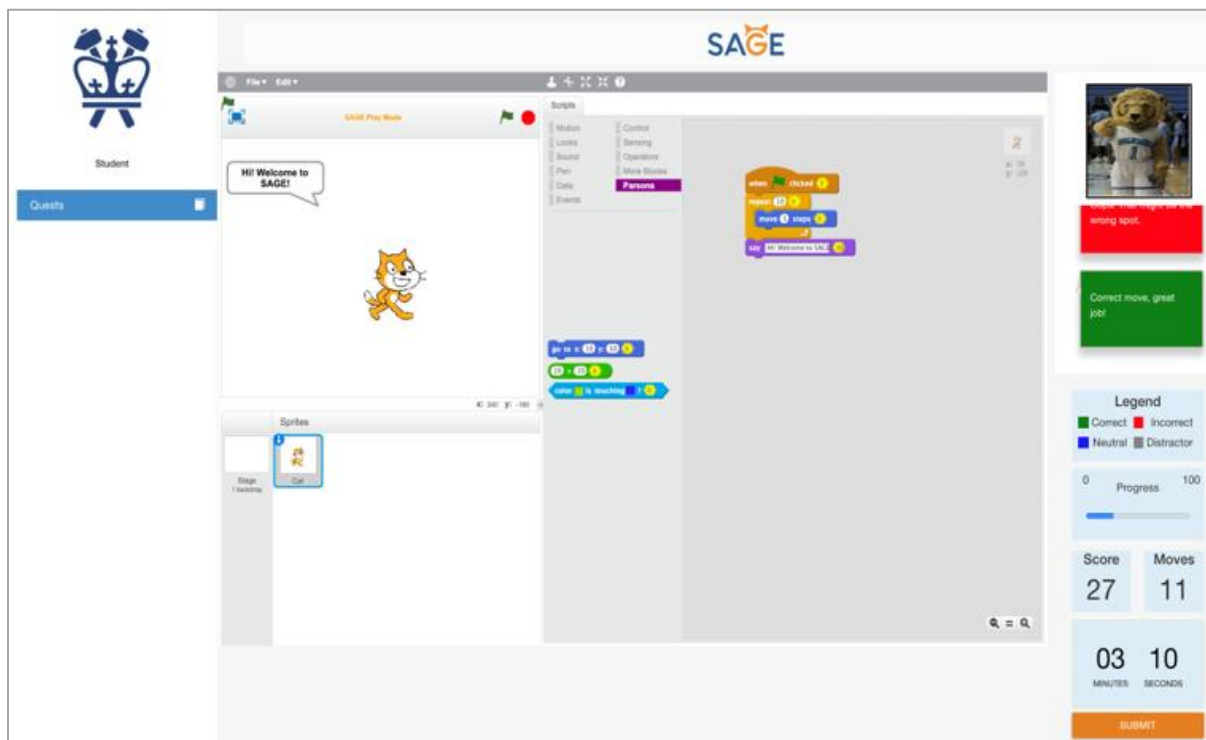


Figure 0.2: Puzzle play and assessment functionality integrated via PPP in Scratch.

To reduce complexity in the scoring algorithm but still discourage trial-and-error behavior, we simultaneously track a count we name *meaningful moves*, which increments when the learner drags a block from the palette to the scripts pane, connects existing sequences together, or disconnects a sequence into two. Other less significant block actions, such as the repositioning of

a block or sequence within the scripts pane are discarded. Since we display this count (11) next to the score (27) and remaining time (3m 10s) as shown in Figure 0.2, we can encourage learners to achieve the highest score in the fewest moves and shortest time.

Additionally, we built auto-initialization and auto-execution functionality to reflect progress visually after each block placement during puzzle play. These mechanisms enable the display of gameful animations while an avatar presents per-block correctness feedback, while concurrently disabling Scratch features that might otherwise distract from CT learning, such as sprite editing controls. According to the feedback classification in [148], this immediate correct-incorrect-distractor feedback is constructivist since it is problem- and instance-oriented, which has been correlated with significantly lower student failure rates than alternative types such as those solution- and theory-oriented. The auto-execution functionality also calculates completion progress so that the learner receives appropriate responses when she correctly solves the puzzle or the allotted time expires.

To help teachers and content developers organize learning, we wrapped these new features in Scratch within custom-built learning management tooling that facilitates the ingestion of class rosters, the structuring of learning paths in which games comprise quests which comprise missions, and the saving and loading of game/quest/mission progress. We intend this playful framing to further gamify the learning experience and nudge learners toward increased motivation as described in [149]. For the motivation to sustain beyond this learning experience, however, further progress in standardizing interoperability protocols between learning systems is necessary throughout the CS education community, similar to the one proposed in [150]. The reference architecture presented in Subsection 1.7.3 is intended as one small step forward in that direction.

2.4 Study Purpose

This extended functionality positioned us to fill gaps in existing research. One study purpose was to explore the adult-use of CT learning system functionality primarily designed for children. Recent research has: 1) found significant correlation of motivation and previous programming experience with self-efficacy and inclination toward a CS career in elementary students [151]; 2) indicated drag-and-drop programming can increase three CS motivational factors in middle school [152]; 3) suggested computing experiences prior to university can affect the world-image of computing habits, perceptions, and attitudes which enable or inhibit pathways into CS [153]; and 4) illuminated benefits of community commitment and a CS/CT focused ecosystem inclusive of the home and community [154,155]. Since demographic factors can drive communal values, and perceptions of how computing fulfills those values can affect sense of belonging and student retention [156], we measure adult motivation and cognitive load while probing for attitudinal change that might influence the CT inclination for participants' children.

A second purpose was to further identify PPP elements that optimize learning efficiency, since the behavior of programming environments can affect novices' learning [157]. While many researchers have hypothesized [140] and less often produced evidence [74] that PPPs can result in more efficient learning than alternatives such as writing or fixing code, some have attempted to measure the contributions of various PPP elements [158,78,159,160], including the effect of displaying the number of lines of code in puzzle solutions (no affect on pre-post improvement, more time spent), of pairing distractors with related variants (effective in the longer of two studied puzzles), of using single-character or mnemonic variable names (no significant differences), and of presenting program visualizations alongside PPPs (visualizations were used

most by novices who sought feedback the most via multiple submissions). We measure PPP learning efficiency with and without distractors, while offering a comparison to programming with LCF. Derived from the literature, our hypotheses were: **H1**) PPP and PPPD training increase motivation and reduce extraneous cognitive load compared to training via programming with LCF; **H2**) PPP training yields highest learning efficiency.

2.5 Formative Evaluation

As an early step in a roadmap of studies intended to explore the efficacy of adding gameful systems to novice programming environments, and with an aim to reinforce construct validity, we engaged in a formative evaluation with grade 6-9 educators. Through design thinking activities, we advanced our learning design technique, similar to the approach described in [161], which illuminates design thinking as a strategy useful for exploration, managing uncertainty, learning from failures, and empathizing with the needs of the learner when connecting learning objectives to learning design. Our goals included: 1) identifying the CT concepts receiving focus; 2) eliciting the pedagogical needs of practicing teachers; 3) and refining puzzle and feedback systems. We focus discussion in this chapter on goal 1.

2.5.1 Participants

The participants included 21 teachers from learning organizations such as Girls Who Code [19] and codeHER [162], and 17 from U.S. schools. 11% had taught with Scratch for at least 2-4 years, 63% for 6-18 months, and 26% had instructed with Scratch for less than 6 months. 34% of the teachers used Scratch for at least 51% of their curriculum, 29% used it for 26-50%, and 29% used Scratch for at least 11-15%.

2.5.2 CT Concept Engagement

[163] highlights the concerning status quo in which most studies in the field focus on a single institution and a single course, without validation by subsequent replicating research, leading to limited understanding of the reasons results occur. To contribute replication results, and to identify the CT concepts receiving focus, we distributed a survey that included a question from a survey previously distributed to K-9 teachers in five European countries [164]. This question asks teachers to respond with their perceptions of student engagement in nine facets of CT. Since we targeted a narrower set of teachers in the U.S., it is perhaps unsurprising that the results do not match the earlier international study, in which teachers reported their students most frequently use CT concepts related to data (e.g. analysis). However, we present this finding to reinforce the replication concerns raised, and to underscore the challenges the community faces when attempting to disseminate CT globally.

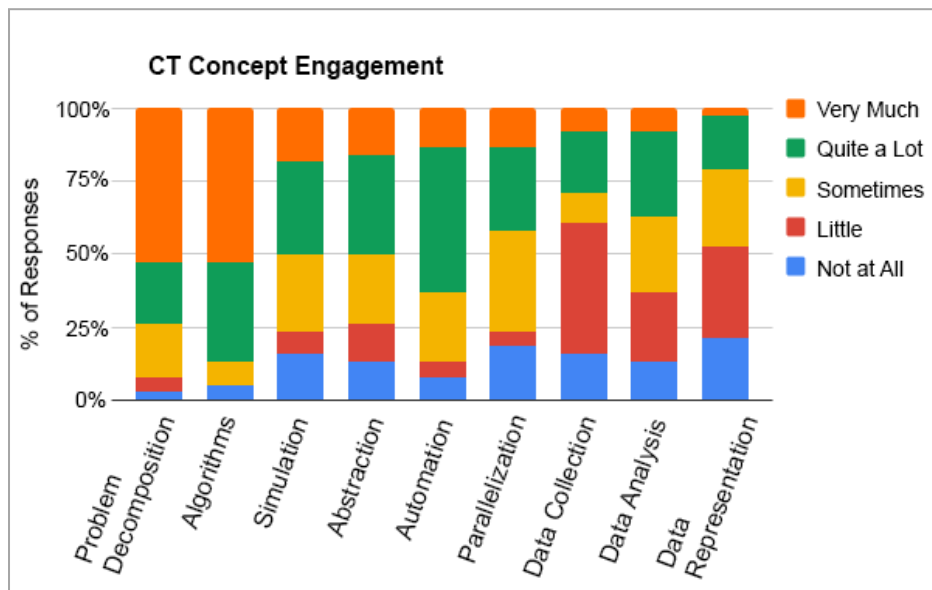


Figure 0.3: How a small sample of U.S. teachers perceive student engagement in CT concepts.

Our findings in Figure 0.3 indicate teachers perceive their students engage in data CT concepts less than others such as abstraction and algorithms. Aside from the differences in population samples, and the associated threat to internal validity due to implicit differences in curricula ([165] notes a low ratio of data knowledge in K-9 U.S. CSTA materials, 2%, compared with the English national curriculum, 14%, English Computing at School, 16%, and Italian guidelines, 25%), an extra explanation for this contrast could be related to the respondent recruitment process, as we specifically targeted Scratch teachers, whereas the earlier study did not. Since the small sample introduces a threat to external validity, future studies could try to replicate these results while controlling for technology and teacher pedagogical content knowledge (PCK) utilizing a Content Representation approach like the one described in [166], in which researchers elicited via interview then charted teachers' PCK across eight categories. Regardless, the lack of student engagement with data warrants investigation, as it is an alarming result for an increasingly data-driven society.

2.6 Summative Evaluation

2.6.1 Study Design

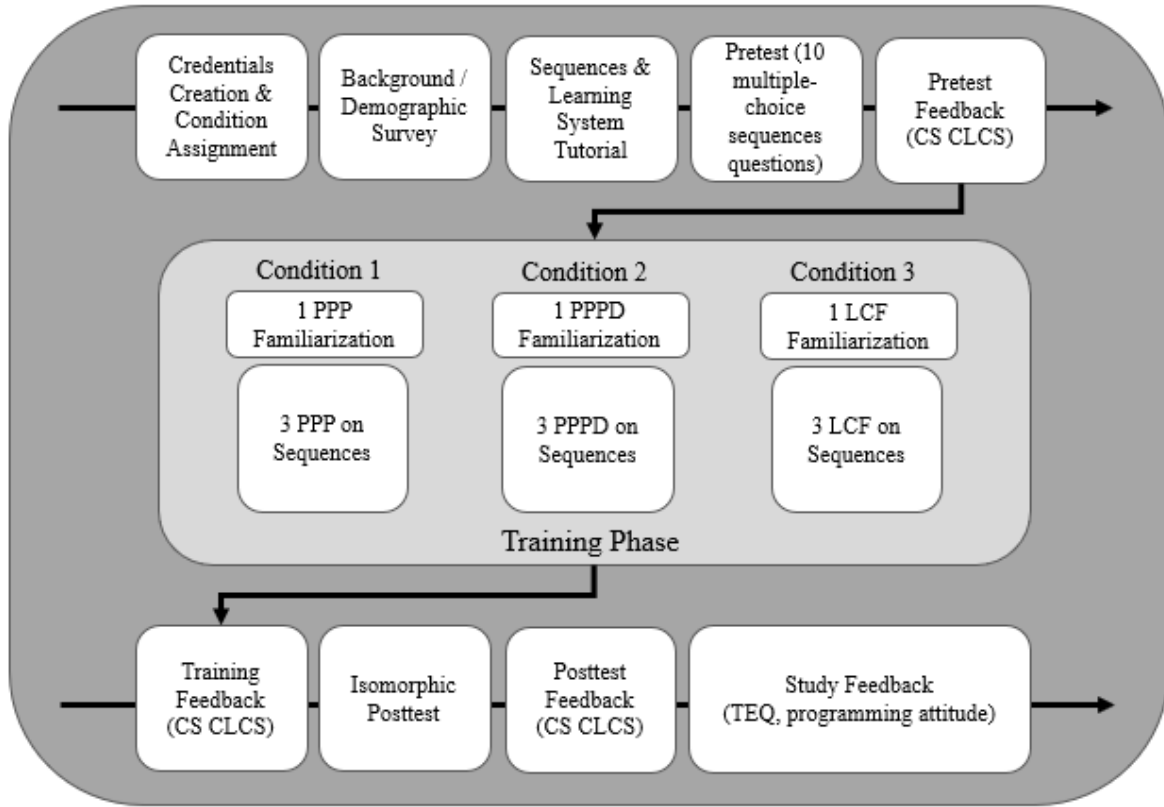


Figure 0.4: Summative evaluation protocol.

The formative evaluation helped us roadmap implementations, craft learning materials, and plan an initial summative evaluation. To produce evidence supporting answers to **R1-2**, we organized a 10-step between-subjects study via Amazon Mechanical Turk [167] with the CT concept *sequences* operating as the learning objective. As depicted in Figure 0.4 and detailed in Table 0.1, the steps involved: 1) creation of credentials in the learning system and assignment to one of three conditions characterized in Table 0.2; 2) a background survey; 3) review of a 6-minute video tutorial on the UI and CT concept *sequences* delivered via Panopto [168]; 4) a pretest; 5) pretest feedback; 6) familiarization and training varied by condition; 7) training feedback; 8) an isomorphic posttest; 9) posttest feedback; 10) study feedback. These steps

required responses to the validated CS cognitive load component survey (CS CLCS) [169], to a programming attitude Likert scale survey derived from categorized text-based responses by adult learners in [170], and to the intrinsic motivation Task Evaluation Questionnaire (TEQ) [106], which is a validated 22-item Likert scale measurement designed to reflect participant experience on four subscales: interest/enjoyment, perceived competence, perceived choice, and pressure/tension. In step 6, participants followed written instructions to guide their solving of four puzzles; instructions for the first puzzle included a graphical representation of the correct solution and an explanation of the behavior of each block used for the purpose of familiarization. Each puzzle auto-submitted upon correct completion or after 500 seconds if the participant had not previously submitted an incorrect solution. We advised participants to complete steps 4, 6, and 8 without interruption and required completion of all steps within two hours. Protocol materials are publicly available in <https://bit.ly/3uhSUd7>.

Table 0.1: Study protocol and measurements.

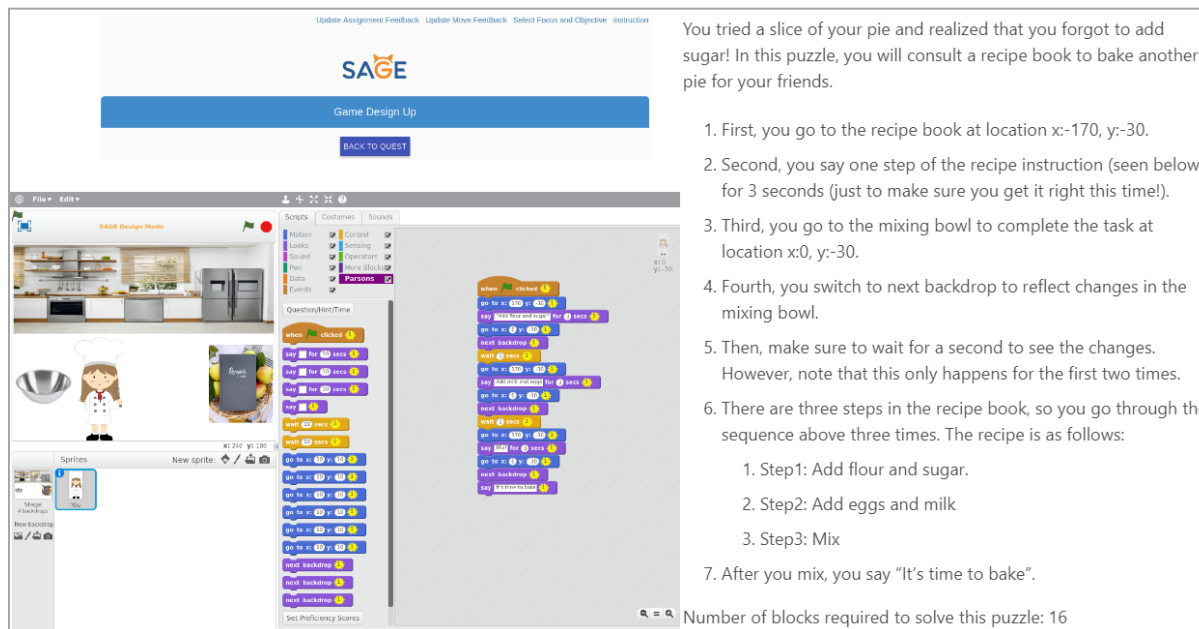
#	Activity	Content	Data Collected
1	Registration	Credentials creation & condition assignment	Username & password
2	Background info	Demographics	Age, gender, education, country, programming experience & attitude, CT perceptions
3	Tutorial	6-minute video on the learning system & <i>sequences</i>	N/A
4	Pretest (isomorphic)	10 multiple-choice <i>sequences</i> questions	Pretest responses & score
5	CS CLCS	10 CL questions with 0-10 scaled responses	Pretest CL & IL/EL/GL components
6	Puzzles	4 puzzles on <i>sequences</i>	Per-puzzle time spent, time-stamped block moves and score, correctness, distractor usage, generated feedback
7	CS CLCS	10 CL questions with 0-10 scaled responses	Puzzle CL & IL/EL/GL components
8	Posttest (isomorphic)	10 multiple-choice <i>sequences</i> questions	Posttest responses & score
9	CS CLCS	10 CL questions with 0-10 scaled responses	Posttest CL & IL/EL/GL components
10	Concluding measurements	Motivation, programming attitude, CT perceptions	TEQ score, programming attitude, CT perceptions

Table 0.2: Training and participant characteristics and differences across three study conditions.

Condition	Presentation	# Distractors	Feedback	# Participants	Avg. Prog. Exp. 0-10	Gender	Country
PPP	1-palette	0	Correctness	31	3.3	65%M 35%F	65% U.S. 19% India
PPD	1-palette	2-4	Correctness & distractors	22	4.7	50%M 50%F	67% U.S. 14% India
LCF	All palettes	All Scratch blocks		22	3.7	68%M 32%F	50% U.S. 23% India

We randomly assigned participants to one of three independent variable conditions: 1) PPP; 2) PPPD; 3) LCF. The dependent variables included time and performance on the pre/posttests, time and block moves in puzzles, and the cognitive load, programming attitude, and TEQ results.

2.6.2 Materials



You tried a slice of your pie and realized that you forgot to add sugar! In this puzzle, you will consult a recipe book to bake another pie for your friends.

1. First, you go to the recipe book at location $x:-170, y:-30$.
2. Second, you say one step of the recipe instruction (seen below) for 3 seconds (just to make sure you get it right this time!).
3. Third, you go to the mixing bowl to complete the task at location $x:0, y:-30$.
4. Fourth, you switch to next backdrop to reflect changes in the mixing bowl.
5. Then, make sure to wait for a second to see the changes. However, note that this only happens for the first two times.
6. There are three steps in the recipe book, so you go through the sequence above three times. The recipe is as follows:
 1. Step1: Add flour and sugar.
 2. Step2: Add eggs and milk
 3. Step3: Mix
7. After you mix, you say "It's time to bake".

Number of blocks required to solve this puzzle: 16

Figure 0.5: Example *sequences* PPP solution and instructions.

Following guidance in [126], we aimed to design motivating scenarios with memorable segments while providing a challenge without being tricky and leaving the participants with a positive impression. To familiarize them, we included in the instructions for the first puzzle the solution and block-use descriptions. We also included more detailed instructions than typically found in PPPs, effectively resulting in a hybridization of the tutorial and PPP approaches described in [141]. We thought this approach might best minimize ambiguity and highly scaffold early learning of new CT concepts in the absence of an instructor. An example puzzle solution and the associated instructions are shown in Figure 0.5.

We tested and refined our materials in collaboration with a high school teacher, 16 of her freshman physics students with little prior exposure to CT, and eight undergraduates with diverse majors. Tests included trials of the surveys and puzzles, and think-alouds in which the participant would interact with puzzles while verbalizing her thoughts. Results led to refinements such as

puzzle theme modification, normalization of pre/posttest difficulty, and simplification of language used in survey questions.

2.6.3 Participants

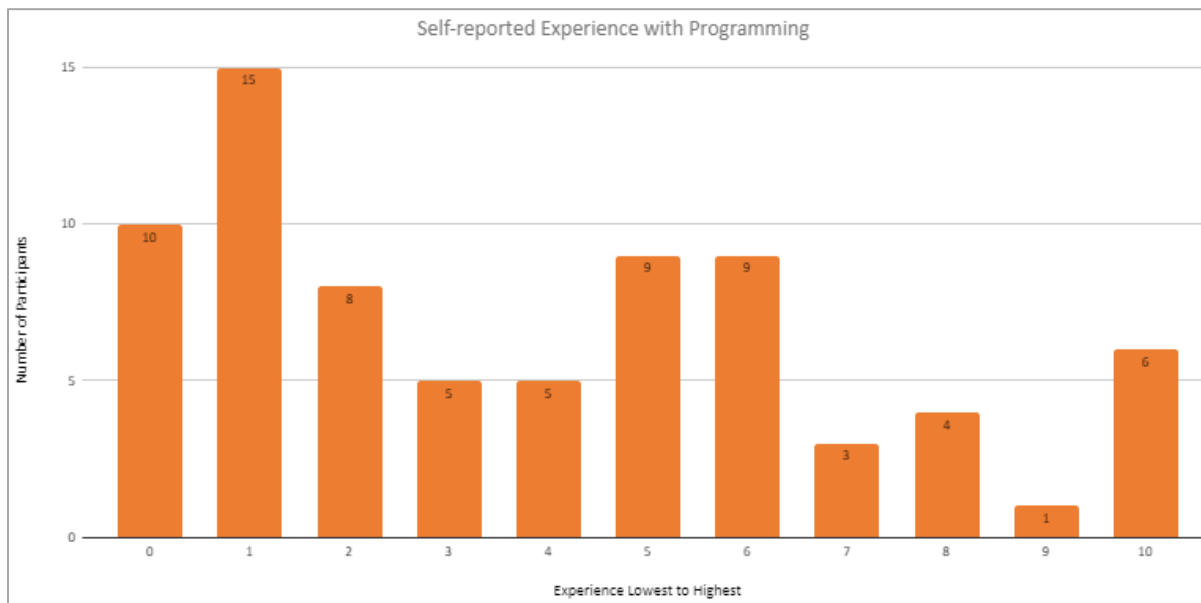


Figure 0.6: Participant self-reported programming experience at the start of the study.

In alignment with Wing’s mobilizing declaration that CT is a “fundamental skill for everyone, not just for computer scientists” [1], and with the interventionist spirit of design-based research [171], we sought a learner population inclusive of those who might not otherwise encounter an opportunity to engage purposefully with CT but regardless might influence its trajectory in the lives of children. The learning objective of the CT concept *sequences* is suitable for this largely novice set of participants, as teachers often present this concept first in a CT curriculum (e.g. [134]). By presenting our study as a Human Intelligence Task on Amazon Mechanical Turk, we recruited from a general population of over 100K individuals [172] 75 adults with varying educational experience (24% graduated high school, 60% earned an undergraduate degree, 16% earned a graduate degree) and the variety of self-reported programming experience presented in Figure 0.6. 46 men and 29 women comprise the sample

population sourced from eight countries including the U.S. (60%), India (20%), and Brazil (11%). As presented in Table 0.2, the backgrounds and self-reported programming experience of participants across conditions are largely homogenous, with slightly higher average programming experience on a 0-10 scale reported for the PPPD condition (4.7) than LCF (3.7) and PPP (3.3), higher female participation in the PPPD condition (50%) than in PPP (35%) and LCF (32%), and lower U.S. representation in the LCF condition (50%), than in PPP (65%) and PPPD (67%). Additional participant demographic detail and all summative evaluation data are available in <https://bit.ly/3uhSUd7>.

2.7 Experimental Results

2.7.1 Data Collection & Processing

We created seven surveys in Qualtrics [173] to capture data not directly collected by our CT learning system. For the pretest and posttest, we recorded time elapsed, and to grade the multiple-choice responses, we wrote a Python script. To help measure performance and efficiency, we added instrumentation to: 1) record time from puzzle start until submission; 2) trace each block moved; and 3) calculate score via the algorithm dependent on block position and points described in Section 2.3. In the following subsections we analyze the collected and processed data. Since they did not exhibit Shapiro-Wilk normality ($p < 0.05$), we used non-parametric statistics, including Kruskal-Wallis H, Mann-Whitney U, and Spearman r tests between-subjects, and Wilcoxon tests within-subjects, to address skewness and kurtosis. We used guidelines for characterizing effect sizes in [174,175] (H) ϵ^2 : .01 < .04 for weak, .04 < .16 for moderate, .16 < .36 for relatively strong, .3 < .4 for strong; U) r: 0 < .10 for small, .11 < .30 medium, .31 < .50 for large). Due to sample-size limitations, we report both significant findings and those non-significant that appeared to have the highest potential to reach significance in

post-pilot studies with hundreds of participants, as those results influenced development, as well as studies discussed in upcoming chapters.

2.7.2 Cognitive Load

We did not find significant differences in overall cognitive load during training between conditions ($H(2)=.51$, $p=.776$), nor in the subtypes. Upon closer review, we found no notable differences in intrinsic and germane load, but moderate non-significant differences in extraneous load (PPP: $M=3.12$; PPPD: $M=3.55$; LCF: $M=3.90$). This result signals weak support for **H1**, as PPP participants self-reported lower extraneous load than PPPD participants, while LCF participants reported the highest. Since the LCF condition presented far more block choices (548 across four puzzles) than the PPPD condition (55), which in turn presented more choices than the PPP condition (41, rank correlation $r(1)=1.0$, $p<.01$), this result indicates reducing impediments to block identification frees capacity for intrinsic and germane load. The higher extraneous cognitive load for training via PPPDs than with PPPs aligns with the findings in [72], which reported no significant difference for intrinsic and germane load, but one for extraneous load with the highest mean in the distractor condition, in a study comparing PPPs with and without distractors. Pedagogically, distractors present an opportunity for the instructor and/or learning system to intentionally challenge the learner to address potential misconceptions. We recommend further study to track cognitive load as agency increases, since misconceptions not addressed during structured learning could amplify in open-ended environments, resulting in higher cognitive load if measured in sum across a longitudinal span. Incremental addition of blocks options, and the associated incremental EL, could help learners prepare for future learning as they advance.

2.7.3 Performance

Though we did not find significant training performance differences across conditions ($H(2)=.85$, $p=.653$), participants in the PPP and PPPD conditions interacted with the blocks significantly more with a relatively strong effect ($H(2)=21.14$, $p<0.001$, $\epsilon^2=0.29$). Using a Bonferroni-adjusted alpha of .017 (.05/3), we found significant differences between conditions PPP ($M=52.2$) and LCF ($M=32.1$), $p=0.001$, and PPPD ($M=57.9$) and LCF, $p<0.001$. The fewer block moves made by participants in the LCF condition indicates some may have perceived the task as sufficiently overwhelming to decrease the probability of exploratory programming behavior.

In addition to analyzing aggregate puzzle performance, we compared individual puzzles. Participants in the PPP and PPPD conditions correctly solved puzzle 3 with a significantly higher score ($H(2)=18.44$, $p<0.001$) and executed more moves ($H(2)=14.77$, $p=0.001$) than those who trained with LCF. Since the solution to puzzle 3 involved 14 blocks, the second-highest count, this result suggests PPPs and PPPDs, which help learners focus on smaller block selection sets, might increase training performance and motivation as the difficulty of the puzzle increases.

Although participants in each condition solved more posttest than pretest questions correctly (PPP: $M=0.65$, PPPD: $M=0.82$, LCF: $M=0.32$), with those in the PPPD condition yielding the highest increase, there is no significant difference in performance gain across conditions ($H(2)=1.34$, $p=0.513$). This lack of transfer performance disparity between PPP and PPPD conditions ostensibly replicates findings in [72], which found no significant difference in performance on transfer tasks for those training via PPPs and PPPDs. It is also similar to findings on PPP inter-problem and intra-problem adaptation in [74], in which no significant differences in

learning gains occurred from pretest to immediate posttest across three conditions involving PPPDs and one involving code writing, which is similar to the LCF condition in our study.

2.7.4 Efficiency

To measure efficiency, we analyzed training and transfer task time across conditions. During training, participants in the LCF condition, despite making fewer block moves, required significantly more time than those in the PPP and PPPD conditions with a moderate effect ($H(2)=6.20$, $p=0.045$, $\epsilon^2=0.08$). After the Bonferroni adjustment, significance moderated slightly: PPP ($M=9.3m$) vs. LCF ($M=11.3m$): $p=0.090$; PPPD ($M=9.6m$) vs. LCF: $p=0.063$). Since transfer task performance did not vary significantly across conditions, this result suggests training via PPPs and PPPDs enables more efficient CT learning, per the EI and P instructional efficiency calculation introduced in Section 2.2. We did not, however, find a significant difference in the transfer task time ($H(2)=0.88$, $p=0.643$).

To emphasize the opportunity for efficient CT learning, we calculated instructional efficiency, using pre/posttest improvement to measure transfer performance and both time and cognitive load as measurements of mental effort during training, as recommend in [176]. Figure 0.7 presents areas of high and low effectiveness separated by the effort line $E=0$. The chart depicts higher instructional efficiency for training with PPPs and PPPDs than with LCF. However, this result does not support **H2**, as the PPPD condition yielded the highest instructional efficiency. This result contrasts with findings in [72], which found evidence of decreased learning efficiency from PPPDs when compared to PPPs, but it aligns with hypotheses regarding distractor learning benefits in [2,124] that propose distractors can facilitate the highlighting of both subtle and complex principles as well as edge cases and common misconceptions.

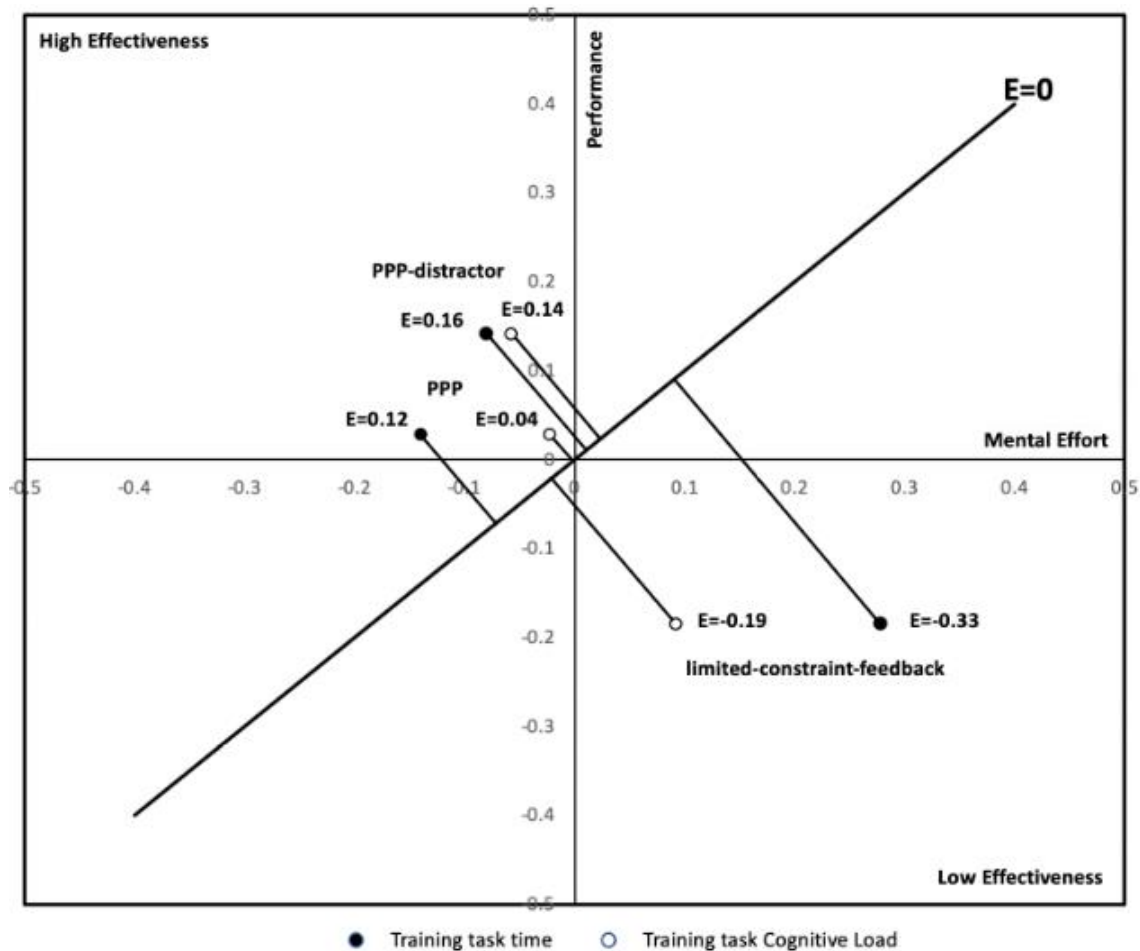


Figure 0.7: Instructional efficiency (E) for each of the three conditions.

2.7.5 Motivation

To analyze motivation quantitatively, we scored the TEQ and calculated the within-subject change in programming attitude that occurred between the start and end of the study. Although there were no significant differences in TEQ results across conditions, for the perceived competence subscale ($H(2)=.16, p=.925$), participants who trained with PPPs ($M=4.89$) and PPPDs ($M=4.91$) scored marginally higher than those who trained with LCF ($M=4.53$). Those who trained via LCF ($M=4.06$) reported marginally higher pressure/tension ($H(2)=2.63, p=.269$) than those training via PPPs (3.42) and PPPDs (3.45), which suggests that a

smaller block option set and correctness feedback after every move can decrease the experience of pressure when solving a timed parsons puzzle.

We also found significant positive attitude changes from the start to end of the study. PPP participants' attitude shifted most by: *perceiving programming as more fun* with a small effect size ($z=2.39$, $p=0.017$, $r=0.07$), *more enjoyable* with a medium effect ($z=2.43$, $p=0.015$, $r=0.28$), *easier to start* with a large effect ($z=3.04$, $p=0.002$, $r=0.55$), *less difficult to understand* with a large effect ($z=-3.34$, $p=0.01$, $r=-0.6$), and *less of a foreign concept* with a large effect ($z=-3.07$, $p=0.002$, $r=0.55$). PPPD participants' attitudes also shifted positively by: *perceiving programming as easier to start* with a large effect ($z=2.51$, $p=0.012$, $r=0.54$). LCF participants shifted least by: *perceive programming as more enjoyable* with a medium effect ($z=2.51$, $p=0.012$, $r=0.46$). When including only those with little prior programming experience, PPP participants reported *programming more as something they want to learn* with a medium effect ($z=2.00$, $p=0.046$, $r=0.48$) and *less boring* with a medium effect ($z=-1.96$, $p=0.050$, $r=0.48$) in addition to the attitude shifts described above. Although these results indicate attitude improvement and signal support for **H1**, the lack of longitudinal data poses a threat to internal validity, as we cannot claim change at study conclusion persists. The small sample also prevented the finding of significant differences when comparing between conditions. For example, participants training via PPPs reported a notable but non-significant decrease in their perception that *programming is a foreign concept* compared to PPPD and LCF participants ($H(2)=4.37$, $p=.113$, PPP vs. PPD: $p=.309$, PPP vs. LCF: $p=.186$). Related mean changes are included in the within-subject results in Table 0.3.

Table 0.3: Within-subject attitude change. Positive shifts (p), negative shifts (n).***p<0.05, **p<0.01**

Programming is...	PPP	PPP-distractor	Limited-constraint-feedback
something I've wanted to learn (p)	M=0.19	M=0.27	M=0.00
fun (p)	M=0.74*	M=0.40	M=0.36
enjoyable (p)	M=0.90*	M=-0.05	M=0.68*
important to know (p)	M=0.25	M=-0.05	M=0.09
easy to start (p)	M=1.35	M=0.68*	M=0.45
something that takes practice (p)	M=0.07	M=0.05	M=-0.32
too difficult to understand (n)	M=-1.48**	M=-0.77	M=-0.64
boring (n)	M=-0.41	M=-0.32	M=-0.54
a foreign concept (n)	M=-1.13	M=-0.27	M=0.00
too time consuming (n)	M=-0.35	M=-0.09	M=-0.09

To supplement the quantitative results, we sought qualitative feedback by requesting that participants describe their attitude or view toward programming after the learning experience. For both those who self-reported low and high prior programming experience, we recorded more hesitant responses from those who trained via limited constraint and feedback than those who trained via PPPs and PPPDs. One LCF participant who selected “have tried programming activities, but have not taken a class” in the demographic survey, reflected on sustained struggle: “I still feel like programming is insanely complex. When I was in college I dropped out of computer science as soon as we started python. I just couldn’t understand what we were doing, and maybe I could understand it if I really tried. It just seems to be better geared towards certain people.” A second LCF participant with the same prior programming experience selection revealed marginal incremental motivational change: “I have already begun to study programming but have not stayed consistent with my studies. This has encouraged me to give more attention to the subject.” A third LCF participant who selected “have tried programming activities, but not taken a class” alluded to seeking external supports: “I hope to translate what I have gained today to my

studies in coding. It is a bit tedious, and there is a lot to know, but I think that many basic codes can be written with the help of a search engine or some material.”

In contrast, PPP and PPPD participants reflected more direct positive attitudinal change. One PPP participant whose prior programming experience selection was “never attempted to program before” noted that she “definitely enjoyed the puzzles and feel[s] more knowledgeable in terms of programming. It made me much more interested in learning to program.” A second PPP participant who recorded the same prior programming experience discovered possibility in her capability: “I feel like it’s not as complicated as I thought it was. I could learn a lot through practicing more of it.” A third PPP participant who selected “have tried programming activities, but have not taken a class” demonstrated confidence in his ability as well as opportunity for novices: “[T]his activity was somewhat easy but programming is really much harder than this. [B]ut this is a good way for a kid to start learning.” Aligned with this viewpoint was one PPPD participant who selected “never attempted to program before” and revealed potential for future pursuit of CT: “I would love to learn more about programming and encourage my son to start learning programming early.” A second PPPD participant who selected “have tried programming activities, but not taken a class” focused on the puzzle approach to learning in his response: “I think it is a skill that can be learned through practice. It was nice to look at programming as a series of puzzles rather than a complex language.” These results support **H1** and those in [170], which found significant attitude improvement regardless of gender and education level after a brief online programming experience.

2.7.6 Findings Summary

We conclude the analysis by summarizing findings for each varied PPP element in Table 0.4.

Table 0.4: General summary of findings across conditions.

Condition	Extraneous Cognitive Load	Instructional Efficiency	Motivation/Attitude
PPP	Lower	Higher	Highest
PPD	Higher	Highest	Higher
LCF	Highest	Lower	Lower

2.8 Conclusions

Our survey of grade 6-9 teachers exposed teacher perceptions of limited student engagement with data concepts central to CT. These results led us to extend the trend of balancing Scratch’s agency with structure to better serve learners and reduce burden on teachers. A small pilot study of an adult population using a learning system that integrates PPPs with Scratch yielded results indicating the structure provided by PPPs catalyzes motivation for CT, reduces extraneous cognitive load, and increases learning efficiency without sacrificing performance on transfer tasks. These findings were originally published in 2021 at the 29th International Conference on Computers in Education, where the article was nominated for two best paper awards [177]. The elaborated study presentation and analysis is also included in volume 18 of the journal *Research and Practice in Technology Enhanced Learning* [178].

While these results revealed opportunities to advance the teaching and learning of CT via augmentations to block-based programming environments, we remained cautious due to external validity limitations: the single CT concept, *sequences*, and small summative evaluation population (75 adults), threatened generalizability. In the upcoming two chapters, we document studies with additional CT concepts, *conditionals* and *looping*, the configurable integration of multiple Scratch palettes for each puzzle, functionality variation, such as offering increasing agency through the introduction of instructor-defined, objective-driven feedback, and the fading of correctness feedback. We also increase the sample size by organizing online studies with over

500 adults. These conditions facilitate the study of CT learning when augmented by tooling that applies incremental cognitive load intended to deepen CT concept uptake, and transition learners toward interest-driven projects that sustain motivation.

Chapter 3: Learning Computational Thinking Efficiently

3.1 Motivation

As earlier introduced, CT is becoming a necessary literacy alongside reading, writing, and arithmetic [1]. However, negative perceptions and poor understanding of computing remains prevalent among the general population across ages [153,170]. The dependency on CT competencies has emerged in an era of standardized testing that limits curricula flexibility in formal education, leading administrators toward zero-sum requirements choices [179]. When schools do integrate CT, they often operate with teachers with limited subject or pedagogical content knowledge who are unlikely to have learned CT in K-12 [12,180].

The CSforAll initiative in the U.S. used a commitment-making model to engage the community and distribute effort to local leaders in both formal and informal education [155]. The scale of the needed transformation, though, introduces risk to lasting equitable CT uptake for all. Since different demographic groups' communal values vary, and computing is not perceived to fulfill those values equally across groups, the opportunity to learn CT and develop a sense of belonging in computing is important to afford both to children and the adults who shape those values to reduce inequity [156]. A CS career orientation for female students is strongly correlated with self-efficacy in computing [151], and computing exposure and encouragement most influence women to study the field [181], yet only 33% of the 2018 enrolments in the Australian National Computer Science School challenge were female [182], and 20% of 2018 Information and Communications Technologies OECD tertiary education students were women [183].

Since motivation and previous programming experience can be highly correlated with self-efficacy and CS career orientation across genders [151], and computing experiences impact computing self-image and habits [153], increasing the general populace's computing motivation and self-perception of CT skill are key objectives. For middle school students, previous experience with block-based programming has correlated with computer use and confidence, as well as interest in future CS courses [152]. Like early efforts to use block programming to introduce constructs and logic prior to syntax for adult university students [184], the study discussed in this chapter explores the motivation changes and CT learning efficiency associated with introducing to an adult population modified versions of a popular block-based environment, Scratch [3]. The work described in Chapter 2 integrated with Scratch PPPs, yielding findings aligned with others indicating this structured approach can lead to more efficient CT learning than alternatives such as via tutorial, or writing/fixing code [126,73,127]. In the study described here, we first further augment Scratch and then seek evidence of PPP learning efficiency and impact on CT motivation.

We ran an online international study targeting adults in which each participant was randomly selected into one of nine conditions to learn the CT concept *conditionals* via four puzzles each limited to eight minutes. Between conditions, we varied the presentation of programming constructs, the inclusion of distractors, feedback activation, and for the control condition, the CT concept. Here, we investigate three research questions: after at most 32 minutes (m) of puzzle solving, what are the effects of PPP variation on adult learner **R1**) CL?; **R2**) CT motivation?; **R3**) learning efficiency? Findings from 624 participants indicate: **F1**) PPPs with feedback and without distractors produce the lowest CL; **F2**) PPPs with feedback produce highest CT motivation; **F3**) PPPs with an isolated block palette and without distractors produce

highest CT learning efficiency. We next review related work and the software developed, then the study purpose, formative and summative evaluations, results, and conclusions.

3.2 Related Work

Section 2.2 documents PPPs emergence as a new form of program completion problem in 2006, as well as their strengths and weaknesses, [74] consolidates results from empirical studies, and [185] reviews the many variants in the research literature. Summarily, the scaffolded support of syntax and semantics learning can lead to shorter training time compared to code writing without reducing performance on transfer tasks, resulting in meaningful learning efficiency gains. Here we review work related to three axes of variation introduced in our study to identify optimal conditions for efficient CT learning: 1) presentation; 2) distractors; 3) feedback.

3.2.1 Presentation

The form of content presentation can significantly influence the learning experience [186]. Block-based languages offer visual, natural language, browsability, drag-and-drop composition, and dynamic rendering affordances beyond those typically available in text-based languages [187]. Open-ended environments like Scratch, however, traditionally offer no clear objectives nor direct instruction. Most PPPs provide learners with a precise challenge to solve via a prompt and focus the learner on arranging a small set of selected programming constructs. In the study discussed in this chapter, we replace short prompts with step-by-step instructions like those used in tutorials and vary the presentation of constructs across conditions.

[141] studied PPPs in comparison to tutorials by offering students choice in modality for each task and found that learners sought a challenge for 57% of PPPs compared to 32% when selecting tutorials, while aiming to avoid challenge in 39% of the tutorials compared to 12% of PPPs. PPP solvers also completed tasks in 23% less time and performed 26% better on transfer

tasks than tutorial followers while reporting higher mental effort. The tutorial tasks, however, attempted to teach both the interface mechanics and the programming concepts, while the PPPs directed learner attention only to the latter. While providing all participants tutorial-like instructions in PPPs, we vary the focus on interface mechanics by providing conditions either with blocks scrambled in an isolated palette, or with blocks scattered across palettes in categories native to Scratch (e.g. Motion, Events).

To investigate the effects on CT learning, we measure learner time-on-task, number of block moves, performance, and self-reported CL. Our study conditions with an isolated palette aim to reduce EL introduced via interface navigation to free learners' capacity to contend with GL. We also attempt to induce GL by requiring self-explanations after PPP play, like [188], since findings indicate self-explanation positively influences near and far transfer [189].

3.2.2 Distractors

Earlier research also has aimed to induce additional GL in PPPs by providing access to constructs not needed in solutions, called distractors. Example distractor types include: random constructs; unrelated control flow constructs; tangentially related constructs (TRCs); and partial suboptimal path distractors that might tempt a learner toward faulty progress without enabling her to solve the problem fully, thereby triggering misconception recognition and productive backtracking toward the correct solution [72]. Results indicate PPPs without distractors are the easiest to solve [144], and that more time and higher CL result when distractors are included, leading to lower learning efficiency [72]. To reduce distractor difficulty, some have tried pairing them with correct constructs to increase GL by focusing learner attention on misconception-revealing differences between two solutions while reducing EL by eliminating the need to search for the relevant options amidst a jumble [140]. In our study we vary across conditions the

inclusion of TRCs intended to help learners address misconceptions, and pair them close to correct constructs, with a goal of contributing insight into whether the efficiency lost by including distractors might be counterbalanced by the learning outcomes gained.

3.2.3 Feedback

Previous work has also explored increasing GL by including feedback mechanisms intended to guide learners to fix errors without providing the correct answer. These include line-based and execution-based feedback, as well as incremental progress indicators that introduce ambiguity to discourage trial-and-error behavior [118,72]. These studies analyze usage patterns without conclusions about which feedback type is the better option, and without evaluating feedback's impact on learning efficiency independent of confounding variables. In our study, we provide feedback through a progress bar reflecting points scored, a count of block-moves made next to the minimum count needed to solve the puzzle, and correctness feedback on the position of each block placed, as shown in Figure 0.1. As noted in Section 2.3, according to the feedback classification in [148], this feedback is constructivist since it is problem- and instance-oriented, which has been correlated with significantly lower student failure rates than alternative types such as those solution- and theory-oriented. We intend this feedback to be part of the landscape of interactions through which the learner constructs their understanding. For each variation of presentation and distractor disposition, we either activate or deactivate feedback, and then compare learning efficiency across conditions.

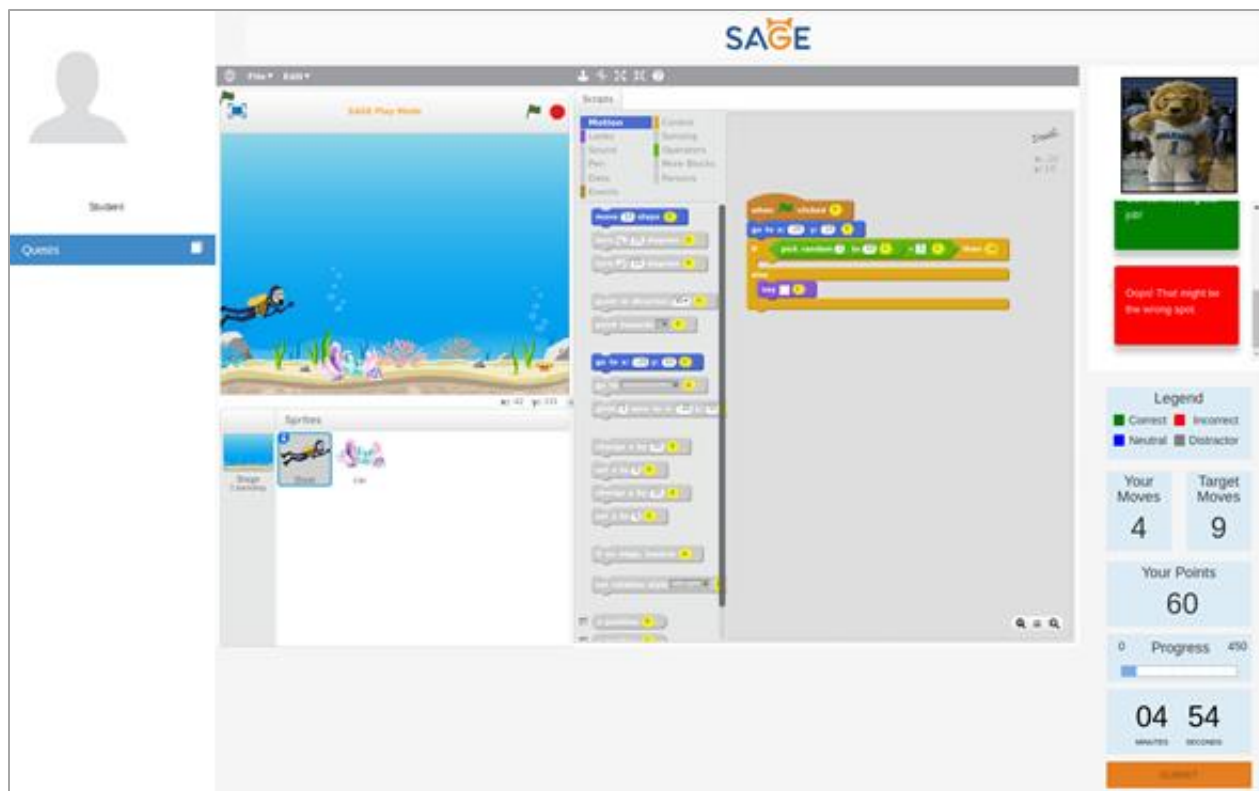


Figure 0.1: PPP with selectively included palettes and blocks.

Previous research found that participants encountering distractors performed similarly on transfer tasks and exhibited decreased learning efficiency compared to those who did not [72]. When PPPs are evaluated in comparison to writing equivalent programs, they have been shown to take half as much time to complete without reducing performance on subsequent assignments [127]. To account for learners who compensate for an increase in mental load by committing more mental effort, thereby maintaining constant performance while load varies, researchers calculate instructional and performance efficiency [145], as introduced in Section 2.2. For example, one study found that PPPs result in increased instructional efficiency compared to writing block-based code [177]; another indicated PPPs with randomly distributed distractors decrease performance efficiency [72]. Since we vary presentation, distractors, and feedback in

ways that might modulate CL, and potentially performance on transfer tasks, these efficiency measurements operate as equalizers in our analysis.

3.3 Implementation

To investigate **R1-R3**, we further modified Scratch to facilitate the design, play, and assessment of PPPs in alignment with the principles described in Subsection 1.4.3. Since user behavior can be affected by the programming environment [157], we aimed to preserve Scratch's strengths while integrating gameful elements like those in SQL-Tutor [146] and feedback systems similar to iSnap integrations which provide progress panels and adaptive messages [127,147]. Here we document development that facilitates presentation, distractor, and feedback variation.

Having previously introduced an isolated palette to Scratch that enables the assembly of blocks from native Scratch palettes (see Section 2.3), we further modified the palette selector to allow the selective inclusion of palettes displayed during puzzle play as shown in Figure 0.2a. Similarly, within each palette, we afforded the selective inclusion of each block during puzzle design (Figure 0.2b), resulting in a play experience in which the learner might encounter a subset of Scratch palettes and blocks enabled for use. This functionality allows us to construct matching PPPs with and without an isolated palette (Figure 0.2c). For example, in a condition without an isolated palette and without distractors (Figure 0.1), we enable only the palettes and blocks that are included within the original PPP isolated palette. Consequently, the learner must explore the enabled palettes to identify block options, instead of choosing each block from one jumbled assortment. While this exploration might induce EL, it could bridge learning from PPPs to open-ended Scratch assignments since it encourages familiarization with the conceptually driven block organization. This approach relates historically to SP/k [190], which offered a sequence of

language subsets (e.g. SP/1, SP/2) that introduced constructs incrementally; our learning system enables the design of palette and block subsets that learners encounter as they advance.

The selective inclusion of palettes and blocks also allows for distractors to be included or not. To provide an additional axis of variation, we configured a play mode with feedback systems disabled; the score-based progress bar, score value, number of moves, and correctness feedback, which includes alerts when distractors are used, are removed. We also remove the display of the minimum target moves needed to solve the puzzle, though this value remains discernable, since we also disable the puzzle submit button until twice the number of minimum moves is made to encourage engagement but also to provide relief if a learner is flailing and ready to move to the next puzzle, similar to [77].



Figure 0.2: Palette and block selective inclusion system.

3.4 Study Purpose

This implementation positioned us to study the effects of presentation, distractor, and feedback variation across conditions in a short intervention with a general adult population. One study purpose was to replicate findings of previous PPP studies that have typically targeted narrower populations, such as university or middle/high school students [74,126]. If CT is a crucial competency for all as Wing evangelized in 2006 [1], it is important for the community to confirm pathways toward motivated and efficient learning for the general populace. In an era of remote schooling acceleration, children with parents who know CT themselves likely are advantaged over their peers [191]. Furthermore, given the crisis in quantity of CS teachers equipped to teach CT in schools [12,192,193], there remains a need to identify scalable professional development tactics for existing teachers so that they can help students engage in creative computing [194,195]. The study described here aims to provide insight into the applicability of the block-based PPP paradigm for adults with the implications for children deliberately considered.

A second study purpose was to identify PPP elements that maximize learning efficiency. Previous PPP studies have varied grading leniency and distractor positioning [78], variable naming conventions [159], distractor inclusion [72], execution and line-based feedback [142], and intra-problem and inter-problem difficulty adaptiveness [74]. Like the approach used in [196] in which the type of instructional support is varied in a programming game, we varied the instructional support provided to learners across PPP conditions. We include eight conditions focused on the CT concept *conditionals*, plus one control on *sequences*, and analyze the effects of variation of presentation, distractors, and feedback, in the context of a series of four puzzles, each time-boxed for eight minutes of solving, that are connected by a story line with explicit

goals related to playful animations. Based on results discussed in Section 3.2, our hypotheses associated with **R1-R3** were: **H1**) training with an isolated palette and without distractors yields lowest CL; **H2**) training with feedback yields highest CT motivation; **H3**) training with an isolated palette with feedback and without distractors yields the highest learning efficiency.

3.5 Formative Evaluation

As first introduced in Section 2.5, to guide software development and reinforce construct validity, we conducted with middle-school teachers a formative evaluation in which we reviewed Scratch PPPs via a prototype using a design thinking approach. We review our survey and semi-structured interview results in this section. Our survey, data, and interview scripts are available at <https://bit.ly/3rhdvzt>, and a subset of the materials are included in Appendix A.

A concerning survey result is the frequency with which students ask for help when starting in Scratch (66% often or always). To assist learners, 58% often or always provide direct instruction using several techniques: concept explanations (used by 58%); 1:1 tutoring (53%); demonstrations (47%). One way to reduce help-seeking would involve the incremental introduction of blocks; 92% would be willing or very willing to utilize such functionality.

We also explored teachers' interest in PPPs and found that 84% had never assigned them. After reviewing PPP descriptions, however, they were willing or very willing to assign PPPs (66%) and author PPPs (71%). Although this indicates enthusiasm for CT teaching through puzzle gameplay, in the semi-structured interviews, several expressed skepticisms about the educational efficacy of CT games they've assigned to students. One teacher raised concern about the lack of provision for student reflection: "A lot of the games hit algorithmic approaches, but they don't give them [students] the critical thinking moment, where they are like, why are we doing this, the real question, why, and what is my problem?". Although articulating the

importance of reflection is non-trivial in the CS context as reported in [197], this teacher's approach entails designing small assignments prior to gameplay that introduce one block at a time and require students to explain their solutions. This feedback influenced the design of the selective palette and block inclusion system described in Section 3.3 and led us to add a self-explanation pop-up at the conclusion of each puzzle in the spirit of the reflective learning journals reported in [198] to provoke reflection that strengthens learning.

3.6 Summative Evaluation

3.6.1 Study Design

The formative study helped prioritize development, refine learning materials, and organize a summative evaluation via a quantitative experiment between-subjects across conditions, with some within-subject measurements. Similar to the study described in Chapter 2, as outlined in Table 0.1, participants: 1) created credentials in the learning system, which randomly assigned them to one of the nine conditions documented in Table 0.2; 2) provided demographic detail; 3) reviewed an eight-minute introductory tutorial on the learning system and the CT concept of *conditionals*; 4, 8) took isomorphic pre/posttests; 5, 7, 9) self-reported CL after tests/training through the validated CS CLCS introduced in Section 2.6.1; 6) trained via four puzzles time-boxed for eight minutes each; 2, 10) recorded CT perceptions and programming attitude via a Likert scale derived from categorized text responses by adults in [170] at study start/end; 10) responded to a validated intrinsic motivation TEQ introduced in Section 2.6.1. Materials are available at <https://bit.ly/3pXjO9o>.

Table 0.1: Study protocol and data collected.

#	Activity	Content	Data Collected
1	Registration	Credentials creation & condition assignment	Username & password
2	Background info	Demographics	Age, gender, education, country, programming experience & attitude, CT perceptions
3	Tutorial	8-minute video on the learning system & <i>conditionals</i>	N/A
4	Pretest (isomorphic)	10 multiple-choice <i>conditionals</i> questions	Pretest responses & score
5	CS CLCS	10 CL questions with 0-10 scaled responses	Pretest CL & IL/EL/GL components
6	Puzzles	4 puzzles on <i>conditionals</i> in 8 of 9 conditions; learning system behavior varies by condition; 4 puzzles on sequences in control condition	Per-puzzle time spent, time-stamped block moves & score, correctness, generated feedback, participant self-explanations
7	CS CLCS	10 CL questions with 0-10 scaled responses	Puzzle CL & IL/EL/GL components
8	Posttest (isomorphic)	10 multiple-choice <i>conditionals</i> questions	Posttest responses & score
9	CS CLCS	10 CL questions with 0-10 scaled responses	Posttest CL & IL/EL/GL components
10	Concluding measurements	Motivation, programming attitude, learning system feedback, CT perceptions	TEQ & programming attitude, learning system viewpoints, CT perceptions

Table 0.2: Variation and participation across nine study conditions.

Condition	CT Concept	Presentation	Distractors	Feedback
C1	Conditionals	1 palette	N	Y
C2	Conditionals	1 palette	Y	Y
C3	Conditionals	Multi-palette	N	Y
C4	Conditionals	Multi-palette	Y	Y
C5	Conditionals	1 palette	N	N
C6	Conditionals	1 palette	Y	N
C7	Conditionals	Multi-palette	N	N
C8	Conditionals	Multi-palette	Y	N
C9	Sequences	1 palette	N	Y

3.6.2 Materials & Participants

We developed four *conditionals* puzzles and reused four on *sequences* from the study described in Chapter 2. To familiarize learners, we included in the instructions for the first puzzle the solution and block-use descriptions. When including distractors, we targeted misconceptions to introduce cognitive conflict and reinforce learning as in [124]. For example, an if-else block operates as a distractor in a puzzle in which only two if blocks are needed.

To create the pre/posttests, we followed the seven-step approach to developing and validating CS knowledge assessments in [199]. To test and refine our materials, we collaborated with eight undergraduates with diverse majors and previous exposure to CS. Tests included trials of the survey content, pre/posttests, and puzzles, and think-alouds in which the tester would interact with the puzzles while verbalizing her thoughts. Although we did not further formally assess validity and reliability, these results led to refinements such as reduced ambiguity in puzzle instructions and eliminated pre/posttest questions deemed too easy or difficult.

Using Amazon Mechanical Turk and Prolific [167,200], we recruited 624 participants with varying degrees (43% high school, 35% undergraduate, 17% graduate), and a variety of self-reported programming experience (low: 52%; medium: 34%; high: 14%). 396 men, 221 women, and two non-binaries comprise the population sourced from 34 countries led by the U.S. (22%), Poland (21%), and the U.K. (13%). While practical, this recruitment introduces risk to external validity, as there could be an element of self-selection in the sample, since those participating in online studies might be more adept at computing than the general populace.

3.6.3 Data Collection & Processing

We measured CL, performance, learning efficiency, and motivation by creating seven surveys and instrumenting the learning system to: 1) record puzzle play duration; 2) trace each

block moved; 3) calculate score using an algorithm inspired by the longest common subsequence approach described in [124] and detailed in Section 2.3 that results in higher scores when nearest to the solution. We calculated instructional and performance efficiency using time-on-task and CL for the EI and ET values during training and transfer tasks as described in Section 2.2. Since the data did not exhibit Shapiro-Wilk normality ($p < 0.05$), we used non-parametric statistics, like Kruskal-Wallis H and Mann-Whitney U tests between-subjects, and Wilcoxon signed-rank test within-subjects. For effect sizes, we used guidelines in [174,175].

3.7 Experimental Results

3.7.1 Cognitive Load

As expected, given random condition assignment, we did not find significant differences between conditions in self-reported CL during the pretest. Kruskal-Wallis tests did reveal significant differences between training conditions with moderate effect for IL ($H(8)=26.24$, $p < .001$, $\epsilon^2=.04$), EL ($H(8)=24.15$ $p=.002$, $\epsilon^2=.04$), and overall CL ($H(8)=29.10$ $p < .001$, $\epsilon^2=.05$). Using a Bonferroni-adjusted alpha of .006 (.05/9), we found significant differences between conditions C6 (M=5.6) vs. C9 (M=3.95) $p=.037$, C8 (M=5.76) vs. C9 $p=0.20$, and C1 (M=4.22) vs. C8 $p=0.31$ for IL, C1 (M=3.03) vs. C6 (M=4.19) $p=.015$ for EL, and C1 (M=4.32) vs. C6 (M=5.50) $p=.002$, C1 vs. C8 (M=5.36) $p=.016$, and C3 (M=4.45) vs C6 (M=5.40) $p=0.22$ for overall CL. The comparatively low mean IL value for C9, the control condition, suggests that participants accurately perceived fewer interacting elements in the puzzles focused on the CT concept *sequences* compared to those on *conditionals*. The IL C1 vs. C8 result corresponds with the conditions that provide the most and least scaffolding (C1: 1 palette, no distractors, feedback; C8: multi-palette, distractors, no feedback). The overall CL result for C1 vs. C8 further reveals the impact of these supports and provides evidence partially supportive of **H1**, indicating PPP

learning systems can induce significantly lower IL and CL by activating maximum rather than minimum scaffolding.

Since the remaining noted differences varied on two axes of support each, we ran Mann-Whitney tests on condition sets for each axis and for all conditions compared to the control. We discovered significant differences with small effect related to distractors and feedback, as well as further evidence of the lower IL induced in the control compared to the treatment ($U(N_{\text{control}}=60, N_{\text{treatment}}=564)=19,991, z=2.32, p=.021, r=.09, M_{\text{control}}=3.95, M_{\text{treatment}}=4.92$). Participants who experienced distractors reported significantly higher IL ($U(N_{\text{distractors}}=301, N_{\text{no-distractors}}=263)=46,408, z=3.54, p<.001, r=.15, M_{\text{distractors}}=5.34, M_{\text{no-distractors}}=4.43$), EL ($U(N_{\text{distractors}}=301, N_{\text{no-distractors}}=263)=93,821, z=4.56, p<.001, r=.19, M_{\text{distractors}}=3.95, M_{\text{no-distractors}}=3.13$), and overall CL ($U(N_{\text{distractors}}=301, N_{\text{no-distractors}}=263)=47,874, z=4.30, p<.001, r=.18, M_{\text{distractors}}=5.16, M_{\text{no-distractors}}=4.52$), then those who did not. This suggests distractors may have been perceived sometimes as interwoven into the challenge, and sometimes as hampering focus.

Additionally, participants who experienced feedback reported with small effect significantly lower IL ($U(N_{\text{feedback}}=298, N_{\text{no-feedback}}=266)=34,771, z=-2.52, p=0.012, r=.11, M_{\text{feedback}}=4.60, M_{\text{no-feedback}}=5.27$), GL ($U(N_{\text{feedback}}=298, N_{\text{no-feedback}}=266)=35,512, z=-2.14, p=0.033, r=.09, M_{\text{feedback}}=5.91, M_{\text{no-feedback}}=6.30$), and overall CL ($U(N_{\text{feedback}}=298, N_{\text{no-feedback}}=266)=33,664, z=-3.09, p=.002, r=.13, M_{\text{feedback}}=4.62, M_{\text{no-feedback}}=5.12$) than those who did not. The lower IL value indicates correctness feedback might reduce the number of interacting elements perceived, due to the focus driven by prompt feedback on each block move. The lower GL value, while attractive, could represent risk that learners lean on feedback without sufficiently learning *conditionals* to use them in feedback's absence. We did not, however, find

the lower GL negatively affected efficiency in the upcoming efficiency analysis in Section 3.7.3. Combined, these training CL results provide only partial support for **H1**, as evidence suggests training without distractors and with feedback produces the lowest CL (**F1**); we found no significant difference from varying presentation (1-palette vs. multi-palette).

During the posttest, there were no significant differences between individual conditions, but participants who trained with distractors reported with small effect significantly higher IL ($U(N_{\text{distractors}}=301, N_{\text{no-distractors}}=263)=43,812, z=2.19, p=.028, r=.09, M_{\text{distractors}}=4.30, M_{\text{no-distractors}}=3.80$), and EL ($U(N_{\text{distractors}}=301, N_{\text{no-distractors}}=263)=44,431, z=2.52, p=.012, r=.11, M_{\text{distractors}}=2.57, M_{\text{no-distractors}}=2.26$) than those who did not. These results could indicate less schema construction occurred during training for distractor than no-distractor participants.

3.7.2 Performance

Though we did not find significant training performance differences across all conditions when measuring the number of puzzles solved correctly, the time needed to solve them was significantly different with moderate effect ($H(8)=87.94, p<.001, \epsilon^2=.14$). After the Bonferroni adjustment, we found participants in the control on *sequences* completed training significantly faster than those in each *conditionals* condition while solving nearly the same number of puzzles correctly ($M_{\text{control}}=2.0, M_{\text{treatment}}=1.9$). Since solving the four *sequences* puzzles requires a minimum of 41 block moves compared to 33 for the *conditionals* puzzles, this result suggests the complexity of the CT concept and puzzle, rather than the length of the solution, affect time spent solving.

Other condition pairs with significant differences in time spent are: C1 vs. C4 ($p=.000$), C1 vs. C7 ($p=.040$), C4 vs. C5 ($p=.000$), and C5 vs. C7 ($p=.012$). Participants in C1 and C5, both with an isolated palette and without distractors, needed less time than participants in C4 with

multiple palettes and distractors, and C7, with multiple palettes ($M_{C1-time}=16.6m$, $M_{C5-time}=16.1m$, $M_{C4-time}=20.8m$, $M_{C7-time}=22.9m$). However, C4 and C7 participants solved more puzzles correctly than those in C5, and C7 participants outperformed those in C1 ($M_{C1-correct}=1.9$, $M_{C5-correct}=1.5$, $M_{C4-correct}=1.8$, $M_{C7-correct}=2.7$). To further investigate these correctness differences, we removed the control from the between-conditions calculations, and found a significant difference with moderate effect across just treatment conditions ($H(7)=91.70$, $p=.000$, $\epsilon^2=.16$). The condition-pair result remaining significant after Bonferroni adjustment is C5 vs. C7 ($p=.020$). This finding suggests that when the presentation varies while distractors and feedback variation are held constant (deactivated), multi-palette participants solve more puzzles correctly than those with an isolated palette. However, we note that despite the random condition assignment, substantially fewer completed the experiment in C7 compared to the mean across conditions ($M_{C7}=37$, $M_{C1-9}=69$). This discrepancy represents a risk to internal validity, as it is possible some participants found the lack of support in C7 too challenging, and dropped their participation, leaving in the condition population a more CT-adept assembly.

We further pursued this finding via Mann-Whitney tests on the group of conditions with an isolated palette vs. the group with multiple palettes, but did not quite find a significant correctness difference ($U(N_{1-palette}=317, N_{multi-palette}=247)=35,609$, $z=-1.89$, $p=.059$, $r=.08$, $M_{1-palette}=1.8$, $M_{multi-palette}=2.0$). We did, though, see significant differences with small effect in the time solving between these two groups, with those with multiple palettes spending 13% more time than those with an isolated palette ($U(N_{1-palette}=317, N_{multi-palette}=247)=31,545$, $z=-3.96$, $p<.001$, $r=.17$, $M_{1-palette}=17.4m$, $M_{multi-palette}=19.7m$). A similar time-spent difference exists between the groups of conditions with distractors vs. those without, with distractor participants spending 7% more time than those without distractors ($U(N_{distractors}=301, N_{no-$

distractors=263)=45,798, $z=3.22$, $p=.001$, $r=.14$, $M_{\text{distractors}}=19.0\text{m}$, $M_{\text{no-distractors}}=17.8\text{m}$), but differences in correctness are limited between these groups, with participants without distractors solving marginally more puzzles correctly on average in less time.

During the transfer phase, participants in each treatment condition solved more posttest than pretest questions correctly. As a full treatment population, they did so at a significant level with a small effect within-subject per a Wilcoxon test ($z=3.4$, $p<.001$, $r=.15$, pretest: $M_{\text{treatment}}=7.8$, posttest $M_{\text{treatment}}=8.0$). We found the largest increase in pre/posttest scores of $M=.4$ from participants who trained in C3 (multi-palette, no distractors, with feedback), indicating that goal-driven search through multiple palettes and guidance from correctness feedback might lead to the greatest performance increase in multiple-choice transfers tasks. However, the difference in pre/posttest scores between treatment conditions was not significant ($H(7)=2.66$, $p=.92$, $\epsilon^2=.00$), meaning further experimentation would be required to vet that possibility. The lack of transfer performance disparity between PPP conditions generally replicates findings in [72,178], and is similar to findings on PPP inter-problem and intra-problem adaption in [74].

3.7.3 Efficiency

Given the training duration differences between the *sequences* and *conditionals* puzzles, and the dependency on standardized values in the EI, ET, and P instructional efficiency (IE) and performance efficiency (PE) calculations introduced in Section 2.3, we focus efficiency analysis on differences discovered between treatment conditions. Though we did not find any significant PE differences, when using time spent as an estimate of mental effort during training, we found a significant difference with a moderate effect in IE ($H(7)=26.32$, $p<.001$, $\epsilon^2=.04$), with condition pairs C1 vs. C4 ($p=.003$) and C4 vs. C5 ($p=.001$) remaining significant after the Bonferroni

adjustment. This indicates that C5 and C1, with the two highest IE values ($M_{C5}=.25$, $M_{C1}=.21$), and each with an isolated palette and no distractors, led to more efficient learning than C4 ($M_{C4}=-.31$), with multiple palettes, distractors, and feedback. This result provides partial support for **H3**. The low mean IE value for C7 ($M_{C7}=-.59$) suggests that training with multiple palettes without distractors and without feedback also degrades learning efficiency, though the C5 vs. C7 ($p=.102$) and C1 vs. C7 ($p=.307$) pairwise comparisons were not significant, due to the smaller participant population in C7 noted in Section 3.7.2. Nonetheless, these results offer evidence that multi-palette PPPs decrease IE compared to isolated-palette PPPs.

We also compared groups of conditions along the axes of variation, which led to no PE findings, but did reveal significant IE differences for each axis. The group of conditions with an isolated palette, with small effect, resulted in significantly higher IE than those with multiple palettes ($U(N_{1\text{-palette}}=317, N_{\text{multi-palette}}=247)=44,503$, $z=2.79$, $p=.005$, $r=.08$, $M_{1\text{-palette}}=.11$, $M_{\text{multi-palette}}=-.13$), which adds support to the IE finding favorable to an isolated palette between individual conditions. We also found, with small effect, significantly higher IE for the no-distractor group compared to the distractor group when measuring mental effort during training both by time ($U(N_{\text{distractors}}=301, N_{\text{no-distractors}}=263)=34,019$, $z=-2.88$, $p=.004$, $r=.12$, $M_{\text{distractors}}=-.06$, $M_{\text{no-distractors}}=.07$) and by CL ($U(N_{\text{distractors}}=301, N_{\text{no-distractors}}=263)=33,229$, $z=-3.29$, $p<.001$, $r=.14$, $M_{\text{distractors}}=-.11$, $M_{\text{no-distractors}}=.13$). Lastly, we analyzed the condition sets with feedback activated vs. deactivated, and found that activated feedback led to substantially higher, but not quite significant, IE when measuring mental effort via CL ($U(N_{\text{feedback}}=298, N_{\text{no-feedback}}=266)=42,923$, $z=1.70$, $p=.080$, $r=.07$, $M_{\text{feedback}}=-.00$, $M_{\text{no-feedback}}=-.08$).

These results indicate IE is higher when training involves an isolated palette, no distractors, and feedback when compared to multiple palettes, distractors, and no feedback. They

nearly fully support **H3**, but since the feedback result did not reach significance, the **F3** finding for maximum efficiency excludes it to focus on an isolated block palette and no distractors. The distractors finding aligns with [72], which similarly found decreased learning efficiency in PPPs with distractors. To the best of our knowledge, PPP learning efficiency of isolated or multiple palettes and feedback activation or deactivation have not previously been reported in the literature.

3.7.4 Motivation

3.7.4.1 Quantitative Results

To analyze motivation quantitatively, we scored the TEQ and calculated within-subject change in programming attitude and CT perception from study start to end. Three participants from the TEQ and seven from the attitude/perceptions tasks were excluded due to participant response insufficiencies ($N_{TEQ}=621$, $N_{attitude/perception}=617$). Though there were not significant differences in TEQ results between conditions for the interest/enjoyment, pressure/tension, and perceived choice subscales, there were with moderate effect for the perceived competence subscale ($H(8)=24.78$, $p=.002$, $\epsilon^2=.04$). After Bonferroni adjustment, the condition pair remaining significant was C1 ($M=4.65$) vs C6 ($M=3.63$) $p=.018$, meaning when the presentation (isolated palette) and distractor status (deactivated) were held constant, feedback activation produced higher perceived competence, which supports **H2**.

When comparing programming attitude and CT perception change across sets of conditions, we found significant differences in each of the axes of variation. Participants in the condition set with feedback reported with small effect significantly higher increase in the Likert scale response to the statement: *I believe I could successfully learn computational thinking* ($U(N_{feedback}=295, N_{no-feedback}=263)=42,778$, $z=2.17$, $p=.030$, $r=.09$, $M_{feedback}=.27$, $M_{no-feedback}=.10$).

This result supports **H2** and **F2** and might suggest feedback has positive implications for CT self-efficacy. Those in the condition set with an isolated palette with small effect reported significant larger decrease in response: *programming is too time consuming* ($U(N_{1\text{-palette}}=314, N_{\text{multi-palette}}=244)=42,244, z=2.14, p=.033, r=.09, M_{1\text{-palette}}=.32, M_{\text{multi-palette}}=.09$), which reflects the 13% less time spent in training noted in Section 3.7.2. Lastly, those in the condition set without distractors with small effect reported significantly higher increase in response to: *programming is easy to start* ($U(N_{\text{distractors}}=296, N_{\text{no-distractors}}=262)=34,347, z=-2.38, p=.017, r=.10, M_{\text{distractors}}=.61, M_{\text{no-distractors}}=.97$), indicating that distractors induced perceptions of higher difficulty.

Additionally, we found many significant positive attitude changes from study start to end within-subject per condition set; selected results are presented in Table 0.3. Although these results indicate attitude improvement and support **H2**, the absence of longitudinal data represents a threat to internal validity, since we cannot claim the change at study conclusion persists.

Table 0.3: Selected within-subject programming attitude change.
Positive shifts (p), negative shifts (n). *p<.05, **p<.01

Programming is...	Feedback ON	Distractors OFF	1 palette ON
fun (p)	M=.49**	M=0.56**	M=0.4**
enjoyable (p)	M=.59**	M=0.6**	M=0.58**
important to know (p)	M=.33**	M=0.34**	M=0.37**
easy to start (p)	M=.77**	M=0.97**	M=0.79**
too difficult to understand (n)	M=-.58**	M=0.62**	M=0.49**
boring (n)	M=-.26**	M=0.33**	M=0.24**
too time consuming (n)	M=-.21*	M=0.32**	M=0.32**

3.7.4.2 Qualitative Results

To supplement the quantitative results, we sought qualitative feedback by requesting that participants describe their attitude toward programming after the learning experience, as well as their perspective on the puzzle presentation, distractors, and feedback. Especially for those who

self-reported low prior programming experience, we recorded more positive attitudinal outcomes for participants who trained with feedback, supportive of **H2** and **F2**. Several reflected a sense of empowerment: 1) “I believe that programming is a complex topic to begin learning without help, it is simple in its basic form, but I think that with enough practice and will to learn, anyone can learn programming”; 2) “it seems much more accessible now”; 3) “it seems like something that I actually could do if I took the time to learn starting out with the basics it is not only for those who are innately gifted or technologically advanced, I can code as well.” 4) “I always thought that programming was something that no “normal” person could do out of the blue. This proved me wrong.”; 5) “it seems more interesting and maybe more accessible – before it felt as if only “chosen ones” could do it well.”

Others appreciated the motivational and gameful effects: 1) “I think this is a great way to get you excited and interested in programming... this would be even enjoyable for the kids and in the meantime they unconsciously learn coding.”; 2) “The puzzles were actually very enjoyable, it felt like a game; 3) “I loved it. I think I will continue learning more about programming by enrolling in other programming courses.”; 4) “I find it a lot easier right now. I wanted to start it for some time but I’m a little bit lazy but this learning experience might give me the boost to get into it finally”; 5) “I thoroughly enjoyed doing this. I thought it was all going to be white data on black screen. I really liked the step by step colourful simplicity of the learning process.”; 6) “It’s made me want to have another go at it as this was such a good way to learn rather than in lectures I had many years ago where it was just slide after slide and then trying stuff... I think this especially would be a great way to get kids interested in programming in school as it showed how it can be fun and satisfying without being overly complex. I really enjoyed this!”; 7) “The

instant gratification of seeing the products of your efforts on-screen spurs you on”; 8) “It’s very addictive and fun :)”.

For those with multiple palettes, we found more hesitant perspectives: 1) “programming scares me a bit less now”; 2) “I think it was a great but exhausting experience”; 3) “I realized just how much I didn’t know about programming, and how difficult it can be. I would have never guessed I’d struggle as much as I did, but I consider it a valuable experience.” Those who trained with feedback, however, noted its positive effect on motivation and efficiency: 1) “I liked receiving feedback, it was like a teacher guiding me, if it did not verify anything I had to search by myself”; 2) “feedback was good as meant i wasn’t too far gone past a mistake before correcting it”; 3) “i was excited that i got it right and sometime feel not putting enough effort if I got I wrong. So i put more effort in getting it right.”; 4) “On the first puzzle I had some red boxes pop up so I knew which moves I had done wrong and I could change things around until I got confirmation from the feedback that it was correct. It helps to know you’re on the right track otherwise it might feel you’re just endlessly trying things without really knowing if it’s correct or not.”

Those who did not receive feedback reported using more traditional testing techniques: 1) “I tried to verify whether I made the correct move by clicking the green arrow and seeing my animation based on the moves I made.”; 2) “I tried to play the animation with the green flag and see if that gives feedback.” Feedback was not appreciated by all, however, as a small set noted discomfort: 1) “it made me a little anxious when my moves were incorrect.”; 2) “when I received information that my movement was incorrect, I got stressed”; 3) “I was happy when I saw that I made the correct choice, however, when thy system turned to red and it said that something is incorrect I began to feel anxiety and I was angry at myself because I did not know what I did

wrong and I just didn't understand why it's wrong and that bothered me.”; 4) “I was worried and tried to understand what I did wrong that it says it was incorrect and how I can make it work.”

These distressed feedback perceptions will lead us to explore throttled feedback options to limit disruption to those negatively affected.

The motivational impact of distractors was mixed. Some valued the simplicity of an isolated palette and lack of distractors: 1) “Having a restricted amount of pieces to work with makes things easier.”; 2) “I liked not seeing any distractor blocks it made me more efficient and could keep me focused.”; 3) “I would have easily been more confused if additional blocks were presented to me. I am very thankful that wasn't the case for me because I probably wouldn't have accomplished anything.” Others embraced the challenge distractors offered: 1) “Distractors are helpful in my opinion, because once selected it trigger the thought “ok why is this not okay?” and then you would compare it with the other blocks understanding the mistake.”; 2) “If there were only “correct” blocks it would be more easier and this is not good, because to learn programming you don't make only “correct” things, you need to make some mistake to understand what are you really doing.”; 3) “Enabled blocks which weren't part of the puzzle solution helped me judge and differentiate between the relevant conditionals that were required to solve the puzzle.”; 4) “I liked that there were some similar blocks that were not used, because that allowed me to learn about different scenarios where I might prefer one block over another in the future... That way I can simultaneously learn about a few different blocks instead of only the ones I am directly using.”; 5) “I thought that having some “dummy” blocks to throw me off the scent was really effective, and would help a younger learner to focus more on the problem.”

3.7.5 Findings Summary

We conclude the analysis by summarizing findings for each varied PPP element in Table 0.4.

Table 0.4: PPP element variation findings summary. *p<.05, **p<.01

Element	CL	Efficiency	Motivation
1-palette	-	Increase**	Increase**
Distractors	Increase**	Decrease**	Decrease**
Feedback	Decrease*	Increase	Increase**

3.8 Conclusions

Our survey of, and interviews with, grade 6-9 teachers revealed the substantial teacher support necessary to help CT learners in Scratch, as well as their limited prior exposure to, but willingness to try, PPPs. This led us to continue to extend Scratch with gameful PPP functionality focused on individual CT concepts. By varying elements of PPPs across nine conditions in a study of 624 adults, we found PPPs with feedback and without distractors produce lowest CL, PPPs with feedback produce highest CT motivation, and PPPs with an isolated block palette and without distractors produce highest CT learning efficiency. The study analysis offers PPP developers insight to advance efficient CT education for all. These findings were originally published in 2022 at the 24th Australasian Computing Education Conference [201].

While these results expose opportunities to advance CT learning via augmentations to popular block-based programming environments like Scratch, external validity would be strengthened were additional CT concepts studied, and other block-based environments included. We might also strengthen construct validity by evaluating how fading of supportive PPP elements within learning progressions helps learners transition toward deepening CT

understanding in open-ended projects. In Chapter 4, we address these issues while also exploring how block-based environments might further equip PPP content developers with tools to customize and differentiate auto-generated feedback for learners.

Chapter 4: Learning Computational Thinking Effectively

4.1 Motivation

Increasingly, government and education leaders position CS as a foundation useful for learning other disciplines, for empowering active citizenship, and for addressing inequalities [202]. CT [1], whether characterized broadly as inclusive of the material and social elements of participatory computational literacy, or narrowly as the cognitive skills and knowledge necessary to reason effectively with a machine, has mobilized expanded access to CS [203].

Internationally, many schools have introduced compulsory coding (e.g. [14,204]), and the CS exposure movement, inclusive of Code.org's Hour of Code [62] and many more (e.g. [19,20]), reach students from schools that have not. This traction, while important, can limit focus on the sustained activity necessary to learn CT effectively [10].

The emerging discipline of learning engineering, which combines theory with data-driven analysis to develop educational methodologies that produce high-quality learning, offers the CS community an architecture to investigate effective CT learning [205]. This field encourages the instrumentation of learning environments, development of student learning models, and rich personalization supportive of equity. It guides us to design human-computer systems that harness and leverage the codependences between learning system, student, and teacher, to operationalize CT learning for all. This harmony among roles is important during an era of generational interdependency in which K-12 students and teachers alike are learning CT. In the U.S., 20% of CS teachers describe themselves as overwhelmed and 30% as under-qualified [206]. The limited CT understanding can produce gaps in teacher pedagogical content knowledge, and testing-tuned curricula constrain time afforded to CT study [207]. While opportunities exist to infuse CT

across curricula (c.f. [208,209]), especially in STEM subjects [154], efficient and effective CT learning remains crucial for universal access initiatives to succeed.

Though learning CT and coding can be difficult for novices [210,211,212], an approach offering efficiencies involves PPPs, which, as earlier introduced, are program completion exercises that facilitate learners practicing CT by assembling into correct order programming constructs that comprise examples of well-written code, typically focused on a single concept [2]. This approach does not typically offer the learner experience in the broader aspects of participatory computational literacy, however. Like the strategy in [150], in which learners extend their skills from code comprehension to construction by reading examples, viewing animations, and addressing misconceptions before trying PPPs, we propose learners might benefit from PPP practice in which scaffolding fades as CT concept familiarity develops. By decrementing supports provided by puzzle presentation and feedback, we can offer learners direct instruction to start [213], then provide pathways toward open-ended, constructionist CT learning via creation of personally meaningful artifacts [129,214].

To investigate this approach, we ran an online study targeting adults who were randomly selected into one of 12 conditions to learn the CT concept *looping* via seven PPPs. Between conditions, we varied the presentation of programming constructs, the feedback types, and for control conditions, feedback activation or the CT concept. In three conditions, we faded scaffolding as learners progressed by suppressing correctness feedback and/or by presenting additional palette and block options. We investigate three research questions: after at most 56 minutes of puzzle solving, what are the effects of PPP variation on adult learner: **R1**) cognitive load (CL)?; **R2**); learning efficiency?; **R3**) CT motivation? Findings from 579 participants indicate **F1**) PPPs with feedback induce lowest CL; **F2**) an isolated palette, correctness feedback,

and fading correctness feedback increases learning efficiency; **F3**) fading scaffolding can increase CT motivation. We next review related work and the software developed, then the study purpose, summative evaluation, and results, before previewing future work.

4.2 Related Work

Sections 2.2 and 3.2 document PPP strengths, weaknesses, and variants in the research literature, as well as empirical results. Generally, PPPs offer learning efficiency gains because scaffolded support enables students to train in shorter time than via code writing exercises, while performing similarly on transfer tasks using either approach. PPPs also have led to more effective learning gain than alternative formats such as animations and annotated examples [150].

Our PPP integration with Scratch differs from previous PPP implementations by its embrace of economical GBL instructional design, the variability of the scaffolding presented to the learner, and its embedding in a popular block-based environment. By leveraging game thinking [215], a blend of game design [40] with design thinking [216], we facilitate the construction of engaging learning experiences with limited content development investment compared to serious games, as recommended in [217]. Educators can design motivating learning progressions that increase in difficulty while fading scaffolding and assessing implicitly, as in stealth assessment [218], as the learner masters individual CT concepts.

Like most PPPs, our implementation provides feedback as the learner positions programming constructs. We extend this with the configurability of per-construct points, a gameful scoring algorithm inspired by the longest common subsequence strategy in [124], target minimal moves needed to solve each puzzle alongside a move counter, and the inclusion of multiple feedback types. In addition to correctness feedback, we enable the customization of

messages and actions triggered when certain puzzle solution conditions are (or are not) satisfied. This facilitates the design of intermediary objectives within puzzles, similar to subgoal labeling shown to help students solve PPPs [139]. The objectives constructed operate as test cases, like the execution-based approach described in [142] and the auto-grader methodology in [219], though we perform static rather than dynamic analysis as the learner positions each construct, and provide dynamic execution feedback concurrently via auto-execution of gameful animations in the Scratch stage. As noted in Section 2.3 and Section 3.2.3, based on the feedback classification documented in [148], we consider this feedback as constructivist because it is problem- and instance-oriented, which is a category of assistance associated with significantly lower student failure rates than alternatives, like solution- and theory-oriented.

We fortify the GBL features with those enabling the instructor to vary scaffolding, similar to the text/example/bug modulation in the BOTS study in [196]. Like text-based PPPs, the instructor can assemble constructs in an isolated palette for use during puzzle play. Alternatively, she can design a PPP requiring navigation through existing Scratch palettes (e.g. Control, Operators). Since the optionality might overwhelm, in a manner historically similar to sp/k [190] and more recently like [220,221], the instructor can select which blocks to enable in each palette, and further, which palettes to enable, as introduced in Chapter 3. This configurability facilitates the design of progressions in which learners encounter an isolated palette before transitioning to increasing numbers of palettes with increasing numbers of blocks enabled. This scaffolding fading can occur in the feedback dimension separately or simultaneously, as the instructor can also vary the correctness and objective feedback activation. As in [222], in which PPPs are used in exercises 1-2 and code writing is used in 3-4, the

scaffolding variability allows for the introduction of CT concepts with constraints before relaxing those as learning proceeds to afford learners increasing agency.

Since the PPPs are in Scratch, learners who develop cognitive CT mastery as scaffolding fades become familiarized with the UI, enabling them to advance to epistemologically pluralistic elements of the curriculum in which they can use CT for open-ended creation [223]. Results from the study described in Chapter 3 indicated that the constraint of an isolated palette can increase learning efficiency and motivation, and the inclusion of correctness feedback can increase motivation. In the study described in this chapter, we introduce new instructional design and feedback capabilities, and explore how CL, performance, learning efficiency, and motivation are affected by scaffolding variation, with an aim to explore the transition to effective longer-term open-ended learning.

4.3 Implementation

To investigate **R1-R3**, we developed an instructional design module using Blockly [61], and integrated the functionality with earlier modifications to Scratch that enable the design, play, and assessment of PPPs. While implementations described in Sections 2.3 and 3.3 introduced block points, scoring, progress bars, configurable blocks and palettes, and correctness feedback, the learning system could not fade feedback. Though feedback increased motivation in earlier studies, some participants noted anxiety, stress, worry, and anger when informed of mistakes move-by-move [201]. To bridge the gap between correctness feedback and none except the animation, we defined a novel grammar and an objective editor for instructors to devise objectives with associated feedback messages and actions for each puzzle. Grammar details are available in this article's supplement site: <https://bit.ly/3wdCowW>.

Since teachers learn CT as they infuse it across subjects [52], we designed the objective editor to be simple to use and include it in the concrete architecture in Section 1.7.2 as a module exposed only to instructors. We frame it as a tool situated within teachers' practice intended to promote teacher learning, in the spirit of educative curriculum materials [224]. The teacher should use pedagogical content knowledge to design useful objectives, such as the timely identification of repeating sequences and loop sentinels.

For example, consider a case in which a teacher introduces the CT concept *looping* to her students, after they have mastered *sequences*. If the puzzle designed requires a sprite to move in a path of a square, a student might begin by selecting a move-steps block, with the intention to later include three additional move-steps blocks, and four turn-left blocks, as shown in Figure 4.1a. However, the teacher can configure the puzzle to exclude the move-steps block initially, and use the objective editor to author guidance for the student to identify a repeating pattern she could embed in a control construct. Figure 4.1b presents an objective that might instead lead to the solution depicted in Figure 4.1c. First, the objective in Figure 4.1b specifies that until the student uses the repeat block, she will receive feedback guiding her toward pattern identification; second, it records the feedback she will receive once she uses the repeat block; third, it designates three actions that will occur once the repeat is used: the move-steps and turn-left blocks will become enabled, and five points will be added to the student's objective score. An example of objective feedback during puzzle play in this study is shown in the top-right of Figure 4.1d.

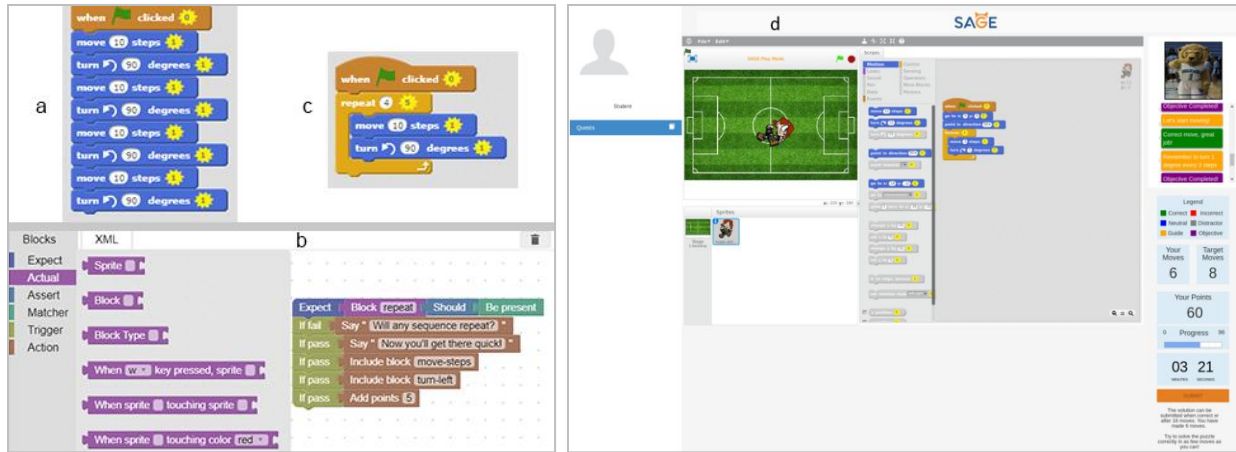


Figure 4.1: Examples of objective configuration and puzzle play experience.

4.4 Summative Evaluation

The software developed prepared us to investigate scaffolding variation. One study purpose was to identify which scaffolding approaches produce the lowest CL and most efficient CT learning. Given the limited skill increases over time and misconceptions developed by Scratchers discussed in Section 2.1, the argument regarding the difficulties with minimally guided instruction in [71] appears applicable to CT learning in Scratch. By augmenting that learning experience with presentation and feedback modifications and comparing the outcomes, we aimed to contribute insight into the types of guidance that might help accelerate the proliferation of CT learning for the general populace.

A second study purpose was to explore the motivational impact of CT learning. In an era in which negative perceptions of CS persist [225], self-directed learning in older adults is rising [226], and deficits in adult CS knowledge limit parental involvement in children’s CT learning [227], insight into the attitudinal change provoked by a short CT learning intervention could inform future outreach. Our study offers insight by examining whether the strategy of fading scaffolding leads to higher CT motivation. While a common approach, high-to-low scaffolding has not always yielded optimal outcomes, particularly in research on preparation for future learning [228]. For

example, [202] found that open-ended instruction followed by detailed guidance led to higher learner activation in subsequent activities than the converse, indicating a carryover effect referred to as epistemological persistence.

We include 11 conditions on the CT concept *looping*, plus one concept control on *sequences*, and analyze the variation of presentation and feedback in seven puzzles, each time-boxed for eight minutes, that connect via a narrative with explicit goals related to playful animations. Based on the results described in Section 4.2, our hypotheses associated with **R1-R3** were: **H1**) PPPs with feedback yield lowest CL; **H2**) an isolated palette yields highest learning efficiency; **H3**) fading scaffolding yields highest CT motivation.

4.5 Experimental Results

4.5.1 Study Design

To test these hypotheses, we developed a quantitative experiment between-subjects, with some within-subject measurements. A study of perceived preferences for game elements in learning environments [229], as well as the review in [230] of the use of game elements for learning programming, inspired us to conduct a similar review of PPP elements in the PPP literature. However, we found limited research distinguishing which PPP elements prove most useful for learning (c.f. [74]). Consequently, we devised 12 study conditions detailed in Table 4.1 to ensure we could granularly analyze the effect of various PPP elements on learning outcomes and help fill this gap in the literature. As documented in Table 4.2, participants followed a 10-step protocol which in part required them to respond to the validated: 1) CS CLCS introduced in Section 2.6.1; 2) intrinsic motivation TEQ introduced in Section 2.6.1; 3) Computing Attitudes Survey (CAS) [231] (details at <https://bit.ly/3wdCowW>).

Table 4.1: PPP scaffolding variation across 12 study conditions.

Cond.	CT Concept	Presentation	Feedback	Fading Scaffolding
C1	Looping	1-palette	Correctness	No
C2	Looping	Multi-palette	Correctness	No
C3	Looping	1-palette	Correctness + Objectives	No
C4	Looping	Multi-palette	Correctness + Objectives	No
C5	Looping	1-palette	Objectives	No
C6	Looping	Multi-palette	Objectives	No
C7	Looping	1-palette	3 Correctness, 2 Correctness + Objectives, 2 Objectives	Correctness feedback faded
C8	Looping	Multi-palette	3 Correctness, 2 Correctness + Objectives, 2 Objectives	Correctness feedback faded
C9	Looping	4: 1-palette, 3: multi-palette	3 Correctness, 2 Correctness + Objectives, 2 Objectives	Correctness feedback & 1-palette faded
C10	Looping	Multi-palette	None	No
C11	Looping	Multi-palette + distractors	None	No
C12	Sequences	1-palette	Correctness	No

Table 4.2: Study protocol and data collected.

#	Activity	Content	Data Collected
1	Registration	Credentials creation & condition assignment	Username & password
2	Background info	Demographic	Demographic, programming attitude, CAS, CT perceptions
3	Tutorial	8-minute video on the learning system & <i>looping</i>	N/A
4	Pretest (isomorphic)	7 multiple-choice (4-choice) <i>looping</i> questions	Pretest responses & score
5	CS CLCS	10 CL questions with 0-10 scaled responses	Pretest CL & IL/EL/GL components
6	Puzzles	7 puzzles on <i>looping</i> in 11 of 12 conditions; learning system behavior varies by condition; 7 puzzles on <i>sequences</i> in the 12 th condition (control)	Per-puzzle time spent, time-stamped block moves & score, correctness, feedback log, self-explanations
7	CS CLCS	10 CL questions with 0-10 scaled responses	Puzzle CL & IL/EL/GL components
8	Posttest (isomorphic)	7 multiple-choice (4-choice) <i>looping</i> questions	Posttest responses & score
9	CS CLCS	10 CL questions with 0-10 scaled responses	Posttest CL & IL/EL/GL components
10	Concluding measurements	Motivation, programming attitude, learning system feedback, CT perceptions	TEQ, CAS, & programming attitude, CT perceptions

4.5.2 Participants

Using Prolific [200] we recruited 579 participants with varying degrees (57% high school, 27% undergraduate, 16% graduate), and a variety of self-reported programming experience (low: 50%; medium: 36%; high: 14%). 405 men, 167 women, and 2 non-binaries comprise the population sourced from 28 countries led by Poland (21%), Portugal (14%), and the U.K. (14%). Since the software development involved Scratch 2.0, which depends on Flash, a technology sunset at the start of 2021, learners needed to download a virtual machine we equipped to bypass Flash disablement. While necessary, this requirement introduced a risk to the

external validity of the sample general population recruited after the sunset date, as participants included only those sufficiently capable of installing virtualization software.

4.5.3 Data Collection & Processing

We created seven surveys and instrumented the learning system to: 1) record puzzle play duration; 2) trace each block moved; 3) calculate score using the algorithm described in Section 2.3 that results in higher scores as construction nears the solution. To quantify the effect on CT learning, we also recorded self-reported CL, similar to the approach used in the studies described in Chapters 2 and 3.

To account for learners who compensate for an increase in CL by committing more mental effort, resulting in constant performance while load varies, we again calculate instructional and performance efficiency (IE: learning process, PE: learning outcome) [145]. Previous studies have found lower PE for PPPs with randomly distributed distractor blocks compared to PPPs [72]; higher IE for PPPs than for writing code (see study described in Chapter 2); and higher IE for PPPs with one palette compared with multiple with distractors (see study described in Chapter 3). We calculated IE and PE using both time and CL during training and transfer tasks. As in earlier studies, since the data did not exhibit Shapiro-Wilk normality ($p < .05$), we used non-parametric statistics, like Kruskal-Wallis H and Mann-Whitney U between-subjects, and Wilcoxon within-subjects. For effect sizes, we used values in [174].

4.6 Analysis & Results

4.6.1 Cognitive Load

We did not find significant differences between conditions in self-reported CL during the pretest. Likewise, the posttest yielded no significant CL differences, suggestive of the acquisition of cognitive structures of equivalent expertise [145]. However, we did find significant

differences between training conditions with moderate effect for GL ($H(11)=26.08$, $p=.006$, $\epsilon^2=.05$). Using a Bonferroni-adjusted alpha of .004 (.05/12), significant difference remained ($p=.019$) between conditions C4 ($M=4.90$) vs. C11 ($M=5.27$). This indicates participants training with multiple palettes and both correctness and objective feedback required less mental effort to contend with instructional features necessary for schema construction than those who received no feedback while navigating multiple palettes that included distractor blocks not part of the puzzle solution. To isolate the mediating variables, we analyzed treatment condition sets grouped by scaffolding variation and found a significant difference in GL with small effect between sets of conditions that did or did not receive objective feedback ($U(N_{objectives=359}, N_{no-objectives=200})=29,307$, $z=-3.60$, $p<.001$, $r=.15$, $M_{objectives}=4.78$, $M_{no-objectives}=5.66$). We also found with small effect significant (GL) and moderate (overall CL) differences between those who received any feedback and those who did not ($U(N_{feedback=461}, N_{no-feedback=98})=18,538$, $z=-2.79$, $p=.005$, $r=.12$, $M_{feedback}=4.95$, $M_{no-feedback}=5.78$; $U(N_{feedback=461}, N_{no-feedback=98})=19,860$, $z=-1.88$, $p=.060$, $r=.08$, $M_{feedback}=4.65$, $M_{no-feedback}=4.97$). These results offer evidence that training with feedback limits the GL and CL experienced by the learner, supportive of **H1**.

4.6.2 Performance

During the transfer phase, the treatment population solved significantly more posttest than pretest questions correctly with small effect ($z=3.4$, $p<.001$, $r=.14$, $M_{pretest}=6.0$, $M_{posttest}=6.25$). Like in [72,74] and the studies described in Chapters 2 and 3, we did not find transfer performance disparity between PPP conditions. During training, however, we found significant differences between conditions in the time spent solving with relatively strong effect ($H(11)=39.29$, $p<.001$, $\epsilon^2=.20$), and with moderate effect, block moves made ($H(11)=113.8$, $p<.001$, $\epsilon^2=.07$) and puzzles solved ($H(11)=74.46$, $p<.001$, $\epsilon^2=.13$). Notable time-saving

conditions, both with an isolated palette and feedback, are C1 (M=21.6) and C7 (M=20.8), which required significantly less time than C10 (M=26.7, C7 vs C10: $p=.033$) and C11 (M=29.0, C1 vs C11: $p<.001$) with multiple palettes and no feedback. Examining conditions sets led to significant differences with small effect between sets with an isolated palette and multiple palettes ($U(N_{1\text{-palette}}=211, N_{\text{multi-palette}}=348)=28,926, z=-4.21, p<.001, r=.18, M_{1\text{-palette}}=23.0, M_{\text{multi-palette}}=2.2$), with and without correctness feedback ($U(N_{\text{correctness}}=359, N_{\text{no-correctness}}=200)=29,188, z=-3.67, p<.001, r=.16, M_{\text{correctness}}=24.0, M_{\text{no-correctness}}=26.7$), and with and without fading scaffolding ($U(N_{\text{fading}}=155, N_{\text{no-fading}}=404)=27,032, z=-2.50, p=.012, r=.11, M_{\text{fading}}=23.5, M_{\text{no-fading}}=25.5$). This indicates an isolated palette, correctness feedback, and fading scaffolding lead to less training time.

Condition sets with the lowest number of block moves were those with an isolated palette and with objective feedback ($U(N_{1\text{-palette}}=211, N_{\text{multi-palette}}=348)=31,653, z=-2.73, p=.006, r=.12, M_{1\text{-palette}}=78, M_{\text{multi-palette}}=84; U(N_{\text{objectives}}=359, N_{\text{no-objectives}}=200)=20,825, z=-8.24, p<.001, r=.35, M_{\text{objectives}}=74, M_{\text{no-objectives}}=95$). Those without objectives solved significantly more puzzles correctly with small effect, as did those with correctness feedback and fading scaffolding ($U(N_{\text{objectives}}=359, N_{\text{no-objectives}}=200)=28,819, z=-3.91, p<.001, r=.17, M_{\text{objectives}}=2.6, M_{\text{no-objectives}}=3.3; U(N_{\text{correctness}}=359, N_{\text{no-correctness}}=200)=42,220, z=3.49, p<.001, r=.15, M_{\text{correctness}}=3.1, M_{\text{no-correctness}}=2.5; U(N_{\text{fading}}=155, N_{\text{no-fading}}=404)=37,576, z=3.70, p<.001, r=.16, M_{\text{fading}}=3.4, M_{\text{no-fading}}=2.7$). These results indicate that objective feedback leads to the fewest puzzles solved and correctness feedback and fading scaffolding lead to the most.

4.6.3 Efficiency

While we did not find significant PE differences between treatment conditions, when using time spent as an estimate of mental effort, we found significant IE differences with

moderate effect ($H(10)=27.73$, $p=.002$, $\epsilon^2=.05$), with condition pairs C1 vs. C11 ($p=.034$) and C7 vs. C11 ($p=.002$) remaining significant after Bonferroni adjustment. This indicates that training with an isolated palette with correctness feedback ($M_{C1}=.29$) and more so with an isolated palette with correctness feedback fading ($M_{C7}=.46$) leads to more efficient CT learning than with multiple palettes without feedback and with distractors ($M_{C11}=-.34$). When grouping condition sets by number of palettes, inclusion of correctness feedback, and fading scaffolding, we identified with a small effect significant differences for the first two and a moderate difference for the third ($U(N_{1\text{-palette}}=211, N_{\text{multi-palette}}=348)=41,474$, $z=2.57$, $p=.010$, $r=.11$, $M_{1\text{-palette}}=.15$, $M_{\text{multi-palette}}=.09$; $U(N_{\text{correctness}}=359, N_{\text{no-correctness}}=200)=40,025$, $z=2.25$, $p=.024$, $r=.10$, $M_{\text{correctness}}=.06$, $M_{\text{no-correctness}}=.11$; $U(N_{\text{fading}}=155, N_{\text{no-fading}}=404)=34,279$, $z=1.74$, $p=.082$, $r=.06$, $M_{\text{fading}}=.11$, $M_{\text{no-fading}}=.04$). These results support **H2** and reveal additional learning efficiency opportunities via the use and fading of correctness feedback in addition to an isolated palette.

4.6.4 Motivation

4.6.4.1 Quantitative Results

To analyze motivation quantitatively, we scored the TEQ, and calculated the within-subject change in CAS scores, programming attitude, and CT perception, from study start to end. Condition sets with significant findings with small effect included those that varied by objective feedback for the interest/enjoyment subscale ($U(N_{\text{objectives}}=359, N_{\text{no-objectives}}=200)=94,948$, $z=-3.13$, $p=.002$, $r=.13$, $M_{\text{objectives}}=3.73$, $M_{\text{no-objectives}}=4.2$) and by objective and correctness feedback for the perceived competence subscale ($U(N_{\text{objectives}}=359, N_{\text{no-objectives}}=200)=32,802$, $z=-1.79$, $p=.074$, $r=.08$, $M_{\text{objectives}}=2.80$, $M_{\text{no-objectives}}=3.15$; $U(N_{\text{correctness}}=359, N_{\text{no-correctness}}=200)=40,229$, $z=2.30$, $p=0.21$, $r=.10$, $M_{\text{correctness}}=3.06$, $M_{\text{no-correctness}}=2.70$). Though narrowly missing significance, the objective feedback condition sets also showed noteworthy differences in the

pressure/tension subscale ($U(N_{\text{objectives}}=359, N_{\text{no-objectives}}=200)=39,639, z=1.94, p=.052, r=.08, M_{\text{objectives}}=3.74, M_{\text{no-objectives}}=3.46$). These results indicate correctness feedback increases perceived competence, while objective feedback reduces interest/enjoyment, lowers perceived competence, and increases pressure/tension.

From the CAS scoring, we found significant within-subject change for the factors of: 3) importance, 4) problem solving – strategies, and 6) personal interest for the treatment population (3: $z=-5.3, p<.001, r=.22, M_{\text{start}}=3.94, M_{\text{end}}=3.76$; 4: $z=5.62, p<.001, r=.24, M_{\text{start}}=3.44, M_{\text{end}}=3.56$; 6: $z=4.28, p<.001, r=.18, M_{\text{start}}=3.42, M_{\text{end}}=3.61$). This decrease in importance and increase in problem solving strategies and personal interest largely held across conditions. For the gender equity (2) and gender bias factors (5), however, only the conditions with fading scaffolding yielded significant differences (2: $z=2.35, p=.019, r=.10, M_{\text{start}}=4.47, M_{\text{end}}=4.53$; 5: $z=-1.97, p=.049, r=.08, M_{\text{start}}=2.01, M_{\text{end}}=1.93$), indicating an attitudinal improvement in gender equity and reduction in gender bias.

Additionally, we found many significant programming attitude and CT perception improvements from study start to end per condition set; selected results are presented in Table 4.3. Overall, the quantitative motivation results provide partial support for **H3**, as fading scaffolding led to improved programming attitude and CT perception scores, but the lack of longitudinal data represents a threat to internal validity, since we cannot confirm the change measured at study conclusion persists.

Table 4.3: Within-subject attitude and CT perception change. *p<.05, **p<.01

Programming is...	Feedback ON	Distractors OFF	1-palette ON
fun	M=.71**	M=.65**	M=.64**
enjoyable	M=.82**	M=.69**	M=.53*
easy to start	M=.94**	M=.67**	M=.73**
CT Perception			
I would recommend children in my family attend a CT camp or after-school program	no significant change	M=.35**	M=.42*
I would ask an employer for CT training	M=.31*	M=.39**	M=.71**

4.6.4.2 Qualitative Results

We studied motivation qualitatively by requiring that participants describe their attitude toward programming at study-end, as well as their perspectives on presentation and feedback scaffolding. Those with low self-reported prior programming experience often reflected surprise:

1) “It is perhaps easier than I thought once you get to grips with how it works and the different ideas and ways of doing it, and it is more accessible to the average person than I thought.”; 2)

“Starting it is much easier than I thought... it can be more fun after this practice I feel more motivated to find some time and finally give it a proper try.” Those with higher prior experience advised on use cases: “this format is good for younger people to learn and get introduced to it. I think is needed to be a class in school about it.”

Those who received correctness feedback reported the utility: “It helped me know when I was straying from course and re-evaluate my decisions. Very important and useful.” Those receiving only objective feedback also perceived value: “The feedback was very motivating for me and each “correct answer” was giving me a lot of joy. I found this very helpful to know if I was heading in the right direction or missing out anything important, and the dynamically enabled blocks helped me make sure I was doing things in the right order.” A C1 participant,

with an isolated palette and correctness feedback, perceived challenge decreasing as she solved puzzles: “At the beginning of the task, I was preoccupied with how to navigate the system, but when I was familiar... I could easily solve the looping puzzles.”

Many responses support **H3**, as they indicate fading scaffolding provides motivating challenge across puzzles, while increasing agency. Several participants in C9, in which both presentation and feedback scaffolding faded, reflected inspiration: 1) “When you start by having a lot of support and then it gets reduced, you need to rely on your skills and understanding to keep learning, and when you see that the program is working fine, it is more rewarding.”; 2) “the ones with the same palette help at a very early stage to familiarize with the concepts, the ones with a different palette broke this familiarity and helps further by making you think more; 3) “The single palette made it easier to focus on how to organize the blocks, instead of wasting time looking for them. Having them easily accessible at first made it easier to find them and use them in the later tasks when the blocks were spread across multiple palettes.” Some with fading feedback, however, noted difficulties: “I found the lack of feedback later on really hard to overcome. It helped me immensely at the beginning. Even though I was more familiar with the concepts and blocks etc, the complexity of puzzles was also increasing so I really struggled.” One participant in C8, with multiple palettes and feedback fading, recommended offering learner-configurability: “Create a settings where you can turn on or turn off the feedback for the puzzle. This way, if the student wants to try solve the puzzle without help, they can turn off the feedback. However, if they find it too difficult and needs help, then they can turn on the feedback for guidance.”

4.6.5 Findings Summary

We conclude the analysis with a summary of key findings in Table 4.4.

Table 4.4: Summary of key findings.

Measurement	PPP elements yielding favorable outcomes
Lowest CL	Feedback included
Increased learning efficiency	Isolated palette, correctness feedback, fading correctness feedback
Increased motivation	Fading correctness feedback and isolated palette presentation

4.7 Conclusions

To illuminate how learning system elements and progressions affect CT learning in block-based environments, we developed Blockly-scaffolded, configurable PPP functionality within Scratch that enables variation of presentation and feedback types. In a between- and within-subjects study of 579 adults, we enumerated several features and scaffolding strategies suitable for a general populace. PPPs with feedback yield lowest CL; an isolated palette, correctness feedback, and fading correctness feedback results in the highest learning efficiency; fading scaffolding can increase CT motivation. The analysis offers PPP developers and instructors insight to advance efficient CT education for all. These findings were originally published in 2022 at the 30th International Conference on Computers in Education [232].

While these results expose opportunities to advance CT learning via augmentations to popular block-based environments, we caution that developing effective technical interfaces between these systems is non-trivial given current implementation stratification. To echo and apply the call for standardization to advance progress in learning engineering in [205], we advocate for CT education data standards supportive of scaling consistent, quality data capture using extendable data models facilitating customization but rooted in commonality. Such anchoring could accelerate efforts to extend functionality in future work, for example by simplifying the way the Blockly objective editor could be integrated with the Scratch PPP

evaluation engine to drive execution-based feedback like the js-parsons method described in [142]. It could also lead to data-driven generation of concept inventories and misconception maps, an approach proposed by EvoParsons developers in [75], that could better equip educators with the operational learning definitions necessary to infuse CT across curricula. We explore some of this forward-looking activity next.

Chapter 5: Future Work

5.1 Gameful Intelligent Tutoring

In Section 1., we hypothesized the affordances of *Gameful Direct Instruction* and *Gameful Constructionism* have potential to incite more efficient and effective CT learning. In field studies described in Chapters 2, 3, and 4, we produced evidence supportive of this hypothesis. However, to deliver these improvements in a scalable way at reasonable cost, we need a cohesive learning environment that enables practicing teachers to utilize the improvements easily. As earlier discussed, due to generational challenges and difficulties with CS teacher certification, many teachers must learn CT alongside their students. This means we need a strategy that roots expertise in the learning environment, so that the teacher can more easily operate as a “guide on the side” rather than as a “sage on the stage.” With further development, we believe a system like SAGE can offer teachers and students this type of community affinity space in which distributed production of content emerges as participants begin to see themselves as “insiders” authoring shared meaning [233].

By augmenting *Gameful Direct Instruction* and *Gameful Constructionism* with intelligent systems, we can provide a *Gameful Intelligent Tutoring* experience that offers a cohesive communal space for instructors and learners to become immersed in CT learning. We believe data-driven methods that require minimal knowledge engineering can enable us to deliver differentiated instruction to each student at reasonable cost without increasing teacher burden. *Gameful Intelligent Tutoring* will extend existing research on ITSs by introducing an across-activity, curriculum-sequencing outer loop that offers game recommendations, and a within-activity, problem-solving-support inner loop that offers hints just-in-time or on-demand. In the

following sections, we introduce these loops and the feasibility of the approach that re-positions teachers as designers and supporters of efficient and effective learning experiences.

5.1.1 Intelligent Tutoring Outer Loop

To implement the curriculum-sequencing outer loop, we will first establish an intelligent tutoring framework via a modular architecture for SAGE that encapsulates subsystems such as modified versions of Scratch and Blockly; software layer and activity flow diagrams are included in Appendix E for reference. The system includes a frontend affinity space that provides authentication and role-based authorization to teacher and student. Among other affordances such as class creation, student enrollment, story composition, gameplay role-provisioning, and instruction authoring, the teacher affinity space enables the creation of three-tier learning progressions in which Games comprise Quests which comprise Missions. The student affinity space offers a similar hierarchy, but for gameplay rather than Mission/Quest creation and game design, game-objective editing, and story and instruction authoring. While we store basic metadata for these hierarchies through a frontend API and Mongo database, the assessment server operates as the primary backend system. Its API offers a variety of crucial functionality, inclusive of gameplay snapshot persistence and static analysis execution, as well as connectivity to data wrangling and machine learning processes for intelligent hinting, which are discussed further in Section 5.1.2.

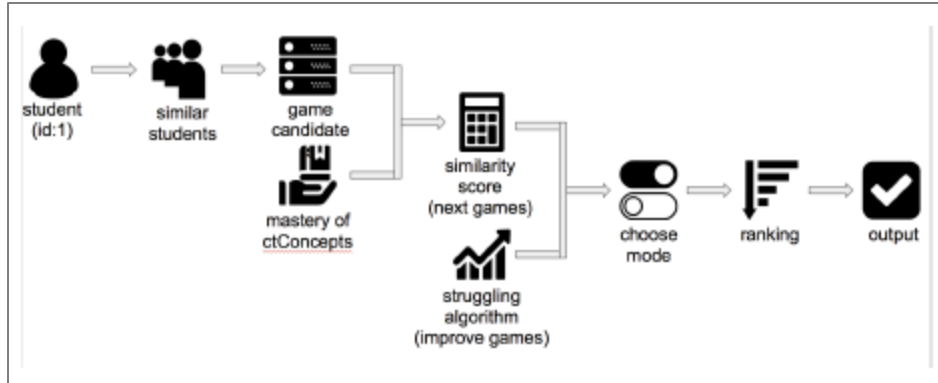


Figure 5.1: Game recommendation process.

The architecture developed will enable us to establish an ITS framework in which we can offer students and teachers across-activity recommendations for next games to pursue or assign. The resulting student-centered, personalized or class-differentiated learning progressions can optimally fill the gaps between the known and unknown when configured to offer additional practice, and can identify incremental learning opportunities when configured to introduce new challenges. Figure 5.1 presents a flowchart of the recommendation engine for the student case, and the teacher flow will be quite similar, with a calculated student representative and performance of the classroom replacing the first two nodes. Overall, the outer-loop will provide the student next-game options tuned to her CT mastery and her learning goals, while providing the teacher with next-game options to add to her Quest tuned to the CT mastery of her class and her teaching goals.

We can accomplish this tuning through multi-criteria collaborative filtering. For the student case, for example, we will first find students similar to student 1 via a calculation that compares student 1 to each other student, across the set of games that each pair has played. This comparison includes several dynamically weighted features, including game difficulty, game score, objective score, as well as CT static analysis scores that will be described further in Section 5.2. We will then select the games student 1 has not yet played from all the games of the

similar student to generate game candidates, and simultaneously compute student 1's highest score across games for each CT concept. If student 1 desires additional practice, for instance in preparation for a quiz, we will activate the struggling algorithm, which identifies both CT concepts in which student 1 has low mastery, and the games that focus on those CT concepts above a configured proportion of all CT concepts, with the proportion assigned as the similarity score for that game. If student 1 desires to learn new material, we will compute the mastery level similarity for each game candidate, with the final similarity score calculated as the average of all the mastery level similarities for each CT concept. The choice between these modes results in a weighting of "improve" and "next" games, meaning desire for practice will lead to the recommendation of more "improve" games, and desire for challenge will lead to the recommendations of more "next" games.

For the teacher case, we will create a representative student by computing the average score of the games played by students in the class, and calculate the average classroom mastery of each CT concept. We will then execute both the struggling algorithm and the mastery similarity calculation on all other games in the system to recommend games the teacher could add to Quests she might assign to students for practice or challenge. With this outer loop ITS functionality, the teacher will be able to optimize game selection for each of her classes, and each student will be able to optimize game selection for herself. Next, we discuss how we will provide inner loop ITS functionality that can enhance problem-solving within the selected games.

5.1.2 Intelligent Tutoring Inner Loop

To provide inner loop problem-solving-support, we will persist granular student gameplay snapshot data so that we can derive student models indicative of programming

behavior, problem-solving progress, and the potential need for guidance. We will save these snapshots by modifying Scratch to post serialized state in JSON to a RESTful endpoint on the assessment server at time-stamped intervals driven by the pace of student activity. Using data extraction utilities described further in Section 5.2, we will distill features most crucial to our analyses: the time-series representation of block placement in the scripts pane. This process evidence of program construction will allow us to identify programming behavior and provide just-in-time and on-demand hints tuned by that behavior and problem-solving progress.

As inspiration for detecting programming behaviors, we will use the classifications delineated by Perkins et al. in a clinical study of novice programmers in the 1980's [234]. These are *stoppers*, who are prone to give up when solving problems, *movers*, who consistently try a variety of approaches without getting stuck, *extreme movers*, who seem not to reflect on their previous ideas and program haphazardly, and *tinkerers*, who make many small changes to code with the hope of finding a solution, but infrequently question their solution strategies. To detect the programming behavior for each student, we distill from the time-series data many primary features such as the interval between two actions, the number of actions in a configurable interval, the interval between two block removal actions, and the number of placement movements for each block in each game. Based on these features, we will further calculate statistical variations, such as the minimum interval between actions and the standard deviation of the interval between actions. We will then use linear discriminant analysis to reduce dimensionality and build classification models using decision tree and k-nearest neighbors algorithms. While these two algorithms have produced higher accuracy (0.83 and 0.82) compared to others such as support vector machines in preliminary testing with mock data generated to represent examples of the four programming behaviors, we expect to revisit the

feature distillation and algorithm choice when working with learner data gathered from the field studies described in Chapters 2, 3, and 4.

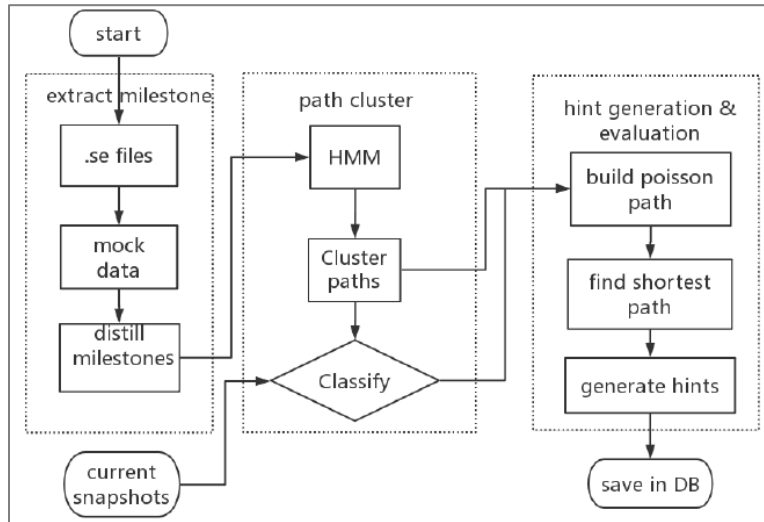


Figure 5.2: Intelligent hinting data flow.

This detected programming behavior will be used as one input into our mechanism for providing intelligent hinting to students potentially in need of individualized assistance during gameplay. As outlined in Figure 5.2, our approach will involve unsupervised identification of game milestones and gameplay paths through those milestones, supervised classification of students by path and programming behavior, and the generation of a Poisson path to help determine the shortest path for a student through the learning graph toward a solution, with hint frequency throttled by programming behavior to support pedagogical efficacy. This strategy is inspired most directly by previous work by Piech et al. [235,115], who autonomously created graphical models of how university students progress through assignments, and later developed a set of algorithms that can predict the way a teacher would guide a student to make forward progress, based on a corpus of student data from a Code.org Hour of Code course. More recent work by Price et al. [236], has used a contextual tree decomposition algorithm to recommend hints based on prior students' actions within the block-based programming environment, Snap!,

with a target of university students in an introductory course for non-CS majors. Since novice block-based programming environments typically involve open-ended assignments, which make providing individualized automated feedback difficult, the CS community has so far devoted limited effort to implement intelligent features in these systems. To the best of our knowledge, these proposed modifications to Scratch would be among the first to adequately position a block-based programming environment for effective intelligent hinting.

To generate useful hints, first we will identify gameplay paths and game milestones by extracting student snapshots. Next, we will treat each student gameplay snapshot as a noisy output of a latent variable, or hidden state, in a hidden Markov model (HMM); for example, these latent milestones might represent the start of the project, or the correct placement of an important `repeat` block. We will then use k-medoids clustering to identify k milestones from a much larger set of snapshots, and parameterize the HMM by the probabilities of a student moving between one state and another and the probability that a particular snapshot started from a particular milestone. To identify common routes through the milestones, we will cluster gameplay paths according to these probabilities, using students' pairwise log-likelihood distance score. We will then use these paths, the student's identified programming behavior, and current gameplay state as inputs for hint generation.

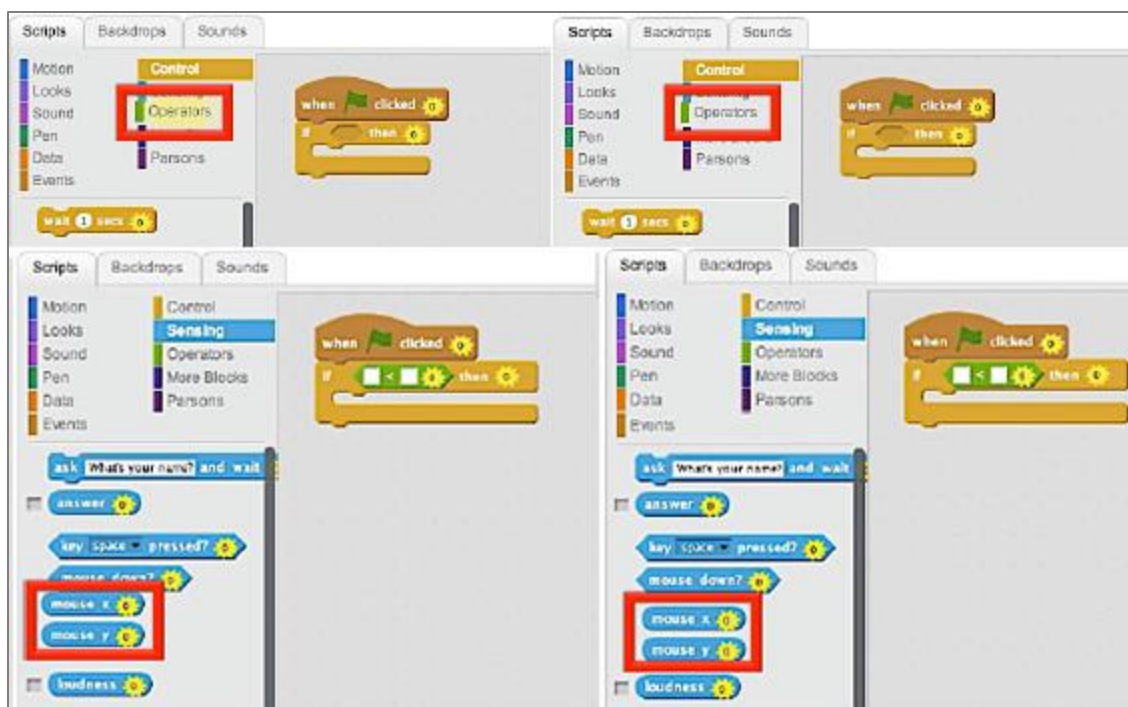


Figure 5.3: Hints delivered through palette highlighting and block shaking.

With an aim to identify blocks a student might try next when stuck, we will model the students' data for a specific HMM cluster and game as a graph. Each node will represent a partial gameplay solution, and each edge will depend on the number of students who have transitioned between two gameplay solutions and reached a successful solution, for example, by adding the same block next given a matching initial state. As in [115], we will define the Poisson path as the path from a partial solution to a correct solution with the smallest expected time to be generated. We will then use Dijkstra's algorithm to find the shortest path from the current snapshot to a correct solution, and output the top three blocks that lead the student along that shortest path. We will then throttle the rate at which we present these blocks as hints, based on the identified programming behavior and teacher-configured settings, to optimize pedagogical effectiveness. As examples, a *stopper* might receive a higher frequency of just-in-time hints than a *mover*, or a teacher might configure the system to provide only on-demand hints, so that students must

proactively choose to receive them, in the case of a formally assigned and assessed game. While we will continue to explore a variety of ways of presenting these hints, including through an avatar driven by chatbot natural language processing techniques, in feasibility implementations, we deliver hints by shaking and highlighting the relevant Scratch palette, and then shaking relevant blocks in the palette selector, as shown in Figure 5.3. With this inner loop ITS functionality, each student will interact with individualized problem-solving feedback while the teacher will monitor class progress and support CT learning when and where needed rather than assuming sole responsibility for class-wide CT delivery and uptake.

5.2 Sustainable Interoperation

The development of the *Gameful Intelligent Tutoring* approach would be a simpler endeavor if novice learning environments for CT adopted standards that facilitated interoperation between them. There are currently consistency deficits across systems in defining CT concept inventories and in modeling student learning. There is also notable dissimilarity among the technologies involved in these systems, as well as in the input/output formats that drive them. As recommended in [205], an ecosystem in which the community focused on creating generally-applicable reusable components could increase the average quality of future learning systems substantially. We are well-positioned to contribute to this area in future work.

To exemplify the need, we illuminate the work entailed in harnessing Scratch for static analysis, which is a technique commonly used for analyzing code without executing it, with the purpose of assisting in promoting code conformance to standard guidelines and identifying code bugs/issues. Generally, tools such as SonarQube, Checkmarx, and Veracode provide simple interfaces to connect most codebases to a wide variety of coding technologies [237,238,239], while offering suites of configurable tests for a variety of analyses, such as security, formatting,

and code complexity. Here, we seek to analyze the code learners input into Scratch via blocks, with the aim of achieving the *Gameful Intelligent Tutoring* capabilities described in Section 5.1.

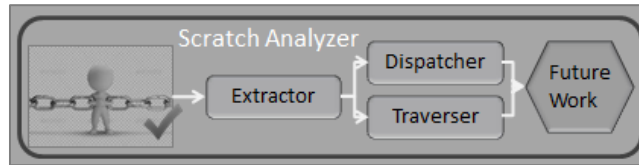


Figure 5.4: Scratch Analyzer components and data flow.

As earlier noted, the JSON gameplay state data are frequently persisted in SAGE for implicit assessment and intelligent tutoring purposes. Many transformations are involved in the data wrangling process, but most of them stem from one simplifying transformation provided by a module of our Scratch Analyzer suite, called Scratch Extractor, which is depicted as the gateway to analysis of SAGE gameplay in Figure 5.4. This utility eliminates irrelevant JSON keys, values, and cluttering syntax, while preserving the sprites and their hierarchical stacks of associated blocks; an example output from Scratch Extractor is depicted in Figure 5.5.

```

whenIReceive
hide
<<Object Snow>>
whenCloned
lookLike:
show
setVar:to:
doUntil
  <
  ypos.
  doIf
  =
  randomFrom:to:
  changeVar:by:
  randomFrom:to:
  heading:
  readVariable
doIf
  >
  readVariable
  setVar:to:
doIf
  <
  readVariable
  setVar:to:
doIf
  >
  xpos.
  Xpos:
doIf
  <
  xpos.
  Xpos:
forward:
  
```

Figure 5.5: Sample Scratch Extractor output.

We can use this output to assess CT learning using evidence-centered assessment design frameworks for Scratch, such as the Progression of Early Computational Thinking (PECT) model, which maps Scratch blocks to evidence variables that roll-up into design patterns and CT concepts [240]. The rubric used to map blocks to evidence variables categorizes proficiency as basic, developing, or proficient; a similar categorization occurs across combinations of evidence variables to determine design pattern proficiency, and ultimately, CT concept proficiency. Although the PECT model was pilot-tested by the original authors with results consistent with expectations, meaning older students achieved higher CT concept proficiencies than younger students, the pilot did not include an automated way of calculating the proficiencies. In SAGE, we can leverage the gameplay state regularly saved to the assessment server to identify the blocks used in each game, and automatically map these blocks to the PECT evidence variables, design patterns, and CT concepts to produce and visualize proficiency scores.

We can further advance automated assessment by integrating a lint-inspired toolset named Hairball that enables automated inspection of Scratch programs [241]. Due to our anticipated addition to the persisted gameplay state of globally unique identifiers with each block for the purpose of aiding the implementation of the intelligent hinting system, we will need to modify Hairball to process Scratch programs that contain a unique identifier associated with each block. Once the blocks are processed, we can leverage an existing Hairball plugin named Mastery, which measures proficiency across seven CT concepts [242]. This plug-in follows a rubric similar to PECT and establishes the same proficiency categories, but also includes functionality that answers a broader range of assessment questions than we can accomplish simply by counting blocks used. For example, we can measure proficiency in parallelization not

only by checking if there are two when-green-flag blocks present, but also by inspecting whether two scripts start executing when receiving the same broadcast message, or when the same background changes, and so forth. We can persist these proficiency scores, and then display them for teachers and students via spider graphs with one series displayed for each game for each student, as shown in Figure 5.6. In combination, the Hairball, PECT, and objective implicit assessments introduced in Section 4.3 will guide students toward CT discovery during gameplay and provide students and teachers real-time, triangulated feedback on CT learning progress. If the CT community were to define a set of standards by which these types of tools could interoperate, the required development efforts for technical feasibility would decrease, which would open the opportunity for further advancement of tooling to advance the teaching and learning experience. Similar opportunities exist for standardizing student modeling, complex competency definitions, and programming behavior intervention strategies.

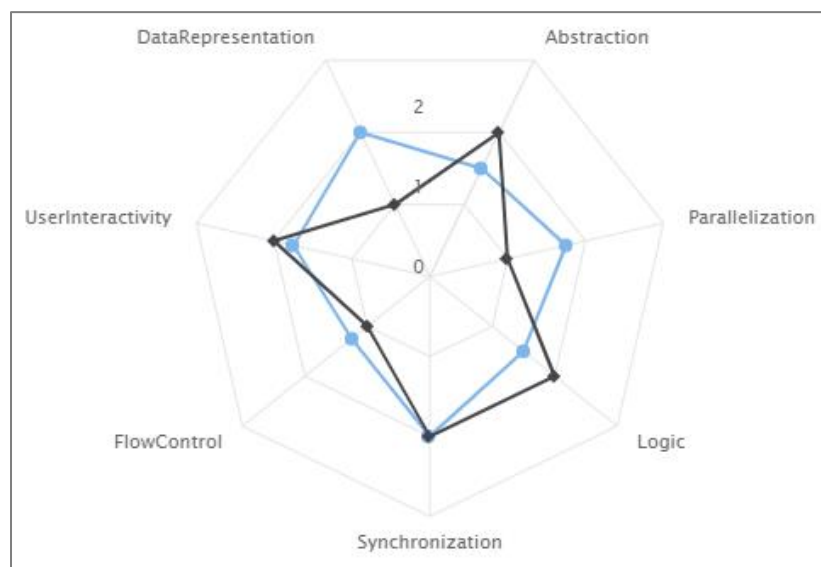


Figure 5.6: Hairball assessment results for two games.

5.3 Educative Curriculum Materials

Improvements in interoperability could also influence the sustainability of thoroughly incorporating educative curriculum materials in learning systems such as SAGE in the future, so that development focus could shift toward reinforcing the uptake and impact of those materials. Designed to support both teacher and student learning, this content could prove a crucial bridge spanning the generational gap earlier discussed that positions teachers without K-12 CT experience, and often with limited training, as teachers of K-12 CT students. While the intelligent systems described in Section 5.1 target learners operating in Scratch, intelligent coaching during objective development in Blockly would aptly serve instructors. Similarly, static analysis of objective suites could provide instructors feedback on the breadth and depth of the coverage of their objective designs, and feed a recommendation engine that steers them toward more complete assessment configuration. The more straightforward addition of content management capabilities throughout the learning system affinity space would also enable experts to seed the CT teaching experience with a knowledge base offered piecemeal just-in-time, while always accessible in entirety on-demand. Such tooling could also equip teachers whose expertise develops to contribute pedagogical content knowledge to the community via a crowdsourcing model, or alternatively, via a monetizable strategy in which teachers can market their puzzles, objectives, and supporting educative curriculum materials to their peers.

Chapter 6: Conclusions

6.1 Contributions

Computing has transformed the world, resulting in demand for CT: a universal fourth foundation beyond reading, writing, and arithmetic. Despite its increasingly widespread acceptance as a crucial competency for all, the commensurate widespread enablement of educational opportunity remains unresolved. The principal contribution of this thesis is that we can improve the efficiency and efficacy of teaching and learning CT by building gameful learning and assessment systems on top of block-based programming environments. By equipping instructors and learners with additional data-driven tooling, we believe this can be accomplished at scale and cost conducive to accelerating CT dissemination for all.

After introducing the requirements, approach, and architecture, we presented a solution named *Gameful Direct Instruction*. This involves the integration of PPPs in Scratch, as described in Chapter 2. PPPs encourage students to practice CT by assembling into correct order sets of mixed-up blocks that comprise samples of well-written code which focus on individual concepts. The structure they provide enables instructors to design games that steer learner attention toward targeted learning goals through puzzle-solving play. Learners receive continuous automated feedback as they attempt to arrange programming constructs in correct order, leading to more efficient comprehension of core CT concepts than they might otherwise attain through less structured Scratch assignments. We measured this efficiency first via a pilot study conducted after the initial integration of PPPs with Scratch, and then again after the addition of scaffolding enhancements included to direct instruction for a larger adult general population, as described in Chapter 3.

We complemented *Gameful Direct Instruction* with a solution named *Gameful Constructionism*. This involves integrating with Scratch implicit assessment functionality that facilitates constructionist video game design and play [4]. CVGs enable learner to explore CT using construction tools sufficiently expressive for personally meaningful gameplay. Instructors are enabled to guide learning by defining game objectives useful for implicit assessment, while affording learners the opportunity to take ownership of the experience and progress through the sequence of interest and motivation toward sustained engagement. When arranged within a learning progression after PPP gameplay produces evidence of efficient comprehension, CVGs amplify the impact of direct instruction by providing the sculpted context in which learners can apply CT concepts more freely, thereby broadening and deepening understanding, and improving learning efficacy. We assessed this efficacy in a study of the general adult population, as described in Chapter 4.

6.2 Findings Summary

The summarized findings from the studies described in Chapters 2, 3, and 4 follow in Tables 6.1, 6.2, and 6.3. They suggest that training with PPPs can induce lower extraneous cognitive load compared to training with PPPs with distractors, or via coding with limited constraint and feedback. Furthermore, PPPs with feedback and without distractors can induce lower overall cognitive load. Educators desiring to create periods of lower intensity in a learning progression can leverage consistent evidence across studies indicating PPPs with feedback offer cognitive load relief without sacrificing learning performance.

The summarized findings also highlight PPPs both with and without distractors induce higher learning efficiency than does coding with limited constraint and feedback. Furthermore, evidence suggests PPPs with an isolated block palette and without distractors can lead to the

highest learning efficiency. By fading the correctness feedback during a learning progression, educators can provide an efficient transition path toward less-structured, open-ended learning.

Lastly, the findings indicate learning motivation increases when training via PPPs or PPPs with distractors compared to coding with limited constraint and feedback. Furthermore, PPPs with feedback induce higher motivation than those without. Educators can sustain high motivation in learning progressions by fading feedback and block presentation scaffolding, which incrementally affords learners increased agency.

Table 6.1: Summary of findings across conditions in study described in Chapter 2.

Condition	Extraneous Cognitive Load	Instructional Efficiency	Motivation/Attitude
PPP	Lower	Higher	Highest
PPD	Higher	Highest	Higher
LCF	Highest	Lower	Lower

Table 6.2: PPP element variation findings summary in study described in Chapter 3.

*p<.05, **p<.01

Element	CL	Efficiency	Motivation
1-palette	-	Increase**	Increase**
Distractors	Increase**	Decrease**	Decrease**
Feedback	Decrease*	Increase	Increase**

Table 6.3: Summary of key findings in study described in Chapter 4.

Measurement	PPP elements yielding favorable outcomes
Lowest CL	Feedback included
Increased learning efficiency	Isolated palette, correctness feedback, fading correctness feedback
Increased motivation	Fading correctness feedback and isolated palette presentation

6.3 Conclusion

Since the *Gameful Direct Instruction* and *Gameful Constructionism* approaches leverage low fidelity yet motivating gameful techniques, they facilitate the development of learning content at scale and cost supportive of widespread CT uptake. In Chapter 5, we glanced ahead toward future work that anticipates further progress in scalability via a solution named *Gameful Intelligent Tutoring*. This involves augmenting Scratch with Intelligent Tutoring System functionality that offers across-activity next-game recommendations, and within-activity just-in-time and on-demand hints. Since these data-driven methods operate without requiring knowledge engineering for each game designed, the instructor can evolve her role from one focused on knowledge transfer to one centered on supporting learning through the design of educational experiences, and we can accelerate the dissemination of CT at scale and reasonable cost while also advancing toward continuously differentiated instruction for each learner. We hope this thesis offers inspiration to those focused on this worthwhile aspiration.

References

- [1] J. Wing. Computational Thinking. *Communications of the ACM* 2006, 49 (3), 33-35.
- [2] D. Parsons, P. Haden. Parson's programming puzzles: a fun and effective learning tool for first programming courses. *Australian Conference on Computing Education*, 2006; pp 157-163.
- [3] J. Maloney, et al. The Scratch programming language and environment. *ACM Transactions on Computing Education* 2010, 10 (4), 1-15.
- [4] D. Weintrop, et al. Computational thinking in constructionist video games. *International Journal of Game-Based Learning* 2016, 6 (1), 1-17.
- [5] S. Papert. *Mindstorms: Children, Computers, and Powerful Ideas*; Basic Books, 1980.
- [6] M. Guzdial, B. du Boulay. The history of computing education research. In *The Cambridge handbook of computing education research*;, 2019; pp 11-39.
- [7] R. D. Pea, et al. Logo and the development of thinking skills. In *Children and Microcomputers: Research on the Newest Medium*; SAGE, 1985; pp 193-317.
- [8] D. M. Kurland, et al. A study of the development of programming ability and thinking skills in high school students. *Educational Computing Research* 1986, 2 (4), 429-458.
- [9] S. M. Carver. *Transfer of logo debugging skill: analysis, instruction, and assessment*; PhD Thesis; Carnegie-Mellon University, 2986.
- [10] P. Blikstein, M. S. Hejazi. Computing education: literature review and voices from the field. In *The Cambridge Handbook of Computing Education Research*;, 2019; pp 56-78.

- [11] J. Wing. Computational thinking and thinking about computing. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 2008, 366 (1881).
- [12] C. Wilson, et al. *Running on Empty: The Failure to Teach K-12 CS in the Digital Age*; CSTA, 2010.
- [13] D. Leyzberg, C. Moretti. Teaching CS to CS teachers: Addressing the need for advanced content in K-12 professional development. *ACM SIGCSE*, 2017; pp 369-374.
- [14] Whitehouse.gov. CS for All.
<https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all>.
- [15] The Royal Society. After the Reboot: Computing Education in UK Schools.
<https://royalsociety.org/~media/policy/projects/computing-education/computing-education-report.pdf>.
- [16] L. Tamatea. Compulsory coding in education: liberal-humanism, Baudrillard and the ‘problem’ of abstraction. *Research and Practice in Technology Enhanced Learning* 2019, 14 (1), 1-29.
- [17] K-12 Computer Science Framework Steering Committee. K-12 Computer Science Framework, 2016. <https://k12cs.org/wp-content/uploads/2016/09/K%E2%80%9312-Computer-Science-Framework.pdf>.
- [18] C. Kelleher, R. Pausch. Lowering the barriers to programming: a taxonomy of programming environments and languages for novice programmers. *ACM Computing Surveys* 2005, 37 (2), 83-137.
- [19] Girls Who Code. Girls Who Code. <https://girlswhocode.com/>.
- [20] Black Girls Code. Women of color in technology. <https://www.blackgirlscode.com/>.

- [21] Code.org. About Us | Code.org. <https://code.org/about>.
- [22] The CSTA Standards Task Force. *K–12 Computer Science Standards*; Computer Science Teachers Association, 2017.
- [23] A. Reppening, et al. Scalable game design and the development of a checklist for getting computational thinking into public schools. *41st ACM technical symposium on computer science education*, 2010.
- [24] V. Barr, C. Stephenson. Bringing computational thinking to K-12: what is involved and what is the role of the computer science education community. *ACM Inroads* 2011, 2 (1), 48-54.
- [25] C. Hu. Computational thinking – what it might mean and what we might do about it. *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, 2011.
- [26] I. Lee, et al. Computational thinking for youth in practice. *ACM Inroads* 2011, 2 (1), 32-37.
- [27] P. Morreale, et al. Measuring the impact of computational thinking workshops on high school teachers. *Journal of Computing Sciences in Colleges* 2012, 27 (6), 151-157.
- [28] D. Barr, et al. Computational thinking: a digital age skill for everyone. *ISTE Learning and Leading* 2011, 38 (6), 20-23.
- [29] A. E. Weinberg. *Computational thinking: an investigation of the existing scholarship and research*; Ph. D. dissertation; Colorado State University, 2013.
- [30] M. Resnick. Sowing the seeds for a more creative society. *Learning & Leading with Technology* 2008, 35 (2), 18-22.

- [31] Y. Kafai, et al. *The Computer Clubhouse: Constructionism and Creativity in Youth Communities*; Teachers College Press, 2009.
- [32] K. Brennan, M. Resnick. New frameworks for studying and assessing the development of computational thinking. *American Educational Research Association*, 2012.
- [33] U.K. Department of Education. National curriculum in England: computing programmes of study. <https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study>.
- [34] J. Lave, E. Wenger. *Situated Learning: Legitimate Peripheral Participation*; Cambridge University Press, 1991.
- [35] F. Garzotto. Investigating the educational effectiveness of multiplayer online games for children. *Proceedings of the 6th International Conference on Interaction Design and Children*, 2007; pp 29-36.
- [36] C. Bachen, C. Raphael. Social flow and learning in digital games: a conceptual model and research agenda. In *Serious Games and Edutainment Applications*; Ma, M. . O. A. . a. J. L., Ed.; Springer, 2011.
- [37] G. Stahl, et al. Computer-supported collaborative learning. In *The Cambridge Handbook of the Learning Sciences*; Sawyer, K., Ed.; Cambridge University Press, 2014.
- [38] A. Collins, M. Kapur. Cognitive Apprenticeship. In *The Cambridge Handbook of the Learning Sciences*; Sawyer, B., Ed.; Cambridge University Press, 2014.
- [39] A. Collins, R. Halverson. *Rethinking Education in the Age of Technology: The Digital Revolution and Schooling in America*; Teachers College Press, 2009.

- [40] K. Salen, E. Zimmerman. *Rules of Play: Game Design Fundamentals*; MIT Press, 2003.
- [41] M. Csíkszentmihályi. *Flow: The Psychology of Optimal Experience*; Harper Perennial: New York, NY, 1990.
- [42] M. P. J. Habgood, et al. Endogenous fantasy and learning in digital game. *Simulation & Gaming* 2005, 36 (4), 483-498.
- [43] S. Järvelä, K. A. Renninger. Designing for learning: interest, motivation, and engagement. In *The Cambridge Handbook of the Learning Sciences, Second Edition*; Sawyer, R., Ed.; Cambridge University Press, 2014.
- [44] A. Przybylski, et al. A motivational model of video game engagement. *Review of General Psychology* 2010, 14 (2).
- [45] C. Kazimoglu, et al. Understanding computational thinking before programming: developing guidelines for the design of games to learn introductory programming through game-play. *International Journal of Game-Based Learning (IJGBL)* 2011, 1 (3), 30-52.
- [46] D. A. Kolb, et al. Experiential learning theory: previous research and new directions. In *Perspectives on Cognitive, Learning, and Thinking Styles*; Sternberg, R. J., Zhang, L. F. E., Eds.; Routledge, 2001.
- [47] G. Greeno, Y. Engeström. Learning in activity. In *The Cambridge Handbook of the Learning Sciences*; Sawyer, K., Ed.; Cambridge University Press, 2014.
- [48] B. J. Reiser, I. Tabak. Scaffolding. In *The Cambridge Handbook of the Learning Sciences, Second Edition*; Sawyer, R., Ed.; Cambridge University Press, 2014.
- [49] D. Abrahamson, R. Lindgren. Embodiment and Embodied Design. In *The Cambridge Handbook of the Learning Sciences*; Sawyer, K., Ed.; Cambridge University Press, 2014.

- [50] R. Schank. Goal-based scenarios. <http://cogprints.org/624/1/V11ANSEK.html>.
- [51] R. Schank. *Teaching Minds: How Cognitive Science Can Save Our Schools*; Teachers College Press, 2011.
- [52] Committee for the Workshops on Computational Thinking. *Report of a workshop on the scope and nature of computational thinking*; National Academies Press: Washington, D.C., 2010.
- [53] J. Schell. *The Art of Game Design: A Book of Lenses*; CRC Press, 2008.
- [54] J. Lu, et al. Problem-based learning. In *The Cambridge Handbook of the Learning Sciences*; Sawyer, K., Ed.; Cambridge University Press, 2014.
- [55] R. M. Ryan, E. L. Deci. Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist* 2000, 55 (1).
- [56] R. M. Ryan, et al. The motivational pull of video games: a self-determination theory approach. *Motivation and Emotion* 2006, 30 (4), 344-360.
- [57] A. K. Przybylski, et al. The ideal self at play: the appeal of video games that let you be all you can be. *Psychological Science* 2012, 23 (1), 69-76.
- [58] J. P. Gee. *What Video Games Have to Teach Us about Learning and Literacy*; Palgrave: New York, 2003.
- [59] M. Resnick, et al. Scratch: programming for all. *Communications of the ACM* 2009, 52 (11), 60-67.
- [60] W. Booth. *Mixed-methods study of the impact of a computational thinking course on student attitudes about technology and computation*; Ph. D. dissertation; Baylor University, 2013.

- [61] Google for Education. Blockly. <https://developers.google.com/blockly/>.
- [62] Code.org. Hour of Code. <https://hourofcode.com/>.
- [63] Microsoft Research FUSE Labs. Kodu Game Lab. <https://www.kodugamelab.com/>.
- [64] J. H. Maloney, et al. Programming by choice: urban youth learning programming with Scratch. *ACM SIGCSE Bulletin* 2008, 40 (1), 367-371.
- [65] MIT Media Lab. Scratch Statistics. <https://scratch.mit.edu/statistics/>.
- [66] I. Harel, S. Papert. *Constructionism*; Ablex Publishing, 1991.
- [67] C. Scaffidi, C. Chambers. Skill progression demonstrated by users in the Scratch animation environment. *International journal of Human-Computer Interaction* 2012, 28 (6), 383-398.
- [68] D. S. Touretzky, et al. Teaching "lawfulness" with Kodu. *ACM SIGCSE*, 2016; pp 621-626.
- [69] S. Grover, S. Basu. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. *ACM SIGCSE*, 2017; pp 267-272.
- [70] O. Meerbaum-Salant, et al. Habits of programming in Scratch. *ACM ITiCSE*, 2011; pp 168-172.
- [71] P. A. Kirschner, et al. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist* 2006, 41 (2), 75-86.
- [72] K. J. Harms, et al. Distractors in parsons problems decrease learning efficiency for young novice programmers. *International Computing Education Research*, 2016; pp 241-250.

- [73] B. J. Ericson, et al. Solving parsons problems versus fixing and writing code. *Koli Calling Conference on Computing Education Research*, 2017; pp 20-29.
- [74] B. J. Ericson, et al. Evaluating the efficiency and effectiveness of adaptive parsons problems. *ACM ICER*, 2018; pp 60-68.
- [75] A. T. M. Bari, et al. EvoParsons: design, implementation and preliminary evaluation of evolutionary Parsons puzzle. *Genetic Programming and Evolvable Machines* 2019, 213-244.
- [76] B. J. Ericson, et al. Identifying design principles for CS teacher ebooks through design-based research. *ACM ICER*, 2016; pp 191-200.
- [77] A. Kumar. Epplets: a tool for solving parsons puzzles. *ACM SIGCSE*, 2018; pp 527-532.
- [78] A. N. Kumar. Helping students solve Parsons puzzles better. *ACM ITiCSE*, 2019; pp 65-70.
- [79] T. Wagner. *The Global Achievement Gap: Why Even Our Best Schools Don't Teach the New Survival Skills Our Children Need - and What We Can Do*; Basic Books, 2008.
- [80] S. Papert. *The Children's Machine: Rethinking School in the Age of the Computer*; Basic Books, 1993.
- [81] K. R. Sawyer. The New Science of Learning. In *The Cambridge Handbook of the Learning Sciences, Second Edition*; Sawyer, K. R., Ed.; Cambridge University Press, 2014.
- [82] D. Garcia, et al. The beauty and joy of computing. *ACM Inroads* 2016, 6 (4), 71-79.
- [83] G. Halfacree. *The official BBC Micro:bit user guide*; John Wiley & Sons, 2017.

- [84] E. J. Slattery, et al. Teachers' experiences of using Minecraft Education in primary school: an Irish perspective. *Irish Educational Studies* 2023, 1-20.
- [85] L. A. Gouws, et al. Computational thinking in educational activities: an evaluation of the educational game light-bot. *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, 2013.
- [86] P. Sengupta, et al. Integrating computational thinking with k-12 science education using agent-based computation: a theoretical framework. *Educational Information Technology* 2013, 18 (2), 351-380.
- [87] C. Crawford. *The Art of Computer Game Design*; Amazon Digital Services, 1984.
- [88] S. Hambrusch, et al. A multidisciplinary approach towards computational thinking for science majors. *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, 2009; pp 183-187.
- [89] M. Prensky. Digital natives, digital immigrants part 2: do they really think differently? *On the Horizon* 2001, 9 (6), 2-6.
- [90] S. De Freitas. *Learning in immersive worlds*; Joint Information Systems Committee: London, 2006.
- [91] M. Bober. *Games-Based Experiences for Learning*.; Futurelab: Bristol, 2010.
- [92] M. Ulicsak, M. Wright. *Games in education: serious games*; Futurelab: Bristol, 2010.
- [93] M. Ulicsak, B. Williamson. *Computer games and learning: a handbook*; Futurelab: London, 2011.
- [94] D. Clark, et al. *Digital games for learning: a systematic review and meta-analysis (executive summary)*; SRI International: Menlo Park, 2013.

- [95] J. M. Randel, et al. The effectiveness of games for educational purposes: a review of recent research. *Simulation and Gaming* 1992, 23, 261-276.
- [96] J. J. Vogel, et al. Computer gaming and interactive simulations for learning: a meta-analysis. *Journal of Educational Computing Research* 2006, 34 (3), 229-243.
- [97] K. A. Kapp. *The Gamification of Learning and Instruction: Game-Based Methods and Strategies for Training and Education*; Pfeiffer, 2012.
- [98] R. T. Hays. *The effectiveness of instructional games: A literature review and discussion (Technical Report 2005-004)*; Naval Air Warfare Center: Orlando, FL, 2005.
- [99] F. Ke. A qualitative meta-analysis of computer games as learning tools. *Handbook of research on effective electronic gaming in education* 2009, 1, 1-32.
- [100] K. Becker. Distinctions between games and learning: a review of current literature on games in education. In *Gaming and Cognition: Theories and Practice from the Learning Sciences*; Van Eck, R., Ed.; IGI Global, 2010.
- [101] T. Sitzmann. A meta-analytic examination of the instructional effectiveness of computer-based simulation games. *Personnel Psychology* 2011, 64 (2), 489-528.
- [102] R. Van Eck. Forward. In *Gaming and Cognition: Theories and Practice from the Learning Sciences*; Van Eck, R., Ed.; IGI Global, 2010.
- [103] R. J. Baker. Gaming the system: a retrospective look. *Philippine Computing Journal* 2011, 6 (2), 9-13.
- [104] R. E. Mayer. Cognitive foundations of game-based learning. In *Handbook of Gamed-Based Learning*; Plass, J. L., et al., Eds.; MIT Press, 2020; pp 83-110.

- [105] M. Resnick. Computer as paintbrush: technology, play, and the creative society. In *Play= Learning: How Play Motivates and Enhances Children's Cognitive and Social-Emotional Growth*; Singer, D. G., et al., Eds.; Oxford University Press, 2006.
- [106] SDT. Self-determination Theory. <https://selfdeterminationtheory.org/intrinsic-motivation-inventory/>.
- [107] W.-H. Wu, et al. Investigating the learning-theory foundations of game-based learning: a meta-analysis. *Journal of Computer Assisted Learning* 2012, 28, 265-279.
- [108] K. A. Wilson, et al. Relationships between game attributes and learning outcomes review and research proposals. *Simulation & Gaming* 2009, 40 (2), 217-266.
- [109] J. Asbell-Clarke, et al. The development of students' computational thinking practices in elementary-and middle-school classes using the learning game, Zoombinis. *Computers in Human Behavior* 2021, 115.
- [110] Y. Kafai, et al. From theory bias to theory dialogue: embracing cognitive, situated, and critical framings of computational thinking in K-12 CS education. *ACM Inroads* 2020, 44-53.
- [111] J. Sulaiman, et al. SAGE-RA: a reference architecture to advance the teaching and learning of computational thinking. *Embedding AI in Education Policy and Practice for Southeast Asia*, 2019; pp 421-431.
- [112] A. E. Hassan, R. C. Holt. A reference architecture for web servers. *Working Conference on Reverse Engineering*, 2000; pp 150-159.

- [113] S. Angelov, et al. A classification of software reference architectures: analyzing their success and effectiveness. *Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, 2009; pp 141-150.
- [114] Code.org. Code.org. <https://github.com/code-dot-org/code-dot-org> (accessed 2022).
- [115] C. Piech, et al. Autonomously generating hints by inferring problem solving policies. *ACM Learning @ Scale*, 2015; pp 195-204.
- [116] V. E. Owen, et al. Gameplay as assessment: analyzing event-stream player data and learning using GBA (a game-based assessment model). *Computer Supported Collaborative Learning*, 2013.
- [117] M. J. Lee, et al. In-game assessments increase novice programmers' engagement and level completion speed. *Proceedings of the ninth annual international ACM conference on International computing education research*, 2013; pp 153-160.
- [118] J. Helminen, et al. How do students solve parsons programming problems?: an analysis of interaction traces. *International Computing Education Research*, 2012.
- [119] D. Weintrop, U. Wilensky. Between a block and a typeface: designing and evaluating hybrid programming environments. *Interaction Design and Childern*, 2017; pp 183-192.
- [120] A. Shabo. Integrating constructionism and instructionism in educational hypermedia programs. *Journal of Educational Computing Education Research* 1997, 17 (3), 231-247.
- [121] S. Grover, R. Pea. Computational thinking in K–12: a review of the state of the field. *Educational Researcher* 2013, 42 (1), 38-43.
- [122] L. C. Kaczmarczyk, et al. Identifying student misconceptions of programming. *SIGCSE*, 2010; pp 107-111.

- [123] A. Emerson, et al. Cluster-based analysis of novice coding misconceptions in block-based programming. *ACM SIGCSE*, 2020; pp 825-832.
- [124] V. Karavirta, et al. A mobile learning application for parsons problems with automatic feedback. *Koli Calling International Conference on Computing Education Research*, 2012; pp 11-18.
- [125] K. V. . G. D. A. . Z. S. . J. M. A. King. *The distractor rationale taxonomy: enhancing multiple-choice items in reading and mathematics*; Pearson, 2004.
- [126] K. J. Harms, et al. Enabling independent learning of programming concepts through programming completion puzzles. *IEEE Visual Languages and Human-Centric Computing*, 2015; pp 271-279.
- [127] R. Zhi, et al. Evaluating the effectiveness of parsons problems for block-based programming. *ACM ICER*, 2019; pp 51-59.
- [128] J. Sweller. *Cognitive Load Theory: Recent Theoretical Advances*; Cambridge University Press, 2010.
- [129] K. A. Brennan. *Best of both worlds: issues of structure and agency in computational creation, in and out of school*; Doctoral Dissertation; MIT, 2013.
- [130] P. J. Rich, et al. Coding in K-8: international trends in teaching elementary/primary computing. *AECT Tech Trends* 2019, 63 (3), 311-329.
- [131] M. M. McGill, A. Decker. Tools, languages, and environments used in primary and secondary computing education. *ACM ITiCSE*, 2020; pp 103-109.
- [132] C. Frädriich, et al. Common bugs in Scratch programs. *ACM ITiCSE*, 2020; pp 89-95.

- [133] Y. Dong, et al. Defining tinkering behavior in open-ended block-based programming assignments. *ACM SIGCSE*, 2019; pp 1204-1210.
- [134] K. Brennan, et al. *Creative computing*; Harvard Graduate School of Education, 2018.
- [135] D. Franklin, et al. Scratch Encore: the design and pilot of a culturally-relevant intermediate Scratch curriculum. *ACM SIGCSE*, 2020; pp 794-800.
- [136] M. Tsur, N. Rusk. Scratch microworlds: designing project-based introductions to coding. *ACM SIGCSE*, 2018; pp 894-899.
- [137] J. Salac, et al. TIPP&SEE: a learning strategy to guide students through use-modify Scratch activities. *ACM SIGCSE*, 2020; pp 79-85.
- [138] S. Kong, et al. Development of computational thinking concepts in Scratch programming. *International Conference on Computers in Education*, 2020; pp 652-657.
- [139] B. B. Morrison, et al. Subgoals help students solve Parsons problems. *ACM SIGCSE*, 2016; pp 42-47.
- [140] P. Denny, et al. Evaluating a new exam question: parsons problems. *International Computing Education Research*, 2008; pp 113-124.
- [141] K. J. Harms, et al. Learning programming from tutorials and code puzzles: children's perceptions of value. *IEEE Visual Languages and Human-Centric Computing*, 2016; pp 59-67.
- [142] J. Helminen, et al. How do students solve parsons programming problems?--execution-based vs. line-based feedback. *Learning and Teaching in Computing and Engineering*, 2013; pp 55-61.

- [143] P. Ihanola, V. Karavirta. Two-dimensional parson's puzzles: The concept, tools, and first observations. *Journal of Information Technology Education* 2011, *10*, 119-132.
- [144] S. Garner. An exploration of how a technology-facilitated part-complete solution method supports the learning of computer programming. *Informing Science & Information Technology*, 2007.
- [145] T. van Gog, F. Paas. Instructional efficiency: revisiting the original construct in educational research. 2008, *43* (1), 16-26.
- [146] F. Tahir, et al. Investigating the effects of gamifying SQL-Tutor. *ICCE*, 2020; pp 416-425.
- [147] S. Marwan, et al. Adaptive immediate feedback can improve novice programming engagement and intention to persist in computer science. *ACM ICER*, 2020; pp 194-203.
- [148] G. Raubenheimer, et al. Toward empirical analysis of pedagogical feedback in computer programming learning environments. *ACE*, 2021; pp 189-195.
- [149] K. Bovermann, T. J. Bastiaens. Towards a motivational design? Connecting gamification user types and online learning activities. *Research and Practice in Technology Enhanced Learning* 2020, *15* (1), 1-18.
- [150] P. Brusilovsky, et al. An integrated practice system for learning programming in Python: design and evaluation. *Research and Practice in Technology Enhanced Learning* 2018, *13* (1), 1-40.
- [151] E. Aivaloglou, F. Hermans. Early programming education and career orientation: the effects of gender, self-efficacy, motivation and stereotype. *ACM SIGCSE*, 2019; pp 679-685.

- [152] J. B. Bush, et al. Drag and drop programming experiences and equity: analysis of a large scale middle school student motivation survey. *ACM SIGCSE*, 2020; pp 664-670.
- [153] C. Schulte, M. Knobelsdorf. Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. *ACM ICER*, 2007; pp 27-38.
- [154] L. Cao, et al. Work in progress report: a STEM ecosystem approach to CS/CT. *ACM SIGCSE*, 2020; pp 999-1004.
- [155] L. A. DeLyser. A community model of CSforALL: analysis of community commitments for cs education. *ACM ITiCSE*, 2018; pp 99-104.
- [156] C. Lewis, et al. Alignment of goals and perceptions of computing predicts students' sense of belonging in computing. *ACM ITiCSE*, 2019; pp 11-19.
- [157] I. Karvelas, et al. The effects of compilation mechanisms and error message presentation on novice programmer behavior. *ACM SIGCSE*, 2020; pp 759-765.
- [158] A. Kumar. The effect of providing motivational support in parsons puzzle tutors. *Artificial Intelligence in Education*, 2017; pp 528-531.
- [159] A. N. Kumar. Mnemonic variable names in Parsons puzzles. *ACM CompEd*, 2019; pp 120-126.
- [160] T. Sirkia. Combining parson's problems with program visualization in CS1 context. *Koli Calling on Computing Education Research*, 2016; pp 155-159.
- [161] D. Kashmira, J. Mason. Empowering learning designers through design thinking. *International Conference on Computers in Education*, 2020; pp 497-502.
- [162] CodeHER. codeHER. <https://www.codehergirls.org/>.

- [163] P. Ihanola, et al. Educational data mining and learning analytics in programming: literature review and case studies. *ITiCSE Working Group Reports*, 2016; pp 41-63.
- [164] L. Mannila, et al. Computational thinking in K-9 education. *Innovation & Technology in Computer Science Education*, 2014; pp 1-29.
- [165] E. Barendsen, et al. Concepts in K-9 computer science education. *ACM ITiCSE-WGR*, 2015; pp 85-116.
- [166] N. Grgurina, et al. Computational thinking skills in dutch secondary education: exploring pedagogical content knowledge. *ACM Koli Calling*, 2014; pp 173-174.
- [167] Amazon. Amazon Mechanical Turk. <https://www.mturk.com/>.
- [168] Panopto. Panopto. <https://www.panopto.com/>.
- [169] B. B. Morrison, et al. Measuring cognitive load in introductory CS: adaptation of an instrument. *International Computing Education Research*, 2014; pp 131-138.
- [170] P. Charters, et al. Challenging stereotypes and changing attitudes: the effect of a brief programming encounter on adults' attitudes toward programming. *ACM SIGCSE*, 2014; pp 653-658.
- [171] S. Barab. Design-based research: a methodological toolkit for engineering change. In *The Cambridge Handbook of the Learning Sciences*; Sawyer, K., Ed.; Cambridge University Press, 2014.
- [172] D. Difallaha, et al. Demographics and dynamics of mechanical Turk workers. *ACM Web Search and Data Mining*, 2018; pp 135-143.
- [173] Qualtrics. Qualtrics. <https://qualtrics.com/>.

- [174] C. O. Fritz, et al. Effect size estimates: current use, calculations, and interpretation. *Journal of Experimental Psychology: General* 2012, 141 (1), 2-18.
- [175] L. M. Rea, R. A. Parker. *Designing and Conducting Survey Research: A Comprehensive Guide*; John Wiley & Sons, 2014.
- [176] F. Paas, J. Merrienboer. The efficiency of instructional conditions: an approach to combine mental effort and performance measures. *Human Factors* 1993, 35 (4), 737-743.
- [177] J. Bender, et al. Integrating Parsons Puzzles with Scratch. *ICCE*, 2021.
- [178] J. Bender, et al. Integrating Parsons puzzles within Scratch enables efficient computational thinking learning. *Research and Practice in Technology Enhanced Learning* 2023, 18.
- [179] D. Ravitch. *Reign of error: the hoax of the privatization movement and the danger to America's public schools*; Vintage, 2013.
- [180] N. Selwyn, et al. High-tech, hard work: an investigation of teachers' work in. *Learning, Media and Technology* 2017, 390-405.
- [181] J. Wang, et al. Gender differences in factors influencing pursuit of computer science and related fields. *ACM ITiCSE*, 2015; pp 117-122.
- [182] J. McBroom, et al. Understanding gender differences to improve equity in computer programming education. *ACE*, 2020; pp 185-194.
- [183] OECD. Enrolment by field. <https://stats.oecd.org/>.
- [184] D. J. Malan, H. H. Leitner. Scratch for budding computer scientists. *ACM SIGCSE Bulletin* 2007, 39 (1).
- [185] Y. Du, et al. A review of research on parsons problems. *ACE*, 2020; pp 195-202.

- [186] E. Bubacher, et al. Where the rubber meets the road: the impact of the interface design on model exploration in science inquiry. *Transforming Learning, Empowering Learners*, 2016.
- [187] D. Weintrop, et al. Block-based comprehension: exploring and explaining student outcomes from a read-only block-based exam. *ACM ITiCSE*, 2019; pp 1218-1224.
- [188] R. K. Atkinson, et al. Transitioning from studying examples to solving problems: effects of self-explanation prompts and fading worked-out steps. *Educational Psychology* 2003, 95 (4).
- [189] R. Moreno, et al. Optimizing worked-example instruction in electrical engineering: the role of fading and feedback during problem-solving practice. *Engineering Education* 2009, 98 (1), 83-92.
- [190] R. C. Holt, et al. SP/k: a system for teaching computer programming. *Communications of the ACM* 1977, 20 (5), 301-309.
- [191] C. Sepúlveda-Díaz, et al. Lessons learned from introducing preteens in parent-led homeschooling to computational thinking. *ACM SIGCSE*, 2020; pp 65-71.
- [192] L. Zhang, et al. Progression of computational thinking skills in Swedish compulsory schools with block-based programming. *ACE*, 2020; pp 66-75.
- [193] B. Munasinghe, et al. Teachers' understanding of technical terms in a computational thinking curriculum. *ACE*, 2021; pp 106-114.
- [194] A. Lamprou, A. Repenning. Teaching how to teach computational thinking. *ACM ITiCSE*, 2018; pp 69-74.

- [195] P. Haduong, K. Brennan. Helping K–12 teachers get unstuck with scratch: the design of an online professional learning experience. *ACM SIGCSE*, 2019; pp 1095-1101.
- [196] R. Zhi, et al. Exploring instructional support design in an educational game for K-12 computing education. *ACM ITiCSE*, 2018; pp 747-752.
- [197] J. W. Coffey. A study of the use of a reflective activity to improve students' software design capabilities. *ACM SIGCSE*, 2017; pp 129-133.
- [198] J. Whalley, H. Ogier. Learning journals in creative programming assessments: exposing bugs, issues, and misconceptions. *ACE*, 2020; pp 39-47.
- [199] P. S. Buffum, et al. A practical guide to developing and validating computer science knowledge assessments with application to middle school. *ACM SIGCSE*, 2015; pp 622-627.
- [200] Prolific. Prolific | Online participant recruitment for surveys and market research.
<https://www.prolific.co/>.
- [201] J. Bender, et al. Learning computational thinking efficiently: how parsons programming puzzles within Scratch might help. *Australasian Computing Education Conference*, 2022; pp 66-75.
- [202] P. Blikstein. Pre-college computer science education: a survey of the field, 2018.
<https://goo.gl/gmS1Vm>.
- [203] C. Proctor, P. Blikstein. How broad is computational thinking? a longitudinal study of practices shaping learning in computer science. *International Society of the Learning Sciences*, 2018.

- [204] L. Hamilton-smith. Learning Curve: Coding classes to become mandatory in Queensland schools. <https://www.abc.net.au/news/2016-11-17/coding-classes-in-queensland-schools-mandatory-from-2017/8018178>.
- [205] R. S. Baker, et al. Learning engineering: a view on where the field is at, where it's going, and the research needed. *Technology, Mind, and Behavior* 2022.
- [206] D. Schaffhauser. CS education week kicks off, but some teachers don't feel ready. <https://thejournal.com/articles/2018/12/03/cs-education-week-kicks-off-but-some-teachers-dont-feel-ready.aspx>.
- [207] M. Celepkolu, et al. Upper elementary middle grade teachers' perceptions, concerns, and goals for integrating CS into classrooms. *SIGCSE*, 2020; pp 965-970.
- [208] A. Campos, et al. piBook: introducing computational thinking to diversified audiences. *Computer Supported Education*, 2017; pp 179-195.
- [209] L. Pollock, et al. Infusing computational thinking across disciplines: reflections & lessons learned. *SIGCSE*, 2019; pp 4435-4441.
- [210] J. R. Anderson. Acquisition of cognitive skill. *Psychological Review* 1982.
- [211] R. D. Pea. Language independent conceptual "bugs" in novice programming. *Educational Computing Research* 1986, 25-36.
- [212] L. E. Winslow. Programming pedagogy—a psychological overview. *SIGCSE Bulletin* 1996, 17-22.
- [213] T. Kim, S. Axelrod. Direct instruction: an educators' guide and a plea for action. *The Behavior Analyst Today* 2005, 6 (2), 111-120.
- [214] J. Garner, et al. Mastery learning in computer science education. *ACE*, 2019; pp 37-46.

- [215] A. J. Kim. *Game thinking: innovate smarter & drive deep engagement with design techniques from hit games.*; gamethinking.io, 2018.
- [216] H. Plattner. *An introduction to design thinking: process guide*; Institute of Design at Stanford, 2013.
- [217] M. D. Dickey. “Ninja looting” for instructional design: the Design challenges of creating a gamed-based learnign environment. *SIGGRAPH*, 2006.
- [218] V. Shute, et al. Maximizing learning without sacrificing the fun: Stealth assessment, adaptivity and learning supports in educational games. *Computer Assisted Learning* 2021, 127-141.
- [219] G. Haldeman, et al. Providing meaningful feedback for autograding of programming assignments. *SIGCSE*, 2018; pp 278-283.
- [220] W. Cazzola, D. M. Olivares. Gradually learning programming supported by a growable programming language. *IEEE Transactions on Emerging Topics in Computing* 2015, 404-415.
- [221] S. Rose, et al. *Pirate plunder: game-based computational thinking using scratch blocks*; Academic Conferences and Publishing International Limited, 2018.
- [222] D. Dicheva, A. Hodge. *Active learning through game play in a data structures course*; SIGCSE, 2018.
- [223] S. Turkle, S. Papert. Epistemological pluralism: styles and voices within the computer culture. *Women in Culture and Society* 1990, 128-157.

- [224] D. Ball, D. Cohen. Reform by the book: what is — or might be — the role of curriculum materials in teacher learning and instructional reform? *Educational Researcher* 1996, 25 (9), 6-14.
- [225] C. Schulte, J. Magenheimer. Novices' expectations and prior knowledge of software development: results of a study with high school students. *Computing Education Research*, 2005; pp 143-153.
- [226] D. Morrison, J. McCutcheon. Empowering older adults' informal, self-directed learning: harnessing the potential of online personal learning networks. *Research and Practice in Technology Enhanced Learning* 2019, 1-16.
- [227] N. Bresnihan, et al. Increasing parental involvement in computer science education through the design and development of family creative computing workshops. *Computer Supported Education*, 2019; pp 479-502.
- [228] D. B. Chin, et al. Preparing students for future learning with teachable agents. *Educational Technology Research and Development* 2010, 649-669.
- [229] M. Denden, et al. Does personality affect students' perceived preferences for game elements in gamified learning environments? *Advanced Learning Technologies*, 2018; pp 111-115.
- [230] A. L. D. Santos, et al. A systematic mapping study on game elements and serious games for learning programming. *Computer Supported Education*, 2018; pp 328-356.
- [231] R. Bockmon, et al. Validating a CS attitudes instrument. *SIGCSE*, 2020; pp 899-904.
- [232] J. Bender, et al. Learning computational thinking efficiently with block-based Parsons puzzles. *ICCE*, 2022.

- [233] J. P. Gee, E. Hayes. Nurturing affinity spaces and game-based learning. In *Games, Learning, and Society: Learning and Meaning in the Digital Age*; Cambridge University Press, 2012.
- [234] D. N. Perkins, et al. Conditions of learning in novice programmers. *Educational Computing Research* 1986, 2 (1), 37-55.
- [235] C. Piech, et al. Modeling how students learn to program. *ACM SIGCSE*, 2012.
- [236] T. W. Price, et al. iSnap: towards intelligent tutoring in novice programming environments. *ACM SIGCSE*, 2017; pp 483-488.
- [237] SonarQube. Code Quality and Code Security | SonarQube. <https://www.sonarqube.org/> (accessed 2022).
- [238] Checkmarx. Application Security Testing | Checkmarx. <https://checkmarx.com/> (accessed 2022).
- [239] Veracode. Confidently secure apps you build and manage with Veracode. <https://www.veracode.com/> (accessed 2022).
- [240] L. Seiter, B. Foreman. Modeling the learning progressions of computational thinking of primary grade students. *Proceedings of International Computing Education Research*, 2013.
- [241] B. Boe, et al. Hairball: Lint-inspired static analysis of scratch projects. *Proceeding of the 44th ACM technical symposium on Computer science education*, 2013.
- [242] J. M. Leon. hairball/mastery.py. <https://github.com/jemole/hairball/blob/master/hairball/plugins/mastery.py>.
- [243] SBECACS. South Bronx Early College Academy. <https://sbecacs.org/>.

- [244] H. Bort, D. Brylow. CS4Impact: Measuring Computational Thinking Concepts. *Proceeding of the 44th ACM technical symposium on Computer science education*, 2013; pp 427-432.
- [245] Google., 2015. Exploring Computational Thinking. <https://www.google.com/edu/resources/programs/exploring-computational-thinking/>.
- [246] M. Prensky. Digital natives, digital immigrants part 1. *On the Horizon* 2001, 9 (5), 1-6.
- [247] M. Prensky. *Don't Bother Me, Mom, I'm Learning!: How Computer and Video Games are Preparing Your Kids for 21st Century Success and How You Can Help!*; Paragon House, 2006.
- [248] A. A. DiSessa. *Changing minds: Computers, Learning, and Literacy*; MIT Press, 2000.
- [249] M. Resnick, B. Silverman. Some reflections on designing construction kits for kids. *Proceedings of International Conference for Interaction Design and Children*, 2005.
- [250] U. Wilensky, M. J. Jacobson. Complex systems and the learning sciences. In *The Cambridge Handbook of the Learning Sciences*; Sawyer, K., Ed.; Cambridge University Press, 2014.
- [251] K. E. DiCerbo, J. T. Behrens. Implications of the digital ocean on current and future assessment. In *Computers and their impact on state assessment: Recent history and predictions for the future*; , 2012; pp 273-306.
- [252] J. Hollingsworth. Automatic graders for programming classes. *Communications of the ACM* 1960, 3 (10), 528-529.
- [253] P. C. Isaacson, T. A. Scott. Automating the execution of student programs. *ACM SIGSCE Bulletin* 1989, 21 (2), 15-22.

- [254] D. Jackson, M. Usher. Grading student programs using ASSYST. *ACM SIGCSE Bulletin* 1997, 29 (1), 335-339.
- [255] T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering* 1976, 4, 308-320.
- [256] S. H. Edwards, M. A. Perez-Quinones. Web-CAT: automatically grading programming assignments. *ACM SIGSCE Bulletin* 2008, 40 (3), 328.
- [257] J. Spacco, et al. Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. *ACM SIGCSE Bulletin* 2006, 38, 13-17.
- [258] C. Wilcox. Testing strategies for the automated grading of student programs. *ACM SIGCSE*, 2016; pp 437-442.

Appendix A: Feasibility Study Materials

Teacher survey, semi-structured interview, and design thinking workshop materials comprising the feasibility study, conducted at the South Bronx Early College Academy [243], and Girls Who Code [19] and codeHER [162] events, follow.

A.1 Teaching and Learning with Scratch: Survey

This 5-10 minute survey aims to surface opportunities for improving teaching and learning with Scratch, with a focus on grade 6-8 students. The responses will be reported in an aggregated and anonymized manner regardless of whether identifying information is provided. If you would like to receive a copy of the report when it is published, please provide your email address. Thank you for your assistance with this study. More information on Scratch is available here: <https://scratch.mit.edu/>.

* Required

1. Email address *
2. Name (optional)
3. Email (optional)
4. How long have you been teaching with Scratch? * (Mark only one)
 - Less than 6 months
 - 6-12 months
 - 1-2 years
 - 2-4 years
 - More than 4 years
5. Which subjects do you teach with Scratch? * (Mark all that apply)

- Science
- Math
- Language Arts
- Computer Science
- Social Studies
- Applied Arts
- Other:

6. What percentage of the curriculum you teach involves Scratch? * (Mark only one)

- 0%
- 1-5%
- 6-10%
- 11-15%
- 16-25%
- 26-50%
- 51-100%

7. Which grades have you most frequently taught? * (Mark only one)

- K-2
- 3-5
- 6-8
- 9-12
- Other:

Computational Thinking

Computational thinking is the thought process involved in formulating a problem and expressing its solution in a way a computer can carry it out. More information on computational thinking is available here: https://en.wikipedia.org/wiki/Computational_thinking.

To what extent do your students engage in the following activities when you teach with Scratch?

8. Gathering appropriate information and selecting relevant information (data collection) * (Mark only one)

1 2 3 4 5

Never Always

9. Making sense of data, finding patterns, drawing conclusions (data analysis) * (Mark only one)

1 2 3 4 5

Never Always

10. Organizing and depicting data in appropriate graphs, charts, words, images, tables, etc. (data representation) * (Mark only one)

1 2 3 4 5

Never Always

11. Breaking down tasks into smaller manageable parts and merging subtasks (problem decomposition) * (Mark only one)

1 2 3 4 5

Never Always

12. Planning and organizing sequences of steps taken to solve a problem (algorithms) * (Mark only one)

1 2 3 4 5

Never Always

13. Reducing complexity to define main idea, finding characteristics and creating models

(abstraction) * (Mark only one)

1 2 3 4 5

Never Always

14. Using or creating simulations, for instance, for running experiments (simulation) * (Mark only one)

1 2 3 4 5

Never Always

15. Recognizing how technology can help us accomplish new tasks that would otherwise be too repetitive, infeasible, or difficult (automation) * (Mark only one)

1 2 3 4 5

Never Always

16. Organizing resources to simultaneously and cooperatively carry out tasks to reach a goal (parallelization) * (Mark only one)

1 2 3 4 5

Never Always

17. Please briefly describe one instance when you felt successful in including some of the above activities in your teaching practice.

Direct Instruction

Direct instruction involves the explicit teaching of a skill-set through lecture, presentation, and

demonstration. More information on direct instruction is available here:

https://en.wikipedia.org/wiki/Direct_instruction.

18. How frequently do your students ask for help when getting started with Scratch? *

(Mark only one)

1 2 3 4 5

Never Always

19. How frequently do you supplement Scratch with direct instruction? * (Mark only one)

1 2 3 4 5

Never Always

20. How do you deliver direct instruction when teaching with Scratch? * (Mark all that apply)

- I don't deliver direct instruction when teaching with Scratch
- Concept explanations
- Demonstrations
- Written tutorials
- One-on-one tutoring
- Other:

21. If Scratch included tools that facilitated introducing blocks throughout a learning progression, how willing would you be to use that functionality to support learning? * (Mark only one)

1 2 3 4 5

Unwilling Willing

Parsons Programming Puzzles

Parsons Programming Puzzles (PPP) provide opportunities for practice with basic programming principles in an entertaining format. They are a family of code construction assignments in which lines of code are given, and the task is to form the solution by sorting and selecting the correct code. Research indicates they might be a more efficient learning approach than fixing code with errors or writing equivalent code. More information on PPP is available here:

<http://crpit.com/confpapers/CRPITV52Parsons.pdf>.

22. How frequently do you assign PPP to students? * (Mark only one)

1 2 3 4 5

Never

Always

23. If you assign PPP, how do you deliver them? * (Mark all that apply)

- I don't assign PPP
- Handcrafted paper assignments
- Hot Potatoes
- ViLLe
- CORT
- Other:

24. If Scratch included effective PPP functionality, how willing would you be to assign these puzzles to students? * (Mark only one)

1 2 3 4 5

Unwilling

Willing

25. If Scratch included effective PPP authoring functionality, how willing would you be to create

these puzzles for your students? * (Mark only one)

1 2 3 4 5

Unwilling Willing

Constructionist Video Games

Constructionist Video Games (CVG) are designed environments in which players construct personally meaningful artifacts in order to overcome conflicts or obstacles resulting in quantifiable outcomes. More information on CVG is available here:

<http://ccl.northwestern.edu/2016/WeintropHolbertHornWilensky2016.pdf>.

26. How frequently do your students engage in discovery-based learning in Scratch by creating their own projects? * (Mark only one)

1 2 3 4 5

Never Always

27. How frequently do students ask for project ideas? * (Mark only one)

1 2 3 4 5

Never Always

28. How frequently do you guide student discovery in Scratch with structured assignments? * (Mark only one)

1 2 3 4 5

Never Always

29. If Scratch included Constructionist Video Games intended to guide your students as they learn computational thinking, how willing would you be to assign those games to students? *

(Mark only one)

1 2 3 4 5

Unwilling Willing

30. If Scratch included Constructionist Video Games creation, how willing would you be to design games for your students? * (Mark only one oval)

1 2 3 4 5

Unwilling Willing

Intelligent Tutoring Systems

An intelligent tutoring system (ITS) provides immediate and customized instruction and feedback to students by a variety of delivery mechanisms such as just-in-time hints, on-demand information, and next activity selection. More information on ITS is available here:

https://en.wikipedia.org/wiki/Intelligent_tutoring_system.

31. When your students are learning in Scratch, how frequently do they seek instructor guidance?

* (Mark only one)

1 2 3 4 5

Never Always

32. How frequently does the rate of student questions exceed your capacity to provide guidance promptly to individuals? * (Mark only one)

1 2 3 4 5

Never Always

33. If Scratch included ITS functionality, how willing would you be to enable that functionality to complement your teaching? * (Mark only one)

1 2 3 4 5

Unwilling Willing

34. If you had the opportunity to request one new feature intended to facilitate teaching and learning with Scratch, what would it be?

35. What other software do you use that is similar to Scratch?

36. Are there any questions and/or terms in this survey that are difficult to understand? Your candid feedback will help inform the interpretation of the results, and help guide the design of future studies.

A.2 Teaching and Learning with Scratch: Semi-Structured Interview Protocol

All 1:1 interviewers used the following outline when conducting semi-structured interviews with teachers. Conversations regularly diverged to follow-up on relevant anecdotes, then converged back toward this standard.

1. Background Info

- a. Which age groups do you predominantly work with?
- b. Do you work through a specific organization? Does that organization have standards through which you teach programming?
- c. How long have you been teaching computer science to children?
- d. How long have you been learning computer science yourself?
- e. Are there any features in any other interfaces you've taught with (Code.org, etc.). that you found especially helpful with teaching computational thinking?

- f. What are some methods you've used to teach computational thinking concepts that have worked best for students (e.g., analogies)?
2. Instructional Paradigm
- a. How do you introduce computational thinking concepts? Do you provide direction instruction?
 - b. What types of assignment materials and scaffolding do you provide students to supplement Scratch?
 - c. If Scratch offered the capability to embed those materials and scaffolding in the environment, would you use it?
2. Gameful Learning
- a. What types of games do your students most frequently create in Scratch
 - b. How are game construction assignments structured so that students learn computational thinking concepts?
 - c. If Scratch offered puzzles and games that students could play in order to learn computational thinking, would you assign those games?
 - d. Would you design them?
3. Formative Assessment
- a. What types of formative feedback do you provide students?
 - b. How frequently do you provide formative feedback to students?
 - c. If you could provide objectives for students to accomplish within Scratch, would you do so?
 - d. If Scratch could provide hints based on students' progress, how would you prefer those hints be delivered?

- e. Should the hints arrive just-in-time or on-demand?
4. Summative Assessment
- a. How do you assess students' Scratch projects? Do you use rubrics?
 - b. Have you considered using automated assessments to complement manual inspection?
 - c. How do you assess students' mastery of computational thinking?
 - d. How do you differentiate instruction for students at different levels of mastery?
5. Call to Action
- a. Which elements of Scratch make teaching computational thinking easier?
 - b. How could students learn computational thinking more effectively and efficiently from an environment like Scratch?
 - c. How could teaching with an environment like Scratch be simpler?

A.3 Teaching and Learning with Scratch: Design Thinking Workshop

Our third method for evaluating feasibility and identifying opportunities for advancing CT teaching and learning in grades 6-8 involved facilitating design thinking workshops that unfolded in iterative sequences of open-ended idea elicitation, followed by SAGE prototype demonstration and discussion. The intent of human-centered design is to include end-users not only in the documenting of the problem, but also in the production of the solution. We conducted three workshops in informal learning organizations, and one in a middle school. Small teams of researchers gathered with groups of teachers who participated in prototype demonstrations and engaged in interactive dialogue, loosely bounded by the following 45-90 minute design thinking outline.

1. Conversation about the typical delivery of CT across grades 6-8
2. Open discussion of pain points in teaching and learning CS in middle school
3. Learning progressions: palette and block restrictions
 - a. Specific computational thinking concepts and sequences
 - b. Instructors can check and uncheck palettes in design mode
 - c. Instructors can still use these
 - i. Instructors can put blocks in the game interface, but the students can't add any new ones.
 - ii. Students can't do anything to the blocks already there
 - d. Left click exclude/include is available to restrict individual blocks
 - e. Quest: several games, increasing amount of freedom allowed as you're learning concepts in a progression
 - i. Different difficulty levels for different games
4. Focusing on CT concepts via points
 - a. Emphasizing looping: the loop has a higher point value than doing it individually
 - i. Explain the point system, preset configurations, save/load configurations
 - b. Explain scoring system options for Parsons Puzzles and Constructionist Video Games
 - c. Parsons operates on a specific scoring rubric
 - i. Incorrect block placement decrements score
 - d. Constructionist: set your own goals for learning objectives, based on instructor preferences.

- e. Record the number of points that students get in their games, translate into leaderboards.
 - f. Teacher gets to weight various ways that students can be incentivized.
 - g. Several ways of assessing mastery: actual mastery plugin with the Hairball analysis, and PECT analysis.
 - i. Login as a student: spider graph is the mastery of results across concepts
 - ii. Automated assessment results from CT concepts
5. Gameful Direct Instruction: create a Parsons Puzzle
- a. Create sprites and put them in the stage
 - b. Game templates under construction
 - i. Might be a calculation, maze, etc.
 - c. Enter question, enter hint
 - i. Find a number: have you tried a loop? Canned vs. dynamic hints.
 - d. Saving/loading Parsons Puzzles in Design and Play modes
 - e. Load from a student view:
 - i. Automatic constriction of which palettes are available
 - ii. Experiment from a selective Parsons palette.
 - f. When you load as a student, the student doesn't see any scripts in the scripts pane.
The goal is to match what the teacher has created.
 - g. Discuss which CT concepts and other subjects (e.g. math) teachers might want to deliver via Parsons Puzzles
6. Gameful Constructionism: game-objective editor and feedback system
- a. Deep dive on principles of Constructionist Video Games

- b. Beginning quest: once upon a programmer
 - c. Capability to create game objectives using block-based software testing grammar:
teacher task
 - d. Expect to have games pre-built and have templates and games customizable with
content relevant for your school/environment.
 - e. Make copies of your own games. Borrow games and objectives from the affinity
space.
 - f. Objective blocks will have a drop own: don't have to know underlying names of
Scratch blocks
 - g. Explore options for student view of feedback in real-time
 - h. Explore teacher capability, time, and willingness to engage in this activity
7. Gameful Intelligent Tutoring: recommender and auto-hinting
- a. Outer loop next-task guidance via Mission/Quest/Game recommendations
 - b. Ordered vs. unordered access to games in the quest
 - c. Guidance toward additional practice games when automated evaluation of
mastery low, challenge games when mastery high
 - d. Describe inner loop and opportunities for problem-solving support through
individualized just-in-time and on-demand hinting
 - e. Explore teacher interest in throttling hinting at the class and individual levels
8. Infusing CT across the curriculum
- a. Demonstrate Mission/Quest/Game creation, and elicit requirements that would
enable teachers from CS and other subjects to fuse learning goals
 - b. Discuss game-based learning that facilitates game-based assessment

- c. Review CS/CT standards and means of incorporation, tracking, and reporting
- d. Identify teacher wish-list for Scratch and CT in middle school

Appendix B: Field Study Protocol Materials

A representative sample of materials external to SAGE from the three field studies conducted via Amazon Mechanical Turk and Prolific follow.

B.1 Example Study Guide

0. What is your Prolific ID?

Summary

We are conducting an academic research experiment about teaching and learning computational thinking (CT). We need you to complete a series of activities designed to help us study a variety of ways to learn CT efficiently. Please complete the activity in each of the 10 steps that follow in order. Due to the constraints of the experimental design, we cannot approve the work if items are completed out of order, if not all of the steps are completed, or if the incorrect Username is input into any form in the activities linked below. Note that some activities (4, 6, 8) are labeled: **Timed Activity**. Please attempt to complete these activities without interruption.

We track completion of each step independently, but request completion confirmation via a button for each activity here.

1. Activity 1 Follow the directions in the [Virtual Machine Setup Instructions](#). If the file download in instruction 2 is not complete by the time you have reached instruction 4, you can proceed here through study activities 2-5. However, the remainder of the Virtual Machine Setup Instructions must be followed before proceeding to study activity 6.
2. Activity 2 Visit <https://uat.cu-sage.org/#/reg> and create a unique new account. Per Prolific policy, do not enter any personally identifiable information, but make sure to record the Username entered because you will need to enter it throughout the experiment.

The Username is not required to be an email address.

Once you have created an account, please close the uat.cu-sage.org browser window.

You will need to remember your Username and password when using this site via the virtual machine in study activity 6.

Please enter your SAGE Username:

3. Activity 3 Complete this [background survey](#), and then review this [introduction](#) to the learning system and to the CT concept of Loops.
4. Activity 4 **Timed Activity** | Complete the [pretest](#).
5. Activity 5 Provide [feedback on the pretest](#).
6. Please confirm you have followed the [Virtual Machine Setup Instructions](#) from study activity 1 through completion before proceeding with study activity 6.

Activity 6 **Timed Activity**

Using the browser launched in the virtual machine, login into <https://uat.cu-sage.org/> using the Username created in study activity 1. Navigate to the assigned Quest, and attempt to complete seven puzzles in the learning system in order (Warm Up, then 1-6).

Please do not start the puzzles without first reviewing the instructions labeled **Start Here!** | **Quest Instructions for All Puzzles**, as those are designed to accompany the puzzles. When you complete a puzzle or the timer expires, navigate to the next puzzle. Confirm activity 6 completion after you have attempted to solve all seven puzzles in order. **Note: we must reject submissions unassociated with evidence of interaction with all seven puzzles.**

7. Activity 7 Provide [puzzle feedback](#).
8. Activity 8 **Timed Activity** | Complete the [posttest](#).

9. Activity 9 Provide [feedback on the posttest](#).

10. Activity 10 **Timed Activity**

Provide [feedback on the experiment](#) and mark activity 10 completed before advancing to the next page to conclude the study.

11. You have reached the conclusion of the study. Upon confirming your Prolific ID and SAGE Username, you will be redirected to Prolific. We are grateful for your time and participation.

Please enter your Prolific ID

B.2 Learning System and CT Concept Video Tutorials

1. Field study 1 on [sequences](#).
2. Field study 2 on [conditionals](#).
3. Field study 3 on [looping](#).

B.3 Demographics & Programming Behavior

1. What is your SAGE Username (used to login to <https://uat.cu-sage.org>)?
2. What is your current age?
 - a. 18-25
 - b. 25-30
 - c. 30-35
 - d. 40-45
 - e. 45-50
 - f. 50-55
 - g. 55-60
 - h. Above 60

3. To which gender identity do you most identify?
 - a. Female
 - b. Male
 - c. Non-binary
 - d. Prefer not to say
4. What is your education level (highest degree completed)?
 - a. 12th grade or less
 - b. Graduated high school or equivalent
 - c. Some college, no degree
 - d. Associate degree
 - e. Bachelor's degree
 - f. Graduate degree
5. In which country do you currently reside?
6. Have you ever written a computer program (can be in a block-based or text-based language)?
7. Please select the option that best describes your prior programming experience.
 - a. Never attempted to program before
 - b. Have tried programming activities, but have not taken a class
 - c. Currently taking a class on programming/computer science
 - d. Already completed one or more classes on programming/computer science
8. How would you describe your level of experience with computer applications like Scratch, Blockly, Alice, and Tynker?
 - a. Very unfamiliar

- b. Unfamiliar
 - c. Unsure
 - d. Familiar
 - e. Very Familiar
9. On a scale of 1 to 10, how do you estimate your programming experience?
10. How many programming languages do you feel you know at least at an intermediate level?
- a. None
 - b. 1-3
 - c. More than 3
11. For the following statements, please indicate how true each is for you. Programming is...
- Scale: Not at all true (1) | 2 | 3 | Somewhat true (4) | 5 | 6 | Very true (7)
- a. Too difficult to understand
 - b. Something I've wanted to learn
 - c. Boring
 - d. Something I did not know about
 - e. Fun
 - f. A foreign concept
 - g. Enjoyable
 - h. Important to know
 - i. Easy to start
 - j. An innate ability
 - k. Too time consuming

- l. Nerdy
 - m. Something that takes practice
12. Describe your attitude/view towards programming
13. Choose one of the following five choices that best expresses your feeling about the statement. If you have no strong opinion, choose “Neutral”.

Scale: Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

- a. After I study a topic in computational thinking and feel that I understand it, I have difficulty solving problems on the same topic.
- b. Errors generated by computers are random, and when they happen there’s not much I can do to understand why.
- c. If I want to apply a method used for solving one computational thinking problem to another problem, the problems must involve very similar situations.
- d. I can usually figure out a way to solve computational thinking problems.
- e. When I solve a computational thinking problem, I break it into smaller parts and solve them one at a time.
- f. I do not spend more than five minutes stuck on a computational thinking problem before giving up or seeking help from someone else.
- g. There are times I solve a computational thinking problem more than one way to help my understanding.
- h. I think about the computational thinking I experience in everyday life.
- i. Tools and techniques from computational thinking can be useful in the study of other disciplines (e.g., biology, art, business).

- j. When working on a computational thinking problem I find it useful to brainstorm about solution strategies before writing code.
- k. I find the challenge of solving computational thinking problems motivating.
- l. When studying computational thinking, I relate the important information to what I already know rather than just memorizing it the way it is presented.
- m. I enjoy solving computational thinking problems.
- n. Reasoning skills used to understand computational thinking can be helpful to me in my everyday life.
- o. Learning computational thinking is just about learning how to program in different languages.
- p. When I am working on a computational thinking program, I try to decide what reasonable output values would be.
- q. When I'm trying to learn something new in computational thinking, I find it useful to write a small program to see how it works.
- r. A significant problem in learning computational thinking is being able to memorize all the information I need to know.
- s. We use this statement to discard the surveys of people who are not reading the questions. Please select "Agree" for this question to preserve your answers.
- t. Understanding computational thinking basically means being able to recall something you've read or been shown.
- u. If I get stuck on a computational thinking problem, there is no chance I'll figure it out on my own.

- v. The subject of computational thinking has little relation to what I experience in the real world.
- w. There is usually only one correct approach to solving a computational thinking problem.
- x. To learn computational thinking, I only need to memorize solutions to sample problems.
- y. I worry that mistakes I make when writing a program may damage my computer.
- z. I am interested in learning more about computational thinking.
- aa. Females are as good as males at programming.
- bb. Studying computer science is just as appropriate for women as for men.
- cc. I would trust a woman just as much as I would trust a man to figure out important programming problems.
- dd. Women certainly are logical enough to do well in computational thinking.
- ee. It's hard to believe a female could be a genius in computational thinking.
- ff. It makes sense that there are more men than women in computational thinking.
- gg. I would have more faith in the answer for a programming problem solved by a man than a woman.
- hh. Women who enjoy studying computer science are a bit peculiar.
- ii. Women and men can both excel in careers that involve computing.
- jj. I would like to take courses in computing.
- kk. The skills in this study will be useful in my life.
- ll. The skills in this study will be useful in my career.
- mm. I know how to use programming to communicate with others.

nn. I know how to use programming to communicate with programmers.

14. Describe your attitude/view towards programming.

15. Choose one of the following five choices that best expresses your feeling about the statement. If you have no strong opinion, choose "Neutral".

Scale: Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

- a. After I study a topic in computational thinking and feel that I understand it, I have difficulty solving problems on the same topic.
- b. Errors generated by computers are random, and when they happen there's not much I can do to understand why.
- c. If I want to apply a method used for solving one computational thinking problem to another problem, the problems must involve very similar situations.
- d. I can usually figure out a way to solve computational thinking problems.
- e. When I solve a computational thinking problem, I break it into smaller parts and solve them one at a time.
- f. I do not spend more than five minutes stuck on a computational thinking problem before giving up or seeking help from someone else.
- g. There are times I solve a computational thinking problem more than one way to help my understanding.
- h. I think about the computational thinking I experience in everyday life.
- i. Tools and techniques from computational thinking can be useful in the study of other disciplines (e.g., biology, art, business).
- j. When working on a computational thinking problem I find it useful to brainstorm about solution strategies before writing code.

- k. I find the challenge of solving computational thinking problems motivating.
- l. When studying computational thinking, I relate the important information to what I already know rather than just memorizing it the way it is presented.
- m. I enjoy solving computational thinking problems.
- n. Reasoning skills used to understand computational thinking can be helpful to me in my everyday life.
- o. Learning computational thinking is just about learning how to program in different languages.
- p. When I am working on a computational thinking program, I try to decide what reasonable output values would be.
- q. When I'm trying to learn something new in computational thinking, I find it useful to write a small program to see how it works.
- r. A significant problem in learning computational thinking is being able to memorize all the information I need to know.
- s. We use this statement to discard the surveys of people who are not reading the questions. Please select "Agree" for this question to preserve your answers.
- t. Understanding computational thinking basically means being able to recall something you've read or been shown.
- u. If I get stuck on a computational thinking problem, there is no chance I'll figure it out on my own.
- v. The subject of computational thinking has little relation to what I experience in the real world.

- w. There is usually only one correct approach to solving a computational thinking problem.
- x. To learn computational thinking, I only need to memorize solutions to sample problems.
- y. I worry that mistakes I make when writing a program may damage my computer.
- z. I am interested in learning more about computational thinking.
- aa. Studying computer science is just as appropriate for women as for men.
- bb. I would trust a woman just as much as I would trust a man to figure out important programming problems.
- cc. Women certainly are logical enough to do well in computational thinking.
- dd. It's hard to believe a female could be a genius in computational thinking.
- ee. It makes sense that there are more men than women in computational thinking.
- ff. I would have more faith in the answer for a programming problem solved by a man than a woman.
- gg. Women who enjoy studying computer science are a bit peculiar.
- hh. Women and men can both excel in careers that involve computing.
- ii. I would like to take courses in computing.
- jj. The skills in this study will be useful in my life.
- kk. The skills in this study will be useful in my career.
- ll. I know how to use programming to communicate with others.
- mm. I know how to use programming to communicate with programmers.

16. For the following statements, please indicate how true each is for you.

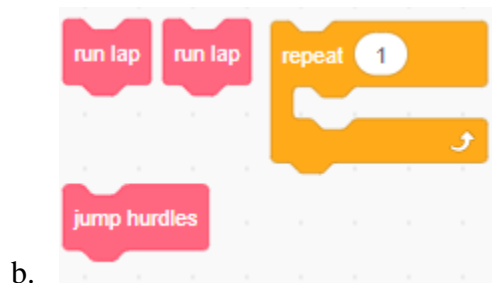
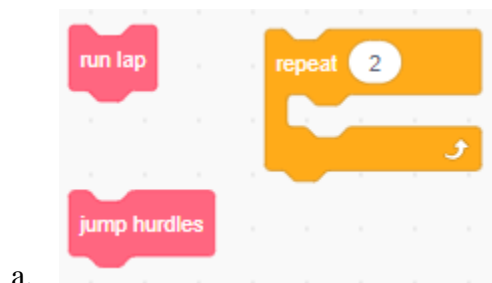
Scale: Not at all true (1) | 2 | 3 | Somewhat true (4) | 5 | 6 | Very true (7)

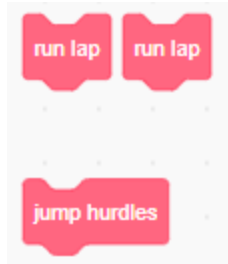
- a. I would recommend that children in my family learn Computational Thinking
- b. I would advocate for Computational Thinking to be taught in schools
- c. I would recommend children in my family attend a CT camp or after-school program
- d. I would be able to offer help to a child learning CT
- e. I have ideas for how I might use Computational Thinking at work
- f. I would ask an employer for CT training
- g. I believe I could successfully learn computational thinking
- h. I am interested in learning more about computational thinking

B.3 Pretests and Posttests

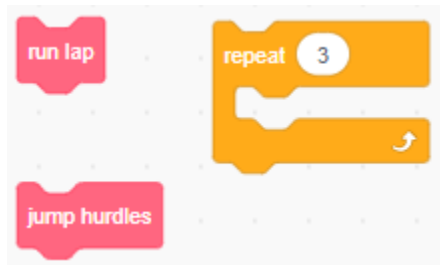
Example isomorphic pretest (CT concept *looping*)

1. What is your SAGE Username (used to login to <https://uat.cu-sage.org>)?
2. In a track and field event, players complete a lap, then jump over hurdles, then run another lap. Which of the following code elements help to achieve this task using the fewest blocks?





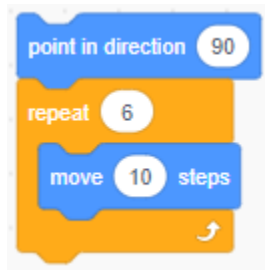
c.



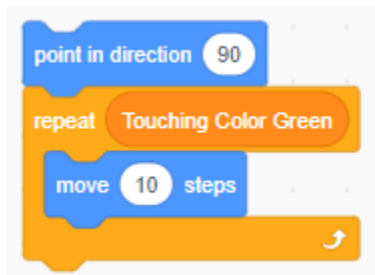
d.

3. Which of the following blocks would be best to use in the following situation:

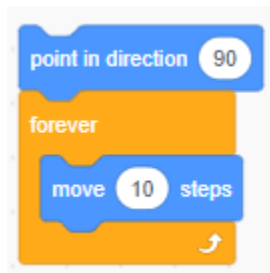
A sprite starts at coordinates (0,0). A green turtle sits at coordinates (50,0). The sprite stops moving when it reaches the turtle.



a.



b.



c.

The code block consists of three parts: a blue 'point in direction' block with the value '90', an orange 'repeat until' loop block with the condition 'Touching Color Green = True', and a blue 'move 10 steps' block inside the loop. The entire sequence is enclosed in an orange loop tail with a refresh icon.

d.

4. How many steps would be moved after the green flag is clicked?

The code block starts with an orange 'when green flag clicked' block. This is followed by a blue 'move 10 steps' block. Then there is an orange 'repeat' loop block with the value '10'. Inside the loop is a blue 'move 10 steps' block. The entire sequence is enclosed in an orange loop tail with a refresh icon.

- a. 100
- b. 30
- c. 20
- d. 110

5. You are ordering tacos in the restaurant. For some reason, you can only get one taco at a time, and you want 5 tacos in total. Which of the following code is the most efficient (uses the fewest blocks)?

```
when clicked
say Can I have a taco please?
say Can I have a taco please?
say Can I have a taco please?
say Can I have a taco please?
say Can I have a taco please?
```

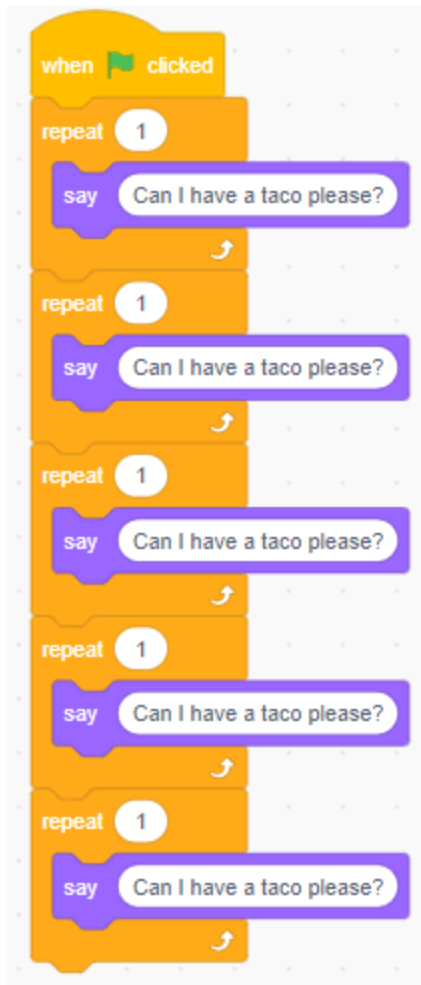
a.

```
when clicked
repeat 5
say Can I have a taco please?
```

b.

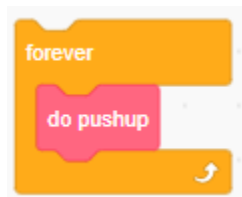
```
when clicked
repeat 5
say Can I have 5 tacos please?
```

c.

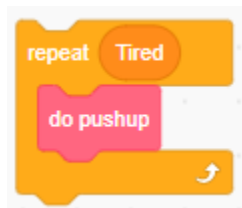


d.

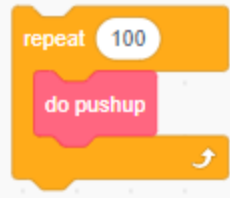
6. Noah is doing pushups and wants to stop only when he is tired. Which blocks would be best used for his workout?



a.



b.



c.



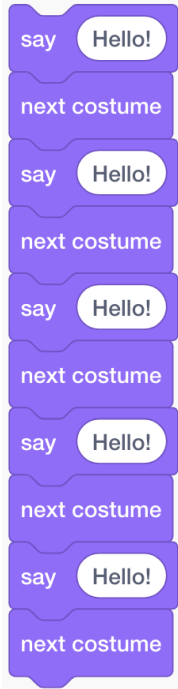
d.

7. How many times does the sprite say "hi!" in the following blocks?



- a. 4
- b. 12
- c. 16
- d. 8

8. What is the minimum number of blocks needed to replicate the following blocks in the most efficient way using looping?



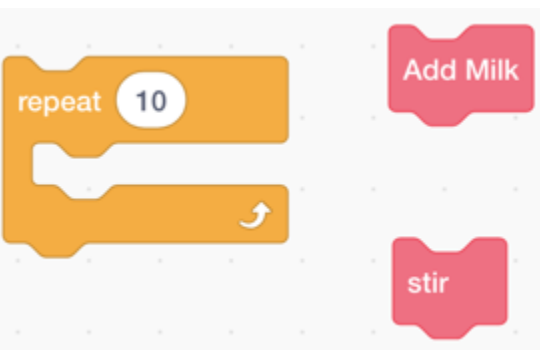
- a. 2
 - b. 3
 - c. 4
 - d. 5
9. A recipe specifies that a baker should repeat the process of adding milk and then stirring 10 times for cycles. Which of the following sets of elements is sufficient to complete this task while using the fewest blocks?



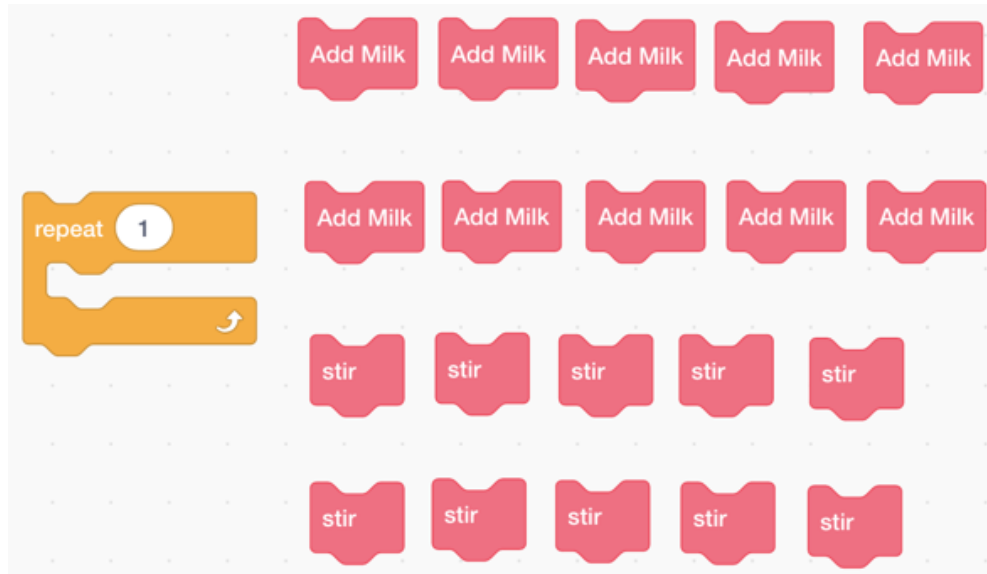
a.



b.



c.



d.

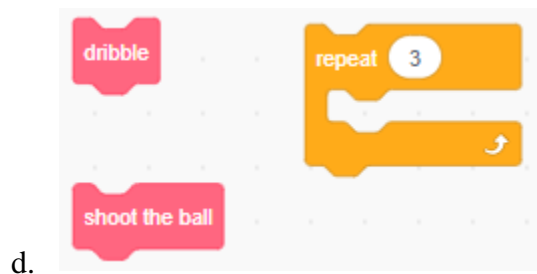
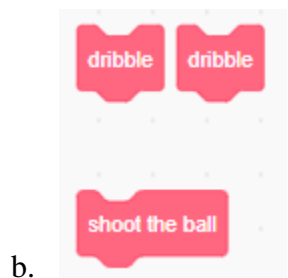
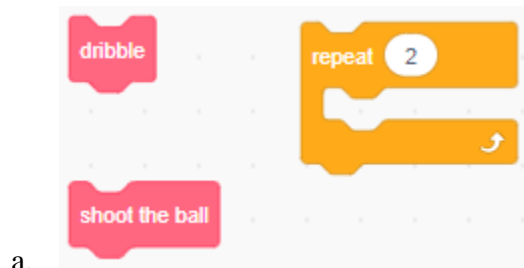
10. How many times does the sprite say "hi!" in the following blocks?



- a. 16
- b. 4
- c. 20
- d. 18

Example isomorphic posttest (CT concept *looping*)

1. What is your SAGE Username (used to login to <https://uat.cu-sage.org>)?
2. In a basketball, a player dribbles, then shoot the ball, then dribbles again. Which of the following code elements helps to achieve this task using the fewest blocks?



3. Which of the following blocks would be best to use in the following situation:
Bob wants to eat an apple. He looks outside the window and sees an apple on the tree that is still green. He waits to pick the apple until it turns red.

```

forever
  wait 1 seconds
  pick apple

```

a.

```

repeat until touching color red ?
  wait 1 seconds
  pick apple

```

b.

```

repeat Touching Color Red
  wait 1 seconds
  pick apple

```

c.

```

repeat until touching color red ?
  wait 1 seconds
  pick apple

```

d.

4. How many steps would be moved after the green flag is clicked?

```

when green flag clicked
  repeat 3
    move 10 steps
  move 10 steps

```

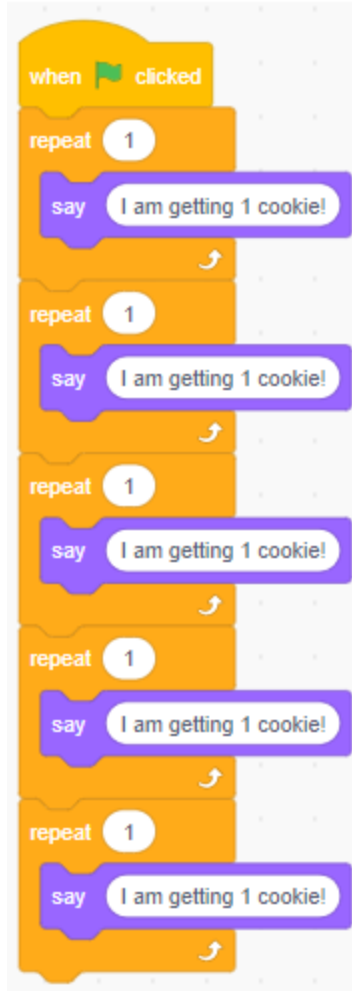
- a. 30
 - b. 23
 - c. 40
 - d. 20
5. There is a tray of cookies and you want to get 6 in total for you and your friends!
- However, you can only get one cookie at a time. Which of the following code is the most efficient (uses the fewest blocks)?

a.

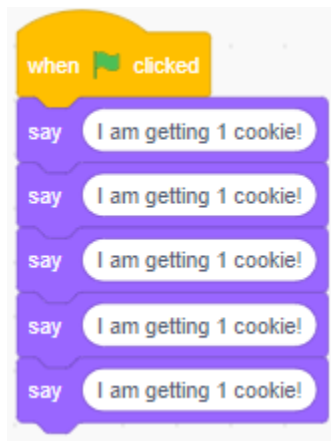
The code block for option a consists of three stacked blocks: a yellow 'when green flag clicked' block, an orange 'repeat 6' block, and a purple 'say I am getting 6 cookies!' block. The 'repeat' block is nested under the 'when green flag clicked' block, and the 'say' block is nested under the 'repeat' block.

b.

The code block for option b consists of three stacked blocks: a yellow 'when green flag clicked' block, an orange 'repeat 6' block, and a purple 'say I am getting 1 cookie!' block. The 'repeat' block is nested under the 'when green flag clicked' block, and the 'say' block is nested under the 'repeat' block.

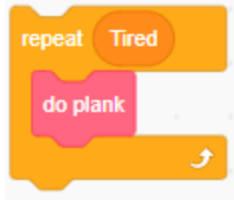


c.



d.

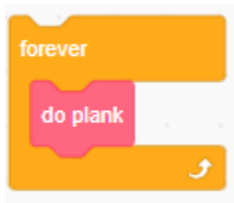
6. Jamie is doing planks and wants to stop only when she is tired. Which blocks would be best used for her workout?



a.



b.



c.



d.

7. How many times does the sprite make the sounds "Meow" in the following blocks?



a. 12

b. 36

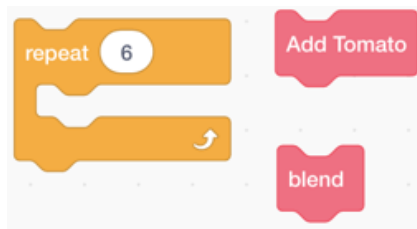
c. 18

d. 6


8. What is the minimum number of blocks needed to replicate the following blocks in the most efficient way using looping?

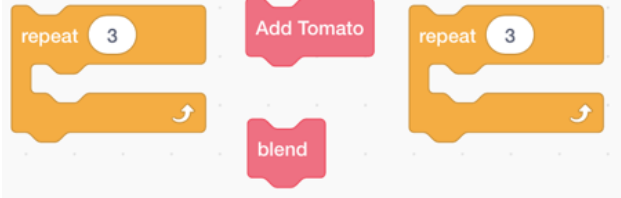


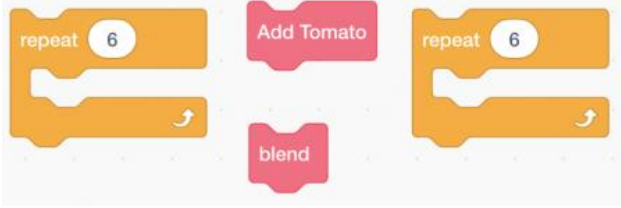
- a. 2
- b. 3
- c. 4
- d. 5
9. A recipe specifies that a chef should repeat the process of adding tomato and then blending 6 times for cycles. Which of the following sets of elements is sufficient to complete this task while using the fewest blocks?



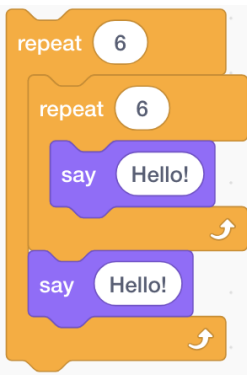
a.

b. 

c. 

d. 

10. How many times does the sprite say "Hello!" in the following blocks?



- a. 6
- b. 36
- c. 38
- d. 42

B.4 Self-reported Cognitive Load

Prompts in this survey required participants to respond with a value between 0-10.

1. What is your SAGE Username (used to login to <https://uat.cu-sage.org>)?
2. The topics covered in the activity were very complex.
3. The activity covered program code that I perceived as very complex.
4. The activity covered concepts and definitions that I perceived as very complex.
5. The instructions and/or explanations during the activity were very unclear.
6. The instructions and/or explanations were, in terms of learning, very ineffective.
7. The instructions and/or explanations were full of unclear language.
8. The activity really enhanced my understanding of the topic(s) covered.
9. The activity really enhanced my knowledge and understanding of computing / programming.
10. The activity really enhanced my understanding of the program code covered.
11. The activity really enhanced my understanding of the concepts and definitions.

B.5 Programming Behavior, Motivation, and CT Perception

1. What is your Prolific ID?
2. What is your SAGE Username (used to login to <https://uat.cu-sage.org>)?
3. For the following statements, please indicate how true each is for you.

Scale: Not at all true (1) | 2 | 3 | Somewhat true (4) | 5 | 6 | Very true (7)

- a. While I was working on the puzzles I was thinking about how much I enjoyed it.
- b. I did not feel nervous at all while doing the puzzles.
- c. I think I am pretty good at the puzzles.
- d. I found the puzzles very interesting.
- e. I felt tense while doing the puzzles.
- f. I think I did pretty well on the puzzles, compared to others.

- g. Doing the puzzles was fun.
 - h. I felt relaxed while doing the puzzles.
 - i. I enjoyed doing the puzzles very much.
 - j. I am satisfied with my performance on the puzzles.
 - k. I was anxious while doing the puzzles.
 - l. I thought the puzzles were very boring.
 - m. I felt pretty skilled at the puzzles.
 - n. I thought the puzzles were very interesting.
 - o. I felt pressured while doing these.
 - p. I would describe the puzzles as very enjoyable.
 - q. After working on the puzzles for awhile, I felt pretty competent.
4. For the following statements, please indicate how true each is for you. Programming is...

Scale: Not at all true (1) | 2 | 3 | Somewhat true (4) | 5 | 6 | Very true (7)

- a. Too difficult to understand
- b. Something I've wanted to learn
- c. Boring
- d. Something I did not know about
- e. Fun
- f. A foreign concept
- g. Enjoyable
- h. Important to know
- i. Easy to start
- j. An innate ability

- k. Too time consuming
 - l. Nerdy
 - m. Something that takes practice
5. Describe your attitude/view towards programming now, after the learning experience.
6. Choose one of the following five choices that best expresses your feeling about the statement. If you have no strong opinion, choose "Neutral".

Scale: Strongly Disagree | Disagree | Neutral | Agree | Strongly Agree

- a. After I study a topic in computational thinking and feel that I understand it, I have difficulty solving problems on the same topic.
- b. Errors generated by computers are random, and when they happen there's not much I can do to understand why.
- c. If I want to apply a method used for solving one computational thinking problem to another problem, the problems must involve very similar situations.
- d. I can usually figure out a way to solve computational thinking problems.
- e. When I solve a computational thinking problem, I break it into smaller parts and solve them one at a time.
- f. I do not spend more than five minutes stuck on a computational thinking problem before giving up or seeking help from someone else.
- g. There are times I solve a computational thinking problem more than one way to help my understanding.
- h. I think about the computational thinking I experience in everyday life.
- i. Tools and techniques from computational thinking can be useful in the study of other disciplines (e.g., biology, art, business).

- j. When working on a computational thinking problem I find it useful to brainstorm about solution strategies before writing code
- k. I find the challenge of solving computational thinking problems motivating.
- l. When studying computational thinking, I relate the important information to what I already know rather than just memorizing it the way it is presented.
- m. I enjoy solving computational thinking problems.
- n. Reasoning skills used to understand computational thinking can be helpful to me in my everyday life.
- o. Learning computational thinking is just about learning how to program in different languages.
- p. When I am working on a computational thinking program, I try to decide what reasonable output values would be.
- q. When I'm trying to learn something new in computational thinking, I find it useful to write a small program to see how it works.
- r. A significant problem in learning computational thinking is being able to memorize all the information I need to know.
- s. We use this statement to discard the surveys of people who are not reading the questions. Please select "Agree" for this question to preserve your answers.
- t. Understanding computational thinking basically means being able to recall something you've read or been shown.
- u. If I get stuck on a computational thinking problem, there is no chance I'll figure it out on my own.

- v. The subject of computational thinking has little relation to what I experience in the real world.
- w. There is usually only one correct approach to solving a computational thinking problem.
- x. To learn computational thinking, I only need to memorize solutions to sample problems.
- y. I worry that mistakes I make when writing a program may damage my computer.
- z. I am interested in learning more about computational thinking.
- aa. Studying computer science is just as appropriate for women as for men.
- bb. I would trust a woman just as much as I would trust a man to figure out important programming problems.
- cc. Women certainly are logical enough to do well in computational thinking.
- dd. It's hard to believe a female could be a genius in computational thinking.
- ee. It makes sense that there are more men than women in computational thinking.
- ff. I would have more faith in the answer for a programming problem solved by a man than a woman.
- gg. Women who enjoy studying computer science are a bit peculiar.
- hh. Women and men can both excel in careers that involve computing.
- ii. I would like to take courses in computing.
- jj. The skills in this study will be useful in my life.
- kk. The skills in this study will be useful in my career.
- ll. I know how to use programming to communicate with others.
- mm. I know how to use programming to communicate with programmers.

7. One of the following two questions is applicable to your experience. Please answer the one that applies by starting your response with "1) " or "2) ".

1

If all puzzles presented the same palette(s) and types of feedback, how did your increasing familiarity with the format help you sharpen focus on learning the CT concept?

2

If the presentation of palette(s) and/or types of feedback varied across puzzles, how did your increasing familiarity with the CT concept help you overcome decremental support from the learning system?

8. One of the following two questions is applicable to your experience. Please answer the one that applies by starting your response with "1) " or "2) " or "3) ".

1

If you encountered when solving each puzzle only one block palette, what advantages did this offer when solving puzzles, and what challenges in learning the CT concept might you anticipate experiencing if blocks were instead organized across palettes representing various categories?

2

If you encountered when solving each puzzle only multiple palettes and never had available just one, what challenges in learning the CT concept did this present, and what advantages might be offered by presenting all of the needed blocks in a single palette?

3

If you encountered when solving puzzles some with one palette and some with multiple

palettes, please compare the experiences by describing any preference between the two types and the extent to which the combination of the two in some way facilitated learning the CT concept.

9. One of the following two questions is applicable to your experience. Please answer the one that applies by starting your response with "1) " or "2) ".

1

If you did not encounter when solving any of the puzzles orange and purple guidance and objective feedback with dynamically enabled/disabled blocks, how could the green/red feedback on the correct/incorrect placement of blocks be improved to facilitate learning the CT concept without making puzzle solving too simple?

2

If you encountered when solving any of the puzzles orange and purple guidance and objective feedback with dynamically enabled/disabled blocks, how could that feedback be improved to facilitate learning the CT concept without making puzzle solving too simple?

10. For the following statements, please indicate how true each is for you after the learning experience.

Scale: Not at all true (1) | 2 | 3 | Somewhat true (4) | 5 | 6 | Very true (7)

- a. I would recommend that children in my family learn Computational Thinking
- b. I would advocate for Computational Thinking to be taught in schools
- c. I would recommend children in my family attend a CT camp or after-school program
- d. I would be able to offer help to a child learning CT

- e. I have ideas for how I might use Computational Thinking at work
 - f. I would ask an employer for CT training
 - g. I believe I could successfully learn computational thinking
 - h. I am interested in learning more about computational thinking
11. If you believe children should be required to learn computational thinking, what age should they start and why? If instead you believe children should not be required to learn computational thinking, is there another new subject you believe should be required instead, or another existing one that should be emphasized more (please specify)?
12. What do you find most frustrating about the computational thinking learning experience?
13. How might you change the learning environment to make it more efficient or effective for learning computational thinking?
14. What did you like best about the computational thinking learning experience?
15. If you encountered difficulty during this experiment, have ideas for improvement, or otherwise would like to share comments, please do so in the field below.

Appendix C: Objective Editor

The objective editor grammar is provided below. Terminals in this grammar are described in more detail in Appendix C.2. The ↓ symbol represents a vertical connection between two blocks. The → symbol represents a horizontal connection between two blocks. Productions that have the suffix *opt* are optional. An assessment begins with the “assessment” non-terminal.

C.1 Grammar

assessment:

expectation ↓ trigger-list opt

expectation:

expect → actual → assert → matcher

actual:

actl-sprite
actl-block
actl-block-type
actl-key-pressed
actl-sprite-touch-sprite
actl-sprite-touch-color

assert:

asrt-should
asrt-should-not

matcher:

mtch-be-present
mtch-be-on-x-y
mtch-point-direction
mtch-move-steps
mtch-say

trigger-list:

trigger
trigger-list ↓ trigger

trigger:

condition action

condition:

cnd-if-pass

end-if-fail

action:

actn-say

actn-include-block

actn-exclude-block

actn-add-points

C.2 Terminal Details

Every terminal in the grammar corresponds to a block described below.

expect



The Expect block signifies the beginning of an assessment. It connects horizontally to an actual block and connects vertically to a trigger block.

actl-sprite



The Actual Sprite block is used to reference a sprite. It takes a single string argument for the ID of the sprite that is being referenced. It connects horizontally to an assert block.

actl-block



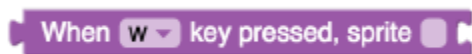
The Actual Block block is used to reference a block. It takes a single string argument for the ID of the block that is being referenced. It connects horizontally to an assert block.

actl-block-type



The Actual Block Type block is used to reference a block type. It takes a single string argument for the type of block that is being referenced. It connects horizontally to an assert block.

actl-key-pressed



The Actual Key Pressed block is used to express a mapping between a key on a keyboard being pressed and a sprite. It takes two arguments. The first argument must be selected from a drop-down menu that is pre-populated with single keyboard characters. The second argument is of type string for the ID of the sprite that is being referenced. It connects horizontally to an assert block.

actl-sprite-touch-sprite



The Actual Sprite Touching Sprite block is used to express a mapping between two sprites. It takes two arguments. Each argument is of type string for the ID of each of the sprites that are being referenced. It connects horizontally to an assert block.

actl-sprite-touch-color



The Actual Sprite Touching Color block is used to express a mapping between a sprite and a color. It takes two arguments. The first argument is of type string for the ID of the sprite being referenced. The second argument is from a drop-down menu that is pre-populated with a list of colors. It connects horizontally to an assert block.

asrt-should



The Should block is used to express an assessment where an action should occur. It connects horizontally to a matcher block.

asrt-should-not



The Should Not block is used to express an assessment where an action should not occur. It connects horizontally to a matcher block.

mtch-be-present



The Be Present block is used to express an assessment that verifies a sprite, block, or block type is present.

mtch-be-on-x-y



The Be on X and Y block is used to express an assessment that verifies a sprite is on specific coordinate in the Scratch stage. It takes two arguments. Each argument is of the integer data type and represents the X or Y coordinate.

mtch-point-direction



The Point in Direction block is used to express an assessment that verifies a sprite is pointing in a specific direction. It takes one argument. The argument must be chosen from a drop-down menu that is pre-populated with the following options: right, left, down, up.

mtch-move-steps



The Move Steps block is used to express an assessment that verifies a sprite moves a certain number of steps. It takes one integer argument represents the number of steps.

mtch-say



The Say block is used to express an assessment that verifies the sprite says (prints) a message to the screen. It takes one string argument that represents the message.

cnd-if-pass



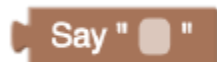
The block If Pass is used to express a trigger that is invoked if the assessment passes. It connects vertically to an expect or another trigger block. It connects horizontally to an action block.

cnd-if-fail



The block If Fail is used to express a trigger that is invoked if the assessment fails. It connects vertically to an expect or another trigger block. It connects horizontally to an action block.

actn-say



The Say block is used to say (print) a message if the previous trigger is invoked. It takes one string argument that represents the message that should be displayed.

actn-include-block



The Include Block block is used to include one block on the Scratch palette builder if the previous trigger is invoked. It takes one string argument that represents the ID of the block that should be included.

actn-exclude-blocks



The Exclude Block block is used to exclude one block on the Scratch palette builder if the previous trigger is invoked. It takes one string that represents the ID of the block that should be excluded.

actn-add-points



The Add Points block is used to update the assessment points. It takes one integer argument that represents the number of points to be added or subtracted (negative values are permitted).

Appendix D: Architecture Detail

Software layers constructed and activity flows supported are diagrammed below.

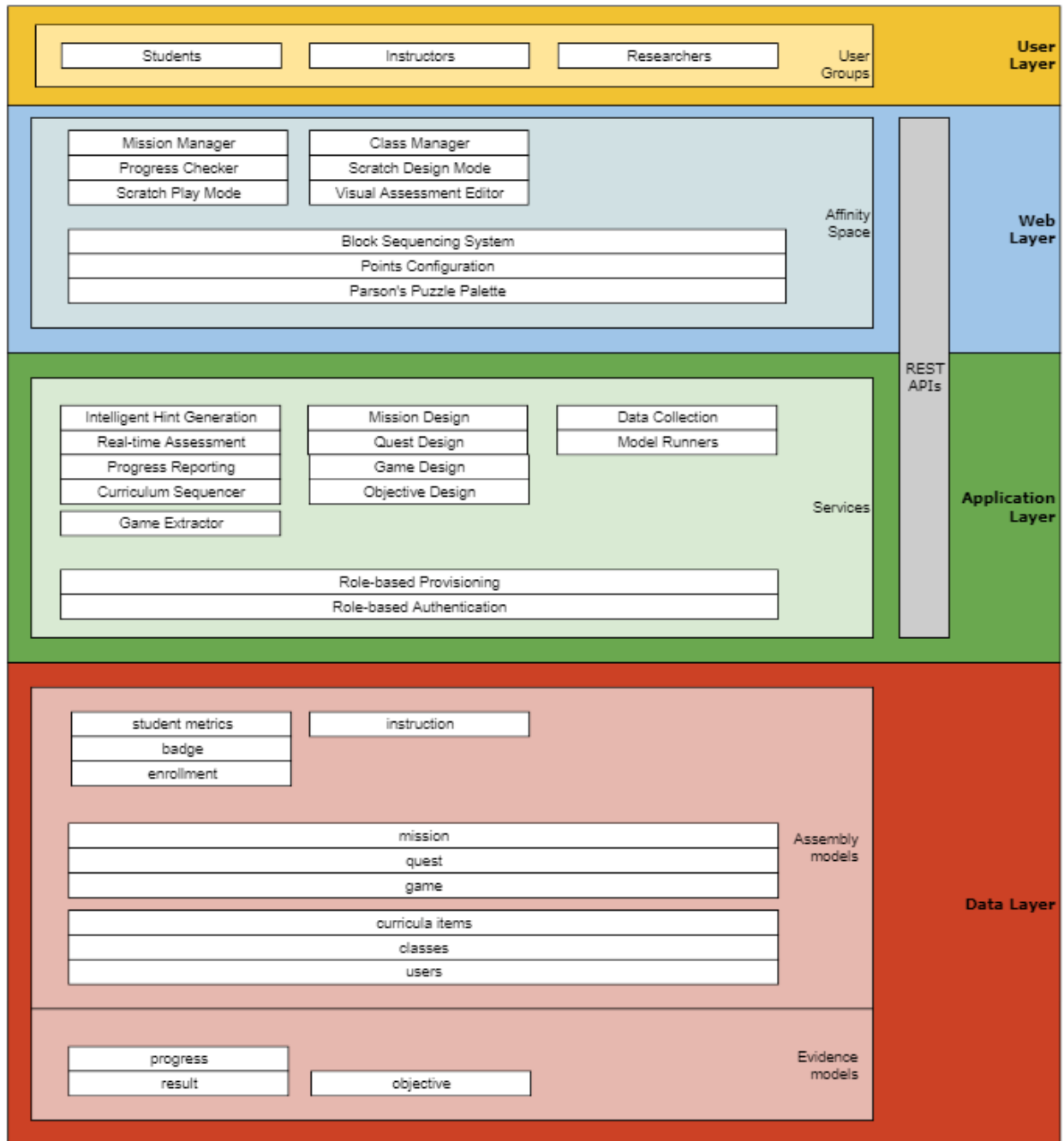


Figure D.1: SAGE software layers.

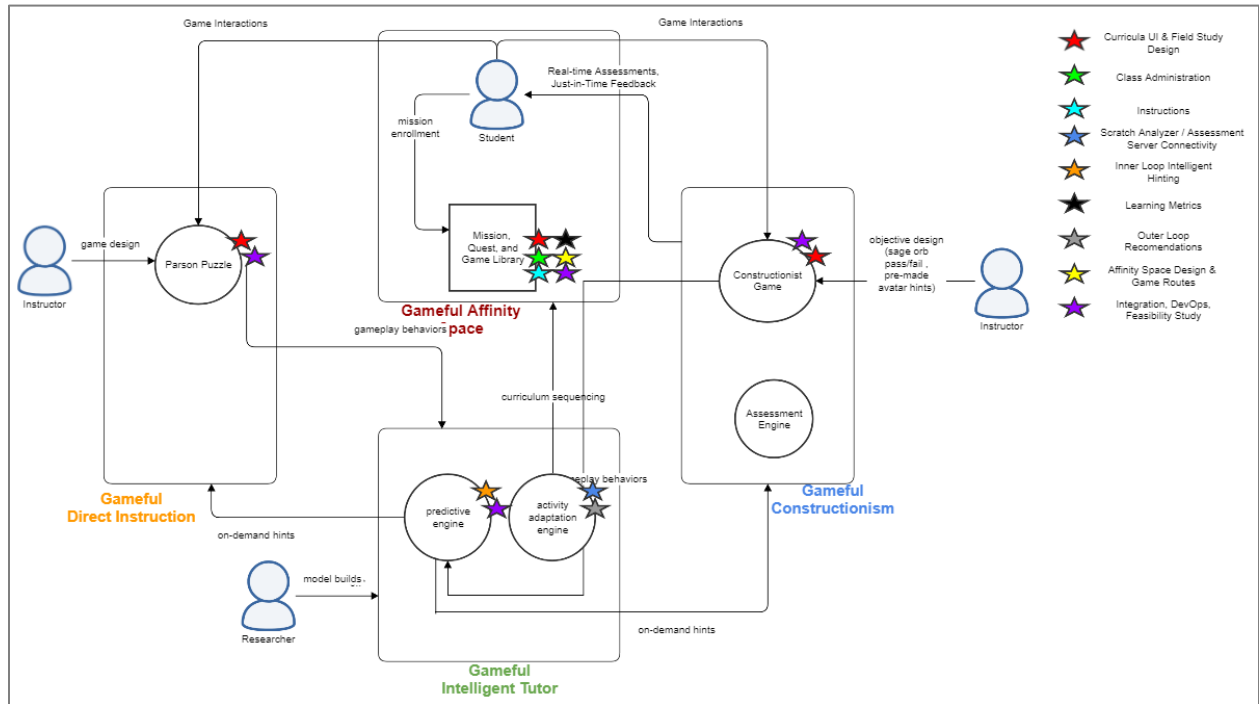


Figure D.2: SAGE activity flow.

Appendix E: Supplemental Related Work

We present here additional related work that has inspired the development of SAGE.

E.1 Computational Thinking

Since the publication of Wing's 2006 essay [1], several researchers have attempted to characterize CT:

- [D]ata collection, data analysis, data representation, problem decomposition, abstraction, algorithms and procedures, automation, parallelization, simulation [244].
- [D]efining, understanding, and solving problems, reasoning at multiple levels of abstraction, understanding and applying automation, and analyzing the appropriateness of the abstractions made [26].
- [D]ecomposition, pattern recognition, pattern generalization and abstraction, and algorithm design [245].
- [T]hinking to solve problems, automate systems, or transform data by constructing models and representations, concrete or abstract, to represent or to model the inner-working mechanism of what is being modeled or represented as an information process to be executed with appropriate computing agents. Such thinking is necessarily: logical; algorithmic; scientific; mathematical; analytical; engineering-oriented; creative [25].
- [A] problem-solving methodology that can interweave computer science with all disciplines, providing a distinctive means of analyzing and developing solutions to problems that can be solved computationally. With its focus on abstraction, automation, and analysis, computational thinking is a core element of the broader discipline of computer science [22].

In grades 6-8, these characterizations are situated in the context of an evolution in technology education inclusive of computer literacy and digital fluency. Educational technology evangelist Marc Prensky coined the classification, digital native, in 2001, as means by which to emphasize the manner in which internet-era students process information fundamentally differently from their predecessors [246]. Although accurately differentiating digital natives from their digital immigrant parents by identifying their desire for instant gratification and penchant for parallel processing, multi-tasking, randomly accessing information, hypertext thinking, and networking, he limits his analysis to the extent to which technologically savvy generations efficiently use computers, video games, and the internet, which are a set of competencies otherwise known as computer literacy [247,248]. Mitchell Resnick, leader of the Lifelong Kindergarten group at the MIT Media Lab, focuses instead on digital fluency, which he argues "requires not just the ability to chat, browse, and interact but also the ability to design, create, and invent with new media" [59]. This spotlight on digital construction naturally elicits an emphasis on learning programming, a skill which substantially expands the range with which students can express themselves [249]. Digital construction thus becomes the setting for students to program while developing problem-solving and design strategies, such as modularization and iterative design, which interlock with non-programming domains.

The resulting interdisciplinary educational experience ultimately underwrites an expansive, yet inclusive, orientation around CT, extending beyond both computer literacy and digital fluency to encompass analytical skills which enhance how we approach problems and how we leverage computing to augment our abilities [29]. Such a sweeping mandate reinforces Wing's compelling vision for how CS educators, researchers, and practitioners can act to change society's sometimes narrow image of the field; instead of equating CS with computer

programming and isolating widely applicable conceptual competencies in CS classes, we can expose the urgent need to foster a computational culture across the curriculum [25]. Existing curriculum standards, infrastructure deficiencies, and limited professional development opportunities for teachers to learn CT all present significant challenges [26], but there is a ripe opportunity for students to learn CT if we can support non-CS teachers effectively.

Programming, for instance, is reflexive with other domains; learning to code while exploring concepts from other subjects can ease the understanding of all involved topics concurrently [86,250]. We can, for example, demonstrate binary search when teaching how to find the roots of an equation, or introduce optimization when training students in Excel [25]. By consistently integrating CT where it fits in practice in established classroom curricula, we infuse a set of shared attitudes, values, goals, and practices which ultimately catalyze this computational culture.

E.2 Game-based Learning

CT's versatility as a glue that binds domain-specific expertise and technical innovation solicits the use of GBL as an instructional paradigm, since game content can support a diversity of subjects within existing curricula while providing students an opportunity to develop and apply CT synergistically [52]. The contextualized representations concretize CT's abstract concepts, resulting in both higher learning gains and increased engagement [88]. Furthermore, the multidisciplinary integration of CT helps students recognize appropriate use cases across domains, thereby instilling a more fluid kind of problem-solving than cultivated through siloed study [52]. By adaptively subdividing big problems into smaller ones solvable with the computational competence of each learner, games inculcate increased levels of self-efficacy. The associated elevated confidence inspires students to practice communicating with CT within

enjoyable social environments designed to encourage both collaborative problem-solving and iterative group reflection that enables not only the digestion of new learning, but also the collective construction of new knowledge [89].

Several literature reviews of the increasing volume of GBL research demonstrate the effectiveness with which games improve general learning [90,91,92,93,94], increase motivation [95,96,97], and improve performance [98,99,100,101], but close examinations of typical experimental designs reveal limited generalizability with results that apply only to the specific population under study [102]. The theoretical foundation from which these studies depart nonetheless has progressed substantially since video games' early years when edutainment gained prominence as "a sugarcoating of entertainment for the bitter medicine of education to become palatable" [105]. The interdisciplinary field of the learning sciences, which studies teaching and learning, most significantly has influenced the structure and robustness of GBL research, but several psychology subdomains, such as self-determination theory, also imprint modern educational game design. Our approach leverages these advances in theory, as well as the PPP and CVG frameworks, but focuses on modularized implementations applicable across block-based programming environments.

E.3 Implicit Assessment

Most current computer-based assessments reflect only a modest utilization of technology for testing purposes. DiCerbo and Behrens identified four levels of technology-enhanced assessments, ranging from level 1, which represents the most basic integration, essentially, paper-and-pencil tests, administered by computer, to level 4, in which the presentation of tasks within technology rich environments are distributed across a variety of experiences, and a constant stream of data is accumulated throughout the assessment ecosystem [251]. In contrast to

traditional snapshot testing that encourages cramming and short-term recall, this continuous assessment paradigm assures that the best way for students to do well is to do well every day. The distinction between formative and summative blurs, and assessments are elevated to the status of informative so that teachers and systems can leverage the results to determine how best to advance each student's learning.

Striving for such automated efficiencies started in 1960, when need for the conservation of teaching and computing resources led to the automatic correct/incorrect classification of student punch-card programs [252]. Progress in providing teachers more articulated evaluations continued in the 1980's with the utilization of testing engines [253], and toward the end of the 20th century, a focus on reducing the burden of time-consuming and error-prone teacher activity led to the development of ASSYST (ASsessment SYSTem) [254]. This automated grading system uses static analysis calculations, such as McCabe's cyclomatic complexity [255], to assess program correctness, efficiency, complexity, style, and test effectiveness, but importantly it includes humans as the directors of the assessment process. At the start of the 21st century, an emphasis on generating formative feedback available to both teachers and students accelerated with the introduction of tools like Web-CAT [256], which grades students on how well they test their code, and Marmoset, which establishes a suite of tests executed against snapshots of students' work [257]. These innovations led us to current and future states in which real-time predictive interventions steer students toward mastery [258].