AngryHEX: an Artificial Player for Angry Birds Based on Declarative Knowledge Bases

Francesco Calimeri[1], Michael Fink[2], Stefano Germano[1], Giovambattista Ianni[1], Christoph Redl[2], and Anton Wimmer[2]

[1] Dipartimento di Matematica e Informatica, Università della Calabria, Italy
{calimeri,ianni}@mat.unical.it, stefanogermano0@gmail.com
[2] Institut für Informationssysteme, Technische Universität Wien
{fink,redl}@kr.tuwien.ac.at, anton.wimmer@tuwien.ac.at

1 Introduction

This work presents a joint project by the University of Calabria (UniCal) and the Vienna University of Technology (TU Vienna), aiming at developing an Intelligent Agent participating in the 2013 Angry Birds Artificial Intelligence Competition [1]. Angry Birds is a very popular video game where the main goal is to shoot at some pigs by means of birds of different characteristics from a slingshot. The game field (which is static until the player moves) features some structures that shelter pigs. Structures can be very complicated and can involve a number of different object categories with different properties, like wood, ice, stone, etc. The game scenario evolves largely complying with physics laws on a bi-dimensional plane; thus, it is possible, in principle, to infer how a structure will change if hit at a certain position by a certain bird.

The Angry Birds AI Competitions [1] are designed to test the abilities of Angry Birds artificial agents, playing on a variety of levels, on the Google Chrome version of the game. The competition runs on a client/server architecture, where the server runs an instance of the game for each participating agent. Each agent runs on a client computer, and communicates with the server according with a given protocol that allows agents to fetch screenshots of their own game screen at any time. An artificial player can also obtain the current high scores for each level, and can prompt the server for executing a shot, which will in turn be performed in the corresponding game screen. The long term goal of the Competition is to foster the building of AI agents that can play any new level better than the best human players. In order to successfully solve this challenge, participants are solicited to combine different areas of AI such as computer vision, knowledge representation and reasoning, planning, heuristic search, and machine learning. Successfully integrating methods from these areas is indeed one of the great challenges of AI.

2 ASP and HEX Programs

Our agent, called AngryHEX, models its internal knowledge of the game by means of an Answer Set Programming (ASP) [2,4] knowledge base. ASP became widely used in AI and is recognized as a powerful tool for knowledge representation and reasoning (KRR), especially for its high expressiveness and the ability to deal also with incomplete knowledge [2]. The fully declarative nature of ASP allows one to encode a large variety of problems by means of simple and elegant logic programs. The semantics of ASP associates a knowledge base with none, one, or many "answer sets", each usually in one-to-one correspondence to the solutions of the problem at hand. Among the different extensions of ASP, we used HEX programs [3], as they are particularly well-suited when external knowledge and/or external sources of computation must be integrated within the same knowledge base.

3 The AngryHEX agent

We started the development of AngryHEX from the Base Framework provided by the organizers and enriching it with new functionalities, particularly the AI. The actual reasoning is carried out by computing the answer sets of an HEX-program P_{AI} that models the knowledge of the game. P_{AI} is enriched with scene information, and its answers sets encode the targets of choice.

3.1 Base Framework

The Base Framework consists of many parts (see fig. 1). The *Angry Birds Extension* works on top of the Google Chrome browser, and allows to interact with the game by offering functionalities such as capturing the game window or executing actions (e.g., moving the mouse, clicking, zooming). The *Vision Module* segments images, recognizes the minimum bounding rectangles of essential objects, such as birds of various colors and types, pigs, the slingshot, or bricks made of several materials. The *Trajectory Module* estimates the trajectory that a bird would follow given a particular release point set at a given distance and orientation with respect to the slingshot. The *AI Agent* stub is supposed to include the artificial intelligence programmed by participants. The *Game Server* interacts with the *Angry Birds Extension* via the *Proxy* module, which can handle commands like `CLICK` (left click of the mouse), `DRAG` (drag the cursor from one place to another), `MOUSEWHEEL` (scroll the mouse wheel), and `SCREENSHOT` (capture the current game window). The *Server/Client Communication Port* receives messages from agents and sends back feedbacks after the server executed the actions asked by the messages. There are many categories of messages (Configuration messages, Query messages, In-Game action messages and Level selection messages).

3.2 Our extensions to the Base Framework

We extended the Base Framework adding an Executor that communicates with the DLVHEX solver; the Executor encodes the information about the current scene into logic assertions, runs DLVHEX on the composed HEX-program and

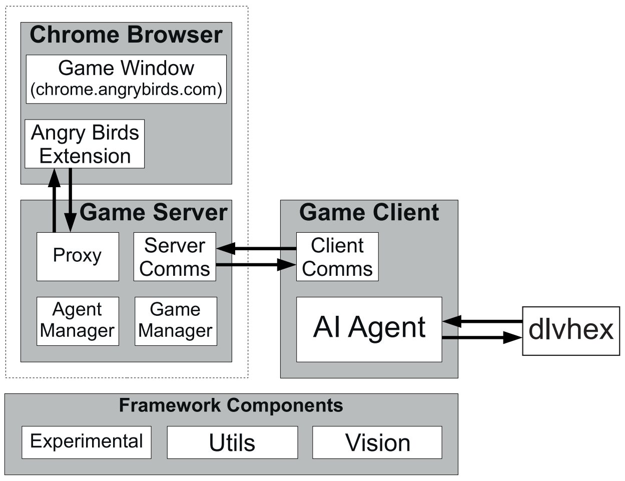

Fig. 1: Base Architecture

parses the output, thus identifying the next target. We also improved the Vision Module, adding the possibility of recognizing the orientation of the blocks; indeed, the Base Framework does not correctly recognize blocks that are not parallel to the coordinate axes. The Trajectory Module has been extended with the possibility of aiming a trajectory at the left and top side of objects; ideed, the original module within the Base Framework aims only at the centroid of an object, even if it is "covered" by other objects, and thus not directly shootable. We implemented also other minor adjustments, fixes and improvements.

3.3 Artificial Intelligence

We developed two different Intelligence layers for AngryHEX: the **Tactic** layer, which plans shots, and decides how to complete a level; and the **Strategy** layer, which decides the order of levels to play and possible multiple attempts to the same level. The **Tactic** layer is declaratively implemented using the DLVHEX solver, which computes optimal shots based on information about the current scene and on knowledge modeled within the HEX program P_{AI}. In particular, the **Tactic** layer behaves according to the following scheme:

- Input: Scene information encoded as a set of logic assertions S (position, size and orientation of pigs, ice, wood and stone blocks, slingshot, etc. as obtained by the Vision Module); a knowledge base P_{AI} encoding knowledge about the gameplay;
- Output: "Answer sets" of $P_{AI} \cup S$, that contain a dedicated atom describing the target to hit, and some information about the required shot;
- Physics simulation results and other information are accessed within P_{AI} via the so-called External Atoms constructs. External atoms allow to access information which is external to the knowledge bases, like, e.g.,

- asking if an object is “stable” (prone to an easy fall);
- asking if object B will fall if A falls;
- asking which objects intersect with the trajectory of a bird after hitting a given object;
- computing distances between objects;
- asking if an object O is “shootable” (i.e., there exist a trajectory where O is the first intersecting object);
- finding the best trajectory for a White Bird (the white birds have a peculiar behavior and require a special treatment);
- asking if an object can be “pushed” by another.

The knowledge modelled by P_{AI} is the following:

- Define each shootable target as an object which has a direct and unobstructed path from the slingshot;
- For each shootable target T, compute a measure of the estimated damage that can occur on all other objects if T is hit. The specific behavior of each bird type is taken into account (e.g. yellow birds are very effective on wood, etc.), and the estimated damage is properly weighted by a probability value;
- Targets are ranked by their priority, e.g., maximizing the estimated damage to pigs first; estimated damage to other objects is taken into account only for targets which tie in the estimated damage for pigs.

As an example of a logic rule used in AngryHEX, the following “computes” the likelihood of damage when an object pushes an adjacent one:

$$\begin{aligned} pushDamage(Obj_B, P_A, P) \leftarrow\ & pushDamage(Obj_A, \_, P_A), \\ & P_A > 0, \&canpush[ngobject](Obj_A, Obj_B), \\ & pushability(Obj_B, P_B), P = P_A * P_B/100. \end{aligned}$$

here the *&canpush* predicate out-sources to a C++ library the geometric calculations required to assess whether an object is within range of another (i.e. Obj_A can make Obj_B fall by domino effect); on the other hand, the *pushability* predicate is defined using empirical knowledge of the game and defines how much an object can be “pushed” in terms of its shape and material (e.g. long rods are very pushable, etc.).

We have developed also other rules that take into account the direct damage and the fall damage of each object. These depend on the probability of destroying a particular object type, and on the importance of the fall of an object, given its material type. Static object damage values are combined in order to form causal chains which terminate using an energy loss estimate. Energy losses take into account the residual energy available when the effects of a shot propagate from an object to another. For instance the following assertions:

$$damageProbability(blue, wood, 10).\ damageProbability(yellow, wood, 100).$$
$$damageProbability(blue, ice, 100).\ \ damageProbability(yellow, ice, 10).$$

encode the damage probability of an object depending on the object material and on the type of bird hitting the object at hand. Such values were encoded

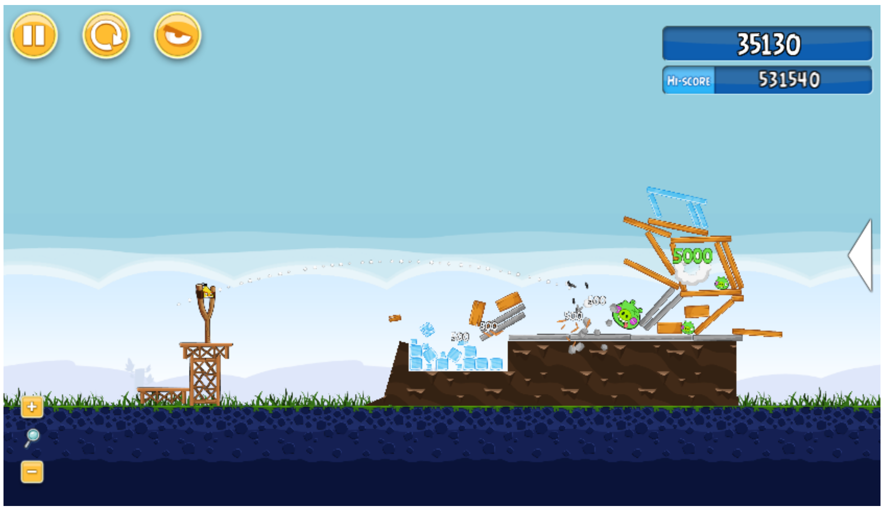

Fig. 2: A successful shot featuring an exploding black bird.

empirically based on specific knowledge of the game (e.g. blue birds are very effective on ice, etc.).

The **Strategy** layer decides, at the end of each level, which level to play next. This layer is implemented in Java according to the following scheme:

1. First, each available level is played once;
2. Then, levels where the agent score differs most from the current best score are selected (up to a limited number of attempts);
3. Next, levels where AngryHEX achieved a score higher than the current best scores and that have the minimum difference from the best score, are selected (up to another limited number of attempts);
4. Finally, the **Strategy** layer tries to play a random level (actually, this never occurred during the available timeframe in the competition rounds).

For each level, the **Strategy** layer keeps tracks of previously achieved scores and previously selected initial target objects. Previously selected targets are excluded, in order to force a different tactic per each attempt on the same level.

4 Computational Performance Considerations

Since the tactic layer is repeatedly prompted for deciding the chosen target on the current scene, its efficiency is crucial for achieving good time performance of the overall agent. The core of the tactic layer is an evaluation carried over the logical knowledge base P_{AI} coupled with a set of assertion S describing the current scene. Thus, both the size of S and P_{AI} affect in principle the performance of the tactic layer. Another performance factor concerns the time performance and the number of calls to external libraries.

Concerning the size of S, this is directly proportional to the number of objects in the current scene: there is one fact per each object in the scene, plus a constant number of facts (three) encoding respectively, the slingshot position, the current bird type, and the scale factor of the scene. P_{AI} is instead a fixed logic program comprising about 3 hundred statements, made of both rules and facts encoding fixed knowledge of the domain. For what calls to external libraries are concerned, these were optimized for reducing the number of possibly redundant calls[3]. External calls proved to be a key-point for what the impact on efficiency is concerned. Indeed, by means of external calls, we were allowed to out-source to imperatively coded modules a number of tasks in which ASP and logic programming in general are not well-tailored at, like simulation of future scenarios according to laws of physics, approximation of ballistic trajectories, etc.

With the help of an hybrid architecture featuring access to external sources of knowledge, we found that the capabilities of current ASP solvers fit very well the setting of the Angry Birds game and its AI Competition: first, Angry Birds can be seen as a game in between a real-time game and a turn-based one, allowing a fairly large time window for deciding the next shot[4]; and second, the size of the search space of the tactic layer is linearly proportional to the number of objects in the current scene. Note that the above setting can be lifted to a variety of other contexts in which *a)* there is an acceptable time window available for reasoning, and *b)* the set of actions that can be taken by the agent at hand is fairly small, do not underlies an exponentially larger search space, and no look-ahead of arbitrary depth on future scenarios is required (or convenient) to be performed. This generalized context covers e.g. automated room cleaning, semi-interactive turn-based games, etc.

Concerning actual performance, we could not yet perform accurate benchmarking, due to the unavailability to the public of the levels used in the Angry birds AI Competition. However, our testing on the publicly known first 21 levels of the "Poached Eggs" set[5], which reflect the competition setting in qualitative terms (type of materials, shape of objects and type of birds available), allowed us to draw some conclusion. First, the reasoning time was a negligible fraction with respect to the overall time of each shot. Most of the shot time is indeed taken by the animation and simulation of the shot itself (within tens of seconds), while reasoning takes generally less than 1-2 seconds, with some outlier case of no more than 5 seconds; second, given the low reasoning times, we could not appreciate any correlation between the reasoning time and the number of objects available[6], although we expect such correlation should appear clearer in more complex levels.

[3] Also, the burden of repeated identical calls to external sources has been mitigated by caching.

[4] In the Angry Birds AI Competition, depending on the round at hand, each shot was allowed about 10-20 seconds.

[5] "Poached Eggs" is the first level set available in the downloadable version of the game.

[6] In the "Poached Eggs" level set, the number of objects ranges from 8 to 65, including "ground" objects describing terrain areas. Levels proposed to participants in the

5 Conclusion

AngryHEX performed very well in the 2013 Competition, reaching semifinals. Notably, AngryHEX kept the first place out of 20 participants, in both the two qualification rounds. Benchmarking performed by the Competition Organizer after the Competition also shows AngryHEX ranking with the best score in the first 21 levels of the "Poached Eggs" level set. The levels used in the semifinal and final stage are unfortunately not publicly available; thus, we could not assess the quality of performance of AngryHEX on them, nor understand the reasons of the lower performance. Nevertheless, we conjecture that levels used in finals and semi-finals are very rich in the number of objects, thus triggering some scalability issues in the architecture of our agent.

We identified many points in which AngryHEX can be improved. In the future, we aim at introducing the planning of multiple shots based on the order of birds that must be shot; we think that it might be useful also to better study the interaction between objects. Some other possible improvements concern performance, and the declarative implementation of the **Strategy** layer.

References

1. Angry Birds AI Competition. http://aibirds.org
2. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
3. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A Uniform Integration of Higher-Order Reasoning and External Evaluations in Answer Set Programming. In: International Joint Conference on Artificial Intelligence (IJCAI) 2005. pp. 90–96. Edinburgh, UK (Aug 2005)
4. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. New Generation Computing 9, 365–385 (1991)

AI Competition qualification rounds were only slightly more complex with levels reaching up to 100 objects.