Project Report

AI Birds – Qualitative Physics Knowledge and Strategy Learning
to Master a Strategy Game

Lukas Lengenfelder, Katharina Rupp, Kevin Volkert

1 Abstract

Developing agents for simulation games are a wide-spread research in artificial intelligence. Nowadays, there are even competitions, where agents compete with each other or with humans.

The following project report describes an intelligent agent developed for the Angry Birds AI Competition. The report focuses on a method that uses unique strategies for different bird types of the game and investigates if this approach is very effective to collect high scores for the levels.

Contents

List of Tables

2 Introduction

Angry Birds has become a popular physics simulation game over the years. The goal of the game is to destroy all pigs in a scene by shooting with birds from a slingshot. Furthermore, there is a score, which rates the player's abilities by adding up points for destroyed objects and pigs.

For humans, the game is easy to understand and figure out where to best aim the birds to achieve the best outcome. Physics simulation games in general are mostly rapidly understood, as humans can analyse what would happen in real world. Thus, they can transfer their knowledge to the game. For Artificial Intelligence, however, it is a big challenge. Many factors have to be integrated in the calculations, such as location, gravity, consistency of the objects and steadiness of structures.

The bird type, with which is shot, furthermore has to be included in the calculations. This is an essential part, as different bird types have different abilities. For instance, a blue bird parts in three little birds when triggering the effect and is especially effective when destroying ice blocks. Using the bird types not correctly can have a huge impact on the destruction and score. Consequently, it is fundamental for AI in Angry Birds to focus on the different bird types and adjust calculations and strategies to them.

The project aimed to develop an agent for the Angry Birds AI Competition[1]. For this competition, the task is "to develop an intelligent Angry Birds playing agent that is able to successfully play the game autonomously and without human intervention. The long term goal is to build AI agents that can play new levels better than the best human player". In the competition, the agent plays prescribed levels of Angry Birds without human intervention. There are multiple rounds, in which the agents with the highest score from all levels continue to the next round. In the final round, the agent with the highest score wins the competition.

The following report describes the development of an agent, which focuses on the different bird types. It furthermore evaluates the effectiveness of this approach. The report first states the relevant research in this field we researched, then details the changes made to the version of 2016, including a description of our rule set. Concluding we discuss a benchmark of the agent and propose further work based on our findings.

[1] `http://aibirds.org/`, accessed on 29. September 2017.

3 State of the Art

Many researchers have developed agents for Angry Birds before. For instance, Zhang and Renz used an approach that analyses a structure to identify strengths and weaknesses [1]. They developed a calculus that evaluates structural properties and rules. For each building block, this calculus identifies which properties and rules are fulfilled. Then it decides which block should be shot.

This so called "Extended Rectangle Algebra" (ERA) is based on Qualitative Spatial Reasoning. The challenge of Spatial Reasoning in AI is that humans posses some kind of common sense knowledge. If an object is falling, for example, any human can tell it is falling without any theoretical knowledge about physics. In [2], the authors state that Qualitative Spatial Reasoning in AI tries to formalize this implicit knowledge about the physical world.

As detailed in [3], Qualitative Spatial Reasoning can also be used to improve AIs in various video games, most notably path finding issues in strategy games.

Another way Qualitative Spatial Reasoning can be used is shown in [4]. Ge et al. present a way to improve computer vision systems for video games by enabling those to infer unknown objects based on the calculated stability of a structure. The calculation of stability is done using the previously mentioned ERA.

Calimeri et al. introduce an agent in [5] that also participated in the 2013 and 2014 Angry Birds competition. They used HEX programs that are an enhancement of answer set programming programs. Furthermore, they integrated answer set programming, which is a paradigm of declarative programming for knowledge representation and reasoning.

After our research for possible approaches, we decided to develop our own, because none of the ideas fit our aspirations. It slightly picks up on previous research, since we still had to face the issue of somehow modelling the implicit knowledge we posses when playing the game. For our approach, we tried to state why we, as human players, choose the shot we perform as formalized as possible. From this foundation we modelled our strategies and the conditions in which they shall be applicable.

4 The BamBird 2017 Agent

The BamBird agent submitted to the 2017 *Angry Birds AI Competition* is based on an agent developed by the BamBirds team of 2016. In the previous competition BamBird won using a main gameplay loop, the shot calculation from the provided naïve agent, a knowledge representation and a set of Prolog rules.

4.1 Changes from the 2016 Agent

The goal of the project was to improve the old agent without discarding our predecessors' work. The previous agent was developed under time pressure, and thus, with only little documentation. After familiarizing with the relevant parts of the code, which is mostly code related to knowledge representation and planning, we spent a large amount of time on refactoring to prepare for changes. The Prolog rules in the old planner were considered too convoluted, however. Therefore, we decided to implement a new planner from scratch (see 4.2) and use the old planner as fallback (see 6.1).

4.1.1 Refactoring and Fixes

In order to implement our ideas we observed the need to do much refactoring on the old code. We made the program logic more obvious and added accurate documentation. We removed a lot of unused code and improved the interfaces to the knowledge representation and planning.

In addition to our functionality-preserving changes we also fixed a few bugs that became apparent during refactoring and testing. Most importantly, the detection of hittable objects (*isHittable* predicate) returned incorrect results in some situations. This was a big problem because it resulted in flawed knowledge bases which prevented several rules in the planner from working as intended. The *isAbove* predicate was incomplete: for each object at most one object was found to be above it. This, too, prevented a few rules from working properly. We changed this to include all appropriate objects in a level.

Further, we added features to the knowledge representation. The coordinates of objects are now added as a comment in the Prolog code to make objects easier identifiable. A new predicate called *isTower* was added, which states whether a given structure is higher than wide.

For the build process of the agent, meaning running the agent for testing purposes or creating a .jar-file for handing in at the competition, we used the tool Apache Ant. This simplified test runs of the agent tremendously.

4.2 Strategies for Bird Types

In the following, the different strategies, which the agent uses, are described. It is especially focused on the different bird types. In total, six different strategies were developed. The first one concentrates on targeting a pig directly, the second one on collapsing a structure and the third one on black birds. The fourth strategy aims for a domino effect, the fifth to shoot a TNT block and the last one to drop a heavy object.

4.2.1 Strategy for Targeting a Pig

This strategy is intended to be a fallback in case no specialized strategy is applicable. It simply targets pigs, with varying confidence based on the colour of the bird, the number and the type of objects it has to penetrate in order to reach the pig. The calculation of this penetrating shot starts with a fixed value of energy, independent of bird type or materials, and is then sequentially reduced using a predefined impact mapping that considers the aforementioned parameters.

The foundation of this map are the values used by the game itself. In order to build a level editor for creating test levels ourselves, another project group reverse engineered the game and found files containing values used presumably for the physics calculation of Angry Birds. We used the values found in the file `Materials.json`, specifically the values for damage multipliers per bird and material combination. This file helped us to keep our own scale relational and consistent with the game's physics simulation. In the game files we detected eight distinct numerical values we adapted to our scale from 0 - 10. The mapping we used can be found in table 1, with minor adjustments applied due to previous experience.

Value from game files	Mapped to...
0,3	4
0,5	4,5
0,8	5
1	5,5
1,2	6
2,4	8
2,5	8,5
4	9

Table 1: Mapping of game file values for our calculation.

4.2.2 Strategy for a Domino Effect

The goal of the domino strategy is to produce a domino effect on multiple structures. The strategy first checks if the target belongs to a structure and if there are more than two structures that would be afftected by the shot. If there are at least three afftected structures, confidence is calculated. For the calculation, energy is first set to a basic value and then sequentially increased for each affected structure. Similar to the strategy for targeting a pig, an impact is integrated. This impact is generated considering the birdtype and the material of the target. The assigned impacts, which are distributed between 0 and 10, can be seen in table 2.

Birdtype	Material	Impact
Red Bird	-	9.5
Yellow Bird	Ice	3
Yellow Bird	Wood	2
Yellow Bird	Stone	1
Blue Bird	Ice	2
Blue Bird	Wood	8
Blue Bird	Stone	8

Table 2: Impacts for the domino strategy.

4.2.3 Strategy for Black Birds

The strategy `blackBird` delivers plans for shooting with a black bird. There is a separate strategy for black birds, because they have a skill that differs a lot from the other birds' skills. Red birds, yellow birds and blue birds each have a different skill, but they all have an impact in only one direction. Black birds have the ability to have an impact in more directions, because they explode and thus, destroy everything that surrounds them in this moment.

`blackBird` delivers two plans with different confidences. First, if a structure can collapse on another structure, an object is shot that belongs to the second structure. Consequently, the bird explodes in between the two structures and both structures are demolished. Second, again if a structure can collapse on another one, there is also the possibility to shoot at an object that belongs to the first structure. Thus, the first structure indeed collapses on the second one. This plan has a lower confidence as the first plan, however, so that it would only be chosen in a second try. It thus states another option for the meta strategies.

4.2.4 Strategy for Collapsing a Structure

The strategy for collapsing a structure strives to detect a structure, which contains a pig, and then tear this structure down. Thus, this strategy is applied, if a structure with a pig is found in the level. It furthermore excludes the case that a stone object is behind the target, because we discovered that the strategy would then fail. Also, the bird is shot on an object that is either on the ground or low in the structure. For the confidence, impacts from the domino strategy are integrated. Those impacts can be seen above in table 2. We decided to include those impacts, as we strive to tip the structure over like in the domino strategy and not penetrate the material like in the strategy for targeting a pig.

4.2.5 Strategy for TNT

The strategy `tnt` focuses on finding a TNT block and aiming for this object. It is assumed that the designer placed this object in that position on purpose and it has an important effect when shooting it.

The plan checks, if there is an object, which is both hittable and of the material TNT. It furthermore examines, if the TNT belongs to a structure. This is essential, as it has to be excluded to shoot a TNT that has no effect at all. Szenarios for this could be that either the designer might want to confuse

the player or that a previous shot changed the szene so much that the TNT has no impact anymore. Once detected this kind of TNT, the object is shot directly.

4.2.6 Strategy for a Heavy Object

The idea behind the predicate *heavyObject* is to find such a heavy object above as many pigs as possible and dropping it on these pigs. The strategy thus relies on the assumption that said heavy object is placed at a position on purpose by the designer of the level (similar to strategies involving TNT blocks). A heavy object is, by definition of our knowledge representation, an object in the shape of a ball, since these are most likely placed in a position so they can either roll over or fall on pigs. Inspiration and benchmark for this strategy are levels 1 to 5 of the Poached Eggs stage, which the agent is able to solve by using only one bird.

To find out if this strategy is applicable, it checks if there exists a ball in the current scene that is above one pig. The strategy then starts calculating the confidence for the various targets of the shot. Possible targets are: the object the ball rests on (predicate *isOn*), the object that supports the ball or the ball itself. The starting values (or potential of the shot) for the calculation differ by target, as we balance the strategy differently for each kind of target. The best possible shot to us is aiming for the object the ball is on if it is hittable, less so if it is not hittable, followed by the supporting object and the ball itself in this order. Although latter targets will probably never be at a higher confidence than the first two, we wanted them included so that the meta strategies have more shot options to chose from in case we play a level twice.

Finally, the confidence of the shot is calculated by increasing the potential of the shot per pig that is below the ball until there are no more (unique) pigs found below the ball or the confidence would exceed our upper boundary of 1.

An idea we had upon playing level 1-14 was to decrease the potential and by that the resulting confidence further if the ball was of small size. In this level there are a number of small-sized balls above pigs, it is not a good idea to aim for those however. The idea was discarded due to time constraints, but it may be worth considering.

Another weak point of *heavyObject* is, that it does not take the material of the target or the bird in the slingshot into consideration. We skipped this due to the limited applicability and universal appeal of the strategy.

5 Evaluation of the Agent

To evaluate the agent, we performed a benchmark as described by the organizers of the AIBirds-competition[2]. This allowed us to compare the agent to the previous version of the BamBirds agent, as well as all previously submitted agents. The conditions of the benchmark stated, that agents had 63 minutes per stage, which means roughly three minutes per level, to do whatever operations it wanted to perform. The levels used were the first and second stage of the Poached Eggs levels, which equals to a total of 42 levels (21 per stage).

The results of the benchmark can be found in Table 3 and 4 respectively. Blank cells represent unused attempts to solve a level, while 0 points mean the agent failed to solve the level. The agent was able to score a total of 798.560 points in all 42 test levels.

Level	First try	Second try	Third try	Max. points
Level 1-1	9.350	30.450		30.450
Level 1-2	52.370			52.370
Level 1-3	41.970	41.970		41.970
Level 1-4	19.900			19.900
Level 1-5	0	0	0	0
Level 1-6	34.940			34.940
Level 1-7	39.600			39.600
Level 1-8	39.160	0	0	39.160
Level 1-9	31.890			31.890
Level 1-10	0			0
Level 1-11	0			0
Level 1-12	55.170			55.170
Level 1-13	0			0
Level 1-14	47.880			47.880
Level 1-15	0			0
Level 1-16	45.240			45.240
Level 1-17	38.810			38.810
Level 1-18	0	0	39.390	39.390
Level 1-19	36.710			36.710
Level 1-20	0	41.790		41.790
Level 1-21	68.670			68.670
SUM	561.660	114.210	39.390	663.940

Table 3: Results of the benchmark for the first stage of Poached Eggs

Comparing these values to those of the previous version, the agent did perform significantly worse. The BamBirds agent submitted to the competition in 2016 was able to score a total of 1.237.230 in all 42 levels. One significant reason for this is that the 2017 version of the agent crashed while benchmarking the second stage and tried to restart itself endlessly to no avail. Since it was not stated how the benchmarks performed by the organizers handled such a situation, we

[2] `http://aibirds.org/benchmarks.html`, accessed on 29. August 2017.

Level	Firts try	Second try	Third try	Max points
Level 2-1	0			0
Level 2-2				
Level 2-3				
Level 2-4	0			0
Level 2-5	0			0
Level 2-6	55.520			55.520
Level 2-7	0			0
Level 2-8				
Level 2-9				
Level 2-10				
Level 2-11				
Level 2-12	35.260			35.260
Level 2-13	0			0
Level 2-14	43.840			43.840
Level 2-15	0			0
Level 2-16	0			0
Level 2-17	0			0
Level 2-18	0			0
Level 2-19	0			0
Level 2-20	0			0
Level 2-21	0			
SUM	134.620			134.629

Table 4: Results of the benchmark for the second stage of Poached Eggs

decided not to restart the benchmark for stage two, especially since we decided to take some leeway before. The agent was stuck on level 2-1, not shooting any more even though the agent gave the order to shoot. Since this could possibly be an issue on our test hardware or the server handling the commands given, we decided to fire the birds manually and score the level with 0 points. Regarding the crash, it has to be noted, however, that the agent performed worse on stage one as well, where no crashes occurred.

The results of stage one further show that the agent was able to improve by over 100.000 points by playing certain levels again. Five levels were played at least twice, in three of which the agent scored more points than before.

While evaluating the agent's performance, we noticed that it took the agent far longer to shoot the bird after deciding for a strategy in its second and third tries. This could be related to the computation time needed for the meta analysis, but it should be worth investigating. If this time could be cut short, the agent could repeat far more levels, which seems highly beneficial to its performance.

As repeatedly playing levels improved our rating, it shows that our strategy component does not always award the most fitting plan possible with the highest confidence. So, either the time needed to solve a level needs to be reduced to factor in repetition of levels by design or the confidence rating needs to be overhauled.

A big issue of our agents performance was also the inability to solve a lot of levels. The agent failed to solve 16 of 42 levels, or 16 of 35 respectively, factoring in the crash on stage two. With an average of roughly 29.500 points per level (using the overall points of the 2016 agent's benchmark) this equals 472.000 missed points, 678.500 when adding the unplayed levels due to the crash. As making the agent robust enough to solve all levels can potentially double the points achieved, this is an area that needs to improve.

6 Future Work

In this section we want to address work that we were not able to complete in time or ideas we find worthwhile considering.

6.1 Handle Multiple Planning Results

Even though we were able to implement concurrently running planners, we were short on time to properly use the additionally generated plans. In the current state, the agent is able to run the new 2017 planner as well as the 2016 version concurrently, combining both their returned plans into one ordered list. The issue, however, is that both planners use different ways to rate their plans: the 2017 version uses a shot confidence with values between 0 and 1 while the 2016 version uses a point based ranking without upper boundary. It is thus impossible to simply combine both results, since the 2016 version would always be favoured over the version of 2017. In an attempted workaround we only used plans of the 2016 version if the 2017 version would fail to return any plans or the plans were known to be bad from previous attempts at the same level. With this extensible framework and a way to properly compare or handle plans with different rating systems, this could be a highly beneficial addition to the meta reasoning of the agent.

6.2 Revise Generation of Knowledge Representation

While working on our strategies, we discovered quite a few inconsistencies within the knowledge representation in form of the situation-files. For example, there are cases in which the *isLeft* and *isRight* predicates are not symmetric, i.e. a is left of b while b is not right of a. We were able to track some other issues down (*isAbove* and *isHittable* as mentioned in 4.1.1), but there might be more we were not able to find. Another big issue we faced was the structure detection algorithm, as it was too imprecise for our purposes. Piles of rubble in shots 2 and beyond were often detected as multiple structures, corrupting the results of our planner.

6.3 Increase Robustness of the Agent

As made apparent by the benchmark and the results of the competition, the agent, in its current state, can get stuck on a level. Since this will lose us the most points (see 5), increasing the robustness of the agent in the case of a crash is most important. Apart from the crash issue, the agent will need improving on solving levels, be it with a lower score. Also, reducing the time needed for the shot selection (see also 5) may further increase the beneficial influence of the meta strategies in place.

6.4 Rebalance Strategies

The benchmark in 5 shows that the shot planner is most confident which may not be the best to take because of the big profit of the meta analysis. As long as the agent is not able to replay levels to a greater extent, it is important to rebalance certain specialized strategies. Thus, some strategies should be made

not applicable in certain situations by reducing their confidence, e.g. playing *collapseStructure* on rubble falsely detected as a structure.

7 Conclusion

Over the course of this project we developed an intelligent agent playing Angry Birds on the code base of the BamBirds agent of 2016. For our approach, we tried to model the implicit knowledge humans posses about physical simulation games by formalizing our way to solve the levels.
The changes made us aware of errors and bugs in the code, which we fixed. Further, we reworked the code to adhere to software engineering principles and simplified the compilation, execution and distribution process.
Evaluation and the competition results showed, however, that our agent does not meet the desired standards of performance and does require improvements also detailed in this paper.

References

[1] Peng Zhang, Jochen Renz, et al. Qualitative spatial representation and reasoning in angry birds: The extended rectangle algebra. In *KR*, 2014.

[2] Kenneth D Forbus, Paul Nielsen, and Boi Faltings. Qualitative spatial reasoning: The clock project. *Artificial Intelligence*, 51(1-3):417–471, 1991.

[3] Kenneth D Forbus, James V Mahoney, and Kevin Dill. How qualitative spatial reasoning can improve strategy game ais. *IEEE Intelligent Systems*, 17(4):25–30, 2002.

[4] Xiaoyu Ge, Jochen Renz, and Peng Zhang. Visual detection of unknown objects in video games using qualitative stability analysis. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):166–177, 2016.

[5] Francesco Calimeri, Michael Fink, Stefano Germano, Andreas Humenberger, Giovambattista Ianni, Christoph Redl, Daria Stepanova, Andrea Tucci, and Anton Wimmer. Angry-hex: an artificial player for angry birds based on declarative knowledge bases. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(2):128–139, 2016.