It's not Magic After All - Machine Learning in Snap*!* using Reinforcement Learning

Sven Jatzlau[1]
Computing Education Research Group
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
sven.jatzlau@fau.de

Tilman Michaeli[1]
Computing Education Research Group
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
tilman.michaeli@fau.de

Stefan Seegerer[1]
Computing Education Research Group
Friedrich-Alexander-Universität Erlangen-Nürnberg
Erlangen, Germany
stefan.seegerer@fau.de

Ralf Romeike
Computing Education Research Group
Freie Universität Berlin
Berlin, Germany
ralf.romeike@fu-berlin.de

***Abstract*—The societal relevance of artificial intelligence is growing rapidly. Advances are primarily driven by machine learning techniques. Recently, many educational tools for teaching AI have been introduced, allowing the user to implement AI features within pedagogical programming environments. However, many of these existing approaches share a common trait: the implementation of the underlying AI framework remains a black box, where external API calls or servers handle the actual computing. For the user, this typically means there is no chance to "see inside" the implementation. As a result, users often cannot gain a deeper understanding of how the "computer is learning". In this paper, we propose design principles for a framework in order to break open that black box. These design principles are implemented in the first part of *SnAIp*, a project aimed at enabling Machine Learning within Snap*!*. The focus of this part is using Reinforcement Learning within Snap*!* games. The corresponding framework enables constructionist learning and is implemented entirely in Snap*!*, which allows for a high degree of transparency and tangibility. Furthermore, we present a curriculum for Reinforcement learning using the *SnAIp* framework. With this, we outline a way to address ML in the classroom using block-based languages, while enabling the all-important "look behind the scenes".**

***Index Terms*—AI, education, Snap!, block-based, programming, machine learning**

I. Introduction

With the growing societal relevance of artificial intelligence (AI), driven primarily by advances in machine learning (ML), AI is now also discussed in pedagogical contexts (e.g. [8], [12], [16]). Consequently, AI is also featured in international curricula (e.g. USA [3] or China [21]) with machine learning being an important aspect.

Block-based languages seem particularly promising for educational settings: not only do they lend themselves to a simple introduction to programming, they enable complex, high-level projects, as well [9]. For this reason, they seem particularly suitable for ML-frameworks to be used in classroom settings. Many of the existing frameworks are aimed at novices and open up a simple and intuitive way for all learners to understand ML concepts – without the hurdle introduced by programming [10].

However, many existing approaches share a common trait: the implementation of the underlying AI framework remains a black box. In many cases, this is because internally, the frameworks rely on external API calls or servers to handle the actual computing. The extent to which a user can "look inside" is therefore extremely limited. As a result, learners often cannot gain a deeper understanding of the actual ML process. In order to eventually enable these learners to analyze the effects AI has on our society in a competent way, understanding ML on a fundamental level is essential.

This paper presents design principles for ML-frameworks suitable for classrooms, as well as an outline of the Reinforcement Learning part of *SnAIp*, which incorporates these design principles. *SnAIp* is based on an implementation entirely in the Snap*!*-environment to enable constructionist learning by "breaking open the black box". Finally, we present a corresponding curriculum to incorporate Reinforcement Learning into the classroom.

II. Related Work

Recently, numerous tools have been introduced to allow learners to use machine learning in programming. A large number of these tools use block-based languages. A popular example is the British Machine Learning for Kids [13]. Machine Learning for Kids provides an online platform that supports Supervised Learning, allowing the Computer to recognize text, images, sets of numbers, or sounds within Scratch projects. The user needs to provide training data, while the learning process is done on external servers.

Williams et al. introduced a hands-on toolkit called PopBots that focuses on Pre-K and Kindergarten children [19]. PopBot builds on the same blocks as ScratchJr and provides teaching

[1]These authors contributed equally to this work.

materials for Knowledge-based Systems, Supervised Learning, and Generative Music.

Druga introduced Cognimates, a framework for learning how to build games, program robots, and train AI models [5]. The focus of this framework is on the interaction with social robots, programming them, and training them [6], [7], [18]. The learning process is again hidden from the user, instead taking place on external servers.

Kahn et al. developed a framework for AI programming in Snap*!* [11] named ecraft2learn. This framework provides a library including many different AI-related blocks that enable the user to experiment with a broad range of AI concepts within the Snap*!*-environment, including image recognition, text recognition, and more. The blocks merely offer an interface for the underlying API. Internally, the framework then utilizes APIs or server-based services for the actual computation.

In summary, there are several approaches aimed at making Machine Learning accessible for all groups of learners. Most educational tools focus upon Supervised Learning (one of the three paradigms of machine learning algorithms) due to the paradigm's widespread distribution and common examples such as image recognition. These approaches incorporate the usage of this Machine Learning paradigm into supporting a constructionist style of learning, but they lack the possibility to explore how these learning processes take place. The Machine Learning happens out of the user's sight, e.g. with an API call or a JavaScript function. As a result, ML remains a magical mystery.

III. Design Principles for Machine Learning in Block-based Languages for Learners

As outlined above, existing materials focus more on using pre-compiled APIs for ML without the user having to, or being able to, visualize and modify the underlying algorithm. By contrast, our approach aims to emphasize the actual implementation by opening up the black box, aligns with constructionist learning theory [14], and embodies the following design principles:

- **Look behind the scenes:** Instead of only applying libraries or making API calls which "do the magic", users should be able to look into the black box, and even make adaptations to the ML algorithms based on their needs – at least to some extent. To this end, languages such as JavaScript should be avoided completely if possible.
- **Everything inside the tool:** Many other systems need extra services on which models are trained or rely on API calls in the background. Doing so tends to reinforce the notion of block-based languages being "learning languages", and if the goal is to do "real programming", they are not sufficient. Therefore, to contradict this perception, everything should work within the framework.
- **Useful projects over learning projects:** according to the constructionist learning theory [14], users should be able to create their own, personally-meaningful projects and use the system to enrich them. This poses a contrast to "learning projects", where, often, artificial problems and examples are chosen, because they are suitable for visualizing certain concepts.
- **Models as artifacts:** With block-based environments that allow users to store and save variables locally, models can become even more concrete, which supports the constructionist idea of artifacts as tangible results. This allows several new and interesting ways to integrate these ideas into the classroom, i. e. as a competition for students to compare their models, discuss, and learn from each other.

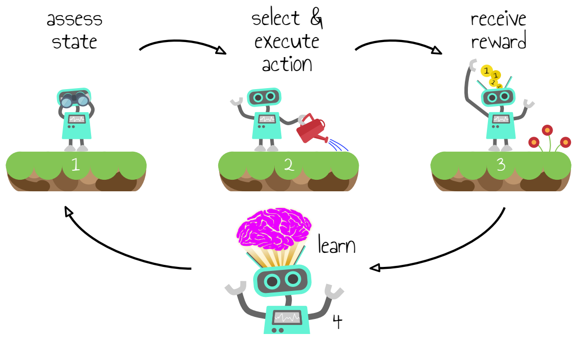


Fig. 1. The agent's RL loop.

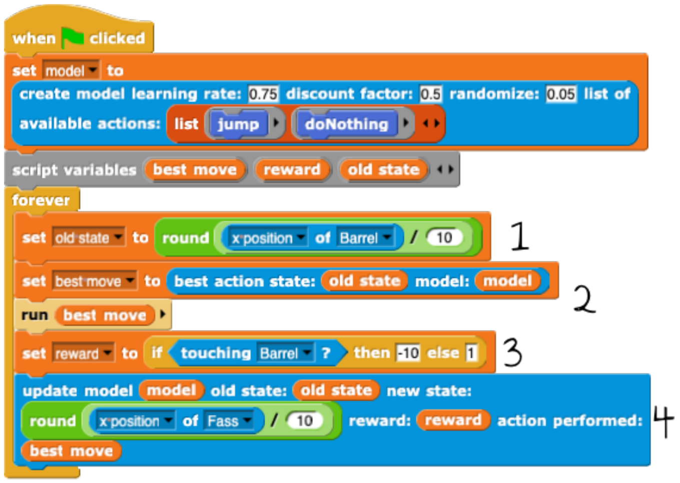
SnAIp follows these design principles. It is intended for use in high school classrooms (ages 14 and older) and has been piloted with a year ten class. While there are many approaches to Supervised Learning, for Reinforcement Learning (RL), another one of the three paradigms of ML, suitable approaches and tools are missing. As games in particular are a common application of block-based languages [20], applying RL to Snap! seems promising. Due to the parallels between human learning and RL, the first part of *SnAIp* focuses on RL.

IV. SnAIp: Reinforcement Learning Implemented in Snap*!*

A. Reinforcement Learning and Q-Learning

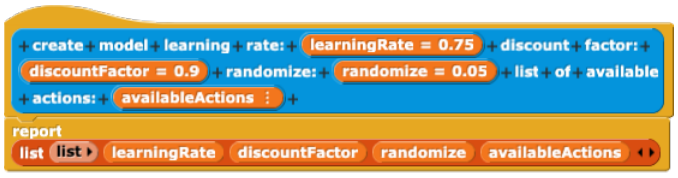
Reinforcement learning is a paradigm of machine learning inspired by psychology: The agent learns through reward and punishment. It learns to master a certain task autonomously through interaction with its environment by attempting to maximize the total reward [1]. Classic arcade video games are a popular way to develop, test, and explore RL algorithms (e.g. [2]). At its core, the agent in RL follows the learning loop depicted in figure 1.

For *SnAIp*, we use an RL-algorithm called Q-Learning [17]. This algorithm generates a Q-table that the agent uses to find the best action for a state. The Q-value in the Q-table indicates the reward to be expected for carrying out an action in the current situation. By associating a certain state with a certain reward, the agent learns (= the probability of showing this behavior again increases).



```
when [flag] clicked
set model to
    create model learning rate: 0.75 discount factor: 0.5 randomize: 0.05 list of
    available actions: list jump doNothing
script variables best move reward old state
forever
    set old state to round x position of Barrel / 10          1
    set best move to best action state: old state model: model   2
    run best move
    set reward to if touching Barrel ? then -10 else 1         3
    update model model old state: old state new state:
    round x position of Fass / 10 reward: reward action performed:   4
    best move
```

Fig. 2. Sample script for an agent.



```
+create+model+learning+rate:+ learningRate = 0.75 +discount+factor:+
discountFactor = 0.9 +randomize:+ randomize = 0.05 +list+of+available
+actions:+ availableActions :+
report
list list learningRate discountFactor randomize availableActions
```

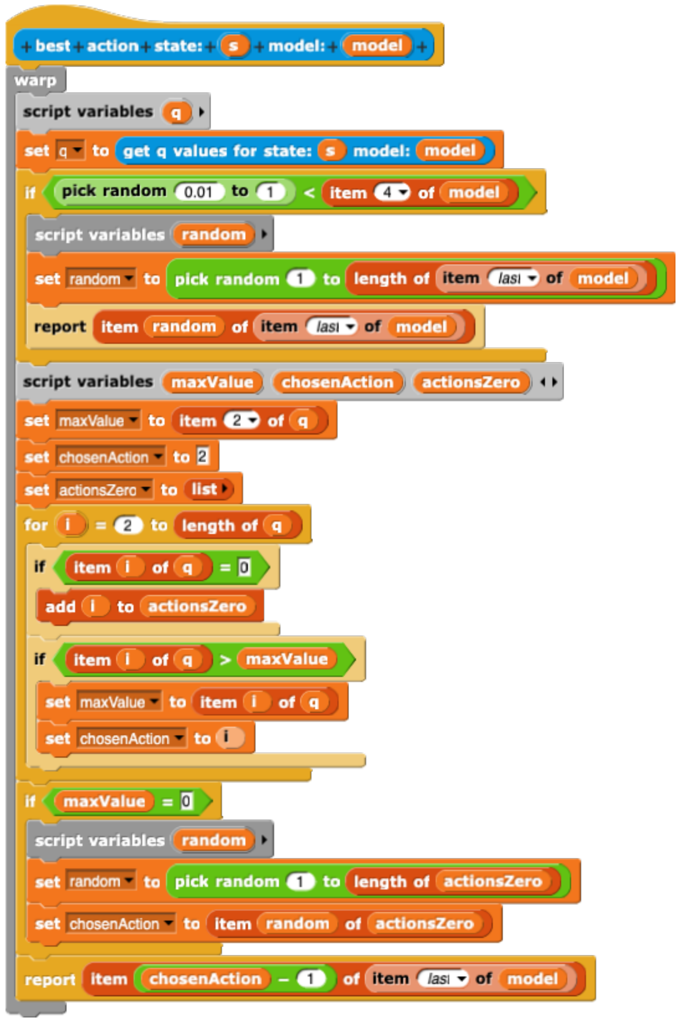
Fig. 3. Implementation of Create Model.

B. The SnAIp-*framework*

A learning agent can be realized in *SnAIp* in a simple and intuitive way by implementing the learning loop in the corresponding sprite (for a sample implementation where an agent is jumping over rolling barrels see figure 2). Before the learning loop can be initiated, an underlying data structure is needed. For this purpose, the first block (displayed in figure 3) simply creates the model, which consists of the Q-Table (initially an empty list), and the parameters *learning rate*, *discount factor*, a *random factor*, and the list of possible actions the agent can choose from.

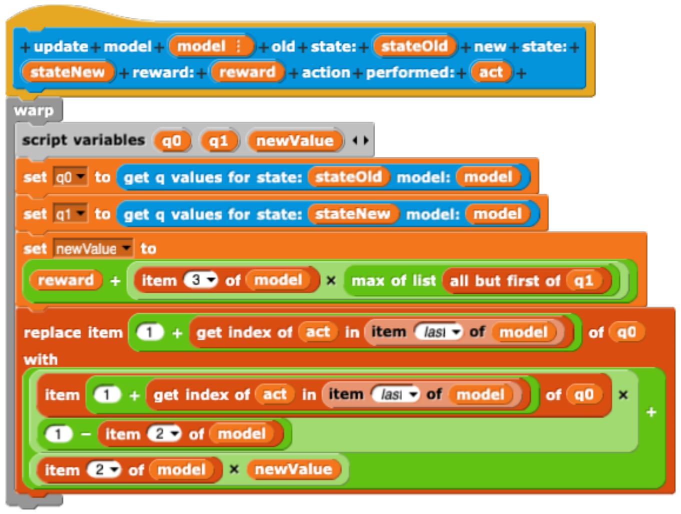
Afterward, the agent follows the previously-specified **learning loop**: In step 1, the current state is assessed, in the example the agents distance to the barrel. Next, the best-action-block is used and the Q-values for the current state are retrieved from the model. With a certain probability (random factor) a random action is executed, i. e. a random element from the list of available actions is returned. In all other cases, the index of the highest value in the row is searched for. This entry represents the most positively evaluated action for this state. In case the maximum is 0, the algorithm shows a random action again. Essentially, this is a simple table look up.

In step 3, the reward is computed, based on the outcome of the action. Afterward, the actual learning takes place: the "update model"-block updates the Q-Table by applying the reward to the actions performed and considering an estimation of future Q-values, resulting in new values for this particular state. For future situations, this means the agent will be more likely to show positively reinforced behavior(see figure 5).



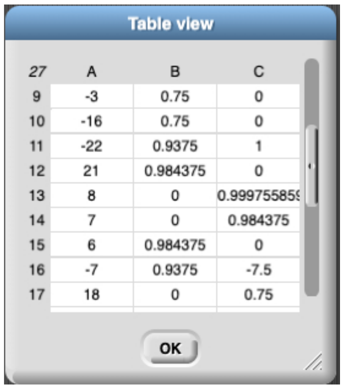
```
+best+action+state:+ s +model:+ model +
warp
    script variables q
    set q to get q values for state: s model: model
    if pick random 0.01 to 1 < item 4 of model
        script variables random
        set random to pick random 1 to length of item last of model
        report item random of item last of model
    script variables maxValue chosenAction actionsZero
    set maxValue to item 2 of q
    set chosenAction to 2
    set actionsZero to list
    for i = 2 to length of q
        if item i of q = 0
            add i to actionsZero
        if item i of q > maxValue
            set maxValue to item i of q
            set chosenAction to i
    if maxValue = 0
        script variables random
        set random to pick random 1 to length of actionsZero
        set chosenAction to item random of actionsZero
    report item chosenAction - 1 of item last of model
```

Fig. 4. Implementation of the block to retrieve the best action.



```
+update+model+ model :+old+state:+ stateOld +new+state:+
stateNew +reward:+ reward +action+performed:+ act +
warp
    script variables q0 q1 newValue
    set q0 to get q values for state: stateOld model: model
    set q1 to get q values for state: stateNew model: model
    set newValue to
        reward + ( item 3 of model × max of list all but first of q1 )
    replace item 1 + get index of act in item last of model of q0
    with
        item 1 + get index of act in item last of model of q0 ×
        1 − item 2 of model
        +
        item 2 of model × newValue
```

Fig. 5. Implementation of the block that updates the Q-Table.



27	A	B	C
9	-3	0.75	0
10	-16	0.75	0
11	-22	0.9375	1
12	21	0.984375	0
13	8	0	0.99975585
14	7	0	0.984375
15	6	0.984375	0
16	-7	0.9375	-7.5
17	18	0	0.75

Fig. 6. Q Table for easy exploration in Snap*!*.

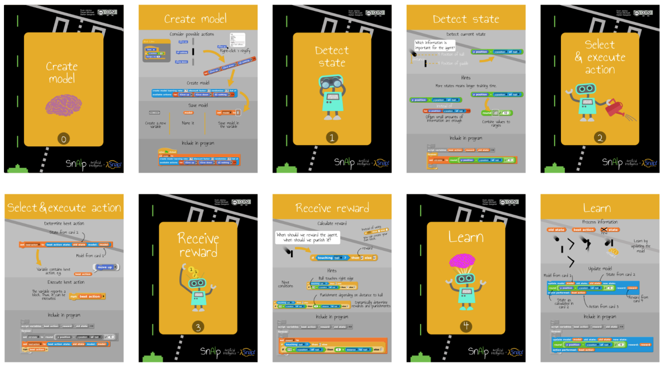


Fig. 7. Coding cards used in the phase *Pimp my Game*.

As per our design principles outlined above, all behavior is implemented in blocks within Snap*!*. There are no wrappers for external JavaScript code or server calls.

We chose Q-Learning for its speed, as – for small problem spaces, – training can be very fast: depending on the selected state description, effects can become visible in less than three minutes. With time often being a crucial resource in classroom settings, Q-Learning seems particularly suitable. In addition, the Q-Learning algorithm can not only be easily understood and visualized by using a table (where the states and Q-values for the respective actions are stored), but can also be implemented by the students themselves. However, this approach also has limitations: The speed at which the agent learns the desired behavior depends on the number of different possible states. If the number of states is too high, and cannot be sensibly limited because the example is too complex, the example is not suitable for classroom settings – the learning process would simply take too long for a single lesson.

V. *SnAIp* in the Classroom

While the *SnAIp* framework can be used individually, it can also be incorporated into a comprehensive curriculum. In the following, we will describe our approach on how to use Reinforcement Learning in Snap*!* within a curriculum. The curriculum is intended for use in upper high school classrooms (ages 14 and older) and roughly divided into 4 parts:

a) What is Reinforcement Learning: For the introduction of Reinforcement Learning without a computer, we used an Unplugged activity to introduce the underlying concept and the learning loop. In this activity, two students each play a game of Hexaspawn against each other [4].

b) Getting started with RL in Snap!: The next step is to help a monkey-agent learn and support him in his banana hunt. This task is introduced with a puzzle activity: the students receive a template for the game *Banana Hunt*. It contains all the relevant blocks for the self-learning agent – but not in the correct order. The students are asked to put them in the right order. Afterward, they explore the program and describe how the agent learns.

c) Pimp my Game: The next step for the students is to add RL to their own games. In principle, RL can be applied to any game. However, there are some rules and restrictions for selecting an example for use in a regular lesson (primarily motivated by time constraints). Examples that are particularly suitable for use in class include games like Breakout, Pong or Flappy Bird. Coding Cards, based on Scratch Cards [15], are available as supporting material (see 7).

d) Behind the Scenes: The next step takes a closer look at how the RL concept is implemented with Q-Learning on a technological level. The students are instructed to view the table that represents what has been learned (see figure 6). Students are encouraged to observe how the values change over time. Building upon that, students can fully grasp the underlying algorithm and make adjustments and optimizations. This phase highlights the advantages of using Snap*!* for the entire implementation: every block can be inspected, every variable can be monitored, and because the framework is made with blocks, the flow of the entire program is made visible and tangible.

VI. Conclusion

In summary, we propose a way to address ML by using block-based languages while enabling the important "breaking open the black box". Furthermore, we provide a comprehensive curriculum for RL in Snap*!* that supports teachers in incorporating these concepts into the classroom.

By being built entirely in Snap*!*, our framework enables a look into the blocks. It emphasizes a constructionist perspective, as users can enrich their own games and projects with RL. Furthermore, we avoid using external libraries or servers – everything is done right there, in the Snap! window on the user's PC. As a result, learners can gain a deeper understanding of how ML actually learns and are able to analyze the effects AI has on our society in a competent way.

In the next phase of the development of *SnAIp*, we intend to transfer the aforementioned design principles to other paradigms of ML, thereby providing additional ways for learners to engage with and understand other major concepts of ML.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[2] Luis Carlos Cobo, Peng Zang, Charles Lee Isbell Jr, and Andrea Lockerd Thomaz. Automatic state abstraction from demonstration. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[3] CSTA. About the CSTA K-12 computer science standards, 2017.

[4] Paul Curzon and Peter W McOwan. The sweet learning computer, 2016.

[5] Stefania Druga. *Growing up with AI: Cognimates: from coding to teaching machines*. PhD thesis, Massachusetts Institute of Technology, 2018.

[6] Stefania Druga, Randi Williams, Cynthia Breazeal, and Mitchel Resnick. Hey google is it ok if i eat you?: Initial explorations in child-agent interaction. In *Proceedings of the 2017 Conference on Interaction Design and Children*, pages 595–600. ACM, 2017.

[7] Stefania Druga, Randi Williams, Hae Won Park, and Cynthia Breazeal. How smart are the smart toys?: children and parents' agent interaction and intelligence attribution. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*, pages 231–240. ACM, 2018.

[8] Clint Andrew Heinze, Janet Haase, and Helen Higgins. An action research report from a multi-year approach to teaching artificial intelligence at the k-6 level. In *First AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 2010.

[9] Sven Jatzlau and Ralf Romeike. How High is the Ceiling? Applying Core Concepts of Block-based Languages to Extend Programming Environments. In Egl Jasut Valentina Dagien, editor, *Constructionism 2018: Constructionism, Computational Thinking and Educational Innovation: conference proceedings*, pages 286–294, 2018.

[10] Ken Kahn, Rani Megasari, Erna Piantari, and Enjun Junaeti. Ai programming by children using snap! block programming in a developing country. *EC-TEL Practitioner Proceedings 2018: 13th European Conference On Technology Enhanced Learning*, 2018.

[11] Ken Kahn and Niall Winters. Child-friendly programming interfaces to ai cloud services. In *European Conference on Technology Enhanced Learning*, pages 566–570. Springer, 2017.

[12] Martin Kandlhofer, Gerald Steinbauer, Sabine Hirschmugl-Gaisch, and Petra Huber. Artificial intelligence and computer science in education: From kindergarten to university. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–9. IEEE, 2016.

[13] Dale Lane. Explaining artificial intelligence. *Hello World*, 4, 2018. https://helloworld.raspberrypi.org/issues/4.

[14] Seymour Papert and Idit Harel. Situating constructionism. pages 1–13, 1991.

[15] Natalie Rusk and other members of the Scratch Team. *Scratch Coding Cards: Creative Coding Activities for Kids*. No Starch Press, 2017.

[16] David Touretzky, Christina Gardner-McCune, Fred Martin, and Deborah Seehorn. Envisioning AI for K-12: What should every child know about AI? In *"Blue sky talk" at the Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19)*, 2019.

[17] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[18] Randi Williams, Christian Vázquez Machado, Stefania Druga, Cynthia Breazeal, and Pattie Maes. My doll says it's ok: a study of children's conformity to a talking doll. In *Proceedings of the 17th ACM Conference on Interaction Design and Children*, pages 625–631. ACM, 2018.

[19] Randi Williams, Hae Won Park, Lauren Oh, and Cynthia Breazeal. Popbots: Designing an artificial intelligence curriculum for early childhood education. *The Ninth Symposium on Educational Advances in Artificial Intelligence*, 2019.

[20] Amanda Wilson, Thomas Hainey, and Thomas M Connolly. Using scratch with primary school children: an evaluation of games constructed to gauge understanding of programming concepts. *International Journal of Game-Based Learning (IJGBL)*, 3(1):93–109, 2013.

[21] Yanfang Yu and Yuan Chen. Design and development of high school artificial intelligence textbook based on computational thinking. *Open Access Library Journal*, 5(09):1, 2018.