2016

Tutor-Complete: An Educational Game and Intelligent Tutoring System for Languages and Automata

Katherine Kjeer
kkjeer@wellesley.edu

WELLESLEY COLLEGE

Tutor-Complete: An Educational Game and Intelligent Tutoring System for Languages and Automata

by

Katherine Kjeer

Submitted in Partial Fulfillment of the
Prerequisite for Honors

in the
Computer Science Department

May 2016

“Play is the highest form of research.”

Albert Einstein

WELLESLEY COLLEGE

Abstract

Computer Science Department

Bachelor of Arts

by Katherine Kjeer

Educational games and Intelligent Tutoring Systems have been shown to improve student learning outcomes by increasing engagement and providing individualized instruction. However, while introductory programming students frequently benefit from such systems, students in upper-level theoretical courses such as CS 235 (Languages and Automata) have dense textbooks and dry mathematical readings as their primary or only resources. Tutor-Complete aims to fill this gap by presenting two fundamental CS 235 concepts in a game environment. In the first activity, students construct Deterministic Finite-State Automata in order to guide their character across a landscape. In the second activity, students build proofs using the pumping lemma to defeat the "villain" character. Tutor-Complete also fosters a peer-learning environment by encouraging students to explain concepts to each other and providing hints based on past student work. Finally, Tutor-Complete uses Bayesian Knowledge Tracing to model students' knowledge and tailor the learning experience accordingly.

Acknowledgements

With sincere appreciation, I thank my advisor, Eni Mustafaraj, for her incredible guidance and support of this thesis. Tutor-Complete would not exist without her encouragement and expertise. I also thank my committee members, Lyn Turbak and Orit Shaer, for their valuable feedback and generosity in this process.

Thank you to all the Computer Science professors that I have studied or worked with (Lyn Turbak, Sohie Lee, Taxis Metaxas, Stella Kakavouli, Randy Shull, Jean Herbst, Scott Anderson, Rhys Price Jones, Ben Wood, Eni Mustafaraj, Orit Shaer, and Brian Tjaden) for sharing their wealth of knowledge about this amazing field.

I am grateful to the Computer Science students that participated in user studies for this thesis, whose thoughtful feedback has strengthened Tutor-Complete and generated ideas for future work.

Finally, I am very thankful to my family, friends, and professors for their support, encouragement, and patience in the process of building Tutor-Complete and writing this thesis.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

DFA	**D**eterministic **F**inite-State **A**utomaton
ITS	**I**ntelligent **T**utoring **S**ystem
BKT	**B**ayesian **K**nowledge **T**racing
MDP	**M**arkov **D**ecision **P**rocess

To Emily, my sister and best friend

Chapter 1

Introduction

"Tell me and I forget. Teach me and I remember. Involve me and I learn."

Benjamin Franklin

Tutor-Complete is both an educational game and an Intelligent Tutoring System (ITS). This chapter introduces the background of both aspects of Tutor-Complete, the motivation for Tutor-Complete's activities, and the educational goals that drove the design and development processes.

1.1 Educational Games

While games have a long history of entertainment applications, they are quickly expanding into education. Researchers name engagement, motivation, improved academic performance, and hightened retention as just a few of the benefits of educational games [9–11]. According to Ibáñez et. al. [9],

> Students who are engaged are attracted to their work, persist in their academic activities despite challenges and obstacles, and take visible delight in accomplishing them . . . Among the most successful approaches (for fostering student engagement) are digital games because they can potentially create engaging learning experiences for students when coupled with effective pedagogy.

Moreover, educational games have been identified as being particularly successful in computer science, due to the motivational nature of games' graphics, scenarios, and interactivity [12].

1.2 Intelligent Tutoring Systems

What is an Intelligent Tutoring System?

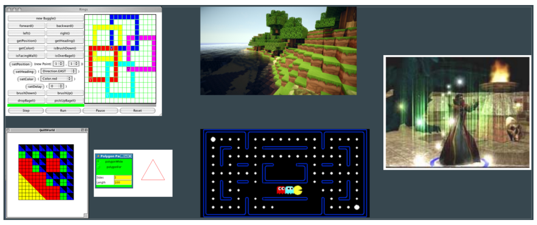
An Intelligent Tutoring System (ITS) is a computer system intended to help students master a given set of material. ITSs are distinct from other types of computer-aided instruction systems since they provide real-time, individualized instruction to students. Classical ITSs consist of four components: [13]

1. **Problem Solving Environment:** Students engage in problem-solving activities, e.g. constructing a DFA or a pumping lemma proof
2. **Domain Knowledge Module:** The system maintains a representation of the material, e.g. a framework for generating regular languages or a structure for pumping lemma proofs
3. **Student Model:** The system models students' mastery of a particular skill in the domain, e.g. choosing the transitions in a DFA or the counterexample string in a pumping lemma proof
4. **Pedagogical Module:** The system determines next actions based on the student model, e.g. promoting students to the next level after the DFA and pumping lemma skills are mastered

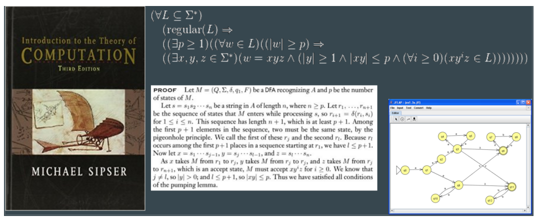
In particular, the student model is important to the tutoring capabilities of a computer-based educational system. Tutor-Complete uses Bayesian Knowledge Tracing (BKT) to implement its student model. The details of the BKT student modeling process are discussed in Chapter 5.

Why use Intelligent Tutoring Systems?

Numerous studies in education have demonstrated the effectiveness of human tutors [14, 15]. However, the use of human tutors is limited by time, money, tutor availability, and other resources. Thus, ITSs aim to "capture the effective behaviors of human tutors, thereby creating an optimal educational tool" [15]. The four components of ITSs described above have been successful in providing individualized instruction in a variety of domains, including algebra, geometry, basic physics, and introductory programming [13, 16, 17].



(a) CS 111 Resources



$$(\forall L \subseteq \Sigma^*)\;(\text{regular}(L) \Rightarrow ((\exists p \geq 1)((\forall w \in L)((|w| \geq p) \Rightarrow ((\exists x, y, z \in \Sigma^*)(w = xyz \wedge (|y| \geq 1 \wedge |xy| \leq p \wedge (\forall i \geq 0)(xy^iz \in L))))))))$$

PROOF Let $M = (Q, \Sigma, \delta, q_1, F)$ be a DFA recognizing A and p be the number of states of M.

Let $s = s_1s_2 \cdots s_n$ be a string in A of length n, where $n \geq p$. Let $r_1, \ldots, r_{n+1}$ be the sequence of states that M enters while processing s, so $r_{i+1} = \delta(r_i, s_i)$ for $1 \leq i \leq n$. This sequence has length $n + 1$, which is at least $p + 1$. Among the first $p + 1$ elements in the sequence, two must be the same state, by the pigeonhole principle. We call the first of these r_j and the second r_l. Because r_l occurs among the first $p+1$ places in a sequence starting at r_1, we have $l \leq p+1$. Now let $x = s_1 \cdots s_{j-1}$, $y = s_j \cdots s_{l-1}$, and $z = s_l \cdots s_n$.

As x takes M from r_1 to r_j, y takes M from r_j to r_j, and z takes M from r_j to r_{n+1}, which is an accept state, M must accept xy^iz for $i \geq 0$. We know that $j \neq l$, so $|y| > 0$; and $l \leq p+1$, so $|xy| \leq p$. Thus we have satisfied all conditions of the pumping lemma.

(b) CS 235 Resources

FIGURE 1.1: Contrasting CS 111 and CS 235 Resorces

1.3 Motivation

ITSs in their most common domains, listed in Section 1.2, generally emphasize applications and specific problem-solving skills rather than abstract theoretical concepts. In addition, most ITSs are directed at elementary, middle, or high school students rather than college students [13, 16]. Educational games are also, for the most part, made for introductory rather than advanced courses (see Chapter 2). As a result, students in introductory courses such as CS 111 (Introduction to Programming) benefit from educational microworlds and games like BuggleWorld, PacMan, and Minecraft. However, CS 235 (Languages and Automata) students' primary resources are textbooks, dense mathematical readings, and simulation programs without gaming or tutoring components (see Figure 1.1).

Tutor-Complete attempts to fill this gap by providing CS 235 students with an engaging game and tutoring system that presents the abstract, theoretical course material in a concrete, understandable way. Tutor-Complete includes activities for two fundamental CS 235 concepts: Deterministic Finite-State Automata (DFAs) and the pumping lemma for regular languages. In the DFA activities, students build structures to guide the "hero" character across a terrain to collect a reward. The pumping lemma activities are structured in an adversarial format, with a back-and-forth style gameplay between the computerized "villain" character and the student-controlled hero character. This design was motivated by observing student difficulties with proof by contradiction (the technique used in pumping lemma proofs). One of the goals of this activity is to clarify which decisions in the proof are the responsibility of the student and which outcomes are dictated by the nature of the problem. A complete discussion of Tutor-Complete's gameplay and design can be found in Chapter 3.

1.4 Educational Goals

The design and implementation of Tutor-Complete are centered on the following educational goals:

1. **Educational Gaming:** Tutor-Complete will increase student engagement and motivation by presenting the DFA and pumping lemma concepts in a concrete and enjoyable manner.
2. **Peer Instruction:** Tutor-Complete will leverage the collective knowledge of its users in order to help students learn from each other and solidify their individual understanding by explaining concepts to other students.
3. **Tutoring:** Tutor-Complete will offer hints and feedback to students as they work on activities, and model students' knowledge in order to adapt the learning experience to each student.

To support these goals, Tutor-Complete includes three game levels with activities focusing on DFAs and the pumping lemma. Students write explanations and use a "fave"-based rating system to foster a peer-learning environment. Hints in the DFA activities are based on the actions of past students, and Bayesian Knowledge Tracing is used to track students' progress.

1.5 Thesis Overview

After reviewing existing educational systems in Chapter 2, Chapter 3 describes Tutor-Complete's gameplay and design. We give an overview of Tutor-Complete's background story and the DFA and pumping lemma concepts the activities focus on. We discuss the design considerations of Tutor-Complete, including the goal to make the story and gameplay applicable and enjoyable without distracting from the educational purpose. Each of the levels is described with its corresponding activities, hero and villain characters, and reward items. Tutor-Complete's features are explained in terms of the educational goals listed above. Chapter 4 describes the algorithms used in the intelligent hint generation process. Chapter 5 discusses the Bayesian Knowledge Tracing implementation and usage in Tutor-Complete to model students' knowledge. The evaluation process and results are discussed in Chapter 6. Finally, Chapter 7 summarizes the results of the thesis and outlines the plans for future work.

Chapter 2

Related Work

"If you want to go somewhere, it is best to find someone who has already been there."

Robert Kiyosaki

Before discussing Tutor-Complete in greater detail, we survey several examples of existing educational systems. This chapter outlines three categories of existing systems: microworlds, concept testers, and educational games. Each type of system has some similarities to Tutor-Complete. However, Tutor-Complete also incorporates different features or addresses different issues than each of these existing systems.

2.1 Microworlds

Many introductory programming courses use microworlds to boost student enjoyment while completing homework assignments. Like Tutor-Complete, these systems aim to present computer science concepts in a concrete and engaging manner, and to provide a visual, fun environment for students to interact with course content. However, these systems are not educational games, since they lack the underlying story and reward system that characterize games. In addition, they are targeted toward introductory rather than advanced courses.

2.1.1 BuggleWorld, TurtleWorld, and PictureWorld

These three microworlds were developed at Wellesley College for use in CS 111 (Introduction to Programming and Problem-Solving). In the former version of CS 111, taught in Java, they were used to increase students' excitement about programming concepts

Rings
new Buggle()
forward()
backward()
left()
right()
getPosition()
getHeading()
getColor()
isBrushDown()
isFacingWall()
isOverBagel()
setPosition
setHeading
setColor
setDelay
brushDown()
brushUp()
dropBagel()
pickUpBagel()
Step
Run
Pause
Reset
Polygon Pa
polygonWhile
polygonFor

Sides 3
Length 100

(a) BuggleWorld (b) TurtleWorld

FIGURE 2.1: BuggleWorld and TurtleWorld [1]

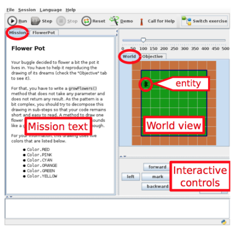


FIGURE 2.2: Programmer's Learning Machine [2]

and provide visual feedback while solving problems, particularly when learning recursion [1].

2.1.2 Programmer's Learning Machine

The Programmer's Learning Machine (PLM) was developed for the CS 1 course at the Université de Lorraine, France. It is a series of microworlds, including a Buggle Universe and a Turtle Universe, based on Wellesley's BuggleWorld and TurtleWorld [2]. PLM contains a set of over 160 predefined exercises for students to complete. It provides examples of the goal for each exercise, and a feedback loop while students work on exercises. PLM is more similar to an educational game in that has a set of goals and rewards, but it lacks the overarching story typical of true educational games, and the exercises are meant to enforce content rather than acquire it.

2.1.3 PacMan

Professors at UC Berkeley developed a set of PacMan projects for their CS 188 (Introduction to Artificial Intelligence) course, which were also used in Wellesley's CS 232 (Artificial Intelligence) course in Fall 2015. In these projects, students implement various AI algorithms in the context of a PacMan game [3]. For example, in one project, students write a perceptron classifier to help PacMan navigate through a maze. The authors found that the projects "have boosted enrollment, teaching reviews, and student engagement", and describe the projects as a tool to teach AI concepts while "allow(ing)

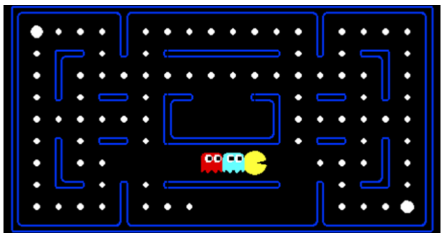


FIGURE 2.3: PacMan [3]

students to visualize the results of the techniques they implement" [18]. The projects present AI algorithms in a game context, but the projects themselves do not constitute a game.

2.2 Concept Testers

Intermediate computer science courses such as Data Structures and Algorithms may use concept testers to allow students to visualize and test abstract concepts. While these systems are designed for more advanced courses and abstract content, they are typically not educational games and also lack the tutoring capabilities of an Intelligent Tutoring System.

2.2.1 JFLAP

JFLAP was developed at Duke University for CPS 140 (Mathematical Foundations of Computer Science), and was also used in Wellesley's CS 235 in Fall 2014. JFLAP is a tool for, among other tasks, constructing and simulating DFAs. It provides immediate feedback in a step-by-step testing mode, as well as a mechanism for students to experiment with DFA construction [4]. The authors also developed PumpLemma, a tool for constructing pumping lemma proofs, which is no longer supported and was not tested by the authors. According to the authors' user testing, students "found the tools useful for testing out their answers" [4].

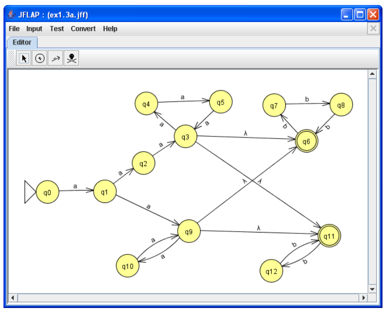


FIGURE 2.4: JFLAP [4]

2.2.2 JDSL Visualizers and Testers

JDSL is a group of visualizers and testers developed jointly at Brown Univeristy and Johns Hopkins University for CS 2 (Data Structures and Algorithms). It is a tool designed for students to interact with data structures and algorithms [5]. The visualizers are a set of animations to help students understand and compare different data structures and algorithms. The testers run student code and compare the output to predefined correct results.

2.3 Games

In addition to microworlds, introductory programming courses often use games to present concepts in a friendly and enjoyable manner. These systems leverage game design, stories and rewards to increase student engagement and motivation, but are typically aimed at introductory courses and lack tutoring capabilities.

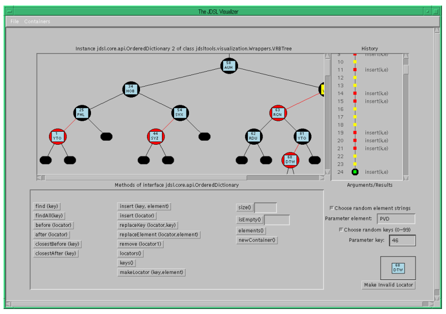
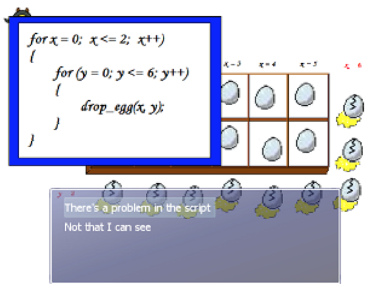


FIGURE 2.5: JDSL [5]



```
for x = 0; x <= 2; x++)
{
    for (y = 0; y <= 6; y++)
    {
        drop_egg(x, y);
    }
}
```

There's a problem in the script

Not that I can see

(a) Saving Sera

(b) Catacombs

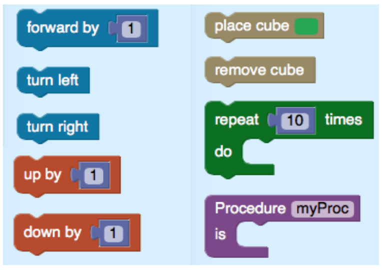
Figure 2.6: Game2Learn [6]

2.3.1 Game2Learn

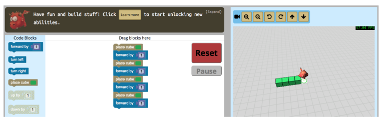
The goal of the Game2Learn lab at North Carolina State University is to "leverage games in retaining students in computer science" [6]. The lab runs a program in which upper-level students develop games for introductory courses, thus providing the students with valuable research experience in addition to the benefits resulting from the introductory games. During one such program, the authors tested two games. In the first, Saving Sera, students write simple programs to complete in-quest tasks, e.g. unscrambling a do-while loop to help a fisherman count his fish. In the second, Catacombs, students solves programming questions in order to construct "spells", e.g. writing nested for loops to construct a bridge.

2.3.2 Dragon Architect

Dragon Architect was developed at the University of Washington for their Introductory Computer Science course. The goal of Dragon Architect is to teach "computational thinking", which the authors break down into concepts, practices, and perspectives [7]. To accomplish this goal, Dragon Architect uses a 3-D game structure in which students use the Blockly language to program a dragon character to assemble a set of blocks, similar to Minecraft. As students progress, they are given more complex code blocks to use.



(a) Code Blocks



(b) Game View

FIGURE 2.7: Dragon Architect [7]

2.3.3 Minecraft

Minecraft is a popular game in many introductory programming courses. Researchers at Flinders University of South Australia studied the effect of Minecraft on introductory programming education by building a set of Minecraft extensions, where students assemble block structures using programming concepts (variables, assignment, selection, looping, etc.) [8]. The researchers' goal was to extend students' CS understanding into higher levels of Bloom's taxonomy by leveraging the familiarity of Minecraft [8].

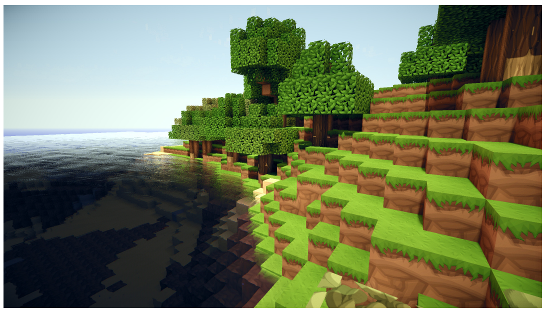


FIGURE 2.8: Minecraft [8]

This chapter concludes the introductory material of the thesis. The next three chapters describe Tutor-Complete in detail. We begin the description by examining the game aspect, including the story, levels, design factors, and specific features. We then consider the algorithms used to implement the hint generation and Bayesian Knowledge Tracing processes that form Tutor-Complete's Intelligent Tutoring System aspect.

Chapter 3

Game Design

"A game is a series of interesting choices."

Sid Meier

This chapter outlines the game aspect of Tutor-Complete, including basic gameplay, activities, and features. We give a brief overview of the two fundamental CS 235 concepts covered in Tutor-Complete (DFAs and the pumping lemma), and discuss the design factors that arose during the construction of Tutor-Complete's game structure. We describe the three game levels and conclude by listing Tutor-Complete's main features according to the educational goals of the project.

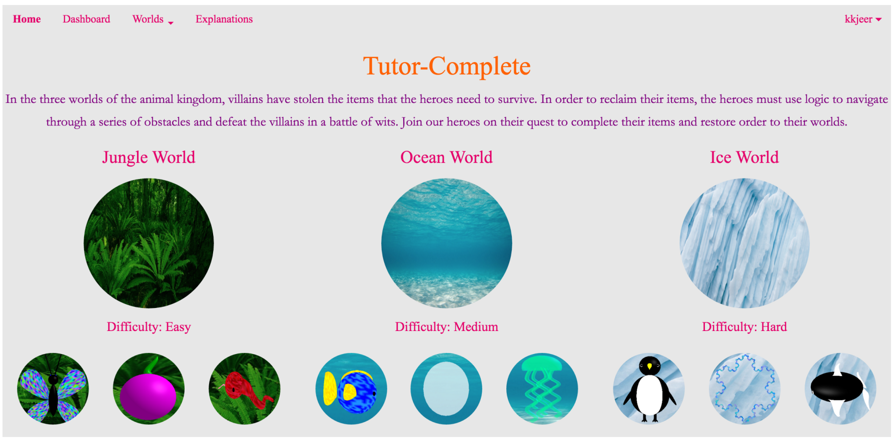
3.1 Game Overview

Tutor-Complete is an adventure game that contains three "worlds", each based on a different part of the natural world: JungleWorld, OceanWorld, and IceWorld. In each world, students play a "hero" character whose quest is to reclaim the "items" that the "villain" character has stolen. In order to reclaim the items, students complete two activities in each world: building DFAs and constructing proofs using the pumping lemma (see Section 3.3).
The hero, item, and villain for each world are listed below:

Tutor-Complete's home page displays the three worlds with their hero, item, and villain underneath them:

World	**Hero**	**Item**	**Villain**
JungleWorld	Butterfly	Berry	Snake
OceanWorld	Fish	Pearl	Jellyfish
IceWorld	Penguin	Snowflake	Killer Whale

TABLE 3.1: Heroes, Items and Villains



Tutor-Complete

In the three worlds of the animal kingdom, villains have stolen the items that the heroes need to survive. In order to reclaim their items, the heroes must use logic to navigate through a series of obstacles and defeat the villains in a battle of wits. Join our heroes on their quest to complete their items and restore order to their worlds.

FIGURE 3.1: Tutor-Complete Home Page

3.2 CS 235 Concepts

Tutor-Complete's activities each focus on one major concept from CS 235. The first activity in each world focuses on DFAs, and the second activity focuses on the pumping lemma. This section gives a brief overview of these two concepts, before the activities themselves are discussed in Section 3.3.

3.2.1 Deterministic Finite-State Automata

The formal definition of a DFA as given by Sipser [19] is below:

Definition 3.1. Finite Automaton
A ***finite automaton*** is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where:

1. Q is a finite set called the ***states***,
2. Σ is a finite set called the ***alphabet***,
3. $\delta : Q \times \Sigma \rightarrow Q$ is the ***transition function***,
4. $q_0 \in Q$ is the ***start state***, and
5. $F \subset Q$ is the set of ***accept states***.

Furthermore, a finite automaton is ***deterministic*** if and only if for each $(q, \sigma) \in Q \times \Sigma$, there is exactly one output $\delta((q, \sigma)) = q' \in Q$ (i.e. δ is a function whose domain is all of $Q \times \Sigma$).

More informally, a DFA is a machine that takes a string of characters from its alphabet Σ and either accepts or rejects the string. A DFA M accepts a string s if and only if M ends on some final state $f \in F$ after reading s. If a DFA M recognizes all strings s in some set L (and rejects all strings not in L), then L is the ***regular language*** recognized by M. A language is regular if and only if it is recognized by some DFA.

3.2.2 The Pumping Lemma for Regular Languages

From Sipser [19]:

Theorem 3.2 (Pumping Lemma)**.** *If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:*

1. *for each $i \geq 0$, $xy^iz \in A$,*
2. *$|y| > 0$, and*
3. *$|xy| \leq p$.*

The pumping lemma is frequently used to prove that some language L is not regular using proof by contradiction: assume that L is regular and give a string $s \in L$ for which the three conditions of the pumping lemma fail, i.e. for each possible parsing of s into $s = xyz$ where $|y| > 0$ and $|xy| \leq p$, there is some $i \geq 0$ such that $s' = xy^iz \notin L$.

3.3 Activities

The DFA and pumping lemma concepts in CS 235 are each emphasized in one activity for each of Tutor-Complete's three worlds. This section details the gameplay of each activity and how its corresponding concept is featured. In keeping with Tutor-Complete's goal of increasing student engagement by presenting concepts in a concrete and enjoyable manner, the activities avoid using CS 235 terminology that may be unfamiliar to students wherever it is feasible and appropriate to do so.

3.3.1 DFA Activity

In the DFA activity in each world, students build a structure that allows the hero character to cross some terrain and collect the reward items. For example, in JungleWorld, the butterfly hero crosses a network of stepping stones in order to cross a river. This structure is a DFA, where the states are stepping stones and the alphabet is {Purple, Orange}. This alphabet is analogous to the $\{0, 1\}$ or $\{a, b\}$ alphabets commonly used in examples from CS 235, but the colors Purple and Orange are used to create a more visual representation.

During this activity, students are given a regular language to build a DFA structure for. The language is called a "pattern" in the activity, to avoid using CS 235-specific terminology. For example, the first pattern in the JungleWorld DFA activity is Purple Orange Orange Purple Orange (analagous to the finite language {01101} in CS 235). Students build and test their structures using the following actions:

1. Add node
2. Mark node as final
3. Add Purple transition
4. Add Orange transition
5. Delete node or transition
6. Test
7. Reset

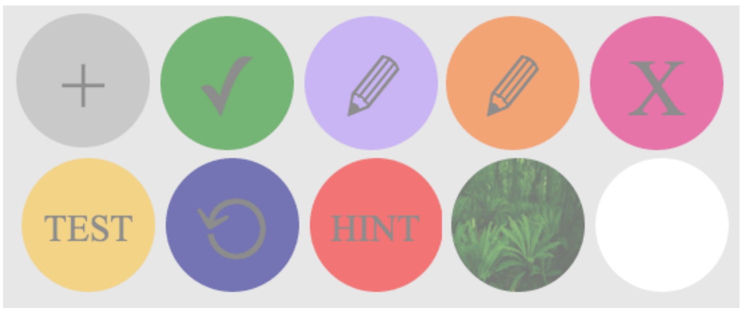


FIGURE 3.2: DFA Activity Buttons

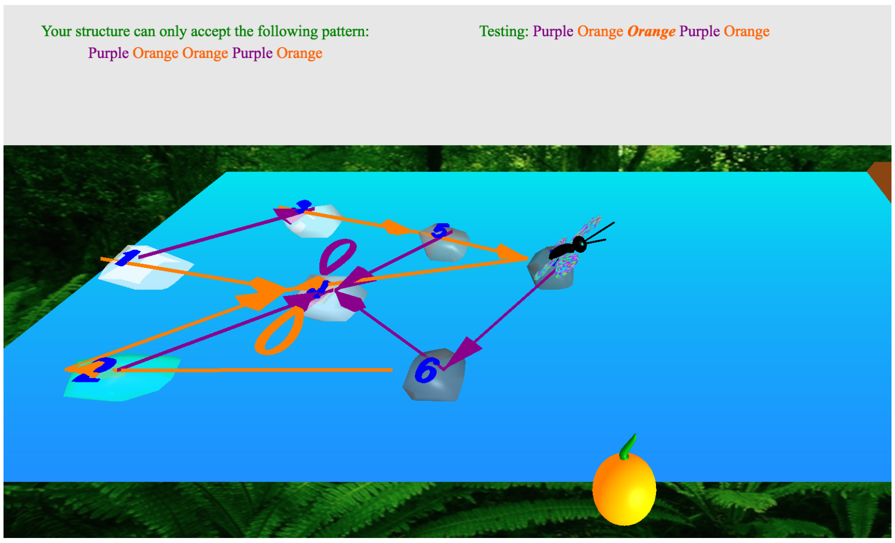
When a student clicks "TEST", their DFA structure will be simulated on two different strings: one that matches the current pattern and one that does not. For example, for the pattern Purple Orange Orange Purple Orange, the system will first test the student's

DFA on the string Purple Orange Orange Purple Orange, and then test the DFA on a random string such as Purple Orange Purple Purple. When testing strings, the system will match the test string character by character against the current pattern. If the characters match, the system will send out a reward (e.g. a berry in JungleWorld). Otherwise, the system will send out a villain character (e.g. a snake in JungleWorld). For the pattern Purple Orange Orange Purple Orange and the test string Purple Orange Orange Purple Orange,

Pattern Symbol	**Test Symbol**	**System Sends**
Purple	Purple	Purple reward
Orange	Orange	Orange reward
Orange	Orange	Orange reward
Purple	Purple	Purple reward
Orange	Orange	Orange reward

Table 3.2: DFA Activity: Test String Matching Pattern

Since the test string matches the pattern, only rewards are sent out.


Your structure can only accept the following pattern:
Purple Orange Orange Purple Orange

Testing: Purple Orange ***Orange*** Purple Orange

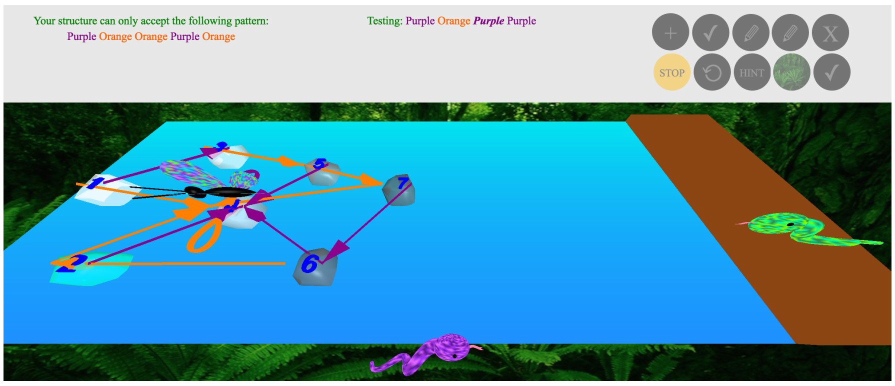
Figure 3.3: DFA Activity: Testing String Matching Pattern

For the pattern Purple Orange Orange Purple Orange and the test string Purple Orange Purple Purple,

Pattern Symbol	**Test Symbol**	**System Sends**
Purple	Purple	Purple reward
Orange	Orange	Orange reward
Orange	Purple	Orange villain
Purple	Purple	Purple reward
Orange	—	—

Table 3.3: DFA Activity: Test String Not Matching Pattern

Since the test string does not match the pattern, at least one villain is sent out.


Your structure can only accept the following pattern:
Purple Orange Orange Purple Orange

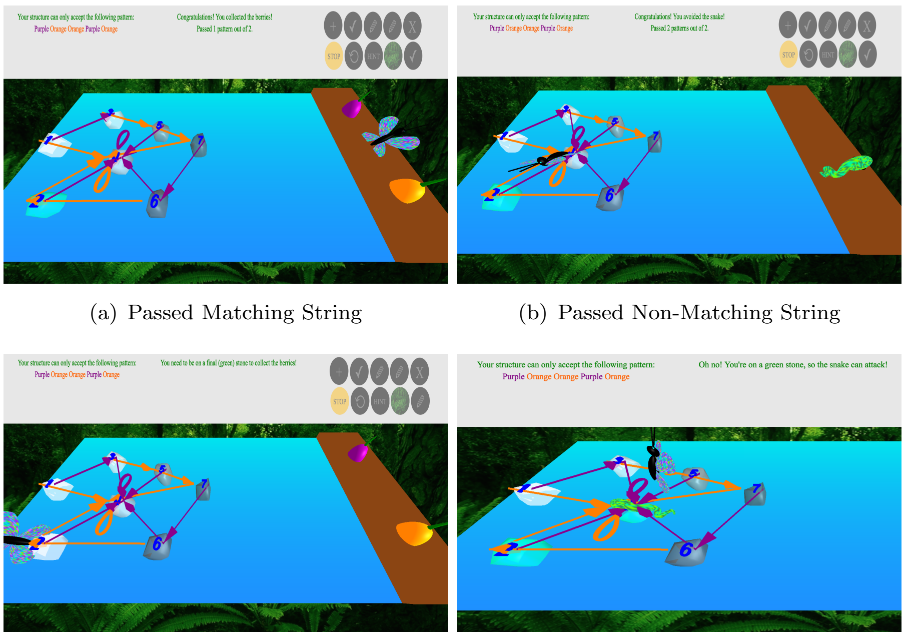
Testing: Purple Orange ***Purple*** Purple

Figure 3.4: DFA Activity: Testing String Not Matching Pattern

To pass a given pattern, the student's DFA must accept the test string that matches the pattern and reject the test string that does not. Students can mark any of their DFA nodes as final by coloring them green. If the hero character ends on a green node while testing a string that matches the pattern, the student passes that test and the hero gets to collect the reward. If the hero character does not end on a green node, the student fails that test case. On the other hand, if the hero character does not end on a green node while testing a string that does not match the pattern, the hero character avoids the villain and the student passes. If the hero does end on a green node, the villain "attacks" and the student fails. These outcomes are summarized below:

Test String	**Ending Node**	**Outcome**	**Pass/Fail**
Matches pattern	Green	Hero collects reward	Pass
Doesn't match	Not green	Villain is evaded	Pass
Matches pattern	Not green	Hero doesn't collect reward	Fail
Doesn't match	Green	Villain attacks	Fail

Table 3.4: DFA Activity: Testing Outcomes


Your structure can only accept the following pattern:
Purple Orange Orange Purple Orange
Congratulations! You collected the berries!
Passed 1 pattern out of 2.

Your structure can only accept the following pattern:
Purple Orange Orange Purple Orange
Congratulations! You avoided the snake!
Passed 2 patterns out of 2.

Your structure can only accept the following pattern:
Purple Orange Orange Purple Orange
You need to be on a final (green) stone to collect the berries!

Your structure can only accept the following pattern:
Purple Orange Orange Purple Orange
Oh no! You're on a green stone, so the snake can attack!

(a) Passed Matching String (b) Passed Non-Matching String

(c) Failed Matching String (d) Failed Non-Matching String

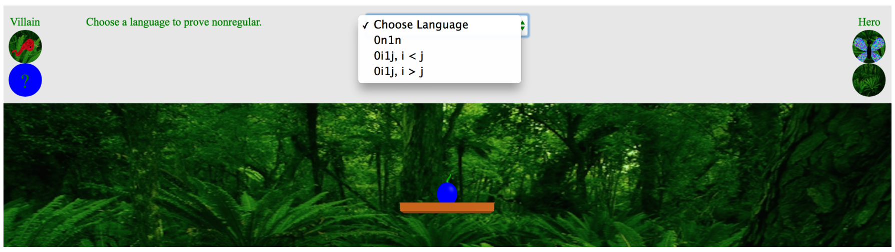
Figure 3.5: DFA Activity: Testing Outcomes

3.3.2 Pumping Lemma Activity

In the pumping lemma activity in each world, the villain character is trying to convince the hero that a given (nonregular) language is regular. The student, playing the hero character, must use the pumping lemma to defeat the villain's argument and prove that the language is in fact nonregular. Each activity represents the villain's argument as some structure that the hero must destroy using a weapon. For example, in JungleWorld, the villain's argument is represented as a table held up by pillars, which the hero knocks down using a bow and arrow. The table holds some reward (e.g. a berry in JungleWorld) that the hero collects once the villain's argument is knocked down.

The activity progresses in the following steps:

1. **The student chooses a language to prove nonregular.**

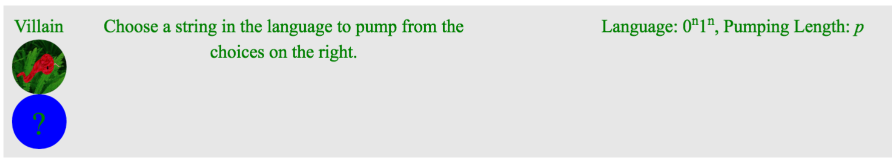


Choose a language to prove nonregular.

FIGURE 3.6: Pumping Lemma Activity: Choosing Language

Each activity has a predetermined set of nonregular languages to choose from (see Section 3.5). For illustration, we will choose the language $\{0^n1^n | n \geq 0\}$.

2. **The villain gives the pumping length p.**



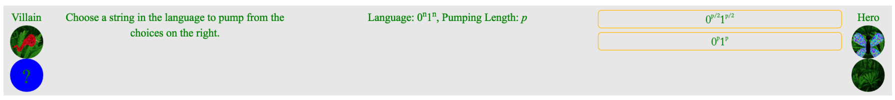
Choose a string in the language to pump from the choices on the right.

Language: 0^n1^n, Pumping Length: p

FIGURE 3.7: Pumping Lemma Activity: Pumping Length

The pumping length of a language is not something the student gets to choose or determine during a pumping lemma proof. Instead, the pumping lemma is typically represented as an unknown constant p. In the game, this is a choice the villain character gets to make, and is given to the hero character.

3. **The student chooses a string s in the language to pump.**



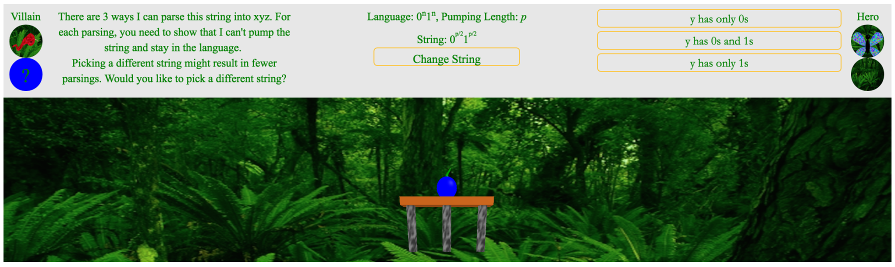
Choose a string in the language to pump from the choices on the right.

Language: 0^n1^n, Pumping Length: p

FIGURE 3.8: Pumping Lemma Activity: Choosing String

Choosing a string in the language (of at least length p) to pump is one of the key decisions that the student must make in a pumping lemma proof. Not all strings "break" the pumping lemma, i.e. some strings can be pumped: for all parsings xyz and all $i \geq 0$, xy^iz is still in the language. Thus, choosing a correct string is vital to the rest of proof. In addition, the string choice can either complicate or simplify the rest of the proof (see **Step 5**).

4. **The villain specifies the ways that he can parse the giving string s into xyz.**



There are 3 ways I can parse this string into xyz. For each parsing, you need to show that I can't pump the string and stay in the language.
Picking a different string might result in fewer parsings. Would you like to pick a different string?

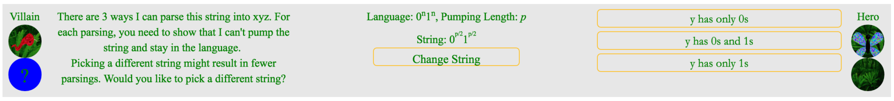
Language: 0^n1^n, Pumping Length: p

String: $0^{p/2}1^{p/2}$

FIGURE 3.9: Pumping Lemma Activity: Possible Parsings

Suppose the student chooses the string $0^{p/2}1^{p/2}$. The villain character can parse this string in three different ways: y can consist of only 0s, 0s and 1s, or only 1s. Since there are three possible parsings, the student has to knock down three "pillars" of the villain's argument. In the game, the table is now held up by three pillars.

5. **If applicable, the student can choose to go back to Step 3 and choose a different string s' in the language to pump.**



There are 3 ways I can parse this string into xyz. For each parsing, you need to show that I can't pump the string and stay in the language.
Picking a different string might result in fewer parsings. Would you like to pick a different string?

Language: 0^n1^n, Pumping Length: p

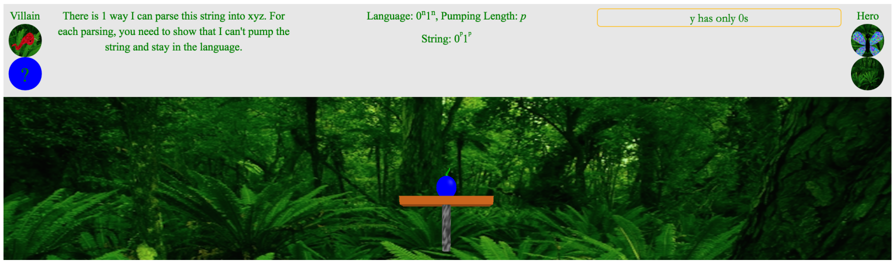
String: $0^{p/2}1^{p/2}$

FIGURE 3.10: Pumping Lemma Activity: Changing String

A judicious string choice will minimize the number of ways the string can be parsed, making the rest of the proof simpler. In our example, if the student chooses $0^{p/2}1^{p/2}$, the villain has three possible parsings. However, choosing the string 0^p1^p would give the villain only one possible parsing, simplifying the remainder of the proof for the student. The activity alerts the student to this fact and gives her the option to change her string choice. As mentioned in **Step 3**, choosing a string is an important skill in pumping lemma proofs, so the activity places a high priority in improving students' ability to make wise string choices.

6. **Once the student is satisfied with her string, she chooses one of the villain's possible parsings in order to prove it is "unpumpable".**

In a pumping lemma proof, the student must show that no matter how the villian parses the student's chosen string s, the villain cannot pump the string and stay in the language, i.e. the parsing is "unpumpable". Thus, the student must do the remaining steps for each possible parsing. In this example, suppose the student chooses to go back and choose the string 0^p1^p. Since the villain now only has one



There is 1 way I can parse this string into xyz. For each parsing, you need to show that I can't pump the string and stay in the language.

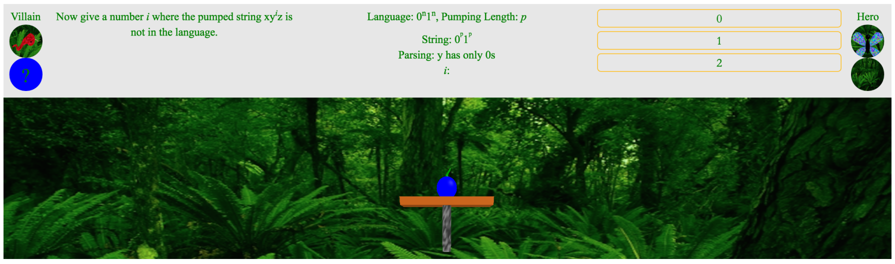
Language: 0^n1^n, Pumping Length: *p*

String: 0^p1^p

FIGURE 3.11: Pumping Lemma Activity: Choosing Parsing

possible parsing, the table is held up by one pillar instead of three. Thus, the student chooses the parsing where y has only 0s for the remainder of the proof.

7. **For the current parsing $s = xyz$, the student chooses an integer $i \geq 0$ such that xy^iz is not in the language.**



Now give a number *i* where the pumped string xy^iz is not in the language.

Language: 0^n1^n, Pumping Length: *p*

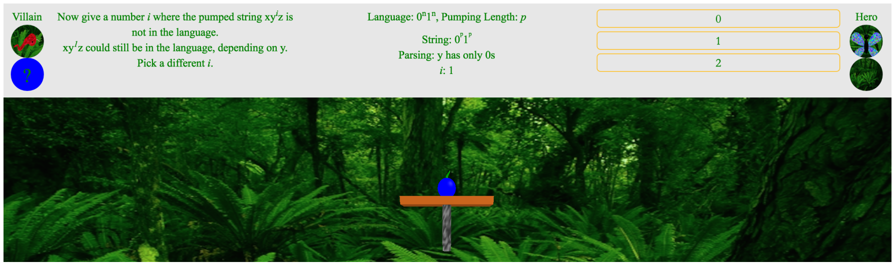
String: 0^p1^p

Parsing: y has only 0s

i:

FIGURE 3.12: Pumping Lemma Activity: Choosing i

The student must choose a value of i such that the villain can't pump the current parsing and stay in the language. Not all values of i fulfill this condition. For example, choosing $i = 1$ allows the villain to pump xyz and stay in the language: $xy^iz = xy^1z = xyz$, which is in the langugage (since xyz is in the language). If the student chooses an incorrect i, no weapon (bow and arrow in JungleWorld) appears.



Now give a number *i* where the pumped string xy^iz is not in the language.
xy^1z could still be in the language, depending on y.
Pick a different *i*.

Language: 0^n1^n, Pumping Length: *p*

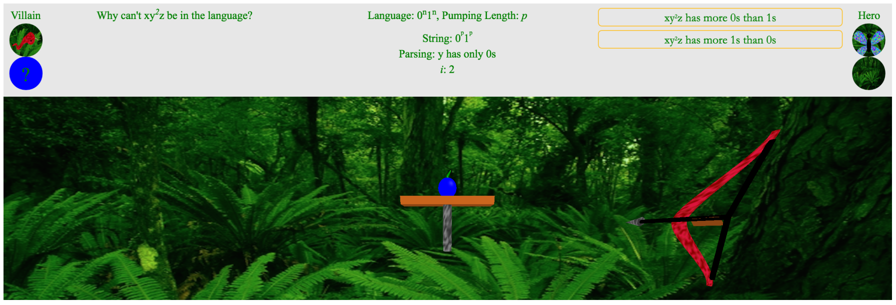
String: 0^p1^p

Parsing: y has only 0s

i: 1

FIGURE 3.13: Pumping Lemma Activity: Choosing an Incorrect i

8. **The student chooses a reason why her chosen value of i forces xy^iz to not be in the language.**



Why can't xy^2z be in the language?

Language: 0^n1^n, Pumping Length: *p*

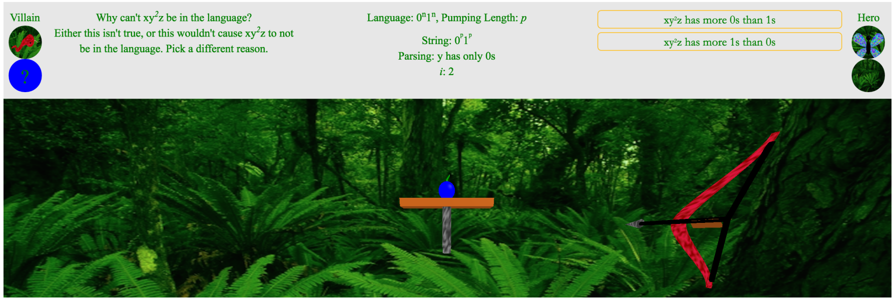
String: 0^p1^p

Parsing: y has only 0s

i: 2

FIGURE 3.14: Pumping Lemma Activity: Choosing Reason

If the student chooses a correct value of i, a weapon appears, but it is not fired. The student must choose a reason in order to fire the weapon. Suppose the student chooses $i = 2$. The student must explain why xy^2z is not in the language, using the fact that, in the current parsing, y contains only 0s. The correct reason in this case is "xy^2z has more 0s than 1s".
If the student chooses an incorrect reason, the weapon fires at a pillar but misses, and then resets so the student can choose a different reason.



Why can't xy^2z be in the language?
Either this isn't true, or this wouldn't cause xy^2z to not be in the language. Pick a different reason.

Language: 0^n1^n, Pumping Length: *p*

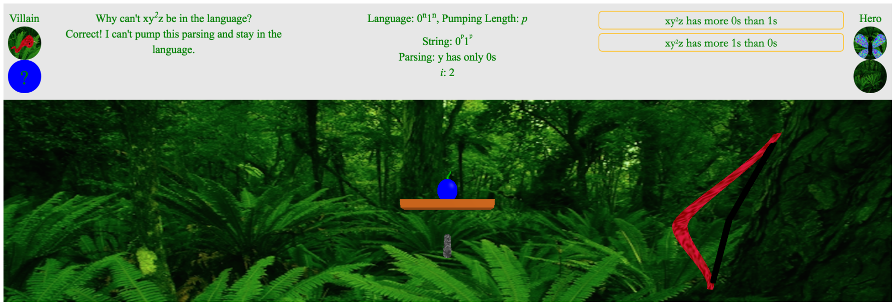
String: 0^p1^p

Parsing: y has only 0s

i: 2

FIGURE 3.15: Pumping Lemma Activity: Choosing an Incorrect Reason

However, if the student chooses a correct reason, the weapon fires at a pillar and knocks it down, and the current parsing is considered complete.



Why can't xy^2z be in the language?
Correct! I can't pump this parsing and stay in the language.

Language: 0^n1^n, Pumping Length: *p*

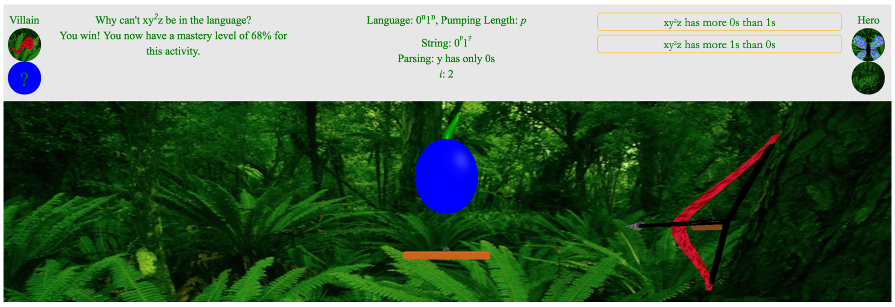
String: 0^p1^p

Parsing: y has only 0s

i: 2

FIGURE 3.16: Pumping Lemma Activity: Choosing a Correct Reason

9. **Steps 6-8 repeat for each of the villain's possible parsings.**



Why can't xy^2z be in the language?
You win! You now have a mastery level of 68% for this activity.

Language: 0^n1^n, Pumping Length: p
String: 0^p1^p
Parsing: y has only 0s
i: 2

xy^2z has more 0s than 1s
xy^2z has more 1s than 0s

FIGURE 3.17: Pumping Lemma Activity: Completing Language

Once each possible parsing has been shown "unpumpable" by the student, the student collects the reward and the proof for this language is completed. The student will not be able to choose this language next time she plays this activity; instead, she must choose a language she has not yet completed (unless she has completed all the languages, in which case she may choose any language).

The steps are summarized in Table 3.5, specifying which steps are the responsibility of the hero character (student) and which are the responsibility of the villain.

Personal experience as both a student and as a tutor for CS 235 suggests that one of the

Step	**Description**	**Responsibility Of**
1	Choose language	Hero
2	Give pumping length	Villain
3	Choose string $\star$	Hero $\star$
4	Specify possible parsings	Villain
5	Change string	Hero
6	Choose parsing	Hero
7	Choose i $\star$	Hero $\star$
8	Choose reason $\star$	Hero $\star$
9	Repeat Steps 6-8 for each parsing	Hero

TABLE 3.5: Pumping Lemma Activity: Steps

major difficulties students have with pumping lemma proofs is keeping track of which parts of a proof are the student's responsibility and which are dictated as part of the problem. For example, students may erroneously think that they are able to choose how the "villain" is able to parse their given string, or that the problem dictates the value of i. Thus, one of the main goals of the pumping lemma activity is to clearly distinguish to the student which choices are her responsibility and which are fixed by the particular problem (chosen by the villain in the game). There are three key choices made by the student in a pumping lemma proof: choosing the string, choosing i, and choosing the reason. These choices are marked with a $\star$ in Table 3.5.

3.4 Design Factors

The design of Tutor-Complete's story and features had two important goals. First, the story needed to be applicable to students with a wide variety of interests and backgrounds. The story was not designed to appeal to any one particular type of student, but to be readily engaging to students with many personal interests. Second, Tutor-Complete needed to increase student motivation and engagement without introducing excessive distractions. One of the main purposes in making Tutor-Complete a game was to engage students with the material; thus, student engagement was a high priority. However, Tutor-Complete is first and foremost an educational system, and any unnecessary distractions would detract from the educational goals of the project.

There were many possible choices for the background story of the game: robotics, medieval fantasy, outer space, etc. The animal kingdom was chosen as the story theme since it fits the following descriptions:

1. **Neutral:** The animal kingdom doesn't invoke gender or similar personal characteristics, and thus avoids any possible complications of gendered characters or other demographics concerns. For example, a game with a princess that needs to be rescued (such as the game Saving Sera - see Section 2.3.1) could involve negative gender stereotypes.

2. **General:** The other possible story themes are more tailored to specific interests. For example, it is less likely that all students would be interested in robotics, and so that story might be less engaging for more users.

3. **Familiar:** Similarly, the animal kingdom is, to at least a reasonable extent, familiar to the majority of possible users. Several of the other possible themes rely on more specific knowledge that would have limited the story's accessibility for some users.

4. **Efficient:** From a development and usability perspective, it is important that the graphical models can be rendered as quickly as possible. The animal kingdom involves models that are more performance-compatible than the humans or other characters in other possible story themes.

5. **Varied:** The animal kingdom allows for the inclusion of a variety of levels. Currently, the game includes the jungle, the ocean, and the Arctic, and other worlds could be added in the future. This theme permits many types of exotic environments while keeping to a unified story.

6. **Simple:** As mentioned above, Tutor-Complete aims to avoid unncessary distractions. One potential distraction is an overcomplicated story that has users reading through long paragraphs of background information. The animal kingdom story has a simple setup that requires very little preliminary reading, yet is still reasonably interesting and engaging.

Another important goal of Tutor-Complete's design was the balance of student enjoyment and distractions. While designing the story, interface, and features, several aspects that were intended to boost student enjoyment gave rise to potential sources of distraction. Below are a few notable enjoyment factors, along with corresponding distraction sources and how they were mitigated:

1. **Graphics and animation:** The three-dimensional graphics and animation were added to increase the enjoyment factor of the game over plain two-dimensional graphics. In part, the three-dimensional graphics aim to increase the similarity between Tutor-Complete and other three-dimensional games that students regularly play for fun.
   Potential distraction: Some entertainment games have long animation sequences with no input from the user. If Tutor-Complete included animation simply for animation's sake, students would be spending time watching animation sequences rather than working through the educational activities. With this in mind, Tutor-Complete only includes animation where it benefits the activity; for example, animating the hero character between nodes in a DFA while testing or animating a weapon to knock down the villain's pumping lemma argument.

2. **Styled web interface:** Designing the user interface was an important process that focused on both usability and aesthetics. Many existing educational systems have minimal interfaces that were not made for aesthetic appeal, but Tutor-Complete's interface was designed to be both easy and pleasant to use.
   Potential distractions: While the interface should be visually appealing, it should not be the focus of the system. Flashy CSS styling or an overabundance of popups could distract students from the educational material. In addition, if the system allowed too much customization, students could get wrapped up in customizing their interface (changing the color scheme, choosing an "avatar", etc.). To avoid these distractions, the interface includes a limited number of clean popups and

minimal CSS animations. Also, while the system does adapt to each student (see Chapter 5), students cannot customize their interface.

3. **Engaging story:** As stated above, a primary goal of the game aspect of Tutor-Complete is to increase student engagement and motivation. Thus, the game should have an interesting background story.
   Potential distraction: An interesting story could have an excessively detailed plotline, requiring students to read long descriptions before beginning the activities, and to remember plot information that could interfere with their ability to focus on the educational material. As discussed earlier in this section, the animal kingdom story was chosen in part for its simplicity and minimal background information.

4. **Points and rewards:** In addition to the story, the game uses points as a motivation factor. Students earn points for their performance in activities, and use their points to unlock subsequent levels. The game also includes a competitive aspect in the explanations feature: students can "fave" each other's explanations, and compete to write explanations that earn the most faves (see Section 3.6.1).
   Potential distractions: The points and rewards are intended to provide additional motivation, not to be the primary focus of the game. Activities that focus solely on accumulating points or earning rewards would detract from the main educational activities. In addition, an overemphasis on competition could cause students to use the system primarily to improve their ranking compared to other students, rather than on improving their understanding of the CS 235 concepts. To avoid these distractions, the game does not include activities other than the DFA and pumping lemma activities discussed earlier in the chapter. The "fave" system is a small part of the system, and the faves do not affect a student's progress in the educational activities. Students do not have to use the explanations feature at all, but the fave system is there to encourage students to write explanations.

3.5 Worlds

Tutor-Complete has three levels, or "worlds" as they are referred to in the game: JungleWorld, OceanWorld, and IceWorld. Each world has a DFA and a pumping lemma activity, with its own type or set of associated languages. The worlds range in difficulty, from Easy (JungleWorld) to Medium (OceanWorld) to Hard (IceWorld). The initial view of each world is shown below.

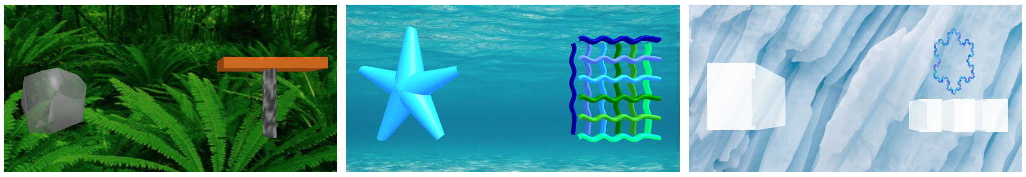

(a) JungleWorld (b) OceanWorld (c) IceWorld

FIGURE 3.18: World Initial Views

3.5.1 World DFA Activities

Each DFA activity draws its problems from a different type of regular language. In each activity, the DFA node is a different type of object. The following lists the language type and node type for each DFA activity.

World	**Language Type**	**Language Example**	**Node Type**
JungleWorld	Finite	01101	Stepping Stone
OceanWorld	English description	Contains at least two 0s	Starfish
IceWorld	Regular expression	01*	Iceberg

TABLE 3.6: World DFA Activities

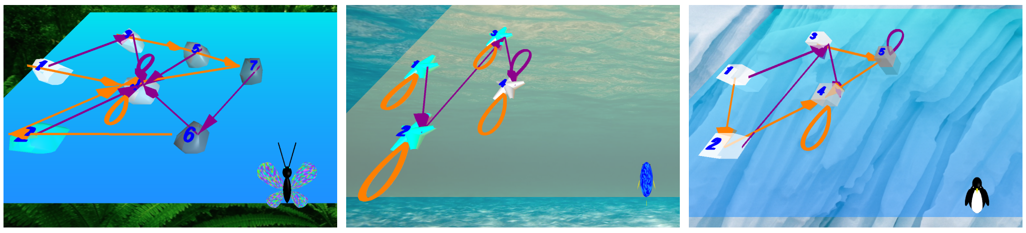

(a) River Activity (b) Starfish Activity (c) Iceberg Activity

FIGURE 3.19: World DFA Activities

Each DFA activity has a set of languages of its corresponding type (e.g. finite languages) that make up the first few problems. After that, the activity will generate random languages of its corresponding type for an indefinite number of problems (so the student can do as many problems from a DFA activity as she wishes).

3.5.2 World Pumping Lemma Activities

Each pumping lemma activity uses a weapon to knock down the villain's argument, which is represented using some structure. Each pumping lemma activity also uses a predetermined set of nonregular languages. The following table lists the weapon and villain argument structure for each pumping lemma activity.

World	Weapon	Villain Argument
JungleWorld	Bow and Arrow	Table, supported by Pillars
OceanWorld	Trident	Net, made up of Strands
IceWorld	Spear	Tower, made up of Icebergs

Table 3.7: World Pumping Lemma Activities

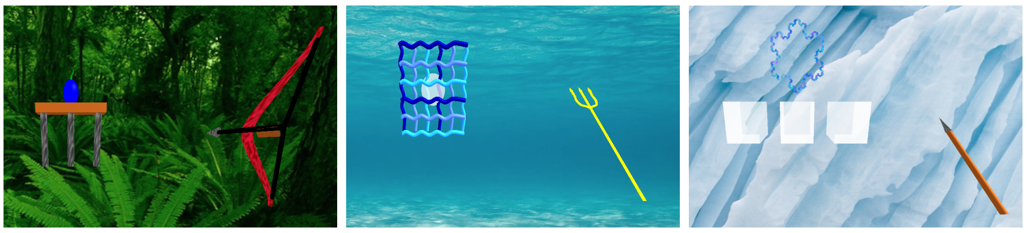

(a) Table Activity (b) Net Activity (c) Tower Activity

Figure 3.20: World Pumping Lemma Activities

3.6 Features

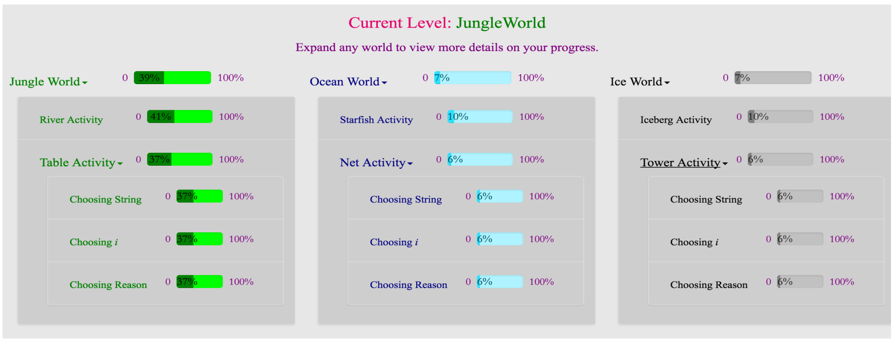
Each of Tutor-Complete's features was designed to fit with the educational goals of the project: educational gaming, peer instruction, and tutoring (see Section 1.4). The major features are described here in terms of the educational goal(s) that they support.

3.6.1 Educational Gaming

The goal of Tutor-Complete's game is to increase student engagement and motivation by presenting the DFA and pumping lemma concepts in a clear and concrete manner. The following features were designed to support the game.

1. **Activity Scores:** In addition to a background story, the game aspect of Tutor-Complete uses a point-based reward system to increase student motivation. Students receive scores in each activity that are determined using Bayesian Knowledge Tracing (see Chapter 5). Each world has a minimum DFA activity score and a minimum pumping lemma activity score that is required to unlock the world.

(JungleWorld has minimum scores of zero, so it is automatically unlocked for all students). Once a student has reached the minimum scores for both activities in a world, she unlocks the next world. For example, once a student's JungleWorld activity scores meet the minimum OceanWorld scores, she unlocks OceanWorld. The activity scores are displayed and updated in real time in each activity, and all activity scores are summarized using progress bars on the student's dashboard.



Current Level: JungleWorld

Expand any world to view more details on your progress.

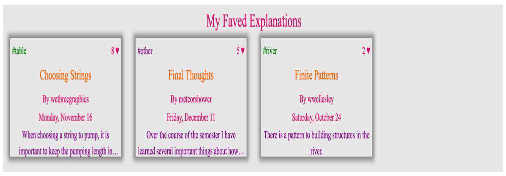
Jungle World 0 39% 100%
- River Activity 0 41% 100%
- Table Activity 0 37% 100%
  - Choosing String 0 37% 100%
  - Choosing *i* 0 37% 100%
  - Choosing Reason 0 37% 100%

Ocean World 0 7% 100%
- Starfish Activity 0 10% 100%
- Net Activity 0 6% 100%
  - Choosing String 0 6% 100%
  - Choosing *i* 0 6% 100%
  - Choosing Reason 0 6% 100%

Ice World 0 7% 100%
- Iceberg Activity 0 10% 100%
- Tower Activity 0 6% 100%
  - Choosing String 0 6% 100%
  - Choosing *i* 0 6% 100%
  - Choosing Reason 0 6% 100%

FIGURE 3.21: Activity Scores

2. **Faves:** Students have the option to write explanations about their ideas and strategies. To encourage high-quality explanations, students can "fave" each other's explanations. Students are given three faves per week that they can use to rate other students' explanations (students cannot fave their own explanations). Students can view explanations they have faved on their dashboard. They can view, sort, search, and fave explanations on the explanations page of the system.



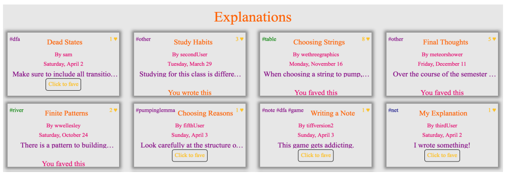
My Faved Explanations

#table 8♥
Choosing Strings
By wethreegraphics
Monday, November 16
When choosing a string to pump, it is important to keep the pumping length in...

#other 5♥
Final Thoughts
By meteorshower
Friday, December 11
Over the course of the semester I have learned several important things about how...

#river 2♥
Finite Patterns
By wwellesley
Saturday, October 24
There is a pattern to building structures in the river.

(a) Student's Faved Explanations



Explanations

#dfa Dead States 1♥
By sam
Saturday, April 2
Make sure to include all transitio...
Click to fave

#other Study Habits 3♥
By secondUser
Tuesday, March 29
Studying for this class is differe...
You wrote this

#table Choosing Strings 8♥
By wethreegraphics
Monday, November 16
When choosing a string to pump,...
You faved this

#other Final Thoughts 5♥
By meteorshower
Friday, December 11
Over the course of the semester ...
You faved this

#river Finite Patterns 2♥
By wwellesley
Saturday, October 24
There is a pattern to building...
You faved this

#pumpinglemma Choosing Reasons 1♥
By fifthUser
Sunday, April 3
Look carefully at the structure o...
Click to fave

#note #dfa #game Writing a Note 1♥
By tiffversion2
Sunday, April 3
This game gets addicting.
Click to fave

#net My Explanation 1♥
By thirdUser
Saturday, April 2
I wrote something!
Click to fave

(b) Explanations Page

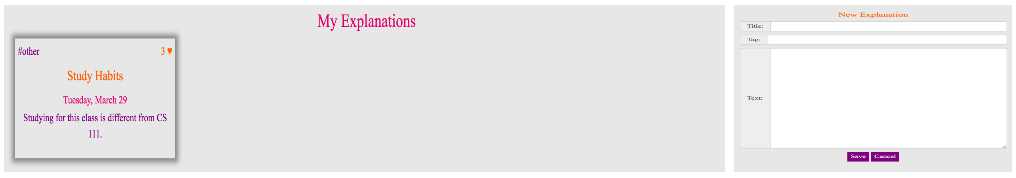
FIGURE 3.22: Faves

3. **Game Actvities:** The DFA and pumping lemma activities discussed in this chapter form the core of Tutor-Complete's game aspect. These activities were designed to be part of a game, while incorporating other educational features such as hints and knowledge tracing, as discussed below.

3.6.2 Peer Instruction

The value of peer instruction has been demonstrated in numerous educational studies, including in technological environments [20–22]. In particular, peer instruction helps students to learn from each other as well as improve their own understanding of the material [22].

1. **Explanations:** While they are not required to do so, students are encouraged to write explanations that all students can view. These explanations are intended to help students benefit from each other's ideas, and also to help students solidify their own understanding by explaining concepts to others. Students can view and edit their own explanations on their dashboard, and they can write new explanations at any time on the explanations page. Explanations can be tagged for quick searching: for example, a student might tag an explanation about finite languages (in the river DFA activity) with #river.



(a) Student's Explanations (b) New Explanation

FIGURE 3.23: Explanations

2. **Hints:** During the DFA activities, students can receive up to two hints per problem (see Chapter 4). These hints are based on the actions that other students have taken when they were at a similar point in the problem. Thus, these hints help students learn from each other's work, and also from each other's mistakes. Since past students may not always have taken the correct action, the hints are not guaranteed to suggest correct actions.

3.6.3 Tutoring

In addition to being a game, Tutor-Complete includes capabilities of a tutoring system. In particular, the system generates hints and tracks students' progress using Bayesian Knowledge Tracing.

1. **Hints:** As stated above, students can receive hints during the DFA activity. This feature helps support the peer instruction goal, but is also a tutoring feature: one characteristic of a computer-aided instructional system is the ability to give hints or feedback to students [13, 23].

2. **Bayesian Knowledge Tracing:** The ability to model students' knowledge of the material is an important characteristic of an Intelligent Tutoring System [13]. Bayesian Knowledge Tracing is used as the student model in Tutor-Complete (see Chapter 5).

This chapter described the first part of Tutor-Complete; namely, the educational game aspect. The following two chapters describe the algorithms used in the Intelligent Tutoring System aspect, beginning with hint generation and then turning to Bayesian Knowledge Tracing.

Chapter 4

Hint Generation

"Talk is cheap. Show me the code."

Linus Torvalds

Hint generation supports both the peer instruction and tutoring educational goals. This chapter first gives an overview of the hint generation process and the role of hints in the DFA activities. The algorithms for implementing hint generation are then described in more detail.

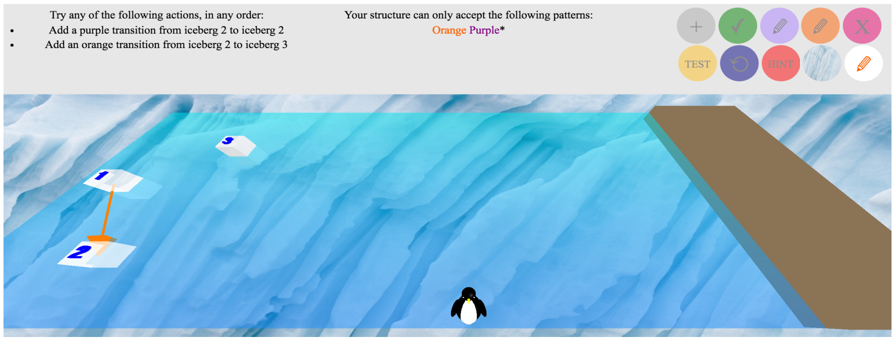
4.1 Overview

Students can request up to two hints per problem in each DFA activity. When a student requests a hint, the system will return up to three suggested actions based on the actions that past students have taken at similar points in the problem. Using past student data to generate hints has been successfully used in existing ITSs [24]. Possible suggested actions take one of the following forms:

1. Add a stone
2. Delete stone X
3. Add a(n) [purple/orange] transition from stone X to stone Y
4. Delete a(n) [purple/orange] transition from stone X to stone Y
5. Mark stone X as final
6. Mark stone X as not final

7. Reset

Below is an example of a hint in the iceberg (IceWorld) DFA activity.


Try any of the following actions, in any order:
- Add a purple transition from iceberg 2 to iceberg 2
- Add an orange transition from iceberg 2 to iceberg 3

Your structure can only accept the following patterns:
Orange Purple*

FIGURE 4.1: Iceberg Activity Hint

Note that not all suggested actions are necessarily correct, since not all past students will have taken the correct action at every point in a problem. In addition, not every suggested action is guaranteed to be relevant. In the above example, the student had already taken the second suggested action ("Add an orange transition from stone 1 to stone 4"). The purpose of the hints is, in part, to support the peer instruction goal by helping students learn from each other's actions and mistakes, not to suggest perfectly optimal and relevant actions every time. However, hints are also intended to be a tutoring feature, so the hint generation process does attempt to suggest at least one helpful and relevant action per hint request. Student feedback on the hints feature is discussed in Chapter 6.

When a student requests a hint, the following process occurs:

1. A set of states and actions is generated for the student's current problem (currently only implemented for JungleWorld).

2. The student's DFA is compared with past student DFAs using a graph similarity algorithm.

3. All past student DFAs and the current student's DFA are grouped according to the similarity calculated in the previous step.

4. A Markov Decision Process (MDP) is built using the DFA groups as MDP states.

5. Value iteration is used to construct a ranked list of up to three suggested actions.

6. Graph similarity is used to translate each suggested action to the structure of the student's current DFA.

The following sections describe the above steps in detail. As an implementation note, each piece of the hint generation process (graph similarity, MDP, etc.) is represented using a JavaScript object, where each function that is shown in pseudocode is part of the object's prototype.

4.2 Initialization

The first hint request for any given problem will not have any data from past students. Thus, in order to ensure that the first hint request for a problem has at least some data to work with, the system generates a set of states and actions for the current problem. Currently, this functionality is only implemented for JungleWorld, which uses finite languages for its DFA activity.

The initialization process constructs the solution to a given language probabilistically: at each step in the solution, the correct next action is taken with probability p and a random action is taken with probability $1 - p$, where p is set to 0.9. This is to ensure that the first hint request is not guaranteed to suggest perfectly correct actions, so that the first student to request a hint does not have an advantage over future students. Subsequent hint requests will have (possibly incorrect) student actions in the database, so the initial hint request intentionally uses possibly incorrect actions as well.

Algorithm 4.1 Hint Initialization

```
function INITIALIZE($language$)
  this.$initialStates \leftarrow [\{nodes : [], transitions : \{purple : \{\}, orange : \{\}\}\}]$
  this.$initialActions \leftarrow []$
  this.ADDSOLUTION($language$)
  this.$lastState.isCorrect = true$
end function

function ADDSOLUTION($language$)
  $langStr \leftarrow language$.RANDOMSTRING()
  for all $s$ in $langStr$ do
    this.ADDSTATE($s$)
  end for
  for $index$ from 0 to this.$initialStates.length$ do
    if Math.RANDOM() $> p$ then
      this.ADDCORRECTTRANSITION($langStr, index$)
    else
      this.ADDRANDOMTRANSITION($langStr, index$)
    end if
  end for
end function
```

The initialization functions are implemented in the Computer object, which maintains an array of solution states (this.*initialStates*) and an array of solution actions (this.*initialActions*). The functions ADDSTATE, ADDCORRECTTRANSITION, and ADDRANDOMTRANSITION manipulate the *initialStates* and *initialActions* fields.

4.3 Graph Similarity

A DFA can be represented as a directed graph, with nodes and transitions (edges) between nodes. In order to find the actions that past students took at a similar point in the problem to the current student, it is necessary to determine what constitutes a "similar point" in the problem: in other words, to determine which past student DFAs are similar to the current student's DFA. Since a DFA can be represented as a directed graph, this step is a graph similarity problem.

Tutor-Complete implements a neighbor-matching graph similarity algorithm from methods developed at the University of Belgrade [25], and adapted from the Java implementation in [26]. The neighbor matching method is summarized below:
Let A and B be two directed graphs. The similarity between node i in A and node j in B is iteratively computed for each iteration k by:

$$x_{ij}^{k+1} \leftarrow \frac{s_{in}^{k+1}(i,j) + s_{out}^{k+1}(i,j)}{2} \tag{4.1}$$

where $s_{in}^{k+1}(i,j)$ and $s_{out}^{k+1}(i,j)$ are the similarity scores for the incoming nodes and outgoing nodes for both i and j, respectively. The similarity between i and j is based on the similarity between all nodes that have edges leading to i and those incoming to j, and on the similarity between all nodes that have edges starting from i and those outgoing from j. The incoming and outgoing similarities have equal weight. The similarity scores are defined as follows:

$$s_{in}^{k+1} \leftarrow \frac{1}{m_{in}} \sum_{l=1}^{n_{in}} x^k_{f_{ij}^{in}(l) g_{ij}^{in}(l)} \qquad s_{out}^{k+1} \leftarrow \frac{1}{m_{out}} \sum_{l=1}^{n_{out}} x^k_{f_{ij}^{out}(l) g_{ij}^{out}(l)} \tag{4.2}$$

$$m_{in} = max(id(i), id(j)) \qquad m_{out} = max(od(i), od(j))$$

$$n_{in} = min(id(i), id(j)) \qquad n_{out} = min(od(i), od(j))$$

where $id(i)$ is the in-degree of node i and $od(i)$ is the out-degree of node i (and similarly for node j). f_{ij}^{in} and g_{ij}^{in} are the enumeration functions that give the maximum similarity value for each node in the given node list [26].

The graph similarity step in Tutor-Complete is implemented using two objects: GraphSimilarity and GraphWrapper. The GraphSimilarity object calculates the similarity between two graphs that contain edges of only one color (purple or orange) using the neighbor matching algorithms from [25, 26]. The GraphWrapper object calculates the similarity between two DFAs using the similarity from both purple and orange edges (transitions), and provides additional functions to map each node a in graph A to its most similar node b in graph B.

Algorithm 4.2 GraphSimilarity Object

```
function GraphSimilarity($nodes1, edges1, nodes2, edges2$)
  this.$nodes1 \leftarrow nodes1$, this.$edges1 \leftarrow edges1$
  this.$nodes2 \leftarrow nodes2$, this.$edges2 \leftarrow edges2$

  this.$inNodeMap1 \leftarrow$this.inNodeMap(this.$nodes1$, this.$edges1$)
  this.$inNodeMap2 \leftarrow$this.inNodeMap(this.$nodes2$, this.$edges2$)
  this.$outNodeMap1 \leftarrow$this.outNodeMap(this.$nodes1$, this.$edges1$)
  this.$outNodeMap2 \leftarrow$this.outNodeMap(this.$nodes2$, this.$edges2$)

  this.$nodeSimilarity \leftarrow new\ Array($this.$nodes1.length)$
  this.$inNodeSimilarity \leftarrow new\ Array($this.$nodes1.length)$
  this.$outNodeSimilarity \leftarrow new\ Array($this.$nodes1.length)$
  this.initializeSimilarityMatrices()
end function

function getGraphSimilarity
  $gSim \leftarrow 0.0$
  this.measureSimilarity()
  if this.$nodes1.length\ ==\ 0$ && this.$nodes2.length\ ==\ 0$ then
    return 1.0
  end if
  if this.$nodes1.length\ <$ this.$nodes2.length$ then
    $gSim \leftarrow$ this.enumFncn(this.$nodes1, this.nodes2, 0)/$this.$nodes1.length$
  else
    $gSim \leftarrow$ this.enumFncn(this.$nodes1, this.nodes2, 1)/$this.$nodes2.length$
  end if
  if this.$edges1.length\ ==\ 0$ && this.$edges2.length\ ==\ 0$ then
    return 1.0 - 0.5 * $Math.$abs(this.$nodes1.length-$this.$nodes2.length$)
  end if
  return $gSim$
end function
```

The pseudocode for the remainder of the GraphSimilarity functions can be found in Appendix A.

Algorithm 4.3 GraphWrapper Object

```
function GraphWrapper($nodes1, edges1, nodes2, edges2$)
  this.$nodes1 \leftarrow nodes1$, this.$edges1 \leftarrow edges1$
  this.$nodes2 \leftarrow nodes2$, this.$edges2 \leftarrow edges2$

  this.$pSim \leftarrow new\ GraphSimilarity(nodes1, edges1.purple, nodes2, edges2.purple)$
  this.$oSim \leftarrow new\ GraphSimilarity(nodes1, edges1.orange, nodes2, edges2.orange)$

  this.$nodeSimilarity \leftarrow new\ Array($this.$nodes1.length)$
  this.computeNodeSimilarityMatrix()
end function

function getGraphSimilarity
  $purpleSimilarity \leftarrow$this.$pSim$.getGraphSimilarity()
  $orangeSimilarity \leftarrow$this.$oSim$.getGraphSimilarity()
  $nodesDifference \leftarrow Math$.abs(this.$nodes1.length$ - this.$nodes2.length$)
  return $0.5 * purpleSimilarity + 0.5 * orangeSimilarity - 0.1 * nodesDifference$
end function

function setNodeMap
  this.$nodeMap \leftarrow \{\}$
  for $index$ from 0 to this.$nodes1.length$ do
    $mostSimilarIndex \leftarrow$ maxIndex(this.$nodeSimilarity[index], index$)
    $mostSimilarNode \leftarrow$ this.$nodes2[mostSimilarIndex]$
    this.$nodeMap$[this.$nodes1[i]$] $\leftarrow mostSimilarNode$
  end for
  return this.$nodeMap$
end function

function computeNodeSimilarityMatrix
  for $i$ from 0 to this.$nodes1.length$ do
    for $j$ from 0 to this.$nodes1.length$ do
      $purpleNodeSim \leftarrow$ this.$purpleGraphSim.nodeSimilarity[i][j]$
      $orangeNodeSim \leftarrow$ this.$purpleGraphSim.nodeSimilarity[i][j]$
      this.$nodeSimilarity[i][j] \leftarrow 0.5 * (purpleNodeSim + orangeNodeSim)$
    end for
  end for
end function
```

The helper function MAXINDEX($array, index$) (not shown) returns the index of the maximum element of the given array, starting at the given index and searching in both directions. It is used to find the node with the highest similarity score to each node in this.$nodes1$ in SETNODEMAP.

4.4 Grouping

Since student DFAs can vary widely, the hint generation process considers groups of student DFAs rather than individual DFAs. After computing the similarity scores between each past student DFA, the DFAs are grouped together by similarity score. Two DFAs D_1 and D_2 are in the same group, or "cluster", only if the similarity score between D_1 and D_2 is greater than or equal to some minimum score MIN_SIM, set to 0.9.

The grouping step is implemented using a Cluster object, which maintains an array of clusters. Each cluster is an array of DFAs (represented as graphs) that are at least 0.9 similar to each other. The Cluster object iteratively builds its clusters by finding the cluster to which each DFA D is most similar. If D is at least 0.9 similar to the first DFA M in the existing cluster, D is added to the cluster. Otherwise, a new cluster is created and D becomes the first DFA in the new cluster.

In addition to building clusters, the Cluster object also computes a reward for each cluster. Each cluster reward is proportional to the number of correct DFAs it contains. For example, if a cluster contains 7 correct DFAs and 3 incorrect DFAs, that cluster's reward is $0.7 * CORRECT\_REWARD$, where $CORRECT\_REWARD$ is set to 100. (Each DFA that is passed to the Cluster object has a field $isCorrect$, which is determined while the student is playing the DFA activity). These rewards are used in the Markov Decision Process to determine which past student actions led to the best outcome, i.e. to the cluster with the highest reward (see Section 4.5).

The pseudocode for the Cluster object is shown below. The "states" field of a Cluster is an array of DFAs. These states are distinct from the states of each individual DFA.

Algorithm 4.4 Cluster Object

```
function CLUSTER($states$)
  this.$states \leftarrow states$
  this.$clusterArray \leftarrow []$
  this.$stateToClusterMap \leftarrow \{\}$
  this.$clusterRewards \leftarrow []$
  for all $state$ in this.$states$ do
    this.CLUSTERSTATE($state$)
  end for
end function

function CLUSTERSTATE($state$)
  $maxSim \leftarrow 0$
  $clusterIndex \leftarrow -1$

  //find the index of the existing cluster to which $state$ is at least 0.9 similar, if any
  for $i$ from 0 to this.$states.length$ do
    $testState \leftarrow$this.$clusterArray[i][0]$
    $graphWrapper \leftarrow new\ GraphWrapper(testState.nodes,\ testState.transitions, state.nodes, state.transitions)$
    $testSim \leftarrow graphWrapper$.GETGRAPHSIMILARITY()
    if $testSim\ > maxSim$ then
      $maxSim \leftarrow testSim$
      if $maxSim\ >= MIN\_SIM$ then
        $clusterIndex \leftarrow i$
      end if
    end if
  end for

  //add $state$ to its most similar cluster or create a new cluster if necessary
  if $clusterIndex\ ==\ -1$ then
    this.$clusterMap$.PUSH[$state$]
    this.$stateToClusterMap[state.name] \leftarrow$this.$clusterArray.length - 1$
  else
    this.$clusterArray[clusterIndex]$.PUSH($state$)
    this.$stateToClusterMap[state.name] \leftarrow clusterIndex$
  end if
end function

function SETREWARDS
  for $i$ from 0 to this.$clusterArray.length$ do
    $correctStates \leftarrow 0$
    for $j$ from 0 to this.$clusterArray[i].length$ do
      if this.$clusterArray[i][j].isCorrect$ then $correctStates \leftarrow correctStates + 1$
      end if
    end for
    $reward \leftarrow$CORRECT_REWARD * correctStates/$this$.clusterArray[i].length
    this.$clusterRewards[i] \leftarrow reward$
  end for
end function
```

4.5 Markov Decision Process

Once a Cluster object has been created to hold an array of clusters of student DFAs, those clusters become states in a Markov Decision Process (MDP). The MDP is defined as follows:

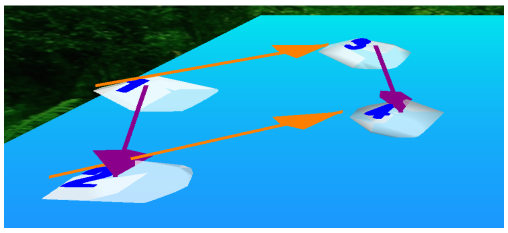
- S (States): set of clusters of DFAs - from Cluster object
- A (Actions): set of past student actions
- $P(s, s')$ (Transition probabilities):

$$\frac{\text{\# of past student actions leading from state } s \text{ to state } s'}{\text{\# of total past student actions leading from } s} \tag{4.3}$$

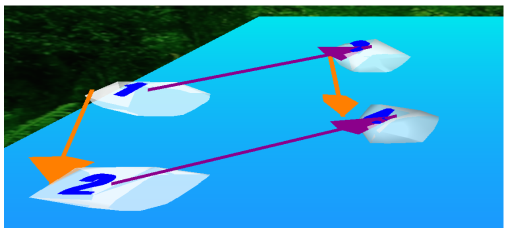
- $R(s)$ (Rewards): scaled proportion of correct DFAs in state s - from Cluster object
- γ (Discount factor): 0.9 - to encourage actions that led to a solution more quickly

This step is implemented using an MDP object. The MDP object uses value iteration to find at most three actions that led from the cluster containing the student's current DFA to the highest-valued cluster(s).

Once the actions are found, the MDP translates the actions back to the student's current DFA using the nodeMap in the GraphWrapper object that maps each node from a DFA $D1$ to its most similar node in another DFA $D2$. For example, suppose the suggested action is "Add a purple transition from stone 4 to stone 2", the past student DFA that generated the suggested action is the DFA $D1$, and the current student has the DFA $D2$ as shown below:



(a) Past Student DFA $D1$



(b) Current Student DFA $D2$

FIGURE 4.2: Translating Suggested Actions

The GraphWrapper object maps each node in $D1$ to its most similar node in $D2$ as follows:

$D1$ **Node**	**Most Similar** $D2$ **Node**
Stone 1	Stone 1
Stone 2	Stone 3
Stone 3	Stone 2
Stone 4	Stone 4

Table 4.1: Mapping Similar DFA Nodes

As above, suppose the suggested action is "Add a purple transition from stone 4 to stone 2". This action was generated from the past student DFA $D1$, but it cannot be presented to the current student as it is, since stone 2 in the past student DFA does not correpond to stone 2 in the current student's DFA. Instead, the hint is translated as "Add a purple transition from stone 4 to stone 3", since the past stone 4 maps to the current stone 4 and the past stone 2 maps to the current stone 3.

The MDP object constructs an array of hints (suggested actions) for a given student DFA using the following process. Suppose the MDP state (cluster of DFAs) containing the student DFA is C.

1. Compute the transition probabilities $P(s, s')$ for each pair of states s and s'
2. Set the value of each state s based on Bellman value iteration:

$$V_{i+1}(s) = max\Big\{\sum_{s'} P(s,\ s')(R(s) + \gamma V(s'))\Big\} \tag{4.4}$$

3. Construct an array of states $[s_1, s_2, s_3, \ldots]$ such that C contains actions leading to each s_i, sorted by decreasing value (s_1, s_2, and s_3 have the highest values)
4. Determine which actions a_1, a_2, and s_3 lead from C to s_1, s_2, and $s3$, respectively
5. Translate each action a_i to the student DFA

Calling the MDP function getHints on a student DFA returns an array of up to three actions, translated to the student DFA as discussed above. For a given student DFA, the MDP may not be able to find three actions: the student DFA could belong to a state (cluster) with an insufficient number of outgoing actions, depending on the past student data. The user testing results for the hints feature is discussed in Chapter 6.

This chapter described the first set of Tutor-Complete's Intelligent Tutoring System algorithms. We now describe the other set of algorithms: those used in Bayesian Knowledge Tracing.

Chapter 5

Bayesian Knowledge Tracing

"The most important questions of life are, for the most part, really only problems of probability."

Pierre Simon, Marquis de Laplace

Modeling the student's knowledge is an important characteristic of an Intelligent Tutoring Sytem [13]. Tutor-Complete uses Bayesian Knowledge Tracing (BKT) to maintain a set of probabilities that a student has mastered, or learned, certain skills involved in the activities. Each activity uses a different implementation of BKT: the pumping lemma activities use "classical" BKT, based on the original work of Corbett and Anderson [27]. The DFA activites use "contextual" BKT, which has additional functionality to contextually estimate the parameters of the BKT model after every student action. After an overview of the BKT model, this chapter explains both approaches in detail.

5.1 Theory

In general, the purpose of BKT is to estimate the probability that a student has "mastered" a given skill. In order to model skill acquisition, BKT assumes the following distinction between *declarative* knowledge and *procedural* knowledge: "declarative knowledge is factual and experiential, [while] procedural knowledge is goal-oriented and mediates problem-solving behavior" [27]. In Tutor-Complete, the following would be an example of declarative knowledge:

> The pumping lemma states that for a regular language L, every string $s \in L$ that has length $\geq p$ is pumpable (where p is the pumping length of L).

The above example could be expressed as an instance of procedural knowledge as follows:

> **IF** the goal is to prove a language L nonregular,
> **THEN** choose a string s of length $\geq p$ and set a goal to prove s is not pumpable

This instance of procedural knowledge corresponds to the skill of choosing a string in Tutor-Complete's pumping lemma activities.

For a tutoring system activity that has been encoded as a set of skills $\{s_1, s_2, \ldots s_n\}$, BKT estimates the probability that a student has learned a skill s_i after the student takes some action. (This proability will be referred to as the "mastery score" for s_i.) The following diagram depicts the BKT process:

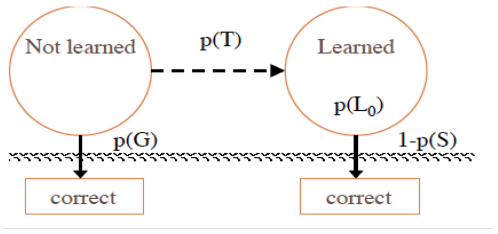


FIGURE 5.1: Bayesian Knowledge Tracing Model

The model uses four parameters. These parameters are computed differently for each activity (see Section 5.2 and Section 5.3).

Parameter	Name	Represents Probability That ...
$p(L_0)$	Initial	Student has learned a skill prior to starting activity
$p(G)$	Guess	Student takes correct action for an unlearned skill
$p(S)$	Slip	Student takes incorrect action for a learned skill
$p(T)$	Transition	Student learns a previously unlearned skill

TABLE 5.1: Bayesian Knowledge Tracing Parameters

If the student has made $n-1$ actions in the activity, the following equations update the probability L_n that the student has learned a skill s_i (the mastery score for s_i) after the student takes the n^{th} action $Action_n$ [28]:

$$P(L_n|Action_n) = P(L_{n-1}|Action_n) + (1 - P(Ln-1|Action_n) * P(T) \tag{5.1}$$

where $P(L_{n-1}|Action_n)$ is the probability that the student had already learned the skill prior to taking $Action_n$, given the action $Action_n$ (in particular, whether $Action_n$ was correct or incorrect. This equation accounts for both the probability that the skill was already learned and the probability that the student learned the skill as a result of taking $Action_n$.

If $Action_n$ is correct, as deemed by the activity, then:

$$P(L_{n-1}|Correct_n) = \frac{P(L_{n-1}) * (1 - p(S))}{P(L_{n-1}) * (1 - p(S)) + (1 - P(L_{n-1})) * p(G)} \tag{5.2}$$

If $Action_n$ is incorrect,

$$P(L_{n-1}|Incorrect_n) = \frac{P(L_{n-1}) * p(S)}{P(L_{n-1}) * p(S) + (1 - P(L_{n-1})) * (1 - p(G))} \tag{5.3}$$

The following sections detail the two different approaches to BKT that Tutor-Complete's activities use to model students' knowledge.

5.2 Pumping Lemma: Classical BKT

The pumping lemma activites adopt a standard, "classical" BKT approach, where each activity uses three skills: choosing a string, choosing i, and choosing a reason (explaining why the pumped string xy^iz can't be in the language).

In the pumping lemma activities, the four BKT parameters take on fixed values:

Parameter	**Pumping Lemma Value**	**Standard BKT Value**
$p(L_0)$	0.05	0.1
$p(G)$	0.3	0.3
$p(S)$	0.1	0.1
$p(T)$	0.1	0.1

Table 5.2: Pumping Lemma BKT Parameters

Each parameter except for $p(L_0)$ is set to the value used in most BKT applications [27, 28]. Since the pumping lemma activities only model three distinct skills, $p(L_0)$ takes a lower value than the standard to prevent students from attaining the minimum mastery score for the activities too quickly. Each world has minimum mastery scores for each activity that students must achieve in order to unlock the next world (except IceWorld, since there is no world after ir). For the pumping lemma activities, students must achieve a minimum mastery score of 0.9 for each skill (choosing string, i, and reason).

Each skill has associated correct and incorrect actions. The mastery score for each skill is updated after the student takes an action (see Chapter 3 for a detailed discussion of the pumping lemma actions). If the student is attempting to prove a language L is nonregular:

Skill	**Correct Action**	**Incorrect Action**
Choosing String	Choosing a string with the minimum number of possible parsings	Choosing a string with extraneous possible parsings
Choosing i	Choosing an i such that xy^iz can't be in L	Choosing an i such that xy^iz could be in L
Choosing Reason	Making a true claim about xy^iz that proves $xy^iz \notin L$	Making a claim about xy^iz that is false or doesn't prove $xy^iz \notin L$

Table 5.3: Pumping Lemma BKT Actions

For example, suppose $L = \{0^n1^n \mid n \geq 0\}$, with pumping length p. The following table illustrates some potential correct and incorrect actions for each skill.

Skill	**Correct Action Example**	**Incorrect Action Example**
String	0^p1^p (one parsing)	$0^{p/2}1^{p/2}$ (three parsings)
i	$i = 2$ ($xy^2z \notin L$)	$i = 1$ ($xy^1z = xyz \in L$)
Reason	Claiming xy^2z has more 0s than 1s	Claiming xy^2z has more 1s than 0s

Table 5.4: Pumping Lemma BKT Actions - Examples

For the skill of choosing a string, the "incorrect" action of choosing a string with more than the minimum number of possible parsings is not strictly incorrect. The proof can still be completed with, e.g., the string $0^{p/2}1^{p/2}$ in the above example, and the pumping lemma activities allow the student to continue without changing to an optimal string. However, this action is a suboptimal choice and suggests that the student has not yet learned the skill of choosing a string, which is one of the key skills in writing pumping lemma proofs. Thus, the action of choosing a suboptimal string is treated as an incorrect action and Equation 5.3 is used to update the mastery score. The other actions fit the ideas of "correct" and "incorrect" in a more straightforward manner: choosing, for example, $i = 1$ cannot be used in a correct pumping lemma proof since $xyz \in L$. Similarity, claiming that xy^2z has more 1s than 0s in the above example cannot be correct, since y must consist of only 0s and thus xy^2z must in fact contain more 0s than 1s.

5.3 DFAs: Contextual BKT

While the pumping lemma activities can be naturally broken down into three distinct skills (choosing a string, choosing i, and choosing a reason), the DFA activities do not lend themselves to this categorization as easily. The "classical" BKT approach of skill breakdown thus is not as suitable for the DFA activities. Instead, these activities only consider one skill: that of constructing DFAs. Thus, the only "correct" action is passing both test patterns, and the only "incorrect" action is failing one or more test patterns. The mastery score for a DFA activity is only updated after the student tests her current DFA (correctly or incorrectly). However, rather than using standard values for $p(G)$ (guess) and $p(S)$ (slip), the DFA activities estimate these parameters after every student action (adding a node, deleting a transition, etc.). This "contextual" approach to BKT has been successfully used in past ITSs [28]. The estimation is done using a method similar to Naive Bayes classification. Using the features listed below, the system estimates $p(G)$ and $p(S)$ based on a set of training data:

Name	**Explanation**
timeTaken	Total time taken on the current language
hintsUsed	Number of hints requested
numResets	Number of times the DFA was reset
numTests	Number of times the DFA was tested
numDeletedTransitions	Number of transitions deleted
numChangedTransitions	Number of transition whose original destination node was changed

Table 5.5: DFA BKT Features

Higher values of each feature result in higher values for $p(G)$ and lower values for $p(S)$. Thus, the student who minimizes each of these features receives higher mastery scores after each action and completes the activity after fewer problems. Intuitively, higher values for each feature generally mean the student has less understanding of the DFA activities. For example, The training data, which was chosen such that the features have this effect on $p(G)$ and $p(S)$, can be found in Appendix B.

$Prob(X)$ where $X \in \{G, S\}$ is estimated using the following equations:

$$Prob(X) = \frac{Prior(X) \times Likelihood(X)}{Evidence(X)} \tag{5.4}$$

where $Prior(G) = 0.3$ and $Prior(Slip) = 0.1$ (standard parameter values), and:

$$Likelihood(X) = \frac{1}{n}\sum_{i=0}^{n-1} likelihood(X, features[i]) \tag{5.5}$$

$$Evidence(X) = \frac{1}{n}\sum_{i=0}^{n-1} evidence(X, features[i]) \tag{5.6}$$

$$likelihood(X, feature) = P(feature \mid X) \tag{5.7}$$

$$evidence(X, feature) = P(feature) \tag{5.8}$$

Equations 5.4, 5.6, 5.7, and 5.8 are part of standard Naive Bayes classifiers. The difference between the standard classifiers and Tutor-Complete's classifier is in Equation 5.5. In a standard Naive Bayes classifier, *Likelihood* is the product of individual probabilities rather than a sum:

$$Likelihood(X) = \frac{1}{n}\prod_{i=0}^{n-1} likelihood(X, features[i]) \tag{5.9}$$

In development, using Equation 5.9 (standard) resulted in very low values for $p(G)$ and $p(S)$ (on the order of 0.001). However, using Equation 5.5 (revised) resulted in more accepted values for $p(G)$ and $p(S)$ (approximately in the ranges [0.2, 0.4] and [0.1, 0.2], respectively). These ranges are close to the values typically used in BKT applications (0.3 and 0.1, respectively). Thus, Tutor-Complete's classifier produces $p(G)$ and $p(S)$ values that fit the general BKT model, but take into account more aspects of the student's actions (time taken, hints used, etc.) than fixed values of $p(G)$ and $p(S)$.

This chapter concludes the description of Tutor-Complete. Having discussed the educational goals, the design decisions, the game mechanics, and the implemented algorithms, we now move on to the evaluation of the system with users.

Chapter 6

Evaluation

"It is a capital mistake to theorize before one has data."

Arthur Conan Doyle

In order to assess Tutor-Complete's usability and effectiveness in motivating and helping students learn, user studies were conducted with a mix of Wellesley students studying Computer Science. This chapter describes the user study methodology and presents the results of the study.

6.1 User Study Design

Thirteen Computer Science students participated in the user study. The students used Tutor-Complete during a one-hour period, in groups of one or two students per hour. The demographics of the students were as follows:

- 7 Seniors
- 4 Juniors
- 2 Sophomores
- 6 Current CS 235 students
- 5 Former CS 235 students (Fall 2014 and Spring 2015)
- 2 Students who had never taken CS 235

Each one-hour session proceeded as follows (the complete user study protocol can be found in Appendix C): Each session began with a brief overview of Tutor-Complete and its activities. The students were then asked to draw a DFA for the language:

$$L = \{w \in \{0, 1\} \mid w \text{ contains at least two 1s}\} \tag{6.1}$$

. This exercise served several purposes: First, to serve as a warm-up for the students, especially for the students who had taken CS 235 several semesters ago. Second, as a diagnostic of the students' initial understanding of DFA construction before they used Tutor-Complete's DFA activity, to better gauge the impact the DFA activity had on students' understanding. Third, as a chance to give a quick (90-second) explanation of DFAs for students who had never taken CS 235. This explanation was kept very brief, since one purpose of the user studies was to assess Tutor-Complete's effectiveness for students who are unfamiliar with CS 235 concepts, particularly the DFA activity.

After the pencil-and-paper exercise, the DFA and pumping lemma activities in the first level (JungleWorld) were demonstrated to the students. The DFA activity was demonstrated on the finite language $L = \{\text{Purple Orange Orange Purple Orange}\}$. Students were able to see how to add and delete nodes (stones in JungleWorld), how to change the destination of a transition from one stone to another, how to mark a stone as final, how to request a hint, and how to test their DFA. The pumping lemma activity was demonstrated on the language $L = \{0^n1^n \mid n \geq 0\}$. Students were able to see how to choose a string, how to change their string in case of a nonoptimal string choice, how to choose i, and how to choose a reason why $xy^iz \notin L$.

Students played both activities in any world of their choice - JungleWorld, OceanWorld, and IceWorld. The students were observed while playing the activities and were free to ask questions and offer feedback at any point. Most of the questions and feedback were related to the interface design (see the later sections of this chapter for a detailed discussion of student feedback). Across the thirteen students, activities from all three worlds were played (though not all students played activities from all three worlds).

The final features that students interacted with were the dashboard and explanations. A brief overview of both features was given, and then students were free to explore their dashboards, write their own explanations and fave past student explanations. Students also commented on the interface and usability of these features (this feedback is discussed in Section 6.4).

Finally, after using Tutor-Complete, students completed an anonymous Google form with a total of fourteen questions about the DFA acvitiy, the pumping lemma activity, and the system as a whole (including the dashboard and explanations). The questions asked for the students' feeback on the interface as well as their thoughts on how Tutor-Complete impacted their understanding of the material. Out of the thirteen students who participated in the study, twelve students completed the form. The results of the form are discussed below.

6.2 DFA Results

The first section of the feedback form consisted of five questions about the DFA activity.

6.2.1 DFA Interface Feedback

The first DFA question asked for students' feedback on the DFA activity interface:

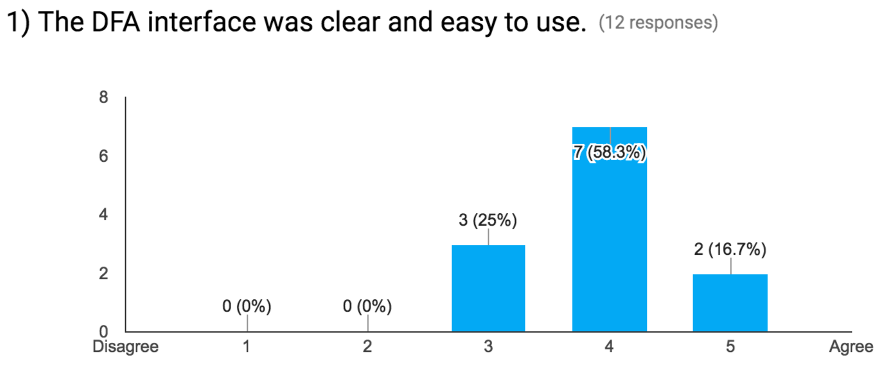


FIGURE 6.1: DFA Activity Interface Feedback

For the most part, the DFA interface feedback was positive, and the interface did not largely hinder the students from completing the activity. Some of the most commonly requested interface features were:

1. The ability to change node locations by dragging
2. An "undo" button to only undo the last action (separate from the "reset" button that clears the entire DFA)
3. Increased testing speed for each problem
4. The ability to "deselect" a node after selecting it to add a transition
5. A warning that informs the student when her DFA is not complete, before the student attempts to test her DFA

6. Additional test cases - not animated as in the first two test cases, but summarized and presented to the student (for more rigorous testing since occasionally an incorrect DFA could still pass both test cases)

These interface updates are planned for future work.

6.2.2 DFA Understanding Feedback

The second DFA question asked for students' feedback on the impact of the DFA activity on their understanding of DFA construction:

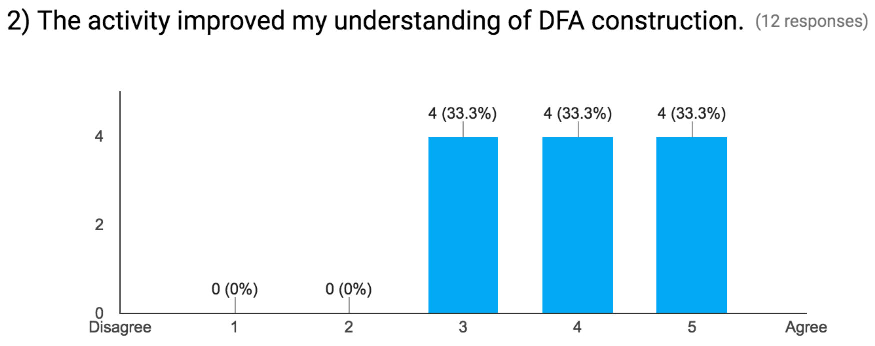


FIGURE 6.2: DFA Activity Understanding Feedback

The DFA understanding feedback was evenly spread across neutral and positive. From observation, this activity was well accessible to students who had never taken CS 235. The two such students in the study were able to complete the activity after the 90-second pencil-and-paper DFA overview. The students were able to play the activities in OceanWorld and IceWorld in addition to JungleWorld, and gained apparent confidence in DFA construction over the course of the hour.

6.2.3 DFA Hints Feedback

The third DFA question asked for students' feedback on the hint generation feature:
The feedback on the hints feature was more varied than that for the preceding questions. During testing, the hint generation process would freeze, suggest actions the student requesting the hint had already taken, or suggest non-helpful actions more frequently than was desirable. One possible reason for this behavior is that students often took non-straightforward paths to complete their DFA, and made non-optimal actions during their solution. Since the hint generation algorithms only consider past student actions, they are not well equipped to handle non-optimal student solutions. Thus, the quality of the hints decreased as the ratio of optimal student solutions to non-optimal student solutions

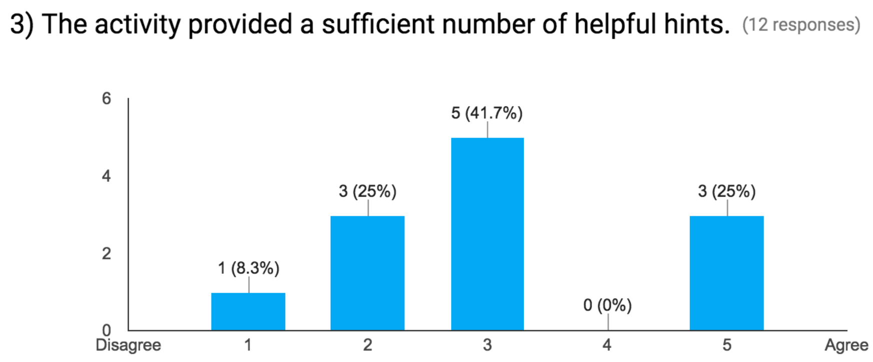


FIGURE 6.3: DFA Activity Hints Feedback

decreased. The hint generation feature provided useful hints for several students, and will remain a part of Tutor-Complete, but in future work will be complemented by additional processes to offer guidance to students as they work through the DFA activities.

6.2.4 DFA Activity Score Feedback

The fourth DFA question asked for students' feedback on their activity scores (calculated by BKT - see Chapter 5):

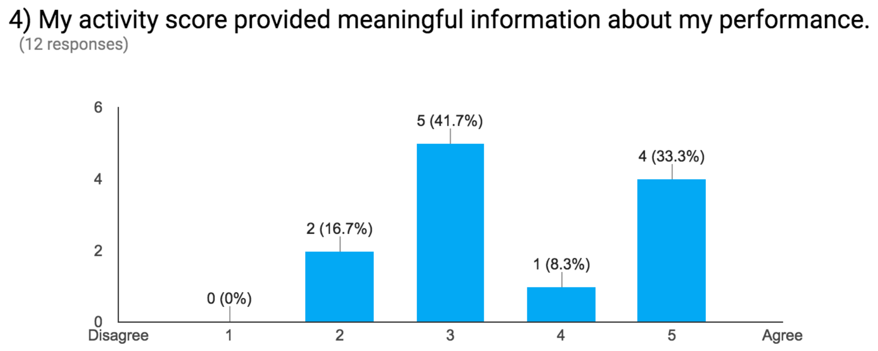


FIGURE 6.4: DFA Activity Score Feedback

The feedback on the activity scores was also more varied than the feedback for the first two DFA questions. Students indicated that they weren't sure what the activity score calculated, and suggested rewording the phrase "activity score" to "mastery level", which has been implemented.

The fifth DFA question asked for students' additional, open-ended feedback on the DFA activity. Students' responses have been summarized in the discussion of the first four DFA questions.

6.3 Pumping Lemma Results

The second section of the feedback form consisted of four questions about the pumping lemma activity.

6.3.1 Pumping Lemma Interface Feedback

The first pumping lemma question asked for students' feedback on the pumping lemma activity interface:

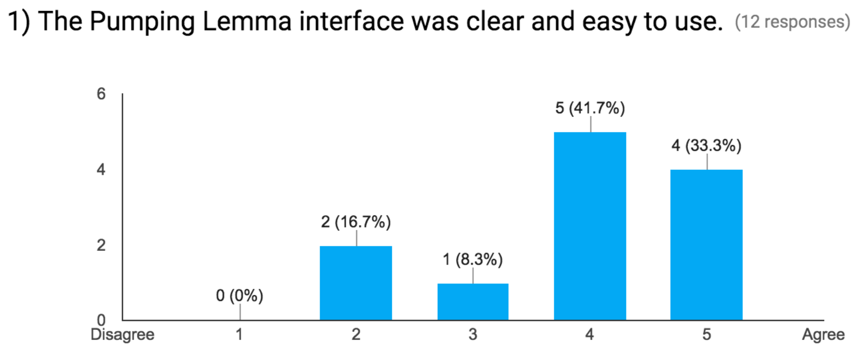


FIGURE 6.5: Pumping Lemma Activity Interface Feedback

The most commonly requested pumping lemma interface features were:

1. The ability to change the string choice even after choosing an optimal string
2. A progress bar or other indication of the number of choices the student has to make during the activity
3. An interactive graphical illustration of pumping a string: the student clicks a button to pump their chosen string, and the graphical representation of their string grows or shrinks, depending on i. For example, if the student has chosen the string 0^p1^p and $i = 2$ for the language $L = \{0^n1^n\}$, then clicking the "pump" button would add more purple objects to the left-hand side of the graphical object that represents the string (the graphical object would appear in the bottom panel that depicts the other graphics, e.g. the bow and arrow and table in JungleWorld).

These interface updates are also planned for future work.

6.3.2 Pumping Lemma Understanding Feedback

The second pumping lemma question asked for students' feedback on the impact of the pumping lemma activity on their understanding of pumping lemma proofs:

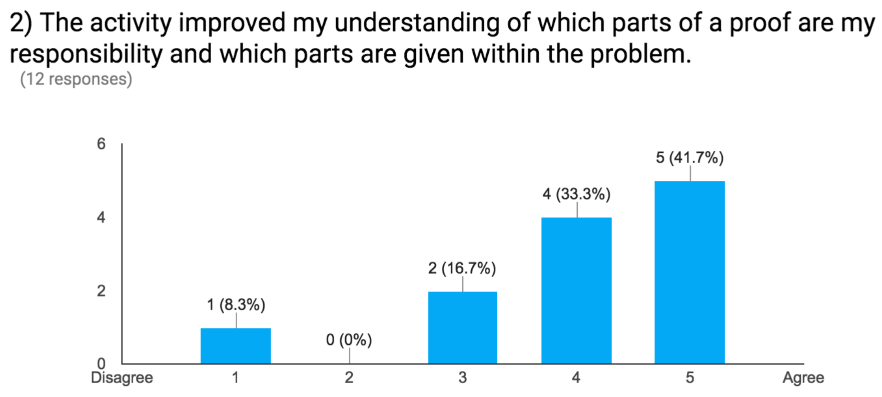


Figure 6.6: Pumping Lemma Activity Understanding Feedback

With one outlier, the feedback for this question was mainly positive. One major goal of the pumping lemma activity was to improve students' understanding of which parts of the proof (choosing the string, specifiying the parsings, etc.) are their responsibility and which parts are given within the problem. For example, it is the student's responsibility to choose a string in the language to pump, but the student does not get to choose how to parse that string. In the game, the parsings are the villain's choice, not the hero's choice. Based on student feedback, both from the form and from observations during user testing, the pumping lemma activity meets this goal.

While the DFA activity was accessible to students who had never taken CS 235, the pumping lemma activity was less so. This was anticipated while designing the activities. The more technical and mathematical nature of the pumping lemma results in a greater need for mathematical notation in the activity, and assumes at least a certain degree of familiarity with the pumping lemma before students begin the activity. However, students who had never taken CS 235 were still able to use the pumping lemma activity during testing, after a slightly longer (2 minute) overview of the pumping lemma, and they reported improved understanding after both the overview and the activity. One of the goals for future work is to reduce the need for prior knowledge of the pumping lemma for these activities, thus increasing their accessibility.

6.3.3 Pumping Lemma Changing String Feedback

The third pumping lemma question asked for students' feedback on the ability to change their string choice:
This feedback was generally positive. Students commented on the usefulness of being able to choose a different string, and also on the ability to proceed with the activity even after choosing a non-optimal string (i.e. changing to an optimal string is not required). Several students did request the ability to choose a different string even after choosing

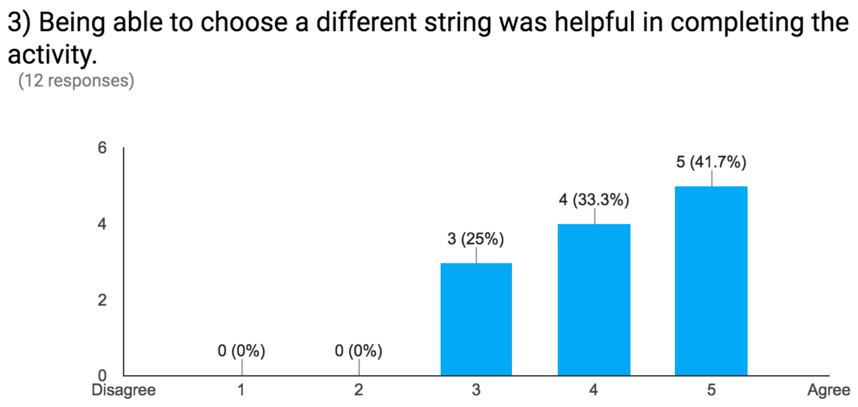


FIGURE 6.7: Pumping Lemma Activity Changing String Feedback

an optimal string (see Section 6.3.1), and this is included in the list of future interface improvements.

The fourth pumping lemma question asked for students' additional, open-ended feedback on the pumping lemma activity. Students' responses have been summarized in the discussion of the first three pumping lemma questions.

6.4 Overall Results

The third and final section of the feedback form consisted of five questions about TutorComplete as a whole. These included questions about the explanations feature and the dashboard.

6.4.1 Explanations Feedback

The first overall question asked for students' feedback on the explanations feature, specifically the impact of writing explanations on their own understanding:
The explanations feedback was generally positive. Several students requested the following features for explanations:

1. The ability to "unfave" previously faved explanations
2. The ability to delete their own explanations

These changes are planned for future work.

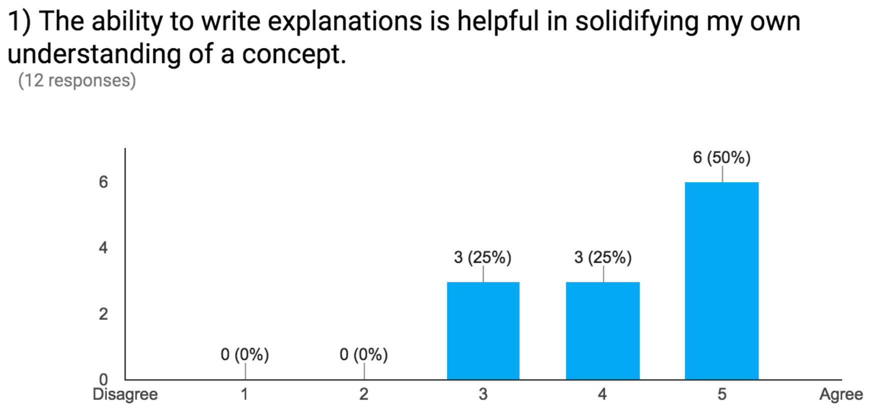


FIGURE 6.8: Explanations Feedback

6.4.2 Story Feedback

The second overall question asked for students' feedback on the back story of the game:

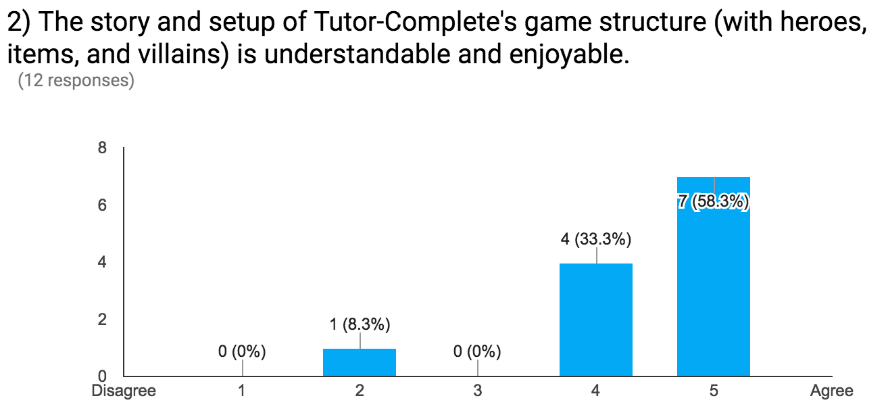


FIGURE 6.9: Story Feedback

Except for one outlier, the story feedback was quite positive. No student brought up any issues with the story being over-specialized or targeted toward any one particular demographic, which fits with the design goals of the story (see Section 3.4).

6.4.3 Dashboard Feedback

The third overall question asked for students' feedback on the dashboard:
The dashboard feedback was generally positive. Students suggested displaying all activity scores as percentages (previously, the pumping lemma scores were displayed as a range from 0 to 0.9), which has been implemented. Some students commented that the breakdown of the pumping lemma scores (scores for choosing a string, choosing i,

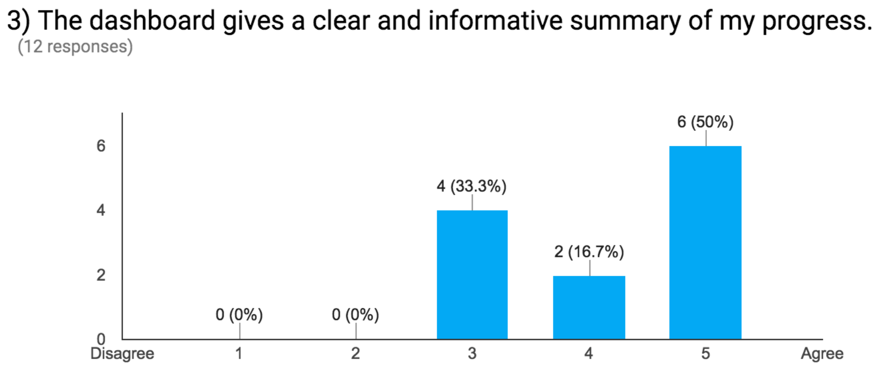


FIGURE 6.10: Dashboard Feedback

and choosing a reason), helped reinforce which parts of a pumping lemma are their responsibility.

6.4.4 General Feedback

The fourth overall question asked how students felt about Tutor-Complete as a whole:

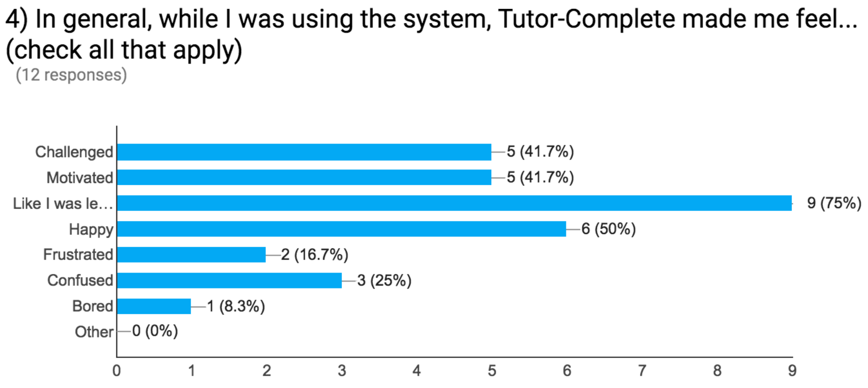


FIGURE 6.11: General Feedback

The majority of the students felt like they were learning while using Tutor-Complete (the third option from the top in the above figure). A reasonable number of students also felt challenged, motivated, and happy. Several students felt frustrated or confused while using the system. They explained their frustration or confusion by making suggestions for interface improvement in the open-ended feedback questions, and many of these suggestions are in plans for future work. The goal is to reduce frustration and confusion for future students.

The fifth overall question asked for students' additional, open-ended feedback on Tutor-Complete as a whole. Students' responses have been summarized in the discussion of the first four overall questions.

This chapter summarized the process and outcomes of evaluating Tutor-Complete with user studies. Having described the background, design, implementation, and evaluation, we conclude by summarizing the results of the thesis and discussing future work.

Chapter 7

Conclusions and Future Work

"Research is creating new knowledge."

Neil Armstrong

As both an educational game and an Intelligent Tutoring System, Tutor-Complete aims to provide CS 235 students with an engaging and educational system for the abstract concepts of DFAs and the pumping lemma. User testing suggests that Tutor-Complete can be effective in helping students learn these concepts. Future work is planned to increase Tutor-Complete's educational potential and usability, particularly by removing causes for frustration or confusion in the interface. This chapter summarizes the conclusions of this research and the items planned for future work.

7.1 Principal Results

Educational games have many potential advantages for student learning. However, while students in CS 111 frequently benefit from games and microworlds such as Minecraft and BuggleWorld, CS 235 students more often learn from dense textbooks and non-game systems such as JFLAP. Tutor-Complete's goal is to provide the benefits of an educational game for students learning the abstract concepts of DFAs and the pumping lemma in CS 235 or similar courses. By combining an educational game with aspects of an Intelligent Tutoring System (ITS), Tutor-Complete aims to:

1. Increase student engagement and motivation by presenting the DFA and pumping lemma concepts in a concrete and enjoyable manner.

2. Leverage the collective knowledge of its users in order to help students learn from each other and solidify their individual understanding by explaining concepts to other students.

3. Offer hints and feedback to students as they work on activities, and model students' knowledge in order to adapt the learning experience to each student.

Tutor-Complete was implemented as a web application, using the Meteor.js framework. Three.js was used to build the three-dimensional graphics used in the game, and Meteor's built-in MongoDB support was used to save students' progress and store information for the hint generation process. Tutor-Complete was developed from scratch for this thesis, with the exception of a small amount of code from the author's prior and concurrent projects. Not including external libraries, Tutor-Complete's implementation took approximately 18000 lines of code.

As an educational game, Tutor-Complete is modeled as an adventure game set in three worlds of the animal kingdom (Jungle, Ocean, and Ice). Each world has two activities that focus on DFAs and the pumping lemma, respectively. In the DFA activity, students build structures (DFAs) to guide the hero character across a terrain in order to collect a reward (items) while avoiding the attack of the villain character. In the pumping lemma, students defeat the villain's argument that a particular nonregular language is regular using the pumping lemma. Students make three distinct choices: choosing a string in the language to pump, choosing a value for i such that the pumped string xy^z is no longer in the language, and choosing a reason why xy^iz is not in the language.

As an ITS, Tutor-Complete generates hints for students in the DFA activity using past students actions. The hint generation compares the student's current DFA to past student DFAs, forms an MDP out of past student DFAs, and solves the MDP to find which past student actions led to the best outcomes. In addition, Tutor-Complete uses Bayesian Knowledge Tracing (BKT) to model students' progress in both activities. The pumping lemma activities use a "classical" BKT approach, which breaks the activity down into three skills: choosing a string, choosing i, and choosing a reason. This approach uses standard values for the guess and slip parameters and computes mastery scores for each skill separately, which are combined to form the mastery score for the pumping lemma activity. The DFA activities use a "contextual" BKT approach, which treats the activity as one skill: constructing DFAs. This approach re-estimates the guess and slip parameters after each student action, using modified Bayesian classifiers and certain features of the action: the time taken, the number of hints requested, etc.

In addition to the game and tutoring features, Tutor-Complete also allows students to write explanations about the concepts they are learning, their approach to the activities, or any topic they deem appropriate for the system. While students are not required to write explanations, they are encouraged to do so by the “fave” system. Students have three faves per week that they can use to rate other students’ explanations. This introduces a mildly competitive aspect to the explanations feature, intended to motivate students to not only write explanations, but to write high-quality explanations. The explanations feature supports Tutor-Complete’s peer instruction goal: to help students learn from each other as well as solidify their own understanding.

In user studies, students generally indicated that the DFA and pumping lemma activites improved their understanding of both concepts. The DFA activities were more accessible to students who had never taken CS 235. The students in the studies were able to complete the DFA activities after only a brief (90-second) overview of DFAs, and their apparent understanding improved as they worked through the activities. The pumping lemma activities were less accessible and required more prior explanation, due in part to the more mathematical and technical nature of the pumping lemma itself. However, students with no CS 235 background were still able to complete the pumping lemma activity after some prior explanation. In addition, the pumping lemma activities were helpful to students in clarifying which parts of a pumping lemma proof are their responsibility and which are embedded in the problem. This was one of the major goals of the pumping lemma activities. Tutor-Complete’s interface was reportedly usable and did not interfere to a great extent with students’ ability to complete the activities. Students did provide valuable feedback on the interface, the most common of which is planned to be addressed in future work.

7.2 Future Work

While Tutor-Complete is a functioning, usable system, there are several improvements that could be made in the future. These improvements are targeted at reducing the amount of frustration and confusion by improving the interface according to students’ feedback, augmenting the tutoring and gaming aspects of the system by adding additional features, and conducting more robust educational study to further assess Tutor-Complete’s impact on students’ learning. The planned improvements to the interface are summarized in List 6.2.1, List 6.3.1, and List 6.4.1.

7.2.1 Tips

While Tutor-Complete's existing hint generation process was reasonably effective during the user studies, the generated hints were either redundant or otherwise not useful more often than was desirable. The limitations of the current process lie mainly in its reliance on past student data, without the ability to reason about the context of the problem or detect when past student actions will not be helpful to the current student. In part, this was an intentional feature of the hint generation process, to support the peer learning goal by helping students learn from each other's mistakes. However, the tutoring capabilities of Tutor-Complete could be improved by adding additional support for hints.

The additional hints support would take the form of a new feature, separate from the existing hints feature, called a "tips" feature. The tips feature would not suggest specific actions, but would offer comments about the structure of the problem that it deems useful based on the student's current DFA. For example, if the student's DFA is missing some transitions, the tips feature would remind the student that their DFA must be complete. If the student has not yet added a dead state to a DFA that could benefit from one (e.g. in a JungleWorld language), the tips feature would provide a brief explanation of the purpose of a dead state. In addition, the tips feature could search the database of student explanations for keywords related to the current problem and present the resulting explanations to the student. As a simple example, in JungleWorld, the tips feature could fetch all the explanations tagged "#river" (the DFA activity in JungleWorld).

7.2.2 Custom Challenges

To further strengthen the motivational value of Tutor-Complete, one desirable feature is the ability for students to create, publish, and play their own custom problems, or challenges, in a style similar to Minecraft. These custom challenges would be added to a database that any student could access, to support the peer instruction goal. Similarly to the explanations, students could search the custom challenges by world and by creator. There would also be a leaderboard of the students whose custome challenges are played the most, to encourage challenge creation and foster competition in a manner similar to the "fave" system for explanations.

To create a custom DFA challenge for a world, a student would create her own language that fits the specifications of the corresponding world (finite languages in JungleWorld, languages with simple English descriptions in OceanWorld, and regular expressions in

IceWorld). She would then build a DFA for her language, to give her additional practice as well as to generate data for hint generation. Optionally, she would write an explanation about her custom challenge, offering suggestions for solving it or any other information she wishes to offer.

To create a custom pumping lemma challenge for a world, a student would create her own language. The language is presumed to be nonregular, but the system would not verify that it is. Instead, future students would have the opportunity to prove that the language is in fact regular by creating a DFA for it in the world's DFA activity. If a future student can show the language is regular, the language would be marked as regular, but no other change would be made to the activity, so additional students could also build a DFA for the language. (The creator of the challenge would be able to see when someone proves her language regular). This gives students practice in not only building DFAs for regular languages and using the pumping lemma on nonregular languages, but in determing whether a given language is regular or nonregular (an important skill in CS 235).

After specifying the language for her pumping lemma challenge, the student would then give at least two strings in the language as string choices. For each string, she would specify at least one way the string can be parsed into xyz. For each parsing, she would give at least two choices for i. Finally, for each choice of i, she would give the correct reason why xy^iz is not in the language and at least one incorrect reason. None of these student-made choices are guaranteed to be correct: for example, the student may specifiy an incorrect list of parsings for a given string. Future students playing the challenge would have the opportunity to tag the challenge with any issues that they find. Only the creator would be able to see the issues that other students have found with her challenge, so she may edit her challenge to fix the issues if she wishes.

7.2.3 Educational Study

The previously described user studies provided valuable insights about Tutor-Complete's educational effectiveness as well as suggestions for future improvement. However, after the interface has been improved according to said suggestions, further testing is desirable to more formally study Tutor-Complete's impact on students' learning. The planned educational study is structured as follows:

Hypothesis: The concrete representation of the abstract concepts in the game, combined with the tutoring features, will result in improved learning for students using Tutor-Complete. In addition, students who use Tutor-Complete will retain the DFA and pumping lemma concepts for longer.

Testing: Tutor-Complete will be tested on two groups of students: students currently taking CS 235 (Group 1) and students who have taken CS 235 in past semesters (Group 2). Each group will be split into two subgroups: Control and Test. The Control group will not use Tutor-Complete, while the Test group will. Both groups will receive a pretest with questions on DFAs and pumping lemmas that students will answer based on their previous knowledge. They will receive a posttest after a period of study time (e.g. 2 weeks), and then a retention test after a period of time with no study (e.g. 3 weeks).

7.2.3.1 Group 1: Current CS 235 Students

Pretest: Group 1 students will receive a pretest with questions that draw from the material that they have learned in class.

Study Period: Group 1 Control students will do regular class activities (textbook reading, homework, etc.) without using Tutor-Complete. Group 1 Test students will also do regular class activities, but in addition use Tutor-Complete during the study period.

Posttest: Group 1 students will receive a posttest with questions that are similar to the pretest.

Retention Test: Group 1 students will receive a retention test with questions that are similar to the pretest and posttest.

Advantage: Since Group 1 students are learning the CS 235 material for the first time, Group 1 data will provide insight into how Tutor-Complete affects the learning process rather than the recall process.

Disadvantage: Group 1 students will be attending CS 235 helproom and office hours, and forming study groups. Thus, there is the possibility that Control and Test students will study together and Control students will benefit from the Test students' experiences with Tutor-Complete. In addition, the help that Control students receive from professors and tutors may boost their posttest and retention test scores, decreasing the difference in scores between the Control and Test groups.

7.2.3.2 Group 2: Former CS 235 Students

Pretest: Group 2 students will receive a brief refresher on DFAs and the pumping lemma that review the concepts that they had learned in their previous CS 235 course, and then receive a pretest with questions based on the refresher. These questions will be similar to the Group 1 pretest.

Study Period: Group 2 Control students will study only texts on DFAs and pumping lemma proofs during the study period (i.e. complete no hands-on activities, tutoring systems, or educational games). Group 2 Test students will use Tutor-Complete during the study period.

Posttest: Group 2 students will receive a posttest with questions that are similar to the pretest.

Retention Test: Group 2 students will receive a retention test with questions that are similar to the pretest and posttest.

Advantage: Group 2 students will not attend CS 235 helproom or office hours, or form study groups, so there is a decreased likelihood that outside factors will influence their test performance compared to Group 1.

Disadvantage: Tutor-Complete is intended as a tool to help students learn CS 235 concepts for the first time. However, since Group 2 students have already taken CS 235, testing Group 2 will result in data about Tutor-Complete's effectiveness in helping students to recall information they have already learned, rather than in helping students to master new concepts.

7.3 Thesis Conclusion

This thesis described Tutor-Complete, an educational game and Intelligent Tutoring System for CS 235 (Languages and Automata). Tutor-Complete's goal is to help students learn the abstract concepts of DFAs and the pumping lemma by presenting these concepts in a concrete and enjoyable game environment, augmented by capabilities of an ITS. While many educational games and ITSs exist for programming courses, there are few such systems that target more advanced, abstract courses such as CS 235.

The game aspect of Tutor-Complete includes an adventure-based story set in three worlds of the animal kingdom, with activities for DFAs and the pumping lemma in each world. The ITS aspect includes hint generation, which uses graph similarity and a Markov Decision Process to suggest actions based on past student data, and Bayesian Knowledge Tracing, which models students' knowledge based on the actions they take.

Tutor-Complete was evaluted by user studies with Wellesley Computer Science students. The studies found that the majority of students felt like they were learning as they used the system, and that the activities improved students' understanding of DFA construction and pumping lemma proofs. While several students felt frustrated or confused while using the system, plans for future work aim to improve the interface according to students' suggestions to reduce future frustration and confusion. Future plans also include adding additional educational features: augmenting the hint generation with a "tips" feature and allowing students to create and play their own custom challenges. Finally, an educational study is planned to further assess Tutor-Complete's impact on student learning.

Appendix A

Hint Generation Functions

Below is the pseudocode to additional functions used in the hint generation feature of Tutor-Complete; specifially, from the graph similarity step (GraphSimilarity object). These functions were based on the Java code at [26].

inNodeMap and outNodeMap return an object mapping each node in *nodesArr* to its predecessors or successors, respectively.

Algorithm A.1 GraphSimilarity: inNodeMap and outNodeMap

```
function inNodeMap($nodesArr, edgesObj$)
  $map \leftarrow \{\}$
  for all $i$ in $nodesArr$ do
    $map[nodesArr[i]] \leftarrow []$
    for all $j$ in $Object$.keys($edgesObj$) do
      $key \leftarrow Object$.keys($edgesObj$)[$j$]
      if $edgesObj[key]$ == $nodesArr[i]$ then
        $map[nodesArr[i]]$.push($key$)
      end if
    end for
  end for
  return $map$
end function

function outNodeMap($nodesArr, edgesObj$)
  $map \leftarrow \{\}$
  for all $i$ in $nodesArr$ do
    $map[nodesArr[i]] \leftarrow []$
    if $edgesObj[nodesArr[i]]$ then
      $map[nodesArr[i]]$.push($edgesObj[nodesArr[i]]$)
    end if
  end for
  return $map$
end function
```

INITIALIZESIMILARITYMATRICES initializes this.$inNodeSimilarity$, this.$outNodeSimilarity$, and this.$nodeSimilarity$ using neighbor matching equations (see Chapter 4).

Algorithm A.2 GraphSimilarity: initializeSimilarityMatrices

```
function INITIALIZESIMILARITYMATRICES
 for $i$ from 0 to this.$nodes1.length$ do
  for $j$ from 0 to this.$nodes2.length$ do
   //in similarity
   $inLength1 \leftarrow$ this.$inNodeMap2$[this.$nodes1[i]].length$
   $inLength2 \leftarrow$ this.$inNodeMap2$[this.$nodes2[j]].length$
   $maxDegree \leftarrow Math.\text{MAX}(inLength1, inLength2)$
   if $maxDegree$ != 0 then
    this.$inNodeSimilarity[i][j] \leftarrow Math.\text{MIN}(inLength1, inLength2)/maxDegree$
   else
    this.$inNodeSimilarity[i][j] \leftarrow 0$
   end if

   //out similarity
   $outLength1 \leftarrow$ this.$outNodeMap1$[this.$nodes1[i]].length$
   $outLength2 \leftarrow$ this.$outNodeMap2$[this.$nodes2[j]].length$
   $maxDegree \leftarrow Math.\text{MAX}(outLength1, outLength2)$
   if $maxDegree$ != 0 then
    $min \leftarrow Math.\text{MIN}(outLength1, outLength2)$
    this.$outNodeSimilarity[i][j] \leftarrow min/maxDegree$
   else
    this.$outNodeSimilarity[i][j] \leftarrow 0$
   end if
  end for
 end for

 //node similarity
 for $i$ from 0 to this.$nodes1.length$ do
  for $j$ from 0 to this.$nodes2.length$ do
   $inSim \leftarrow$this.$inNodeSimilarity[i][j]$
   $outSim \leftarrow$this.$outNodeSimilarity[i][j]$
   this.$nodeSimilarity[i][j] \leftarrow (inSim + outSim)/2$
  end for
 end for
end function
```

MEASURESIMILARITY calculates the similarities between nodes until there is little change ($< EPSILON$) in the node similarity space.

Algorithm A.3 GraphSimilarity: measureSimilarity

```
function MEASURESIMILARITY
 $EPSILON \leftarrow 0.01$
 $maxDifference \leftarrow 0.0$
 $terminate \leftarrow false$

 while $!terminate$ do
  $maxDifference \leftarrow 0.0$
  for $i$ from 0 to this.$nodes1.length$ do
   for $j$ from 0 to this.$nodes2.length$ do
    //calculate in-degree similarities
    $inLength1 \leftarrow$ this.$inNodeMap1$[this.$nodes1[i]].length$
    $inLength2 \leftarrow$ this.$inNodeMap2$[this.$nodes2[j]].length$
    $similaritySum \leftarrow 0.0$
    $maxDegree \leftarrow Math.\text{MAX}(inLength1, inLength2)$
    $minDegree \leftarrow Math.\text{MIN}(inLength1, inLength2)$
    if $minDegree == inLength1$ then
     $inNodes \leftarrow this.inNodeMap1[this.nodes1[i]]$
     $outNodes \leftarrow this.outNodeMap1[this.nodes2[j]]$
     $similaritySum \leftarrow this.enumFncn(inNodes, outNodes, 0)$
    else
     $inNodes \leftarrow this.inNodeMap1[this.nodes1[i]]$
     $outNodes \leftarrow this.outNodeMap1[this.nodes2[j]]$
     $similaritySum \leftarrow this.enumFncn(outNodes, inNodes, 1)$
    end if

    //repeat the above with $outNodeMap$ to calculate out-degree similarities
   end for
  end for

  //calculate node similarity
  for $i$ from 0 to this.$nodes1.length$ do
   for $j$ from 0 to this.$nodes2.length$ do
    $temp \leftarrow (this.inNodeSimilarity[i][j] + this.outNodeSimilarity[i][j])/2$
    if $Math.\text{ABS}(\text{this}.nodeSimilarity[i][j] - temp) > maxDifference$ then
     $maxDifference \leftarrow Math.\text{ABS}(\text{this}.nodeSimilarity[i][j] - temp)$
    end if
    this.$nodeSimilarity[i][j] \leftarrow temp$
   end for
  end for

  if $maxDifference < EPSILON$ then
   $terminate \leftarrow true$
  end if
 end while
end function
```

ENUMFCN calculates the similarity between two graphs.

Algorithm A.4 GraphSimilarity: enumFcn

```
function ENUMFCN($neighborMapMin, neighborMapMax, graph$)
  $similaritySum \leftarrow 0.0$
  $valueMap \leftarrow \{\}$
  $minKeys \leftarrow Object.keys(neighborMapMin)$
  $maxKeys \leftarrow Object.keys(neighborMapMax)$
  for $i$ from 0 to $minKeys.length$ do
    $minKey \leftarrow minKeys[i]$
    $node \leftarrow neighborMapMin[minKey]$
    $max \leftarrow 0$
    $maxIndex \leftarrow -1$
    if $graph\ ==\ 0$ then
      for $j$ from 0 to $maxKeys.length$ do
        $maxKey \leftarrow maxKeys[j]$
        $key \leftarrow neighborMapMax[maxKey]$
        if $Object$.KEYS($valueMap$).INDEXOF($key$) $==\ -1$ then
          if $max\ <$ this.$nodeSimilarity[minKey][maxKey]$ then
            $max \leftarrow$ this.$nodeSimilarity[minKey][maxKey]$
            $maxIndex \leftarrow key$
          end if
        end if
        $valueMap[maxIndex] \leftarrow max$
      end for
    else
      for $j$ from 0 to $neighborMapMax.length$ do
        $maxKey \leftarrow maxKeys[j]$
        $key \leftarrow neighborMapMax[maxKey]$
        if $Object$.KEYS($valueMap$).INDEXOF($key$) $==\ -1$ then
          if $max\ <$ this.$nodeSimilarity[minKey][maxKey]$ then
            $max \leftarrow$ this.$nodeSimilarity[minKey][maxKey]$
            $maxIndex \leftarrow key$
          end if
        end if
        $valueMap[maxIndex] \leftarrow max$
      end for
    end if
  end for
end function
```

Appendix B

DFA BKT Training Data

Below is the data used to train the semi-Bayesian classifier for the BKT process in the DFA activities.

Each feature has a defined set of ranges of values that the feature can take on. These ranges are mapped to training data (see subsequent tables). For example, a student who requests a hint after taking 2 minutes will fall in the first range (0 to 3 minutes) of the timeTaken feature, so the guess evidence for timeTaken will be 0.05 (the first training guess value for timeTaken).

Feature	**Ranges**
timeTaken (in minutes)	0, 3, 6, 9, 12, Infinity
hintsUsed	0, 1, 2, Infinity
numResets	0, 1, 2, 3, Infinity
numTests	0, 2, 4, 6, Infinity
numDeletedTransitions	0, 2, 4, 6, Infinity
numChangedTransitions	0, 2, 4, 6, Infinity

Table B.1: DFA BKT Training Data - Ranges

Each feature has one training value for the guess parameter per range. For example, timeTaken has five ranges (0 to 3, 3 to 6, 6 to 9, 9 to 12, and more than 12 minutes), so timeTaken has five guess values. These values serve as evidence for each feature in the calculation of the guess parameter. For example, if a student has taken 2 minutes on a problem when she requests a hint, the evidence suggests that she is guessing with probability 0.05.

Feature	**Guess Values**
timeTaken	0.05, 0.15, 0.2, 0.25, 0.35
hintsUsed	0.1, 0.35, 0.55
numResets	0.05, 0.2, 0.3, 0.45
numTests	0.1, 0.2, 0.3, 0.4
numDeletedTransitions	0.1, 0.2, 0.3, 0.4
numChangedTransitions	0.1, 0.2, 0.3, 0.4

Table B.2: DFA BKT Training Data - Guess Values

Similar to the guess values, each feature has one training value for the slip parameter per range. These values serve as evidence for each feature in the calculation of the slip parameter. For example, if a student has taken 2 minutes on a problem when she requests a hint, the evidence suggests that she is slipping with probability 0.4.

Feature	**Slip Values**
timeTaken	0.4, 0.3, 0.2, 0.1, 0.05
hintsUsed	0.6, 0.3, 0.1
numResets	0.4, 0.35, 0.15, 0.1
numTests	0.4, 0.3, 0.2, 0.1
numDeletedTransitions	0.4, 0.3, 0.2, 0.1
numChangedTransitions	0.4, 0.3, 0.2, 0.1

Table B.3: DFA BKT Training Data - Slip Values

Finally, each feature has one total training value per range. These values represent the presumed total proportion of students who had values within each range. For example, the data presumes that twenty percent of all students took between 0 and 3 minutes on each DFA problem. Note that the data assumes that prior students were evenly distributed across the ranges for each feature, which is a naive assumption. However, this assumption did not cause the DFA BKT algorithms to calculate any theoretically degenerate values for guess and slip. (A guess value g outside the range $0 \leq g < 0.5$ or a slip value s outside the range $0 \leq s < 0.5$ is said to be theoretically degenerate [28]).

Feature	**Total Values**
timeTaken	0.2, 0.2, 0.2, 0.2, 0.2
hintsUsed	0.33, 0.33, 0.33
numResets	0.25, 0.25, 0.25, 0.25
numTests	0.25, 0.25, 0.25, 0.25
numDeletedTransitions	0.25, 0.25, 0.25, 0.25
numChangedTransitions	0.25, 0.25, 0.25, 0.25

TABLE B.4: DFA BKT Training Data - Total Values

Appendix C

User Study Protocol

C.1 Introduction

Hello! Thank you for volunteering your time to participate in this study. Today you'll be using Tutor-Complete, an educational game for CS 235. Tutor-Complete has two activities that focus on DFAs and the pumping lemma. You'll be playing these activities in three different levels of the game. I will be taking notes while you are doing the activities, and afterward I will ask you for your feedback on the system: how you thought it impacted your understanding of the material, and how you felt about using the interface. Please feel free to ask questions at any point. Don't worry about making a mistake or doing something "wrong" while playing the game - this is a test of the system's design, not your ability to use it. Do you have any questions before we start?

Before we begin, I'd like you to write a DFA for the language
$L = \{w \in \{0,1\} \mid w \text{ contains at least two 1s}\}$ using pencil and paper.
[Give user pencil and paper for DFA construction]

C.2 Demo

C.2.1 DFA Activity

[Open JungleWorld]
First, I'll give a brief demo of the DFA activity in the first level.

[Enter River (DFA) Activity]
The goal of this activity is to build a structure (DFA) out of stepping stones that will

guide the butterfly character across the river. Your DFA should accept the pattern shown here [**Indicate pattern**] and reject all other patterns. If your DFA accepts the pattern, the butterfly will collect the reward of berries. If your DFA rejects any other pattern, the butterfly will avoid the attack of the snake character.

[**Add stones to DFA (including extra for later deletion)**]
Click the gray button to add stepping stones. These are the states of your DFA.

[**Add some transitions**]
Click the purple and orange pen buttons to add transitions between stones. These indicate the moves the butterfly should make when it sees a Purple or and Orange.

[**Click the hint button**]
Click the red button to receive a hint. Hints are based on what past students have done.

[**Add the remaining transitions**]

[**Mark the final stone**]
Click the green button to mark a stone as final. This makes it an accept state in your DFA. You can have as many accept states as you want.

[**Delete the extra stone**]
Click the pink button to delete stones or transitions.

[**Test**]
Click the yellow button to test your DFA on two test patterns: the first is a pattern your DFA should accept, and the second is a pattern your DFA should reject.

[**Indicate updated BKT score**] Since my DFA passed both test cases, I've passed this pattern and my score for the activity has been adjusted. Now you can do the next pattern.

C.2.2 Pumping Lemma Activity

[**Go back to JungleWorld**]
Next, I'll give a demo of the pumping lemma activity.

[**Enter the Table (Pumping Lemma) Activity**]

First, choose the language that you want to prove is nonregular.

[**Choose** 0^n1^n **from dropdown**]
The villain has indicated the pumping length p. Now, it's the hero's job to choose a string.

[**Choose the string** $0^{p/2}1^{p/2}$]
The villain tells me that there are three different ways he can parse my chosen string, and I can make the proof easier on myself by choosing a different string. I will choose to go back and make a different choice.

[**Go back and choose** 0^p1^p]
Now, the villain says that there is only one way he can parse my chosen string into xyz, and I know that, in that one parsing, y can contain only 0s. Now I have to show that the villain can't pump my string using that parsing

[**Click the parsing "y has only 0s"**]
I have to choose a value for i such that xy^iz is not in the language. First, I'll try $i = 1$.

[**Choose** $i = 1$]
That value of i didn't work, since $xy^1z = xyz$ is actually still in the language, so my attempt to knock down the villain's argument failed and a have to try again. This time, I'll choose $i = 2$.

[**Choose** $i = 2$]
Finally, I have to indicate why xy^2z is not in the language, i.e. why this value of i "breaks" the pumping lemma.

[**Choose "more 1s than 0s"**]
This is incorrect, since repeating y adds more 0s to the pumped string, not more 1s. This time I'll pick the correct reason.

[**Choose "more 0s than 1s**]"
This is correct, so I knock down the villain's argument and my proof is complete and I get to collect the reward. My activity score has also been updated [**Indicate updated BKT score**]. The language 0^n1^n is now complete, so you can choose your language to prove nonregular.

C.3 Explanations

Another feature of Tutor-Complete is student-written explanations, intended to help students learn from each other as well as reinforce their own understanding. Please write a short explanation on what you learned or how you felt about using the system. Your name is not associated with the username of the account you are using on the system, so while your username will be attached to the explanation, your real name will not. The reason the username is attached to the explanation is the “fave” system, which allows users to rate one another’s explanations. The rating system will not be used in this study, so don’t worry about how your explanation might be rated.

Bibliography

[1] F. Turbak, C. Royden, J. Stephan, and J. Herbst. Teaching recursion before loops in cs1. *Journal of Computing in Small Colleges*, 14(4):86–101, May 1999. URL http://cs.wellesley.edu/~fturbak/pubs/jcsc99.pdf.

[2] M. Quinson and G. Oster. The programmer's learning machine: A teaching system to learn programming. *ACM Conference on Innovation and Technology in Computer Science Education*, 1(1):1–6, July 2015. URL http://people.irisa.fr/Martin.Quinson/Research/Publications/2015-itiCSE-plm.pdf.

[3] J. DeNero and D. Klein. Teaching introductory artificial intelligence with pac-man. *AAAI Conference on Artificial Intelligence*, 1(24):1–5, January 2010. URL https://www.aaai.org/ocs/index.php/EAAI/EAAI10/paper/viewFile/1954/2331.

[4] A. Bilska, K. Leider, M. Procopuic, O. Procopuic, S. Rodger, J. Salemme, and E. Tsang. A collection of tools for making automata theory and formal languages come alive. *Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education*, 29(1):15–19, March 1997. URL https://www.cs.duke.edu/csed/rodger/papers/cse97flap.pdf.

[5] R. Baker, M. Boilen, M. Goodrich, R. Tamassia, and B. Stibel. Testers and visualizers for teaching data structures. *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education*, 31(1):261–265, March 1999. URL http://www.cs.cmu.edu/~rsbaker/sigcse.pdf.

[6] T. Barnes, E. Powell, A. Chaffin, A. Godwin, and H. Richter. Game2learn: Building cs1 learning games for retention. *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, 39(3):121–125, September 2007. URL https://ncsu.pure.elsevier.com/en/publications/game2learn-building-cs1-learning-games-for-retention.

[7] A. Bauer, E. Butler, and Z. Popovic. Approaches for teaching computational thinking strategies in an educational game: A position paper. *Blocks and Beyond*

Workshop (Blocks and Beyond), 2015 IEEE, 1(1):121–123, October 2015. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7369019.

[8] B. Wilkinson, N. Williams, and P. Armstrong. Improving student understanding, application and synthesis of computer programming concepts with minecraft. *The European Conference on Technology in the Classroom*, 1(1):1–13, 2013. URL http://iafor.org/archives/offprints/ectc2013-offprints/ECTC2013_0477.pdf.

[9] M. Ibáñez, Á. Di-Serio, and C. Delgado-Kloos. Gamification for engaging computer science students in learning activities: A case study. *IEEE Trans. Learning Technol. IEEE Transactions on Learning Technologies*, 7(3):291–301, June 2014. URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6827214.

[10] D. G. Oblinger. The next generation of educational engagement. *Journal of Interactive Media in Education*, 8(1):1–18, May 2004. URL http://www-jime.open.ac.uk/articles/10.5334/2004-8-oblinger/.

[11] I. M. Devonshire, J. Davis, S. Fairweather, L. Highfield, C. Thaker, A. Walsh, R. Wilson, and G. J. Hathway. Risk-based learning games improve long-term retention of information among school pupils. *PLoS ONE*, 9(7):1–9, July 2014. URL http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0103640.

[12] C. Malliarakis, M. Satratzemi, and S. Xinogalos. Designing educational games for computer programming: A holistic framework. *The Electronic Journal of e-Learning*, 12(3):281–298, 2014. URL www.ejel.org/issue/download.html?idArticle=288.

[13] A. T. Corbett, K. R. Koedinger, and J. R. Anderson. Intelligent tutoring systems. *Handbook of Human-Computer Interaction*, 2(37):849–883, September 1997. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.5216&rep=rep1&type=pdf.

[14] R. S. Baker, A. T. Corbett, and K. R. Koedinger. Detecting student misuse of intelligent tutoring systems. *Lecture Notes in Computer Science*, 3220(1):531–540, 2004. URL http://www.cs.cmu.edu/~rsbaker/BCK2004MLFinal.pdf.

[15] D. C. Merrill, B. G. Reiser, M. Ranney, and J. G. Trafton. Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. *The Journal of the Learning Sciences*, 2(3):277–305, November 2009. URL http://www.jstor.org/stable/1466610?seq=1#page_scan_tab_contents.

[16] K. R. Koedinger, J. R. Anderson, W. H. Hadley, and M. A. Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8(1):30–43, 1997. URL http://ctat.pact.cs.cmu.edu/pubs/Koedinger-Anderson.pdf.

[17] J. R. Anderson, C. F. Boyle, A. T. Corbett, and M. W. Lewis. Cognitive modeling and intelligent tutoring. *Artificial Intelligence*, 42(1):7–49, 1990. URL http://act-r.psy.cmu.edu/wordpress/wp-content/uploads/2012/12/119CogMod_IntTut.pdf.

[18] Berkeley ai materials. http://ai.berkeley.edu/project_overview.html.

[19] Michael Sipser. *Introduction to the Theory of Computation*, volume 1 of *1*. Course Technology, 25 Thomson Pl, Boston, MA 02210, 3 edition, 7 2012. ISBN 978-1133187790.

[20] C. H. Crouch and E. Mazur. Peer instruction: Ten years of experience and results. *American Journal of Physics*, 69(9):970–978, September 2001. URL http://dx.doi.org/10.1119/1.1374249.

[21] N. Lasry, E. Mazur, and J. Watkins. Peer instruction: From harvard to the two-year college. *American Journal of Physics*, 76(11):1066–1069, November 2008. URL http://mazur.harvard.edu/sentFiles/Mazur_61464.pdf.

[22] M. Keppell, E. Au, A. Ma, and C. Chan. Peer learning and learning-oriented assessment in technology-enhanced environments. *Assessment & Evaluation in Higher Education*, 31(4):453–464, August 2006. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.515.3600&rep=rep1&type=pdf.

[23] Y. Zhou, R. Freedman, M. Glass, and M. W. Evens J. A. Michael, A. A. Rovick. Delivering hints in a dialogue-based intelligent tutoring system. *Association for the Advancement of Artificial Intelligence*, 16(1):128–134, July 1999. URL https://www.aaai.org/Papers/AAAI/1999/AAAI99-019.pdf.

[24] T. Barnes and J. Stamper. Toward automatic hint generation for logic proof tutoring using historical student data. *Intelligent Tutoring Systens*, 5091(9):373–382, June 2008. URL http://www.ifets.info/journals/13_1/2.pdf.

[25] M. Nikolić. Measuring similarity of graphs and their nodes by neighbor matching. *Intelligent Data Analysis*, 16(6):865–878, November 2012. URL http://argo.matf.bg.ac.rs/publications/2011/similarity.pdf.

[26] Measuring graph similarity using neighbor matching. https://wadsashika.wordpress.com/2014/09/19/measuring-graph-similarity-using-neighbor-matching/, 2014.

[27] A. T. Corbett and J. R. Anderson. Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, 4(1):253–278, 1995. URL http://liris.cnrs.fr/~pchampin/2014/m2iade-ia2/_static/893CorbettAnderson1995.pdf.

[28] R. S. Baker, A. T. Corbett, and V. Aleven. More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. *Proceedings of the 9th international conference on Intelligent Tutoring Systems*, 9(1):406–415, 2008. URL http://repository.cmu.edu/cgi/viewcontent.cgi?article=1006&context=hcii.