Revisiting RumbleBlocks with Apprentice Learning: Examining and Learning Gameplay with AL

Cayden Codel
ccodel@andrew.cmu.edu
15-400 student

Ken Koedinger
krk@cs.cmu.edu
Advisor, Human-Computer Interaction Institute

Erik Harpstead
eharpste@cs.cmu.edu
Advisor, Human-Computer Interaction Institute

ABSTRACT

Apprentice learning has been shown to be an effective way to model student learning in several educational contexts, such as fraction addition. At CMU, the HCII has developed the Apprentice Learner Architecture (AL) that models student learning. In my research, I examine how AL learns and behaves in the educational game RumbleBlocks. Initial results show that AL can learn the skills to complete a few levels, but does not reach mastery given the current training methods and data.

KEYWORDS

educational data mining, apprentice learning

ACM Reference Format:
Cayden Codel, Ken Koedinger, and Erik Harpstead. 2020. Revisiting RumbleBlocks with Apprentice Learning: Examining and Learning Gameplay with AL. In *15-400*. ACM, New York, NY, USA, 4 pages. https://doi.org/XX.XX/XXXXXX.XXXXXXX

1 INTRODUCTION

The Apprentice Learner (AL) Architecture has found success as a cognitive model of student learning [3] [4], particularly for tasks that can be performed in an intelligent tutoring system (ITS). The goal of cognitive models such as AL is to create agents that can solve problems in the same ways that humans do. Successful models are useful to researchers and educators in several ways. First, good models give researchers insight into how humans learn. Second, trained agents can test out new assignments and worksheets that instructors have prepared. For example, a teacher can see if a worksheet on fraction addition is too difficult for her students, or if the ordering of problems best contributes to student learning [10]. Third, trained agents can be used to construct an expert model for ITSs. [3] shows promise that using agents to author expert models in this way reduces overhead for instructors in both time and training needed for export model creation.

Despite its success, AL in particular has been evaluated on only a few general formats and a narrow range of tasks, almost all housed in ITSs. Thus, there is an opportunity to test AL's ability to learn like humans for other tasks, in other mediums. One such medium is educational video games.

The goal of an educational video game is to teach students something in an interactive, fun way. And just like normal instruction, the way an educational video game goes about teaching students can range from almost classroom-like to a free sandbox environment with little direction or instruction. The wide range of game types and appearances makes investigating how educational games teach students an interesting field of study in and of itself. But here, I examine how AL learns in an educational game format.

The game AL will be learning from is called RumbleBlocks. RumbleBlocks is a game designed by master's students in CMU's Entertainment Technology Center to teach young children how to build stable towers [9]. The game presents players with a narrative of aliens crash-landing their UFOs on planets, and players must use differently sized blocks to construct towers that are high enough to reach the aliens and wide enough to cover blue energy checkpoints, but sturdy enough to survive an earthquake and keep the UFOs on top of the towers. See Figure 1 for a screenshot of gameplay.

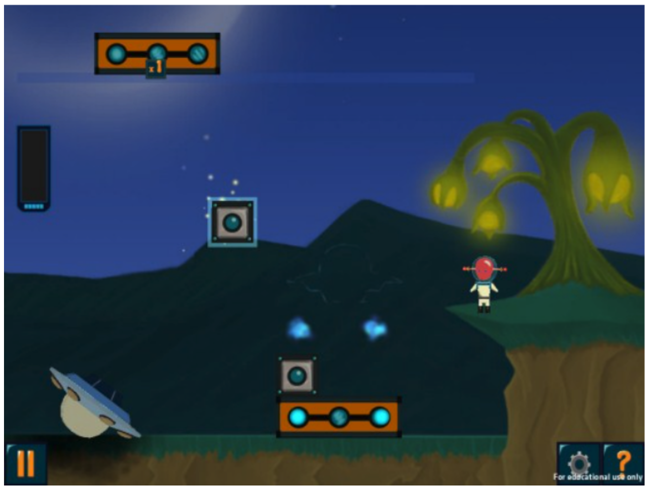


Figure 1: Screenshot of RumbleBlocks gameplay. Players must stack building blocks to cover the energy checkpoints and get the UFO to the alien. The tower must survive an earthquake upon completion.

My advisors have collected RumbleBlocks student play data for the purposes of analyzing the structures students created to see if the game was correctly teaching its concepts of structure stability [6] [7]. Part of the original RumbleBlocks game was to present constructed towers to students and ask them which would survive the earthquake as a method of mid-game testing. Two structures were pictured: one demonstrated one of the three concepts the game sought to teach its players (symmetry, wide bases, low center of gravity), while the other was deficient in one of the three areas and would fail to keep itself upright or the UFO on top. Students were asked to select which tower was more stable. Additional work has been done on that data to investigate whether AL is able to classify towers with the same success as students. Experiments showed that AL learned as well as students did on tower stability classification [8].

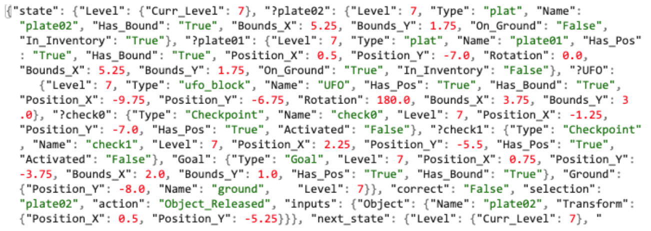
However, classifying structures is an easier task than constructing them from scratch. In this paper, I present the methods and initial results of training AL to play RumbleBlocks. So far, the continuous game state and non-optimal game replay data is not as effective as training AL, but by instilling enough prior knowledge into AL and cleaning up the replay data, some progress has been made towards getting AL to train quickly and play successfully.

2 REPLAYING RUMBLEBLOCKS

Like most machine learning algorithms, AL learns best when given labeled examples. But AL is not a supervised learning agent. Instead, AL learns through apprentice learning. At its core, apprentice learning reflects how students, traditionally those found in trade schools, learn from their teachers. Through a close relationship of demonstration and correctness feedback on worked examples, apprentice learning algorithms come to solve problems in the same ways as the expert system that taught them. When the expert system is human performance data, then apprentice learning algorithms come to behave like humans in both how they solve problems and in the errors they make. Part of developing cognitive models of human learning is to get the agents to perform in the same ways as humans do, and so apprentice learning lends itself as a good model of how students learn.

The general goal, then, is to extract from human performance data a typical model of how humans perform. In video game contexts, this is called a *persona* [1] [2]. For many games, a dominant strategy exists for beating levels or winning the game, and AIs may be trained to find that dominant strategy. For puzzle games such as RumbleBlocks, it is likely easy to train agents to play the game flawlessly, particularly using deep reinforcement learning techniques (games with a much larger solution space, like chess and Go, have been "solved" with RL AI like AlphaZero [5]). But training an agent in this way gives an optimal player, not a persona. Unless humans play optimally, other methods must be used to achieve an agent that behaves like a human and not as an expert. The persona literature finds that training agents on human gameplay provides a good basis for creating personas. Thus, with RumbleBlocks play data in hand, AL can be leveraged as a method for making RumbleBlocks personas.

In order to play the game, AL must receive the game's current state, process it, and then send back the action it wishes to take. Through the Selection, Action, Input (SAI) method, AL can extract important relations between the various objects in the game. For RumbleBlocks in particular, the selection is the block or item the player or agent wishes to manipulate, the action is how that block or item is manipulated (e.g. grabbing, dropping, rotating), and the input is the game state that induced the player or agent to take that action. See Figure 2 for an example of an SAI vector that is sent to AL to request it to take an action.



```
{"state": {"Level": {"Curr_Level": 7}, "?plate02": {"Level": 7, "Type": "plat", "Name":
  "plate02", "Has_Bound": "True", "Bounds_X": 5.25, "Bounds_Y": 1.75, "On_Ground": "False",
  "In_Inventory": "True"}, "?plate01": {"Level": 7, "Type": "plat", "Name": "plate01", "Has_Pos"
  : "True", "Has_Bound": "True", "Position_X": 0.5, "Position_Y": -7.0, "Rotation": 0.0,
  "Bounds_X": 5.25, "Bounds_Y": 1.75, "On_Ground": "True", "In_Inventory": "False"}, "?UFO":
    {"Level": 7, "Type": "ufo_block", "Name": "UFO", "Has_Pos": "True", "Has_Bound": "True",
  "Position_X": -9.75, "Position_Y": -6.75, "Rotation": 180.0, "Bounds_X": 3.75, "Bounds_Y": 3
  .0}, "?check0": {"Type": "Checkpoint", "Name": "check0", "Level": 7, "Position_X": -1.25,
  "Position_Y": -7.0, "Has_Pos": "True", "Activated": "False"}, "?check1": {"Type": "Checkpoint"
  , "Name": "check1", "Level": 7, "Position_X": 2.25, "Position_Y": -5.5, "Has_Pos": "True",
  "Activated": "False"}, "Goal": {"Type": "Goal", "Level": 7, "Position_X": 0.75, "Position_Y":
  -3.75, "Bounds_X": 2.0, "Bounds_Y": 1.0, "Has_Pos": "True", "Has_Bound": "True"}, "Ground":
  {"Position_Y": -8.0, "Name": "ground",    "Level": 7}}, "correct": "False", "selection":
  "plate02", "action": "Object_Released", "inputs": {"Object": {"Name": "plate02", "Transform":
  {"Position_X": 0.5, "Position_Y": -5.25}}}, "next_state": {"Level": {"Curr_Level": 7}, "
```

Figure 2: An example of RumbleBlocks state sent to AL. Note the rounded position and rotation values, discussed below.

AL then consults the skills it has learned so far to determine if it can take an action. If AL's various learning mechanisms produce two or more candidates, then AL takes the best one. The RumbleBlocks engine then performs this action and evaluates the state of the game. If the level has been completed, then AL is given correctness feedback. If the game is not yet complete, more actions are requested, or, if too many actions have been taken, the level is ended early and AL is given feedback labelled "incorrect."

If instead AL finds that it does not have enough knowledge to take any actions, it sends back an empty response. RumbleBlocks interprets this as AL needing a demonstration of what to do next, and so the next human action in the replay sequence is performed. This way, entire levels can be "played" by AL as demonstration for how towers are constructed.

During training, raw game state containing position and rotation vectors of the blocks and the UFO "as-is" confused AL, leading to little to no actions taken. The lack of actions from AL during training was determined to come from the floating-point precision of the position vectors, as serializing the JSON objects the game state was stored in would obscure, for example, orthogonal relationships between the blocks that would be present if the positions were instead integers. Thus, the positions and rotations are rounded to a configurable value before being given to AL.

In addition to the general replay-action-feedback loop, AL can be given defined prior knowledge of how to play the game before training. Given enough prior knowledge, AL could theoretically play the game from the start, but some of the learning mechanisms, particularly the *when*-learner, has not yet associated the skills with the conditions necessary to fire them. Among these skills are the ability to rotate blocks, place blocks orthogonally to each other, and to snap blocks to energy checkpoints necessary to complete a subset of the levels. While prior knowledge speeds up training, it is also a method of "cheating," and so the fewer prior skills AL needs at the start of training in order to develop into an agent capable of playing like a human, the better.

3 EVALUATING AL'S PERFORMANCE

Ideally, after each of AL's actions, AL receives feedback based on how correct the action was. Earlier versions of AL all but demanded feedback after each action to ensure training would go smoothly. More recent versions do allow for backpropagating correctness feedback, which the end-of-level correctness labelling uses. However, training is generally streamlined when given immediate feedback. Thus, RumbleBlocks has a toggle that allows for immediate feedback of actions taken by AL. To facilitate the conditions necessary to complete a level, AL is rewarded for covering energy checkpoints and is punished for removing blocks from previously lit checkpoints. The full reward is received when all checkpoints are covered.

Along with rewarding checkpoint covering, AL is punished for taking the same action more than once. The RumbleBlocks replay engine tracks all actions taken by AL, and if it detects that the action is identical to one made before, then AL receives immediate feedback with negative reward. In theory, this should dissuade AL from getting stuck in a loop. In practice, however, AL may continue to take the same actions over and over, even in face of the negative rewards. To prevent such a loop, the game engine also has an option for training to force the next human action to be taken instead of AL's if the same action is seen too many times in a row. This policy ensures that levels are completed in a finite amount of time.

Finally, to "incentivize" AL to complete levels in as few actions as possible, various mechanisms could be used, including a diminishing reward on level completion based on the number of actions taken. The type of AL learner used in the latter half of the semester has an internal reinforcement learning model which has a discount factor built in, already persuading AL to take as few actions as possible. Therefore, the toggle to diminish the weight of the correctness feedback on level completion was not implemented, as its function was already present in the AL agent being trained.

4 PARSING HUMAN DATA

This semester, I had access to data from play sessions conducted several years ago. At the time of the user studies, as students played, any interactions made in the game environment were recorded. The interactions that were cause for a snapshot of the game state included any time a block collided with another block or the ground, any time a block was rotated or removed from inventory, or whenever the UFO changed position. Such "diff"-based recording allows for accurate playback of student actions, recreating the structures the students made at a pace controllable by the RumbleBlocks game engines and configurable settings.

However, as a way to train AL, the student play data had far too many extraneous actions. Often, blocks would be dragged across the ground, or a tower would topple halfway through its construction. During end-of-level training, AL would essentially be taught that these intermediate actions or mistakes are "correct," which inhibits training (or, at least, training AL to play the game). To ameliorate this, a toggle was added to the replayer engine to take only those actions that were "meaningful" by running through the human play data backwards and taking only the last occurrence of block manipulation for each block ID. With the option enabled, the tower is constructed more directly, and as a bonus, AL trains much faster.

5 INTERACTIVE TRAINING

Early in its development, the RumbleBlocks replayer had an interactive training feature, which was deprecated. The interactive training feature allowed for more fine-grained feedback and demonstration from a human trainer, essentially making the human play data "live." I liked the concept, especially given the issues discussed above regarding the human data I had at my disposal. However, I couldn't get the interactive training feature to work after a bit of tinkering, so I let it be and left it for future work.

6 INITIAL RESULTS

Getting AL to correctly play RumbleBlocks is difficult, given the open-endedness of the game and the continuous nature of the 2D environment the blocks exist in. What's more, AL has trouble producing "original" x- and y-coordinate values, and can only take actions based on the state provided to it and those states derivable from there, e.g. through provided primitive operations such as addition of an x-coordinate by 1 or through defined prior skills. While the manipulation of state through primitive operations works well for integral or numerical tasks, such as fraction addition, in a game environment, it limits the actions AL can naturally take.

I attempted to combat the restricted action output of AL in several ways. The first was to give AL a primitive operation of shifting any block one unit an any orthogonal direction. Aligning the unit of shift to the unit used to discretize the state space effectively gives AL the ability to take any action in the action space. A few things hampered this method, though. One issue was that, given the option, AL prefers to manipulate the state in "working memory" before committing to an action, allowing for multiple operation actions to be performed. But given the high degree of freedom available to AL when given the opportunity to shift blocks by fractions of its width will send AL into a "thinking spiral," where it spends all its time in its working memory and not taking actions. While this behavior may eventually resolve itself due to the discount factors in the RL portion of its algorithm, the training I conducted on my computer didn't indicate it ever would, so I looked for other solutions.

The next was to restrict the action space to only those actions that are meaningful to completing RumbleBlocks levels. This was encapsulated in the prior skills given to AL. In theory, the only skills needed to complete levels were knowing how to place blocks in checkpoints and how to stack blocks next to and on top of each other. However, when training, AL matches these prior skills to the SAI inputs it is given, and student play data didn't always line up with the prior skills given to AL. Thus, while the prior skills were sufficient for completing levels, they weren't expressive enough for AL to copy what humans did, limiting the effectiveness of the human actions during training.

Even with the complications of matching human play data to prior skills, AL did show some signs of learning during training. On simpler levels, AL would often be able to place the first block or two of a more complex structure that would be able to complete a level. See Figure 3 and Figure 4 below. So while AL never completed an entire level on its own, it did make progress towards it.

In general, though, AL is still a ways off from learning how to play RumbleBlocks. But through careful tweaking of prior skills, human play data vetting, and immediate feedback rewards, it seems very possible that AL could learn to play RumbleBlocks, and if given enough examples, could learn to play much like the humans it trains on do.

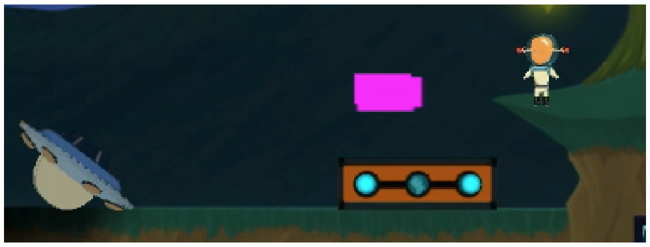


Figure 3: An instance where AL demonstrated positive behavior. The block is placed to cover both energy checkpoints.

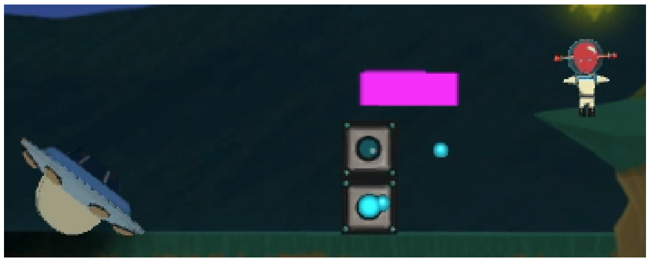


Figure 4: Another instance where AL demonstrated positive behavior. To complete the level, two sets of two blocks need to be stacked like the pair pictured here.

7 FUTURE WORK

Currently, the bottleneck in training lies in how AL interprets and manipulates the game state to take its actions. The closer AL gets to being able to take exactly those actions taken by students, the better. There are various ways of accomplishing this feat.

The first is to pick out those gameplay sequences that are "nice," in that the structures made by the students are orderly, built step-by-step, and can be expressed using the prior skills given to AL. Unfortunately, classifying which sequences are "nice" would likely have to be done by hand and vastly limits the number of training examples available to AL.

Another method would be to increase the expressiveness of prior skills to more closely match the actions taken by students. While this would increase AL's solution space, it provides the greatest change of getting AL to play RumbleBlocks "from scratch," at least with how the replayer is designed at the current moment.

A third would be to give AL the meta-skills/primitive operations needed to access the full action space, but put mechanisms in place to limit the number of non-actions AL can take in its working memory before being forced to take an action. This would be an edit on AL's side, rather than in the RumbleBlocks engine, and so would be more core to the computational model and not necessarily how it is trained.

The way the game state is discretized could also be played around with, as well as how AL handles continuous action spaces. After all, not every task can be reproduced in an ITS with a well-defined interface.

8 LESSONS LEARNED

Inheriting software from any source, research or industry, brings with it the risk of out-of-date versioning, and I certainly faced that this semester. RumbleBlocks was developed in a prior version of the Unity game engine, and a nontrivial amount of time was spent updating RumbleBlocks to the newer versions, as it would not work on my computer in its original version. Thus, I have learned that, moving forward, I should budget more time to installing and becoming familiar with third-party pipelines.

In general, I got to read literature in the cognitive modeling and human learning fields, which I haven't been exposed to in my studies here at CMU. I even got to read a paper by Newell and Simon!

ACKNOWLEDGMENTS

I would like to thank my mentors, Dr. Koedinger and Dr. Harpstead, and particularly thank Dr. Harpstead's weekly meetings with me to diagnose problems with AL and the replayer in general. I hope that we can get AL playing RumbleBlocks sometime in the future —or at least prepare the groundwork for AL to play some other game.

REFERENCES

[1] Alessandro Canossa Anders Tychsen. 2008. Defining Personas in Games Using Metrics. *FuturePlay* (Nov. 2008).
[2] Antonios Liapis Julian Togelius Christoffer Holmgard, Michael Cerny Green. 2019. Automated Playtesting With Procedural Personas Through MCTS with Evolved Heuristics. *IEEE Transactions on Games* (July 2019).
[3] Rony Patel Kenneth R. Koedinger Christopher MacLellan, Erik Harpstead. [n.d.]. The Apprentice Learner Architecture: Closing the Loop Between Learning Theory and Educational Data. In *Proceedings of the 9th International Conference on Educational Data Mining.*
[4] Erik Harpstead Christopher MacLellan Napol Rachatasumrit Kenneth R. Koedinger Daniel Weitekamp, III. [n.d.]. Toward Near Zero-Parameter Prediction Using a Computational Model of Student Learning. In *Proceedings of the 12th International Conference on Educational Data Mining.*
[5] et al. David Silver, Thomas Hubert. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:cs.AI/1712.01816v1
[6] Kenneth R. Koedinger Erik Harpstead, Christopher MacLellan. [n.d.]. Investigating the Solution Space of an Open-Ended Educational Game Using Conceptual Feature Extraction. In *Proceedings of the 6th International Conference on Educational Data Mining.*
[7] Vincent Aleven Erik Harpstead, Brad Myers. [n.d.]. In Search of Learning: Facilitating Data Analysis in Educational Games. In *CHI 2013.*
[8] Christopher MacLellan. 2017. *Computational models of Human Learning: Applications for Tutor Development, Behavior Prediction, and Theory Testing.* Ph.D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.
[9] Bryan Maher Erik Harpstead et al. Michael Christel, Scott Stevens. [n.d.]. RumbleBlocks: Teaching science concepts to young children through a Unity game. In *17th International Conference on Computer Games.*
[10] Kenneth R. Koedinger Rony Patel, Ran Liu. 2016. Evidence Against Teaching Fraction Addition before Fraction Multiplication. *CogSci* (July 2016).