Teaching Computers to Win Games

by Tracey Surrett, 2014 CTI Fellow
Irwin Academic Center

This curriculum unit is recommended for:
(Math/Technology/3-5th Grade)

Keywords: Tic-Tac-Toe, coding, programming, computer science, mathematical practice, mathematics, problem-solving

Teaching Standards: See Appendix 1 for teaching standards addressed in this unit. (Insert a hyperlink to Appendix 1 where you've stated your unit's main standards. For directions on how to insert a hyperlink, see Fellows Handbook, p. 29.)

Synopsis: Computers programmed with varying forms of artificial intelligence are all around us. Computers are found in our cars, they generate advertisements targeted to you based on your internet browsing history, and more.

Children of the 21st century do not know a world without computers being a part of their everyday lives. Many of them have been using smartphones and tablets since before they entered preschool.

This unit will explore various methods in which a computer program can be written to create an algorithm to play Tic-Tac-Toe successfully. A simple game was chosen to make the processes easier to comprehend for elementary students. Some activities have students actually programming using a tool created for children, while others replicate the coding concepts without actually programming a computer.

I plan to teach this unit during the coming year in to 24 students in 4th grade.

I give permission for the Institute to publish my curriculum unit and synopsis in print and online. I understand that I will be credited as the author of my work.

Teaching Computers to Win Games

Tracey Surrett

Introduction

We're in an exciting time for computer science and artificial intelligence. Right now, artificial intelligence appears in everything from the recommendations Netflix makes for something you might enjoy watching to Google's driverless cars. Computers are programmed to play games, not just well enough to follow the rules, but well enough to win against champion players in chess and Jeopardy.

My students are surprisingly savvy users of various forms of technology, sometimes to the point that they code programs in their free time. Those who do not have such a high level of familiarity with how the technology works still perk up when I ask them to take out their devices, pass out the school-owned iPads, or announce a surprise trip to the computer lab. This unit will introduce the concepts behind how programmers are able to create computers that play games successfully, even becoming better players as the game is played again and again.

I currently teach fourth grade at Irwin Academic Center, a Learning Immersion/Talent Development Magnet school. My entire class consists entirely of Talent Development students. As a magnet school, our students come from every part of Charlotte. Beginning in kindergarten, students are part of the Learning Immersion program, and others join the school as third graders after having been identified as Talent Development at their neighborhood schools in second grade. The premise of the school is that the Learning Immersion program nurtures students' abilities from kindergarten by using teaching strategies designed for gifted students, and providing advanced learning opportunities for students throughout the day in their classroom. Then, beginning in third grade, students who have been identified as gifted are placed in homogeneous classes for the Talent Development program. Because of the high level of achievement typically found in these classrooms, students are able to learn the curriculum at a deeper level then they would in a traditional classroom. This method allows students to be with their gifted peers throughout the day, in every subject, rather than only having that level of rigor in certain subjects or only during their scheduled time to be pulled out of the regular classroom by the Talent Development teacher. Students who attend the school in kindergarten through second grade, but do not qualify for the Talent Development program, continue as part of the Learning Immersion group also benefit from these teaching strategies and rigorous expectations throughout their time at the school. Our school also has a partnership with Discovery Place, allowing our students to take part in a free afterschool program led by a Discovery Place employee who serves as a liaison between the museum and our school. This employee is also available during the school

day to share ideas for teaching scientific concepts, share materials from Discovery Place, and lead some lessons in classrooms.

This unit is designed for my fourth grade Talent Development students, and will be used during math instruction to meet the Common Core Standards for Mathematical Practice[i]. I believe that any student, from third grade and up, could successfully complete the thinking exercises and activities in this unit. Younger students could do many of them with support and modifications to simplify the activities.

I will teach this unit during my math block, integrating basic concepts of coding along with the Standards for Mathematical Practice. It will take 10-12 class periods to complete the unit. Students will utilize the engineering process and scientific method to create and test ideas for solving problems presented in each activity. The Standards for Mathematical Practice include many ideas behind logical thinking that students need to fine-tune at this age to successfully problem solve in math and other areas of their lives.

Rationale

Our students need to be 21st century learners, and this unit will address several of the skills cited in the Framework for 21st Century Learning[ii]. Among those this unit will cover are: Critical Thinking and Problem Solving, Communication and Collaboration, and Information, Communications and Technology Literacy. I opted to address these standards through principles of computer science and artificial intelligence due to the high interest level of these topics to my students, and to provide a real-world context for problem-solving. These standards will be addressed through collaborative work on activities, discussion of ideas as a class and in small groups, learning some basic concepts of coding, and solving specific problems for each activity. In addition, students will utilize many of the Standards for Mathematical Practice to identify the problem they need to solve, use logical reasoning and critique ideas, and reasoning abstractly and quantifiably. Students will also utilize the scientific method and engineering design process to plan and test their solutions.

Objectives

My overall goal for this unit is for students to improve their ability to utilize creative and logical thought processes to solve problems collaboratively. Students are already familiar with the scientific method and engineering design process through our science labs and interaction with our Discovery Place liaison both in the classroom and in our afterschool program. Students will combine these problem-solving strategies with the Common Core standards for mathematical practice and ideals from the Framework for 21st Century Learning. Students will learn basic coding concepts and types of algorithms programmers use to program computers to not just play games, but to play them well.

Background Information

Classroom Expectations

Students participating in this unit will need to be familiar with the scientific method and engineering design process. All computer science skills necessary will be taught throughout the unit, but students will need access to a computer or mobile device for some activities.

Rules will need to be in place for sharing and critiquing ideas in both whole group and small group discussions, along with expectations for working in small groups in general. Expectations such as ensuring that everyone is participating, and that ideas are always received with respect and critiqued with kindness will be essential in keeping this a positive experience for all students.

Coding Terminology

Command

A direction to complete a specific action.

Loop

A direction to complete an action, or a series of actions repeatedly. A loop can be set to run a certain number of times, until a specific condition is met, or indefinitely.

If-Then Statement

A direction to complete a command only if a specific condition is met.

If-Then-Else Statement

A direction to complete one set of commands if a specific condition is met, and a different command if that condition is not met.

Game Tree

A diagram that shows all of the possible moves for each turn on the same row, with possible subsequent moves branching out of the resulting configuration of the game board.[iii] It is used to determine specific moves that will result in wins, losses, or draws as far in advance as possible to encountering those moves. The more types of pieces and places to go on the game board, the larger a game tree will be for that game. Games with

extremely large game trees are difficult to solve completely due to the sheer number of possibilities.

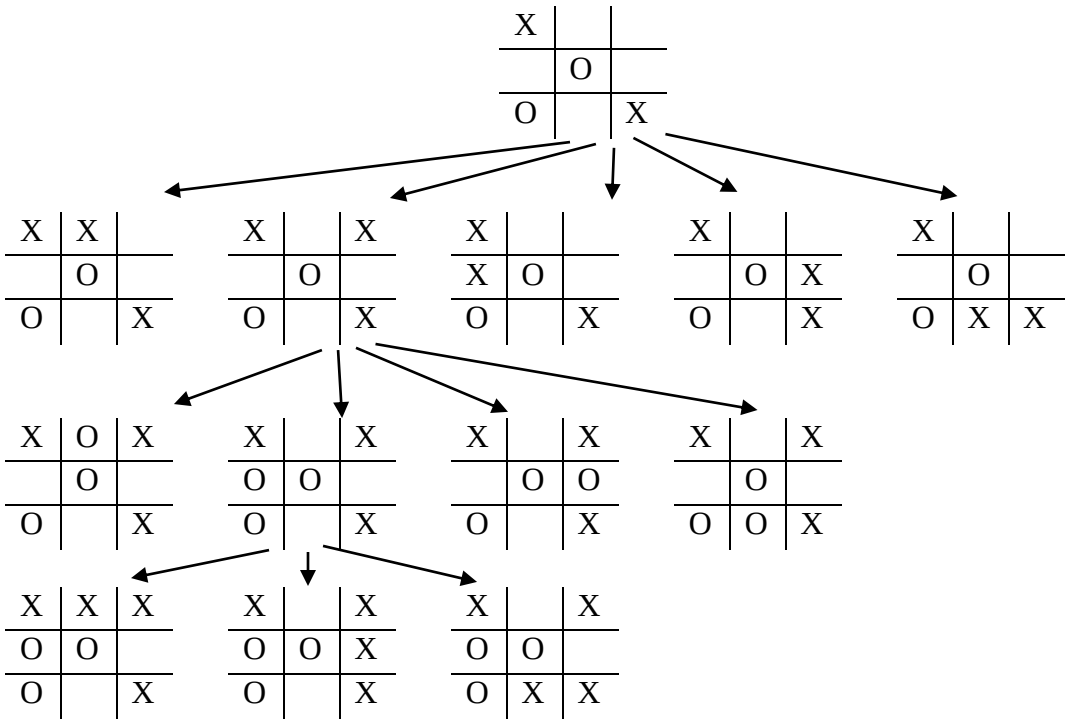


Figure 1

Evolutionary Algorithm

An algorithm that utilizes the idea of "survival of the fittest" to solve a problem. This algorithm is a reasonably effective manner in which to solve guess and check problems by creating random solutions and assigning each one a score based on specific criteria. Pairs of solutions with high fitness scores are spliced to create offspring that include traits from both parent solutions.[iv] The premise is, if two solutions are reasonably effective, a combination containing aspects of each solution might end up being even more effective. This type of algorithm tends to converge on a solution slowly, and is not the best method for every problem.

Reinforcement Learning

An algorithmic way to build machines that learn. The agent is rewarded for displaying correct behavior and/or punished for displaying undesirable behavior. This type of algorithm makes a computer's strategy more effective over time by systematically decreasing the likelihood that the computer makes sub-optimal choices and increasing the likelihood that it makes optimal choices, based on outcomes.

Teaching Strategies

This unit contains a combination of whole group instruction, interactive demonstrations, class discussions where students turn-and-talk with a partner to think through initial ideas before sharing with the class or working independently, and small group lab activities in which groups of 3-4 work together to solve a specific problem or perform a specific skill. Working collaboratively will help to scaffold students who are not confident with the coding concepts being presented, and will allow students with more prior knowledge in this area to take on leadership roles.

As we transition from discussing what constitutes artificial intelligence and the basics of coding to making our own primitive programs, students will use a variety of materials and strategies, discussed below.

Socratic Seminars

Socratic Seminars are discussions among students where they have the opportunity to share ideas and opinions, and others may respectfully agree or disagree, sharing alternative strategies or opinions. This is already a strategy students are familiar with as we use it frequently to discuss reading. They understand the guidelines of allowing everyone the opportunity to speak, that they are to speak to the group rather than to the teacher, and that everyone's ideas are valid and to be respected. Students are welcome to share new ideas or piggyback off of a previous statement by saying that they agree or disagree and why.

Science Technology Engineering and Math (STEM) Journals

Students will maintain journals of their work and reflect upon their thinking as they complete each activity. Some tasks will require students to make notes as they work, while others will require students to respond to open-ended questions after an activity or discussion.

Problem-Solving

Students will utilize problem-solving strategies and basic coding concepts using Scratch. It is free programming tool intended for children aged 8-16. It can be run in a browser, but is also available as a stand-alone program and as an iTunes app. There is a version intended for younger children (ages 5-7) called ScratchJr[v] if you work with primary age students. Students will get a taste of real programming through explorations with this program.

Classroom Activities

Socratic Seminar – What is artificial intelligence?

Students will be posed the following questions to begin thinking about artificial intelligence.

- What do you think of when you hear the term "artificial intelligence"?
- How is artificial intelligence often shown in movies, books, or video games?
- What are some examples of artificial intelligence you use in your life?

Following the seminar, students reflect upon the discussion in their STEM journals. This is an opportunity for them to react to the discussion, and to ask or answer questions that were not included in the class discussion.

Introduction to Coding

Following Directions As a Computer

Tell the class that computers are only able to follow a given set of instructions. This set of instructions is called a program, and it is comprised of a series of steps to follow called commands. Break the class into groups of three. In their STEM journals, each group must create a program that a person can follow to put on a coat. Students are encouraged to attempt to follow their program as they write to ensure the accuracy of their instructions.

When all groups are finished writing their program, the teacher attempts to follow each one, but remembering to follow each direction in the least cooperative manner possible. Follow the directions very literally, and intentionally misinterpret vague directions. For example, if directed to put your arm through the sleeve, put the wrong arm in so the coat will be on backwards, or push your hand through the cuff rather than the arm scythe, resulting in the coat be put on inside out.

After one or two programs are unsuccessful, allow students to revise their instructions before attempting to continue executing more programs. After attempting to execute each group's program, have students reflect in their STEM journals about the activity with questions such as:

- What did you learn about how computer programs work?
- What do you still wonder about computer programs?
- How could your program be improved further?

Scratch Session 1

The teacher will give students an overview of the commands used in Scratch in the scripts subsections: motion, looks, sounds, and events. Next, the teacher will demonstrate how to use some of these simple commands based on ideas presented early in the Scratch Curriculum Guide[vi], such as walking, changing costume, playing a sound, and starting the program when the green flag is clicked. Ultimately, students are creating a game of tag, but the assignment will be given in a series of steps.

Using their newfound understanding that computers can only follow the directions they are given, students are initially instructed to create a program using one sprite, and moving that sprite in a path in the shape of any polygon. Have students turn-and-talk with their neighbors to share ideas about what commands will probably be the most successful at this task, and how the program should differ for each polygon they are considering using. Once students have their ideas from their partner talk, have students sketch out their plans for their programs in their STEM journals. This may consist of a sketch of the shape they intend to use for the path, and ideas for which commands they probably need to include. Students who have completed their plan are allowed to begin writing their program in Scratch. Encourage students who finish early to experiment with additional changes to their sprite through the program, but make sure everyone is at least able to make their sprite move in a path in the shape of a polygon.

Once all students have created their first program, as a mid-workshop interruption, pose the question of how they could make their sprite do the same group of tasks more than once. For example, how could they change the program to make their sprite move in that same path repeatedly? Introduce the idea of a loop, which allow a series of actions to repeat a given number of times, until a specific condition is met, or indefinitely. This is the type of command that allows a Roomba to vacuum an entire room without falling down the stairs or getting stuck in a corner. Point out the control and sensing subsections within the scripts tab in Scratch, especially the three commands in the control subsection that allow for a loop to repeat a given number of times, and indefinitely. For the loop to complete until a certain condition is met, the sensing subsection will also need to be used, dragging one of the sensing controls into the hexagonal opening in the block for the loop. Students should alter their program to include a loop, preferably one that lasts either a set number of times or infinitely for the sake of simplicity.

In their STEM journals, students should reflect on today's activity with questions such as:

- What are some other situations in which a loop could be helpful to programmers?
- What was the most difficult part of today's task?
- How has your understanding of how a computer program works changed from today's activity?

Scratch Session 2

At the beginning of the second session using Scratch, introduce the concept of if-then statements. Discuss situations in which you might want a command followed only in certain situations. This allows the user to make the computer only perform a task if a certain condition is met. If that condition occurs, then the command is followed. In other situations, the command is ignored. For example, when adding, you only increase the tens place by one if the total of your ones place is any number from 10 and 19. Model creating a simple program with an if-then statement.

Students will be adding on to their program from their first session of Scratch. Next, students need to add another sprite to their program, but make sure the new sprite has at least two available costumes. The second sprite must also move in a path in the shape of a polygon, with the commands set in a loop, but its path must be a different shape than that of the original sprite. Once that is in place, an if-then command must be added that causes the new sprite to change costume each time it touches the original sprite. Because of the nature of the command, students need to choose path shapes and the placement their sprites in a way that will result in the second sprite changing costume at least once. Again, students should turn-and-talk with a partner to consider initial ideas, and then plan in their STEM journals before actually writing their programs.

As a mid-workshop interruption, pose the question of what a programmer would need to do if they want one command followed if one condition is met, and a different command followed in any other situation. They could create two or more separate if-then statements, or a single if-then-else statement. The teacher models creating a program with an if-then-else statement. Students then need to alter their programs to make the second sprite change costume if it touches the first sprite, and to say "Coding is cool!" if the two sprites are not touching.

In their STEM journals, students should reflect on questions such as:

- What have you learned about programming so far?
- What questions do you still have?
- What was the most challenging part of this task?
- What do you want to improve about your program?

Creating a Tic-Tac-Toe Algorithm

Exploration

Students will need to break into pairs to play Tic-Tac-Toe. As they play, they need to name the different moves that can be made. Some moves may have specific names, describing what should be done in relation to that player's last move or the opponent's last move, or they may be very general, such as, "go in a corner." Once students believe

they have a complete list of commands, compare as a group and come up with an agreed-upon wording for each direction that the entire class will use. Keeping those commands posted, or printing a copy for each group to keep handy, make students play a few more games, trying to use only those commands to describe their moves. If they wish to make a move that they cannot describe adequately with the given language, they can create a new command and add it to their list. After all groups have played an additional 2-3 games, take a few moments to discuss whether new commands need to be added, agree upon their wording, and add them to the class's approved list of commands.

Creating the Algorithm

In advance, determine the number of groups that students will need to be divided into. You will need an equal number of groups creating an algorithm for player one and player two. Students should be placed into groups of 2-4, using the agreed upon commands to create an algorithm that never loses a game of Tic-Tac-Toe.

Each team will need a copy of the list of commands printed once for each of the five turns, a set of number cards 1-9 to serve as a random number generator, and will need to know whether they are player 1, who will play as X, or player 2, who will play as O. A game board for Tic-Tac-Toe with each square numbered 1-9 will be needed for each pair of teams as well.

Students will essentially create an if-then-else statement for each turn by making a pile of commands in the order in which they should be played. If the first command cannot be executed, then the computer attempts to follow the second command, and so on until either a move can be made, or there is no command to play, resulting in a loss. Students must number their cards with the turn number, and the number the order of the cards within that turn for their algorithm. Students do not need to utilize every command, but are allowed to do so. If a command does not include a specific location, the number cards that serve as a random number generator are used to determine which place in which their mark will be placed. For example, if a team uses the command, "Go in a corner," the random number generator cards must be used, drawing one card at a time until the number of an available corner has been drawn. Once a team for player 1 and a team for player 2 are ready to test their results, they play a game against one another using only their algorithm. If they are not satisfied with the results, they may return to their group to revise the algorithm before reattempting a game.

Once an algorithm has been tested for 10 games and never results in losses, it may be considered successful. It is helpful to change opponents as frequently as possible while testing the algorithm to ensure that it was not only successful against that particular opponent.

In their STEM journals, students should reflect on questions such as:

- Keeping in mind that if both players are playing at their best, Tic-Tac-Toe will always result in a draw, how successful was your algorithm?
- Would this method have worked as well for you if you had never played Tic-Tac-Toe before today? Why or why not?
- In this activity, you took information you knew, and systemically created instructions that a computer could follow, based on what you do when deciding what move to make in Tic-Tac-Toe. How would this change if you were playing a more complex game, such as chess or backgammon?

Using Evolutionary Algorithms to Create a Self-Learning Tic-Tac-Toe Agent

Introduction

The concept of how a computer is programmed with evolutionary learning to become better at a task, in this case playing Tic-Tac-Toe, will be illustrated through this activity. This is intended to work as an interactive demonstration.

For this activity, the teacher will need to prepare in advance the following set of materials for each team of 2-3 students:

- The student-generated commands from the previous activity, or the commands found in, "Winning games: the perfect tic-tac-toe player,"[vii] in the Teacher Resources section. Each command needs to be labeled with a number prior to making copies.
- A Tic-Tac-Toe board with each space numbered
- A set of number cards 1-9 to randomly determine placement for vague commands, or use of the website http://www.random.org[viii]
- A large number of copies of the recording sheet

An evolutionary learning system works by rewarding successful combinations of moves (ones that lead to wins and draws) with the opportunity to reproduce, while unsuccessful combinations of moves (ones that lead to losses) do not. The value of a combination of moves works as follows from greatest to least: win in three turns, win in four turns, win in five turns, draw, and loss. The scoring system is in place because in the idea of survival of the fittest, the most successful reproduce and pass on their DNA. A combination resulting in losing the game will not reproduce. This is equivalent to an animal not getting enough food and dying off before it has the chance to reproduce. The level of fitness of each algorithm is determined by the result of the game for player 1. An algorithm is rated based on not only the final outcome of a win, loss, or draw, but also rewards the speed at which the agent wins the game. While all algorithms that result in a win or a draw are eligible to reproduce, only the ones with the best fitness level available would be chosen to do so.

Successful combinations of commands are spliced with one another to create the next generation. Each successful combination of commands only reproduces one time, resulting in two "children" in the next generation. Once offspring are produced, the parents are retired from the activity.

Evolutionary Learning Activity

Each pair of students will need a recording sheet featuring a Tic-Tac-Toe board and a place to mark the commands drawn for each turn. One student plays as a human for player 2 (O), the other plays as a computer for player 1 (X). A combination of commands followed by player 1 throughout the game will be referred to as their answer.

The recording sheet for each algorithm used to play X includes the generation number and the game number at the top. The first generation will be determined randomly, using Random.org to generate numbers ranging from 1-10. Whether or not a command can be executed, each command drawn is recorded as part of the algorithm. Once the first generation is populated, successful algorithms will be paired off to reproduce, creating a second generation. Once the second generation has been created, first generation algorithms are retired. The process of successful algorithms reproducing, then being retired while the algorithms from the newest generation are used as agents for player 1 in an attempt to create the most effective algorithm available.

Beginning with generation 2, students will also need to note which generation and game number the "parent" answers were. The Tic-Tac-Toe board on the recording sheet is where the game should be played. The turn numbers indicated in the table beneath the Tic-Tac-Toe board is where the list of command numbers drawn is written for each turn, in the order in which the commands are drawn from the stack. In the worked examples, the commands from "Winning games: the perfect tic-tac-toe player," were used, and numbered as follows:

1. Go for three in a line.
2. Block three in a line.
3. Go in the opposite corner to my last corner move.
4. Go in the opposite corner to the other player's last corner move.
5. Go in a free space.
6. Go in a free corner.
7. Go in the center.
8. Go in an edge square (not a corner).
9. If the other player holds opposite corners, then go in an edge (not a corner).
10. Go in a corner on the same side as my last corner move.

Generation #_1__ Game #_6__

Child of Generation #_N/A___ Games # ____ & ____

X	X	O
O	O	X
X	X	O

Sequence of Commands

Turn 1	Turn 2	Turn 3	Turn 4	Turn 5
3, 5	5	8	7, 9, 2, 8	5

Points: __1___

Figure 2

As seen above in Figure 2, begin by marking the sheet with the generation number, along with labeling each game with a specific number within that generation. Mix up all of the command cards, and draw the top card. If that command can be executed, that is the move played. If the command cannot be followed due to the current state of the board, note the number of that command on the recording sheet and draw the next command card, repeating both of these steps until a command can be completed. Note the number of the command followed on the recording sheet. At the start of each new turn, mix all the number cards back into the stack so the same command could be followed for more than one turn. Repeat these steps until the game ends. At the end of the round, note the number of answers with each point value.

Once the first round of games ends, determine the value of each algorithm's answer to how to play Tic-Tac-Toe with the following criteria: a win in 3 moves is 4 points, a win in 4 moves is 3 points, a win in 5 moves is 2 points, a draw is 1 point, and a loss has 0 points. Have students play several rounds of Tic-Tac-Toe in the manner explained above to create a large first generation. As each game ends, find the value of the algorithm followed. There needs to be enough algorithms with point values of 1 or higher for each student before moving to the next step. If you have an odd number of students, partner two of them to share one algorithm to allow each algorithm from the first generation eligible for reproduction to do so.

Every answer that reproduces must have a value greater than zero, and must have another answer with which to reproduce in order to pass on its chromosome. These answers are going to reproduce by splicing the first and second portions of each parent's command sequence by rolling the die. The number the die lands on indicates the point at which the answer, which here serves as the chromosome, is spliced. For instance, if a 3 is rolled, as in figure 3 below, the splice occurs after the third turn. To splice the parents'

chromosome, draw a line where the splice occurs, and copy the correct portion of the command sequence from each parent. In figure 3, color coding was used to visualize the process more clearly. For child 1, (yellow in figure 3) the first portion of the genes come from parent 1, and the second portion comes from parent 2. Child 2 (green in figure 3) has the opposite; the first portion is from parent 2, and the second portion of the genes come from parent 1. See the worked example below. This process must be repeated for each pair of "parent" answers until each parent answer has been replaced with a child answer that is a combination of its parents.

Generation 1, Game 6 (Parent 1)				
Turn 1	Turn 2	Turn 3	Turn 4	Turn 5
3, 5	5	8	7, 9, 2, 8	5

Generation 1, Game 7 (Parent 2)				
Turn 1	Turn 2	Turn 3	Turn 4	Turn 5
5	7, 6	2	4, 1, 8	8

Generation 2, Game 1 (Child 1)				
Turn 1	Turn 2	Turn 3	Turn 4	Turn 5
3, 5	5	8	4, 1, 8	8

Generation 2, Game 2 (Child 2)				
Turn 1	Turn 2	Turn 3	Turn 4	Turn 5
5	7, 6	2	7, 9, 2, 8	5

Figure 3

As a demonstration, splice the answers together as shown in the description and worked example above. Make a stack of all available algorithms, sorted from highest point value to lowest point value. Take the top two answers from the stack and pair them up. The next two highest point values are paired next, and so on until every algorithm with a fitness score of 1 or greater has been partnered with a similarly valued answer. If there is an odd number of answers available, one with the lowest point value that is eligible to reproduce will not have a mate. Once the answers are partnered up with as similar a point value as possible, they will need to be spliced to create two second-generation answers. This process will be done for each pair of answers eligible for reproduction. Keep the first generation responses intact to study later.

For the next round of games, teams will adopt one of the generated children as their algorithm for player 1, while the role of player 2 i1s played by a human. If there were a series of commands to follow for a single turn, it is used as an if-then-else statement. If the first command can be followed, follow it, else move on to the next command in the sequence. If a command from the sequence cannot be followed and the game has not ended, that player has lost the game. If an adopted algorithm does not have a command for a turn number (for example, it was generated from two parents who won on turn three, and therefore does not have a command for turns four and five), then the player cannot make a move for that turn and forfeits the game. A forfeit counts as a loss. At the end of this round, again determine each answer's value, and record the number of

answers at each point value, keeping the data sorted by generation. Once the data are tallied, pair teams with the highest valued answers to create third generation (G3) answers.

This process is repeated until the group is satisfied that continuing the process will not result in solutions more effective than the ones already found, or there are no longer enough answers in that generation to reproduce. Make sure to keep track of the number of answers at each point value for each generation.

Analyzing Data

Taking the data that was found from the evolutionary learning demonstration, graph the generation number as the x-axis, and the total number of wins as the y-axis. Any generation where there are more losses than wins will result in negative numbers on the y-axis. Connect the plotted points to show the learning curve of the algorithm. In another color, follow the same procedure to graph the total number of algorithms created in each generation.

Pose the question of what students noticed about the shape of the graph, and what that means for how well the computer played the game at different points in the process of learning to play effectively. What factors seemed to contribute to the most successful round(s)? What about the least successful round(s)? Does this seem like an effective strategy to learn to play this particular game? What might it be more effective for?

Have students reflect in their STEM journals on questions such as:

- Does this seem like a reasonable way to program a computer to learn to play Tic-Tac-Toe? Why or why not?
- Did the algorithm become better over the course of several generations, or did it simply stop working? Why do you think that happened?
- What conclusions can you draw from this activity about how this type of algorithm works for this type of problem?

Using Reinforcement Learning to Create a Self-Taught Tic-Tac-Toe Agent

A reinforcement learning algorithm attempts to discover which moves will result in losses and which moves will result in wins or draws by exploring different game situations through play. When the computer determines what move lost the game, that move is deleted as a possible option in that scenario, effectively ensuring that the mistake is not repeated. A variation of this exercise can be to positively reinforce selecting moves that result in wins and draws by increasing the likelihood of choosing one of those moves. Putting another piece of candy or paper with a successful command number in

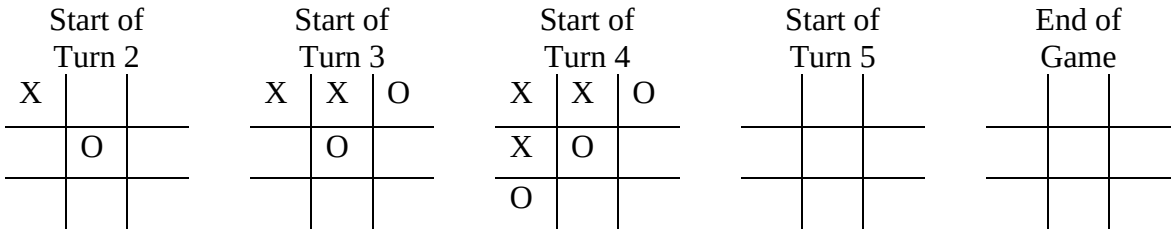
the bag of commands from which to choose would increase the likelihood of randomly selecting a successful command.

Reinforcement Learning Examples

Use the following game boards and the description below as a demonstration of how reinforcement learning works. In the worked examples, the commands from "Winning games: the perfect tic-tac-toe player," were used, and numbered as follows:

1. Go for three in a line.
2. Block three in a line.
3. Go in the opposite corner to my last corner move.
4. Go in the opposite corner to the other player's last corner move.
5. Go in a free space.
6. Go in a free corner.
7. Go in the center.
8. Go in an edge square (not a corner).
9. If the other player holds opposite corners, then go in an edge (not a corner).
10. Go in a corner on the same side as my last corner move.

Game #1



	Turn 1	Turn 2	Turn 3	Turn 4	Turn 5
Possible Commands	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1, 2, 3, 4, ~~5~~, 6, 7, 8, 9, 10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Commands Followed

	Turn 1	Turn 2	Turn 3	Turn 4	Turn 5
Command	6	8	5		
Result			LOSS		

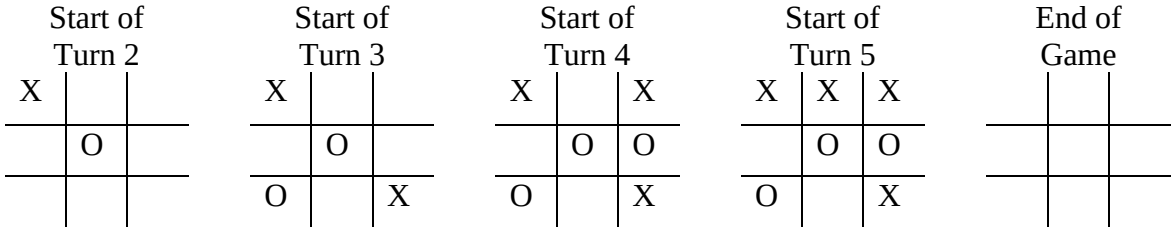
Figure 4

Look at the worked example of game #1 above. In reinforcement learning, it is determined whether a move was good or bad after the move has been made and its effect on the overall result of the game can be determined. This is initially done by simply noting the outcome of the game after the last move was made. As the computer explores

more of the game space, it is able to prune the game tree to eliminate moves from board configurations if that move causes the computer to lose the game.

To determine which command is followed for each turn, the computer randomly selects a move from the list of 10 commands used previously, and if that move can be carried out, it is, regardless of the effectiveness of that move. If a command cannot be followed, it will be removed from the possible commands for that configuration of the game board, and another command will be randomly selected. The commands are faithfully executed until the game ends. The outcome of the game is then used to adjust the agent's strategic choices in future games. For example, in game #1, the move made on turn 3 was a bad move because it lost the game. The computer's mistake is corrected by removing command #5, "Go in a free space," from the list of possible commands on that configuration of the game board.

Game #2



	Turn 1	**Turn 2**	**Turn 3**	**Turn 4**	**Turn 5**
Possible Commands	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	**1**, 2, 3, 4, 5, 6, 7, 8, 9, 10	1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Commands Followed

	Turn 1	**Turn 2**	**Turn 3**	**Turn 4**	**Turn 5**
Command	6	3	2	1	
Result				WIN	

Figure 5

In game #2, the command followed in turn 4 won the game, making it a good move. If a rewards system is being used in this program, another paper or candy marked with command 1 would be added to the bag of possible moves for the configuration of the board at the start of turn 4. If only a punishment system is being used, then command 1 is safe from being pruned at this time because of its success in this configuration of the board.

Punishment-Based Reinforcement Learning Activity

Tic-Tac-Toe, even when accounting for rotational symmetry and reflections, has over 31,000 distinct games.[ix] To prevent the need to play quite that many games of Tic-Tac-Toe, this activity will have students constructing and studying the game tree of a game in progress. The commands used for this activity are from "Winning games: the perfect tic-tac-toe player," were used, and numbered as follows:

1. Go for three in a line.
2. Block three in a line.
3. Go in the opposite corner to my last corner move.
4. Go in the opposite corner to the other player's last corner move.
5. Go in a free space.
6. Go in a free corner.
7. Go in the center.
8. Go in an edge square (not a corner).
9. If the other player holds opposite corners, then go in an edge (not a corner).
10. Go in a corner on the same side as my last corner move.

You'll need at least 3 paper bags. On the front of each bag, draw the state of the game board at the start of that turn. Inside each bag, put 10 different types and/or colors of candies, each relating to a different command number listed above. Alternately, slips of paper numbered 1-10 or listed with each command can be used. Throughout this activity, a person will select moves for player 2, while the commands for player 1 will be randomly selected using the algorithm below.

Given the game tree in Figure 6 below, which is also found in the materials section, look at the first and second configurations of the game board in the row labeled, "Turn 4 for X." Because both of these moves resulted in a win, they are considered good moves. The third configuration of the game board in that row could result in either a win or a loss, depending upon the choices made by player 2. The game tree doesn't show which commands were followed to result in each of these configurations of the game board.

This activity will have students begin by playing turn 4 for X (player 1) as a computer. The board at the beginning of turn 4 will be the framed board in the row marked, "Turn 3 for O," in Figure 6 below. Get one of the paper bags and draw the configuration of the game board framed in the row indicated above. Then fill the bag with one of each candy determining the ten possible commands.

Next, have a student randomly select one of the candies from the bag. Can the move be executed? If it cannot, prune that command from this configuration of the game board by allowing the student who drew it to eat the candy. (If using numbered slips of paper, throw away that number.) If it can be executed, follow the command. If it is a command that requires a box to be selected, such as, "Go in an edge square," use Random.org or number cards 1-9 to randomly select the number of the Tic-Tac-Toe tile that will be used. If a number is selected that cannot be used (for example, the number for a corner is drawn

when the command states to go in an edge square), draw again until a number is drawn to a square that meets the criteria.

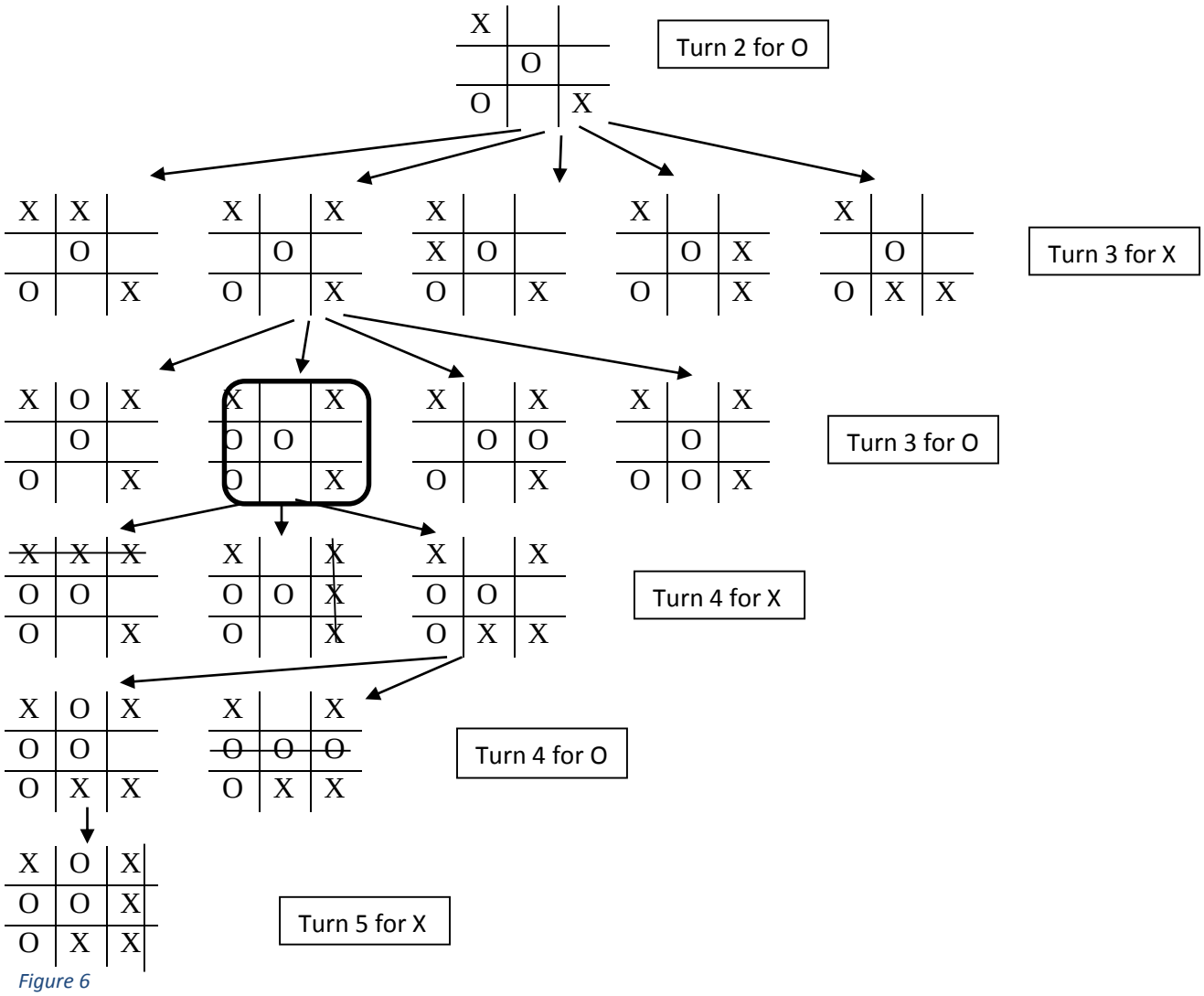


Figure 6

1	2	3
4	5	6
7	8	9

Figure 7

Once the command is followed, look at the game tree. Of the possible subsequent moves, will having followed this command ultimately result in a win, loss, draw, or is it uncertain at this time? For instance, in the game tree in Figure 6 above, placing an X in

box 2 or box 6 (when the game board is numbered as in Figure 7) both result in a win, while placing an X in box 8 can result in a win or a loss depending upon the choices of player 2. A person would look at the options for player 2 in turn 4, and know that it is unlikely that player 2 would go in box 2, but an agent cannot reason through it that way. The agent needs to lose due to the move selected for player 1 in turn 3 in order to eliminate that command from the list of options and avoid repeating the error.

If, based on the game tree, following the command selected ultimately results in a win or a draw for player 1, return the candy to the bag. If following that command ultimately results in a loss for player 1, allow the student who drew the candy to eat it. If following that command could eventually result in either a positive or negative outcome for player 1, return the candy to the bag and continue the game to completion. Eliminate commands resulting in a loss, and return commands to the bag that result in a win or a draw. Continue to test this configuration of the game board until all ten commands have been tested. In some cases, a command may need to be tested multiple times to account for the options player 2 has for his move.

In the event of an uncertain outcome, commands in later turns would be pruned first. If every command in a configuration of the game board results in a loss, the agent will move to an earlier turn on the game tree in an attempt to prune the command that resulted in a configuration of the game board in which losing was inevitable. Even in outcomes that are certain, the agent will eventually move up to earlier turns on the game tree to essentially look ahead more turns to determine which commands are overall good moves and which commands are bad moves for every turn.

X		X
O	O	
O		X

Figure 8

Just as the computer does, the next step is to have students move up a level of the game tree. Now that all of the moves resulting in losses after turn 3 when the game board is configured as shown above in Figure 8, students need to determine which commands need to be pruned from turn 3 for player 1. Figure 9, below, shows the game board after player 2 has made their second turn.

X		
	O	
O		X

Figure 9

At this point, if the computer selects a command that results in the game board being configured as it is in Figure 8, the agent already has commands pruned that result in losses, leaving only moves that will always result in a win or a draw. If the randomly selected command results in a configuration of the game board that the agent has not yet encountered, commands will be randomly selected and faithfully followed until the end of the game. At the end of the game, the final command will be judged to be a good move, and allowed to stay in the possible commands, or a bad move that is pruned from the list of possible commands for that board.

Get another bag filled with one of each of the ten candies correlating to the ten possible Tic-Tac-Toe commands. Draw the game board in Figure 9 on the front of the bag. Have a student randomly select a candy from the bag. If it results in the placement of the third X found above in Figure 8, return the candy to the bag for the moment. Move to the bag with the board configuration in Figure 8 drawn on it, and have a student randomly select a candy to determine which move to follow. If following that command for turn 4 results in a win or a draw, return the candy to its bag. If it results in a loss, prune the command from those available by eating the candy.

When the randomly selected moves for player 1 (X) on turn 3 results in a configuration of the board for which you do not have a game tree, create one as you play using the blank game tree found in the materials section. Remember to return candies to the bag that represent commands that resulted in wins and draws, and permanently remove candies from the bag that represent commands that resulted in losses. Once all possible moves for player 1 on turn 3 that result in a loss have been pruned, it's time to move back yet another level to determine the efficacy of each command for player 1 in turn 2.

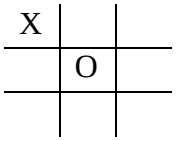


Figure 10

Get another bag filled with the 10 candies representing each possible command, and draw the board configuration shown in Figure 10 on the bag. Using the blank game tree for turn 2 in the materials section, select a command to follow for player 1 on turn 2, and complete the game tree to show all possible subsequent configurations of the board.

In their STEM journals, have students reflect on questions such as:

- How did the commands left after pruning the game tree compare to the commands you chose in the activity where you created generalized if-then-else statements for each turn?

- Does this method seem more efficient, less efficient, or similar to the other methods we have used to create Tic-Tac-Toe algorithms? Why?

Complete Map of Optimal Tic-Tac-Toe Moves

Does the skill of the opponent the computer faces make a difference in how efficiently the computer learns to play the game well? What if player 2 always makes the optimal move? Repeat the evolutionary learning simulation, but this time player 2's turn is always determined by the infographic, The Complete Map of Optimal Tic-Tac-Toe Moves[x]. Again, keep track of the data as you play and graph the results. Compare the graphs from the first trial; did the skill of the opponent make a difference in the speed at which the computer learned to play well?

Socratic Seminar – Reliable Ways to Code Machines to Win

Students will be posed the following questions to consider the best ways to reliably code a machine to win as frequently as possible:

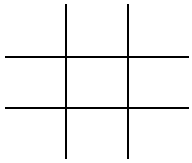
- Which method seemed a more reliable way to ensure that your "computer" won - when we created our own Tic-Tac-Toe algorithms, or when the computer was rewarded for playing well by being able to pass on those skills to its children? Explain.
- Thinking about more complicated games, such as chess, which method would you rather use if you needed to teach a computer to win?
- What if you wanted to code a computer to win a game that includes an element of chance, such as backgammon?

Materials

Tic-Tac-Toe Evolutionary Algorithm Activity

Generation #___ Game #___

Child of Generation #____ Games # ____ & ____

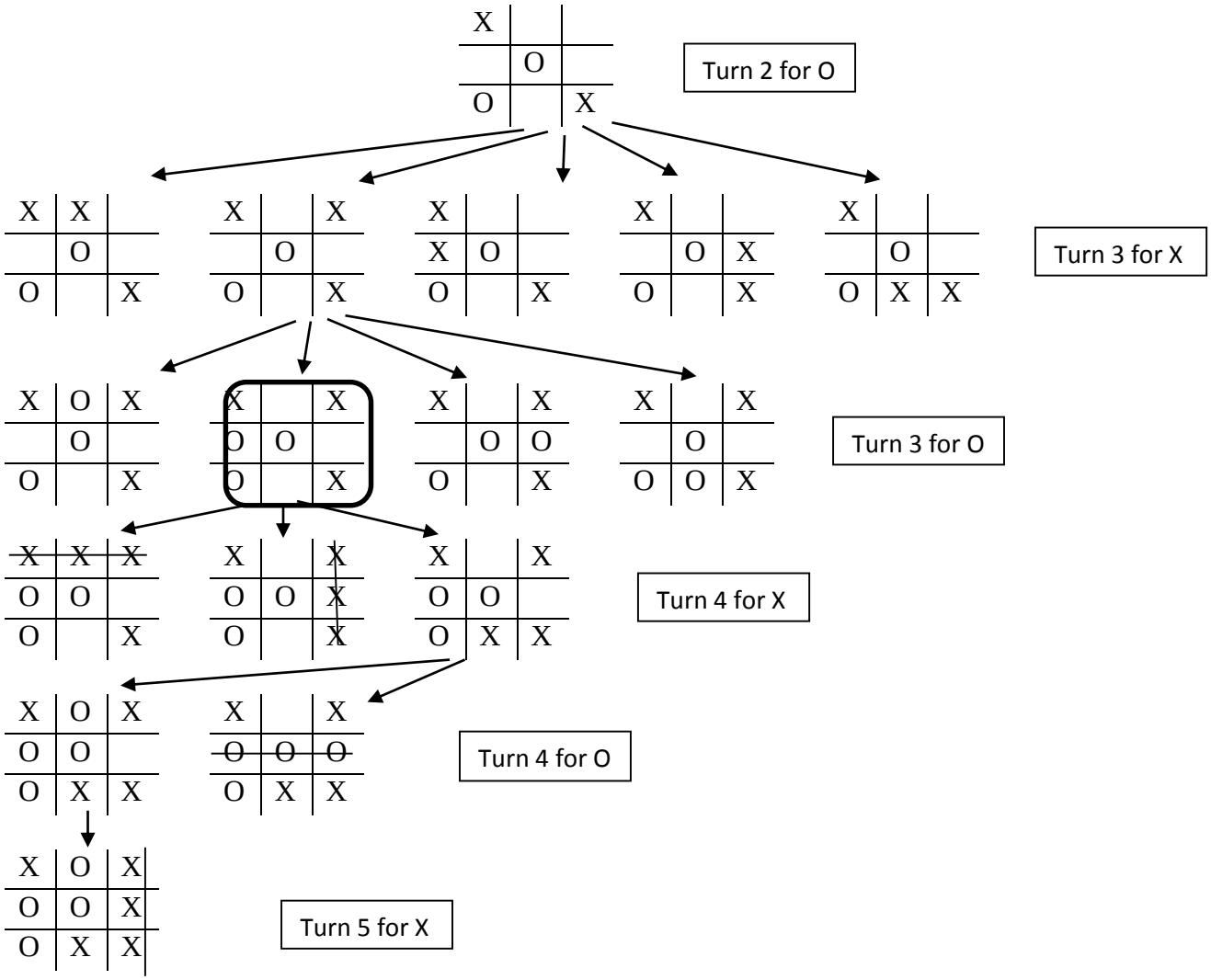


Sequence of Commands

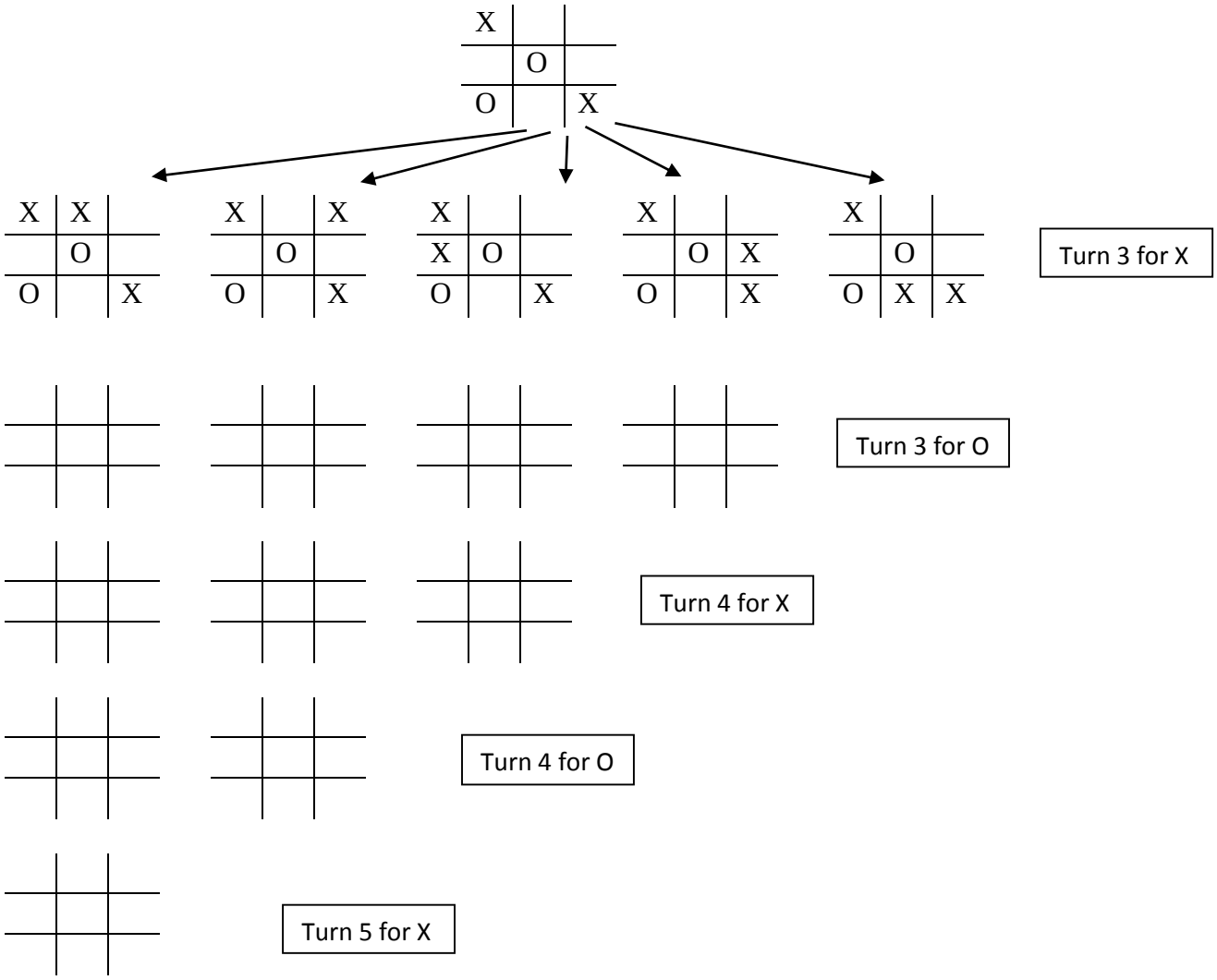
Turn 1	Turn 2	Turn 3	Turn 4	Turn 5

Points: _____

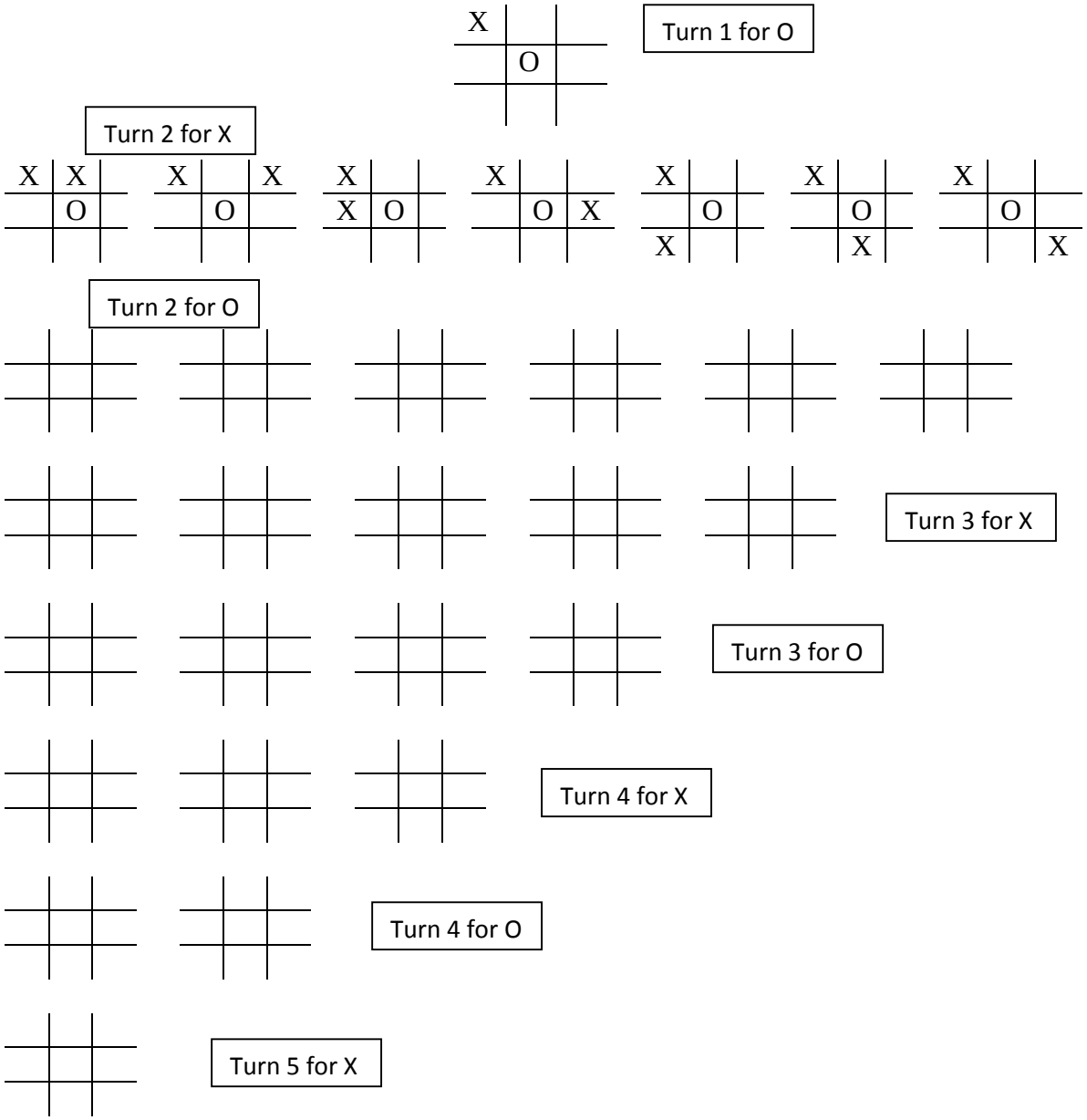
Tic-Tac-Toe Reinforcement Learning Activity



Additional Game Trees for Reinforcement Learning Activity (Turn 3)



Additional Game Trees for Reinforcement Learning Activity (Turn 2)



Appendix 1: Implementing Teaching Standards

Common Core Standards for Mathematical Practice

These standards are applicable to all grade levels, from kindergarten through twelfth grade. Rather than focusing on specific mathematical tasks, these standards concentrate on various aspects of problem-solving. While these standards are written with the intention of applying them to mathematical thinking, the skills learned from them can be generalized to help students to solve problems in general.

CCSS.Math.Practice.MP1 Make sense of problems and persevere in solving them.

CCSS.Math.Practice.MP2 Reason abstractly and quantitatively.

CCSS.Math.Practice.MP3 Construct viable arguments and critique the reasoning of others.

CCSS.Math.Practice.MP4 Model with mathematics.

CCSS.Math.Practice.MP5 Use appropriate tools strategically.

CCSS.Math.Practice.MP6 Attend to precision.

CCSS.Math.Practice.MP7 Look for and make use of structure.

CCSS.Math.Practice.MP8 Look for and express regularity in repeated reasoning.

Resources

Materials for Classroom Use

DevTech Research Group at Tufts University, the Lifelong Kindergarten Group at the MIT Media Lab, and the Playful Invention Company. n.d. *Scratch Jr.* Accessed October 2014. http://www.scratchjr.org/.
Scratch Jr. is a programming tool intended for children aged 5-7.

Lifelong Kindergarten Group at MIT Media Lab. n.d. *Scratch.* Accessed October 2014. http://scratch.mit.edu/.
Scratch is a programming tool intended for children aged 8-16.

McOwan, Peter; Curzon, Paul; support from EPSRC and Google. n.d. "Computer Science for Fun." *cs4fn.org.* Accessed October 15, 2014. http://www.cs4fn.org/teachers/activities/winatoxo/winatoxo.pdf.
This activity includes ten commands that can be used to describe any move in Tic-Tac-Toe. These are the commands used throughout the unit in the worked examples.

Munroe, Randall. n.d. "Complete Map of Optimal Tic-Tac-Toe Moves." *xkcd.* Accessed May 2014. http://imgs.xkcd.com/comics/tic_tac_toe.png.
Complete Map of Optimal Tic-Tac-Toe Moves is a comic that shows all of the optimal moves for both x and o in a Tic-Tac-Toe game. All moves are shown, but the optimal moves are shown in red to make them stand out.

Randomness and Integrity Services Ltd. 1998. *Random.org.* Accessed November 2014. http://www.random.org/.
This web tool can be used to randomly generate numbers, and is used several times throughout this unit in lieu of dice or number cards.

Bibliography for Teachers

Brennan, Karen, Christan Balch, and Michelle Chung. 2014. "Creative Computing: a design-based introduction to computational thinking." *ScratchEd.* August 7. Accessed September 2014. http://scratched.gse.harvard.edu/guide/files/CreativeComputing20141015.pdf.
Creative Computing: A Design-Based Introduction to Computational Thinking is a great resource for a teacher who needs to understand the basics of programming in order to model how to use Scratch for their students. It includes step-by-step lessons that can be used to introduce the different capabilities within Scratch, to a much deeper extent than was included in this unit.

Chu-Caroll, Mark C. 2008. *Scienceblogs.com.* July 30. Accessed October 20, 2014. http://scienceblogs.com/goodmath/2008/07/30/solving-tictactoe-game-tree-ba/.
This article includes diagrams of portions of the game tree for Tic-Tac-Toe, as well as an explanation of how game trees are used to predict the best move.

Common Core State Standards Initiative. n.d. *Common Core State Standards Initiative.* Accessed May 2014. http://www.corestandards.org/Math/Practice/.
This document details each of the eight standards for mathematical practice used for students in kindergarten-12th grade as part of Common Core State Standards.

Hochmuth, Gregor. 2003. "On the Genetic Evolution of a Perfect Tic-Tac-Toe Strategy." *genetic-programming.org.* Spring. Accessed November 4, 2014. http://www.genetic-programming.org/sp2003/Hochmuth.pdf.
This article utilized a genetic algorithm to evolve an algorithm for Tic-Tac-Toe.

Partnership for 21st Century Skills. 2011. "Framework for 21st Century Learning." *Partnership for 21st Century Skills.* March. Accessed May 2014. http://www.p21.org/storage/documents/1.__p21_framework_2-pager.pdf.
This document outlines the 21st century skills our students need to become proficient in to be successful in the future.

Schaefer, Steve. 2002. "Tic-Tac-Toe." *Mathematical Recreations.* January. Accessed November 15, 2014. http://www.mathrec.org/old/2002jan/solutions.html.
This article discusses the mathematics behind statements of the number or unique games of Tic-Tac-Toe that can be played.

Notes

[i] (Common Core State Standards Initiative n.d.)
[ii] (Partnership for 21st Century Skills 2011)
[iii] (Chu-Caroll 2008)
[iv] (Hochmuth 2003)
[v] (DevTech Research Group at Tufts University, the Lifelong Kindergarten Group at the MIT Media Lab, and the Playful Invention Company n.d.)
[vi] (Brennan, Balch and Chung 2014)
[vii] (McOwan, Peter; Curzon, Paul; support from EPSRC and Google n.d.)
[viii] (Randomness and Integrity Services Ltd. 1998)
[ix] (Schaefer 2002)
[x] (Munroe n.d.)