# Unplugged Computer Science Activities for High Schoolers: An Invitation to the Field

Vee Pettit
Virginia Tech
Blacksburg, USA
vpettit@vt.edu

John Davis Tingle
Virginia Tech
Blacksburg, USA
jdtingle@vt.edu

Tianyang Robin Lu
Virginia Tech
Blacksburg, USA
robinlu@vt.edu

## ABSTRACT

Broadening participation and diversity in computer science is a popular area, with significant work already done in the space. Much of this research focuses on the retention of students already in computing disciplines. However, by that point, many students have already been turned away from, or rather *not invited into*, the CS community. Two of the biggest factors that discourage high school students from studying computing are lack of interest and lack of self-efficacy. For this reason, we aim to create an on-ramp for these students to gain excitement and knowledge about the field. Unplugged activities, which are learning activities not involving technology, are effective at raising student self-efficacy and interest in computing. In this paper, we start by evaluating existing literature for recommendations for unplugged activity design. From that, we construct a set of universally applicable guidelines. Next, we use these guidelines to create two unplugged activities, complete with lesson plans and teacher scripts. One activity is an introduction to reinforcement learning using checkers and candy. The other is an introduction to sorting using playing cards. We ran a pilot study on two participants that gave light to many logistics issues with the activities. The success of these activities will be evaluated by giving participants a survey before and after the activity. Finally, we discuss the dissemination of these activities and how to convince school administrators and teachers to use them in class.

## KEYWORDS

Computer Science Education, Unplugged Activities, Broadening Participation, Self-efficacy

**ACM Reference Format:**
Vee Pettit, John Davis Tingle, and Tianyang Robin Lu. 2024. Unplugged Computer Science Activities for High Schoolers: An Invitation to the Field. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 21 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

One major goal of the computer science community over the last decade has been to broaden participation. Much work in this area involves supporting college students in computing disciplines by attempting to improve retention. However, by the time students choose to major in CS, a large percentage of the population has already been lost. One major factor discouraging students from declaring a CS major is a lack of understanding of CS. A majority of students have no idea what computer scientists do, and only a select few high school students have a good grasp of what the CS field offers [4, 29, 35, 36]. Some high schools do offer CS courses, but students with no experience are just as hesitant to take these courses as they are to sign up for the major. Increasing an understanding - and sparking an interest - in computer science would enable students to take CS courses in high school which would lead many students to an on-ramp to a career in computing.

One promising strategy for lowering barriers in participation among students is the use of unplugged learning activities[6]. There is relatively little research into CS unplugged activities for high school students, as most of it is aimed at elementary-level students [16, 39]. Further, and even if this is attained, there are still few resources for teachers to use for unplugged activities at the high school level [14]. Therefore, we aimed our project at this education level.

Our project includes two unplugged activities that teachers can implement without much setup. These activities have lesson plans and scripts and are in machine learning and algorithms. When used effectively they should help students have a deeper understanding of these topics and be ready for a more in-depth discussion. Students also should understand what computer science is, and feel better about their abilities to do computer science. Our project also includes a survey to be used before and after each activity which will measure students' feelings towards computer science. Lastly, the activities and survey are put together in a way that makes the unplugged activities easy to access and use for both teachers and administrators. In completing this project, we have added resources for teachers to use on high school students, hopefully supporting the growth of computer science.

## 2 RELATED WORKS

### 2.1 Interest and Self Efficacy

For many students, the interest in computing drops during primary and secondary school. This is also an issue of broadening participation as it more highly affects students that are female, people of color, or of lower socioeconomic status [17, 19, 34].

One major barrier to computing is the lack of self-efficacy [22, 27, 32]. This barrier is even more detrimental for already underserved populations [12, 20]. Self-efficacy is how much a person believes they can succeed at a certain task [1, 13]. For example, a student used to scoring high on exams may have a high self-efficacy

*Conference acronym 'XX, June 03–05, 2018, Woodstock, NY*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

toward taking exams, but a student with test anxiety may experience low self-efficacy toward taking exams. Students with a lower self-efficacy towards success in one area are less likely to pursue that area. [26, 30]. Based on the MUSIC model of motivation, four main factors contribute to self-efficacy: past performance, watching others succeed, receiving feedback, and emotion [21]. By giving students an experience of succeeding at a computer science-related activity, we are targeting their past performance, the most influential of the four. A higher self-efficacy is also correlated with higher success in a student's courses [37].

## 2.2 Intersectionality

Broadening participation in computer science is a difficult problem. Everyone has different experiences that influence one's perception of computer science. Using the lens of intersectionality can help us ensure equitable and accessible outcomes [41]. Students with prior access to computers, or who have parents in the computer science industry, are more likely to look at computer science favorably than someone with no access to computers. Furthermore, it has been shown that self-efficacy levels influence motivation and desire to acquire more knowledge [37]. Students with no knowledge of computer science or exposure to computers will naturally have lower confidence in themselves to do computer science. This poses problems to those trying to broaden participation in computer science from the start. Our unplugged activities consider this and expose everyone to computer science in an approachable way. No matter what a person's prior knowledge of computer science is, they will be able to participate in our activities. Designing our activities with this in mind will help students feel that computer science is an area for them when they otherwise would not have felt that way.

## 2.3 Unplugged Activities

Unplugged activities are learning activities that do not use a computer or other technology. Examples of unplugged activities include board games, building blocks, worksheets, and even playground games. There is relatively little research into cs unplugged activities. What exists predominantly focuses on the elementary age group [39]. However, these activities have proven to be an effective means of teaching computing concepts and computational thinking [2, 10]. Existing UPA research shows that these types of activities lower the barriers of gender, race, and socioeconomic status [5, 8]

## 3 METHODOLOGY

Interest and Self-efficacy are significant barriers to participation in computer science. These barriers are even bigger for students from under-served populations. Because they positively affect interest and self-efficacy and have fewer barriers than many computer-based activities, unplugged learning activities are ideal for lowering those barriers. These activities act as an on-ramp to welcome all students into the computer science community. This leads us to ask two questions:

RQ1: How can unplugged cs activities be designed to help lower barriers to participation for high school students with little to no computer science experience?

RQ2: How can the lens of intersectionality inform the design of said activities?

### 3.1 Design Suggestions

While unplugged activities are not common in high school, research suggests that they have been done at lower grade levels. It is reasonable to expect the lessons learned there will apply to older students. First of all, it is important to use universal design to enable lessons to be accessible for a wide range of students [18]. This includes making accommodations and modifications as needed, as well as ensuring students feel included. This can by achieved by making sure students are represented and given as much autonomy as possible [25]. Using familiar types of activities, like board games and puzzles, helps students connect to the activity and makes them feel more at ease [23, 40]. We also draw from the unplugged design pattern proposed by Nishida et. al [31]

Design Guidelines:

- Activities must be as equitable and accessible as possible
- Use board games and puzzles
- Use familiar examples like popular movies and memes
- Make real-world example culturally relevant and appropriate
- Don't expect math proficiency
- Allow for creative choice when possible
- Encourage kinesthetic movement
- Facilitate participation from all students
- If using human characters, use diverse sprites and allow for character customization when possible

One fairly common method of teaching children about algorithms is discussing how to make a peanut butter and jelly sandwich [9]. For many American students, this does an excellent job of using a relevant and familiar example. However, if this activity was deployed in a country where this food was not readily available, this would only complicate the lesson for them. Before giving the lesson, the teacher should figure out an equivalent food item for their local population. This is an example of how activities can be made culturally relevant.

### 3.2 Activities

*3.2.1 Learning Objective Selection.* Because the target audiences for these activities are school administrators and school teachers, we are using the standards of learning as outlined by the Virginia Department of Education [11]. As researchers, we chose Virginia for its local proximity and ease of access for experimentation. The standards of learning for the computer science foundations section has 6 categories: Algorithms and Programming, Computing Systems, Cybersecurity, Data and Analysis, Impacts of Computing, and Networks and the Internet. In the long term, we would like to make activities for each area. However, practical limitations determine we make only two activities. Therefore, we used the first two categories: Algorithms and Programming, and Computing Systems. Specifically, we chose these subsections:

The playing card activity has been designed to satisfy Algorithms and Programming subsection 7 parts 2, 3, and 5.

> AP.7:
> *The student will use search algorithms and sort algorithms*

(1) *Define the concept and role of a search algorithm.*
(2) *Define the concept and role of a sort algorithm.*
(3) *Compare and contrast bubble sort, quick sort, and merge sort.*
(4) *Compare and contrast linear search and binary search.*
(5) *Evaluate and determine the best search or sort algorithm to use based on intended results.*

The checkers and Skittles activity has been designed to satisfy Computer Systems subsection 4 parts 1 and 2.

CSY.4:
*The student will describe and explain the methods by which computers learn through the use of machine learning.*
(1) *Compare and contrast the learning process of humans and computers.*
(2) *Identify mathematical models used by supervised learning to produce classifications and predictions.*

*3.2.2 Reinforcement Learning with Skittles.* This activity is designed to help students understand reinforcement learning, and feel more knowledgeable about artificial intelligence. AI is extremely popular at the moment, and its familiarity will help draw students' attention. Further, it will also make examples familiar and culturally relevant. The choice to play checkers and use Skittles also follows guidelines for board games and kinesthetic movement. This activity is inspired by a game of chess-playing matchboxes created to explore machine learning first invented in 1962 [15], and related activities also based on the matchbox computer [7, 28].

This activity starts with a talk about what it means to learn and how to teach a dog to do a trick. Next, the abstract concept of reinforcement learning is explained. The activity has the students model a learning agent which will play 4x4 checkers. For each possible game state, there is a cup with the game state printout taped to it. Each of these states also demonstrate arrows that correlate to possible moves the computer could make. In turn, these arrows are colored to each match a different Skittles color. In each cup are 3 Skittles of each color arrow.

Whichever color skittle is randomly drawn from the cup is the move that the learning agent makes. When the learning agent wins a game, that decision is reinforced by adding another skittle, the color of the winning move used, to that cup. For example, one state could have two possible moves that are represented by a green arrow and a red arrow. Say the green arrow is drawn, its corresponding move is made, and the learning agent wins. The original green skittle would be replaced in the cup and another green skittle would be added. The chance of making that winning move will have risen from 1/2 (three out of six Skittles) to 4/7 (four out of seven Skittles). If the model makes a losing move, that skittle gets removed. So the probability would go from 1/2 to 3/7.

The game is repeated over and over as the model learns. After enough games are played, the students will see that many of the Skittles cups have changed to have one color as a majority. And the learning agent will be "good" at checkers. See Figure 1 for a screenshot of the activity video. The activity concludes with a lesson about reinforcement learning being used to make the backgammon playing machine TD-gammon, and what lessons can be learned there. See Appendix A for the lesson plan, video, graphics, and printouts.

*3.2.3 Sorting with Playing Cards.* This activity is designed to help students feel prepared for sorting and time complexity concepts, which are foundational in computer science. The use of playing cards aligns with the guideline to use familiar items and board games. This activity also requires no significant math proficiency, another barrier common to algorithmic lessons. Having the students design their sorting methods also plays to their autonomy and ability to make creative choices.

This activity teaches sorting using a deck of cards. In this activity, students will be divided into groups where they will have 5-10 minutes to come up with a strategy to sort a shuffled deck of cards with the fewest number of comparisons. After this planning phase, there will be a competition between groups with the winner being the group that can sort a deck of cards with the fewest number of comparisons. After the competition is finished, each group will have the opportunity to explain their sorting strategy. This allows for discussion of sorting algorithms and provides an opportunity for teachers to explain time complexity at a high level. After the activity is over, students should be ready for a deeper dive into algorithms. See Appendix B for the lesson plan.

## 3.3 Evaluation Framework

Many learning activities evaluate their success by testing the computational thinking skills of the participants [3, 33]. However, the problem we are addressing is getting students to that point where they can be tested on their computational thinking skills. This is why, instead, we will be measuring the constructs of self-efficacy and interest [24].

*3.3.1 Survey.* To measure self-efficacy and interest, we will be using the survey created by Shehzad et al. [38]. The original survey measures interest, self-efficacy, and parental support. The survey has been altered to only include the questions in the sections for "interest in programming" and "self-efficacy." All items are measured on a Likert scale ranging from 1 to 6, with "1" denoting "strongly disagree" and "6" indicating "strongly agree." Questions are as follows:

Interest:

- Computer programming sounds fun.
- I think computer programming is interesting.
- I think computer programming is boring.

Self Efficacy:

- If I took a class on computer programming, I could do well.
- If I wanted to, I could be a computer programmer in the future.
- I think I could do more challenging computer programming.
- I can learn to do computer programming.
- I am a good computer programmer.
- I am confident in my ability to program.
- I can program computers well.

The survey will be given to the students before and after the learning activity. Depending on the participants, it may be appropriate to also gather demographic data to deepen our understanding of the results and ensure equity.

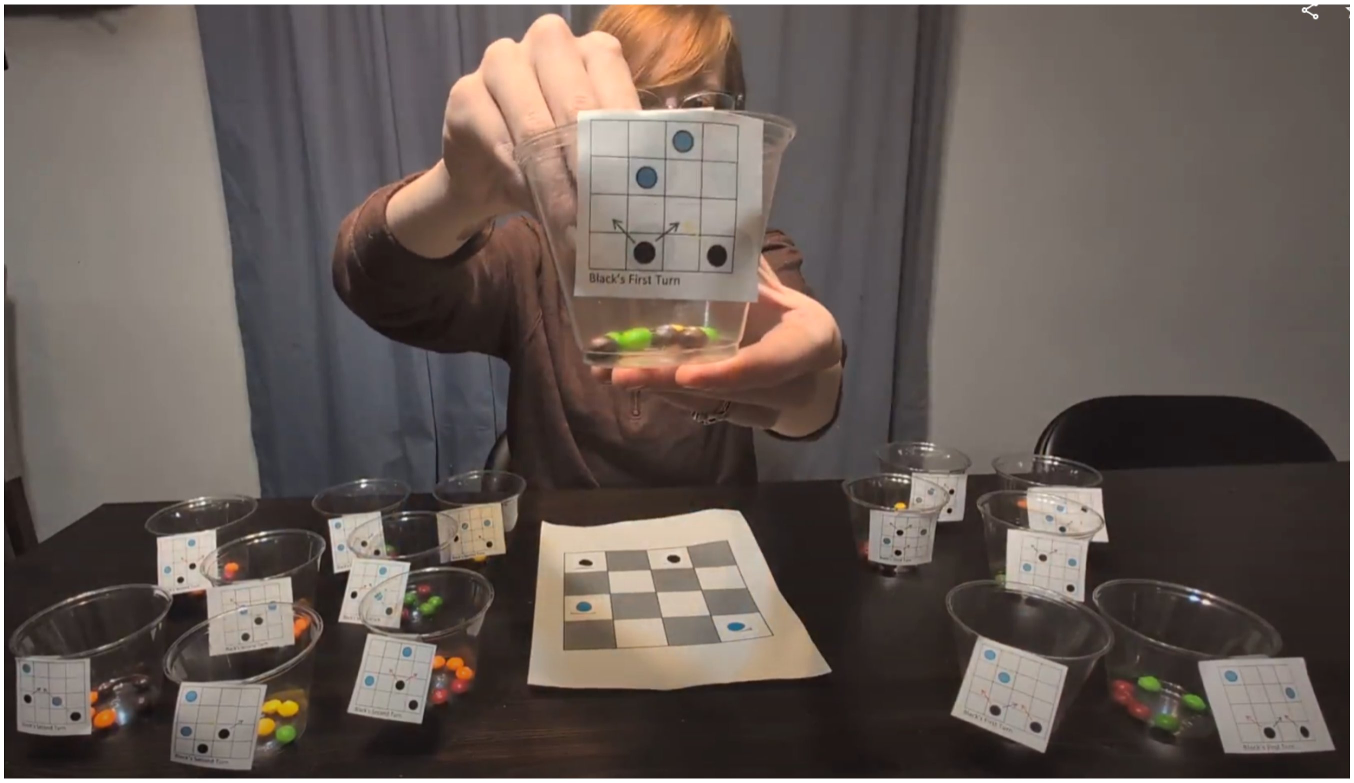


**Figure 1: Screenshot from the reinforcement learning video lesson**

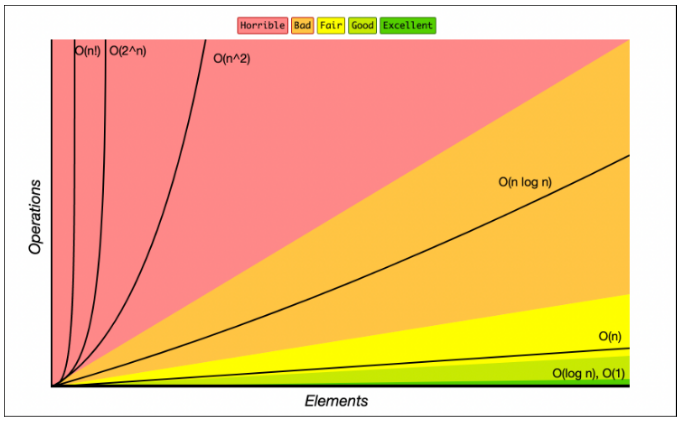


**Figure 2: Graph used to explain sorting time complexity**

*3.3.2 Population.* In the interest of the study's main objective, broadening participation in Computer Science, our target audience is first and second-year high school students. Many high school students have a lack of understanding of CS. This pre-college level is a formational time in which students entering college have to decide on a field to study, most of whom do not have an in-depth understanding of what could be best suited for them. While there already exists material for younger students at lower levels, we have identified a lack of unplugged materials that are well-suited for this particular audience at the high school level and have specifically targeted these students in our activities.

Depending on the specifics such as admission policy and assumed knowledge level, if the students are of similar knowledge level, we believe the material could potentially also be applied to entry-level college students that are under general education or simply still deciding on a field of study to pursue.

## 4 RESULTS

### 4.1 Pilot Study

To test out the logistics of our learning activities, we did a small pilot study where we ran the lessons with two different participants. The participants were chosen as a sample of convenience. The first participant was close to the target population as they were a community college student with no CS background. The second participant was not near the target population as they were a graduate student with a CS background. Even though they are not the target population, it was still very useful to run the activities and see what logistical issues came up. Each session was run one-on-one between one of the research team and the participant. The researcher took notes, including timings, throughout the sessions. At the end the participants were asked for unstructured feedback.

*4.1.1 Survey Results.* One participant declined to take the survey, so there is only one result. With this sample size and population, no statistical significance can be claimed. However, the result of the single survey was mildly promising. Of the 10 questions, 7 questions received the same answer on the post survey and 3 questions received a more positive answer on the post survey. These questions were as follows:

(1) 1. I think computer programming is interesting: 4 raised to 5
(2) 2. If I wanted to, I could be a computer programmer in the future: 4 raised to 5
(3) 3. I am confident in my ability to program : 3 raised to 4

This extremely anecdotal evidence does point towards success at raising both interest (question 1) and self-efficacy (questions 2 and 3) in computing.

#### 4.1.2 *Survey Logistics.*

There were also some lessons to be learned from the survey. For one, it took longer than the researcher expected, with the longer time being approximately 6 minutes. Second, the participant with no CS background asked questions and expressed confusion over the wording of several questions. "I think I could do more challenging computer programming" and "I can program computers well" were confusing to them, as they had not done any computer programming prior.

#### 4.1.3 *Concepts.*

During the study, one concept caused trouble for a participant. In the sorting cards activity, there was confusion over the difference in the number of comparisons and the amount of time in seconds it would take to sort the cards. The concept was explained at the beginning of the activity, but the researcher had to pause the participant during the activity to remind them which spurred the researcher to explain again. Additionally, the participant with no CS background said they were very interested in the beginning of the reinforcement learning lesson and the comparisons to a dog learning a trick were key to their understanding. Both participants also expressed that discussing AI tools seen on the news made them more interested.

#### 4.1.4 *Practical Logistics.*

Finally, there were plenty of practical logistics learned. First, for the sorting activity, there was some confusion about putting suits in order. This could be fixed in the future by giving each group a set of just 13 cards all in the same suit. This also would address the second card issue. Since 52 cards would be far too many to sort in a short time, we gave participants a random set of 20 cards from the deck. There was a sense of dissatisfaction from participants at having not felt like they completed the task.

Many issues popped up in the moment related to the Skittles activity. First of all, setup took a non-trivial amount of time. Each game state had to be cut out and taped to a cup, which was quick. However, each cup had to be pre-populated with the correct Skittles. This took time to do, and problems came up. The pilot study used Skittles in individually wrapped packages. The distribution of Skittles colors was not even, and in one case an extra 5 packages were required because there was no orange in several of the other packages. The individual packages of Skittles were used because they were convenient and on sale. In the future, it would be more practical to get larger bags, or even to purchase Skittles by the color in bulk.

There was also some unease from one participant about the Skittles being sticky. In the lesson plan, there is the option to use marbles, beads, or the like as a substitute. The excitement of Skittles is a major draw to the activity, but certain environments may not allow their use due to health and safety concerns.

When running the game, the yellow arrows on the printout were hard to read, and the red and orange arrows were difficult to distinguish in poor lighting. The use of marbles or beads would allow for more legible color options. The colors of arrows on each game state were distributed in an even pattern. However, it ended up where one color was favored and more of those colors were needed to add to the cups for positive reinforcement. This worsens the problem of uneven color distribution.

### 4.2 Dissemination

These unplugged activities are useful in multiple scenarios. The first scenario, and most related to the goals of our project, is to use our activities in mathematics or science classes around the time when students are selecting their future courses. Implementing the activities around this time exposes students to computer science, and will hopefully change their misconceptions of this field. Implementing our activities at this time may encourage students who did not think computer science was for them. As a result, more students will be likely to sign up for a computer science class the next semester.

The second scenario to use our activities is in a computer science classroom before the respective units that our activities cover. Our machine learning activity should be used before a machine learning section, and our sorting algorithm activity should be used before a sorting section. Both of our activities help students have a general understanding of each topic in an easy and approachable way. By starting each unit with our activities, the students will have something to refer back to if they get confused with some of the more complex topics in those areas.

Before our activities are used in the classroom though, teachers and administrators need to be convinced that our activities are useful. To do this, we would pitch our activities and their benefits to interested teachers and administrators. This pitch would include the goals of our activities, like getting more students into computer science classrooms. Our pitch would also include that our activities are easy to set up, and implement, and require little extra work from teachers.

### 4.3 Conclusion

In this project, we have put together guidelines for the creation of unplugged computer science activities. Within these guidelines, we have also created two activities to be used at the high school level. We have also determined how to evaluate the effectiveness of the activities in raising a student's interest and self-efficacy in regard to computer science. The pilot study raised many logistical issues that are less important pedagogically, but very important practically. Finally, we have discussed the way in which to disseminate these activities to school administrators and teachers.

## 5 FUTURE WORK

The continuation of this project includes deploying our activities with more users and the creation of more activities. Since we have very few survey results, we do not know if our activities meet the goals of this project. To get there, we need to test our activities and measure the survey results with a test group. Future researchers can also create more activities for the high school level. More quality unplugged activities should help bridge the gaps in computer science.

## 6 LIMITATIONS

The main limitations of this study are the small sample size used in the pilot study and a lack of teacher input. Unfortunately, finding students to test our activities on proved more difficult than expected. ßThis resulted in a small testing size and few survey results. Along with the difficulty in finding students, finding teachers to discuss their wants and needs was difficult as well. Having their input would have been useful, but was not required in the making of our activities.

## REFERENCES

[1] Albert Bandura. 1977. Self-efficacy: Toward a unifying theory of behavioral change. *Psychological Review* 84, 2 (1977), 191–215. https://doi.org/10.1037/0033-295x.84.2.191

[2] Christian Andrés Benavides-Escola, Estefanía Martín-Barroso, María Zapata-Cáceres, and Marcos Román-González. 2024. Improvement of Computational Thinking skills through unplugged activities in Upper Secondary Education. In *Proceedings of the 19th WiPSCE Conference on Primary and Secondary Computing Education Research* (Munich, Germany) *(WiPSCE '24)*. Association for Computing Machinery, New York, NY, USA, Article 11, 9 pages. https://doi.org/10.1145/3677619.3678110

[3] Christian Andrés Benavides-Escola, Estefanía Martín-Barroso, María Zapata-Cáceres, and Marcos Román-González. 2024. Improvement of Computational Thinking skills through unplugged activities in Upper Secondary Education. In *Proceedings of the 19th WiPSCE Conference on Primary and Secondary Computing Education Research* (Munich, Germany) *(WiPSCE '24)*. Association for Computing Machinery, New York, NY, USA, Article 11, 9 pages. https://doi.org/10.1145/3677619.3678110

[4] Lori Carter. 2006. Why students with an apparent aptitude for computer science don't choose to major in computer science. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (Houston, Texas, USA) *(SIGCSE '06)*. Association for Computing Machinery, New York, NY, USA, 27–31. https://doi.org/10.1145/1121341.1121352

[5] Jay Chen, Azza Abouzied, David Hutchful, Joy Ming, and Ishita Ghosh. 2016. printr: Exploring the Potential of Paper-based Tools in Low-resource Settings. In *Proceedings of the Eighth International Conference on Information and Communication Technologies and Development* (Ann Arbor, MI, USA) *(ICTD '16)*. Association for Computing Machinery, New York, NY, USA, Article 23, 11 pages. https://doi.org/10.1145/2909609.2909649

[6] Vincent A. Cicirello. 2013. A CS unplugged activity for the online classroom. *J. Comput. Sci. Coll.* 28, 6 (2013), 162–168.

[7] Paul Curzon and Peter McOwan. 2016. *The Sweet Learning Computer.* https://www.cs4fn.org/machinelearning/sweetlearningcomputer.php

[8] Gabriela de Carvalho Barros Bezerra, Wilk Oliveira, Ana Cláudia Guimarães Santos, and Juho Hamari. 2024. Exploring the Use of Unplugged Gamification on Programming Learners' Experience. *ACM Trans. Comput. Educ.* 24, 3, Article 42 (Sept. 2024), 25 pages. https://doi.org/10.1145/3686165

[9] Myra Deister, Leslie Aaronson, and Brad Ashley. [n. d.]. *Using the PBJ Sandwich Activity to Introduce Important Components of Algorithms.* http://website-url.com

[10] Havva DELAL and Diler ONER. 2020. Developing Middle School Students' Computational Thinking Skills Using Unplugged Computing Activities. *Informatics in Education* 19, 1 (2020), 1–13. https://doi.org/10.15388/infedu.2020.01

[11] Virginia department of education. 2024. 2024 computer science standards of learning. https://www.doe.virginia.gov/teaching-learning-assessment/k-12-standards-instruction/computer-science/2024-computer-science-standards-of-learning

[12] Jennifer DeWitt and Louise Archer. 2017. Participation in informal science learning experiences: the rich get richer? *International Journal of Science Education, Part B* 7, 4 (2017), 356–373. https://doi.org/10.1080/21548455.2017.1360531

[13] Mart Dinther, Filip Dochy, and Mien R.Segers. 2011. Factors affecting students' self-efficacy in higher education. *Educational Research Review* 6 (12 2011), 95–108. https://doi.org/10.1016/j.edurev.2010.10.003

[14] Yvon Feaster, Luke Segars, Sally K. Wahba, and Jason O. Hallstrom. 2011. Teaching CS unplugged in the high school (with limited success). In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education* (Darmstadt, Germany) *(ITiCSE '11)*. Association for Computing Machinery, New York, NY, USA, 248–252. https://doi.org/10.1145/1999747.1999817

[15] Martin Gardner. 1969. *The Unexpected Hanging and Other Mathematical Diversions.* University of Chicago Press, Chicago 60637, Chapter Chapter 8: A Matchbox Game-Learning Machine, 90–102.

[16] Anaclara Gerosa, Maria Kallia, and Quintin Cutts. 2023. A Systematic Literature Review on Physical and Action Based Activities in Computing Education for Early Years and Primary. In *Proceedings of the 18th WiPSCE Conference on Primary and Secondary Computing Education Research* (Cambridge, United Kingdom) *(WiPSCE '23)*. Association for Computing Machinery, New York, NY, USA, Article 10, 10 pages. https://doi.org/10.1145/3605468.3605500

[17] Sandy Graham and Celine Latulipe. 2003. CS girls rock: sparking interest in computer science and debunking the stereotypes. *SIGCSE Bull.* 35, 1 (Jan. 2003), 322–326. https://doi.org/10.1145/792548.611998

[18] Alexandria K. Hansen, Eric R. Hansen, Hilary A. Dwyer, Danielle B. Harlow, and Diana Franklin. 2016. Differentiating for Diversity: Using Universal Design for Learning in Elementary Computer Science Education. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (Memphis, Tennessee, USA) *(SIGCSE '16)*. Association for Computing Machinery, New York, NY, USA, 376–381. https://doi.org/10.1145/2839509.2844570

[19] Lucia Happe, Barbora Buhnova, Anne Koziolek, and Ingo Wagner. 2020. Effective measures to foster girls' interest in Secondary Computer Science Education. *Education and Information Technologies* 26, 3 (Nov 2020), 2811–2829. https://doi.org/10.1007/s10639-020-10379-x

[20] Sarah Hug. 2024. A Shift To Praxis- Towards a More Inclusive Research Agenda about Belonging in Computer Science. In *Proceedings of the 2024 on RESPECT Annual Conference* (Atlanta, GA, USA) *(RESPECT 2024)*. Association for Computing Machinery, New York, NY, USA, 302–306. https://doi.org/10.1145/3653666.3656070

[21] Brett Jones. 2009. Motivating Students to Engage in Learning: The MUSIC Model of Academic Motivation. *International Journal of Teaching and Learning in Higher Education* 21, 2 (2009).

[22] Brett D. Jones, Margaret Ellis, Fei Gu, and Hande Fenerci. 2023. Motivational climate predicts effort and achievement in a large computer science course: Examining differences across sexes, races/ethnicities, and academic majors - International Journal of STEM Education. https://stemeducationjournal.springeropen.com/articles/10.1186/s40594-023-00457-0

[23] Victor R. Lee, Frederick Poole, Jody Clarke-Midura, Mimi Recker, and Melissa Rasmussen. 2020. Introducing Coding through Tabletop Board Games and Their Digital Instantiations across Elementary Classrooms and School Libraries. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 787–793. https://doi.org/10.1145/3328778.3366917

[24] Luzia Leifheit, Katerina Tsarava, Korbinian Moeller, Klaus Ostermann, Jessika Golle, Ulrich Trautwein, and Manuel Ninaus. 2019. Development of a Questionnaire on Self-concept, Motivational Beliefs, and Attitude Towards Programming. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education* (Glasgow, Scotland, Uk) *(WiPSCE '19)*. Association for Computing Machinery, New York, NY, USA, Article 26, 9 pages. https://doi.org/10.1145/3361721.3361730

[25] Lorraine Lin, Dhaval Parmar, Sabarish V. Babu, Alison E. Leonard, Shaundra B. Daily, and Sophie Jörg. 2017. How character customization affects learning in computational thinking. In *Proceedings of the ACM Symposium on Applied Perception* (Cottbus, Germany) *(SAP '17)*. Association for Computing Machinery, New York, NY, USA, Article 1, 8 pages. https://doi.org/10.1145/3119881.3119884

[26] Alex Lishinski, Aman Yadav, Jon Good, and Richard Enbody. 2016. Learning to Program: Gender Differences and Interactive Effects of Students' Motivation, Goals, and Self-Efficacy on Performance. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16)*. Association for Computing Machinery, New York, NY, USA, 211–220. https://doi.org/10.1145/2960310.2960329

[27] Lauren E. Margulieux, James Prather, Brent N. Reeves, Brett A. Becker, Gozde Cetin Uzun, Dastyni Loksa, Juho Leinonen, and Paul Denny. 2024. Self-Regulation, Self-Efficacy, and Fear of Failure Interactions with How Novices Use LLMs to Solve Programming Problems. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1* (Milan, Italy) *(ITiCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 276–282. https://doi.org/10.1145/3649217.3653621

[28] Erik Marx, Thiemo Leonhardt, David Baberowski, and Nadine Bergner. 2022. Using Matchboxes to Teach the Basics of Machine Learning: an Analysis of (Possible) Misconceptions. In *Proceedings of the Second Teaching Machine Learning and Artificial Intelligence Workshop (Proceedings of Machine Learning Research, Vol. 170)*, Katherine M. Kinnaird, Peter Steinbach, and Oliver Guhr (Eds.). PMLR, 25–29. https://proceedings.mlr.press/v170/marx22a.html

[29] Irene T. Miura. 1987. The relationship of computer self-efficacy expectations to computer interest and course enrollment in college. *Sex Roles* 16, 5 (March 1987), 303–311. https://doi.org/10.1007/BF00289956

[30] Sukanya Kannan Moudgalya, Chris Mayfield, Aman Yadav, Helen H. Hu, and Clif Kussmaul. 2021. Measuring Students' Sense of Belonging in Introductory CS Courses. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science*

*Education* (Virtual Event, USA) *(SIGCSE '21)*. Association for Computing Machinery, New York, NY, USA, 445–451. https://doi.org/10.1145/3408877.3432425

[31] Tomohiro Nishida, Susumu Kanemune, Yukio Idosaka, Mitaro Namiki, Tim Bell, and Yasushi Kuno. 2009. A CS unplugged design pattern. *SIGCSE Bull.* 41, 1 (March 2009), 231–235. https://doi.org/10.1145/1539024.1508951

[32] Vidushi Ojha, Leah West, and Colleen M. Lewis. 2024. Computing Self-Efficacy in Undergraduate Students: A Multi-Institutional and Intersectional Analysis. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 993–999. https://doi.org/10.1145/3626252.3630811

[33] Brandon Rodriguez, Stephen Kennicutt, Cyndi Rader, and Tracy Camp. 2017. Assessing Computational Thinking in CS Unplugged Activities. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 501–506. https://doi.org/10.1145/3017680.3017779

[34] Merilin Säde, Reelika Suviste, Piret Luik, Eno Tõnisson, and Marina Lepp. 2019. Factors That Influence Students' Motivation and Perception of Studying Computer Science. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19)*. Association for Computing Machinery, New York, NY, USA, 873–878. https://doi.org/10.1145/3287324.3287395

[35] Linda J. Sax, Chantra Nhien, and Kaitlyn N. Stormes. 2024. A Quantitative Methodological Review of Research on Broadening Participation in Computing, 2005-2022. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1182–1188. https://doi.org/10.1145/3626252.3630768

[36] Carsten Schulte and Maria Knobelsdorf. 2007. Attitudes towards computer science-computing experiences as a starting point and barrier to computer science. In *Proceedings of the Third International Workshop on Computing Education Research* (Atlanta, Georgia, USA) *(ICER '07)*. Association for Computing Machinery, New York, NY, USA, 27–38. https://doi.org/10.1145/1288580.1288585

[37] Dale H. Schunk. 1985. Self-efficacy and classroom learning. *Psychology in the Schools* 22, 2 (1985), 208–223. https://doi.org/10.1002/1520-6807(198504)22:2<208::AID-PITS2310220215>3.0.CO;2-7

[38] Umar Shehzad, Jody Clarke-Midura, and Mimi Recker. 2024. Examining the Role of Parental Support on Youth's Interest in and Self-Efficacy of Computer Programming. *ACM Trans. Comput. Educ.* 24, 3 (Sept. 2024).

[39] Umar Shehzad, Mimi Recker, and Jody Clarke-Midura. 2023. A Literature Review Examining Broadening Participation in Upper Elementary CS Education. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2023)*. ACM, 570–576. https://doi.org/10.1145/3545945.3569873

[40] Yukyeong Song, Xiaoyi Tian, Nandika Regatti, Gloria Ashiya Katuka, Kristy Elizabeth Boyer, and Maya Israel. 2024. Artificial Intelligence Unplugged: Designing Unplugged Activities for a Conversational AI Summer Camp. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1* (Portland, OR, USA) *(SIGCSE 2024)*. Association for Computing Machinery, New York, NY, USA, 1272–1278. https://doi.org/10.1145/3626252.3630783

[41] Marisol Wong-Villacres, Arkadeep Kumar, Aditya Vishwanath, Naveena Karusala, Betsy DiSalvo, and Neha Kumar. 2018. Designing for Intersections. In *Proceedings of the 2018 Designing Interactive Systems Conference* (Hong Kong, China) *(DIS '18)*. Association for Computing Machinery, New York, NY, USA, 45–58. https://doi.org/10.1145/3196709.3196794

## A APPENDIX A: LEARNING PLAN DOCUMENTS

### A.1 Reinforcement Learning with Skittles

Lesson video (20:32) available as shared in Google Drive at https://drive.google.com/file/d/1kjcfvFLmPrSKBX-nZs4JUFmqgxTofUtu

# Unplugged Reinforcement Learning Activity

**Subject:** Computer Science Foundations
**Grade Level:** 9-12
**Date:** 10/29/24
**Duration:** 30-60 mins
**Teacher:** any

---

## Lesson Title/Topic: Teaching a computer to play checkers: with candy!

---

## Learning Objectives:

- By the end of this lesson, students will be able to:
    - Define the concept and role of reinforcement learning
    - Explain how the “computer” from the activity “learns”

---

## Materials Needed:

- Printouts - game states, checkers board, checkers pieces
- Cups
- Skittles (can be substituted with marbles, beads, etc)

---

## Introduction (5-10 minutes):

- **Hook:** Group discussion - How do humans learn? How does a baby learn? How does a dog learn to sit? Getting a treat is reinforcement. But how do we motivate a computer with no stomach?

---

## Instruction (5-10 minutes):

1. **Explanation of activity:** Rules and materials

2. **Example:** Run through a few rounds of the activity all together.

---

## Activity (10-20 minutes):

- **Activity:** Each group will repeatedly “play” against the “computer” and update the game state (adding or removing skittles) as they go

---

## Conclusion of activity (5-10 minutes):

- **Presentation:** Have groups share what changes they have seen
- **Reflection:** Explain how this type of “learning” is used in computer systems people use every day.

---

## Reflection (Post-Lesson):

- What went well?
- What could be improved?
- Notes for next time:

**Script for lecture:**
Today, we're going to peek behind the scenes at one type of AI. We're going to talk about reinforcement learning. You might see headlines that say something along the lines of "Scientists taught an AI to detect the flu" or "An AI learned to predict football games." But what does that actually mean, for a computer to learn?

Well, that's a hard question. Let's move it a little closer to home. How does a human learn? Well, that's also a tricky one. Let's go a bit simpler: a dog. How do you teach a dog to do a trick? *discuss*
You give it a treat when it does the right thing, like sitting. So, we can use treats to help a dog learn something. But computer's don't have stomachs so we'll have to think of something else there. We can't tell a computer to like something, but we can alter it to make decisions based on our feedback. A dog might do something because it likes the reward, and a computer might do something because we've told it that's the best choice of action.

So, we're about to develop our own computer and have it learn how to play tiny checkers. First, I'll give you a refresher on checkers. Each side has two pieces. They start on the light squares and can only move diagonally. There are two ways to win. One: by jumping over your opponent. Two: by reaching the other side. The blue player, our human player, always goes first. Very simple. And we'll talk later about why this example has to be so simple.

Alright, how does this computer work and why in the world do I have so many Skittles in cups? Each of these cups represents a specific state, which is a snapshot of the game in that moment in time. In each state, the computer has a decision to make: what checker do I move and where do I move it? Here's where the skittles come in.
*Grabs cup from turn 1*
If you see here, each possible move the computer can make matches one of these arrows, and these arrows match the color of the Skittles in the cup. When we get to this game state and need to make our decision, we will close our eyes and randomly pick out a skittle. Now we make the move that matches that color. This skittle is *color* so we make this move here *make the move*

Now, that's how we make choices. But how do we learn? First things first, we need to set this Skittle out to remember what move we made. We are going to learn from our past mistakes or successes. So, let's finish this game real quick.
*Have the human move then go through another choice*

*talk about win first if win, loss first if loss*

Allright, we've won. Yes! Now we have to tell our system that it did a good job. It should make those moves more often. How could we take this move here and say good job, next time we come to this game state, you should favor that specific move?
*discuss*
What we can do is put this Skittle back and then add one more of this same color. Now, instead of having an even chance of making each move, the computer will have a bigger chance of making this winning move. So what can we do if the computer is lost and we need to say that was a bad move?
*discuss*
We can remove that Skittle from the cup. Just don't put it back. Now, the computer has a smaller chance of making that losing move. Alright, we've done it. Our computer has learned! But just a tiny bit. Now we're going to break up into our groups and train our computer for about twenty minutes. Then we'll meet back up and talk about it.

**********
*group time*
**********
Welcome back, everyone. I see some very colorful computers. There's a lot of difference from when we started. Who has a cup that is only one color?
*student answer*
Nice, now what does that tell us about that color move?
*student answer*
Yeah, our computer has decided that's the best move. Whose computer was kinda bad at checkers at the start? I should see all hands. *moment for hands* But that's ok. Whose computer is kinda good now? *moment for hands* Alright, everyone give yourself a pat on the back for being excellent teachers.

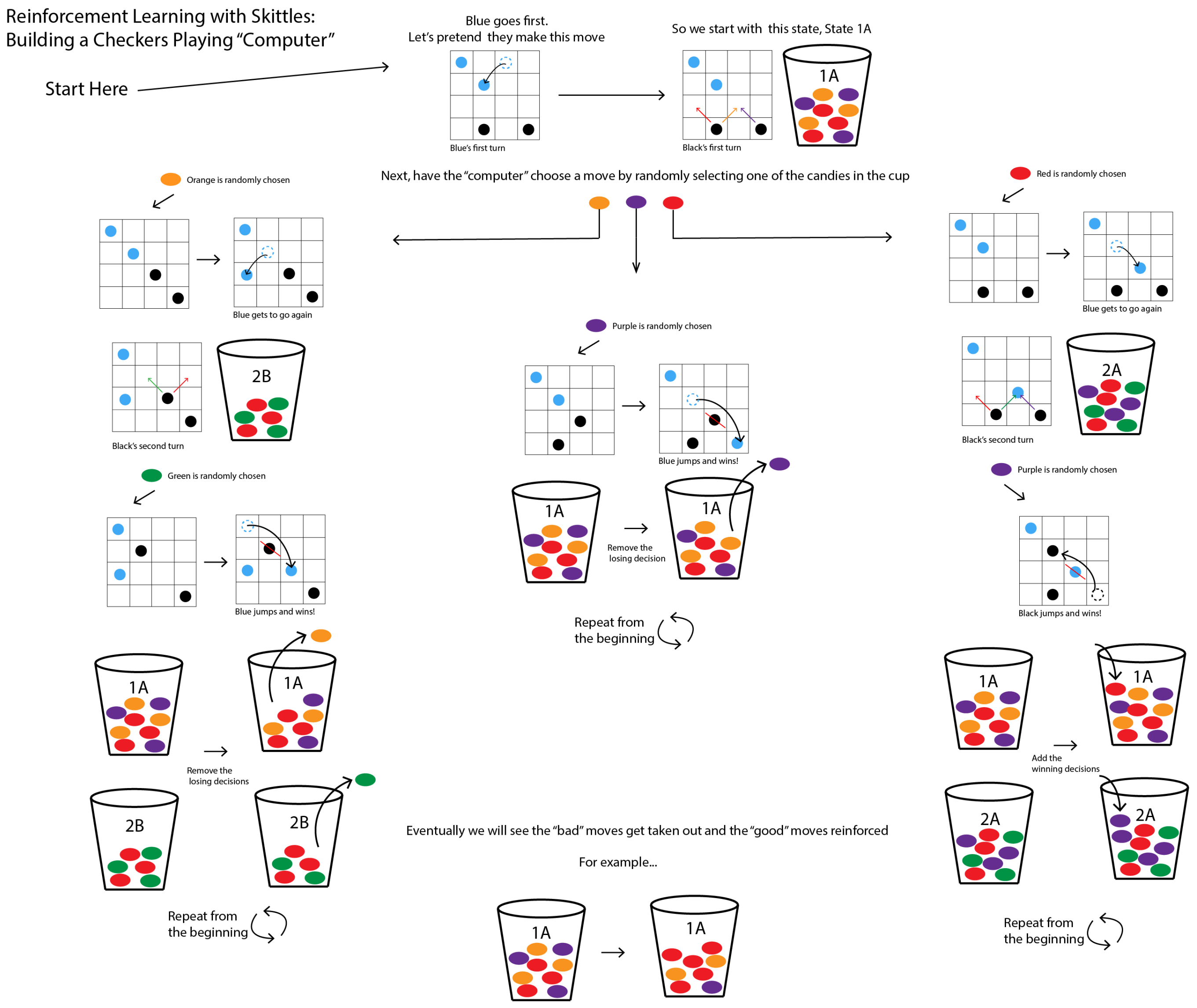
Let's talk about why we choose checkers. In our game, there are only 3 states at turn one. Then we get way more at turn 2. Imagine how many cups we would have for full sized checkers. Instead of two checkers you would have 8. That's thousands of cups! Let's not even mention chess. It's so many. Even though our game may seem a little silly today, board games have been a big part of computer science for a long time.

Let's talk about another game: backgammon. Backgammon is another game with just two players who move pieces on a board. It's in the same family as checkers and chess. Back in 1992 a man named Gerald Tesauro made his own computer to play and he named it TD-Gammon. He spends years working on this thing, and finally in 1998 he takes TD-Gammon to a tournament. The machine plays against experts and it holds its own. However, all the experts notice - hey,

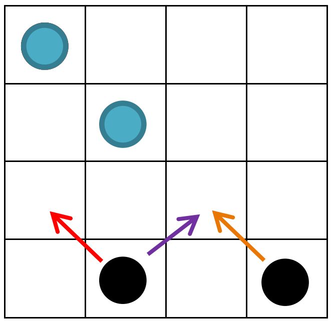
there’s one move TD-Gammon is making that’s kind of a bad move. No experts would ever make that move. But TD-Gammon was winning lots of games using this move. The computer faces off with the world champion. TD-Gammon just barely loses by 8 points across 100 games.
So after the debut of this machine, all the backgammon experts get together and they look at this new move. They try it out, run some calculations, talk about it. And they realize, hey, this is actually a good strategy. Humans have been playing backgammon for over 300 years - and this machine figured out a new strategy! I think that’s pretty awesome. At no point did a human tell that machine a strategy. At no point did any of us humans tell our Skittles computer a strategy. But it learned.

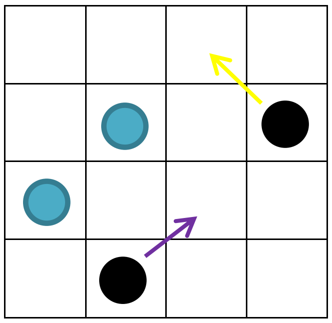
That’s pretty cool, but also, what does it have to do with me? Well, this approach, reinforcement learning, is used to train lots of the big AI tools we use today. Reinforcement learning is one method used to train ChatGPT. When you ask ChatGPT a question, imagine how many billions and trillions of Skittles in cups it has! Ok, it’s all numbers, but the idea is the same.

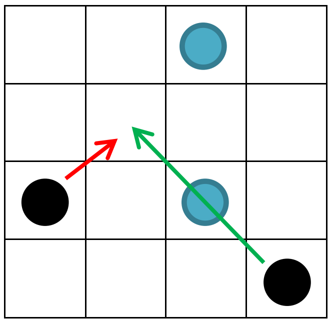
Let’s do a little recap of what we’ve talked about today. Reinforcement Learning is when a computer does an activity over and over and over, and adjusts its decisions based on the feedback of how it did. We did a version of this where our activity was playing checkers, our decisions were making moves, and the computer learns by adding or removing skittles. So, next time you read about or use an AI, think about what we did today. And if you were super duper interested, this is one type of thing that computer scientists do. And you can do it too!

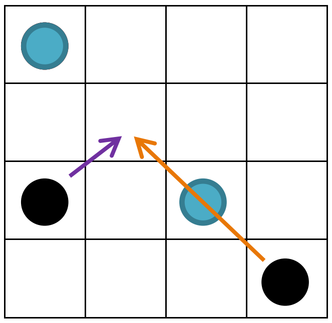


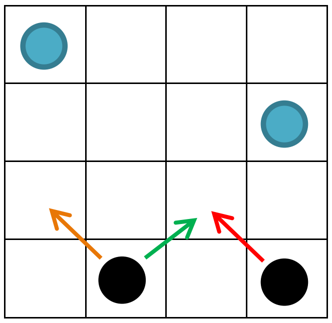
Above: Infographic for Teachers
Below: Printouts

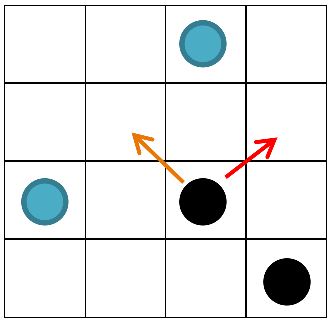

Black's First Turn

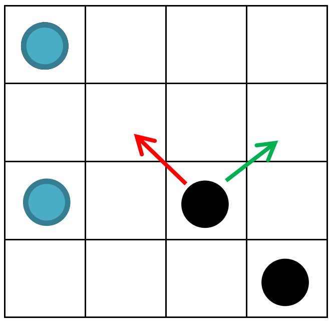

Black's Third Turn

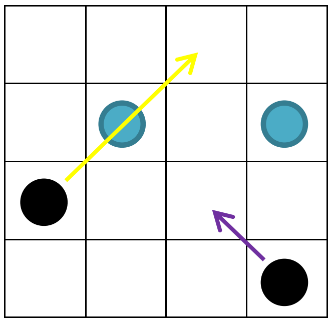

Black's Second Turn

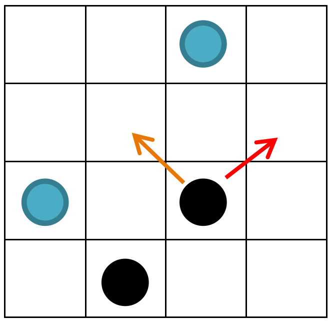

Black's Second Turn

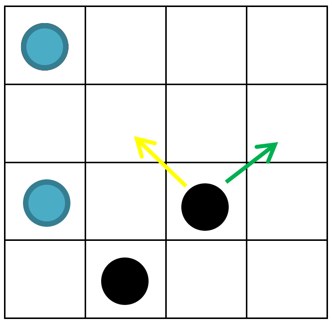

Black's First Turn

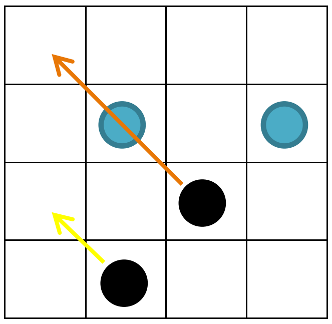

Black's Second Turn

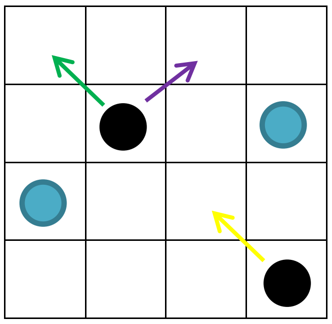

Black's Second Turn

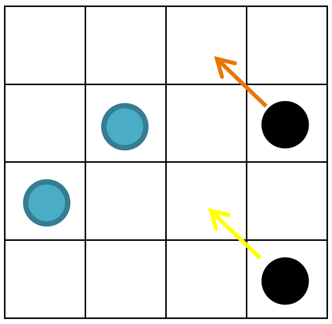

Black's Second Turn

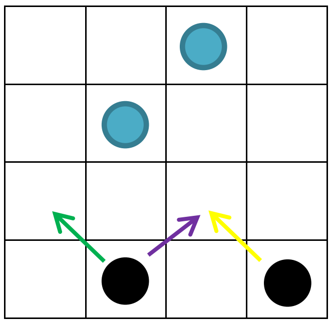

Black's Second Turn

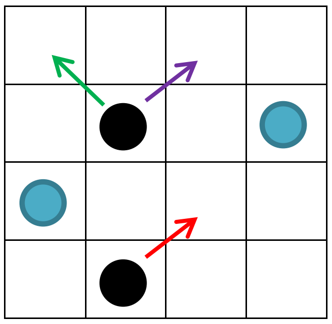

Black's Second Turn

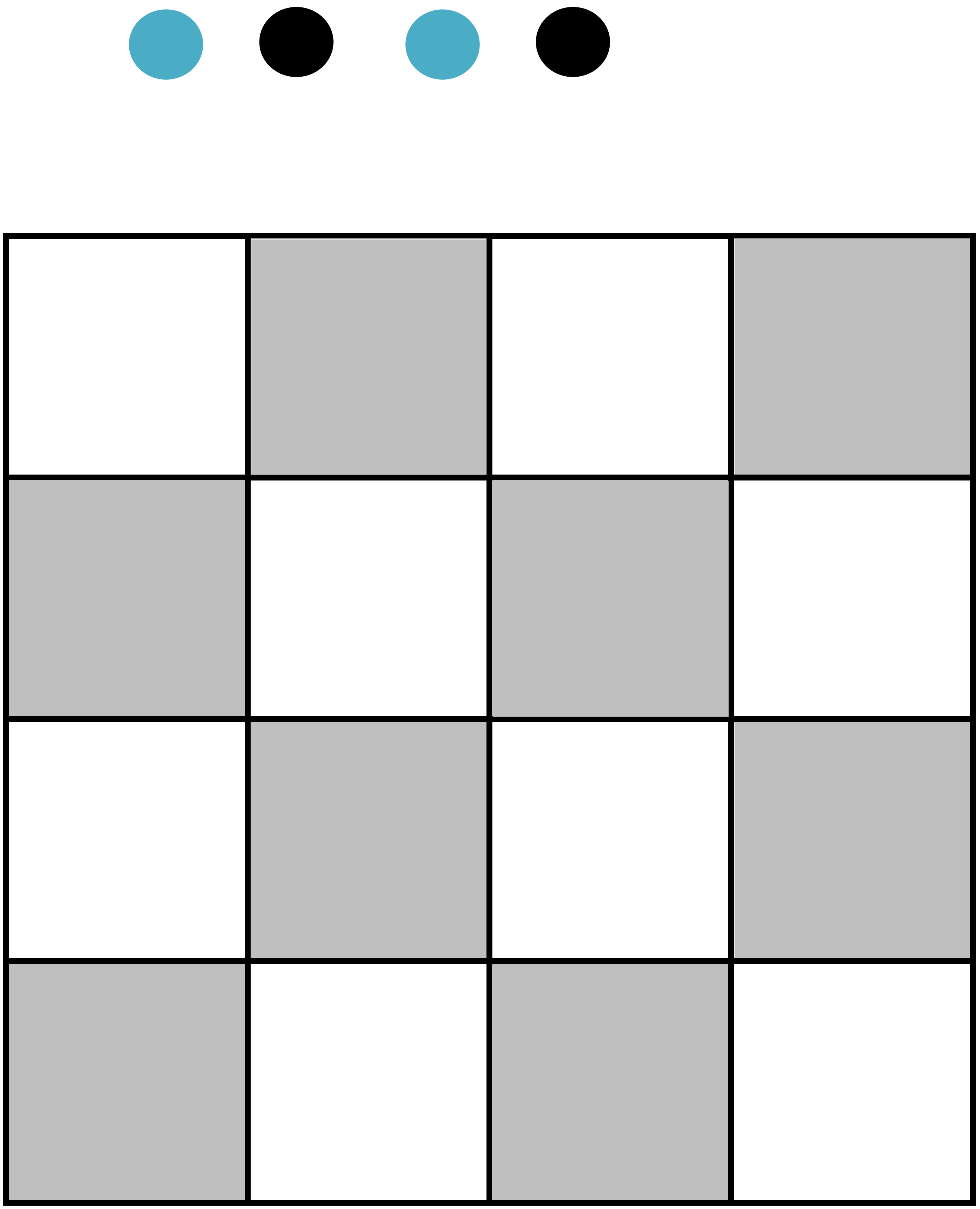

Black's Second Turn


Black's Third Turn


Black's Third Turn


Black's First Turn


Black's Third Turn



### A.2 Sorting with Playing Cards

# Unplugged Algorithms Lesson Plan

**Subject:** Computer Science Foundations
**Grade Level:** 9-12
**Date:** 10/29/24
**Duration:** 1 hour
**Teacher:** any

---

## Lesson Title/Topic: Algorithmic Complexity Activity

---

## Summary:

- This is an activity to help students get ready for sorting algorithms and time complexity. In this activity, students will break into groups, and discuss how they will sort a shuffled deck of cards. After figuring out their strategy, there will be a competition between groups to see which group can sort the cards with the least number of comparisons. By the end of the activity, students should understand at a basic level that some algorithms will be faster than others.

---

## Learning Objectives:

- By the end of this lesson, students will be able to:
    - Define the concept and role of sorting algorithms
    - Have a basic understanding of why some algorithms are faster than others

---

## Materials Needed:

- Paper
- Multiple decks of cards (#of students / 4 = #decks needed)

---

## Introduction (5-10 minutes):

- **Hook:** Good algorithms on bad hardware perform better than on supercomputers

---

## Instruction (5-10 minutes):

1. **Explanation of activity:** Describe the activity's goals, including what sorting algorithms do and why algorithms with fewer comparisons are faster. Include an explanation of how to count comparisons.
2. **Modeling:** Give an example of what the cards should look like when sorted.

---

## Planning (5-10 minutes):

- Divide the class into groups and have them figure out their strategy to get the least comparisons.

---

## Activity (10-15 minutes):

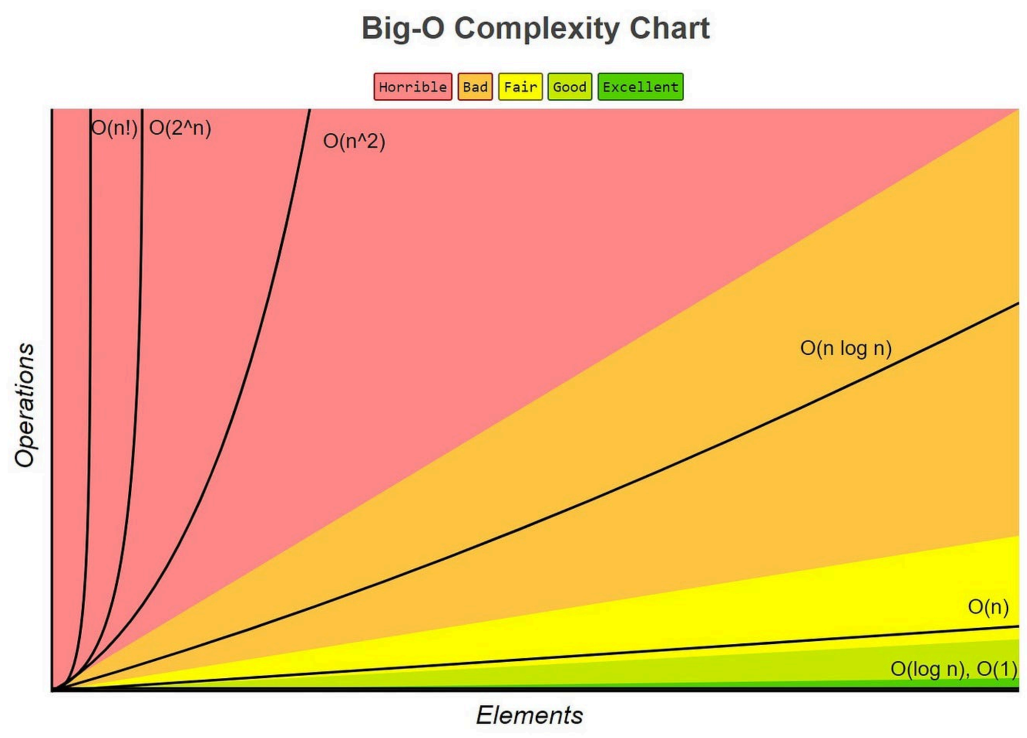
- After each group figures out its strategy, the competition starts.
- The Groups sort the cards and count the number of comparisons. The group with the least number of comparisons wins the competition.

---

## Conclusion of activity (5-10 minutes):

- **Presentation:** Have groups present their strategy and how many comparisons it took to sort the card deck.

- **Reflection/Exit Ticket:** Describe informally how some algorithms will work faster. Show charts like the one below to help students understand time complexity.



## Reflection (Post-Lesson):

- What went well?
- What could be improved?
- Notes for next time:

## Notes

- 52 cards can take a long time for groups to sort. This activity can be done with a smaller number of cards including 26, or 13. Be careful not to go too low or else sorting the cards will be too easy.
- Comparisons are counted instead of time because time complexity is calculated through the number of comparisons. Some algorithms that have resulted in fewer comparisons will take longer for humans to complete. Using time instead of comparisons somewhat defeats the purpose of this lesson as students will not be shown that few comparisons are what makes algorithms fast.

# Unplugged Algorithms Activity Script

**1. Introduction (5-10 minutes)**

- Greet students and introduce the activity.
  Example: “Today, we're going to explore algorithms and time complexity with an unplugged activity."
- Explain the rules and objectives clearly
  Example: “For the lesson I am going to have you break into groups, and discuss with your partners a strategy to sort a shuffled deck of cards with the minimum number of comparisons. This should help us get ready for sorting algorithms and time complexity.”

**2. Activity Setup (5 minutes)**

- Divide the class into groups
- Distribute decks of cards to each group

**3. Planning (5-10 minutes)**

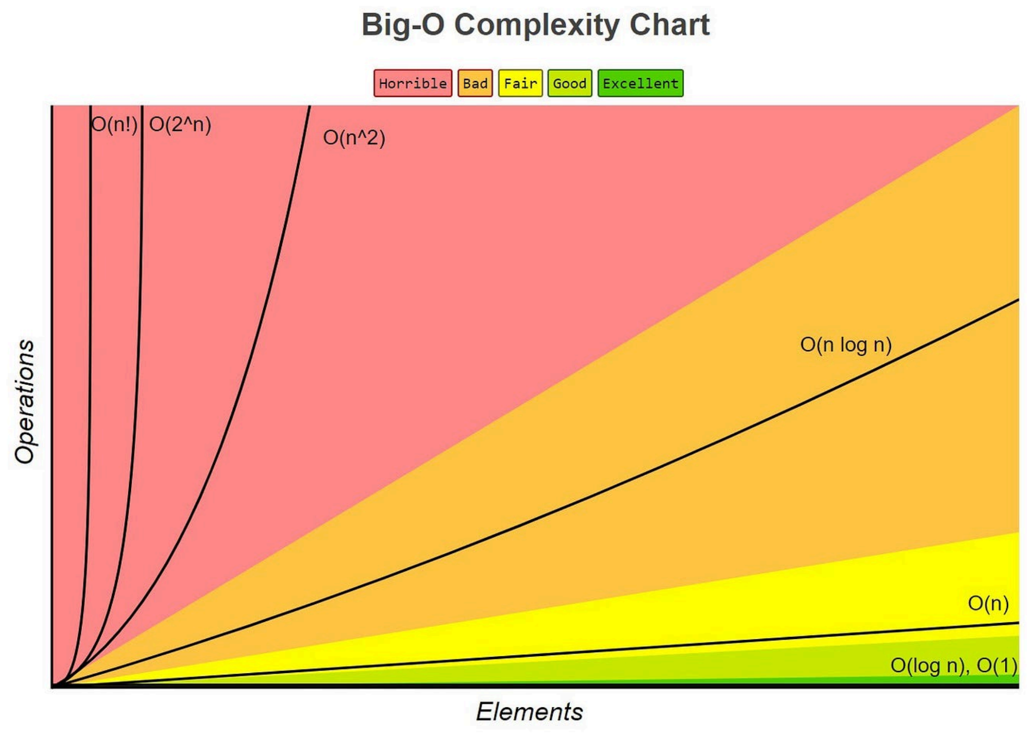
- Now is a good time to explain the activity more in depth.
  Example: “Now that you are in groups I want every group to spend 10 minutes coming up with a strategy that they will use to sort a shuffled deck of cards. I want you to think of a strategy that will result in the least number of card comparisons. For example, if your strategy includes comparing each card to every other card, that strategy will have a lot of comparisons. At the end of this time I will have each group shuffle their cards and then we will start the competition where the winning group is the one who came up with the strategy that has the least amount of card comparisons”
- Here is the time to answer any questions students may have about the activity. Example questions include:
  - Q: “How is a comparison counted?” A: “if you compare a cards value to another card then that is 1 comparison.”
  - Q: “Why are we using comparisons instead of time” A: “Time complexity for algorithms is based off comparisons. You do not need to know what time complexity is yet, just that this is the standard for computing how fast an algorithm is.”
- During this time monitor the room, provide guidance, and answer questions.

**4. Competition(10-15 minutes)**

- There is nothing scripted to say here once the competition starts. This is a time to monitor the classroom and answer any questions that come up.

**5. Conclusion of activity (10 minutes)**

- Go around the room and write down each groups number of comparisons. This will determine the wining group. Feel free to give out fitting prizes to the winners.
- Have each group come to the front of the class and describe their strategy.
- This is a good time to ask reflective questions such as:
    - What challenges did you face?
    - How did you solve them?
    - What did you learn?
- This is also a good time to show the chart below and give a brief overview of time complexity.



Example: “Here is a chart which outlines different time complexities. Algorithms with fewer number of comparisons are shown in the green and are the fastest ones. As algorithms time complexity goes into the yellow or red, the algorithms get much slower.”

**6. Wrap-Up (5 minutes)**

- Thank the class for their attention and gather up all decks of cards.