Combining Qualitative Spatial Representation Utility Function and Decision Making Under Uncertainty on the Angry Birds Domain

Leonardo Anjoletto Ferreira[1,2], Guilherme Alberto Wachs Lopes[2], Paulo Eduardo Santos[2]
[1]Universidade Metodista de São Paulo
São Bernardo do Campo - São Paulo - Brazil
[2]Centro Universitário da FEI
São Bernardo do Campo - São Paulo - Brazil
leonardo.ferreira@metodista.br, guilhermewachs@gmail.com, psantos@fei.edu.br

Abstract

Angry Birds is a well known game in which a player must shoot birds in order to kill pigs. In 2012, Australia National University released a framework that allows the developement of autonmous agents that are able to play the Google Chrome's version of Angry Birds. This paper presents the development of an autonomous agent for playing Angry Birds using concepts of qualitative spatial representation, utility function and decision making under uncertainty. The agent had a good overall performance during testing and in the levels that it completed in the benchmark, but it also had problems when dealing with the configurations of some levels.

1 Introduction

The Angry Birds game has been well known for some time and has had a great success in the game market. The game consists of shooting birds with a slingshot in pigs or structures in order to destroy the structure and kill the pigs. The fewer birds used and the more objects destroyed, the highest is the score of the player.

In 2012, the Australia National University (ANU) began to develop a framework that allows for the development of Angry Birds playing agents and by the end of the same year ANU held the first competition of autonomous Angry Birds playing agents.

The framework provided by the ANU uses the Chrome browser's version of Angry Birds with a plug-in to perform commands, whereby the client and server softwares were developed in Java. This framework allows for the change in both server and clients codes and thus the development of new agents for the competition.

As informed in the documentation provided by the ANU, the framework works in a client-server mode in a way that the server interacts with the Chrome's plug-in to send commands to perform the shooting of the birds and tapping and also to take the screenshot of the game, so that a vision system can recognize the objects in the scenario.

The client software is responsible to choose the coordinates for the shooting of birds and the time until the tapping. The agent provided as an example, called `NaiveAgent` aims directly at the pigs, with no regards to the objects around it.

The challenge is to develop a client side of the application capable of (at least) outperforming the `NaiveAgent`. In other words, the goal is to develop an autonomous agent that is able to play without human intervention and that would consider the environment in which it is immersed at a given moment and to choose the best object to use as a target.

This paper presents one of the agents developed at Centro Universitário da FEI for the 2012 competition, which had the main focus in choosing the best target to use. To develop the FEI2 agent, three concepts were combined. First, qualitative spatial representation was used, since the position of the objects and the relation between them is important in the Angry Birds game. The second concept is that of Utility Function, which allows the agent to represent preferences for the options that are given to it. Lastly, decision making under uncertainty was also used in the development of the agent, because it was perceived that there was some imprecision in the launching of the birds and that imprecision would make the bird hit a completely different target – or no target at all – in some situations.

The combination of these three concepts is discussed along the following sections and the equations used to develop the agent are also presented and discussed.

This paper is organized as follows: the next section presents the foundation of qualitative spatial representation, followed in Section 3 by a description of Utility Function. Section 4 presents a formalization for an agent to make decision under uncertainty. Next, in Section 5, the development of the FEI2 agent and how the concepts were combined is presented and in Section 6 the performance of the agent is evaluated. Finally, Section 7 concludes this work.

2 Qualitative Spatial Representation

The objective of a Qualitative Spatial Reasoning (QSR) system is to symbolically represent objects in a space and provide a system of inference for those objects [Santos, 2010], with the main challenge being to provide the calculi that allows the representation and reasoning without using quantitative methods [Cohn and Renz, 2008].

One of the assumptions of the qualitative representation is the relationship between objects. A relation R in the domain $\mathcal{W}$ can be formally represented by the set of tuples $(w_1, w_2, \dots, w_k)$ of the same arity k, with $\forall w_i \in \mathcal{W}$.

Considering a set of n relations $\mathcal{R} = \{R_1, R_2, \dots, R_n\}$ existing in a domain $\mathcal{W}$, it is possible to obtain an algebra of relations using algebraic operations.

In order to perform QSR, there is the need to represent objects in space. Although there is a large number of representation methods [Ligozat, 2011], this paper focus only on direction and orientation representation, because those were the ones used in the development of the agent. Future work shall consider other modes of spatial representation.

Fundamentally, a direction relation can be defined by three components: an object, a reference and a reference frame. Some representations use cardinal directions (North, South, East and West) while others – such as this work, for example – use a special case of direction calculus, called Orientation Calculi, which assumes that the objects in the space have an *instrinsic front*.

In the Angry Birds domain for example, we use the relation between objects that can be used as targets, such as pigs, woods and stones and by doing so we have relations such as `above(pig, wood)` to know if the pig is above the wood and `leftOf(stone, pig)` that returns if the stone is in the left of the pig, for example.

3 Utility Function

An utility function is used to map states in the environment to real numbers in a way that it can represent the preferences of an agent regarding the states. The notation used to describe this preference relation is [Russell and Norvig, 2004]:

- $A \prec B$: B is prefered than A;
- $A \sim B$: the agent is indifferent between A and B;
- $A \preceq B$: B is prefered than A or is indifferent among them.

In order to constrain the utility function, so that the agent cannot behave irrationally, six axioms can be used to provide the proof, as demonstrated by [Russell and Norvig, 2004]. This proof will not be made here as it is out of the scope of this article.

From these axioms, [Russell and Norvig, 2004] define Utility as the function that maps a set of states S to a real value U, so that an agent can prefer a state to another with higher utility.

Finally, the principle of Maximum Expected Utility (MEU) is defined as:

$$U([p_1, S_1; p_2, S_2; \dots; p_n, S_n]) = \sum_{i=1}^{n} p_i \times U(S_i) \tag{1}$$

for all states $S_i \in S$ that has respective probabilities p_i of occurring.

Once we know how to calculate the utility of a state, it is possible to use this information to decide which action to take. The next section presents how to make those decisions when there is uncertainty involved.

4 Decision Making Under Uncertainty

Hardly ever an agent in the real world has access to all information available in the environment, being it by the inability to read it with its sensors or by the complexity of the world itself, but even in situations of uncertainty, an intelligent agent must decide what action should it make.

A simple method to formalize a decision problem is using the tuple $\langle \mathcal{D}, \mathcal{N}, \mathcal{O} \rangle$, a function $f : \mathcal{D} \times \mathcal{N} \mapsto \mathcal{O}$ and the preference relation $\preceq$ [Bertsekas, 2005], in which:

- $\mathcal{D}$: is the set of possible decisions that the agent can take at any given time step;
- $\mathcal{N}$: is the set of "states of nature" that gives the indexes of uncertainty in the problem;
- $\mathcal{O}$: is the set of all possible outcomes given a decision $d \in \mathcal{D}$ and a state $n \in \mathcal{N}$ chosen by the agent;
- $f : \mathcal{D} \times \mathcal{N} \mapsto \mathcal{O}$: is the function that maps the possible outcome $o \in \mathcal{O}$ given the decision $d \in \mathcal{D}$ and the state $n \in \mathcal{N}$, so that $\forall d \in \mathcal{D}$, $\forall n \in \mathcal{N}$, $\exists o \in \mathcal{O} \mid f(d, n) \mapsto o$
- $\preceq$: is the preference relation between possible outcomes $o \in \mathcal{O}$, so that $o_1 \preceq o_2$ indicates that o_2 is at least as preferable for the agent as o_1;

It is possible to use a utility function $U(\mathcal{O})$ to map the preferences of the set $\mathcal{O}$ to a real number ($U : \mathcal{O} \mapsto \mathbb{R}$), so that we can say that given two options $o_1, o_2 \in \mathcal{O}$, if $o_1 \preceq o_2$, then the utility $U(o_1) \leq U(o_2)$.

Considering that $\mathcal{O}_1 = f(d_1, n), \forall n \in \mathcal{N}$ and $\mathcal{O}_2 = f(d_2, n), \forall n \in \mathcal{N}$ we can also say that if $U(\mathcal{O}_1) \leq U(\mathcal{O}_2)$, then $d_1 \preceq d_2$ and, using the maximum expected utility principle, in a situation in which the agent must choose between the decisions d_1 and d_2, it should choose d_2, since it is at least as preferable as d_1 for every possible outcome and maximizes the expected utility [Russell and Norvig, 2004].

Another form to write this relation is by using the equations [Bertsekas, 2005]:

$$d_1 \preceq d_2, \text{ iff } \begin{cases} U(d_1) \leq U(d_2), \forall n \in \mathcal{N} \text{ and} \\ f(d_1, n) \preceq f(d_2, n), \forall n \in \mathcal{N}; \end{cases} \tag{2}$$

Given all possible decisions $d \in \mathcal{D}$ that the agent can choose, the one that it must select in order to have the best outcome is called the dominant decision $d^* \in \mathcal{D}$, so that $d \preceq d^*, \forall d \in \mathcal{D}$. Nevertheless, it is possible that for a given problem there is no dominant solution, but there are partially dominant solutions, which is a set of solutions that dominate one another in one dimension but not in another.

In this case, a partially dominant solution set $\mathcal{D}_m$ formed only by the solutions d_m that partially dominate any other is considered. This set is formed so that $\forall d_m \in \mathcal{D}_m$ there is no $d \in \mathcal{D}$ that has $U(d_m, n) \leq U(d, n)$, $\forall n \in \mathcal{N}$ and $U(d_m, n) \leq U(d, n)$ for any $n \in \mathcal{N}$.

For problems in which there is uncertainty in the outcome, we assume that the uncertainty n follows a given probability distribution $P(\cdot|d)$ defined under $\mathcal{N}$. From this probability, we can write the probability $P_d(o)$ of occurring the outcome $o \in \mathcal{O}$ for each decision $d \in \mathcal{D}$ using the function $f(d, \cdot)$ and the relation

$$P_d(o) = P(\{n | f(d, n) = o\} | d), \forall o \in \mathcal{O} \tag{3}$$

As explained by [Bertsekas, 2005], from this equation it is possible to use the principle of expected utility so that the agent can choose between two decisions $d_1, d_2 \in \mathcal{D}$ that have the same outcome $o \in \mathcal{O}$. If the agent knows the probabilities $P_{d_1}(o)$ and $P_{d_2}(o)$, the decision can be made using the relation $d_1 \preceq d_2$ if and only if $P_{d_1} \preceq P_{d_2}$.

The next section presents how the principles described above were applied on the developement of our Angry Birds agent.

5 Developing the FEI2 Agent

To develop the FEI2 agent, we proposed a combination of utility function and qualitative spatial representation for objects from the Angry Birds levels that the vision system can recognize. Along with this combination, decision making under uncertainty was used when dealing with the process of choosing the lauching coordinates of birds.

Since at every time the agent has to make a decision, a screenshot is taken, the FEI2 agent uses the position and relation of elements (mainly pigs, stones, ice and wood) to calculate utilities and find the element with the highest utility and probability to be used as target.

In the following subsections, we describe in details the use of each of the methods described above.

5.1 Utility function with Qualitative Spatial Reasoning

To evaluate the objects that could be used as targets in the current scenario, the agent calculates an utility value for some of the objects found in the image segmentation process. We define the following three attributes to compose the utility function:

1. Pigs' utility (equation 4);
2. Destructible Objects: wood (equation 5) and ice (equation 6);
3. Indestructible Objects: rocks (equation 7). It is important to observe that, although some rocks can be destroyed, we consider only red birds in our agent and this kind of birds needs more than one hit to break a rock. Thus, we considered every rock as an indestructible object.

Although only four objects were used as possible targets, the vision system can recognize some others, such as indestructible wood, floor and birds. Those objects were not considered as possible targets because birds are the shooting objects, the floor in most cases can only deviate the trajectory of the bird and as such, can not help it to achieve its goal. Finally, the indestructible wood is usually recognized in the slingshot used for the birds and some constructions of the scenario, and as the floor, it can only deviate the bird's trajectory.

To implement the methods that describe the relation between two objects recognized in the scenario, we used the information provided by the `BoundingBox` method already provided for the objects. The methods developed to described the relations between an object of reference (`ref`) and a possible target (`obj`) are:

- `above(obj, ref)`: `obj` completely covers the superior side of `ref`;
- `below(obj, ref)`: `ref` completely covers the superior side of `obj`;
- `left(obj, ref)`: `obj` completely covers the left side of `ref`;
- `right(obj, ref)`: `ref` completely covers the left side of `obj`;
- `isAbove(obj, ref)`: `obj` is at some point above `ref`, but not necessarily directly above `ref`;
- `isBelow(obj, ref)`: `obj` is somewhere below `ref`, but not necessarily right below;
- `isLeft(obj, ref)`: `obj` is at the left of `ref`, independently of the vertical relations;
- `isRight(obj, ref)`:`obj` is at the right of `ref`, independently of the vertical relations;
- `diagAboveLeft(obj, ref)`: `isAbove(obj, ref)` ∧ `isLeft(obj, ref)`. `obj` is at the same time above of `ref` and at the left of it;
- `diagAboveRight(obj, ref)`: `isAbove(obj, ref)` ∧ `isRight(obj, ref)`. Analogously, `obj` is at the right and above of `ref`;
- `diagBelowLeft(obj, ref)`: `isBelow(obj, ref)` ∧ `isLeft(obj, ref)`. `obj` is at the left of `ref` and at the same time below it;
- `diagBelowRight(obj, ref)`: `isBelow(obj, ref)` ∧ `isRight(obj, ref)`. `obj` is below of `ref` and at the left of it;

As described in the beginning of this section, each possible target has its own function to calculate the utility. For the pigs, it is considered its relation with every other destructible and indestructible object in the scenario and also with every other pig. In this case, for every object that tries to avoid the collision of the bird with the pig, a value is discounted based on the type of object and its relation to the pig. The function used to calculate the pig's utility is:

$$
\begin{aligned}
U(pig) = & \sum_{d \in \text{Destructables}} -2 \times above(d, pig) - left(d, pig) \\
& + right(d, pig) - 2 \times diagAboveLeft(d, pig) \\
& + 2 \times diagBelowLeft(d, pig) \\
& + diagBelowRight(d, pig) \\
& + \sum_{u \in \text{Undestructables}} -2 \times above(i, pig) \\
& - 2 \times below(i, pig) + right(i, pig) \\
& - 2 \times diagAboveLeft(i, pig) \\
& + 2 \times diagBelowLeft(i, pig) \\
& + diagBelowRight(i, pig) \\
& + \sum_{p \in Pigs} diagAboveRight(pig, p) \qquad (4)
\end{aligned}
$$

As it is possible to notice in equation 4, when considering destructible objects we subtract a value when the object hinders the possibility of hitting the pig and add values when the object does not interfere with the trajectory of the bird. For indestructible objects, the process follows the same principles, but with slightly different weights.

It is important to notice that the weights for every relation was obtained by empirical evaluation. Considering that we receive $+1.0$ when the spatial relation is true and 0.0 when false, the initial proposal used those values directly, but as we could perceive by the choice of target, some relations and properties influenced more in the trajectory of the bird and completing the level than others. Thus, the main change was doubling the weights that provided a better overall solution during development.

Regarding the destructible objects, the agent considers two types of objects: wood and ice. Although both can be broken when the bird hits it, they may deviate the trajectory of the bird depending on the angle that the bird hits it. Thus, when developing the utility function of destructible objects, we considered when the collision of the bird with the object can help to indirectly kill the pigs (`diagBelowLeft` and `diagBelowRight`), when the change of trajectory provided by the collision may help hitting another bird (`right` and `diagAboveRight`) and when the collision will not help the bird (`above`, `left` and `diagAboveLeft`).

The final equation, used for objects that can be destroyed (d), compares the position of those objects with the centroid calculated from the current position of every pig left in the scenario (cp) and was defined as:

$$\begin{aligned}U(d) = &-2 \times \mathit{above}(d, cp) - \mathit{left}(d, cp) + \mathit{right}(d, cp) \\ &- 2 \times \mathit{diagAboveLeft}(d, cp) + \mathit{diagAboveRight}(d, cp) \\ &+ \mathit{diagBelowLeft}(d, cp) + \mathit{diagBelowRight}(d, cp) \quad (5)\end{aligned}$$

Equation 5 above describes the utility used for wood. As with the pig's utility, the weights of this equation was obtained empirically. Since destructible objects when destroyed may slow down the bird – and by doing so when the bird hits the pig, it may not kill the pig, but only harm it – when objects are in the left or above the centroid of the pigs, they receive a negative weight. In other cases, when the diversion caused by the collision with the object may help the bird hit another pig, the spatial relation has a positive weight.

Initially, we expected to use a single equation for every destructible object available in the game, but since ice is easier to be destroyed than wood and in various scenarios, ice is presented as a key object to be destroyed, in order to finish the level using as fewer birds as possible, the equation used for this object was different than the equation used for wood. The equation used to calculate the utility of an ice i in relation to the centroid of all pigs cp and other ice blocks $i2$ is:

$$\begin{aligned}U(i) = &\ 1 - 2 \times \mathit{above}(i, cp) - \mathit{left}(i, cp) + \mathit{right}(i, cp) \\ &- 2 \times \mathit{diagAboveLeft}(i, cp) + \mathit{diagAboveRight}(i, cp) \\ &+ \mathit{diagBelowLeft}(i, cp) + \mathit{diagBelowRight}(i, cp) \\ &+ \sum_{i2 \in Ice} (\mathit{below}(i2, i) \wedge \mathit{left}(i2, i)) + \mathit{isLeft}(i2, i) \quad (6)\end{aligned}$$

As with the previous utility functions, the weights for the equation 6 were obtained empirically. The main difference between equations 5 and 6 is the bias added in the beginning of the equation, so that an ice may have a greater utility than other objects, and the relations between every ice block is also considered in this equation. This was considered during development for the cases of a sequence of ice block, so that the agent may choose as target the block in the lowest position, that when broken may cause a chain of destruction of other ice blocks.

The last type of object that the vision system can recognize and that was used to calculate an utility function is the rock, that has a different function since it is an indestructible object.

Analogously, the utility of every rock r in the scene is calculated using its relation with regard to the centroid of the pigs cp. Nevertheless, since rock is an indestructible object, it was only considered to deviate the trajectory (`above`, `left`, `diagAboveLeft` and `diagBelowRight`), it can be used to change the trajectory (`right` and `diagAboveRight`) or it can be moved (`diagAboveLeft`). The final equation for the utility of a rock r is calculated using:

$$\begin{aligned}U(r) = &-2 \times \mathit{above}(r, cp) - 2 \times \mathit{left}(r, cp) + \mathit{right}(r, cp) \quad (7) \\ &+ \mathit{diagAboveLeft}(r, cp) - 2 \times \mathit{diagAboveRight}(r, cp) \\ &- 2 \times \mathit{diagBelowLeft}(r, cp) - 2 \times \mathit{diagBelowRight}(r, cp)\end{aligned}$$

As with the previous equations, the weights of equation 7 have been empirically evaluated. It is possible to notice a similarity between equations 7 and 5, being the difference the weights of the relations `left` and `diagBelowLeft` – since the rock can not be destroyed, but only moved – and the `diagAboveRight` and `diagBelowRight`, because in a destructible object, it can be destroyed and maybe it will slow down the bird, but in the case of the rock, it will stop the bird or deviate it from the original trajectory.

Using these equations, the agent can calculate the utility of each object that it can use as a target, but there is still the uncertainty regarding the chance of the bird hitting the target. The next subsection explains how the probability was used when making a decision to where to shoot.

5.2 Decision Making with Probability

The vision system provides the agent with the position of the objects regarding the information obtained from the last screenshot taken by the framework. There is no uncertainty in the position of the objects and, as noted in the documentation, the release point chosen for a given target is calculated based on the learning of a series of data already obtained, provided and used by the framework, which is also without any uncertainty.

However, as noted when developing the agent in a configuration that is different from the one used in the competition, there is a chance of missing the selected target when launching the bird. In some shoots the bird was able to hit precisely the selected coordinate, but in other cases, the bird had a completely different trajectory and hit another object. Thus, the concepts of probability and uncertainty were used to determined the chance of a bird to hit a given target.

To calculate these probabilities, a class was developed in which, given the x and y coordinates of the launching point of the bird and the target, it calculates possible trajectories. Once the agent has a set of calculated trajectories, it chooses the launch point that hits the highest quantity of targets in a predefined range (so that only some of the highest utility objects are considered and not all of possible targets).

Now that the concepts used to develop the decision making for the Angry Birds agent has been presented, the next section presents the implementation of the concepts, along the environment that was used.

5.3 Implementation

Our agent was implemented using version 1.2s of the framework provided by the Australia National University (ANU) for the 2012 Angry Birds competition. This version of the framework already uses the client-server method, but still needs Mathwork's Matlab to segment the screenshot from the game. `NaiveAgentClient` provides an example of interaction with the server with level selection, restart, which also selects the coordinates for shooting the bird when given the coordinate of a target.

The focus of the development of the agent was in the decision making and as such, most of the interaction with the server was derived from the `NaiveAgentClient` that used the `NaiveSolver` class to choose the object to use as target. Initially, the `NaiveAgentClient` class was duplicated and named as `FeiAgentClient`[1], but instead of using the `NaiveSolver` class to choose the target, it instantiated the `FeiSolver` class and used it to select the next target using Utility Function as described in Section 3.

Along with the `FeiSolver` class, some other classes and methods were implemented so that it facilitated the usage of the concepts described in the previous sections. The three developed classes were:

- `FeiUtility`: the class that provides the methods to calculate the utility for each object considered as possible target. In the case of this implementation, only the pigs, destructible objects (wood and ice) and indestructible objects (rock) were considered.
- `Positions`: class that implements the methods that returns the possible relations between an object and a reference, as described in Section 5.1. Twelve methods were implemented, but not all of them were directly used in the utility function.
- `Trajectory`: a class that calculates possible trajectories for the birds, given a target's coordinates. It considers that the trajectory is always a parabola.

The `FeiUtility.java` class is responsible for the implementation of the utility functions and as such provides the methods `destructiblesU`, `indestructibleU`, `pigsU` which are responsible to calculate the utility of the destructible objects (wood and ice) indestructible objects (rock) and the pigs that exists in the level, respectively. Those methods calculate the utility for each of its respective objects and then returns a linked list that contains every value with its object. Those linked lists are them used by the `utility` method implemented in the same class. This method takes every list and combines them into a single list.

[1]The class received some minor changes during the development, but it kept most of its original code

Finally, the method `getUtilities`, also implemented in the same class, receives lists of objects divided by types and uses the other methods to calculate the utility. Once it has the list of utilities of every object, it returns a liked list ordinated by the utility value of every possible target.

The possible trajectory of the birds were developete in the `Trajectory.java` class and the method `computeFlight`, which receives a target object calculates possible launch coordinates and returns only the ones that can hit the desired target.

The `Positions.java` class is the responsible to implement the relations presented in Section 5.1. This class is used mostly by `FeiUtility` to calculate the utility of each object.

The `FeiSolver` class uses these three classes to choose a target from the list of objects (pigs, wood, ice or rock). It receives the objects from the vision system and then calculates the utility for all of them. Next, it traces the possible trajectories only for the five objects with the highest utility and chooses as target the one that it has found the largest amount of launch points that hits the object. After it chooses the target, the agent returns the selected object to the `FeiAgentClient`, so that this class would perform the shooting process.

The complete code of the agent is available at https://bitbucket.org/anjoletto/ab2012-fei2.

Now that we presented how the FEI2 agent was implemented, it is possible to verify the performance of the agent during the game.

6 Performance in the game

There were two performance evaluation for the FEI2 agent. The first one was made during the development of the agent and the second one was presented by the ANU after the competition ended.

6.1 Performance During Development

During development, a great variation in the trajectory of the bird was observed. This resulted in the development of the `Trajectory.java` class to obtain the probability of hitting a given target.

Since we could not compare the results of the other agents in the competition, the evaluation of the agent was based mostly in the number of stars that the agent got and how many birds it needed to solve the level. The final evaluation was performed in the first fifteen levels of the Poached Eggs levels.

In the first and second levels, the FEI2 agent received one or two stars, depending on the error in the trajectory. In most of the next levels the agent could get the three available stars using only one or two birds to solve the problem. In those levels it was noticed that the second bird onwards was used mostly when there were too many pigs in the level or a great number of objects around the pigs.

It was also noticed that in some ocasions the agent did not complete the level or received only one star. Those usually happened when there were too many objects around the

Level	ABC-AI-UNICAL	ABC-IS-UNICAL	CU	Dan	FEI2	Naive Agent	RedBacks	Zonino	3 stars
2	52420	34600	52590	42960	**60880**	43440	44080	52560	60000
3	33460	41070	**42880**	32880	21070	41350	23480	40800	41000
5	36280	62780	52740	55560	66050	54350	**67630**	63230	64000
6	17870	17500	19270	23630	**32550**	24510	24530	17900	35000
8	47400	40440	47780	47180	34190	**48270**	47920	46820	50000

Table 1: Benchmark performed by the ANU with the agents of the 2012 competition (available in the AIBirds.org website). The table presents only the levels that the FEI2 agent has finished during the benchmarking.

pigs and the birds could not harm them or even the structure around them or when there was an error in the trajectory, which made the bird miss the target.

When comparing with the NaiveAgent, FEI2 obtained a good performance. But in the environment of the competition, the performance was completetly diferent.

6.2 ANU Benchmark

A benchmark test with all the agents that participated in the 2012 competition was performed by the ANU after the event and the results are presented in the AIBirds website. In the benchmark, FEI2 completed only five of the levels used in the competition (these are presented in table 1), the other levels it did not complete because it used all the available birds and could not finish the level.

As informed in the AIBirds website, in the benchmark test the agent had a limit of thirty minutes to solve each of the twenty one levels. When the agent was not able to do so, the agent was forced to proceed to the next level.

Although the agent could not complete every level, from the five levels it completed it had the highest score for two levels (level 2 and 6) and the third highest in the fifth level. Nevertheless, on the other two levels that it completed, our agent had the lowest score from all the agents.

However, improvements in the agent are necessary so that it can deal with other levels that it could not complete and the possible problems in the trajectory that were not considered in the screen resolution used during development.

We believe that the main cause of the difference in performance between development and benchmark is the difference in the game's configuration, mainly the screen resolution. Since the documentation of the framework does not mention the possibility of imprecision in trajectories, but it was observed during the development.

7 Conclusion

This paper presented the main concept behind the development of the FEI2 agent. Although its interaction with the server was mostly based from the `NaiveAgentClient` example agent provided with the framework, the target choosing method was developed using different concepts such as utility function, decision making under uncertainty and qualitative spatial representation.

In the benchmark test made by the ANU, the agent had a good performance in the five levels that it completed, having the highest scores in two of them. Nevertheless, this is not a good performance overall, since the agent could not complete sixteen out of the twenty one levels.

Future works are initially focused in setting a testing environment as close as possible to the one that will be found in the competition.

With the environment set, changes in the weights of the spatial relations of the utility functions might be performed or even the function itself may be redesigned to accommodate unexpected events or combination of objects.

Another possible change in the utility function is the usage of other representations along the positions, such as sizes and directions, so that it is possible to have a better description of the problem that needs to be solved and, as a result, a better utility function.

For the trajectory probability, the number of objects that the agent calculates the trajectories might change. So that instead of taking a fixed number of objects, it takes a fraction of the objects in the scenario to perform the calculation.

Acknowledgments

Leonardo Anjoletto Ferreira acknowledges support from CAPES. Paulo E. Santos acknowledges support from FAPESP grant 2012/04089-3 and CNPq, grant PQ2 - 303331/2011-9.

References

[Bertsekas, 2005] D. P. Bertsekas. *Dynamic programming and Optimal Control*, volume 1. Prentice-Hall, Inc. Upper Saddle River, MT, USA, 3 edition, 2005.

[Cohn and Renz, 2008] A. G. Cohn and J. Renz. Qualitative spatial representation and reasoning. *Foundations of Artificial Intelligence*, 3:551–596, 2008.

[Ligozat, 2011] G. Ligozat. *Qualitative Spatial and Temporal Reasoning*. Wiley-ISTE, 2011.

[Russell and Norvig, 2004] S. J. Russell and P. Norvig. *Artificial Intelligence*. Pearson Education India, NJ, USA, 2 edition, 2004.

[Santos, 2010] P. E. Santos. Spatial reasoning and perception: a logic-based approach. *Tutoriais do XVIII Congresso Brasileiro de Automática*, 2010.