ABSTRACT

AKRAM, BITA. Assessment of Students' Computer Science Focal Knowledge, Skills, and Abilities in Game-Based Learning Environments. (Under the direction of Drs. James C. Lester and Eric N. Wiebe).

Computer science has become a critical competency for all students. With national K-12 education initiatives such as CSforAll, block-based programming environments have emerged as widely used tools for teaching computational thinking (CT) and computer science (CS) concepts to novice programmers. A key challenge posed by block-based programming environments is assessing students' CT and programming competencies within these environments. While this assessment role would traditionally be fulfilled by a teacher, the dramatic growth of computer science as both a necessary skill and area of interest for students presents the need for automated and semi-automated assessment tools that support teachers in this role. Developing assessment methods that can evaluate students' CS focal knowledge, skills, and abilities help teachers evaluate students' learning and provide appropriate scaffolding.

It is also important to foster students' motivation and engagement toward learning CT and programming. Game-based learning environments utilize the motivational elements of games to promote students' learning and engagement. Studies have shown that for a broad range of student populations and subject matters, students who engage in game-based learning experience higher motivation compared to those who learn with conventional methods. Intelligent game-based learning environments integrate the adaptive learning support of intelligent tutoring systems and the motivational elements of games to provide learners with adaptive scaffolding without interfering with their state of flow. As a result, intelligent game-based learning environments have emerged as a promising tool that can provide students with

engaging learning opportunities while providing educators with ample opportunity to unobtrusively assess students' knowledge and skills.

This dissertation introduces a novel temporal analytics framework for stealth assessment based on students' problem-solving strategies. The strategy-based temporal analytic framework incorporates long short-term memory networks to analyze students' problem-solving behaviors across consecutive tasks to inform an evidence model used in stealth assessment. When evaluating the temporal analytic framework on a dataset collected from middle grade students' interactions with the ENGAGE game-based learning environment, results show that it outperforms competitive baseline models with respect to predictive accuracy when used for predicting students' post-test scores.

Furthermore, this dissertation presents the development and evaluation of a stealth assessment framework that utilizes a blended hypotheses-driven assessment design approach to assess middle grade students' CS focal knowledge, skills and abilities (FKSAs) within game-based learning environments. We follow an Evidence-centered design-based assessment to evaluate students' CS FKSAs based on evidence extracted from their programming trajectories in a block-based programming environment. The results reveal distinctive patterns in students' approaches to problem solving for CT challenges, which provides first steps toward identifying and assessing productive CS practices.

Finally, this dissertation presents a semi-automated assessment framework that utilizes a corpus of graded programming artifacts to infer students' ability to develop a bubble sort algorithm based on their submitted programming artifacts. The dataset containing graded submissions is prone to noise due to graders' subjective idea about the quality of students' submitted artifacts. Utilizing a systematic approach to labeling the training dataset and

applying Gaussian process regression help reduce this noise. The semi-automated assessment framework utilizes a supervised learning approach that infers the algorithmic quality of a submitted programming artifact based on its hierarchical and ordinal encoded *n*-grams. Our results demonstrate the effectiveness of the proposed approach in inferring artifacts' algorithmic quality. Furthermore, according to our results, Gaussian process models outperform other models that are unable to accommodate the level of noise in our dataset. Overall, our stealth assessment framework has shown to be an effective approach to unobtrusively assess students' CS and CT competencies within a game-based learning environment.

Assessment of Students' Computer Science Focal Knowledge, Skills, and Abilities in Game-Based Learning Environments

by
Bita Akram

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina
2019

APPROVED BY:

_____	_____
Dr. James C. Lester. Committee Co-Chair	Dr. Eric N. Wiebe Committee Co-Chair
_____	_____
Dr. Thomason W. Price	Dr. Bradford W. Mott

DEDICATION

Dedicated to my academic advisors Drs. James Lester and Eric Wiebe for their invaluable guidance, unconditional support, and genuine attentiveness.

BIOGRAPHY

Bita Akram was born in Tehran, Iran on July 23, 1990. She obtained her Bachelor of Science from Sharif University of Technology in 2013, where she majored in computer engineering. After finishing her undergraduate studies, she immediately started her Master of Science in Computer Science at University of Calgary where her research was focused on devising algorithms for conducting accurate and efficient scientific data visualization.

During her undergraduate and master's studies, Bita both taught K-12 level mathematics and served as TA for several undergraduate computer science (CS) courses. There she realized her strong passion for education and, in particular, CS education. Therefore, she decided to pursue a PhD in educational technology where she could join her enthusiasm for computer science with her passion for education.

She started her PhD at North Carolina State University in 2013. She later joined the Department of Computer Science and the Intellimedia group to continue her studies under the supervision of Dr. James Lester and Dr. Eric Wiebe. During her PhD studies she remained a research assistant at both Center for Educational Informatics and Friday Institute for Educational Innovation with a research focus on improving access and quality of CS Education by both developing innovative CS curricula and designing advanced learning technologies for instructional support. During her PhD studies, she has worked extensively on building stealth assessment algorithms for assessing students' computer science skills and problem-solving strategies in game-based learning environments. As part of her PhD work, she has also designed several computational modeling curricula for middle-grades classrooms and has held many professional development workshops for K-12 teachers across the country.

She accepted a teaching faculty position at North Carolina State University in March 2019 to teach computer science while she stays engaged with research on advanced learning technologies for CS education.

ACKNOWLEDGEMENTS

My special thanks go to my advisors, Drs. James Lester and Eric Wiebe. Their endless support and invaluable guidance during my PhD studies made this possible. They have supported me in doing research in the area I was most excited about and guided me through taking the most intriguing career path. I want to thank them for their numerous hours of support, their belief in my academic path, and the professional model they laid before me.

I also want to thank others who have provided guidance to my academic path. My committee members: Dr. Thomas Price for his early and insightful directions to my research, Dr. Bradford Mott for his accurate comments in study design and implementation, and Dr. Sarah Carrier for her kindness and flexibility. I also want to express my special gratitude to Drs. Min Chi, Kristy Boyer, Collin Lynch, Tiffany Barnes, Noboru Matsuda, Margareta Thomson, and Roger Azevedo for being invaluable sources of advice at different stages of my research and academic path.

Also, I want to thank senior researchers in the Center for Educational Informatics who have provided valuable advice to my research including Drs. Wookhee Min, Hamoon Azizsoltani, Jonathan Rowe, Andy Smith, Randall Spain, and Seung Lee. Additionally, special thanks go to my colleagues Andrew Emerson, Dan Carpenter, Nathan Henderson, and Jennifer Houchins for their helpful review of my dissertation. I also want to thank my other colleagues at the Center for Educational Informatic and Friday institute of Educational Innovation at NC State and the LearnDialogue lab at University of Florida for their help, encouragement, and collaboration. These people include but are not limited to: Niki Gitinabard, Farzaneh Khoshnevisan, Adam Gaweda, Kyungjin Park, Ziwei Wu, Robert Sawyer, Pengcheng Wang, Jennifer Tsan, Michael Geden, Robert Taylor, Sandra Taylor, Barry Liu, Kirby Culberston,

Sarah Reaves, Megan Frankosky, Yihuan Dong, Nick Lytle, Veronica Cateté, Danielle Boulden, Osman Aksit, Cody Smith, Jessica Vandenberg, Zarifa Zakaria, and Arif Rachmatullah.

Finally, I want to thank my family and friends, who stood by me through it all and provided me with unconditional support, love, and trust. I want to particularly thank my sister, Sahar, who has always been my role model, my faithful encourager, and my best friend.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1:

INTRODUCTION

Recent years have seen a growing interest in K-12 computer science curricula. As technology transforms our everyday lives at a rapid rate, it is creating an increasing demand for skilled computer scientists. However, there are not enough students attending and graduating from computer science related fields to fill these positions (*2018 State of Computer Science Education Policy and Implementation Advocacy Coalition*, 2018). A solution to this rapidly increasing need is to provide appropriate and inclusive CS education to K-12 students to encourage and enable them to join this field. There are many challenges involved in providing effective CS education to students. Designing appropriate curricula, offering sufficient scaffolding and support, providing access to CS education opportunities, and promoting inclusion are some of the important challenges that CS education research aims to answer.

Among the most important challenges faced by CS education research is the assessment of students' CS and CT competencies. Effective assessment of students' CS and CT focal knowledge, skills, and abilities enables educators to provide students with the appropriate scaffolding and support they need. However, due to the dramatic growth in students' interest in computer science, providing every student with individualized, high-quality, and timely support and feedback has become more challenging. Effective automated and semi-automated assessment tools can help bridge the gap between the demand for support and restricted resources by providing adaptive formative and summative feedback.

Another important challenge in CS education is fostering motivation among students and engaging them in computational thinking activities. This is particularly important for CS

minority students. An effective way to improve students' motivation and engagement while learning is to use game-based learning environments (Clark, Tanner-Smith, & Killingsworth, 2016). Recent years have seen significant growth in the design and development of game-based learning environments. A growing body of research is investigating the potential of such environments to foster students' learning and engagement (Clark et al., 2016; Rowe, Shores, Mott, & Lester, 2011; Sabourin & Lester, 2014). As an example, many studies have shown that learning through game-based environments results in higher motivation for learning compared to learning through conventional methods (Cordova & Lepper, 1996; Shute & Ventura, 2013). Intelligent game-based learning environments can take advantage of motivational elements of games in addition to adaptive learning support of intelligent tutoring systems (ITSs) (Jackson & McNamara, 2013). Similar to ITSs, intelligent game-based learning environments use data obtained from students' learning interactions to build models of students' cognitive, affective, and metacognitive states (Lester et al., 2013; Min, Mott, Rowe, Liu, & Lester, 2016; Wouters, Van Nimwegen, Van Oostendorp, & Van Der Spek, 2013). The resulting learner models can then be used to provide students with adaptive problem-solving scenarios, cognitive feedback, and affective support (Brusilovsky & Millán, 2007; Min et al., 2017).

The process of inferring students' cognitive, affective, and metacognitive states by collecting performance data from their interactions with an education environment is called stealth assessment (Shute, 2011). Utilizing a stealth assessment framework, we can unobtrusively assess students' CS and CT focal knowledge skills and abilities (FKSAs) as they interact with learning environments and provide formative feedback. Moreover, in the future, these assessments can be used in dashboards to help teachers provide individualized

feedback to students. Consequently, game-based learning environments provide a robust platform for engaging students in learning CS and CT concepts, while enabling educators to unobtrusively assess their competencies.

Following a stealth assessment approach, we analyze data from students' interactions with ENGAGE, a well-structured, problem-based, game-based learning environment that is designed to teach middle-school students' the basics of block-based programming and computational thinking to assess their problem-solving strategies and CS knowledge, skills, and abilities.

1.1 Problem

This work focuses on assessing two important aspects of CS and CT proficiencies: problem-solving strategies as demonstrated in students' interactions with a computational thinking challenge, and proficiency in CS FKSAs as demonstrated in programming artifacts. Research has shown that problem-solving strategies can profoundly affect students' learning outcomes (Eagle & Barnes, 2014; Rowe, Baker, Asbell-Clarke, Kasman, & Hawkins, 2014). Further, problem-solving proficiency is an important skill for solving computational thinking problems (Sengupta et al., 2013; Weintrop et al., 2016). For example, problem-solving is introduced as one of the four main practices in Weintrop et al.'s (2016) computational thinking framework.

A key application of CS is programming. Hence, it is important to develop reliable assessment tools to effectively evaluate students' programming practices and resulting artifacts (Grover et al., 2017) as evidence of CS knowledge, skills and abilities. We utilize a systematic approach to rubric design to effectively assess students' CS FKSAs based on their interactions with a game-based learning environment. We then utilize a supervised learning

approach to build a semi-automated framework for assessing students' programming artifacts' algorithmic quality based on a training dataset labeled by the devised rubric. Conducting effective assessment is the first step in providing students with the adaptive scaffolding and support they need to thrive in this field. Furthermore, assessing students' CS and CT competencies provides instructors with summative feedback about their instruction.

In this dissertation, we present a stealth assessment framework for assessing students' CS and CT competencies as demonstrated through their interactions with the ENGAGE game-based learning environment and resulting programming artifacts. Below, we explain our approach for conducting stealth assessment corresponding to each aspect of CS and CT proficiencies.

1.2 Approach

To identify students' problem-solving strategies when solving a computational thinking problem, we present an approach to stealth assessment that leverages a novel temporal analytics framework that utilizes students' problem-solving strategies to infer their post-test performance. Building on findings that show problem-solving strategies significantly influence learning outcomes (Eagle & Barnes, 2014; Rowe et al., 2014), we cluster students based on a set of *n*-gram encoded features and investigate whether problem-solving strategies identified from clustering students' interaction patterns can improve the predictive accuracy of evidence models for stealth assessment.

We further utilize these clusters to assess their learning of the corresponding CT concept. After clustering students based on their problem-solving behaviors, we predict their post-test performance using their cluster assignments across several tasks as predictive features for a suite of classifiers. This approach is based on the intuition that, as students

progress through a series of learning tasks, their choice of strategy affects their learning outcomes. For example, if a student incorporates a trial-and-error approach for initial tasks when she is not familiar with the challenge and later switches to a more effective strategy, her strategy shift might be an indicator of learning, as demonstrated through higher post-test scores. We hypothesize that drawing inferences about strategy shifts may serve as an efficient basis for accurate predictions of learning performance.

Because both strategy and strategy shifts are determined through dependent interactions over time, we propose a strategy-based temporal analytics approach to stealth assessment based on long short-term memory networks (LSTMs). In this approach, we develop predictive models that capture the temporal dependencies between students' dynamically changing problem-solving behaviors. Findings demonstrate that the strategy-based temporal analytics framework achieves higher predictive accuracy compared to baseline models that do not capture strategic temporal dependencies. Furthermore, our results show that the strategy-based temporal analytics framework utilizing both student problem-solving behavior traces and pre-test performance outperforms a model that uses only pre-test data. The results suggest that strategy-based temporal analytics can serve as the basis for effective stealth assessment in game-based learning.

To effectively assess students' proficiency of CS FKSAs, we propose an evidence-centered design assessment approach (Mislevy & Haertel, 2007) to assess middle-school students' CS competencies utilizing evidence from students' sequential and computational problem-solving activities. We collect data from students' interactions with a block-based programming interface embedded in the ENGAGE game-based learning environment in a pair-programming setup. Deriving evidence from sequences of students' proposed solutions, we

assess CS practices, such as generating appropriate algorithms and programming artifacts, debugging and refinement strategies, and appropriate usage of computer science constructs, such as loops and conditionals. Qualitative analyses reveal distinctive patterns in students' approaches to computational problem solving that can inform middle school instructors of the methods that best promote students' CS competencies in their classrooms. We further revise the rubric based on preliminary results obtained from piloting the rubric on data collected from students' paired gameplay. We then apply the refined rubric on data collected from a larger population of students who interacted with the game individually.

One of the goals of this dissertation is to present a semi-automated assessment framework that can evaluate students' CS competencies based on their submitted programming artifacts. To this end, we devise a supervised learning approach that utilizes the revised rubric to label a corpus of programming artifacts. Applying this rubric reduces the amount of noise in the training dataset, which results from graders' subjective opinions about the quality of a programming artifact. Utilizing an ECD-based rubric, we can reduce this noise by guiding instructors to follow a unified approach to assessing programming artifacts.

We further encode programming artifacts as structural *n*-grams to extract essential structural and semantic information within them. Finally, we utilize a variety of well-established regression models including lasso, ridge, and Gaussian process regression models on the generated feature set to infer programming artifacts' qualities. Although we can reduce the noise in our training dataset by utilizing an ECD-based rubric, we cannot completely eliminate this noise. In this dissertation, we utilize Gaussian process regression models to handle the remaining noise in our dataset. To further evaluate the effectiveness of our feature set, we also apply the regression models on a bag-of-words-based baseline feature

set. We discuss our proposed stealth assessment framework design and evaluation in details in Chapters 4-7.

1.3 Thesis Statement and Hypotheses

The objective of this dissertation is to build an effective stealth assessment framework to assess students' CT proficiencies and CS FKSAs while interacting with game-based learning environments. In this dissertation, we aim to investigate the following thesis statement:

> *Stealth assessment is an effective approach for unobtrusive, effective and accurate evaluation of students' CT proficiencies and CS focal knowledge, skills, and abilities in game-based learning environments.*

To effectively evaluate this thesis statement, the individual components of the framework should be explored. A key challenge posed by stealth assessment is translating the raw data into meaningful representations necessary for modelling students' skills and performance (Shute & Ventura, 2013). Another important factor that must be considered when addressing this challenge is the dependency among students' consecutive actions within a learning environment. Focusing on these aspects, the proposed dissertation will specifically investigate the following hypotheses:

- **H1.** Students' computational thinking proficiencies can be effectively inferred by utilizing their problem-solving strategies identified from their interactions with a computational thinking problem in a well-structured game-based learning environment.

- **H1.1** Clustering students' patterns of interaction within a well-structured educational game while solving a computational thinking problem can identify their problem-solving strategies.
- **H1.2** Students' problem-solving strategies over similar tasks can be utilized to infer their knowledge of the corresponding computational thinking proficiencies.
- **H1.3** Utilizing the temporal dependency among students' choice of problem-solving strategies over consecutive, similar tasks can improve the prediction accuracy of the model in regard to students' post-test performance.

- **H2.** Students' CS FKSAs can be effectively assessed based on their programming trajectories.
  - **H2.1** Students' knowledge of essential CS constructs can be assessed based on their block-based programming artifacts.
  - **H2.2** Students' common testing and refining approaches while solving a computational problem using block-based programming can be assessed using their programming trajectories.
  - **H2.3** The effectiveness of students' testing and refining approaches can be assessed by exploring the relationship between their knowledge of essential CS constructs, in-game behavior, and their testing and refining patterns.

- **H3.** Utilizing a supervised learning approach, programming artifacts' algorithmic quality can be automatically assessed based on their submitted block-based

programming artifacts when solving an algorithmic problem in a well-structured game-based learning environment.

- **H3.1** Important structural and ordinal information within students' block-based programming artifacts can be preserved by performing appropriate feature engineering.
- **H3.2** Students' algorithmic proficiency can be accurately inferred from their submitted programming artifacts encoded as structural *n*-grams.
- **H3.3** The accuracy of the predictive model can be improved by considering the noise in the graded assignments.

1.4 Contributions

We have designed a comprehensive stealth assessment framework to effectively and unobtrusively assess students' CT proficiencies and CS FKSAs from data collected from their interactions with a computational thinking game-based learning environment. The contributions are as follows:

- A novel strategy-based temporal analytics framework to assess students' computational thinking knowledge and skills (Akram et al., 2018).
  - An assessment framework for identifying students' problem-solving strategies based on their interactions with a game-based learning environment while solving well-structured computational thinking problems.
  - An assessment framework for assessing students' conceptual understanding of CT concepts based on their identified problem-solving strategies over consecutive relevant tasks by utilizing an LSTM network that takes into account the temporal dependency among students' choice of strategy.

- A qualitative assessment framework for assessing students' CS focal knowledge, skills, and abilities, such as the ability to effectively test and refine their programs, and the ability to develop appropriate algorithms (Akram et al., 2019).
    - An assessment framework for assessing students' knowledge of essential CS constructs such as algorithms, loops, and conditionals.
    - An assessment framework for identifying students' patterns of testing and refining their programming artifacts.
    - An assessment framework for assessing students' problem-solving efficiency.
- A systematic and automated framework for accurate and effective assessment of programming artifacts' algorithmic quality based on students' generated block-based programming artifacts. This framework can effectively handle the noise in the training dataset by following a blended hypothesis-driven assessment design approach for assessing and labeling students' data and by incorporating a supervised learning technique that can deal with noise.
    - A feature-engineering approach for effectively extracting structural features from students' block-based programming artifacts. To capture hierarchical and ordinal information within the programming artifacts, this approach encodes students generated artifacts as structural *n*-grams.
    - A supervised learning framework for automatically assessing students' CS competencies such as the ability to design and implement generalizable algorithms, and knowledge of loops and conditionals. In order to deal with the noise in the training dataset, this framework utilizes Gaussian process

regression to assess students' generated programming artifacts based on previously graded artifacts.

1.5 Organization

The remainder of this dissertation is organized as follows. Chapter 2 discusses background and related work on computer science education, evidence-centered assessment design, problem-solving strategy assessment, programming assessment, game-based learning environments, and stealth assessment. Chapter 3 discusses the stealth assessment data corpora along with the ENGAGE game-based learning environment. Chapter 4 describes the strategy-based temporal analytics framework. Chapter 5 discusses the design and development of a CS FKSAs competencies assessment and the preliminary results of its application on students' block-based programming artifacts. Chapter 6 presents the results of applying a revised version of the rubric presented in Chapter 5 on data collected from a new round of data collection. The semi-automated assessment framework and its evaluation is presented in Chapter 7. Finally, Chapter 8 concludes the dissertation with a summary, a brief revisit of our hypotheses, a brief discussion on limitations, and some proposed directions for future work.

CHAPTER 2:

BACKGROUND AND RELATED WORK

2.1 Computer Science Education

Computer Science (CS) and Computational Thinking (CT) have become a foundational skill for students to thrive in a digital economy (Hansen et al., 2017; *K-12 Computer Science Framework*, 2016).To prepare students for future studies and professions in Science, Technology, Engineering, and Math (STEM), it is essential to ensure that they acquire sufficient CS and CT competencies. There is no universally accepted definition of computational thinking and its practices among researchers and practitioners (Brennan & Resnick, 2012). In this work, we focus on a computational thinking framework proposed by Weintrop and colleagues (Weintrop et al., 2016) that identifies four main practices for computational thinking: data practices, modeling and simulation practices, computational problem-solving practices, and systems thinking practices.

A key strategy for developing CS and CT is through programming. However, learning how to program is challenging for novices, in part because of the intricacies of syntax in text-based programming languages (Grover & Basu, 2017; Grover et al., 2017). Block-based programming environments have emerged as a promising mechanism for introducing computational thinking to novices (Grover & Basu, 2017; Price, Dong, & Lipovac, 2017). Embedded in game-based learning environments, block-based programming environments can lower the "syntax" barrier for students including groups traditionally underrepresented in computer science (Brennan & Resnick, 2012). While block-based programming environments effectively facilitate programming for novices, significant scaffolding is still needed to support effective use of CS and CT practices and constructs,

such as abstraction, decomposition, and iterative testing and refinement of programming artifacts.

2.2 Computer Science Assessment

To provide students with the most helpful scaffolding and feedback, it is important to build effective assessments that can evaluate their CS and, in particular, programming competencies (Fields, Giang, & Kafai, 2014; Grover et al., 2017; Meerbaum-Salant, Armoni, & Ben-Ario, 2013). Conducting effective assessment lays the foundation for providing students with adaptive pedagogical strategies such as offering hints, feedback, and appropriate next problems. Consequently, efforts have also begun to develop effective instruments for assessing CS FKSAs (Grover & Basu, 2017; Grover et al., 2017; Seiter & Foreman, 2013).

2.2.1 Evidence-Centered Assessment Design

A particularly promising approach to assessment is evidence-centered design (ECD). ECD has been used successfully to assess students' skills and knowledge constructs in a broad range of game-based learning environments and simulations (Akram et al., 2018; Min et al., 2017; Mislevy, 2013; Shute, Ventura, & Kim, 2013; Shute, Wang, Greiff, Zhao, & Moore, 2016). The ECD process of assessment development relies on data drawn from students' actions during a learning task to make inferences about students' knowledge and skills (Mislevy & Haertel, 2007; Mislevy, Steinberg, & Almond, 2003). Following an ECD approach, we can identify explicit learning outcomes and measures to inform our assessment development (Grover & Basu, 2017). An important first step in ECD is domain modelling where the FKSAs are identified through the collaborative work of the field experts, teachers and assessment designers (Mislevy & Haertel, 2007). The identified FKSAs are then used in

a conceptual assessment framework to shape the specifications of an assessment of FKSAs. The conceptual assessment framework consists of: 1) the student model, which represents what students know or can do, 2) the evidence model, which contains behavioral evidence that drives the student model, and 3) the task model, which contains tasks that elicit the behaviors of interest that provide evidence (Grover & Basu, 2017). Student actions (i.e., behaviors) during the learning task are used as evidence in the conceptual assessment framework model for representing FKSA goals set in the assessment. An important phase of this work is to match evidence derived from student actions in the game to FKSA goals in the assessment.

In the following sections, we review some of the previous work in assessing CT proficiencies, in particular problem-solving strategies, and programming including block-based and text-based programming assessments. Additionally, in each section, we explain in detail how we utilized the evidence-centered assessment design framework for devising the stealth assessment framework within the ENGAGE game-based learning environment.

2.2.2 Problem-Solving Strategy Assessment

Previous work has explored approaches to detect students' problem-solving strategies using trace data. For example, one effort focused on building a probabilistic model that jointly represent students' knowledge and strategies (Käser, Hallinen, & Schwartz, 2017). This work represented domain knowledge as distinct rules and built an HMM model for each of the rules, in which the binary latent variable is whether the student knows the rule and the observation is whether the student has applied the rule correctly. Within this environment, students need to learn the rules by making hypotheses about the rules and testing their hypotheses. Based on a set of expert generated criteria, the quality of students' hypotheses

are categorized as low, medium, or high. To account for the quality of hypothesis in the final model, the researchers assigned a weight to each of the observations based on the utilized strategy for generating the hypothesis. An evaluation of the predictive accuracy of the final model with regard to within-game and post-game performance showed a higher accuracy when taking the quality of strategies into account compared to a model that considers domain knowledge exclusively.

Another attempt to model strategy focused on selecting features for classifying students' efficiency in solving challenges (Malkiewich, Baker, Shute, Kai, & Paquette, 2016). In this work, the problem-solving efficiency is determined through badges assigned to students upon completing a task based on the efficiency of their incorporated solution. Efficient solutions receive a gold badge, while non-optimal solutions receive a silver badge. To evaluate the effect of a set of pre-defined features in predicting the efficiency of students' problem-solving strategies, two J48 decision-tree based classification models were built to predict whether a student will receive a gold or silver badge. A set of pre-defined features are tested in both models to measure their effectiveness in predicting whether a student receives a gold badge (i.e. acquired an optimal strategy) or a silver badge (acquired a non-optimal strategy). Their results demonstrated that features representing students' general behavior that spans over multiple tasks are more predictive of students' performance compared to features representing students' task-specific behaviors.

The temporal analytics framework we introduce in this work uses problem-solving strategies that are automatically discovered through clustering based on *n*-grams of players' sequences of interactions with a game-based learning environment, thus eliminating the need for labeling or expert knowledge. We cluster students to categorize them based on their in-

game strategy utilization per task, and then use sequences of in-game strategy features over multiple tasks to predict their post-test performance. We use *n*-grams of students' sequences of interactions with the game environment to cluster them based on their in-game strategy. *n*-grams are commonly used in classifying text and temporal data (d'Aquin & Jay, 2013; Xing, Pei, & Keogh, 2010). For example, in one study, the *n*-gram approach is applied on students' sequences of actions available in a modeling software to generate features representing their modeling practices (Quigley, Ostwald, & Sumner, 2017). The generated *n*-grams were then used along with other features including variety and frequency of the performed actions to classify students based on the teacher who led the modeling activity.

2.2.3 Text-Based Programming Assessment

This section presents a review of the work done on automated assessment of text-based programming artifacts. Although block-based programming differs from text-based programming in syntax and visual representation, they can both be transformed into the same intermediate representation. Therefore, the techniques used for assessing one type of programming can be adapted to assess the other type of programming. There are two primary categories for assessing text-based programming artifacts: static assessment and dynamic assessment (Ala-Mutka, 2005; Ihantola, Ahoniemi, Karavirta, & Seppälä, 2010).

In dynamic assessment of programming artifacts, programs are executed against pre-defined test data to determine their correctness. Evaluation metrics include successful compilation, consideration of security threats, generation of expected outcome, and efficiency metrics such as CPU runtime and clock (Ihantola et al., 2010; Lajis et al., 2018). One of the oldest examples of a system that follows a dynamic assessment approach is Try (Reek, 1989). Try allows students to test their solutions by conducting a character-by-

character comparison between the results generated by students' submitted code and the expected results. Some other examples of dynamic program assessment systems are CourseMaker (Pratt, 2003), Online Judge (Cheang, Kurnia, Lim, & Oon, 2003) and BOSS (Joy, Griffiths, & Boyatt, 2005). Another category of dynamic assessment is designed to evaluate students' capability in testing and refining their own code. An example is Assyst (Jackson & Usher, 1997) that assesses students' test data suite by its coverage of all the program's paths. Another example of this category of dynamic assessment is used by Chen et al. (2004) that assesses a student's test data suite by running a series of buggy programs against it (Chen, 2004). Although dynamic assessment approaches are very helpful in uncovering execution errors, they fail to assess partial solutions or solutions with bugs.

Another important category of automated assessment is static assessment. Static assessments are capable of assessing programs that are not necessarily complete. To perform a static assessment, an intermediate representation of the program should be selected and generated from the source code. Examples of intermediate representations are textual representations, abstract syntax trees, control flow graphs, and program dependence graphs. After forming the intermediate representation, the representation is analyzed for its correctness, efficiency, and quality. In static assessments, correctness is usually assessed through character analysis, string analysis, syntax analysis, and semantic analysis. Quality is assessed by software metrics such as number of lines of code, number of variable statements and expressions, and also through plagiarism detection.

Work by Wang and colleagues (Wang, Su, Wang, & Ma, 2007), follows a semantic similarity-based approach to assess the correctness of a C programming artifact by comparing it against a correct program model. In this work, they first reduce the state space

of program artifacts by conducting a set of program standardizations including expression, control structure, and function invocation standardization. They also remove redundant statements, rename variables and reorder statements to make programs avoid unnecessary variations in students' codes. Then, they calculate a similarity factor based on size, structure and statement similarity subfactors weighted by grading criteria. The equation for calculating the similarity score of an assignment in comparison to a correct program model is demonstrated in Equation 2.1.

$$SemanticSim = \lambda_{size} \times SizeSim + \lambda_{stru} \times StructureSim + \lambda_{stat} \times StatementSim \quad (2.1)$$

Two other common applications of automated assessment of programming artifacts are software evaluation and plagiarism detection. In software evaluation, software verification tools automatically check functional correctness properties of the program, including the absence of bugs that could make memory violations or raise runtime errors. Examples of common software evaluation approaches are symbolic execution (King, 1976), model checking (Clarke, 2008), and abstract interpretation (Cousot & Cousot, 1977). Plagiarism detection tools use a variety of techniques to measure the similarity of different programming artifacts to warn about possible plagiarism. Some of the common techniques used by plagiarism tools are attribute-based comparison, token-based comparing, and structure-based comparison (Martins, Fonte, Henriques, & da Cruz, 2014). An important example of a plagiarism tool is Moss (Hage, Rademaker, & Van Vugt, 2010), which uses a token-based comparison to identify and present the pieces of codes that are similar.

Most of the work on static assessment of programming artifacts utilizes a variety of similarity measurements to calculate the relative correctness of a programming artifact in reference to other graded programming artifacts. This approach will mostly result in an

overall grade that is unable to specify any details on the specific concepts, skills, and knowledge that students are missing. In an educational context, this information is essential for providing students with the appropriate scaffolding and support they need. In order to provide students and their instructors with detailed and interpretable automated assessments, we follow a hypothesis-driven learning analytic approach to assess students' programming artifacts (Grover et al., 2017). In the following section, we describe this framework in detail. Furthermore, we describe how our devised assessment fits in this framework.

2.2.4 Block-Based Programming Assessment

Block-based programming languages have become an increasingly central part of new K-12 CS/CT curriculum. As a result, it has become important to design effective approaches to assessing CT in block-based programming environments. In early work, Koh presented two metrics for measuring computational thinking using program behavior similarities and computational thinking pattern graphs in order to compare two programs with respect to their computational pattern similarity (Koh, Basawapatna, Bennett, & Repenning, 2010).

Assessing computational thinking has also been widely investigated for informal learning environments. For example, Xie and Hal explored student skill progression and its relationship with CT learning in MIT App Inventor by investigating the existence and frequency of different blocks (Xie & Hal, 2016). Seiter and Foreman proposed a framework for assessing computational thinking in primary grades that maps measurable evidence from students' programming code to computational thinking concepts (Seiter & Foreman, 2013). In this approach, they first extracted evidence variables that are directly inferable from students' programming artifacts such as utilization of certain blocks that are necessary for the task at hand. Design patterns are then extracted from the evidence variables to provide

evidence of students' understanding of certain functions necessary to the context of the programming assignment. Finally, a qualitative analysis of the design patterns leads to an assessment of students' proficiency in certain computational thinking concepts. In a related approach, Grover et al. presented an ECD-based assessment method to measure novice learners' misconceptions about loops, variables, and Boolean logic (Grover & Basu, 2017). While these approaches provide great insight on students' understanding of CS and CT constructs, they require extensive manual analysis. As a result, these approaches are not scalable.

In two parallel efforts, Diana et al. have followed a string analysis approach to assess students' block-based programming artifacts (Diana et al., 2017a, 2017b). They used a token-based approach to generate code-chunks from the programming artifacts and generated a feature set that contains the count number of each of the code-chunks. They used a ridge-regression model to infer the grades for programming artifacts based on previously graded artifacts. In a follow up work, they used seed-based selection and L1-regularization through lasso regression to perform feature selection. This work does not consider the noise in the training dataset when conducting the prediction. Although they propose an instructor dashboard in their work, their assessment approach can only generate overall reports of the classroom and general grades for each student.

In a broader interdisciplinary effort, Grover et al. proposed a hypothesis-driven framework for the design and assessment of computational thinking and programming tasks (Grover et al., 2017). The hypothesis-driven framework consists of three initial phases: the domain modeling of CS and CT skills and practices in programming, the task modeling, and the programming task piloting. The domain modeling phase in this framework is analogous

to the domain modeling phase in the evidence-centered assessment design approach in which the FKSAs are identified in collaboration with field experts, instructors, and assessment designers. In the task modeling phase, identified FKSAs guide the design of programming tasks that can provide relevant evidence of students' competencies, as well as the design of corresponding rubrics that can relate the prior evidence to identified FKSAs. Finally, in the programming task piloting phase, the designed tasks are piloted and multiple datasets are collected in the form of programming artifacts, process data from observations and videos, and log data. The collected data is then analyzed through a variety of quantitative and qualitative approaches and the combined results are utilized for the refinement of the identified FKSAs and programming tasks.

To analyze our second and third hypotheses, we follow a slightly revised version of this framework where we first implement the domain modeling phase in correspondence with one of the existing programming tasks in the ENGAGE game-based learning environment to identify its relevant FKSAs. We then design a rubric in accordance with the identified FKSAs to assess students' programming artifacts. Subsequently, we use the collected data from our pilot studies to refine the identified FKSAs and the corresponding rubric. We then apply the refined rubric to assess students' CS FKSAs. Figure 2.1 illustrates the framework adapted from the blended learning analytics framework proposed by Grover at al. (2017).

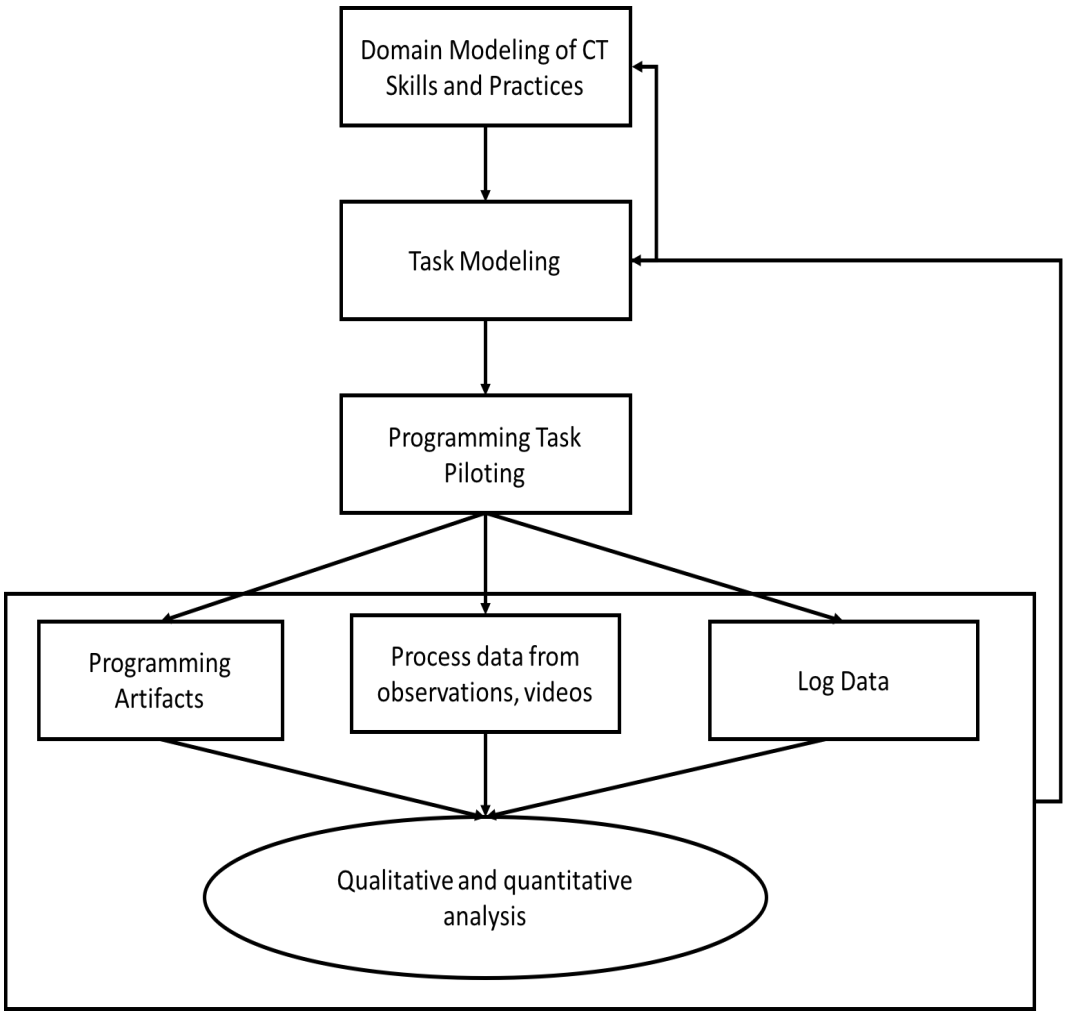

Figure 2.1. Adapted hypotheses-driven learning analytic framework. The rubric design and learning analytics process for assessing students' block-based programming artifact. Adapted from (Grover et al. 2017).

To automatically assess students' programming artifacts, we use the devised rubric to manually grade students programming artifact submissions. We then transform the block-based programming artifacts to their corresponding abstract syntax trees. Subsequently, we extract structural *n*-grams to preserve ordinal and hierarchical information that can represent the semantic relationships in a programming artifact. We then use a supervised learning approach to automatically assess newly-submitted programming artifacts based on a corpus of graded artifacts. A similar approach to feature extraction is explored in a work by Zhi et

al. (2018). In their work, they utilize *pq*-grams (*p*-1 immediate ancestors of each node and *q* of its children in an abstract syntax tree) to identify important features within students' programming artifacts (Zhi, Price, Lytle, Dong, & Barnes, 2018). They use the previously-identified features to reduce the state space of possible solutions. The reduced state space is then used to provide students with adaptive feedback based on the closest successful solution.

2.3 Game-Based Learning Environments

In order to foster engagement in learning, game-based learning environments leverage game design elements such as engaging narratives, badges, and immersive story worlds (Clark et al., 2016). Intelligent game-based learning environments integrate game-based learning and intelligent tutoring system functionalities into one system (Jackson & McNamara, 2013; Lester et al., 2013). These learning environments have been explored for a broad range of subjects including science (Nelson, Kim, Foshee, & Slack, 2014), mathematics (Kebritchi, Hirumi, & Bai, 2010), computer science (Buffum et al., 2016; Min et al., 2015) and public policy (Sabourin & Lester, 2014). Furthermore, these environments can embed stealth assessment, which has emerged as a promising approach to unobtrusively assessing learners while interacting with the intelligent game-based learning environments (Quellmalz, Timms, Silberglitt, & Buckley, 2012; Sahebi, Huang, & Brusilovsky, 2014; Shute & Ventura, 2013).

We apply our assessment framework on data collected from students' interactions with a game-based learning environment called ENGAGE that is designed to develop middle-grade students' computational thinking competencies and block-based programming skills.

2.4 Stealth Assessment

Stealth assessment can assess learners' competencies solely based on data obtained from their interactions with the game-based learning environment. This approach enables the ITS to assess learners' competencies without interfering with their natural flow of learning (Shute, 2011). Stealth assessment uses evidence derived from students' game-based learning activities to infer students' competencies with respect to knowledge, skills, and performance, often utilizing evidence-centered design (ECD) (Mislevy et al., 2003).

In game-based learning environments, stealth assessment can build a dynamic competency model of the student by monitoring his granular behaviors across multiple tasks to generate evidence for the model. Following this dynamic, stealth assessment has been used to unobtrusively assess a wide range of constructs (Wouters et al., 2013). Since stealth assessment can continuously assess students' competencies during gameplay, it can provide formative feedback to students and teachers to inform instruction and enhance learning (Chen, Gully, & Eden, 2001; Kerr & Chung, 2012; Shute & Ventura, 2013).

In one attempt to conduct stealth assessment, a directed graphical model was built based on relevant competencies, and related variables were extracted from the observed data to be used as evidence for the targeted competencies (Kim, Almond, & Shute, 2016). Consequently, a Bayesian network-based model was built by specifying the conditional probabilities between parent nodes and corresponding child nodes in the directed graph model to assess students' competencies. Bayesian knowledge tracing is also incorporated in Shute's work (Shute et al., 2016). Where a problem-solving competency model is generated based on literature, the evidence model is generated from students' interactions with the game, and the stealth assessment is conducted using a Bayesian network.

In another approach, Falakmasir and colleagues investigated two hidden Markov models (HMMs) trained for high-performing and low-performing students (Falakmasir, González-Brenes, Gordon, & DiCerbo, 2016). Subsequently, for observed sequences of events, log-likelihoods of belonging to each HMM were calculated. Finally, the difference between the two log-likelihoods was used in a linear regression model to predict post-test scores. This approach reduces the need for labor-intensive domain knowledge engineering.

An alternative approach to stealth assessment is to use artificial neural networks. This approach is offered by DeepStealth, a stealth assessment approach that uses deep neural networks (Min et al., 2015). DeepStealth used a deep feedforward neural network (FFNN) to learn multi-level, hierarchical representations of the input data for evidence modeling. This work leverages a variety of features including domain-expert engineered input features and external learning features. Their results showed the effectiveness of their proposed FFNN in predicting students' post-test performance. In a subsequent work, they addressed the structural limitations in the FFNNs using a long short-term memory network-based stealth assessment framework (Min et al., 2017).This work utilizes an end-to-end approach for assessing students' competencies using raw data obtained from students' interactions with an educational game environment.

Our first study on stealth assessment is built on this work. While the long short-term memory network-based stealth assessment framework proposed in Min et al. (2017) focused primarily on computational methods to model evidence within ECD, our approach extracts temporal evidence from students' dynamic in-game strategy use throughout their problem solving. In addition, we devise a stealth assessment framework to assess students' CS FKSAs within the ENGAGE game-based learning environment. For this part of our study, we follow

the blended learning analytics framework proposed by Grover et al. (2017) to automatically assess students' FKSAs as demonstrated within their submitted block-based programming artifacts.

CHAPTER 3:

STEALTH ASSESSMENT DATA CORPORA

The data used for this dissertation is collected from students' interactions with a game-based learning environment named ENGAGE that is designed to develop middle-grade students' computational thinking competencies and block-based programming skills. In this chapter, we describe the ENGAGE learning environment, its in-game problem-solving challenges that we are using as testbeds for our stealth assessment framework, and the stealth assessment data corpora.

3.1 ENGAGE Game-Based Learning Environment

ENGAGE is a game-based learning environment designed to introduce computational thinking to middle school students (ages 11-13). A sample binary to decimal challenge from the ENGAGE game is presented in Figure 3.1. The game was developed with the Unity multi-platform game engine and features a rich, immersive 3D story world for learning computing concepts (Buffum et al., 2016). The game-based learning environment present students with challenges that require problem solving and programming to promote computational thinking skills including abstraction and algorithmic thinking. The computational challenges within the game are designed to prepare middle school students for computer science work in high school, and to promote positive attitudes toward computer science.

A diverse set of over 300 middle school students participated in focus group activities, pilot tests, and classroom studies with the game. Of the students who provided demographic information, 47% were female, 24% were African American or Black, 16% were Hispanic or Latino/a, 17% were Asian, 38% were White, and 5% of the students were

Figure 3.1. ENGAGE game-based learning environment. A sample binary to decimal challenge from the ENGAGE game.

Multiracial. The research team worked closely with a similarly diverse group of teachers throughout the project. A subset of teachers helped to co-design the game-based learning activities providing iterative feedback throughout development. Each of the teachers implementing the game in their classrooms attended either one or two summer professional development workshops that introduced computational thinking concepts and the ENGAGE game-based learning environment.

In the game, students play the role of a specialist who is sent to investigate an underwater research facility that has lost communications with the outside world. As students progress through the game, they discover that Murdock, a rogue scientist, has taken control of the computing devices within the facility. Students navigate through a series of interconnected rooms and solve a set of computational challenges while interacting with a programming interface that controls devices.

To solve each challenge, students need to operate devices either by programming them or interacting with them in reference to their pre-written programs. Students use a visual block-based programming language to program the devices (Min et al., 2017). Students are accompanied throughout the game by some non-player characters who help them progress through the narrative, and provide them with clues, and positive reinforcement (Min et al., 2015).

To complete the game, students need to finish three major levels: the Introductory Level, in which students learn the basics of the game and simple programming; the Digital World Level, in which students learn how digital data is represented with binary sequences; and the Big Data Level, in which students have the opportunity to work with various datasets and retrieve hidden information by cycling through data and filtering it based on different variables.

3.2 Corpus A - Binary Challenge

This section presents information about the study that is done to analyze our first hypothesis. Following, the backstory and game play for the binary challenges, binary related challenges within the ENGAGE game that are used as the testbed for this study, is described.

3.2.1 Backstory and Game Play

The work presented here, focuses on students' problem-solving activities within the Digital World level. The first set of tasks in this level consists of binary locks that are programmed to open if the binary representation of a specific base-ten number is generated by the students (Figure 3.2). Each lock provides students with five consecutive flip tiles representing bits of a five-digit binary number. Players can swap each bit between 0 and 1 by flipping the

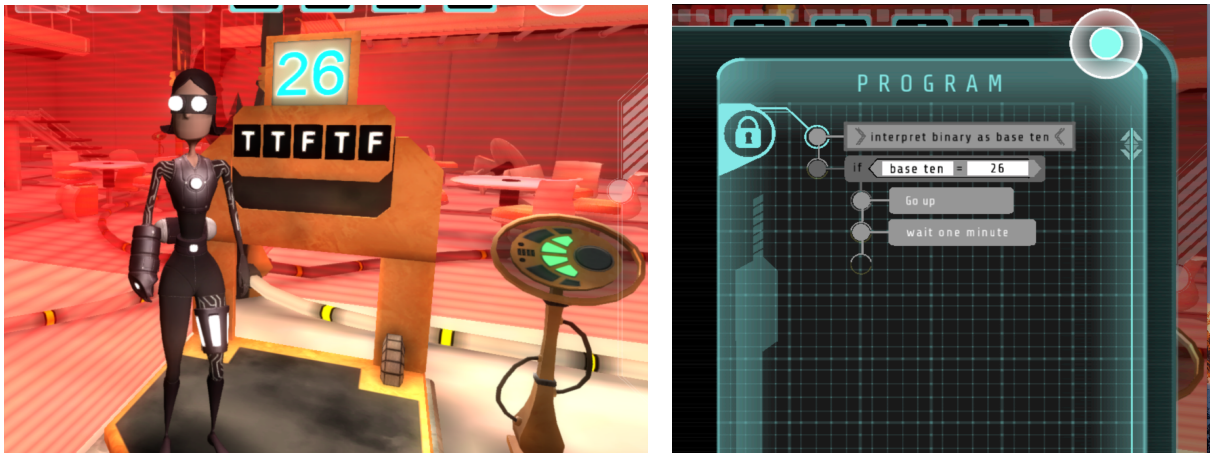

Figure 3.2. Binary challenge. (Left) a binary lock device that students must unlock. The T (true) tiles indicate the bits are 1, whereas F (false) tiles denote 0. The current binary number is 11010 and the corresponding base-ten number, 26, is displayed on the device as immediate feedback. (Right) the visual programming interface displaying the binary lock's program.

corresponding tile on the tile panel (Figure 3.2, left). The decimal representation of their generated binary number will be presented on a small screen above the panel. Students can find the target number by reviewing an existing program (Figure 3.2, right) associated with the binary lock device. For example, to advance to the next task, students must flip binary tiles on the binary lock device to generate the target decimal number (Figure 3.2, left), execute its program, and unlock the binary device. Similarly, the second set of tasks in the Digital World level features lift devices that are activated when students generate the target base-ten value by flipping binary tiles and execute the program associated with the lift.

Through these tasks, students learn about the concept of bits in binary numbers and the weight assigned to each bit. In the analyses reported here, we used behavior trace data from students' interactions with 11 binary tasks from the Digital World level, where students learn the weights associated with each of the five bits through the first five tasks and then learn how to combine multiple bits to make more complex numbers with binary

representations. To teach variations of binary representations, the game enables students to flip tiles between '0' and '1', 'black' and 'white', and 'F' (False) and 'T' (True).

3.2.2 Data Collection

We analyzed 244 students' behavior trace data obtained from a teacher-led study in four public middle school classrooms in the United States. On average, students play throughout the game over the course of two weeks. To support collaborative learning, we collected student behavior trace interaction data from pairs of students in which they took turns serving as navigator (traversing the game) and driver (action planning). Collaboration has shown to positively effects students' learning in computer science education (Buffum et al., 2016). Pre- and post-test assessments measuring computer science attitudes (Wiebe, Williams, Yang, & Miller, 2003), self-efficacy (Chen et al., 2001) and content knowledge (e.g., binary representation) were completed individually by students before starting the Digital World level (pre-test) and immediately after finishing it (post-test). Both pre-test and post-test are on a scale of 0 to 1.00. Out of 244 students, 168 students completed the pre-test and post-test for content knowledge as well as all 11 binary representation tasks for this level. The results of conducting a paired t-test on students' content knowledge pre-test (M=0.44, SD=0.20) and post-test (M=0.59, SD=0.24) revealed a significant improvement from pre-test to post-test scores (t (167) =11.24, p<0.001).

3.3 Corpus B - Bubble Sort Algorithm

We first present the study that was conducted to investigate the second and third hypotheses. Next, we present the backstory and gameplay for the bubble-sort challenge in the ENGAGE game that is used as the testbed for this part of the study, is described. We then present the details of the study and the data collection process are described.

3.3.1 Backstory and Game Play

In this study, we focus on students' problem-solving approaches within a specific level of the game where students write a bubble sort algorithm to order a set of radioactive containers (Figure 3.3). This room has two devices: a containment device which holds six randomly positioned containers and a lock device which opens only when the containers are sorted in the increasing order. The player can escape the room through a door by correctly implementing bubble sort and executing the lock program when the containers are sorted. The lock has a pre-written program that will check the positions of containers and opens if they are in the correct positions (Figure 3.4, left). The containment device provides students with the necessary blocks for implementing a bubble sort algorithm using a small robotic arm inside the device's protective housing (Figure 3.4, right). Students need to test the correctness of their program in two steps. First, they need to run the bubble sort device to sort the containers. Second, they need to run the open lock program which checks if the containers are sorted and opens the door accordingly.

3.3.2 Data Collection - Round 1

The dataset used in this study is the same dataset as the previous study. For this part, we analyzed data obtained from students' interactions with the bubble-sort challenge of the ENGAGE game. As students interact with the game-based learning environment, all of their interactions with the game are captured, such as dragging programming blocks to write a program and programs being executed. For this study, we used trace data from 22 students' (10 males, 12 females; mean age 12.6 years) interactions with the bubble-sort challenge in the game. The data used for this study is the behavior trace interaction data from pairs of students interacting with the game. In addition, we collected pre- and post-test assessments measuring

Figure 3.3. Bubble sort challenge. The bubble sort challenge in the game-based learning environment.

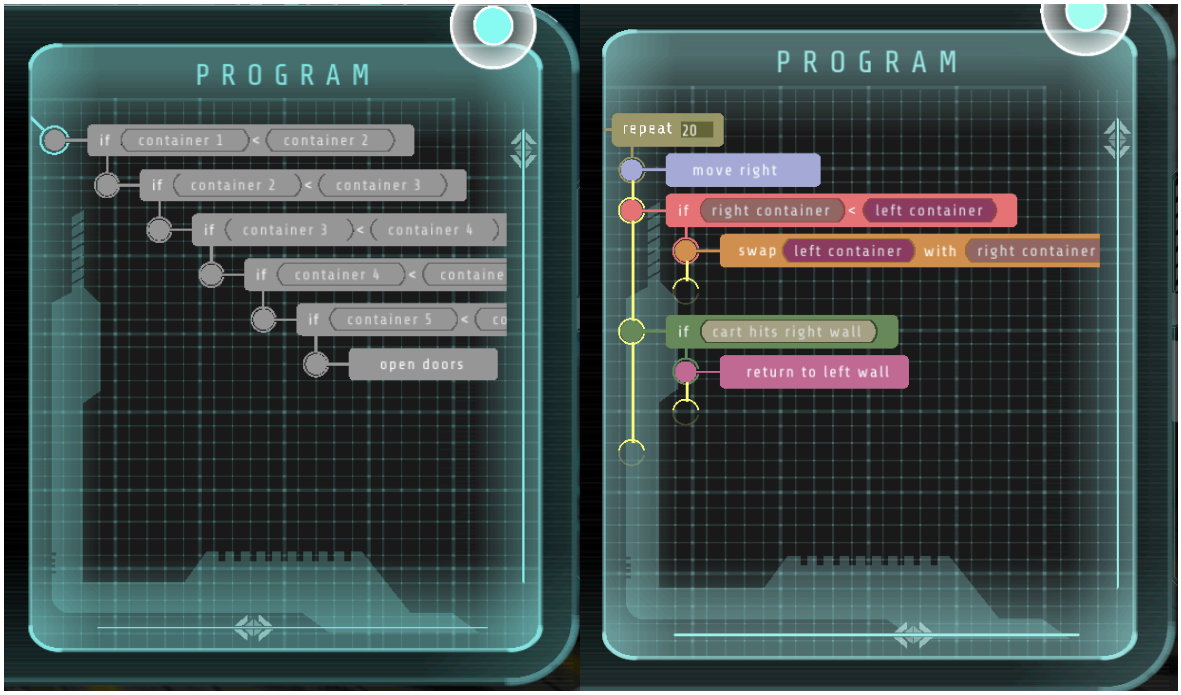

Figure 3.4. Bubble sort challenge door and containers programs. (Left) the door lock program that opens the door if the containers are sorted. (Right) an example of a bubble sort algorithm implemented through block-based programming environment.

content knowledge which were completed individually by the students before and after the game. Both the pre-test and post-test cumulative scores are reported on a scale from 0 to 1.

3.3.3 Data Collection - Round 2

In the second round of the study, we captured similar game-play data from students' individual game-play. In this round of study, students' played with the Engage game individually with a new pre- post-test assessment that measure students general computational thinking competencies (Wiebe et al., 2019) being administered. The pre- and post-test assessments was completed individually by students before and after the game. We collected data from five classrooms across three schools in urban areas in United States. Sixty-nine students (29 males, 13 females, 27 unknowns, mean age ~10.7 years) consented for their data to be used in our research. We collected 1570 submission including 642 unique programming artifacts. Each programming artifact is a snapshot of students' submitted solution for the bubble sort problem.

CHAPTER 4:

IMPROVING STEALTH ASSESSMENT IN GAME-BASED LEARNING ENVIRONMENTS WITH LSTM-BASED ANALYTICS

4.1 Modeling Students' Problem-Solving Strategies

Students exhibited a broad spectrum of problem-solving strategies while solving the binary challenges in the Digital World level. For example, some students pursued random trial-and-error strategies to find solutions, while at the other end of the spectrum, some students pursued thoughtful systematic approaches to solve the challenge. As would be expected, some students fell in the middle of this spectrum by utilizing more thoughtful trial-and-errors. For each of the 11 consecutive binary challenges, students' tile-flip sequences are used to cluster them into distinct groups. Subsequently, these clusters are interpreted in terms of the problem-solving strategies exhibited by members of each cluster. Below, we first describe the process of clustering students' in-game strategies based on their binary tile flip sequences and then describe the most common problem-solving strategy in each cluster.

4.1.1 Methodology

In order to group students based on their problem-solving strategies, features are derived from students' binary tile flip sequences. The flip sequences are encoded as *n*-grams, commonly used as a representation for sequential data such as text and speech (Xing et al., 2010), as well as for sequential trace data (Falakmasir et al., 2016). Sequences of *n* adjacent elements are extracted from the original string. Each unique *n*-gram is a feature in our feature vector. The value for each *n*-gram for each flip string is the frequency of the *n*-gram's occurrence in the flip string.

Since each task differed slightly from the other tasks, students problem-solving behavior were analyzed separately for each task. For each of the 11 binary challenges, students' behaviors (i.e., the flip sequence generated for that specific task) are clustered based on the extracted *n*-gram features. This resulted in 11 sequential cluster-memberships per student. In the following sections, we describe how we identified different problem-solving strategies using the proposed clustering method.

4.1.2 Feature Engineering

Students' problem-solving behavior were analyzed for 11 consecutive binary tasks. Here, each task refers to generating the binary representation of a decimal number determined by a program that operates the device associated to this task (e.g., 16 in Figure 3.2). Each device features a panel with five tiles which is set to 00000 by default representing the binary number. Considering tiles' indices starting at one from the right most tile, if a student has flipped tile number four (i.e., 01000 with the decimal representation of 8), followed by flipping tile number five (i.e., 11000 with the decimal representation of 24), their tile flip string would become {4, 5}. Students' interactions with binary flips are extracted in the format of a string containing students' consecutive flips of the binary tiles for each task.

In order to capture the most fine-grained information present in the series of flips, *n*-grams with varying lengths of *n* are used. Preliminary explorations showed including sequences of lengths larger than four exponentially increases the sparsity of the dataset. To eliminate the sparsity issue, the *n*-gram size is limited to be 4. The final feature set includes sequences of length one (i.e., unigram features) to sequences of length four (i.e., 4-grams) that are repeated at least three times throughout our dataset. The natural language processing toolkit (NLTK) library for Python is used to extract *n*-grams and their associated frequency

from each flip string. For example, a total of 2,495 unique *n*-grams with at least three occurrences were extracted from the student flip strings generated for one of the tasks. These *n*-gram feature vectors are then used to cluster students' in-game strategy use per task, and an *n*-gram feature vector is generated separately for each of the 11 tasks. Each student's per-task behavior is represented with an *n*-gram feature set, in which the value for each feature is the number of occurrences of the specific *n*-gram in the student's flip string.

Flip strings provide a fine-grained representation of students' problem-solving behaviors in solving binary representation challenges and their adopted strategies. As an example, consecutive flips of the same tile (i.e., double flips) by a student can be an indicator of the student's intention to learn the weight assigned to that binary digit. Further, the overall number of flips conducted to generate the target base-ten value can be used to gauge the students' overall efficiency in solving the problem.

4.1.3 Clustering

Next, expectation-maximization (EM) clustering (Dempster, Laird, & Rubin, 1977) is applied on students' flip behaviors represented by *n*-gram feature vectors to identify students' problem-solving strategies. Because each of the 11 tasks in the Digital World level targets a different base-ten number, the binary code needed to solve the task is different. Consequently, flip sequences obtained from students' interactions with a binary device reveal information specific to the target value designed for the task. Thus, clustering is performed separately for each of the 11 tasks. The MClust package in R (Scrucca, Murphy, Raftery, & Fop, 2016) is used to cluster the feature vectors. EM clustering can explore a range of cluster numbers and return the (local) optimal number of clusters based on the maximum likelihood estimation. A different optimal number of clusters was identified for each task. Three, four,

and nine clusters emerged most frequently when we explored the number of clusters between two to ten. A preliminary investigation on these different number of clusters found that three clusters showed coherent patterns for problem-solving strategies across all tasks, and we thus chose three as the number of clusters for all tasks in this work.

4.1.4 Interpreting Clusters

To interpret students' problem-solving strategies using the clusters as identified above, two novel methods that consider the error between the target value and the student-generated value are presented. To analyze students' problem-solving patterns for each cluster of each task, the average error at each flip relative to the solution target for students who belong to the same cluster is calculated. As the default value on each device binary panel is 00000, all of the students start with the same non-zero initial error. For each task, only students who completed the task successfully are included. Hence, the average error for each cluster decays toward zero. We expect to observe distinct error-decay patterns across the three clusters. In addition to decimal error, an error measurement based on the hamming distance between the binary representation of the target value and the students' generated binary number: the binary error is introduced. Both decimal and binary errors are described in details below.

Decimal error. The decimal error is the absolute difference between the target base-ten value and the base-ten representation of the student-generated binary string. As the default decimal value on each device is zero, the initial error for all students is the value of the target decimal number. Furthermore, all students end with an error equal to zero. The decimal error is calculated after every new flip. As a result, a sequence of decimal errors is generated for each student when completing each of the 11 tasks. For each task, the average decimal error

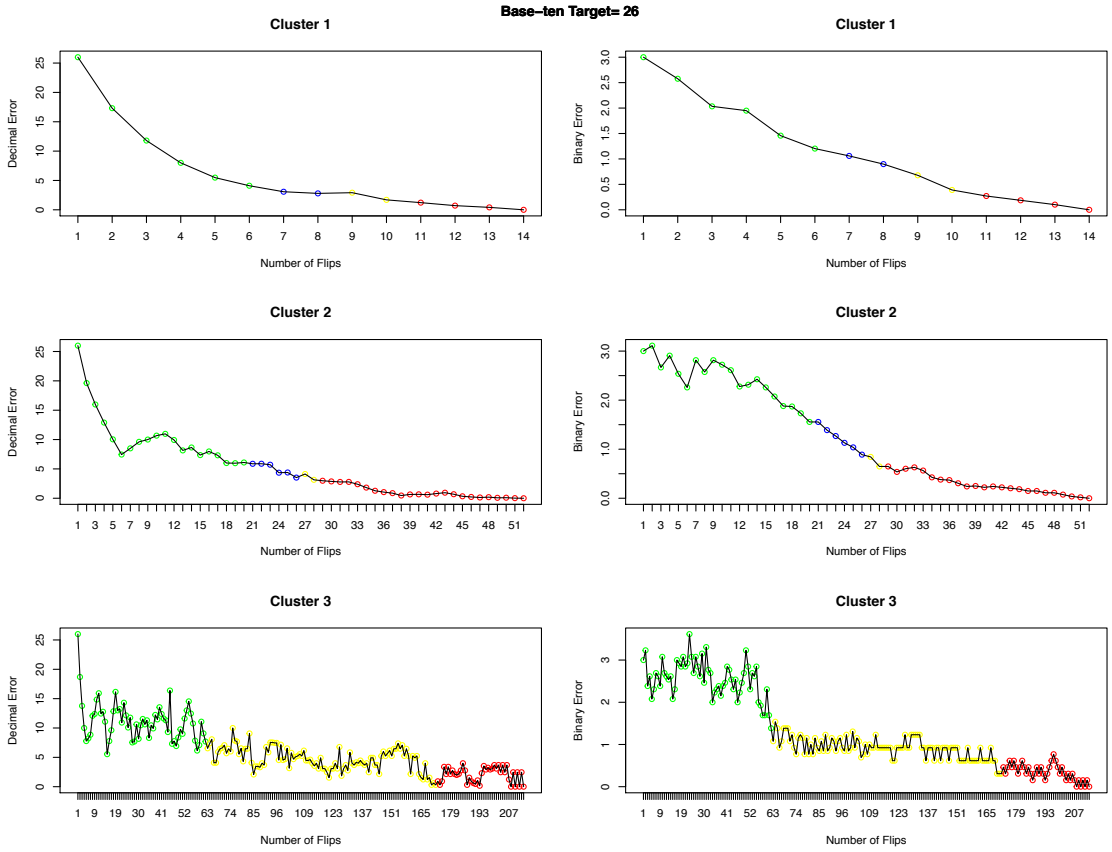

Figure 4.1. Students' average decimal and binary error at each flip when generating binary representation of number 26. (Left) students' average decimal error at each flip. (Right) students' average decimal error at each flip.

over time (number of flips) is then plotted where the x-axis shows the flip number and the y-axis shows average error in that flip for students in that cluster. For example, in flip number one, the y-axis shows the average decimal error of everyone in that cluster after the first flip. Because the total number of flips is different for each student in a cluster, the decimal error value of zero is used for students who already completed the task and the average decimal error is calculated over all students in the cluster. For example, suppose there are two students in a cluster, where student A's decimal error sequence is {2, 1, 2, 0} and student B's error sequence is {2, 0}. The task is to generate a base-ten number with the value two using binary flips. We take the maximum length of the two sequences, four, obtained from student

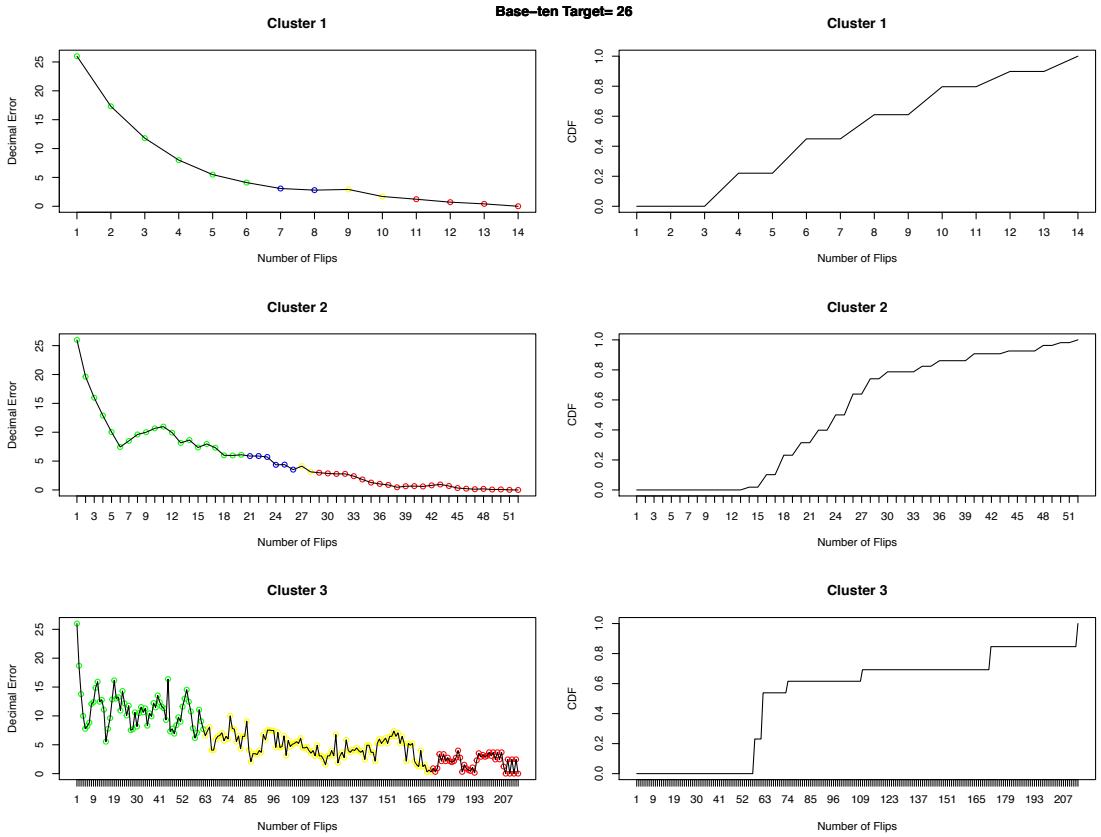

Figure 4.2. Students' average decimal error when generating binary representation of number 26 and the CDF of the present population at each flip. (Left) students' average decimal error at each flip. (Right) the CDF of the present population at each flip.

A, and reformulate student B's sequence to {2, 0, 0, 0}. As a result, the average decimal error sequence becomes {2, 0.5, 1, 0}.

The average error at each flip for a particular binary challenge in the game is shown in Figure 4.1. In this task, students are asked to generate the binary representation for the base-ten target number 26. After clustering students' *n*-grams for this task, 118 students were grouped in the first cluster, 108 students were grouped in the second cluster, and 19 students were grouped in the third cluster. Because the target value for this task is 26, the average error for students is 26 in the beginning and zero at the end for all three clusters, while decay patterns differ across clusters (Figure 4.1). Students in each cluster solved the problem within

a varying number of flips. The error for students who finished earlier is represented with zero. The percentage of students still working on the challenge at each flip is shown using a color-coding scheme. In Figure 4.1, green points mark flips where between 70% to 100% of the population is present, blue points indicate the presence of 50% to 70% of the population, yellow points mark 30% to 50% of the population, and red points indicate flips were less than 20% of the population of that cluster are still working on the problem. These percentages are derived from the cumulative density functions (CDFs) of clusters' present population at each flip that are plotted in Figure 4.2 (right). The observed patterns of error decay in different clusters were consistent across all 11 tasks. This suggests that students follow certain problem-solving strategies when solving the binary tasks within the game.

Binary error. Binary error incorporates the Hamming distance to measure the distance between the current state of student-generated binary strings and the binary representation of the target base-ten value. Hamming distance is the number of different elements in two strings of the same size. The approach for plotting the binary error is similar to the approach for plotting the decimal error. Figure 4.1 shows binary (right) and decimal (left) error plots for each cluster of the same binary challenge (base-ten target number 26). Same CDFs as in (Figure 4.2, right) hold for binary error plots, as the binary error plots are generated from the same population used for generating decimal error plots.

4.1.5 Resulting Strategies

As noted above, the same general patterns of error decay for each cluster apply to all 11 challenges analyzed in this study. Furthermore, for all 11 tasks, there is a consistency in the error decay pattern between the decimal and the binary error represented in each of the three clusters. While the decimal error captures students' strategies to reduce the base-ten error

between the target number and their generated number, the binary error places more emphasis on the representational difference between binary sequences. The binary error focuses on students' understanding of each bit and its associated weight. The analyses reveal a clear distinction in students' problem-solving strategies in solving the in-game challenges. After clustering, following distinct strategy groups emerge for all analyzed tasks:

- **Thoughtful systematic approaches**: Students who completed the task with fewer trial-and-error attempts. This can be an indicator of students' systematic thinking while solving the problem.
- **Thoughtful trial-and-error**: Students who had a moderate number of flips and their average error decayed continuously toward zero with some sudden spikes that suggest trial-and-error attempts. This pattern could be an indicator of learning through trial-and-errors.
- **Random trial-and-error**: Students who completed the activity with many more flips compared to students in the other clusters, with high spikes in the error pattern and without a continuous decay toward zero. This error pattern suggests fully random trial-and-error attempts which justifies the high number of flips needed to complete the task.

The binary and the decimal error decay patterns confirmed each other for every cluster of every task. The analyses reveal that this two-error metrics efficiently capture *n*-gram-encoded student behaviors. Furthermore, the analysis demonstrate that students' per-task behaviors naturally fall into one of the three groups. These identified clusters that are represented as game strategy features constitute our evidence model for stealth assessment.

4.2 Stealth Assessment

Students' choice of strategy has shown to be important in their learning process. Hence, modeling their choice of strategy can contribute to the prediction of their learning outcomes (Eagle & Barnes, 2014; Rowe et al., 2014). As a result, we aim to evaluate the predictive power of models of students' in-game problem-solving strategies over time to predict their post-test performance. We seek to determine if the in-game strategies derived from students' interactions with the game environment can be used as evidence for stealth assessment.

Our proposed feature set contains inherent rich temporal dependencies among students' behaviors over the course of interactions with the ENGAGE game-based learning environment. We expect their gradual learning over consecutive tasks to affect their choice of strategy over time. To effectively model temporal dependencies in the feature set an LSTM-based neural network (Hochreiter & Schmidhuber, 1997) is investigated. After training the LSTM-based neural network based on students' in-game strategy use over time, the trained network is used to classify their post-test performance. In addition, two well-established classification techniques, random forest (RF) (Breiman, 2001) and support vector machines (SVM) (Cortes & Vapnik, 1995), are examined to predict students' post-test performance. It is important to note that, in contrast to the LSTM approach, both random forest and the support vector machine treat the in-game strategy features as independent features in their predictions. Hence, they are suitable for evaluating the importance of taking the temporal dependency among features into account when predicting post-test performance.

To isolate the efficacy of the strategy-based feature set, two LSTM-based evidence models are used. The first LSTM-based evidence model is trained on a feature set that contains pre-test performance only, and the second LSTM-based model is trained with a

feature set combining both pre-test performance and in-game strategy. As the in-game strategy features are derived from a dataset that is the result of collaboration between two team-members, the pre-test performance is included in both feature sets as an individualized measurement.

4.2.1 Data Preparation

For the classification task, data from 168 students who finished both pre- and post-tests and also completed all 11 binary challenge tasks within the game us used. The initial dataset included students' pre- and post-test scores along with their flip sequences. The flip sequences are then transformed into *n*-gram features for each of the 11 tasks. Students' post-test performance is then predicted using students' in-game strategy features derived from their *n*-gram features using a clustering approach. In order to keep the train and test set completely separate, before clustering the dataset, it is divided into training and held-out test sets. Clustering is then performed using students' *n*-gram data in the training set. After identifying distinct clusters for each challenge in the training set, the Gaussian finite mixture models estimated by the MClust package is used to cluster students' data in the test set. After this process, students' data in both the training and test sets include sequences of in-game strategies across the 11 binary-representation tasks in addition to their pre-test performance, a categorical representation of the pre-test score. These input features are utilized to predict post-test performance, a categorical representation of the post-test score.

The initial pre- and post-test scores are continuous variables, ranging between 0 to 1.00. We categorize pre- and post-test scores into three levels of performance with a tertile split obtained from their distributions. For the pre-test, scores between ($0 \leq \text{score} \leq 0.36$) are categorized as low, scores between ($0.36 < \text{score} \leq 0.54$) as medium, and scores between

(0.54 < score ≤ 1.00) are categorized as high. Similarly, for the post-test, scores between (0 ≤ score ≤ 0.45) are categorized as low, scores between (0.45 < score ≤ 0.72) as medium, and scores between (0.72 < score ≤ 1.00) are categorized as high. Table 4.1 presents the distribution of students (*n*=168) with respect to their pre- and post-test performance.

Table 4.1. Pre- post-test performance distribution of students. Distribution of students (n=168) in relation to their pre- and post-test performance.

Test	Low	Medium	High
Pre-test	74	53	41
Post-test	60	59	49

In order to prepare the feature set for classification task, one-hot encoding is applied on the categorical variables (i.e., pre-test performance and the 11 in-game strategy changes), to transform them into a trainable representation. One-hot encoding represent categorical variables with a feature vector whose length is the size of the possible values of the categorical variable. In the one-hot encoded feature vector, only the associated feature bit is on (i.e., 1) and all other feature bits are off (i.e., 0). Two distinct feature sets are prepared to evaluate the predictive power of the in-game strategy features:

- **Full feature set**: For RF and SVM, 36 features including 33 one-hot encoded features representing the cluster membership among the three clusters for each of the 11 binary tasks and three one-hot encoding-based features (i.e., low, medium, and high) representing students' pre-test performance. Since LSTMs take as input the pre-test performance and a task-specific in-game strategy per time step, it utilizes six features including three one-hot encoded features to represent the cluster membership for a strategy and three one-hot encoded features for the student's pre-test performance.

- **Pre-test performance feature set**: This set includes three one-hot encoding-based features representing low, medium, and high pre-test performances.

4.2.2 Classification Methods

5-fold cross-validation is used within the training data to tune the hyperparameters of the classification techniques based on the full feature set. After tuning the hyperparameters, each of the classifiers are trained using the full training set and are evaluated on the held-out test set. The best performing classifier is then chosen and an additional model is trained based on the pre-test performance feature set, using the same test/train splitting approach for the full feature set-level analysis. The classification process for each classifier and their results are described below.

Baseline. A majority class-based method is used as baseline. This method assigns the most frequent label in the dataset as the predicted label for all data instances. Since the most common label is the grade 'low', all labels will simply be predicted as the first class (i.e., low post-performance). The result of applying the baseline method on the full feature set shows an accuracy of 35.71% and a macro average recall of 33.33%. Since the baseline method predicts the most frequent label for all instances, the precision and F1-score are undefined.

Random forests. Utilizing a bagging sampling approach, the random forest technique generates multiple decision trees using different subsets of the training data. A random forest tree is generated by trying a random subset of available features at each split. It then classifies each point in the test set using all the trees and uses the majority vote for classifying the test point. The RandomForest (Liaw & Wiener, 2002) library in R is used to apply random forest classification on our feature set.

The set (10, 25, 50, 100, 200) is used to tune the number of trees for our model. Using a 5-fold cross-validation approach on our training set, and 25 was found to be the best number of trees for the full feature set. Due to the feature bagging (i.e., a random selection of the features as candidates for each split) and the bagging sampling approach performed in training a random forest, these classifiers are subject to randomness when being trained on a dataset. Thus, the predictive performance of random forests trained utilizing the same set of hyperparameters can vary depending on the random procedure. As a result, each round of training and evaluation on the same training/test sets will result in slightly different accuracies. Hence, the average result of 100 rounds of training and evaluating the classifier is reported. The mean and standard deviation of the results of applying random forest on the full feature set are shown in Table 4.2. The results achieve an average accuracy of 50.43%, an average precision of 52.20%, an average recall of 50.98%, and an average F1-score of 51.03%. The precision, recall and F1 measures are calculated using a macro-average of all three classes (i.e., simple average of the relative measurement of all three classes).

Support vector machines. Support vector machines (SVMs) can be used for both regression and classification tasks. In classification tasks for which data are not linearly separable, data will be transformed to a higher-dimensional space for linear separability, and SVMs are applied to classify the transformed data. The e1071 (Meyer, Dimitriadou, Hornik, Weingessel, & Leisch, 2015) library in R is used to perform the SVM-based classification. For this classification task, a third-degree polynomial kernel is used. The parameter *C* is tuned as the hyperparameter of our SVM model. *C* is the regularization parameter that controls models' tolerance for incorrect classifications during training. The set (0.01, 0.1, 1, 10, 15) is used to tune *C* on the full feature set. Using a 5-fold cross-validation approach on

the training set, $C = 1$ was found to be the best parameter to be used in the model. The results of applying the SVM model on the dataset achieve an accuracy of 41.17%, a precision of 44.14%, a recall of 39.25%, and an F1-score of 35.44%. Similar to the RF classifier the average result of 100 rounds of training and evaluating the trained classifier is reported. As there is no random parameter for this method, the standard deviation for the estimated accuracies is 0.

Both the random forest and SVM approaches achieve higher accuracies compared to the simple majority class baseline, suggesting that these methods are effective for stealth assessment. We hypothesize that the accuracy could be increased by exploiting the temporal relationships across students' sequential problem-solving tasks. Next, we describe the LSTM-based approach and its results.

Long short-term memory networks. Long short-term memory networks (LSTMs), a type of recurrent neural networks (RNNs), are a class of deep learning methods that are capable of learning temporal patterns in data. This characteristic makes LSTMs a promising candidate for classifying sequential data, including our in-game strategy features representing students' strategy use across the 11 binary challenges they solve during gameplay. A sequence of cluster associations (i.e., in-game problem-solving strategies for the 11 in-game binary representation tasks) can reveal students' problem-solving progressions as they progress through the game and can be leveraged to predict students' learning outcomes. In this study, LSTMs are investigated to model dynamic changes in students' problem-solving strategies, motivated by LSTMs' ability to preserve long-term dependencies through their three gating units (i.e., input, forget, and output gates).

Keras (Chollet, 2015) and scikit-learn (Pedregosa et al., 2011) libraries in Python are used for LSTM-based classification task. The number of LSTM layers and the number of hidden units are tuned within each layer by conducting a 5-fold cross-validation on the training set. To find the best set of hyperparameters, 15 different hyperparameter combinations with different numbers of hidden layers (1, 2, 3) and different numbers of hidden units in each layer (10, 15, 25, 50, 100) are examined. The results of this search demonstrated that networks with 2 layers and 15 units per layer using a dropout rate of 0.15 for each hidden layer produced the best results for predictive accuracy.

Like random forest models, the LSTM-based approach is also subject to an inherent randomness when being trained on the same training set multiple times. Hence, evaluating these models on the same test-set generates different outputs. This is due to the fact that deep learning approaches are sensitive to the random weights used to initialize the network. In addition, these techniques are trained on batches and the input order of the batches influence the models that are generated. Here, an average of 100 runs of training and evaluating the LSTM classifier on the same training and test set is reported. The results of applying this LSTM on the held-out test set achieve an average accuracy of 64.82%, an average precision of 63.88%, an average recall of 65.14%, and an average F1-score of 63.68%.

A summary of the results of applying all the mentioned classification methods is provided in Table 4.2. The highest score per metric is indicated in bold. The baseline and SVM approaches are deterministic so their metrics' standard deviations are zero. All classification methods outperform the majority class baseline. The LSTM-based evidence model yields considerable improvement over the other approaches, since investigating students' choice of problem-solving strategy *over time* can inform predictions about the

strength of their learning as measured by post-test performance. The results indicate that the LSTM model appears to successfully capture the temporal dependencies among features in students' problem solving.

Table 4.2. Performance (± standard deviation) of classifiers. Accuracy, precision, recall and F1-score evaluation of each classifier.

Method	Accuracy	Precision	Recall	F1
Baseline	35.7(±0.0)	N/A	33.3(±0.0)	N/A
RF	50.4(±2.5)	52.2(±2.7)	51.0(±2.4)	51.0(±2.6)
SVM	41.2(±0.0)	44.1(±0.0)	39.3(±0.0)	35.4(±0.0)
LSTM	**64.8**(±2.7)	**63.9**(±2.8)	**65.1**(±2.8)	**63.7**(±2.5)

4.2.3 In-game Strategy for Stealth Assessment

To further investigate the effectiveness of the in-game strategy features in predicting students' post-test performance, two versions of the LSTM-based model are compared. One LSTM-based model is trained on the full feature set (pre-test features together with in-game strategy features) and its prediction results are compared to the results of applying another model that is trained on a reduced set version (pre-test features only). The results of these evaluations are shown in Table 4.3, where the highest score per metric is indicated in bold.

Table 4.3. Results for LSTM-based models. Results of applying LSTM on pre-test only, in-game strategy, and full features feature sets.

Feature set	Accuracy	Precision	Recall	F1
Full FS	**64.8** (±2.7)	**63.9**(±2.8)	**65.1**(±2.8)	**63.7**(±2.5)
Pre-test FS	44.7(±7.9)	N/A	42.8(±8.3)	N/A

The results demonstrate that incorporating the in-game strategy features into the model significantly improves the predictive accuracy. Compared to the 44.66% accuracy achieved by the reduced set version (pre-test features only), the model that uses in-game strategy features in addition to pre-test features improves the accuracy by 20.16%, resulting in an accuracy of 64.82%. The considerably higher accuracy achieved by the full-set model suggests that the strategy-based features that uses sequences of strategies seems to capture an important quality of students' problem-solving strategies that are predictive of learning performance.

4.3 Discussion

Stealth assessment utilizes evidence models inferred from student behavior traces obtained from their interactions with advanced learning environment. The results of this study revealed that student behavior traces translated as students' in-game problem-solving strategies can serve as the foundation for evidence models utilized by stealth assessment. For each of the 11 problem-solving tasks in the ENGAGE game-based learning environment, we first encoded sequences of student behavior interactions into sequences of *n*-gram features to capture fine-grained temporal information that lies within the sequences. We then clustered the *n*-gram encoded features using an EM Clustering approach.

The results revealed clear distinctions in students' approaches toward solving the computational thinking challenges. More specifically, the clustering approach used grouped students into those who solved the problem in a few flips with thoughtful systematic approaches demonstrated through a sharp decay of their average error toward zero, those who solved the problem with a moderate number of flips and with thoughtful trial-and-error demonstrated through an overall decay of average error with some spikes, and those who

solved the problem with a long sequence of flips and with seemingly random trial-and-error demonstrated through no meaningful average error decay toward zero.

We then used students' cluster memberships across different tasks as an indicator of their in-game problem-solving strategies and used these problem-solving strategies to inform the ECD model for predicting students' post-test performance. The results demonstrated that the in-game strategy features provide strong evidence for LSTM-based ECD models and more generally for the use of stealth assessment. Because LSTM-based ECD models that encompass in-game strategy features appear to capture the temporal relationships between strategies chosen among consecutive tasks, they can accurately predict students' post-test performance.

It is important to note that the in-game strategy features are derived directly from log data and are generated based on an unsupervised method, EM Clustering. The automated process of extracting students' in-game problem-solving strategy makes it a promising approach for evidence modeling. This approach can be used on other learning environments in which students solve problems by performing sequences of actions from a limited pool of available actions to generate ECD models.

Evidence models derived from students' interactions with game-based learning environments such as the one introduced in this work can be used to infer students' problem-solving strategies. Intelligent game-based learning environments can then use this information to scaffold students to use better problem-solving strategies if needed (e.g. when signaled by the evidence model that a student is following a strategy associated with a poor learning outcome). In addition to strategy scaffolding, the evidence models can also get combined with knowledge modeling to support knowledge scaffolding. For example, in the

ENGAGE game-based learning environment, generating a desired binary sequence through long series of flips and random trial-and-error might be an indicator of lack of knowledge about digit weights in a binary string. This lack of sufficient knowledge can be addressed with a timely tutorial on binary digit weights.

The results of the work reported here, as well as those found in related work on inferring student problem-solving strategies from behavior trace data (Kerr & Chung, 2012), suggest that modeling students' problem-solving strategies may lead to improved assessment. Inferring students' problem-solving strategies can also contribute to designing adaptive learning environments that can detect knowledge deficiency and inappropriate choice of strategy, and thus provide learners with appropriate scaffolding.

4.4 Conclusion

Stealth assessment holds considerable potential for making adaptive and effective game-based learning environments. Despite the abundance of dynamic student data that can be captured from learners' interactions with these environments, effective stealth assessment poses significant challenges. In this work, we have introduced a strategy-based temporal analytics framework for stealth assessment that uses an LSTM-based ECD model trained on sequences of student problem-solving strategies learned from clustering *n*-gram representations of their in-game behaviors. Based on the results emerged from an evaluation of the predictive accuracy in regard to learning outcomes, the strategy-based temporal analytics framework outperformed baseline models that did not capture the temporal dependencies of strategy incorporations through time.

In the future, multiple granularities of strategy representations can be explored in order to take advantage of hierarchical deep learning methods. Furthermore, the real-time

effectiveness of the LSTM-based model proposed in this work can be explored by incorporating this model in an actual game-based learning environment for providing stealth assessment and supporting adaptive scaffolding.

CHAPTER 5:

ASSESSING MIDDLE SCHOOL STUDENTS' COMPUTER SCIENCE THROUGH PROGRAMMING TRAJECTORY ANALYSES-1

CS practices such as creating and systematic testing and refining of computational artifacts are signaled by sequences of students' actions over time. Inspecting steps students take to develop a program may reveal richer information about their CS knowledge and competencies than inspecting their final program alone (Blikstein et al., 2014). Similarly, when drawing evidence from students' testing and refining of CS artifacts, each submission needs to be assessed in relation to the preceding submissions instead of in isolation.

5.1 Modeling CS Competencies

In our game-based learning environment, ENGAGE, students test their proposed solutions for a programming CT challenge by generating a block-based programming artifact and submitting it to the system. These proposed submissions represent intermediate points of development by the student until a submission correctly solves the room challenge. Hence, we analyze the intermediate solutions to identify students' strategies in testing and refining their submitted artifacts. We follow a blended top-down and bottom-up approach where we first determine related FKSAs through a series of analyses of the K-12 computer science framework (*K-12 Computer Science Framework*, 2016) and previous work in assessing middle and high-school CS proficiencies (Diana et al., 2017; Grover & Basu, 2017; Grover et al., 2017; Grover, Bienkowski, Niekrasz, & Hauswirth, 2016; Seiter & Foreman, 2013). We then revise the FKSAs based on the evidence found through a preliminary investigation of the dataset. The final set of FKSAs are as follows:

1. Students can develop and implement effective and generalizable algorithms to solve a specific computational problem.
2. Students can use conditional statements appropriately.
3. Students can use loop statements appropriately.
4. Students can select and combine control structures, such as loops, event handlers, and conditionals, to create more complex program behavior.
5. Students can identify patterns as opportunities for abstraction, such as recognizing repeated patterns of code that could be more efficiently implemented as a loop.
6. Students can create a program to solve a specific computational problem.
7. Students can examine existing programs to understand their functionality.
8. Students can compare results to intended outcomes and determine whether given criteria and constraints have been met.
9. Students can identify and fix errors using a systemic and iterative approach.

5.2 Assessment Design

The primary goal of this study is to investigate students' productive and unproductive usage of CS practices and concepts. Thus, after determining the relevant FKSAs in the domain modelling step, we identified the specifications for the assessment following the conceptual assessment framework. We used a similar blended top-down and bottom-up approach used in the domain modeling step devising a measurement model based on the task (i.e., bottom-up) and identified FKSAs (i.e., top-down). In particular, evidence rules are designed specific to the task at hand to provide assessment arguments for the proficiency of the FKSAs in the student model.

To achieve this goal, a three-dimensional assessment model was devised where the first dimension measuring students' proficiency in static CS constructs, such as effective utilization of loops and conditionals. The second dimension measures proficiencies based on students' dynamic behaviors of testing and revising programming artifacts over time. The third dimension measures students' problem-solving efficiency by performing an overall measurement of quantifiable observations across all submissions (i.e., number of submissions and the percentage of identical submissions).

To assess students' artifacts along the first dimension, each construct of each submission is graded for each student individually and then an average score is calculated among all submissions per construct. The average score over their trajectory of submissions provides insight into students' knowledge of CS constructs, such as appropriate use of loops and conditionals, as it develops over time. Table 5.1 shows the detailed rubric for assessing the CS constructs identified through the domain modeling phase.

The second dimension focuses on evidence that represents the practice of testing and refining programming artifacts. The evidence for this practice unfolds over consecutive submissions. Hence, to effectively interpret the changes students made to their code in each submission, we examine not only at the immediate previous submission but the history of preceding submissions. Three aspects of students' testing and refining strategies are assessed: 1) novelty, i.e., whether the same program has been submitted in the previous 10 submissions; 2) positivity, i.e., whether the new submission is closer to the correct solution relative to the previous submission; and 3) scale, i.e., what is the scale of the change that the student has made in their current submission compared to the previous one. Table 5.2 shows each aspect, possible scores and explanation of how each value is assigned to a refinement.

Table 5.1. CS constructs assessment. CS constructs assessment items, their underlying FKSAs, and detailed rubrics for each item.

Rubric Category	FKSA	Detailed Rubric
Design and implementation of effective and generalizable algorithms	1, 6	• All three important constructs (one loop statement and two conditional statements) are implemented properly. • Constructs and blocks are positioned in the correct order and hierarchy. • The loop construct runs for appropriate amount of time.
Appropriate use of loop statements	3, 5	• The repeat block is present. • The iteration value is set to a positive number. • It encompasses at least one block.
Appropriate use of conditional statements	2	• At least one conditional statement is used. • A conditional statement checks the size of two adjacent containers and swap them if they are not ordered properly. • A conditional statement checks if the arm has reached the right wall and reset it to the left wall.
Appropriate combination of loops and conditional statements	4	• Each if is nested under a repeat statement. • Ifs are not nested under each other.

Table 5.2. Testing and refining assessment. Refinement scores, possible values, and explanation.

Refining Scores	Possible Values	Explanation
Novelty	• New	• The submitted artifact has not been submitted in the 10 previous submissions.
	• Repetitive	• The submitted artifact is present in the 10 previous submissions.
Positivity	• Positive	• The current submission is closer to the correct solution compared to the previous one.
	• Not positive	• The current submission is not closer or is farther from the correct solution compared to the previous submission.
Scale	• Large change	• More than two changes of any kind.
	• Small change	• One or two changes of any kind.
	• No change	• No change has been made.

The third dimension assesses quantifiable observations across all submissions representing students' problem-solving efficiency. These observations include the normalized number of total submissions, proportion of identical submissions to the total number of submissions, and proportion of attempts of unlocking the door to the total number of

submissions. Students' total number of submissions are normalized by the highest number of submissions among all students to get a measure of their relative efficiency in solving the problem.

FKSA 9 requires students to engage in systematic refinement of their artifacts. Hence, a high ratio of identical submissions can indicate a student's lack of systematic test and refinement skills. As described in section 3.1, to solve the bubble sort challenge, students need to first submit a programming artifact that sorts the containers. They then need to attempt to unlock the door by executing a program that checks if the containers are sorted and opens the door if they are. Hence, attempting to open the door without sorting the containers can indicate a student's inability to understand an existing artifact (FKSA 7) or their lack of ability to compare results to intended outcomes (FKSA 8).

While most of the measures introduced in this work are objective, the positive refinement score can be subjective depending on the annotator's interpretation of an artifact. To validate the labeling process of this specific measure, two human annotators labeled 20% of the total submissions of programming artifacts extracted from trace data, achieving an inter-rater agreement of 0.801 using Cohen's Kappa (Cohen, 1960). Before tagging the remainder of the corpus, all instances that were tagged differently were discussed. Then, one annotator tagged the rest of the dataset.

5.3 Analysis and Results

Below, we report the results of applying the three-dimensional analytic framework on data collected from 22 students' attempts to solve the bubble sort challenge. Students' common problem-solving strategies are identified in terms of test and refinement practice patterns.

The findings and how they can help researchers and teachers identify and assess productive CS practices are then discussed.

5.3.1 Computer Science Constructs

To assess FKSAs 1 through 6, four assessment items are designed: 1) design and implementation of algorithms, 2) appropriate use of loop statements, 3) appropriate use of conditional statements, and 4) appropriate combination of loops and conditional statements. Table 5.3 depicts the distribution of students based on their average scores for each assessment item categorized as low, medium, and high. For each assessment item score, low is less than one third of the full score, medium is between one third and two third of the full score, and high is between two third and the full score. The design and implementation of algorithms item is a generalized assessment item that assesses not only the appropriate use and combination of loops and conditionals, but also assesses the appropriate position and hierarchy of code blocks to generate the correct solution.

A closer look at the programming artifact trajectories of students with high average algorithmic scores shows that they all have scored high in other categories. Similarly, students who scored low on the algorithm item also scored low in all other items. Additionally, 75% of students who scored low in the appropriate use of repeat as well as the appropriate combination of repeats and conditionals items also scored low in the algorithm category, while all students who scored low in appropriate use of loops category scored low in algorithms too. To quantitatively analyze the results, we evaluated the Pearson correlation coefficients for several variables, revealing significant correlation between algorithmic scores and: appropriate use of loop statements ($r(20)=0.908$, $p<0.001$), appropriate use of

conditional statements ($r(20)=0.969$, $p<0.001$), and appropriate combination of loops and conditional statements ($r(20)=0.974$, $p<0.001$).

The results suggest that not only having basic knowledge of essential CS concepts is helpful in being able to develop and implement effective algorithms, but it is very difficult for students to generate meaningful algorithms without having knowledge of these constructs. Further, the fact that all the scores are obtained from averaging students' scores along their programming artifact trajectories suggests that there is a consistency in students' demonstration of CS competencies as they unfold over time.

Table 5.3. Students' distribution based on their CS constructs assessment scores. Distribution of students based on their average score for each CS constructs assessment item.

Assessment Items	Low	Medium	High
Design and implementation of algorithms	40.90%	27.27%	31.81%
Appropriate use of loop statements	36.36%	31.81%	31.81%
Appropriate use of conditional statements	36.36%	31.81%	31.81%
Appropriate combination of loops and conditional statements	54.54%	13.63%	31.81%

5.3.2 Testing and Refining Programming Artifacts

Students' testing and refining behavior is assessed over their course of consecutive submissions. To evaluate students' refining strategies three aspects are defined: novelty, positivity and scale. Students' scores in the novelty and positivity aspects are divided into

high and low scores based on the distribution of their scores (i.e., median split). Based on these results, 31.1% of students have high positivity and high novelty scores, 13.6% of students have high positivity but low novelty scores, 18.18% of students have low positivity but high novelty scores, and 36.4% of students have low positivity and low novelty scores.

Results of a set of correlation analyses demonstrate that having high novelty ($r(20)=0.5301$, $p<0.01$), high positivity ($r(20)=0.5605$, $p<0.001$), and low scale ($r(20)=0.6776$, $p<0.001$) are significantly correlated with higher algorithmic scores. This indicates that students who have more knowledge about CS constructs tend to make smaller, incremental, positive changes while also attempting novel solutions. A follow-up correlational analysis between positivity and total number of submissions demonstrates that higher ratio of positivity is negatively correlated with total number of submissions. This suggests that students who acquire more positive changes obtain the productive results faster than students who engage in random trial-and-errors. Finally, a negative correlation exists between number of open door trials and the algorithmic scores ($r(20)=0.553$, $p<0.1$). This supports the hypotheses discussed in the methodology section that a high number of attempts to open the door before correctly sorting the containers could be due to students' lack of ability to parse and troubleshoot an existing programming artifact.

Conducting small scale studies of students' submitted programming artifact trajectories provides us with interesting insights on how students engage in testing and refining their programs. For example, even though trying novel solutions is considered an efficient refinement strategy, there are instances of efficient and skillful students (as demonstrated by their algorithm and total number of submissions scores) that have a relatively high number of repetitive submissions. This could be explained by the possibility

of these students trying to observe and learn the effect of their code by applying a highly iterative systematic change strategy to their submissions. Other examples show that an effective refinement strategy is to make big changes in the beginning and the refining the code utilizing smaller but positive steps until the solution is reached.

Finally, the pre- and post-test assessments administered in this study were not aligned with the CS concepts and practices that we assessed within the bubble sort challenge. Therefore, our results didn't show a significant correlation between students' in-game assessed competencies and their post-test performance.

5.4 Conclusion and Future Work

In this study, an assessment framework is designed to measure students' CS FKSAs. In particular a three-dimensional assessment is created, where the first dimension assesses students' knowledge of essential CS constructs, the second dimension assesses students' dynamic testing and refining strategies, and the third dimension assesses their overall problem-solving efficiency. The results demonstrate that students' knowledge of basic CS constructs, such as appropriate use and combination of control structures, serves as the foundation for designing and implementing effective algorithms. Further, students' testing and refining strategies are tested over the three dimensions of novelty, positivity, and scale. The results demonstrate that students with higher algorithmic capabilities tend to make more novel, positive, and small-scale changes. Moreover, common effective refinement strategies are identified by performing analyses of students' trajectories in developing their programming artifacts.

This study represents a first step toward building a generalizable assessment to understand students' testing and refining approaches over time. In the future, it will be

important to collect and analyze a larger corpus of students' problem-solving data since the dataset used for this study was relatively small, representing a smaller set of strategies than we anticipate in larger populations of students. It will also be important to refine the assessment to capture more fine-grained aspects of students CS FKSAs. Another important direction for future work will be to use the programming trajectory patterns discovered in this study to inform the design of adaptive scaffolding for block-based programming environments that can evaluate students' testing and iterative refinement approaches automatically, as well as guide teachers' individualized instruction and feedback for students.

CHAPTER 6:

ASSESSING MIDDLE SCHOOL STUDENTS' COMPUTER SCIENCE THROUGH PROGRAMMING TRAJECTORY ANALYSES-2

In Chapter 5, we presented an assessment to evaluate students' CS FKSAs based on their submitted solutions to a bubble sort algorithm programming challenge. We also presented the results of piloting this assessment on data collected in the first round of a data collection where students played the ENGAGE game in pairs and completed a game content pre- post-test assessment (Buffum et al., 2016). Utilizing the preliminary results of applying the devised assessment in Chapter 5, we applied appropriate adjustments in the rubric and the identified FKSAs. For example, preliminary analysis of students' submissions revealed evidence that students' knowledge of certain FKSAs were not captured with our rubric. Furthermore, there were overlaps between different items of the rubric in capturing evidence from students' submitted artifacts. Some aspects of the rubric were also revised to allow for more objective assessment of the items. We applied the revised rubric on data collected in the second round of the data collection where students played the game individually. Furthermore, in this round of the data collection, the students completed a general CT pre-post-test assessment (Wiebe et al., 2019).

6.1 Rubric Revisions

According to our preliminary analysis, there were many overlaps between rubric items corresponding to design and implementation of effective and generalizable algorithms and rubric items corresponding to other categories in the computer science concepts section of the assessment. In the revised version, we substituted the overlapping rubric items with two general and subjective measurements of effectiveness and conciseness of the submitted

programming artifacts. While objective rubric items utilize specific code structures that represent students' CS FKSAs competencies, the two subjective measurements allow instructors to capture any overlooked aspects of the programming artifacts. In this rubric, extra wrong code is captured under item "Conciseness" and overall quality of the implemented algorithm is captured under item "Effectiveness". The revised rubric for computer science constructs is shown in Table 6.1.

We also revised the testing and refining section of the assessment to make it more objective. While the overall criteria of the assessment remain unaffected, the approach to assess those criteria is revised in the new assessment to allow for more consistency across graded artifacts. Making rubric items more objective helps to reduce the manual labor needed to apply this assessment as well as the noise in our training dataset. The two affected rubric items are positivity and scale. In the previous assessment, positivity was assessed based on a graders' opinion about whether the new submission has become closer to the correct solution. In the revised rubric, we consider the numerical grade difference between two consecutive computer science competencies to distinguish between positive, negative, and neutral changes.

Furthermore, in the previous version of the assessment, a human annotator looked for instances of basic changes between two consecutive artifacts to evaluate the value of the item scale. The basic changes consist of adding a block, removing a block, and changing a block's position. In the current assessment, we measure the scale of the change by calculating the difference between two subsequent submissions encoded as structural *n*-grams. To assign the value "large change" to the scale item of the rubric, we expect a minimum of three basic changes between two consecutive submissions. At the same time, one single change in the

new submission can result in several changes in the encoded structural *n*-grams. As a result, we consider 40 units of difference between two consecutive encoded structural *n*-grams as the approximate threshold for making large scale changes.

6.2 Results and Analysis

We replicated the study presented in Chapter 5 by applying the revised rubric on data collected in the second round of data collection. In this round, we collected data from 69 students resulting in 1570 submissions and 642 unique programming artifacts. In the following sections, we present the results of applying the revised CS assessment on data collected in the second round of data collection.

6.2.1 Computer Science Constructs

Figure 6.1 depicts a stacked bar graph of the distribution of students' average scores for each assessment item: 1) design and implementation of algorithms, 2) appropriate use of loop statements, 3) appropriate use of conditional statements, and 4) appropriate combination of loops and conditional statements, categorized as low, medium, and high. For each assessment item, low is less than one third of the full score, medium is between one third and two third of the full score, and high is between two third and the full score.

In the revised rubric, the design and implementation of algorithms item is a generalized assessment that subjectively assesses the effectiveness and conciseness of the code. It also contains objective items that assess the presence of essential blocks and code structures that represent students' understanding of the algorithm being implemented.

Table 6.1. Revised CS constructs assessment. Revised CS constructs assessment items, their underlying FKSAs, and detailed rubrics for each item.

Rubric Category	FKSA	Detailed Rubric
Design and implementation of effective and generalizable algorithms	1, 6	• The artifact contains all necessary code elements • The code elements have the correct order, and hierarchy (Effectiveness). • The artifact does not contain redundant code elements that falsify the logic of the algorithm (Conciseness).
Appropriate use of loop statements	3, 5	• The repeat block is present. • The iteration value is set to a positive number. • It encompasses at least one block.
Appropriate use of conditional statements	2	• Both necessary conditional statements are used. • A conditional statement checks the size of two adjacent containers and swaps them if they are not ordered properly. • A conditional statement checks if the arm has reached the right wall and reset it to the left wall.
Appropriate combination of loops and conditional statements	4	• There is at least one instance of each conditional nested under a repeat statement. • There is at least one instance of two conditionals at the same level.

According to the results, 20.29% of students scored low, 59.42% scored medium, and 20.29% scored high in the algorithm section. This suggests that most of the students have moderate skills and knowledge corresponding to algorithm thinking, and a smaller percentage have very high or very low understanding of this concept. Moreover, 20.20% of students scored low, 20.29% scored medium, and 59.42% scored high in the understanding of loops section. This suggests that most students have a good understating of loops and how to implement them. Furthermore, 47.83% of students scored low, 21.74% scored medium, and 30.43% scored high in the understanding of conditional section. This indicates students' general lack of understanding of conditionals and demonstrates the need for providing them with better scaffolding and practice corresponding to this concept. Finally, 59.42% of students scored low, 26.09% scored medium, and 14.49% scored high in the combination of loops and conditionals section. This is not surprising since this category is more complicated and requires proper knowledge of each individual concept involved in this section.

Similar to the results presented in Chapter 5, students with high average algorithmic scores scored high in other categories. Similarly, students who scored low in the algorithm section also scored low in every other section within the computational thinking concepts category of the assessment. Furthermore, 57.14% of students who scored low in appropriate use of loops, and appropriate use of conditionals sections also scored low in the algorithm section. Moreover, all the students who scored low in appropriate combination of loops and conditionals category scored low in algorithms too.

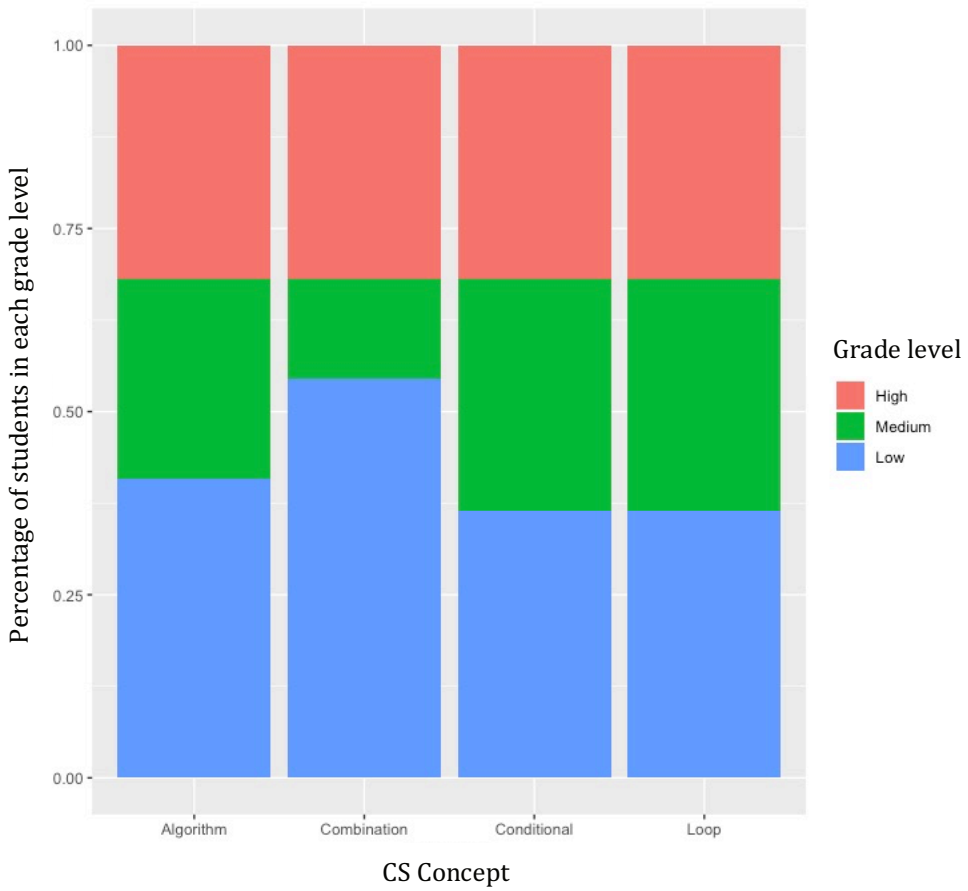

Figure 6.1. Students' distribution based on their CS constructs average scores. Stacked bar graph of students' distribution of scores for each CS construct item.

To quantitatively analyze the results, we evaluated the Pearson correlation coefficients of several variables, revealing significant correlations between algorithmic scores and: appropriate use of loop statements ($r(67)=0.603$, $p<0.001$), appropriate use of conditional statements ($r(67)=0.862$, $p<0.001$), and appropriate combination of loops and conditional statements ($r(67)=0.872$, $p<0.001$).

These results confirm the preliminary findings that having basic knowledge of essential CS constructs is crucial for developing and implementing effective algorithms. In fact, devising meaningful algorithms is challenging for students who does not have sufficient knowledge of basic constructs.

6.2.2 Testing and Refining Programming Artifacts

We also utilized the revised assessment to evaluate students' testing and refining strategies over their course of consecutive submissions by the aspects of novelty, positivity and scale. Students' scores in the novelty and positivity aspects are divided into high and low scores based on the distribution of their scores (i.e., median split). Based on these results, 37.68% of students have high positivity and high novelty scores, 13.04% of students have high positivity but low novelty scores, 13.04% of students have low positivity but high novelty scores, and 36.23% of students have low positivity and low novelty scores.

Results of a set of correlation analyses demonstrated that having high novelty ($r(67)= 0.378$, $p<0.01$), high positivity ($r(67)= 0.398$, $p<0.001$), and low scale changes ($r(67)= 0.293$, $p<0.001$) are significantly correlated with higher algorithmic scores. This confirms our previous findings which indicated that students who are more competent in designing and implementing algorithms, and thus, have more knowledge about essential CS constructs, tend to make smaller, incremental, and positive changes while also attempting novel solutions.

6.2.3 Problem-Solving Efficiency

Follow-up correlational analysis on students' efficiency in problem-solving demonstrated a negative correlation between normalized total number of submissions, higher ratio of positive changes ($r(67)= -0.657$, $p<0.001$) and higher algorithmic scores ($r(67)= -0.301$, $p<0.05$). A negative significant correlation also exists between the algorithmic scores and percentage of repetitive submissions ($r(67)= -0.383$, $p<0.05$). Not surprisingly, a significantly negative correlation exists between the number of repetitive submissions and the ratio of positive changes ($r(67)= -0.802$, $p<0.001$). Finally, a negative correlation exists between normalized number of open door and the algorithmic scores ($r(67)= -0.207$, $p<0.1$).

These results suggest that students who acquire more positive changes obtain the productive results faster than students who engage in random trial-and-errors. Furthermore, it suggests that a high number of attempts to open the door before correctly sorting the containers could be due to students' lack of ability to parse and troubleshoot an existing programming artifact. Overall, these results demonstrate that students who have higher competency in CS constructs are more efficient in their problem-solving.

6.3 Discussion

The results of applying the revised rubric on students' submitted programming artifacts demonstrated an overall alignment with the preliminary results presented in Chapter 5. In this study, the revised rubric is applied on students' individual submissions as opposed to students' paired submissions in the first study. Furthermore, in the second study, we applied the assessment on data obtained from a larger population (n=69) of students compared to the previous study (n=20).

While the relationship among different sections of the assessment generally followed the patterns observed in Chapter 5 in terms of direction and significance of correlations, the degree of correlation varied from one study to the other. This is understandable as we have made fundamental revisions to the original rubric in the second study. Furthermore, the different nature of data utilized in the first and the second studies makes it hard to replicate the exact same results among them.

Overall, the preliminary and main results of assessing students' CS FKSAs demonstrated a strong dependency between students' understanding of algorithm design and implementation and their understanding of other essential CS constructs, such as loops and conditionals. Furthermore, it confirmed the previous relationship between algorithmic

competency and patterns of testing and refining between consecutive submissions. According to both set of results, students with higher knowledge of algorithms tend to make more positive, novel, and small-scale changes. Furthermore, students with higher proficiency in design and implementation of algorithms tend to solve the problem more efficiently as demonstrated through fewer number of overall submissions, lower percentage of repetitive submissions, and fewer number of unsuccessful executions of the open door command.

6.4 Conclusion

Following an evidence-centered assessment design approach, we devised a revised version of the rubric presented in Chapter 5 to assess students' understanding of essential CS constructs, their testing and refining approaches and their problem-solving efficiency. Our results demonstrated the effectiveness of this approach in capturing insightful evidence from students' submitted artifacts that represent their understanding of essential CS concepts and practices.

We revised the original rubric based on the preliminary results obtained from applying it on data collected from students' paired game play. We replaced overlapping rubric items in the design and implementation of algorithms section of the assessment with two subjective items that capture effectiveness and conciseness of the generated algorithms. Furthermore, we provided detailed guidelines for instructors with regard to grading the "Effectiveness" and "Conciseness" of the artifacts. While the previous rubric used subjective guidelines for assessing positivity and scale of consecutive submissions, the revised rubric uses objective guidelines to assess these two aspects of the assessment. This will reduce the need for manual labor when grading the artifacts as well as the noise in our dataset.

The results of applying the revised rubric confirmed the overall findings of the preliminary study. Based on our finding, students' knowledge of essential CS constructs serves as the foundation for their understanding of algorithmic scores. Furthermore, students with higher algorithmic scores tends to make more positive, novel, and incremental small-scale changes. These results clearly indicate that in order to develop students' CS and CT problem-solving skills, it is essential to build a strong foundation for their CS FKSAs.

CHAPTER 7:

A SEMI-AUTOMATED FRAMEWORK FOR ASSESSING STUDENTS' UNDERSTANDING OF ESSENTIAL CS CONSTRUCTS BASED ON THEIR PROGRAMMING ARTIFACTS

In chapter 6, we presented an assessment framework to assess students' CS FKSAs in a CT challenge that involved creating a program for the bubble sort algorithm. In this study, we present a semi-automated learning approach for assessing students' understanding of essential CS constructs in the bubble sort challenge based on their submitted programming artifacts. We utilize the CS constructs assessment items presented in Table 6.1, to label a training dataset collected from students' submitted programming artifacts to the bubble sort challenge.

Following a blended hypothesis-driven approach adopted from (Grover et al., 2017), we conduct a data-driven assessment of students' essential CS constructs. Adopting this approach enables us to assess students' understanding of each essential CS construct individually. While this holds for our assessment framework, in this study, we focus on automatically assessing programming artifacts' overall algorithmic quality as many CS constructs such as loops, conditionals, and their combination can be trivially assessed using our assessment framework. Hence, from here on, we only mention the semi-automated assessment of the algorithmic quality of programming artifacts. The details of this semi-automated assessment framework are discussed below.

7.1. Methodology

We utilize a supervised learning approach for assessing programming artifacts' algorithmic quality. The supervised learning approach involves three main steps. The first step is to systematically label our training dataset in accordance with the identified essential CS

constructs. We do this by utilizing the revised rubric presented in Table 6.1. It is important to notice that the overall algorithmic quality of an artifact is the result of students' proficiency in each essential CS construct. As a result, we use the overall grade to label our training dataset as opposed to using the "appropriate design and implementation of algorithms" grade alone. The second step is to extract features from students' submitted programming artifact snapshots that represent their understanding of the identified essential CS constructs. This is accomplished by encoding students' programming artifacts in terms of structural *n*-grams. Finally, the last step is applying learning models on the generated feature set to infer students' grades based on the labeled data corpus. In this study, we utilize a variety of regression models including linear, lasso, ridge, Support Vector Regression (SVR), and Gaussian process regression models to infer artifacts' overall algorithmic quality. These three steps are described in detail in the following sections.

7.1.1 Data Preparation

We collected data from 69 consented students' interactions with the bubble sort challenge in the ENGAGE game. In total, students submitted 1,570 artifacts resulting in 642 unique programming artifacts that we used as our training dataset. To grade the artifacts in our training dataset, we utilized the revised rubric presented in Chapter 6 (Table 6.1). While most items in this rubric are objective and can be hard-coded into an assessment algorithm, the algorithmic "Effectiveness" and "Conciseness" grades are subjective and depend on the annotator's interpretation of the quality and conciseness of the artifact. To validate the labeling process of these two subjective measures, two human annotators with a computer science background separately labeled 20% of the entire submissions for algorithmic quality and conciseness, achieving an inter-rater agreement of 0.848 for quality and 0.865 for

conciseness using Cohen's kappa (Cohen, 1960). Before tagging the remainder of the corpus, all instances that were tagged differently were discussed. Afterwards, one annotator tagged the rest of the dataset. The annotated dataset is used to grade the data corpus for our learning task where the assigned grades serve as the ground-truth for our data corpus.

It is important to note that the human labeling process introduces noise to our training dataset. This is because different graders might have different perceptions about a programming artifacts' algorithmic "Effectiveness" and "Conciseness". In fact, one annotator might grade the same artifact differently at different times. As a result, our dataset is inherently noisy, and this must be taken into account when designing our inference model. Even though we cannot eliminate all the noise by utilizing a systematic rubric, we can reduce this noise by making parts of the rubric objective and giving guidelines to instructors for the subjective parts. Furthermore, we use a Gaussian process regression model as one of our regression techniques to help handling the remaining noise in our dataset.

7.1.2 Feature Engineering

In order to infer students' grades based on their submitted programming artifacts, we need to extract structural features that are representative of the semantic information in students' codes. One advantage of block-based programming environments is that they prevent students from making syntactic errors, and therefore eliminate the need for considering that kind of error in our feature extraction process.

In order to extract features that capture essential structural and semantic information stored within the programming artifacts, we perform a structural *n*-gram encoding of the artifacts' abstract syntax trees (ASTs). Since the submitted artifacts are stored as programming snapshots, we first apply an intermediate transformation from programming

snapshots to their corresponding ASTs. We then encode the generated ASTs into their corresponding structural *n*-grams. Each of these steps are described in detail in the following sections.

Generating abstract syntax trees. To transform students' programming snapshots to their corresponding ASTs, we first follow an approach proposed by (Shamsi & Elnagar, 2012) to encode each block in the programming snapshots into a string of digits that jointly represents the code block type and its relative position and hierarchy in the overall code structure. To demonstrate the code block type, we assign an identification code to each block and use it as the first digit in the encoded representation of the block. Table 7.1 demonstrates the identification code for each block. To encode the structural information of each code block, we concatenate the structural code of its parent with the blocks' position relative to its siblings. Finally, each encoded block is represented by a string of digits, in which the first digit is the block type obtained from Table 7.1, and the remaining digits demonstrate the code hierarchy and order relative to other blocks in the programming artifact. We use these encoded blocks to generate the corresponding abstract syntax tree of the programming artifact. Figure 7.1 shows a snapshot of a sample programming artifact, its numerically encoded representation (left), and its corresponding AST (right).

Extracting structural *n*-gram encoded features. After generating the ASTs from their corresponding programming artifacts, we conduct a structural *n*-gram encoding to capture essential structural information stored within the artifacts. Two important structural information types in ASTs are hierarchical and ordinal. The hierarchical information demonstrates what blocks are nested under another, and the ordinal information demonstrates the placement order of blocks.

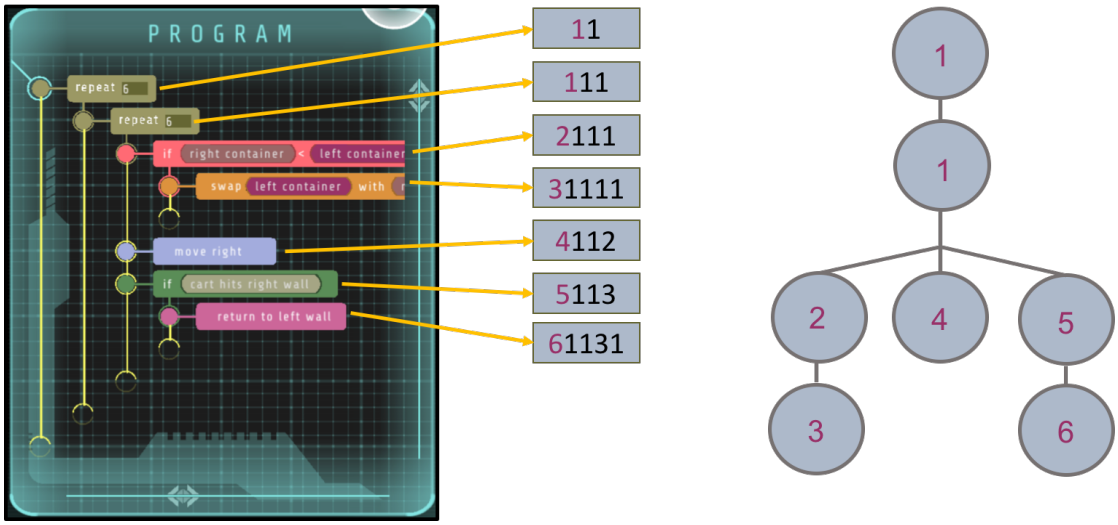

Figure 7.1. A sample programming artifact submitted for the bubble sort challenge and its corresponding abstract syntax tree. (Left) a sample programming artifact and its corresponding encoded blocks. (Right) the abstract syntax tree generated from the string encoded blocks.

Table 7.1. Block identification codes. Identification code for each block type in the bubble sort challenge.

Block Type	Identification Number
Repeat	1
If right container < left container	2
Swap left container with right container	3
Move right	4
If cart hits right wall	5
Return to left wall	6

One of the main affordances of our proposed semi-automatic assessment framework is the ability to assign partial grades to partially correct solutions. In order to assess partially correct solutions, structural information with different levels of granularity needs to be captured from ASTs. Hence, we extract *n*-grams with varying lengths of *n* to capture the most fine-grained structural information present in an AST. We repeat the *n*-gram encoding process separately for hierarchical feature extraction and ordinal feature extraction. A sample hierarchical and ordinal *n*-gram feature set generation is demonstrated in Figure 7.2 and Figure 7.3.

In Figure 7.2 (left), each colored circle shows hierarchical (top to bottom) *n*-gram encoding of a specific *n*. In this example, we have hierarchical encoding of *n*-grams of size one (pink ovals), two (purple ovals) and three (the green ovals). Figure 7.2 (right) demonstrates the extracted values for each *n*-gram encoded feature for one sample vertical path in the AST. All the other *n*-gram feature values are zero since they are not available in this AST. Figure 7.3 (left) demonstrates the same sample AST with its ordinal (left to right) *n*-gram encoding. In this example, we have ordinal encoding of *n*-grams of size one (pink ovals), two (purple ovals) and three (the green ovals). Similar to Figure 7.2 (right), Figure 7.3 (right) shows the extracted values for each *n*-gram feature for one sample horizontal path of the AST.

We then merged the two feature sets together to build the final feature set containing both hierarchical and ordinal *n*-gram encodings corresponding to each programming artifact. Note that unigrams are repeated in both hierarchical and ordinal *n*-gram encoding of the ASTs, and thus, only one of them is used in the final feature set.

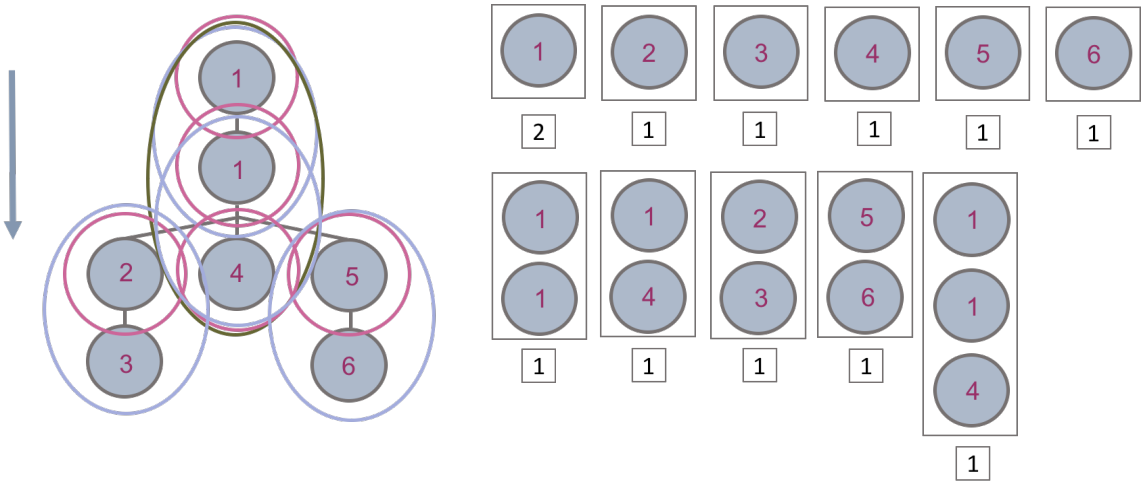

Figure 7.2. An AST generated from a sample programming artifact submitted for the bubble sort challenge and its hierarchical *n*-gram encoding. (Left) an AST and its hierarchical unigrams, bigrams, and *3*-grams marked by pink, purple and green ovals respectively. (Right) the feature set generated from hierarchical *n*-gram encoding of one sample vertical path in the AST.

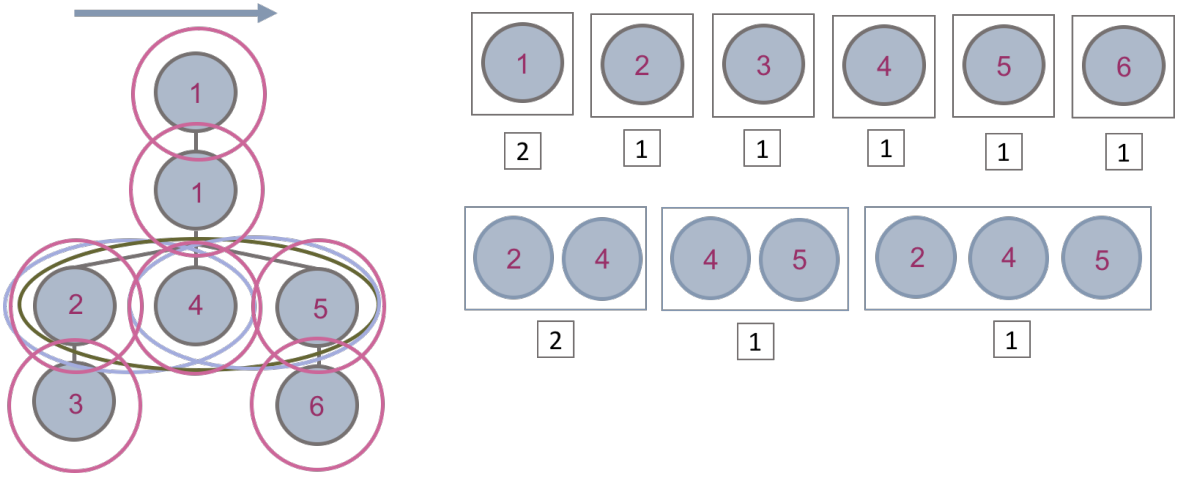

Figure 7.3. An AST generated from a sample programming artifact submitted for the bubble sort challenge and its ordinal *n*-gram encoding. (Left) an AST and its ordinal unigrams, bigrams, and *3*-grams marked by pink, purple and green ovals respectively. (Right) the feature set generated from ordinal *n*-gram encoding of one sample horizontal path in the AST.

Preliminary explorations showed including sequences of lengths larger than four for hierarchical *n*-grams and three for ordinal *n*-grams exponentially increases the sparsity of the dataset. To eliminate the sparsity issue, we capped the *n*-gram size at four for the hierarchical *n*-gram encoding and at three for the ordinal *n*-gram encoding. Our final feature set consists of sequences of length one (i.e., unigram features) to sequences of length four (i.e., 4-grams) that are repeated at least three times throughout our dataset, resulting in 184 distinct features.

7.1.3 Inferring Programming Artifacts' Scores

We trained a variety of regression models on the structural *n*-gram encoded features to infer submitted programming artifacts' grades based on the previously graded data corpus. We then utilized a 10-fold cross-validation approach to evaluate the performance of our incorporated models. In particular, we used linear regression as our baseline regression model, and four well-established regression models including lasso, ridge, Support Vector Regression (SVR), and Gaussian process to infer the grades for the submitted programming artifacts.

Lasso and ridge regression are used due to their utilization of L1 and L2 regularization which result in reduced overfitting and variance error in comparison to linear regression. Lasso regression also enables us to perform feature selection due to L1 regularization. We utilize lasso regression to identify the most influential features in our structural *n*-gram encoded feature set corresponding to the inferred grades. This will provide us with insight into the most deterministic evidence in students' codes related to their learning. Furthermore, we use support vector regression and Gaussian process since their utilization of kernels make them suitable candidates for datasets similar to ours where the

number of features is high relative to the number of data points. Finally, we utilize Gaussian process regression to handle the noise resulting from the subjective nature of human grading.

To evaluate the effectiveness of the structural *n*-gram encoded feature set, we apply the aforementioned regression techniques on a baseline feature set. We use a bag of words approach to build the baseline feature set. In this approach, features are constituted of different block types available for solving the computational thinking problem at hand, and the value for each feature is how many times that block is represented in the corresponding artifact. Since our programming artifacts are small in length and are constructed from a limited number of different block types, the bag-of-word baseline approach is suitable for evaluating our proposed approach.

We use the scikit-learn library from Python to perform linear, lasso, ridge, SVR and Gaussian process regression. We use a 5-fold cross-validation approach to tune the hyperparameters of lasso, ridge, and SVR regression. We also use the 5-fold cross-validation approach to identify the appropriate kernel to be used by the Gaussian process regression model. For other hyperparameters, the Gaussian process regression model uses an internal limited-memory BFGS approach to tune its hyperparameters. After optimizing the hyperparameters, we incorporate a 10-fold cross-validation approach to train and evaluate each regression model. Results of applying each of the regression models on the structural *n*-gram encoded feature set is demonstrated in Table 7.2. Below we will discuss each of the regression techniques in detail.

Linear regression. Linear regression is a simple regression approach that works under the assumption that there is a linear relationship between features and the predicted value. Since this assumption does not hold for most real-world problems, this technique uses the least

squared method to find coefficients that minimizes the error between the approximated values and true values. The10-fold cross-validation evaluation of applying linear regression on our feature set resulted in a Mean Squared Error (MSE) of $3.03e^{24}$ and an R-squared of $1.19e\text{-}^{23}$. The MSE of applying the linear model on our feature set is unreasonably high. This is understandable since the high number of features in our dataset dramatically increases the complexity of the model, which in turn significantly increases the overfitting of our predictive model. To accommodate for this problem, we utilize ridge and lasso regression models that can reduce the overfitting and variance error by utilizing L1 and L2 regularization techniques.

Ridge regression. To reduce the variance error, ridge regression includes a penalty term in the optimization process that penalizes high coefficient values. This penalty term is equivalent to the sum of the squares of the coefficients' magnitudes timed by a penalty coefficient λ. Equation 7.1 demonstrates the optimization equation for ridge regression.

$$L_{Ridge}(\hat{\beta}) = \sum_{i=1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j=1}^{m} \hat{\beta}_j^{\,2} \qquad (7.1)$$

In the above equation, $\hat{\beta}$ is the vector of estimated coefficients for the regression model, n is the number of data points, m is the number of coefficients to be estimated, and λ is the penalty coefficient. We used the set [0.05, 0.1,0.5,1.0,10] to tune the value for λ and found λ=10 to be the best value for our regression task. Applying ridge regression on our dataset resulted in an MSE of 5.24 and an R-squared of 0.81. We can see that ridge regression considerably outperformed linear regression with respect to MSE, and R-squared.

Lasso regression. Similar to ridge regression, lasso regression utilizes a penalty term in its optimization process to regularize the regression coefficients. Unlike ridge regression, however, lasso regression includes a penalty equivalent to the sum of the coefficients'

magnitude multiplied by a penalty term λ. This allows the optimization process to push weights to zero if necessary. As a result, lasso regression can reduce overfitting as well as perform feature selection. Equation 7.2 displays the optimization equation for lasso regression.

$$L_{Lasso}(\hat{\beta}) = \sum_{i=1}^{n}(y_i - x_i'\hat{\beta})^2 + \lambda \sum_{j=1}^{m}|\hat{\beta}_j| \qquad (7.2)$$

We used the set [0.05, 0.1,0.5,1.0,10] to tune the value for λ and found λ=0.05 to be the best value for λ. Utilizing lasso regression resulted in an MSE of 6.30 and an R-squared of 0.77.

Support vector regression. Support vector regression is a variation of support vector machines that can predict real values as opposed to limited discrete values. Similar to SVMs, SVRs use kernels to transform data from a non-linearly separable space to a linearly separable space. For our regression task, we explored with linear, polynomial, and radial basis function kernels. For each kernel, we tuned the hyperparameters of C, epsilon, and gamma. For polynomial kernels, we also tuned the coef0, the intercept, and degree hyperparameters. Utilizing a 5-fold cross-validation approach, we found the polynomial kernel with a degree of four to be the best kernel for our dataset. Also, the grid search returned C=100, coef0=1, degree= 4, epsilon=1, gamma= 0.001 as the best parameters for this kernel. Incorporating the SVR model resulted in an MSE of 5.09 and an R-squared of 0.82. SVR performed better than both lasso and ridge in terms of MSE and R-squared. This could be due to the fact that kernel methods are specifically well-performed with datasets whose feature set is relatively large relative to the size of the dataset.

Gaussian process regression. Gaussian process (GP) regression provide an analytically tractable solution for regression problems with an infinite or uncountable set of considered

functions. It assigns a prior probability to all possible functions that we expect to observe. After observing the data, Gaussian process favors the functions that closely follow the observed data. Moreover, by adding Independent and Identically Distributed (IID) Gaussian noise to the prior, Gaussian process regression becomes capable of handling noise in outcome variables. Such a regression technique is extremely data-efficient and is appropriate for small datasets. Because our dataset is labeled by human subjects, it is highly prone to noise. Furthermore, since Gaussian process is a probabilistic approach, there is no limitation on the number of samples against the number of features.

We hypothesize that the Gaussian process regression should outperform other regression techniques due to its capability of handling noise and its propriety for our dataset. We use 5-fold cross-validation to choose the kernel for our Gaussian Regression processes. We split the data into five subsets and validate our model on each subset recursively. Then, we select the model with the least square error loss. For this process, we cross-validated our model for radial basis functions, rational quadratic, and Matern kernels and we found radial basis functions to perform the best on our dataset. To find the hyperparameter or priors parameters of the Gaussian process on the training set in each cross-validation round, we maximize the probability of observing data given hyperparameters of the process, known as marginal likelihood. In this work, we use a limited-memory BFGS optimization technique to maximize the log marginal likelihood conditioned on the length vectors and the noise level of the kernels.

Applying Gaussian process regression resulted in an MSE of 1.71, and an R-squared of 0.94. Gaussian process regression performed considerably better than other regression models. Not only is GP a kernel-based model similar to SVR, but by adding an additional

noise kernel it can also account for the potential noise in our dataset. As a result, it is expected that the GP model will outperform other models in our prediction task.

Table 7.2. *n*-gram feature set results. Results of applying regression models on the structural *n*-gram feature set.

Regression	MSE	R^2
Linear	$3.03e^{24}$	$1.19e^{-23}$
Ridge	5.24	0.81
Lasso	6.30	0.77
SVR	5.09	0.82
Gaussian Process	**1.71**	**0.94**

7.1.4 Feature Selection

As described earlier, lasso regression can eliminate less influential features by setting their weights to zero during the optimization process. We observed the coefficient values of lasso regression to find the most influential features corresponding to predicted grades. Overall, lasso picked 55 variables and eliminated the other 125 variables. "Repeat" and two available conditional blocks were shown to be the most important constructs in students' programming artifacts. Also, the appropriate implementation of conditionals and placement under the repeat block were shown to have great influence on predicted grades. Another important construct identified through this process is placement of a move block after an "If right container < left container" block.

On the other side of the spectrum, having multiple repeat blocks at the same hierarchical level is shown to be the most negatively correlated with grades. Furthermore,

having “If cart hits right wall” and “return to left wall” blocks at the same level as loop blocks is negatively correlated with the predicted grades. Having a repeat block inside of a conditional and having two consecutive swaps are other negatively correlated features in our dataset. The most important features chosen by lasso are aligned with the rubric items that are designed through the evidence-centered assessment design approach. This suggests that our feature set is effective in extracting important structural and semantic information from students’ programming artifacts.

7.1.5 Evaluating the Effectiveness of Structural *n*-grams

In order to better evaluate the effectiveness of our proposed feature set, we compared the results of utilizing the structural *n*-gram encoded feature set in the supervised learning with utilizing a bag of words-based baseline feature set. The results of applying ridge, lasso, SVR, and GP regression models on the baseline feature set are demonstrated in Table 7.3.

Table 7.3. Bag-of-words-based baseline feature set results. Results of applying different regression models on the bag-of-words-based baseline feature set.

Regression	MSE	R-squared
ridge	9.35	0.62
lasso	8.27	0.69
SVR	9.18	0.63
Gaussian processes	**3.07**	**0.88**

According to the results, applying ridge regression resulted in an MSE of 9.35 and an R-squared of 0.62. Furthermore, applying lasso regression resulted in an MSE of 8.27 and an R-

squared of 0.69. Applying SVR regression resulted in an MSE of 9.18 and an R-squared of 0.63. Finally, applying Gaussian process regression resulted in an MSE of 3.07 and an R-squared of 0.88.

Comparing the results of applying different regression models on the baseline feature set as well as the structural *n*-gram encoded feature set demonstrated the effectiveness of the structural *n*-gram encoded feature set in predicting programming artifacts' grades. It is worth mentioning that Gaussian process regression still worked considerably better on the baseline feature set compared to other regression models with regards to MSE and R-squared.

7.2 Discussion

Evidence-centered design (ECD) holds significant promise for guiding educators in designing mindful assignments for learners. Furthermore, it facilitates designing assessments and rubrics that can effectively assess the knowledge constructs intended for the assignments. In this work, we utilized an ECD approach for devising a semi-automatic assessment framework to assess students' programming artifacts algorithmic quality as demonstrated through their submitted solutions to a CT challenge within a game-based learning environment.

We used a rubric designed through an evidence-centered assessment design approach for grading students' submitted coding artifacts. Furthermore, we utilized a novel structural *n*-gram encoding scheme to extract important structural and semantic information from students' programming artifacts. Utilizing a systematic approach to rubric design reduces the disparity in human tutors' grading of the assignments which results in reduced noise in the training dataset. We then applied a variety of regression models to infer students' grades based on their programming artifacts encoded as structural *n*-grams. The results of our

prediction demonstrated the effectiveness of the *n*-gram encoded feature set in capturing important semantic and structural information in students' programming artifacts.

We conducted a 10-fold cross-validation to evaluate the results of applying different regression techniques on our feature set when inferring submitted programming artifact grades. All regression models performed better than our baseline model, linear regression. It is important to note that the subjective part of the rubric adds up to 5 points. As a result, ridge, lasso, SVR, and GP all did reasonably well in terms of mean squared error. While ridge, lasso, SVR, and GP all did reasonably well in terms of mean squared error and R-Squared, Gaussian process regression had the best mean squared error and R-Squared. This is not surprising since Gaussian process regression models can handle noisy data and are particularly efficient for datasets in which the number of features is particularly high relative to the number of data points.

We further utilized the feature selection capability of the lasso regression to identify the most influential features that are correlated with students' grades. The results revealed an alignment between the human tutor-designed rubric items and the system-selected important coding constructs. This further suggests that our feature set is effective in capturing important structural and semantic information from students' artifacts. Finally, we compared the results of applying the regression models on our feature set with the results of applying them on a bag-of-words-based baseline feature set. Utilizing the *n*-gram encoded artifacts resulted in considerably lower mean squared error and higher R-squared compared to utilizing the baseline feature set.

The semi-automatic assessment framework presented in this section can be generalized to assess any well-structured programming artifacts in learning environments that

presents students with well-structured programming assignments. Furthermore, the ECD approach can facilitate rubric design and assessment for non-expert CS teachers while providing them with automatic assessment of students' programming artifacts. A teacher dashboard can further be utilized to analyze and aggregate the results of the assessment to inform teachers about students' learning and the quality of their instruction accordingly.

7.3 Conclusion & Future Work

The increased interest in computer science education and students' need for guided practice of CS concepts have resulted in an increased need for accurate and effective automatic assessments. In this chapter, we presented a semi-automatic assessment framework for assessing programming artifacts' algorithmic quality following a blended hypothesis-driven approach.

We designed a rubric corresponding to a set if identified CS FKSAs for a bubble sort challenge and utilized the rubric for labeling our training dataset. Furthermore, we extracted structural information from students' programming artifacts in terms of *n*-gram encoded hierarchical and ordinal coding constructs. We applied a variety of regression models including Gaussian process regression to build models that can predict students' algorithmic quality grades based on their submitted programming artifacts. Our results showed that *n*-gram encoded features can effectively extract important semantic and structural information from programming artifacts. Furthermore, the results suggested that the Gaussian process regression can handle the existing noise in our dataset.

In the future, we plan to expand our assessment framework to assess more open-ended programming assignments. Furthermore, we plan to design a teacher dashboard that can integrate the process of assignment and rubric design while guiding teachers through the

ECD process and use the same dashboard to conduct automatic assessment of students' programming artifacts and provide teachers with appropriate feedback.

CHAPTER 8:

CONCLUSION

Recent years have seen a dramatic increase in interest in computer science and in the demand for computer science education. As a result, there is a rising need for quality and accessible computer science education. One challenge associated with offering computer science education at scale is providing CS learners with sufficient opportunities for guided practice. While CS learners require extensive practice to master CS and CT competencies, it is getting more challenging for educators to provide students with the sufficient support and scaffolding they need. The autonomous nature of emerging learning technologies such as game-based learning environments makes them a promising platform for offering students with unlimited opportunities for guided practice to improve their CS and CT skills. However, one important challenge is how to effectively assess students' CS and CT skills within these learning environments. Implementing effective and accurate automated assessment of students' CS FKSAs in the context of learning environments enables educators to provide students with sufficient guided practice while maintaining engagement.

Stealth assessment proposes a framework for unobtrusively assessing students' knowledge and competencies while interacting with the learning environment. Stealth assessment relies on accurate models utilizing evidence captured from student behavior traces and holds significant promise for effectively assessing students' knowledge and competencies during their interactions with game-based learning environments.

The focus of the presented research is on the design and development of a stealth assessment framework to unobtrusively and effectively assess students' CT proficiencies, in particular problem-solving strategies, and CS FKSAs. To this end, we focused on assessing

two different aspects of students' CS and CT competencies: in-game problem-solving strategies while solving a CT problem and focal CS competencies as evident from their submitted programming artifacts when solving a CT problem.

We present the results of devising and applying a novel strategy-based temporal analytics framework for assessing students' knowledge of binary number representation by utilizing their problem-solving strategies. We also presented results of designing and applying a rubric to assess students' focal computer science knowledge, abilities, and skills while solving an algorithmic challenge in the ENGAGE game-based learning environment using block-based programming. Finally, we presented a semi-automated assessment framework for assessing students' CS competencies based on their submitted programming artifacts and presented the results of applying this framework on data collected from students' interactions with an algorithmic challenge in ENGAGE.

8.1 Summary

We apply stealth assessment on two different CT-focused challenges within the ENGAGE game-based learning environment. The first problem focuses on building binary representation of a decimal number. To assess students' CT competencies for this challenge, we have introduced a novel strategy-based temporal analytics framework for stealth assessment that utilizes n-gram representations of students' interaction with a game-based learning environment to cluster them based on their problem-solving strategies. After generating clusters of students' problem-solving strategies, we evaluated the predictive accuracy of cluster membership derived directly from dynamic student in-game behaviors. The evaluation revealed that models using strategy-based temporal analytics together with pre-test data significantly outperformed models using pre-test data alone and enhanced

stealth assessment in game-based learning. Furthermore, the strategy-based temporal analytics framework utilizing LSTMs outperformed baseline models that did not capture the temporal dependencies of strategy use.

The second problem focuses on assessing students' CS FKSAs based on their submitted solutions to a bubble-sort algorithm challenge. We presented a preliminary assessment framework for assessing students' CS proficiencies in three facets of essential CS constructs, testing and refining artifacts, and problem-solving efficiency. The preliminary results of applying this assessment framework on a dataset collected from students' interactions with the ENGAGE game-based learning environment demonstrated significant correlations between students' proficiency of essential CS constructs and their in-game behavior. Furthermore, it demonstrated that students with higher algorithmic scores tend to make significantly more novel, positive, and small-scale changes. We refined this assessment based on the preliminary results and applied the revised assessment to the data collected from students' individual gameplay with the bubble sort algorithm challenge. The results of the second round of assessment confirmed the patterns identified in the first study.

We then extracted structural features from students' block-based programming artifacts to capture the essential structural information of the programming artifacts. A core challenge in applying reliable and effective learning analytics is deriving significant features from fine-grained learning interaction data. We transformed students' block-based programming artifacts into their corresponding abstract syntax trees and encoded them in terms of hierarchical and ordinal *n*-gram features to capture the essential structural information of the artifacts. These features can also capture code redundancies, partially correct code, and partially incorrect code when compared with correct solution models.

After building a training dataset and extracting structural features from students' programming artifacts, we applied a variety of regression models to the training dataset to evaluate their effectiveness in inferring the overall algorithmic quality of programming artifacts. We applied linear regression as our baseline model and compared its result with the results of applying ridge, lasso, SVR, and Gaussian process regression. Gaussian process regression is appropriate for making inferences on noisy datasets and datasets that have a relatively large feature space compared to the number of observations in the dataset. While all non-baseline techniques including lasso, ridge, SVR, and Gaussian process performed reasonably well, Gaussian process regression model outperformed all other techniques. This could be due to the fact that the Gaussian process regression model can consider and handle the noise within our training dataset.

Finally, to evaluate the effectiveness of our structural *n*-gram encoded feature set we performed a feature selection using lasso regression. The results demonstrated an alignment between the most important features selected by lasso and the rubric items designed by CS experts. Furthermore, we applied the regression models on a bag-of-words-based feature set and compared its results with results obtained from utilizing the *n*-gram encoded feature set. The models that utilized the *n*-gram encoded feature set outperformed the models that incorporated the bag-of-word-based feature set with regard to MSE and R-squared.

8.2 Hypotheses Revisited

In this dissertation, we aimed to investigate the following thesis statement:

> *Stealth assessment is an effective approach for unobtrusive, effective and accurate evaluation of students' CT proficiencies and CS focal knowledge, skills, and abilities in game-based learning environments.*

As part of this evaluation, this work explored three primary hypotheses and produced the following results:

- **H1.** Student' computational thinking proficiencies can be effectively inferred by utilizing their problem-solving strategies identified from their interactions with a computational thinking problem in a well-structured game-based learning environment.
    - **H1.1** Clustering students' patterns of interaction within a well-structured educational game while solving a computational thinking problem can identify their problem-solving strategies.

      Clustering students' patterns of interactions with the CT challenge under investigation resulted in three distinct problem-solving patterns among them. The identified problem-solving strategies were consistent across consecutive tasks within the game. These findings provide a meaningful portrait of students' problem-solving strategies in game-based learning environments. Thus, this hypothesis is confirmed.

    - **H1.2** Students' problem-solving strategies over similar tasks can be utilized to infer their knowledge of the corresponding computational thinking concepts.

      Several well-established classification techniques including random forest, SVM, and LSTM that utilized problem-solving strategies in their feature set considerably outperformed the majority class-based baseline model with regard to prediction power when predicting students' post-test performance. As a result, this hypothesis is accepted.

H1.3 Utilizing the temporal dependency among students' choice of problem-solving strategies over consecutive, similar tasks can improve the prediction accuracy of the model in regard to students' post-test performance.

An LSTM-based network was incorporated to capture the temporal dependency among students' choice of problem-solving strategy over time. The LSTM model considerably outperformed other prediction models that did not take this temporal dependency into account. These findings point to possible means of intervention based on student-identified problem-solving strategies to improve their learning. Furthermore, these findings confirm the proposed hypothesis.

- **H2.** Students' CS focal knowledge, skills, and abilities can be effectively assessed based on their programming trajectories.
    - **H2.1** Students' knowledge of essential CS constructs can be assessed based on their block-based programming artifacts.

      Following an evidence-centered design approach, an assessment was designed to assess students' knowledge of CS essential constructs as demonstrated through their submitted programming artifacts for a bubble sort algorithm. The results demonstrated that to design and implement appropriate and generalizable algorithms, students need to have a strong foundation in CS focal knowledge constructs. These findings provide valuable insight about effective CS pedagogical strategies. This hypothesis is accepted.

- **H2.2** Students' common testing and refining approaches while solving a computational problem using block-based programming can be assessed using their programming trajectories.

  Students' common testing and refining approaches were investigated under three aspects: novelty, positivity, and scale. According to our findings, students who have higher algorithmic scores tend to make significantly more novel, positive and small-scale changes. These results were confirmed both through the preliminary and the main study. This hypothesis is confirmed.

- **H2.3** The effectiveness of students' testing and refining approaches can be assessed by exploring the relationship between their knowledge of essential CS constructs, in-game behavior, and their testing and refining patterns.

  Normalized total number of submissions, percentage of repetitive submissions, and percentage of open-door attempts were calculated as a measure of problem-solving efficiency. According to our results, students with a higher understanding of algorithms had a significantly lower number of submissions, lower percentage of repetitive submissions, and fewer unsuccessful open-door attempts. These results suggest that students with higher algorithmic understanding solve CT problems more efficiently. This hypothesis is accepted.

- **H3.** Utilizing a supervised learning approach, programming artifacts' algorithmic quality can be automatically assessed based on their submitted block-based programming artifacts when solving an algorithmic problem in a well-structured game-based learning environment.

- **H3.1** Important structural and ordinal information within students' block-based programming artifacts can be preserved by performing appropriate feature engineering.

  A structural n-gram encoding approach was incorporated to extract hierarchical and ordinal structures from submitted programming artifacts. Inference models that utilized n-gram encoded features considerably outperformed other inference models that utilized simple block counts as their feature set. Furthermore, a feature selection process utilizing lasso regression demonstrated strong alignments between most influential features with regard to predicated grades and specified rubric items. These results demonstrate the effectiveness of our feature engineering approach in extracting structural and semantic information from students' submitted artifact. Thus, the hypothesis is accepted.

- **H3.2** Students' algorithmic proficiency can be accurately inferred from their submitted programming artifacts encoded as structural *n*-grams.

  A variety of well-established regression models were incorporated to infer students' algorithmic proficiency from the encoded structural n-grams. Applying ridge, lasso, SVR, and Gaussian process regression models resulted in reasonably close prediction of grades. Furthermore, ridge, lasso, SVR, and Gaussian process regression models outperformed linear regression models in terms of the mean squared error and R-squared in a 10-fold cross-validation evaluation approach. These results confirm the hypothesis.

- **H3.3** The accuracy of the predictive model can be improved by considering the noise in the graded assignments.

  A Gaussian process regression was incorporated to account for the noise within the training dataset. The Gaussian process regression model outperformed ridge, lasso, and SVR regression models in terms of the mean squared error and R-squared in a 10-fold cross-validation. Our results suggest that it is important to account for noise in the dataset in order to achieve accurate results. However, to fully confirm this hypothesis, a comparative study is required to compare the results of utilizing the GP model on a noise-free baseline dataset with the results of applying it on the current noisy dataset.

8.3 Limitations

This dissertation presents a hypothesis-driven stealth assessment framework for assessing students' CT proficiencies and CS FKSAs within game-based learning environments. While this framework is designed to accurately and effectively assess students' CT proficiencies and CS FKSAs, there are limitations when considering generalization of this assessment to other similar learning environments. One important limitation of this work is that it only evaluates the framework in the context of ENGAGE which is quite different from other learning environments. It is important to investigate how our results will generalize to other learning systems, especially those that are less structured and present student with more open-ended challenges. Furthermore, we have only evaluated this framework on data collected from a specific age range (middle grade students). It would be worthwhile to evaluate how these results generalize to other age groups.

Two out of four studies presented here utilized data collected from students' paired gameplay. On the other hand, the pre- post-test assessments used for these two studies were administrated individually. This resulted in some disparities between inferences made from interaction patterns and the connections made to the pre- post-test performances. For the other two studies, we addressed this issue by collecting a new dataset from students' individual gameplay.

Another limitation of this study is the alignment between game content and assessments through the administered pre- post-tests. While the first study utilized a pre-post-assessment that was perfectly aligned with the game content, the pre- post-assessments utilized for the last three studies were not sufficiently aligned. As a result, we did not observe significant correlations between students' CS competencies as measured through their interaction patterns and though their pre- post-test performances. We expect that by utilizing a more aligned pre- post-test we can better correlate students in-game and outside of the game performances.

Furthermore, the stealth assessment framework presented in this dissertation is mostly suitable for well-structured problems where the steps can be chosen from a limited pool of actions. Nevertheless, this is an effective first step in systematic design of assessments for evaluating students' CS FKSAs based on their interaction patterns with game-based learning environments. We hope that in the future we can extend this approach to assess more open-ended problems.

8.4 Future Directions

There are several promising directions for future work. A key area of future work to explore is to investigate how well the presented framework generalizes beyond the presented

findings. The stealth assessment framework presented in this dissertation is applied only on data collected from two challenges within the ENGAGE game. It would be interesting to investigate the results of applying this framework on similar levels in the game and on related problems in other learning environments. Another interesting future direction is to expand this framework to assess more open-ended problems. Currently, this framework is applicable on problems that are solvable by a limited number of actions. It would be interesting to extend this framework to accommodate for more choices while building a solution.

Another area of extension is to continue training current models on larger populations of students. The second round of data collection is an ongoing process and the subpopulation used for this dissertation is a small portion of the data being collected. Applying the proposed stealth assessment on a larger dataset should yield more accurate and reliable results. Furthermore, the ENGAGE game is designed for middle grade students, and as a result, the data collected and utilized for this study focuses on middle grade students (11-13 years old). It would be interesting to investigate the effectiveness of our stealth assessment framework on other age groups such as high school and college CS students.

Early prediction of students' performance is another interesting area of exploration. Effective early prediction of students' learning enables the system and the instructor to provide students with effective scaffolding in time to improve their learning. As an example, the LSTM network used in Chapter 4 lends itself to predicting students' learning in varying number of steps. It is worth exploring what is the minimum number of steps for reliably predicting students' post-test performances. Another interesting exploration is to investigate the relationship between students' learning and their CS self-efficacy and attitude. We have

administered CS self-efficacy and attitude questionnaires before and after students' gameplay. The results of these assessment can be used in our feature set to explore their effectiveness in improving the prediction power of the learning models.

Finally, an effective assessment of students' knowledge and skills serves as the foundation for adaptive scaffolding such as adaptive hints, feedback, and tailored next-problems. Hence, a natural and promising next step would be to use the results of the assessment to provide students with automatic and adaptive scaffolding. This is important since learning computer science and computational thinking requires persistence and guided practice. As the interest in computer science and therefore the number of CS learners are increasing, it is becoming more challenging for teachers to provide each student with the individualized feedback they need. Building adaptive systems that can effectively assess students' knowledge and skills and provide them with appropriate scaffolding is significantly helpful to computer science education.

REFERENCES

2018 State of Computer Science Education Policy and Implementation Advocacy Coalition. (2018).

Akram, B., Min, W., Wibe, E., Mott, B., Boyer, K., & Lester, J. (2018). Improving Stealth Assessment in Game-based Learning with LSTM-based Analytics. In *Proceedings of the Eleventh International Conference on Educational Data Mining* (pp. 208–218).

Ala-Mutka, K. M. (2005). A Survey of Automated Assessment Approaches for Programming Assignments. *Computer Science Education*, *15*(2), 83–102.

Blikstein, P., Worsley, M., Piech, C., Sahami, M., Cooper, S., & Koller, D. (2014). Programming Pluralism: Using Learning Analytics to Detect Patterns in the Learning of Computer Programming. *Journal of the Learning Sciences*, *23*(4), 561–599.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Brennan, K., & Resnick, M. (2012). *New frameworks for studying and assessing the development of computational thinking*. In *Proceedings of the 2012 annual meeting of the American Educational Research Association*. Vancouver, Canada.

Brusilovsky, P., & Millán, E. (2007). User models for adaptive hypermedia and adaptive educational systems. In *The adaptive web* (pp. 3–53). Berlin, Heidelberg: Springer.

Buffum, P., Frankosky, M., Boyer, K., Wiebe, E., Mott, B., & Lester, J. (2016). Collaboration and Gender Equity in Game-Based Learning for Middle School Computer Science. *Computing in Science & Engineering*, *18*(2), 18–28.

Cheang, B., Kurnia, A., Lim, A., & Oon, W. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education*, *41*(2), 121–131.

Chen, G., Gully, S., & Eden, D. (2001). Validation of a new general self-efficacy scale. *Organizational Research Methods*, *4*(1), 62–83.

Chen, P.(2004). An Automated Feedback System for Computer Organization Projects. *IEEE Transactions on Education*, *47*(2), 232–240.

Chollet, F. (2015). Keras. Retrieved from https://github.com/keras-team/keras

Clark, D., Tanner-Smith, E., & Killingsworth, S. (2016). Digital Games, Design, and Learning: A Systematic Review and Meta-Analysis. *Review of Educational Research*, *86*(1), 79–122.

Clarke, E.(2008). The Birth of Model Checking. In *25 Years of Model Checking* (pp. 1–26). Berlin, Heidelberg: Springer.

Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, *20*(1), 37–46.

Cordova, D., & Lepper, M.(1996). Intrinsic motivation and the process of learning: Beneficial effects of contextualization, personalization, and choice. *Journal of Educational Psychology*, *88*(4), 715–730.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, *20*(3), 273–297.

Cousot, P., & Cousot, R. (1977). Abstract interpretation. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages* (pp. 238–252).

d'Aquin, M., & Jay, N. (2013). Interpreting data mining results with linked data for learning analytics: motivation, case study and directions. In *Proceedings of the Third International Conference on Learning Analytics and Knowledge* (pp. 155–164).

Dempster, A., Laird, N., & Rubin, D.(1977). Maximum likelihood from incomplete data via

the EM algorithm. *Journal of the Royal Statistical Society.Series B (Methodological)*, 1–38.

Diana, N., Eagle, M., Stamper, J., Grover, S., Bienkowski, M., & Basu, S. (2017a). An Instructor Dashboard for Real-Time Analytics in Interactive Programming Assignments. In *International Learning Analytics & Knowledge Conference* (pp. 272–279).

Diana, N., Eagle, M., Stamper, J., Grover, S., Bienkowski, M., & Basu, S. (2017b). Data-Driven Generation of Rubric Parameters from an Educational Programming Environment. In *International Conference on Artificial Intelligence in Education* (pp. 490–493). Springer, Cham.

Eagle, M., & Barnes, T. (2014). Exploring differences in problem solving with data-driven approach maps. In *Proceedings of the 7th International Conference on Educational Data Mining* (pp. 76–83).

Falakmasir, M., González-Brenes, J., Gordon, G., & DiCerbo, K. (2016). A data-driven approach for inferring student proficiency from game activity logs. In *Proceedings of the Third ACM Conference on Learning@ Scale* (pp. 341–349).

Fields, D., Giang, M., & Kafai, Y. (2014). Programming in the wild: trends in youth computational participation in the online scratch community. In *Proceedings of the 9th workshop in primary and secondary computing education* (pp. 2–11).

Grover, S., & Basu, S. (2017). Measuring Student Learning in Introductory Block-Based Programming. In *Proceedings of the forty-eighth ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 267–272).

Grover, S., Basu, S., Bienkowski, M., Eagle, M., Diana, N., & Stamper, J. (2017). A Framework for Using Hypothesis-Driven Approaches to Support Data-Driven Learning

Analytics in Measuring Computational Thinking in Block-Based Programming Environments. *ACM Transactions on Computing Education*, *17*(3), 1-25.

Grover, S., Bienkowski, M., Niekrasz, J., & Hauswirth, M. (2016). Assessing Problem-Solving Process At Scale. In *Proceedings of the Third ACM Conference on Learning @ Scale* (pp. 245–248).

Hage, J., Rademaker, P., & Van Vugt, N. (2010). *A comparison of plagiarism detection tools*. Utrecht, The Netherlands.

Hansen, A., Dwyer, H., Iveland, A., Talesfore, M., Wright, L., Harlow, D., & Franklin, D. (2017). Assessing Children's Understanding of the Work of Computer Scientists: The Draw-a-Computer-Scientist Test. In *Proceedings of the 48th ACM SIGCSE technical symposium on computer science education* (pp. 279–284).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of Recent Systems for Automatic Assessment of Programming Assignments. In *Proceedings of the 10th Koli calling international conference on computing education research* (pp. 86–93).

Jackson, D., & Usher, M. (1997). Grading student programs using ASSYST. In *Proceedings of the twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (pp. 335–339).

Jackson, G., & McNamara, D. (2013). Motivation and performance in a game-based intelligent tutoring system. *Journal of Educational Psychology*, *105*(4), 1036–1049.

Joy, M., Griffiths, N., & Boyatt, R. (2005). The boss online submission and assessment system. *Journal on Educational Resources in Computing*, *5*(2), 1–28.

K-12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org.

Käser, T., Hallinen, N. R., & Schwartz, D. (2017). Modeling exploration strategies to predict student performance within a learning environment and beyond. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference* (pp. 31–40).

Kebritchi, M., Hirumi, A., & Bai, H. (2010). The effects of modern mathematics computer games on mathematics achievement and class motivation. *Computers & Education*, *55*(2), 427–443.

Kerr, D., & Chung, G. (2012). Identifying key features of student performance in educational video games and simulations through cluster analysis. *Journal of Educational Data Mining*, *4*(1), 144–182.

Kim, Y., Almond, R., & Shute, V. J. (2016). Applying evidence-centered design for the development of game-based assessments in physics playground. *International Journal of Testing*, *16*(2), 142–163.

King, J. (1976). Symbolic Execution and Program Testing. *Communications of the ACM*, *19*(7), 385–394.

Koh, K., Basawapatna, A., Bennett, V., & Repenning, A. (2010). Towards the Automatic Recognition of Computational Thinking for Adaptive Visual Language Learning. In *38th IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 59–66).

Koh, K., Basawapatna, A., Nickerson, H., & Repenning, A. (2014). Real Time Assessment of Computational Thinking. In *IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 49–52).

Lajis, A., Arfah Baharudin, S., Kadir, D., Ralim, N.., Nasir, H., & Aziz, N. (2018). A Review

of Techniques in Automatic Programming Assessment for Practical Skill Test. *Journal of Telecommunication, Electronic and Computer Engineering* , *10*(2), 109–113.

Lester, J., Ha, E., Lee, S., Mott, B., Rowe, J., & Sabourin, J. (2013). Serious games get smart: Intelligent game-based learning environments. *AI Magazine*, *34*(4), 31–45.

Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. *R News*, *2*(3), 18–22.

Malkiewich, L., Baker, R. , Shute, V., Kai, S., & Paquette, L. (2016). Classifying behavior to elucidate elegant problem solving in an educational game. In *International Educational Data Mining Society* (pp. 448–453).

Martins, V., Fonte, D., Henriques, P., & da Cruz, D. (2014). Plagiarism detection: A tool survey and comparison. *Languages, Applications and Technologies*.

Meerbaum-Salant, O., Armoni, M., & Ben-Ario, M. (2013). Learning computer science concepts with scratch. *Computer Science Education*, *23*(3), 239–364.

Meyer, D., Dimitriadou, E., Hornik, K., Weingessel, A., & Leisch, F. (2015). e1071: Misc Functions of the Department of Statistics (e1071), TU Wien, 2014. *R Package Version*, 4.

Min, W., Frankosky, M., Mott, B. , Rowe, J. , Wiebe, E., Boyer, K., & Lester, J. (2015). DeepStealth: Leveraging Deep Learning Models for Stealth Assessment in Game-based Learning Environments. In *International Conference on Artificial Intelligence in Education* (pp. 277–286).

Min, W., Frankosky, M., Mott, B., Wiebe, E., Boyer, K., & Lester, J. (2017). Inducing Stealth Assessors from Game Interaction Data. In *International Conference on Artificial Intelligence in Education* (pp. 212–223).

Min, W., Mott, B., Rowe, J., Liu, B., & Lester, J. (2016). Player Goal Recognition in Open-World Digital Games with Long Short-Term Memory Networks. In *International Joint Conference on Artificial Intelligence* (pp. 2590–2596).

Mislevy, R. (2013). Evidence-Centered Design for Simulation-Based Assessment. *Military Medicine*, *178*(10S), 107–114.

Mislevy, R., & Haertel, G. (2007). Implications of Evidence-Centered Design for Educational Testing. *Educational Measurement: Issues and Practice*, *25*(4), 6–20.

Mislevy, R., Steinberg, L., & Almond, R. (2003). On the Structure of Educational Assessments. *Measurement: Interdisciplinary Research & Perspective*, *1*(1), 3–62.

Nelson, B. , Kim, Y., Foshee, C., & Slack, K. (2014). Visual signaling in virtual world-based assessments: The SAVE Science project. *Information Sciences*, *264*, 32–40.

Park, K., Mott, B., Min, W., Boyer, K., Wiebe, E., & Lester, J. (2019). Generating Educational Game Levels with Multistep Deep Convolutional Generative Adversarial Networks. In *To appear in: 33rd Proceedings of the International IEEE Conference on Games*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Dubourg, V. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Pratt, D. (2003). *The making of CourseMaker, a web-based shell program which can be set up by the teacher to run online courses*.

Price, T., Dong, Y., & Lipovac, D. (2017). iSnap: Towards Intelligent Tutoring in Novice Programming Environments. In *Proceedings of the 48th ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 483–488).

Quellmalz, E., Timms, M., Silberglitt, M., & Buckley, B. (2012). Science assessments for all: Integrating science simulations into balanced state science assessment systems. *Journal of Research in Science Teaching*, *49*(3), 363–393.

Quigley, D., Ostwald, J., & Sumner, T. (2017). Scientific modeling: using learning analytics to examine student practices and classroom variation. In *Proceedings of the Seventh International Learning Analytics & Knowledge Conference* (pp. 329–338).

Reek, K.(1989). The TRY system -or- how to avoid testing student programs. *ACM SIGCSE Bulletin*, *21*(1), 112–116.

Rowe, E., Baker, R., Asbell-Clarke, J., Kasman, E., & Hawkins, W. (2014). Building automated detectors of gameplay strategies to measure implicit science learning. In *Poster presented at the 7th annual meeting of the international educational data mining society*. London, England.

Rowe, J., Shores, L., Mott, B., & Lester, J. (2011). Integrating learning, problem solving, and engagement in narrative-centered learning environments. *International Journal of Artificial Intelligence in Education*, *21*(1–2), 115–133.

Sabourin, J., & Lester, J. (2014). Affect and Engagement in Game-BasedLearning Environments. *IEEE Transactions on Affective Computing*, *5*(1), 45–56.

Sahebi, S., Huang, Y., & Brusilovsky, P. (2014). Predicting student performance in solving parameterized exercises. In *International Conference on Intelligent Tutoring Systems* (pp. 496–503).

Scrucca, L., Murphy, T., Raftery, A., & Fop, M. (2016). mclust 5: clustering, classification and density estimation using Gaussian finite mixture models. *The R Journal*, *8*(1), 289–317.

Seiter, L., & Foreman, B. (2013). Modeling the Learning Progressions of Computational Thinking of Primary Grade Students. In *Proceedings of the ninth annual international ACM conference on International computing education research* (pp. 59–66).

Sengupta, P., Kinnebrew, J., Basu, S., Biswas, G., Clark, D., Sengupta, P., Biswas, G. (2013). Integrating computational thinking with K-12 science education using agent-based computation: A theoretical framework. *Education and Information Technologies*, *18*(2), 351–380.

Shamsi, F., &, Elangar, E. (2012). An intelligent assessment tool for students' Java submissions in introductory programming courses. *Journal of Intelligent learning systems and applications*, *4*(1), 59–69.

Shute, V. (2011). Stealth assessment in computer-based games to support learning. *Computer Games and Instruction*, *55*(2), 503–524.

Shute, V., & Ventura, M. (2013). *Measuring and supporting learning in games: Stealth assessment*. Cambridge, USA: The MIT Press.

Shute, V., Ventura, M., & Kim, Y. (2013). Assessment and Learning of Qualitative Physics in Newton's Playground. *The Journal of Educational Research*, *106*(6), 423–430.

Shute, V., Wang, L., Greiff, S., Zhao, W., & Moore, G. (2016). Measuring problem solving skills via stealth assessment in an engaging video game. *Computers in Human Behavior*, *63*, 106–117.

Wang, T., Su, X., Wang, Y., & Ma, P. (2007). Semantic similarity-based grading of student programs. *Information and Software Technology*, *49*(2), 99–107.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics and Science Classrooms.

Journal of Science Education and Technology, *25*(1), 127–147.

Wiebe, E., London, J., Aksit, O., Mott, B., Boyer, K., Lester, J. (2019). Development of a Lean Computational Thinking Abilities Assessment for Middle Grades Students. In *In Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 456–461).

Wiebe, E., Williams, L., Yang, K., & Miller, C. (2003). Computer science attitude survey. *Computer Science*, *14*(25), 0–86.

Wouters, P., Van Nimwegen, C., Van Oostendorp, H., & Van Der Spek, E. (2013). A meta-analysis of the cognitive and motivational effects of serious games. *Journal of Educational Psychology*, *105*(2), 249–265.

Xie, B., & Hal, A. (2016). Skill progression in MIT app inventor. In *IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 213–217).

Xing, Z., Pei, J., & Keogh, E. (2010). A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter*, *12*(1), 40–48.

Zhi, R., Price, T., Lytle, N., Dong, Y., & Barnes, T. (2018). Reducing the State Space of Programming Problems through Data-Driven Feature Detection. In *Educational Data Mining Workshop*.