# AngryHex: an Angry Birds-playing Agent based on HEX-programs

Francesco Calimeri [1] Michael Fink [2] Stefano Germano [1]
Andreas Humenberger [2] Giovambattista Ianni [1] Christoph Redl [2]
Daria Stepanova [2] Andrea Tucci

[1] Università della Calabria, Dipartimento di Matematica e Informatica,

[2] Technische Universität Wien, Institut für Informationssysteme

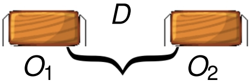
ECAI, Angry Birds Competition–August 19, 2014

# Motivation

- **Approach:** design an agent based on declarative logic programming
- **Challenge:** plan optimal shots under consideration of some physics
- **Our means:** HEX-programs, i.e. Answer Set Programs (ASP) with external sources

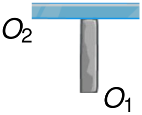
# HEX-programs

- HEX-program Π is a set of ASP rules, where external atoms are allowed in rule bodies:
  - $\&distance[O_1, O_2](D)$ is true iff distance between $O_1$ and $O_2$ is $D$
  - $\&canpush[ngobject](O_1, O_2)$ is true iff $O_1$ can push $O_2$ given additional info in the extention of *ngobject*



- $Rule_1$ estimates the likelihood that object $O_2$ falls when $O_1$ is hit

$$Rule_1: \quad pushDamage(O_2, P_1, P) \leftarrow pushDamage(O_1, \_, P_1), P_1 > 0$$



$$\&canpush[ngobject](O_1, O_2),$$

$$pushability(O_2, P_2), P = P_1 * P_2/100.$$

# Architecture of Angryhex

- We use the provided framework (browser plugin, vision module, . . . )
- Agent builds on tactics and strategy, both are realized declaratively

- **Tactics:** reasoning about the next shot is done in a HEX-program $\Pi$
  - **Input**: scene info from the vision module (facts of $\Pi$)
  - **Output**: desired target (models of $\Pi$)

- **Strategy:** next level to played is computed in an ASP program $\Pi'$
  - **Input:** info about the number of times levels were played, best scores achieved, scores of our agent (facts of $\Pi'$)
  - **Output:** next optimal level to be played (models of $\Pi'$)

# HEX Encoding for Tactics

- **Physics simulation results** are accessed via external atoms:
  - decide which $O'$ intersect with trajectory of a bird after hitting $O$
  - decide whether $O_1$ falls whenever $O_2$ falls . . .

- **Tactics in details:**
  - Consider each shootable target
  - Compute the estimated damage on each non-target object
  - Rank the targets (=answer sets) using weak constraints
  - Consider history: never play a level in the same way again!

# ASP Encoding for Strategy

- **Decides which level to play next** based on info about:
  - number of times each level was played
  - best scores
  - our agent's scores . . .

- **Strategy in details:**
  - First play each level once
  - Second play levels in which our score maximally differs from the best one
  - Third play levels in which we played best and the difference to the second best score is minimal

# Conclusion and Future Work

- **Wrap-up:**
  - Agent is realized using declarative programming means
  - Vision module provided by the organizers is integrated
  - Declarative strategy is realized (used to be in java)
  - Fixes and improvements in comparison to previous version

- **Possible improvements:**
  - Combine objects which behave like a single one
  - Plan over multiple shots
  - Improve object recognition and general precision of shots