# DualHEX: an extension of the AngryHEX Artificial Player for AngryBirds

Ana Oliveira da Costa, Isabelly Louredo Rocha, Slavco Elena

Technische Universität Dresden
Computational Logic Group
Seminar: Practical Planning for AngryBirds

**Abstract.** The goal of the Angry Birds AI competition is to build an intelligent agent which can complete the levels of the game better than human players. This task is very challenging, because humans have a good prediction about the physic world, while for computers it is hard to reason about an unknown environment. In this paper we describe our DualHEX AI agent, which is based on the AngryHEX agent of participants of the Angry Birds competition 2013. Our agent models the knowledge of the game by means of Answer Set Programming. In this project we improved the AngryHEX approach by extending the knowledge base of the domain. Our DualHEX agent plans a shot taking into consideration the current and the next bird. It compares the damage probability of both birds to discard targets that suits more the next bird features.

**Keywords:** Answer Set Programming, Artificial Intelligence, Angry Birds

## 1 Introduction

Angry Birds is a popular video game series which was released in December 2009 and immediately gained users attention. The main goal of the game is to destroy pigs and damage their shelters, with the help of different types of birds. The number of birds is fixed for each level and every bird shoots only once per level. Pigs shelters can represent very difficult constructions made of different materials like: ice, wood, stone and etc. Each bird has different abilities that are activated when the user taps on the playable area. For example, blue birds split in three smaller birds, yellow birds are speeding up and white birds drop a egg bomb.

The game is popular due to its colorful and fun interface, free basic use of the program and interesting and challenging tasks. The game respect the laws of physics and the problem of choosing the best target is not trivial.

The Angry Birds AI Competition was created to deal with this problem [7]. The task of the competition is to create an artificial intelligent agent that can play the game better than human players and without human interference. The task itself include a lot of problems to be solved: analyzing the structures and

birds abilities, planning the shot, developing the intelligent agent. Organization provides the basic game software that consists of:

- **computer vision component**, which segments the image and outputs a list of the minimum bounding rectangles of essential objects in the image and outputs real shapes instead.
- **trajectory model**, which evaluate the trajectory that a bird will follow from the exact point.
- **game playing component**, which represents the intelligent agent we need to create.

Our project is based on the work of the competition participants in 2013, the AngryHEX agent [4]. Their approach was to create an agent which models the knowledge of the game by means of an Answer Set Programming knowledge base. They implemented around 300 rules and our goal is to improve their approach by extending the knowkedge base. In our implenentation, DualHEX, we consider the probability of damage on the objects in the game scene for two birds: current and next one, and use this knowledge to choose the most advantageous target.

## 2 Preliminaries

In this section we will introduce the syntax and semantics of Answer Set Programming (ASP).

### 2.1 Answer set programming (ASP)

Answer Set Programming (ASP) is a declarative programming approach with roots in Logic Programming and Non-monotonic Reasoning [3]. It is a fully declarative paradigm optimized for finding solutions of search problems [2]. Its declarative nature along with its non-monotonicity makes it a powerful tool to use in Knowledge Representation related problems [2, 4]. In the ASP paradigm, solutions to a problem correspond to the answer sets of our problem's logic encoding. We will now define the syntax of ASP programs and later we introduce answer sets semantics.

**Syntax** A general logic program is a finite set of general rules. A general rule is an expression of the form [6]:

$$a \leftarrow b_1 \wedge \cdots \wedge b_m \wedge \ not\ b_{m+1} \wedge \cdots \wedge \ not\ b_n$$

where $a$ and $b_j$, with $1 \leq j \leq n$, $m \leq n$, are atoms; *not* is negation as failure. Rules are implicitly universally quantified over the set of variables that occur in them. A rule is composed by *head* and *body*. Given a general rule $r$ they are defined as follows:

- $head(r) := \{a\}$;

– $body(r) := \{b_{l+1}, \ldots, b_m,\ not\ b_{m+1}, \ldots,\ not\ b_n\}$;

In this paper we use an extended version of general logic programs. The rules of our program will include classically negated atoms as well as more than one element in the head of the rule. Those rules are called extended rules with disjunction in the head and are defined as follows:

$$A_1 \vee \cdots \vee A_k \vee\ not\ A_{k+1} \vee \cdots \vee\ not\ A_l \leftarrow \\ B_{l+1} \wedge \cdots \wedge B_m \wedge\ not\ B_{m+1} \wedge \cdots \wedge\ not\ B_n \quad (1)$$

where $A_i$ and $B_j$, with $1 \leq i \leq l$ and $l < j \leq n$, are literals; *not* is negation as failure; and $k \leq l \leq m \leq n$. The *head* and *body* of a rule $r$ of this form are:

– $head(r) := \{A_1, \ldots, A_k,\ not\ A_{k+1}, \ldots,\ not\ A_l\}$;
– $body(r) := \{B_{l+1}, \ldots, B_m,\ not\ B_{m+1}, \ldots,\ not\ B_n\}$;

A rule $r$ with no literals in the body is called a fact. In our setting, facts can be used to describe the expected target's damage after a specific type of bird hits it. This knowledge is known beforehand and it is independent of the current game state. We can encode it by adding the predicate *damageProbability (Bird, Material, Damage)* where *Bird*, *Material* and *Damage* are instantiated with bird types, materials (like wood or ice) and a numerical value that quantifies the expected damage when the first hits the second. In the example shown below we are defining that a yellow bird induces more damage in a wood structure than in an ice structure:

$$damageProbability(yellowbird, wood, 100) \leftarrow \\ damageProbability(yellowbird, ice, 10) \leftarrow$$

A rule $r$ with no literals in the head is called constraint. Constraints are used to remove undesired answer sets from our solution. For example, in our Angry Birds encoding into ASP we want to have at most one definition per possible target in the list of admissible targets to shoot at. This can be encoded by defining the predicate *target(Object, Trajectory)*. This predicate associates a trajectory to an object in the game field. The constraint shown below encodes this restriction:

$$\leftarrow\ target(Obj1, Traj1) \wedge\ target(Obj2, Traj2) \wedge\ Obj1 \neq Obj2$$

**Answer Set Semantics** An ASP program describes properties of our problem's solution. We can then intuitively define the meaning of an ASP program as justifications for our intended solutions. This is the main idea behind stable model semantics for general logic programs introduced by Gelfond and Lifschitz [5]. Later they defined answer set semantics [6] as an extension to stable model semantics to handle programs with rules as defined in (1). We start by defining

the answer set semantics of general programs without occurrences of *not* (negation as failure), and later we explain how to extend this definition to programs with rules with occurrences of *not* and disjunction in the head.

The semantics of an ASP program $\mathcal{P}$ is defined over the set of the ground literals defined by the language of $\mathcal{P}$. This set will be denoted by $Lit$. Without loss of generality, a rule with variables can be seen as shorthand for the set of its ground instances [6]. Consider an arbitrary program $\mathcal{P}$ defined by rules of the form:

$$A \leftarrow B_1 \wedge \cdots \wedge B_m \tag{2}$$

where $A$ and $B_j$, with $0 \leq j \leq m$, are literals. The answer set $\mathcal{M}^{\mathcal{P}}$ of $\mathcal{P}$ is the smallest subset of $Lit$ such that:

- $\mathcal{M}^{\mathcal{P}}$ is a classical model of $\mathcal{P}$;
- all literals in $\mathcal{M}^{\mathcal{P}}$ are justified by some rule of $\mathcal{P}$, i.e. each element in $\mathcal{M}^{\mathcal{P}}$ is in the head of some rule $r \in \mathcal{P}$ such that $body(r) \subseteq \mathcal{M}^{\mathcal{P}}$.

This definition is a straightforward implementation of the idea of programs as justifications.

Lets now consider that our arbitrary program $\mathcal{P}$ contains rules with occurrences of both classical negated atoms and negation as failure, i.e., $\mathcal{P}$ is an extended program. We start by defining $\mathcal{P}^{\mathcal{R}}$ as the reduct of $\mathcal{P}$ with respect to the subset $\mathcal{R}$ of $Lit$. Given a set of ground literals $\mathcal{R}$ such that $\mathcal{R} \subseteq Lit$, $\mathcal{P}^{\mathcal{R}}$ is the extended program obtained from $\mathcal{P}$ by performing the following actions:

- if a literal $L \in \mathcal{R}$, then delete each rule that has a formula *not* $L$ in its body,
- delete all formulas of the form *not* $L$ in the bodies of the remaining rules.

$\mathcal{P}^{\mathcal{R}}$ is a program containing only rules of the form (2). Hence, its answer set is already defined. Lets assume that $\mathcal{M}^{\mathcal{P}^{\mathcal{R}}}$ is the answer set of $\mathcal{P}^{\mathcal{R}}$, then if $\mathcal{M}^{\mathcal{P}^{\mathcal{R}}} = \mathcal{R}$ we say that $\mathcal{R}$ is an answer set of $\mathcal{P}$. It is important to note that an extended logic program may have zero, one or more answer sets.

Lets now consider that our program $\mathcal{P}$ contains rules with disjunction in the head and none of its rules have occurences of *not*. Rules of $\mathcal{P}$ will be of the form:

$$A_1 \vee \cdots \vee A_k \leftarrow B_1 \wedge \cdots \wedge B_m \tag{3}$$

where $A_i$ and $B_j$, with $0 \leq i \leq k$ and $0 \leq j \leq m$, are literals. The answer set of this program is defined in a similar way to a program with rules of form (2). The answer set $\mathcal{M}^{\mathcal{P}}$ of $\mathcal{P}$ is the smallest subset of $Lit$ such that:

- $\mathcal{M}^{\mathcal{P}}$ is a classical model of $\mathcal{P}$;
- for each rule $r$, if $body(r) \subseteq \mathcal{M}^{\mathcal{P}}$ then for some $1 \leq i \leq k$ $A_i \in \mathcal{M}^{\mathcal{P}}$, i.e. our body justifies one or more literals in head's rule.

Finally, we define answer sets for extended rules with disjunction in the head. For an arbitrary program $\mathcal{P}$ that contains such rules a set of ground literals $\mathcal{R}$ is an answer set of $\mathcal{P}$, if $\mathcal{M}^{\mathcal{P}^{\mathcal{R}}} = \mathcal{R}$ where $\mathcal{M}^{\mathcal{P}^{\mathcal{R}}}$ is the answer set as defined above for the reduct of $\mathcal{P}$ with respect to $\mathcal{R}$.

## 3 The DualHEX agent

### 3.1 Framework

The framework used in our project is the combination of a basic framework given by the organizators of AI competition with extensions that were implemented by the participants of the AI competition 2013 [4]. The basic framework interacts with the game while it is running on Google Chrome with Angry Birds' Extension. The framework's Game Server component processes this interaction. This reciprocity is implemented by the Proxy module, which consists of four commands: CLICK, DRAG, MOUSEWHEEL, SCREENSHOT.

The Vision Module divides the image, that is displayed to the user into small parts and recognizes the bounding box of every object. The bounding objects can be: all types of birds, different types of shelters, pigs and the slingshot. The AngryHEX team implemented an extension in the Vision module that improved the orientation recognition of the block.

Trajectory Module is needed to plan the shot. As soon as we have a target point, this module can be used to evaluate the trajectory that the bird will follow from the exact point. In the framework provided by the organizers of the competition, this module aims only at the center of the target object. Therefore, the AngryHEX team extended it with the ability of guiding the trajectory to the top or left side of the object.

The Server and Client Communications Ports allow the interaction between Game Server and Game Client. They receive commands from the server and after executing them send a feedback. In our implementation the AI Agent represents the DualHEX agent which is an extension of AngryHEX agent. AngryHEX added an Executor to the original Framewok that encodes the information about the environment into logic assertions and then runs the DLVHEX solver to compute a list of good shots based on that information and on the knowledge base. DLVHEX [1] is an ASP solver that computes models of HEX-programs, which are answer set programs that allow integration of external computing sources.

### 3.2 AngryHEX AI agent

Our approach is based on the AngryHEX AI agent, which makes use of Answer Set Programming (ASP) to model the internal knowledge of the game. It has two main layers: Strategy and Tactic. The Strategy layer is implemented in Java and decides which level to play next. It considers the achieved scores and keeps track of the previously selected objects, in order to exclude them and to force a different tactic every time a level is played again. The Tactic layer does the reasoning by taking information about the current scene from the vision component and returning the best shot according to the result of the HEX program, which is declaratively implemented using ASP.

Our logic program has about 300 statements represented by rules and facts encoding fixed knowledge of the domain. We would like to introduce some examples.

Facts in the program can describe different types of objects. For example, the following fact encodes all type of birds: white, red, blue, yellow and black.

$$birdType(Bird) \leftarrow$$

In the knowledge base there are three types of trajectories: low, high and egg. Trajectory egg is used when the white bird is in the slingshot, because this bird has the property to shoot with an egg bomb. Low and high trajectories are used to aim at the left and top side of an object, respectively. Therefore, facts to represent different trajectories were introduced. They respect the following schema:

$$trajectory(TrajectoryType) \leftarrow$$

Targets are represented by pigs, different kinds of material or TNT, as it is described by instances of the following fact:

$$objectType(Obj, ObjMaterialType) \leftarrow$$

Facts are also introduced to count the value of energy which can be spread from an object to another after the execution of a shot. Those are instances of the following:

$$energyLoss(Bird, ObjectType, EnergyLoss) \leftarrow$$

Besides that, we can describe that a specific bird is not good for the object stone by listing an instance of the fact:

$$nogoodForStone(Bird) \leftarrow$$

The importance of the target object to be destroyed is also estimated, in order to calculate the damage of a fall:

$$materialFallImportance(MaterialType, FallImportance) \leftarrow$$

Rules in the program can be represented as normal rules with literals in their head and body, rules containing negation as failure, constraints and weak constraints. To calculate the percentage perturbations on tap time depending on bird type we have the rule

$$tap(Perturbation) \leftarrow birdType(Bird)$$

There are three different types of damage for a target object. *pushDamage*, when it has been pushed by another object. *fallDamage*, when the target is on the top of an object which has been shot. *directDamage*, when the bird is shot directly on the target. The rules to describe this types of damage are similar to

each other, so here we will only describe how *pushDamage* is evaluated.

$$\begin{aligned} fallDamage(Obj, P\_N) \leftarrow & directDamage(RemovedObj, P\_R, E), \\ & E >= 10, \\ & P\_R > 0, \\ & ot(RemovedObj, Obj), \\ & objectType(Obj, T), \\ & materialFallImportance(T, P\_N), \\ & P\_temp = P\_R * P\_N, P = P\_temp/100. \end{aligned}$$

First of all, the damage probability of a direct shoot on the removed object and the energy left is computed. The removed object can be any object that will suffer damage according to our prediction. Then, we check whether this probability and energy are greater than zero and ten, respectively. After that, we check whether our object is on the top of the removed object. If so, the damage probability of the object which is on the top is computed by checking its type and computing the fall importance of an object with such a material type. This material fall importance is defined by facts and it's known, for instance, that the fall of an object with stone material is more important than one with ice.

All rules of damage follow the same structure. The object, we want to evaluate the damage for, is compared with the other objects from our scene, because in game's structures all objects are interacting with eact other. The damage probability of a given object is computed in the last line of the rules, where its probability of damage is multiplied by the probability of the objects related to it. However, that is not the case for the *fallDamage* rule, because, in our point of view, the target that falls doesn't influence the other objects from our scene.

### 3.3 AngryHEX extension

A suggestion of improvement for the AngryHEX agent reported by its developers in [4] was to introduce the planning of multiple shots based on the order of the birds that must be shot. We liked this suggestion and decided to work on it. In our point of view, when people try to solve a level they plan according to the available birds to shoot. One possible strategy is to use each bird to hit the material that it affects the most. As AngryHEX AI agent selects a random target from the answer set that result from the reasoning on the knowledge base and scene information, we would like to improve this approach by minimizing this answer set. This can be done by also considering the next bird to be shot. In other words, we want to remove from our solutions the targets with high probability of damage if hit by the next bird in line. However, this is advantageous only if the birds are of different type.

Our idea was to make a small change to guarantee that it will not have a big impact on the overall performance. We decided to restrict the possible targets

instead of establishing a plan, because it is hard to have a good prediction of consequences in the game scene after a shot and we would need to check if our plan is still valid after each shot. If the plan is not valid anymore we would need to compute a new plan.

In order to implement this, we need to have additional information about the current scene: the type of the next bird (first bird after the slingshot). We compare the *damageProbability* of an object for the current bird and the next one. If the next bird induces more damage on the target than the current one, the target is discarded. In other words, we want to keep the targets which are good for the next bird. This idea can be encoded by adding a rule to check if a target will suffer more damage if hit by the next bird:

$$\begin{aligned} goodForNextBird(Obj1) \leftarrow & secondBird(secondBirdType), \\ & birdType(currentBirdType), \\ & currentBirdType \neq secondBirdType, \\ & target(Obj1, Traj1), target(Obj2, Traj2), \\ & objectType(Obj1, Obj1Type), \\ & objectType(Obj2, Obj2Type), \\ & damageProbability(secondBirdType, Obj1Type, D1), \\ & damageProbability(secondBirdType, Obj2Type, D2), \\ & D1 > D2. \end{aligned}$$

And then we don't consider targets that we prove to be good for next bird by adding *not* $goodForNextBird(secondBirdType)$ to the predicate that represents the possible targets for the current bird:

$$\begin{aligned} targetData(X, Y, Z, T, 0) \leftarrow & target(X, Y), \\ & not\ goodForNextBird(X), \\ & offset(T),\ tap(Z),\ Y \neq egg \end{aligned}$$

We show above how the rule for possibe targets with trajectories that are not egg's trajectories ($Y \neq egg$) looks like after our extension.

In the original AI agent, the decision of the target is already reasonable. With our approach, we don't guarantee that the next bird will be shot to one of the removed targets. As we will play levels more than once, the probability of having a good combination of shots increases. We strongly believe that this small improvement will minimize our answer sets such that it gets closer to people choices and, thus, allow our agent to achieve good scores.

## 4 Conclusion

In this paper we presented an extension called DualHEX of AngryHEX AI agent, which performed very well in the Angry Birds Competition 2013. The DualHEX agent represents a declarative approach implemented by means of Answer Set

Programming (ASP) techniques to play Angry Birds. Our goal is to minimize the answer set by taking into consideration the damage probabitly of targets for two birds, current and next. Since the answer set is already a good solution for the current bird, this simple improvement allows to keep more advantageous solutions for the next one. This may lead to complete the levels with better scores with few impact on performance.

In order to evaluate our approach, we first started by setting up the AngryHEX agent to run. In this process, an error while running the code was identified. We then contacted its developers and they are also facing the same problem. Due to this problem, we were not able to test our extension of the AngryHEX agent. In the future, we plan to run Benchmarks to compare the efficiency and performance of its results with the results obtained by the AngryHEX agent.

## References


1. DLVHEX http://www.kr.tuwien.ac.at/research/systems/dlvhex/
2. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Commun. ACM 54(12), 92–103 (2011), http://dblp.uni-trier.de/db/journals/cacm/cacm54.html#BrewkaET11
3. Eiter, T., Ianni, G., Krennwallner, T.: Answer set programming: A primer. In: Tessaris, S., Franconi, E., Eiter, T., Gutierrez, C., Handschuh, S., Rousset, M.C., Schmidt, R.A. (eds.) Reasoning Web. Lecture Notes in Computer Science, vol. 5689, pp. 40–110. Springer (2009), http://dblp.uni-trier.de/db/conf/rweb/rweb2009.html#EiterIK09
4. Francesco Calimeri and Michael Fink and Stefano Germano and Giovambattista Ianni and Christoph Redl and Anton Wimmer: AngryHEX: an Artificial Player for Angry Birds Based on Declarative Knowledge Bases. In: Baldoni, M., Chesani, F., Mello, P., Montali, M. (eds.) Proceedings of the Workshop Popularize Artificial Intelligence co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), Turin, Italy, December 5, 2013. CEUR Workshop Proceedings, vol. 1107, pp. 29–35. CEUR-WS.org (2013), http://ceur-ws.org/Vol-1107/paper10.pdf
5. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. pp. 1070–1080. MIT Press (1988)
6. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991)
7. Renz, J., Ge, X., Gould, S., Zhang, P.: The Angry Birds AI Competition. AI Magazine 36(2), 85–87 (2015), http://www.aaai.org/ojs/index.php/aimagazine/article/view/2588