SME-Project-M
Project Report

at

Otto-Friedrich-Universität Bamberg
Junioprofessor für Angewandte Informatik, insbesondere Smart Environments
Fakultät Wirtschaftsinformatik und Angewandte Informatik (WIAI)

Harshit Gupta, B. Tech
harshit-kumar.gupta@stud.uni-bamberg.de
Matrikelnummer: 1853351

1 Abstract

This report describes the work of the Meta-Team in the SME-Project-M in the Summer Semester 2018, supervised by Prof. Dierich Wolter and Prof. Ute Schmid. The aim of this project was to improve the BamBird 2017 agent by adding heuristic intelligence and to win the “Angry Birds AI Competition”.

It begins by describing the Angry Birds AI competition and provides a brief explanation of heuristics in artificial intelligence before proceeding to the BamBird 2018 no_eclipse Agent.

2 Introduction

2.1 Angry Birds

Angry Birds is a popular physics-based simulation game [4]. In the game, the player controls a flock of multi-colored birds that are attempting to retrieve their eggs, which have been stolen by a group of hungry pigs [5]. In each stage of the gameplay, enemy pigs are sheltered by structures made of various materials such as wood, glass, ice and stone resembling children's toy building blocks. The objective of the game is to eliminate all the pigs on the level. Using a slingshot, players launch a limited set of birds with the goal of either striking the enemy pigs directly or damaging their surrounding structures, causing the blocks to collapse and squash the pigs.

2.2 Challenge for AI

The challenge is to then build an AI player that can play new game levels as good as or better than the best human players. This is not an easy task although it would appear so when comparing to hard games like chess because of a number of reasons. Assuming all the parameters of the game world are known, we could simulate a number of actions and based on the outcome of the simulation, we could simulate a number of follow-up actions and so on until victory condition is reached. If we select the actions we simulate in an intelligent way, this could lead to a successful solution strategy. However, the main issue in physics-based simulation games is that the outcome of an action is only known once we simulate it, which in turns requires us to know all parameters necessary for simulation. This is very different from games such as Chess where the outcome of each action is known in advance. Accurately predicting or approximating the outcome of actions is one of the main challenge in the game. Combining this with the potentially infinite action space and the possible lack of complete information about all required parameters means that we are in for a big challenge. Humans are very good at predicting consequences of physical actions but this form of reasoning for AI agents remains a difficult task [4].

3 Background

3.1 AI Birds and IJCAI

The International Joint Conference on Artificial Intelligence is a gathering of artificial intelligence researchers and practitioners. The IJCAI 2017 conference was held in Melbourne, Australia and the IJCAI 2018 conference will be held in Stockholm, Sweden.

The Angry Birds AI Competition was help in conjunction with IJCAI 2017 conference and will be held again at the IJCAI 2018 conference. The goal of the competition is to develop an intelligent agent that can play the Angry Birds game as good as or better than human players.

In the AI Birds Competition, we play Angry Birds using the web version of the game. The competition server interfaces with the website using a Chrome browser extension which allows us to take screenshots of the live game and to execute different actions using simulated mouse operations. Participating agents run on a client computer and interact with the server using a fixed communication protocol. This allows agents to request screenshots and to submit actions and other commands which the server then executes on the live game. The participants obtain only the information of sequences of screenshots of the live game, therefore the agents have exactly the same information as a human player[4].

3.2 Heuristic AI

What is Heuristic AI?

Heuristic function is a function that generates alternatives at each branching step in a search algorithm to decide which branch to traverse next [2].

To understand what a heuristic search is and why it is useful, let us consider the example of 8-puzzle problem. The goal of the puzzle is to move the slides into empty spaces starting from the initial configuration to reach the goal configuration.

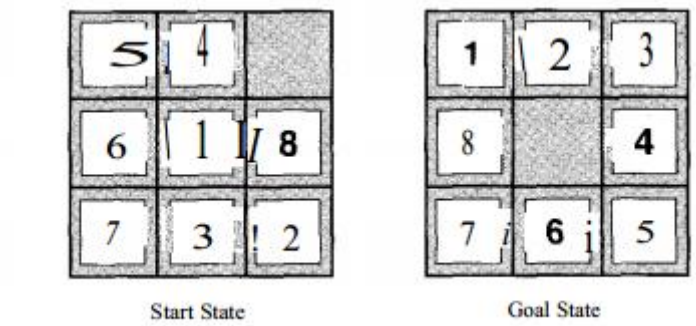

Fig 1 8-Puzzle Example

In most cases the goal state can be achieved in 20 steps, varying depending on the start state. The branching factor is about 3 (when the empty tile is in the middle, there are four possible moves; two

when it's in the corner, and three on the edge). With an exhaustive search to depth 20, this would give us 3^{20} states [3].

This quickly becomes a problem if the game is much more complicated such as Angry Birds. Two factors immediately make it very difficult to even construct all states exhaustively during the game time. First is the unpredictable physics simulation. Because the operability of the physics of the game is unknown, producing all states exhaustive is restricted. We can create states and reproduce those states but only if the sequence of actions (shots taken for example) are kept same. The second factor is the limitation of time. To explore all possible states of all levels in the time given is not possible right now.

Thus we use heuristic functions to reduce this state count drastically. We keep track of states visited and use the information gained from those states to decide which action (or shot) to take next.

4 The BamBird 2018 Agent no_eclipse Meta

4.1 Creation of One Jar File

Before we implemented the change, the BamBird agent was built with the Prolog function files external to the jar.

In this task the Prolog files into the `Java/src` folder so that when the jar file is created the Prolog files are within the jar file. And in the next step, the class `PrologFilesCopier` was created which is responsible for the copying of the Prolog files from the jar to an external directory so that these files can be accessed by the SWI-Prolog.

The `PrologFilesCopier` uses the class loader to request for a resource. The resource is read line by line and written to the external file again line by line.

4.2 Meta

Below we present that Meta algorithm.

Algorithm 1 Meta

```
1: Load the ShotLearner database
2: loop
2:     if GameState != Playing then
3:           choose new Level using the new equation
4:     end if
5:     take a screenshot from the scene and analyse
6:     generate plans
7:     send the plans to the ShotLearner
8:     select the shot returned by ShotLearner to execute
9:     execute the selected shot
10:    add shot to the shot-list of the current level
11:    if GameState == WON or GameState == LOST then
12:          add level to database
13:    end if
14:    add the shot to ShotLearnerDatabase
15:    create a ShotResult object with the results of the shot
16:    save the ShotResult to a log file
17: end loop
```

There are a few changes in the Meta algorithm from the BamBird 2017 agent. In the BamBird 2018 `no_eclipse` agent that we present in this report, two new classes were added. The first is the `ShotLearner` class, described in more detail below, the second `ShotResult`, which is the POJO collecting results of the shot.

In the modified algorithm, the `ShotLearner` database is loaded before entering the game loop. The `ShotLearner` database is a list of `ShotResult` objects stored in a file. Then the game loop starts and if the game state is not playing, then we choose a level, using the new level equation as described in the

section below. The algorithm then forwards the plans that it receives from Prolog to the `ShotLearner` class. The `ShotLearner` class returns one shot using heuristics and probabilities, this shot is then executed next. The results of the shot are stored as a `ShotResult` object into two different files, first is the `ShotLearner` database file and the other is the logs file. The logs file just collects information about the last run of the agent, as opposed the `ShotLearner` database which collects information about `ShotResults` over all runs of the agent.

4.2 Level Selection

There is one restriction when it comes to level selection – time, or in other words, the number of levels that can be played. In practice we found that on average the number of attempts are 1.5 times the number of levels in the competition. In the 2018 `no_eclipse` Agent, we start playing the game from the first level and play each level once linearly. We call this the first round. After the first round, the agent creates a list of levels that it did not manage to clear them. This list of failed level is iterated once, and we call this the second round. In the third round the Agent uses the heuristic function to determine which level to play next.

Below, we present the heuristic function equation that we used to determine the potential gain.

$$P = \frac{(E - B)}{(E * N)}$$

Where,
P is Potential Gain
E is Estimated Maximal Points
B is Best Score
N is the number of times this level has been played.

In Round 2, levels are sorted by their potential gain and the level with the highest potential gain is selected to play next. Potential Gain is defined as the ratio of points that can still be gained for this level. This ratio is obtained by estimating the maximal points that can be achieved for this level. This estimation is achieved is by adding points for all pigs, plus some points for structure such as wood or ice, and assuming that we solve the level with just using one bird, the points of the remaining birds. Then we retrieve the best score we had for this level in Round 1 and then find the potential gain for this level by subtracting this best score from the estimated maximal points and dividing the result by the estimated maximal points.

Potential Gain is indirectly proportional to the number of levels played. This is significant because we want to agent to not repeat a level that has been found to be unsolvable in the previous attempts.

4.3 Level Selection Evaluation

Number of Levels:	19			
MaxAllowedAttempts:	29			
Attempt	Level	EstimatedMaximalPoints	Score	Status
1	1	66226	10865	LOST
2	2	17255	662	LOST
3	3	27988	18522	WON
4	4	76735	50426	WON
5	5	14936	2125	LOST
6	6	45815	5270	LOST
7	7	69067	28510	WON
8	8	8149	8066	WON
9	9	56308	9491	LOST
10	10	3925	1100	WON
11	11	82866	78314	WON
12	12	22126	1613	LOST
13	13	97424	93837	WON
14	14	32486	26414	WON
15	15	80048	46450	WON
16	16	68069	21172	WON
17	17	57907	11368	LOST
18	18	43115	26074	WON
19	19	8212	3153	WON
20	1	66226	55459	WON
21	2	17255	3748	LOST
22	5	14936	5452	WON
23	6	45815	2061	LOST
24	9	56308	10343	LOST
25	12	22126	478	LOST
26	17	57907	11166	LOST
27	17	57907	49069	WON
28	17	57907	25127	WON
29	17	57907	22870	WON

Fig 2 Level Selection Evaluation

In an evaluation run, we randomized the number of levels, the estimated maximal points and the score. The status of winning the level was dependent of scoring more than one fifth of the estimated maximal points. The maximum number of attempts are calculated as 1.5 times the number of levels.

We note that the Round 0 starts from Attempt 0 and finishes on attempt 19. Round 1 starts from attempt 20 and finishes on attempt 26. From attempt 27 to attempt 29 is the Round 2. Level 17 is won on attempt 27 but it is still the level with the highest potential gain and thus the agent tries to play the level for two more attempts.

4.3 Shot Selection

4.3.1 Shot Learner

Shot Learner is the class where the learning happens. After reading the database, the list of Shot Results is passed to the Shot Learner. This list is the background data for making probabilistic decisions. The main function of the Shot Learner class is to give feedback on whether a shot is good to execute or not. In the absence of data, the default answer is always true, meaning try out the shot.

For the following strategies the heuristics and probabilities are used: `targetPig`, `domino`, `blackBird`, `collapseStructure`, `tnt`, `heavyObject` and `whiteBird`. And for the following strategies the default value of true is returned: `defrost`, `woodworm`, `bunker` and `roof`. The default value of true was used for the last set of strategies because these strategies were encountered rarely and hence the number of examples were not sufficient enough to make a sound learning judgment.

Below, we present the algorithm of Shot Learner.

Algorithm 2 Shot Leaner

```
1: Load the strategy from the Target argument
2: if strategy == strategyTypes then
3:     get sub list of Shot Results with strategy as supplied in the argument
4:     get sub list of Shot Results from (3) with the confidence as supplied in
       the argument
5:     retrieve strategy specific counter
6:     set result to true or false based on heuristics of the strategy specific
       counter
7: end if
```

The heuristic for Shot Learner is more semantic than Level Selection. There is no equation to make a decision on but rather an assumption that doing an action that results in some quantified value is better than doing some action that results in no value. We now proceed to discuss in more detail the strategy specific counter and heuristics for the first set strategy types.

- For the strategy type `targetPig`, the strategy specific counter is the number of pigs killed for the argument Target. Probability is calculated as the ratio of number of pigs killed to number of times this target has been executed. The strategy specific heuristic for `targetPig` is that if on average one pig is killed for this Target, then this Target is a good Target.
- For the strategy type `domino`, the strategy specific counter is the damage points. The probability is calculated as the ratio of total damage points to the number of times this target has been executed. The strategy specific heuristic for `domino` is that if the average of damage points is greater than 500 for this Target, then this Target is a good Target.
- For the strategy type `blackBird`, the strategy specific counter is the damage points. The probability is calculated as the ratio of total damage points to the number of times this target has been executed. The strategy specific heuristic for `blackBird` is that if the average of damage points is greater than 500 for this Target, then this Target is a good Target.
- For the strategy type `collapseStructure`, the strategy specific counter is the damage points. The probability is calculated as the ratio of total damage points to the number of times this target has been executed. The strategy specific heuristic for `collapseStructure` is that if the average of damage points is greater than 750 for this Target, then this Target is a good Target.
- For the strategy type `tnt`, the strategy specific counter is the damage points. The probability is calculated as the ratio of total damage points to the number of times this target has been executed. The strategy specific heuristic for `tnt` is that if the average of damage points is greater than 1000 for this Target, then this Target is a good Target.

- For the strategy type `heavyObject`, the strategy specific counter is the number of pigs killed. The probability is calculated as the ratio of total number of pigs killed to the number of times this target has been executed. The strategy specific heuristic for `heavyObject` is that if the average of number of pigs killed is greater than 1.5 for this Target, then this Target is a good Target.
- For the strategy type `whiteBird`, the strategy specific counter is the damage points. The probability is calculated as the ratio of total damage points to the number of times this target has been executed. The strategy specific heuristic for `whiteBird` is that if the average of damage points is greater than 250 for this Target, then this Target is a good Target.

4.3.2 Shot Result

After a shot has been executed, we retrieve the results of the shots and store them as an object of the `ShotResult`. The results of the shots include, the number of pigs killed and the damage points. This is then combined with the previous known knowledge of the shot, the bird type on the slingshot, the strategy to be used in the shot and the confidence of the shot. This class is a Plain Old Java Object (POJO).

4.3.3 TargetAndShot

This class collects two objects. First is the Target object that the `ShotSelection` had chosen to hit next and the Shot object that was generated for the corresponding Target object. Instead of returning just the Shot from the `ShotSelection`, we modified it to return a `TargetAndShot` object which consists of both the Target object and the Shot object.

4.4 Shot Level Evaluation

Below we present the evaluation of the Shot Selection for the first ten levels from the original levels of the game, compared with the agent at the start of the semester –

Original Levels	BamBird (at the Start of the Semester)	BamBird (no_eclipse)
Level 1	29820	18700
Level 2	50589	51130
Level 3	30380	42530
Level 4	20878	27880
Level 5	0	64030
Level 6	33013	17510
Level 7	28348	27420
Level 8	24316	37930
Level 9	29768	41000
Level 10	47458	38040
Total	294566	366170

Table 1 Shot Selection Comparison

We observe that our agent `no_eclipse` performed 24.30% better than the agent at the start of the semester, scoring 71604 points more. We also observe that the `no_eclipse` agent was able to clear Level 5. The maximum score for of Level 5 of other agents is, `DATALAB` 64460, `EagleWing` 64460 and `IHSEC_wAli` 64740.

5 Weka Machine Learning

5.1 Introduction

Weka is a collection of machine learning algorithms for data mining tasks [6]. It includes methods for data mining problems such as regression, classification, clustering, association rule mining, and attribute selection [6]. Weka is used by applying a learning algorithm to a given dataset and analyzing its output. Another way of using Weka, which is relevant for our use in BamBird agent, is to generate predictions on new instances, and a third way is to apply different learning algorithms and compare their outputs and performances.

5.2 Implementation

For the use in BamBird, Weka is used to classify a new instance. The class to be learned is whether a shot is good or bad, this is called a relation in Weka. This relation is dependent on attributes, and the attributes for the BamBird agent are, strategy name, confidence class, damage points and pigs killed. Weka uses an ARFF[7] file. The ARFF file contains two sections, the header section and the data section. We give a brief description. Below we show the header of the ARFF file. The header contains the relation, a list of attributes and their data types.

```
@relation good-shot

@attribute strategy-name    {targetPig, domino, blackBird, collapseStructure, tnt, heavyObject, whiteBird}
@attribute confidence-class {A, B, C}
@attribute damage-points    REAL
@attribute pigs-killed      REAL
@attribute class            {good, bad}
```

Fig 3 ARFF Header Section

The header section of the ARFF file is followed by the ARFF Data Section. It contains the data declaration line and the dataset. Below we show a snapshot of our training dataset. Default values from section 4.1 are used to mark a shot as good or bad. Confidence class is identified as follows: class A from 0.5 to 0.6, class B from 0.6 to 0.9 and class C from 0.9 to 1.0.

```
@data

domino,C,0,0,bad
domino,C,0,0,bad
targetPig,C,8760,2,good
domino,C,16100,3,good
targetPig,C,5320,2,good
targetPig,C,12530,3,good
targetPig,A,1160,0,bad
targetPig,A,6440,2,good
heavyObject,C,32580,6,good
```

Fig 4 ARFF Data Section

In order to classify a new instance, a new ARFF file is created at runtime, called `unknown.arff`. The header of the `unknown.arff` is the same as the header described above, but the data section is different. Instead of having an entire dataset, only one data is provided, described below.

```
targetPig, A, ?, ?, ?
```

The `?` is the dataset signifies unknown data. To classify this instance, we know that the strategy is `targetPig` and we know that the `confidence-class` is `A`, but `damage-points`, `pigs-killed` and shot `class` are unknown. We use the NaiveBayes algorithm as implemented by Weka to classify this new instance.

Following we present the updated `Meta Algorithm` for Weka and the `ClassifyShot` Algorithm, before proceeding to evaluation of the results.

Algorithm 3 Meta-Updated

```
1: Load the ShotLearner database
2: loop
2:     if GameState != Playing then
3:           choose new Level using the new equation
4:     end if
5:     take a screenshot from the scene and analyse
6:     generate plans
7:     select the first Target from the plans and send it to WekaTester.
8:     WekaTester classifyShot marks the Target good or bad.
9:     if goodTarget then
10:          execute the selected shot
11:    else
12:          remove the target from plan
13:          go to 7.
14:    end if
15:    add shot to the shot-list of the current level
16:    if GameState == WON or GameState == LOST then
17:          add level to database
18:    end if
19:    add the shot to ShotLearnerDatabase
20:    create a ShotResult object with the results of the shot
21:    save the ShotResult to a log file
22: end loop
```

Algorithm 4 Weka:ClassifyShot

```
1: create unknown.arff file.
2: write the header.
3: transform target into ARFF data instance
4: invoke NaiveBayes classifyInstance and save into prediction string
5: return prediction string.
22: end
```

Line 7 – 14 mark the difference in the updated `Meta algorithm`. A `Target` is selected from the generated list of plans and is sent to `WekaTester`, where this target is transformed into an instance and written to `unknown.arff` file.

There are two points of interest in `Algorithm 4`. The first is the transformation of `Target` object into ARFF data instance. This is trivial and is achieved by string manipulation and string writing onto the `unknown.arff` file. The second interest point is the invoking of `classifyInstance` of `NaivesBayes` algorithm. This is also trivial because the logic is abstracted behind a simple method call, shown below.

```
double predNB = nb.classifyInstance(newInst);[8]
```

where `predNB` is the predicted class, `nb` is the instance of `NaiveBayes` class algorithm and `newInst` is the new data instance that we wish to classify and predict the class of.

5.3 BamBird Weka Evaluation

Below we present the evaluation of the results achieved by Weka, and compare it to the results of BamBird agent at the start of the semester and the BamBird no_eclipse agent.

Original Levels	BamBird (at the Start of the Semester)	BamBird (no_eclipse)	BamBird Weka
Level 1	29820	18700	18164
Level 2	50589	51130	50153
Level 3	30380	42530	42530
Level 4	20878	27880	27729
Level 5	0	64030	64088
Level 6	33013	17510	17314
Level 7	28348	27420	27091
Level 8	24316	37930	37387
Level 9	29768	41000	40655
Level 10	47458	38040	37946
Total	294566	366170	363057

We observe that the overall score achieved with Weka is 3113 points less. In practice, we conclude that this is not a significant difference, hence the results of `BamBird no_eclipse` and `BamBird Weka` are comparable. This is so because of the choice of the default values of strategy types to define a good shot, as described in section 4.1. Somehow these default values mark a sweet spot. If the default values are increased or decreased, `BamBird Weka` performance goes down in terms of total points. This is so because if the default values are increased, then more good shots are identified as bad shots, leading to shot scarcity which leads to `BamBird` shooting default shots. If the default values are decreased, then more bad shots are classified as good shots which leads executing shots that don't generate any positive result.

6 Conclusion

6.1 Future Improvements

6.1.1 Time Remaining Level Selection

The new level selection formula does not use the amount of time remaining and the heuristics of how long a level takes to finish into account. It would be very effective to include such two factors into the equation because the 2018 Agent considers all levels to take equally long to solve, which is an assumption that works impractically in the gaming world. Some heuristics can be saved which could map level maximum points to time taken to finish the level or another strategy could be to use a mapping between the number of birds available in a level to the time taken to finish the level.

6.2.2 Type of Level

The 2018 Agent does not currently identify the level type. A definition of what constitutes a level type also has to be identified. It could be that some levels have something similar, for example level X has a structure and level Y also has. So the shot selected for Level X could also be used for level Y. There is a scope of employing machine learning and identifying and classifying similar levels.

6.3 Conclusion

In the duration of the Summer Semester 2018, we improved an intelligent agent by supplementing it with learning. Our approach has yielded success and we conclude that the BamBird 2018 no_eclipse Agent performs better than the BamBird 2017 agent. Implementing learning algorithms from Weka do not influence the results.

7 Literature

[1] BamBird 2017 Project Report

[2] https://en.wikipedia.org/wiki/Heuristic_(computer_science)

[3] Artificial Intelligence a Mordern Approach Russel Peter

[4] Jochen Renz et al. Aibirds: The angry birds artificial intelligence competition. In AAAI, pages 4326–4327, 2015.

[5] Chris Holt. "Angry Birds Review". Macworld. Archived from the original on June 14, 2010. Retrieved June 23, 2010.

[6] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[7] https://www.cs.waikato.ac.nz/ml/weka/arff.html

[8] http://weka.sourceforge.net/doc.stable/