Proceedings of the Seventeenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2021)

Birds in Boots: Learning to Play Angry Birds with Policy-Guided Search

Lucas V. S. Pereira,[1] **Luiz Chaimowicz,** [1] **Levi H. S. Lelis** [2, 3]
[1] Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Brazil
[2] Department of Computing Science, University of Alberta, Canada
[3] Alberta Machine Intelligence Institute (Amii), Canada
lucaspereira@dcc.ufmg.br, chaimo@dcc.ufmg.br, levi.lelis@ualberta.ca

Abstract

In this paper we present Birds in Boots (BiB), a system that uses a sampling-based search algorithm to learn a neural policy for solving Angry Birds levels. Our learning procedure is based on the Bootstrap algorithm, which was previously used to learn heuristic functions for solving classic heuristic search problems. BiB starts its learning procedure with a policy given by a randomly initialized neural network. This initial policy is used to guide the search algorithm on a set of procedurally generated Angry Birds levels. The levels the search algorithm is able to solve are used to improve the neural policy. We repeat this procedure a number of times, until solving all levels or reaching a time limit. We perform several experiments with different instances of our method and show that it can solve more levels than other approaches, including learning-based and rule-based methods.

Introduction

Physics-based simulation games, such as the world-acclaimed Angry Birds, bring a series of challenges to Artificial Intelligence methods (Renz et al. 2016). These games typically have large state and action spaces, and agents have to decide which action to take based on partial and imperfect information. Thus, in recent years, different frameworks and competitions have been proposed to foster the development of intelligent agents for these games.

One of these competitions is the Angry Birds AI Competition (AIBirds) (Renz et al. 2015). The objective is to build AI agents who can play new game levels as well or better than human players. Agents do not have access to the game's internal state and have to act based on the information from the screen, in the same way as human players. Moreover, the game has no efficient forward model, and the outcome of each action is only known after simulating it. So far, agents based on machine learning have had limited success in this competition (Renz et al. 2019). The very large action and state spaces and the lack of an efficient forward model may explain the lack of success of learning-based agents.

In this paper, we present *Birds in Boots* (BiB), an agent that uses the Bootstrap system (Arfaee, Zilles, and Holte 2011) in conjunction with a sampling-based search algorithm to learn a neural policy for solving Angry Birds levels. We show that the search algorithm BiB employs is suitable to domains with computationally expensive simulations such as Angry Birds. BiB incrementally learns to solve more difficult levels by solving simpler ones. BiB starts with a policy given by a randomly initialized neural network and uses this policy to guide its search on a set of procedurally generated Angry Birds levels. The solved levels are used to train and improve the neural policy, and this process is repeated until solving all levels or reaching a time limit.

Empirical results show that the policy BiB learns generalizes well not only to unseen levels sampled from the same distribution used in training, but also to levels sampled from a similar but different distribution. Namely, BiB performs well on levels from the AIBirds competition, solving more levels than all systems tested, which included a rule-based system that placed second in the latest competition.

Related Work

Several Angry Birds agents have been created in the last five years to participate in the AIBirds competition. These agents used different approaches, including state-space search, machine learning, or manually constructed rule-based systems.

One of the most prominent Angry Birds search-based agents is AKBABA (Schiffer, Jourenko, and Lakemeyer 2016), which uses search and simulation to find appropriate parameters for launching birds. Launching parameters are composed of a target within the scene, the bird's initial velocity, the bird's initial angle, and the point on the bird's trajectory at which its ability is triggered in case it has one. AKBABA relies on a heuristic that uses the estimated score and the estimated number of destroyed pigs to traverse this potentially infinite search space. The heuristic is computed by simulating launching a bird given a particular set of parameters. Unlike AKBABA, which does not use machine learning and searches on a very large parameter space, BiB searches on the game's action space and it learns a policy by solving a set of procedurally generated levels.

Others have explored the use of reinforcement learning (RL) for solving Angry Birds levels. For example, in the 2017 competition (Stephenson et al. 2018), there were two RL agents: *Vale Fina 007* and *AngryBNU*. The Vale Fina 007 agent uses Q-learning to identify suitable shots for unknown levels. To describe the current state of a level, a list of objects is used that contains information about every object

within it. Q-learning is used to compute a policy that associates the state description with successful shots. The AngryBNU (Yuan et al. 2017) agent uses deep reinforcement learning to predict optimal shot angles and tap times based on the features within a level. These features include the bird type, the distance to the target points, and a 128x128 pixel matrix around each target. A more recent agent that also uses deep reinforcement learning is *DQ-Birds* (Nikonova and Gemrot 2019). DQ-Birds use a *Double Dueling Deep Q-Network* (Mnih et al. 2015) architecture which is trained using more than $95,000$ images from 21 levels of the classic Angry Birds game. We compare the performance of our proposed methodology with DQ-Birds.

Thus far, agents based on machine learning have not obtained a good performance in the Angry Birds Competitions (Renz et al. 2019). More traditional reasoning agents tailored for the game have successfully obtained higher scores, which is the primary metric used in the competition. One example is the *SimbaDD* agent, which finished in second place in the 2019 competition, defeating strong agents such as the champion of the 2017 and 2018 competitions. SimbaDD tries to emulate the human decision-making process by building a three-layer structure with reactive, predictive, and reasoning processes (Starke et al. 2019). We use SimbaDD as one of the baseline agents in our experiments.

AlphaGo (Silver et al. 2016) and its successor AlphaGo Zero (Silver et al. 2018) also learn to play games from experience. The main difference between BiB and the Alpha agents is the environment in which they operate. BiB deals with a single-agent problem while the Alpha agents solve two-player zero-sum games. Another difference is the search algorithm used. The Alpha agents assume the existence of an efficient forward model of the game and employ variants of Monte Carlo Tree Search (Kocsis and Szepesvári 2006). BiB uses a simpler sampling-based search procedure to deal with the lack of an efficient forward model of the game.

BiB uses the Bootstrap system (Arfaee, Zilles, and Holte 2011) to learn a policy, while in its original paper Bootstrap was used to learn a heuristic function Iterative-Deepening A* (IDA*) (Korf 1985). In a recent work the Bootstrap system was used to learn both a policy and a heuristic function for an algorithm called Policy-Guided Heuristic Search (PHS) (Orseau and Lelis 2021). Similarly to the Alpha agents, all these works assume the existence of an efficient forward model of the game as they might require a large number of simulations to solve Angry Birds levels. Instead, we use a simpler sample-based search that is able to find solutions with fewer simulations in domains with a bounded search depth such as Angry Birds (Orseau et al. 2018).

There are several other agents that have participated in the Angry Birds competitions. A work that tries to take advantage of this great diversity of agents is presented in (Stephenson and Renz 2017a). The authors propose the construction of a hyper-agent that selects from a portfolio of agents whichever agent it believes is best at solving any given level. Stephenson and Renz used a collection of eight agents that participated in the 2016 AIBirds competition and relied on an assortment of score prediction models to rank the sub-agents available in its portfolio based on a given level's features. In general, the proposed hyper-agent presented a better performance in comparison to the individual sub-agents.

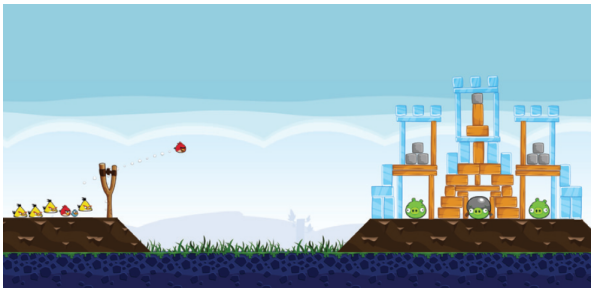

Figure 1: A level of Angry Birds with its main elements: blocks and a slingshot that is used to throw birds. The type of block (ice, wood, or stone) determines the resistance of the block (Ferreira 2016).

Problem Formulation

An Angry Birds level is defined by an ordered set of birds B, pigs P, and a set of objects. Figure 1 shows an example of an Angry Birds level. The pigs and objects are distributed on a 2-dimensional environment M where physics rules such as friction and gravity apply; the birds are placed on the left-hand side of a slingshot, which the agent uses to throw the birds. The agent throws birds in the order given in B. The location of birds, pigs, and objects in M determine a game state. The agent sequentially chooses the force and angle in which each bird is thrown with the goal of hitting the pigs, which are eliminated from the game once hit. A level is solved if all pigs are eliminated from the game. In addition to force and angle, we also consider the birds' special abilities. All but one type of bird have a special ability, which can be activated during the birds trajectory, after it leaves the slingshot and before it hits an object or pig. The agent must decide when the special ability is activated. An action is either a tuple defining the force and angle or a scalar determining the time that a special ability is activated during the bird's trajectory. A solution path of a level is a sequence of actions $a_1, a_2, \cdots, a_n$ that eliminates all pigs from M.

Learning Agent

BiB trains an agent for playing Angry Birds in a self-supervised fashion. The agent receives a set of procedurally generated I levels of Angry Birds and employs a policy-guided sampling search known as `sample_traj` (Orseau et al. 2018) and a neural network to guide the search. We use the Bootstrap algorithm (Arfaee, Zilles, and Holte 2011) to generate training data to train the neural policy that `sample_traj` employs.

Learning with Bootstrap

Bootstrap starts with a randomly initialized neural network and `sample_traj` tries to solve each instance in I with the computational budget of n throws (a simulation of a throw of a bird and the use of the bird's special ability, if it has any). The solution paths of the levels `sample_traj` solves

form a data set that is used to train the policy (a solution path contains the action one needs to take in each visited state to solve the level). The generated training data is organized in mini-batches that are used to update the neural policy with one gradient descent step. The Bootstrap process is repeated with the updated (and hopefully stronger) policy on the set of instances that `sample_traj` could not solve in previous iterations. Bootstrap stops when it reaches a time limit or it is able to solve all instances in I. BiB accumulates the training data across iterations, i.e., in iteration i the model is updated with training data from iterations j for $j \leq i$.

In the original formulation of Bootstrap one increases the computational budget if the search algorithm is unable to solve any instances in a given iteration. We did not notice the need of increasing the computational budget n in our experiments as the updated policy always allowed `sample_traj` to solve levels that were not solved in previous iterations.

Action Space Formulation

The action space of Angry Birds is continuous and thus infinite. BiB uses the maximum force in all throws and selects the angle value of the throw. BiB discretizes the angle values and learns a probability distribution over the discretized values. We evaluate two angle discretization schema in BiB. In the first scheme, dubbed angles-model (AM), the agent can choose angles from -6 to 78 degrees, with increments of 0.3, which results in 281 possible values of angle. The second scheme, dubbed functions-model (FM), uses a set of 64 simple domain-specific functions for choosing angles. Each function receives a state of the game and returns an angle. For example, given a state of the game the function returns an angle that hits the leftmost pig on M. We use the formula

$$\arctan\left(\frac{v_0^2 \pm \sqrt{v_0^4 - g(g\Delta x^2 + 2\Delta y v_0^2}}{g\Delta x}\right), \quad (1)$$

to determine the angle needed to reach a specific position on M. Here, v_0 is the initial velocity, $\Delta x = x_0 - x_s$, $\Delta y = y_0 - y_s$, where (x_0, y_0) and (x_s, y_s) are the initial and the target coordinates of the bird. Note that the formula is a second-order polynomial and thus returns two valid angles for reaching the same position. BiB uses the larger of the angles as it results on a trajectory that crosses a usually less populated area of M. The functions we employ in the functions-model do not account for other objects in M; it computes an angle that would allow the bird to reach a given position in M should the environment was empty. The functions-model is inspired on the action abstractions used in real-time strategy games (Tavares et al. 2018; Churchill and Buro 2013; Lelis 2020). The set of functions used in our experiments can be accessed in our codebase.[1]

The functions-model allows the agent to learn over a smaller and hopefully helpful set of actions; the angles-model is more fine grained as the agent has more actions to choose from, which can allow it to learn stronger policies at the cost of a possibly more difficult learning process.

For both angles and functions models, we discretize the time intervals in which the special ability can be activated. We consider values from 0.5 to 2.5 seconds and use increments of 0.05 for values between 0.5 to 1.20 seconds and increments of 0.1 for values between 1.20 to 2.50 seconds. We use finer intervals in the beginning of the trajectory because that is when most of the special abilities are used. BiB also considers the option of not activating the special ability, totalling 29 actions.

[1] https://github.com/lucasvictorsp/science-birds.

Neural Model

With the goal of speeding up training and inference, BiB uses a simplified version of the game's screenshot image as input to the neural model. BiB uses only the part of the image that can contain objects; specifically, it considers a box of size 397×264 on the right-hand side of the slingshot. We further reduce the size of the image by using a coarser version of the reduced image. The original $840 \times 480 \times 3$ image is reduced to an image of size $159 \times 106 \times 3$, where the last dimension of the image represents the RGB channels. The reduction is performed by traversing the matrix representing the image and skipping a few values. We start by selecting the RGB values at index $(0, 0)$ and skipping values as follows. If the value (i, j) is selected to be used in the coarser image, then we skip $(i+1, j)$, select $(i+2, j)$ and skip both $(i+3, j)$ and $(i+4, j)$; the process is then repeated by selecting $(i + 5, j)$. The same process described for rows is also applied to columns. Instead of providing the RGB channels, we use a one-hot matrix of size 159×106 for each of the following objects of the game: ice, wood and stone blocks, indestructible platforms, explosive objects, and pigs, resulting in a tensor of zeros and ones with dimensions $159 \times 106 \times 6$.

Figure 2 shows the architecture of the neural network used in our experiments. The $159 \times 106 \times 6$ tensor goes through three convolutional layers and a fully connected layer before producing a probability distribution of the discretized angles (see `output 1`) with a Softmax layer of size 281 and 32 for angles-model and functions-model, respectively. Both the probability distribution over angles and the output of the last convolutional layer is provided as input to a fully connected layer for computing the probability distribution over the discrete time steps of the special ability. We provide `output 1` as input to the last fully connected layer because deciding when to use a special ability depends on the angle in which a bird is thrown.

The ordered set B of birds also provides valuable information to the agent, as it needs to plan the current throw based on how many birds it has left. BiB uses a 7×5 matrix with zeros and ones (see matrix "Birds" in Figure 2) for encoding information of the next 7 birds in B. Each column of the matrix represents a type of a bird and each row represents the position of a bird in the ordered set B. If the position i, j is one, then the i-th bird in B is of type j. The "Birds" matrix is flattened and provided as input to the two dense layers of the model. Note that BiB is able to plan for levels with more than 7 birds; it only has to account for the next 7 birds when deciding its next throw.

We use the sum of the cross entropy loss of `output 1` and `output 2` as the training loss for BiB's model. We place a stop gradient on the values of `output 1` that are provided as input to the last dense layer of the model. This is

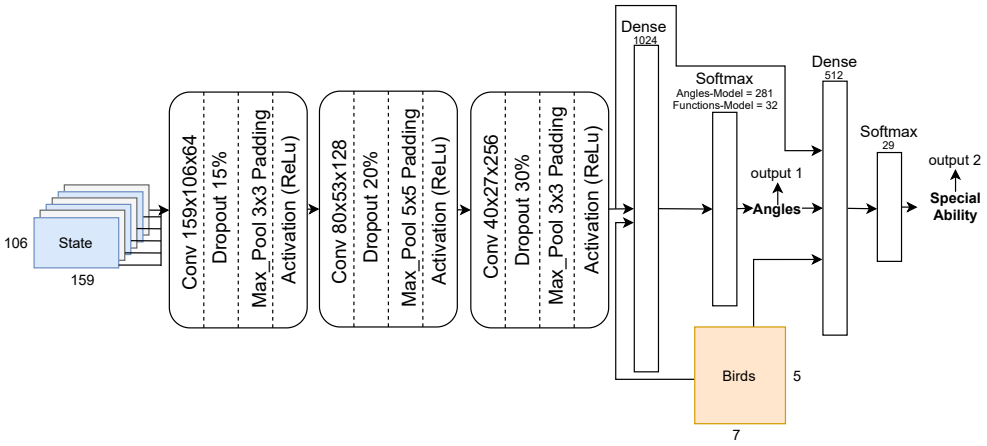

Figure 2: BiB's neural architecture. The probability distribution over angles is provided as input to the last fully connected layer, which computes the probability distribution over the discretized time steps defining when the bird's special ability is activated.

to prevent gradient descent changing the angle decisions of the model while trying to attempt to correct for the use of the birds' special abilities. The cross entropy loss of `output 1` and `output 2` will serve to adjust the weights of the shared convolutional layers and each will serve to adjust the weights of its Softmax and dense layers.

All models used in our BiB experiments are trained with Adam with learning rate 0.01 and, unless stated otherwise, mini-batches of size 64.

Sampling Search Algorithm

Algorithm 1 presents `sample_traj`, which receives an initial state s_0 of the game, a budget n on the number of action executions the algorithm can perform, and a policy π; `sample_traj` either returns a sequence of actions that solves the level represented by s_0 or it returns failure. `sample_traj` sequentially samples a tuple of actions $a = (g, t)$, where g is angle and t a time step, until all birds have been thrown and their special abilities have been used (if applicable) — see inner loop of Algorithm 1). If a solution is found, the sequence of actions used to achieve it is returned (see lines 7 and 8). The process is restarted (line 2) until `sample_traj` finds a solution or it uses all its computational budget and it returns failure (lines 9 and 10).

Algorithm 1 `sample_traj`

Require: initial state s_0, budget of n throws, policy π
Ensure: sequence of actions

```
1:  while True do
2:      s ← s_0, i ← 0
3:      while has not thrown all birds do
4:          sample a tuple of actions a = (g, t) from π(s)
5:          s' is the state obtained after applying a to s
6:          i ← i + 1
7:          if s' is a solution then
8:              return sequence of actions that led to s'
9:          if i equals n then
10:             return failure
```

We also implemented and evaluated LevinTS, which is a more systematic policy-guided search algorithm that builds a tree with action sequences that are promising according to the policy (Orseau et al. 2018). Since the number of throws allowed in Angry Birds is small (simulating actions in the game engine is computationally expensive), LevinTS often fails to encounter a solution because it is unable to visit states in which all birds have been thrown before exhausting its computational budget. `sample_traj`'s more aggressive sampling approach allows one to find solutions even with the limited computational budget imposed by the game.

Empirical Methodology

In this section we detail the empirical methodology used in our experiments with BiB. We start with the generation of the set of instances I used in the Bootstrap procedure.

Problem Instances

We use a modified version of the award-winning procedural generation system of Angry Birds levels, IratusAves (Stephenson and Renz 2017b), to generate the set of instances I used in our experiments. IratusAves is guaranteed to generate stable and solvable Angry Birds levels. The problem with the levels IratusAves generates is that they tend to be easy to solve, as we show below. Ideally, the set I will contain levels of varied difficulty, so that the easy levels allow BiB to learn an initial policy that will allow it to learn policies able to solve the harder levels. If the set of instances I contains only easy instances, then BiB is unlikely to learn strong policies that will generalize to unseen instances, such as those used in the Angry Birds competitions.

We modify IratusAves with the goal of generating levels of varied difficulty. Whenever a level is generated, with 30% of chance, we remove a random bird from the level. Levels with one bird removed might become unsolvable, but this change allows us to generate harder levels that BiB can solve and learn from. We used this modified version of IratusAves to generate 280,000 levels. A simple agent, which we refer

to as the Naïve Agent (NA), is able to solve 264,148 of these instances with a computational budget of 75 action executions. NA chooses its actions with the goal of hitting a random pig in M. NA uses the maximum force and Equation 1 to compute the angle that will hit the randomly selected pig; if the bird has a special ability, NA randomly chooses the time step in which the special ability is activated.

Since NA has generated labeled data for 264,148 levels (i.e., it has encountered solution paths to all these levels), we sample 17,438 levels without replacement from the solved levels to warm start the neural policy. We perform one epoch of training with this training data in a randomly initialized neural network with mini-batches of size 32. The resulting pre-trained neural policy is provided as input to the Bootstrap procedure we describe above. The set I is formed by sampling without replacement 5,000 levels out of the 15,852 that NA could not solve with 75 throws.

Searching with a Trained Model

We use a depth-first search (DFS) with the trained models in our test experiments (see "test and competition evaluations" below). In our DFS, the actions are ordered according to the policy's probabilities: the search selects the actions with highest probability before selecting actions with lower probability. Initially, the algorithm will visit through the sampling related to the actions with highest probability at each state of the game. If the search does not find a solution, then we perform chronological backtracking and try actions with lower probability. Similarly to `sample_traj`, the search stops when it either finds a solution or when it exhausts its computational budget of executions. We use DFS with a trained model instead of `sample_traj` because DFS is greedy with respect to the learned policy, which is better suited for testing the system. `sample_traj` samples an action according to the probability distribution given by the neural model, which could result in suboptimal choices even for a strong policy.

Experiments Performed

We perform three sets of experiments. We have implemented all algorithms and baselines of our experiments in the Science Birds platform (Ferreira and Toledo 2014).

Training Evaluation In the first experiment, we verify how fast different systems learn a strong policy by measuring how long it takes to solve the training instances. We compare the BiB models with the two discretization schema, angles-model (AM) and functions-model (FM), with DQ-Birds, which is an implementation of DQN for Angry Birds (Nikonova and Gemrot 2019). We also provide DQ-Birds with a model that is also pre-trained on the same 17,438 levels used to pre-train the BiB models.

We evaluate several versions of BiB. Two versions that are trained with the Bootstrap procedure and `sample_traj`, which we denote as BiB(AM) and BiB(FM). Two versions that use a random walk planner (RW) on the action space induced by either AM or FM, which are denoted as BiB(AM, RW) and BiB(FM, RW). RW selects an action uniformly at random at a given state. We also evaluate versions of BiB with `sample_traj` that do not receive a pre-trained model, denoted BiB(AM, NPT) and BiB(AM, NPT), where NPT stands for "no pre-training". We evaluate these variants to understand how much learning and the pre-training affect the results. We call this experiment the "training evaluation."

Test Evaluation In the second experiment we evaluate the trained models on a set of held-out levels, which are also generated with IratusAves. The test set has 1,008 levels out of which NA cannot solve 252 with 85 action executions; NA solves the remaining levels with at most 85 throws.

We compare the same BiB models used in the training evaluation and the DQBird agent. However, instead of using `sample_traj`, we employ DFS with the policies learned in the training experiment. We use DFS to allow for a direct comparison of the quality of the learned policies as DFS greedily selects the actions according to the policy probabilities. We compare the agents in two settings: with a single throw, one for each bird, and with 40 throws in total for both DFS and RW. In both settings the evaluated methods use a smaller computational budget than the 85 action executions NA used in our procedure to select the test instances. Thus, even if NA is able to solve some of the levels in the test set, the levels can still be challenging for BiB and DQ-Birds as they need to find a solution with many fewer throws. We call this experiment the "test evaluation."

Competition Evaluation Finally, the last experiment compares the BiB models with DQ-Birds and simbaDD on 20 of the 24 levels used in the 2017 competition. We did not use 4 of the levels because they were unstable in the Science Birds platform. The simbaDD is a domain-specific rule-based agent that placed second in the latest AIBirds competition. We use simbaDD because it performed well in the competition and because its code was open source and the system was easy to run. Here, in addition to DFS and RW, we also evaluate the BiB models with `sample_traj`. The levels used in the competition can be very different from the levels the models were trained on since they come from a different distribution (i.e., they were not generated by IratusAves). We believe that the sampling-based nature of `sample_traj` can be helpful in solving levels that come from a distribution different than the one used in training.

Following a similar approach to the Angry Birds competition, in this experiment, each method was allowed 3 minutes per level, for a total of 60 minutes for the experiment. We call this experiment the "competition evaluation."

Empirical Results

In this section we present the results of the three experiments: training, test, and competition evaluations. The training evaluation was run on a machine with 2.30 GHz CPUs, 256 RAM, and a NVidia GeForce GTX Titan X; the test and competition evaluations were run on a machine with 4.00 GHz CPUs, 16 GB RAM, and a NVidia GeForce GTX 1060. Table 1 contains the acronyms of the methods used in our experiments.

Symbol	Meaning
BiB(AM)	Trained BiB agent that uses the angles-model for actions.
BiB(AM, NPT)	BiB(AM) without the pre-training step.
BiB(AM, RW)	BiB(AM) using uniform distribution.
BiB(FM)	Trained BiB agent that uses the functions-model for actions.
BiB(FM, NPT)	BiB(FM) without the pre-training step.
BiB(FM, RW)	BiB(FM) using uniform distribution.
`sample_traj`	Sampling algorithm that uses a policy.
DFS	Depth-first search that uses a policy.
DQBirds	DQN Agent for Angry Birds (Nikonova and Gemrot 2019).
NA	Naïve Agent
SimbaDD	Baseline from the 2019 Competition (Starke et al. 2019).

Table 1: Methods used in our evaluation.

Training Evaluation

Figure 3 shows the results for the training evaluation. The x-axis shows the number of throws executed in the game by each of the approaches during training; the y-axis shows the number of training levels solved. The curves for BiB(AM), BiB(FM), and DQ-Birds are shifted to the right to account for the number of action executions NA performed while labelling the data used to pre-train the model.

The methods that learn a policy solve more instances than the methods that simply use `sample_traj` with a uniform policy. The RW methods plateau at around 1,000 solved levels. The pre-trained model substantially improves the performance of BiB with AM, as BiB(AM) solves almost 1,000 more levels than BiB(AM, NPT). The improvement pre-training provides BiB with FM is much less pronounced. The action space of the AM scheme is larger than the action space of the FM scheme and simple strategies such as "aim at a pig" are harder to learn from scratch with the AM scheme. The pre-trained model already assigns high probability to such simple strategies, which are refined with further training with the AM scheme. BiB with FM is also able to play such strategies in the beginning of training due to the pre-trained model. However, BiB with FM is unable to learn stronger policies. We conjecture that this happens because FM constrains the action space and the good policies BiB with AM learns are not available in FM space.

DQ-Birds is competitive with BiB(FM), but it solves approximately 1,000 fewer levels than BiB(AM) at the end of training. We conjecture that the search BiB performs helps the agent be more effective at collecting training data than DQ-Birds. That is, the policy `sample_traj` employs might be unable to quickly guide the agent to a solution of a given level, but `sample_traj` can compensate for the weak policy with search. The use of abilities might also explain the difference between BiB and DQ-Birds. While BiB explicitly learns a policy for deciding when to use the abilities, DQ-Birds uses a hard-coded heuristic to use them.

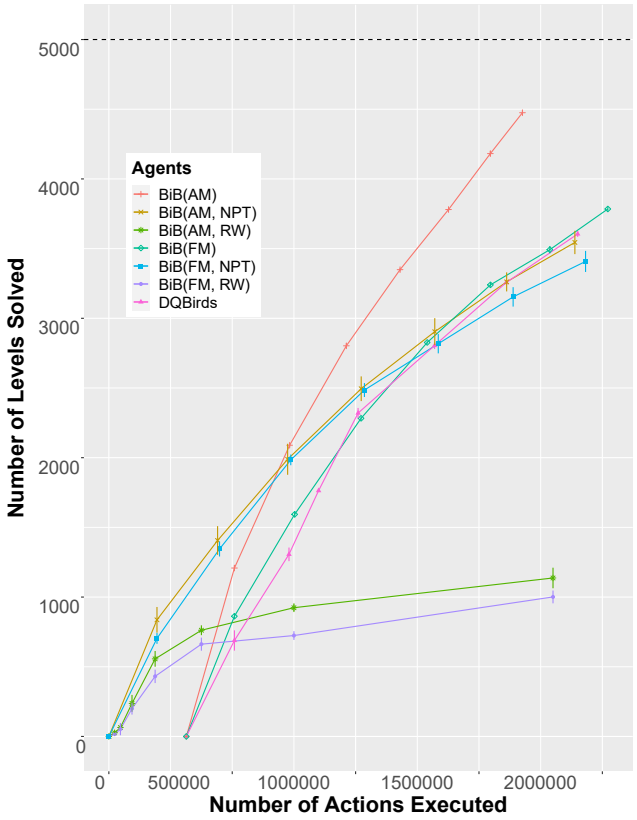

Figure 3: Number of training levels solved by the number of throws executed for different learning schema. The curves represent an average of 3 independent runs and the error bars denote standard deviation across runs. The dashed line at the top mark the total size of the training set. The curves for BiB(AM), BiB(FM), and DQ-Birds are shifted to the right to account for the number of throws NA performed while labelling the data used to pre-train the model.

Test Evaluation

Table 2 presents the results of the test evaluation. The average number of levels solved is computed for different runs of the system trained with a different models (the three models trained in the first experiment). We highlight the best results in bold. Naturally, all methods perform better if allowed a larger computational budget (see rows for $n = 1$ and $n = 40$) and the search algorithms using a trained model, DFS and DFS(NPT), perform better than the RW method for a fixed budget. The BiB(AM) model with DFS and $n = 1$ performs comparably to other RW models with $n = 40$. This result provides an approximated measure of how much one gains in terms of computational budget when training a policy with BiB.

Similarly to what was observed in the training evaluation, the use of pre-trained model increases the performance of both AM and FM, with the difference being more pronounced for AM. Finally, both BiB(FM) and BiB(AM) solve

n	DQ-Birds		BIB(AM)			BiB(FM)			NA
	RW	DFS	RW	DFS(NPT)	DFS	RW	DFS(NPT)	DFS	
1	20.5 ± 7.8	175.0 ± 9.9	48.3 ± 25.1	243.3 ± 7.8	$\mathbf{379.0 \pm 3.6}$	25.3 ± 11.1	198.0 ± 6.2	287.3 ± 7.6	116.3 ± 11.5
40	371.0 ± 49.5	596.0 ± 14.1	395.2 ± 79.3	649.7 ± 10.7	$\mathbf{729.7 \pm 12.0}$	406.0 ± 58.2	649.3 ± 13.0	689.7 ± 7.8	529.0 ± 40.8

Table 2: The table shows the average number of levels solved and the standard deviation of different agents with computational budget n of 1 and 40 throws. The average is computed by running DFS with the 3 different models trained in the training evaluation experiment. We also run RW 3 times and report its average results as well as its standard deviation.

DQBirds		BiB						SimbaDD	NA
RW	DFS	AM `sample_traj`	AM DFS	AM RW	FM `sample_traj`	FM DFS	FM RW		
6.67 ± 3.10	14	$\mathbf{19.33 \pm 1.15}$	15	7.33 ± 3.51	14.33 ± 0.58	12	4.67 ± 4.04	15.67 ± 0.58	9.00 ± 1.00

Table 3: Average number of levels solved and standard deviation on 20 levels from the 2017 AIBirds competition.

more levels than DQ-Birds, with BiB(AM) with the pre-trained model and DFS solving more levels than all the other methods for a fixed computational budget.

Competition Evaluation

In this section we evaluate BiB's generalization to levels sampled from a distribution different from the one used in training. Table 3 presents the results on 20 levels from the 2017 AIBirds competition. We highlight the best result in bold. In addition to RW and DFS, we also evaluate the BiB `sample_traj`. For both DQ-Birds and BiB we selected the best performing model in terms of levels solved during training, out of the three trained. We run `sample_traj` and RW three times with the selected model and present the average number of levels solved and the standard deviation of the results. Since DFS is deterministic, we run the algorithm only once with the selected model.

The RW models solved a small number of levels on average; RW with the AM action space performed best amongst the RW methods with 7.33 levels solved. The DFS with DQ-Birds and BiB models solved fewer levels than SimbaDD. The BiB models performed better with `sample_traj` than with DFS, especially AM. BiB(AM) with `sample_traj` solved almost all levels on average, which is far better than all other methods tested. Both the DFS and `sample_traj` results show that the policies learned with DQ-Birds and BiB can generalize to levels sampled from a distribution different than the one used in training. The results of BiB(AM) and `sample_traj` show that the policies BiB learned can outperform successful approaches such as SimbaDD.

Conclusions

In this paper we presented Birds in Boots (BiB), a system that combines search and learning to train a policy for playing Angry Birds. BiB uses the Bootstrap system to iteratively solve levels from a training set. BiB uses the data collected from the solved levels to further improve its policy. The neural model BiB employs predicts both the angle in which a bird is to be thrown and the time in which the bird's special ability is to be activated. BiB uses a sampling-based policy-guided search algorithm that is suitable to the settings where it is computationally expensive to simulate actions. Empirical results showed that the policy BiB learns generalizes well not only to unseen levels sampled from the same distribution used in training, but also to levels sampled from a similar but different distribution. Namely, BiB performed well on levels from the AIBirds competition, solving more levels than all systems tested, including a rule-based system that placed in second in the latest competition.

Enhancing PCG Systems As future work, BiB might be used to enhance procedural content generation (PCG) systems. Once BiB is able to train a strong neural policy, one can then use it to check for the solvability of levels a PCG system generates. We can then try to generate harder and hopefully more interesting levels by removing birds from the levels, similarly to how we generated the levels for our experiments. This is an interesting direction for future works.

Acknowledgements

We would like to thank Lucas N. Ferreira for helping with the Science Birds platform. This research was funded by Brazil's CAPES and by Canada's CIFAR AI Chairs program.

References

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17): 2075–2098.

Churchill, D.; and Buro, M. 2013. Portfolio greedy search and simulation for large-scale combat in StarCraft. In *Proceedings of the Conference on Computational Intelligence in Games*, 1–8. IEEE.

Ferreira, L.; and Toledo, C. 2014. A Search-based Approach for Generating Angry Birds Levels. In *Proceedings of the 9th IEEE International Conference on Computational Intelligence in Games*, CIG'14.

Ferreira, L. N. 2016. *Uma abordagem evolutiva para geração procedural de níveis em jogos de quebra-cabeças baseados em física*. Master's thesis, Universidade de São Paulo.

Kocsis, L.; and Szepesvári, C. 2006. Bandit Based Monte-Carlo Planning. In *ECML*, 282–293. Springer Berlin Heidelberg.

Korf, R. E. 1985. Depth-first iterative-deepening. *Artificial Intelligence* 27(1): 97 – 109.

Lelis, L. H. S. 2020. Planning Algorithms for Zero-Sum Games with Exponential Action Spaces: A Unifying Perspective. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 4892–4898. International Joint Conferences on Artificial Intelligence Organization.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Nikonova, E.; and Gemrot, J. 2019. Deep Q-Network for Angry Birds. *arXiv preprint arXiv:1910.01806* .

Orseau, L.; Lelis, L.; Lattimore, T.; and Weber, T. 2018. Single-agent policy tree search with guarantees. In *Advances in Neural Information Processing Systems*, 3201–3211.

Orseau, L.; and Lelis, L. H. S. 2021. Policy-Guided Heuristic Search with Guarantees. In *AAAI Conference on Artificial Intelligence*, 12382–12390. AAAI Press. URL https://ojs.aaai.org/index.php/AAAI/article/view/17469.

Renz, J.; Ge, X.; Gould, S.; and Zhang, P. 2015. The Angry Birds AI Competition. *AI Magazine* 36(2): 85–87.

Renz, J.; Ge, X.; Stephenson, M. J. B.; and Zhang, P. 2019. AI meets Angry Birds. *Nature Machine Intelligence* 1(7): 328–328. ISSN 2522-5839. doi:10.1038/s42256-019-0072-x. URL https://doi.org/10.1038/s42256-019-0072-x.

Renz, J.; Miikkulainen, R.; Sturtevant, N. R.; and Winands, M. H. 2016. Guest Editorial: Physics-Based Simulation Games. *IEEE Transactions on Computational Intelligence and AI in Games* 8(2): 101–103.

Schiffer, S.; Jourenko, M.; and Lakemeyer, G. 2016. Akbaba: An Agent for the Angry Birds AI Challenge Based on Search and Simulation. *IEEE Transactions on Computational Intelligence and AI in Games* 8(2): 116–127. doi:10.1109/TCIAIG.2015.2478703.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529(7587): 484–489.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362(6419): 1140–1144.

Starke, C. R.; Miemietz, T.; Hennig, T. R.; Thies, L.; and Schweizer, L. 2019. SimbaDD (Simulation Based Agent Dresden). http://www.aibirds.org/2019/SimbaDD.pdf. Accessed: 2021-08-07.

Stephenson, M.; and Renz, J. 2017a. Creating a Hyper-Agent for Solving Angry Birds Levels. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'17, 234–240. URL https://aaai.org/ocs/index.php/AIIDE/AIIDE17/paper/view/15828.

Stephenson, M.; and Renz, J. 2017b. Generating varied, stable and solvable levels for Angry Birds style physics games. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, 288–295. IEEE.

Stephenson, M.; Renz, J.; Ge, X.; and Zhang, P. 2018. The 2017 AIBIRDS Competition. *CoRR* abs/1803.05156: 1–11. URL http://arxiv.org/abs/1803.05156.

Tavares, A. R.; Anbalagan, S.; Marcolino, L. S.; and Chaimowicz, L. 2018. Algorithms or Actions? A Study in Large-Scale Reinforcement Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2717–2723.

Yuan, Y.; Chen, Z.; Wu, P.; and Chang, L. 2017. Enhancing Deep Reinforcement Learning Agent for Angry Birds. http://aibirds.org/2017/aibirds_BNU.pdf. Accessed: 2021-08-07.