Deliberately Planning and Acting for Angry Birds with Refinement Methods

Ruofei Du, Zebao Gao, Zheng Xu*
University of Maryland, College Park
Computer Science Department
{ruofei, gaozebao, xuzh} @cs.umd.edu

Abstract

Angry Birds has been a popular game throughout the world since 2009. The goal of the game is to destroy all the pigs and as many obstacles as possible using a limited number of birds. Since the game environment is subject to change tremendously after each shot, a deterministic planning model is very likely to fail. In this paper, we integrate deliberately planning and acting for Angry Birds with refinement methods. Specifically, we design a refinement acting engine (RAE) based on ARP-interleave with Sequential Refinement Planning Engine (SeRPE). In addition, we implement greedy algorithm, Depth First Forward Search (DFFS) and A^* algorithm to perform the actor's deliberation functions. Eventually, we evaluate our agent to solve the web version of Angry Birds in Chrome using the client-server platform provided by the IJCAI 2015 AI Birds Competition. In our experiments, we find out that our agent using SeRPE with A^* algorithm greatly outperforms the agent using greedy algorithm or forward search without SeRPE. In this way, we prove the significance of refinement methods for planning in practice. Please see the supplementary video `https://youtu.be/u7XJ0g6d9po` for more results.

1 Introduction

Angry Birds[1] has attracted over 200 million players all over the world since 2009. The task for the player is to shoot birds with different properties from a slingshot at a structure that houses pigs and to destroy all the pigs. [AI birds, 2015] The structure can be very complicated and can involve a number of different object categories with different properties, such as wood, ice and stone, as shown in Fig. 1. After each shot, the entire game environment is subject to tremendous change. Consequently, for human players, this game is very challenging for rookies to achieve a very high score while for automatic AI agent, a deterministic planning model is very likely to fail.

Figure 1: The example image shows the complicated structure involving different objects with different properties. The bounding boxes are extracted by basic code provided.

In this project, we develop a deliberately planning and acting system that is able to successfully play the game autonomously and without human intervention. (See video at `https://youtu.be/u7XJ0g6d9po`). This may require analyzing the structure of the objects and to infer how to shoot the birds in order to destroy the pigs and to score most points by causing more damage. In order to simplify the problem and focus on the AI planning algorithms, we use the basic game playing software provided by the IJCAI competition organizer.[2] The basic framework provides a computer vision component to analyze the game and identifies the location of all the objects (observor) and an acting component that actually shoot birds in the browser (actor).

Based on the framework, we integrate both planning and acting with refinement methods using Sequential Refinement Planning Engine (SeRPE). Finally, we evaluate our agent to solve the web version of Angry Birds in Chrome.[3] Moreover, we design the greedy algorithm, Depth First Forward Search (DFFS) and A^* algorithm to perform the actor's deliberation functions. Our contribution are as follows:

1. design and implementation of greedy algorithm, DFFS and A^* algorithm for Angry Birds
2. design a refinement acting engine (RAE) based on ARP-interleave with Sequential Refinement Planning Engine (SeRPE) for Angry Birds
3. evaluation and comparison amongst different algorithms with or without SeRPE

*Sorted by alphabetical order, with equal contribution

[1] `https://www.angrybirds.com/`

[2] `http://ijcai-15.org/index.php/angry-birds-competition`

[3] `http://chrome.angrybirds.com`

4. illustration of the significance of refinement methods in practice and lessons learnt from planning and acting for Angry Birds

2 Background and Related Work

In this section, we introduce the background of Angry Bird, the IJCAI Angry Bird competition and related work regarding the problem.

2.1 Angry Bird

The game Angry Birds is popular throughout the world due to its simple rules and the behaviour of the game environment reflecting the laws of physics. The main aim in each level is, given a certain number of birds, to kill all the pigs, doing as much damage to the surrounding objects as possible. Usually the pigs are not exposed to direct shots - in these situations one first has to get through sheltering structures protecting the pig.

2.2 Angry Birds AI Competition

This competition has been held through 2012 to 2014. Each year in IJCAI Symposium on AI in Angry Birds, researchers present their original scientific work related to computer vision, heuristic search, knowledge representation and reasoning, AI planning, and machine learning in the context of Angry Birds. However, the rules differ from year to year. The latest software is developed by [Ge *et al.*, 2014]. This community also provides an online forum[4]. We can find accepted papers in year 2013 [AI birds, 2013] and year 2014 [AI birds, 2014] as references. There exists a framework for competitors to focus on the AI planning part of the agent. The framework includes the following components:

1. a computer vision component that can analyze a video game frame and identifies the location, category and bounding box of all relevant
2. a trajectory component that calculates trajectories of birds and computes where to shoot from in order to hit a given location
3. a game playing component that executes actions and captures screen shots

2.3 AI Planning for Angry Birds

In 2013, [Ge and Renz, 2013] present the first version of AIBirds framework as well as their agent in IJCAI. They develop and analyze a qualitative spatial representation for General Solid Rectangles (GSR), which assists the agent to build relations between GSR thus estimating the statbility of the physical structures in the game of Angry Birds. [Ferreira *et al.*, 2013] present an automatic framework to play Angry Birds using concepts of qualitative spatial representation, utility function and decision making under uncertainty. [Calimeri *et al.*, 2013] models its internal knowledge of the game by means of an Answer Set Programming (ASP) knowledge base [Shah, 2004]. [Zhang and Renz, 2014] propose to compute a heuristic value for each block that corresponds to how useful it is to hit that block. The heuristic value is derived from a qualitative spatial calculus for representing and analysing the structure. [Wa, 2014] formulate the problem using qualitative representation for touch, positional direction, shelters and physical rules. [Polceanu, 2014] propose a generic framework which allows an agent to reason and perform actions using multiple simulations of automatically created or externally inputted models of the perceived environment.

[4] http://forum.aibirds.org/

2.4 Machine Learning for Angry Birds

[Narayan-chen *et al.*, 2013] present their agent by using Weighted Majority algorithm and Naive Bayesian Networks to learn how to judge possible shots. [Jutzeler *et al.*, 2013] focus on a particular issue in AI Birds competition: different strategies are suitable for different levels. There are trade-off between different strategies which can be models as Multi-armed Bandit (MAB) in machine learning. Their proposed agent tries to select the best strategy for the next level. [Tziortziotis *et al.*, 2014] propose the assumption that each type of object material and bird pair has its own Bayesian linear regression model. In this way, they design a multi-model regression framework that simultaneously calculates the conditional expectations of several objects and makes a target decision through an ensemble of regression models.

3 Angry Birds Platform

We utilize the Basic Game Playing Software [Ge *et al.*, 2014] as our platform. The platform is built on Java and the game is executing in Chrome. The structure of the platform is shown in Fig. 2.

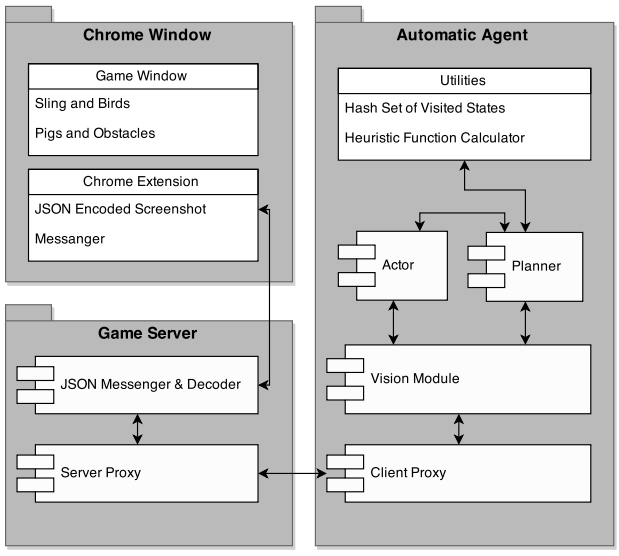

Figure 2: The executing platform for the Angry Birds Game: the server-client architecture

The main focus is to design the client of the system: the Automatic Agent. The AI Agent could use the built-in Vision Module to analyze game scenarios and use the trajectory

module (Actor) to perform shots. Server Proxy receives command messages from agents and send them to server, as well as provides feedback from the Chrome Window after the execution of the action. The Game Server communicates with Angry Birds Extension in Chrome through Extension Messenger. The Angry Birds Extension offers functionality of capturing screenshots and executing actions.

The Vision Module contains image segmentation components which could output a list of the minimum bounding rectangles (MBR) of essential objects, which takes about 100 ms for a typical scenario, an example image is shown in Fig. 1. The Actor Module estimates the trajectory that a bird will follow given a particular release point.

4 Problem Formalism

The angry bird task involves both planning and acting. Our approach is mainly based on the refinement methods from Chapter 3 of [Ghallab *et al.*, 2015]. We use the Server-Client Architecture introduced in Section 3 to command and execute actions. We use the Vision Module to monitor the current status in Angry Birds game. We design the Refinement Acting Engine (RAE) based on ARP-interleave, which replans after each action, and use SeRPE as the simulation algorithm in each replanning. We will formulate the Angry Birds game with state variable representation, define commands and tasks, design refinement method and draw the refinement tree in the following part this section

4.1 State Variable Representation

We define the state variable representation for Angry Birds. The objects are

$$
\begin{aligned}
Sling &= \{sl\} \\
Birds &= \{b_1, b_2, \cdots, b_B\} \\
Pigs &= \{p_1, p_2, \cdots, p_M\} \\
Ices &= \{i_1, i_2, \cdots, i_G\} \\
Woods &= \{w_1, w_2, \cdots, w_W\} \\
Stones &= \{s_1, s_2, \cdots, s_S\} \\
Obstacles &= Ices \cup Woods \cup Stones \\
Points &= \{(x, y) | x, y \in Integers\} \\
MBRs &= \{(x_i, y_i) | x_i, y_i \in Integers, i \in \{1, 2, 3, 4\}\} \\
TNTs &= \{t_1, t_2 \cdots, t_T\} \\
Things &= Sling \cup Birds \cup Pigs \cup Obstacles \cup TNTs
\end{aligned}
$$

The following state variables are kept up-to-date by the vision component of angry birds:

$$
\begin{aligned}
&\forall obj \in Things, \text{exist}(obj) \in Booleans \\
&\forall obj \in Things, mbr(obj) \in MBRs \\
&\forall obj \in Things, blocked(obj) \in Booleans \\
&\forall obj1, obj2 \in Things, dist(obj1, obj2) \in Intergers
\end{aligned}
$$

exist(obj) indicates whether an object exists. mbr(obj) is the bound rectangle for an object. blocked(obj) indicates whether the object can be reached. dist(obj1, obj2) is the distance between two objects.

4.2 Commands and Tasks

The angry birds platform could execute the following commands:

- findMBR(): perceive all the MBR rectangles of Objects in the current state.
- estimateLauchPoint(sl, target): estimate the Start Points used to shoot bird from the Sling and the target Point.
- shot(b, start): shot a bird by pulling it to the start point.
- isReachable(sl, start, target): estimate if the Target Point is reachable based on the Start Point, or the Target is blocked by Obstacles.
- getSupporters(obj): get all the objects that suppot the object

We define the following commands that are executable in the program:

- getMBR(obj): get the MBR rectangle for an object.
- getHitPoint(obj): get the hit point for an object.
- isBlocked(obj): test whether an object is blocked
- getAllPigs(): perceive all the pigs in the scene
- getAllBirds(): perceive all the birds in the scene
- getCurrentBird(): get the current bird to use
- getTNTs(): get the TNTs in the scene
- getObstacles(obj): get the obstacles for obj
- estimateScoreOfObjs(objs): estimate the score based on the obstacle objects

We define the following tasks for angry birds:

- destroyAllPigs(): destroy all the pigs
- chooseTarget(): choose the next target to hit
- estimateScore(p): estimate the score of destroying a pig
- destroy(obj): destroy a object (a pig or a TNT)
- destroy(obj, b): destroy a obj (a pig or a TNT) with a bird
- hit(b, obj): hit an obj with a bird

4.3 Refinement Method

Our main task is to destroy all the pigs. To address the tasks listed , we define the following methods.

- m-destroyAllPigs(): destroy all pigs in a certain level
- m1-chooseTarget(): choose the TNT as the target of the highest priority if a TNT exists on the map
- m2-chooseTarget(): choose the pig with the highest expected score based on the distance and surrounding obstacles of the pig
- m1-selectTarget(): directly select a target based on its distance to the bird
- m2-selectTarget(): select a target based on the obstacles on path of the trajectory
- m1-destroyTarget(t): destroy a single target by hitting the TNT on the map

- m2-destroyTarget(p): destroy a single target by hitting the pig
- m1-destroyPig(p, b): destroy a pig with a bird
- m1-hit(a, obj): hit directly towards the target point of the object, ignore the status of the target or whether it is blocked by obstacles
- m2-hit(a, obj): when the object is blocked, choose to hit one of the obstacles
- m3-hit(a, obj): instead of hitting directly towards the object, choose to hit one of the supporters of the object
- m4-hit(a, obj): choose to hit one of the TNTs to indirectly cause the object to be hit

Refer to Appendix A for details.

4.4 ARP-interleave with SeRPE

We will use ARP-interleave in book Chapter 3.5 to integrate planning and acting. We use SeRPE in book Chapter 3.4 to simulate the plan. We test different search method with different heuristic to perform the nondeterministic choice in the algorithm.

SeRPE (Sequential Refinement Planning Engine) is a refinement planning engine in which all of the action models have deterministic outcome. SeRPE well serve the planning for a single step in our problem.

The refinement tree for this planning problem is shown in Figure 3.

The very top level task *destoryAllPigs()* is implemented in the method *m-destroyAllPigs* which further divide the task into two stps: the first task *chooseTarget()* which choose the next target (which can be a pig or a TNT) to destroy, and the second task *destroyTarget()* which targets to destroy a single target.

We provide two methods that implement the task *chooseTarget()*. Note that whenever there are multiple implementations, different heuristic functions will be applied to deterministically choose one implementation. More details about this will be discussed in later sections.

The first implementation *m1-chooseTarget()* will try to find the TNT on the map. It uses two actions *getTNTs()* and *isReachable()* to determine if there exisits any reachable TNT on the map. As a result, the first reachable TNT will be returned. Otherwise this method will return failure.

The second implementation *m2-chooseTarget()* will return the pigs on the map that has the maximum estimated scores. Inside the method, three actions will be applied: *getAllPigs()* which find all pigs on the map; *isBlocked()* which determined whether this pig is blocked; and *selectTarget()* which select the target based on two implementations based on the distance of target or the obstacles on the path of the trajectory.

Two different implementations for the task *destroyTarget()* is also provided in our refinement tree.

The first implementation *m1-destroyTarget(t)* will focus on destroying a TNT. Given that there is a unblocked TNT on the map, the task *hit(b, t)* can directly calculate the trajectory angle and return a point *start* where the bird should be pulled to. The reason behind this design is that TNTs may destroy a lot of blocks and pigs when explore.

To estimate the trajectory angle θ of the shoot, we exploit the trajectory function provided by [Ge and Renz, 2013]:

$$\theta_{\{1,2\}} = \arctan\left(\frac{v^2 \pm \sqrt{v^4 - g(gx^2 + 2 * y * v^2)}}{gx}\right) \quad (1)$$

where the coordinates (x, y) are the target position relative to the sling and the gravity constant variable $g = 1.0$, the velocity v is a constant velocity given by the slingshot.

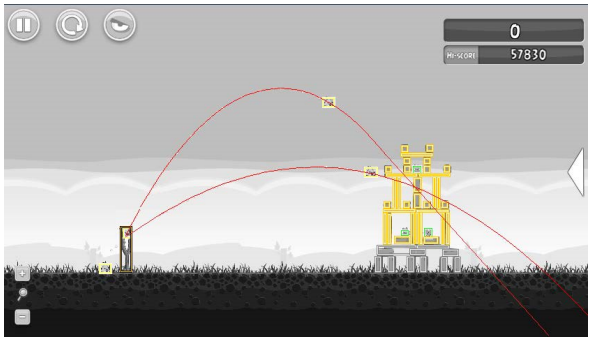

Figure 4: Example of Two Possible Trajectory Angles Aiming at the Same Target Point

After calculating the trajectory angle, the *shoot(b, start)* action will shoot the bird.

The second implementation *m2-destroyTarget(p)* will focus on destroying a pig. This method will be applied when there are remaining pigs whereas no more reachable TNTs on the map. Further, we provide several different implementations for the task *hit(b, p)* which uses a bird b to hit a target pig p.

The trajectory equation shown above is able to return multiple different angles with different hights given only two fixed points. Figure 4 just shows an example where two different trajectories that hit the same point.

Based on this fact, we designed several different methods for the task *hit(b, p)*. In the first implementation, we simply select any trajectory angle that will pass the position of the target p. This can be finished by three actions: *estimatedLaunchPoint(b, p)* which will return a *start* point to pull the bird b to, and *shoot(b, start)* will shot the bird b by releasing it at the point *start*. This approach is the simplest but may not work well because it does not take into consideration the affects of obstacles. In the second implementation, we will choose to hit the obstacles when the target p is blocked. In the third implementation, we will firstly execute an action *getSupporters(b)* which will get the supporters of the target; then the following actions *estimatedLaunchPoint(b, p)* and *shoot(b, start)* will calculate angle and shoot at a supporter.

Integrating Refinement Planning and Acting

The refinement methods and refinement tree shown above describes how we make plan for one single shot. More importantly, we are adopting the ARP-interleave framework which combines planning and acting and recalculates the plan for the next step each time a step is finished. The current state of the actual world after the previous step will be used thus resulting in a safer future plan.

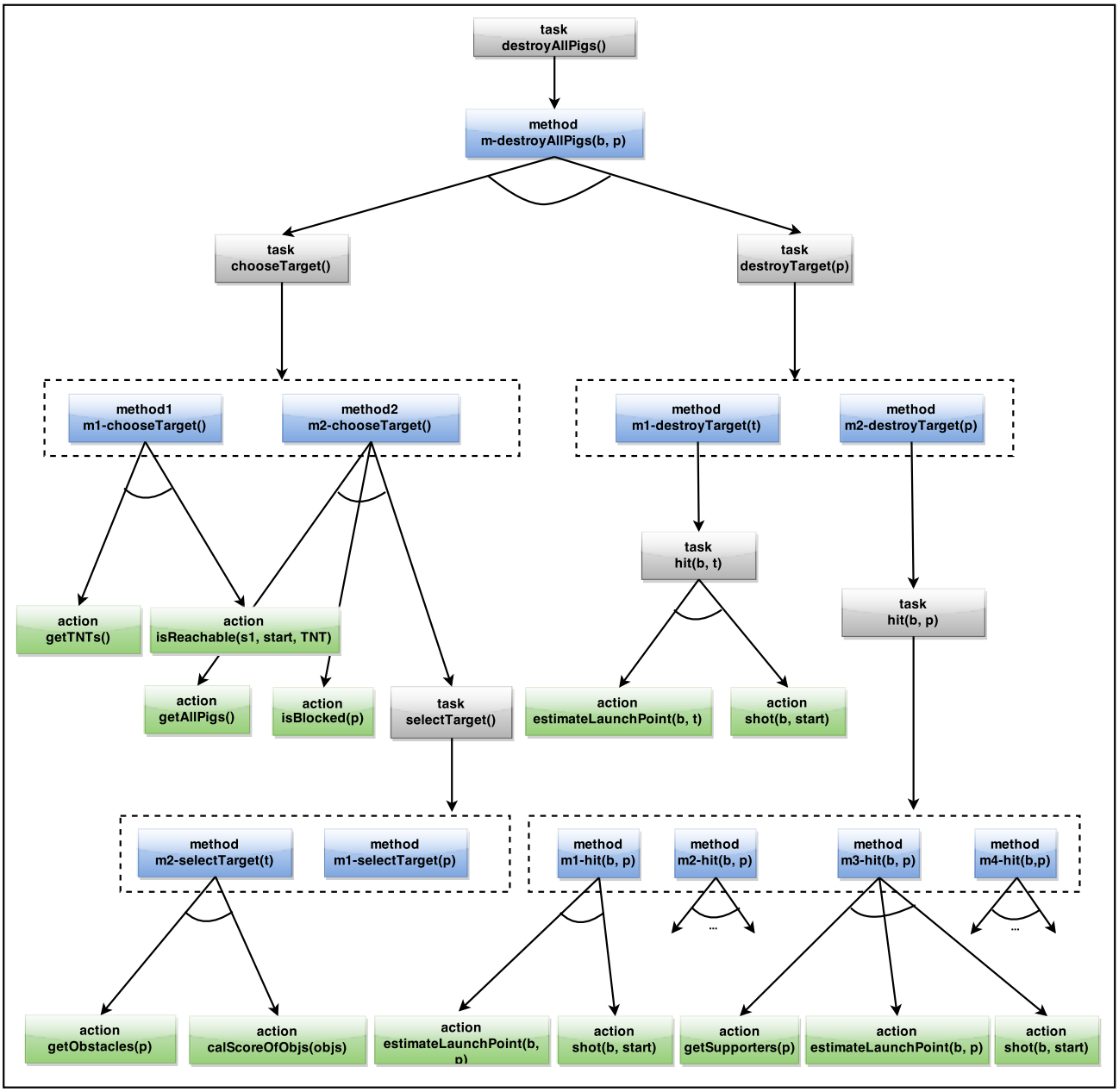

Figure 3: Refinement Tree of Planning for Angry Birds Puzzles

One of the biggest challenges in Angry Birds is that the world is seldom predictable. Even a well-trained player cannot tell the exact states of all objects after a single shot. For example, in some shoots, the bird is able to hit precisely the selected coordinate, but in other cases, the bird had a completely different trajectory and hit another objects. As a result, each action may trigger the following concrete consequences introduced by [Ghallab *et al.*, 2015]:

1. **Actions failures** A shot may not accomplish the goal of destroying a targeted pig.
2. **Unexpected side effects of actions** A shot may changing the future planning state by destroying other pigs.
3. **Exogenous events** A shot may put the stage into an unsolvable state like surrounding a pig with irons.

Without a precise predication function, a purely planning model or a SeRPE engine will not work without combining planning and acting. And specifically in our problem, the predication function for each step will be difficult and the actual current state of the world will be crucially important for the following steps. Taking these factors into consideration, the ARP-interleave model will work perfectly for our problem.

To be specific, our ARP-interleave framework will do the followings:

1. Before each step, it will generate a plan based on the SeRPE refinement tree. For tasks with multiple implementations, it will nondeterministically choose a implementation that will get the best score best on a certain type of heuristic function. All tried methods for the current subgoal will be recorded by the system.
2. After each step, the system will check the current status of the world. It will check whether the previous subgoal was achieved by the previous step. Also, it will update the status of all the objects (like pigs, stones, woods, etc.) according to their latest actual state.
3. If the current step fails to achieve the current goal, the planner will try to use the remaining methods that are feasible in current state. If no such method can be found, the current subgoal will be skipped for the moment.
4. If the current step succeed. The system will go on with the next subgoal.
5. Each time the system will check the latest status of the world to determine whether a previously skipped goal has a feasible refinement tree. If it is true, a new plan will be generated for the previously skipped subgoal.

have several advantages:

1. It models the problem more accurately.
2. Plan generated by this system is more practical because the actual status of the world are always taken into consideration.
3. A better decision can be made about choosing the next subgoal.

4.5 Search methods for nondeterministic choice in SeRPE

In this section, we introduce our search method to plan actions in the simulation algorithm SeRPE. We adopt three forward search methods in our implementation: greedy search, depth-first search and A* search. In the remainder of the paper, we will call the SeRPE algorithm with those forward search methods as SeRPE-Greedy, SeRPE-DFFS and SeRPE-A*, respectively. We will discuss how to apply those methods in the angry birds game in details as follows.

SeRPE-Greedy

We discuss how to implement SeRPE with the greedy search method. Our greedy search is based on two heuristics

1. always choose a pig as target
2. choose the pig that is closest to the sling

Heuristic 1 force the AI to destroy all pigs to accomplish the current level of the game. Heuristic 2 is based on the observation that when hitting the closest pig, there is high probability to cause the other pigs to be destroyed, and hence accomplish the current level of the game. Each time SeRPE-Greedy is called, it will return a plan that hits the pigs one by one in the order according to the distance. In our planning and acting framework, simulation algorithm is called after each action. When we use SeRPE-Greedy, a plan that hits the existing pigs one by one in the order according to the distance of pigs and sling is returned.

SeRPE-DFFS

We introduce SeRPE-DFFS for angry birds, which utilize the depth first search method while generating plans with SeRPE. The depth-first search uses one heuristic: always choose a pig as target. One advantage of depth first search comparing with greedy search is that DFFS guarantees to find a solution if it exists. This is guaranteed by backtracks in DFFS. DFFS randomly choose each applicable action to form a planning sequence. If the sequence is not a feasible solution, then the algorithm backtracks to the previous state and choose another action. In our SeRPE-DFFS, DFFS is used to choose pigs to hit. Each time SeRPE-DFFS is called, it will return a plan the hits the existing pigs in a random order.

SeRPE-A*

We also investigate A* search method to implement SeRPE, which we name as SeRPE-A*. A* search surpass greedy search and DFS when the state space is not too large, as A* guarantees to return the optimal solution. The key problem in applying A* search method is to design the heuristic function. In our SeRPE-A*, the cost of each action is evaluated by the predictable consequences of the action. We list the scores of objects in angry bird in Tab. 4.5. Our heuristic function is based on those scores. We define the cost of hit action as:

$$cost(hit(b, obj)) = -score(obj) \quad (2)$$

. where obj refers to And cost of all the other actions as 0. When SeRPE-A* is called, a planned action with the lowest cost (highest predictable score) will be returned. As TNT has the highest score in our design, the first action will have high probability to target on hitting the a TNT.

The heuristic function used to calculate the score. The scores for destroying each types of different objects are shown in Table 1.

Table 1: Basic Score Metrics

Event	Score
TNT	10000
Bird left	10000
Pig destroy	5000
Destroy sth	500
Damage sth	50

Moreover, we design an extra feature while applying SeRPE-A* in the planning and acting framework. Each time SeRPE-A* is called, we will estimate the final score by adding up the predictable score returned by SeRPE-A* and the achieved score the AI agent got by performing previous actions. If the estimated score is less than the scores achieved by performing our previous plans, we will stop the current ARP-interleave procedure and restart the whole refinement process by restarting the game in the current level.

5 Implementation

5.1 Greedy Algorithm

One naive approach to this problem is a purely greedy algorithm. For each bird $b \in Birds$, the baseline planner aim at pigs one by one, without considering any other obstacles.

In this approach, a planning domain is constructed with several assumptions about the plan: (1) a naive predication function will assume each action being executed successfully without impacting states of obstacles; and (2) a heuristic function will be used to measure the cost from each state to the sub-goal using the score of achieving that sub-goal. Based on the planning domain constructed, a greedy algorithm will try actions in decreasing order of the possible scores.

This algorithm is fast and easy to implement. However, the drawbacks for this approach are obvious. Firstly, it doe not guarantee to return a solution. More importantly the predication function may be unrealistic and without acting and planning, the state set of the planning domain will be different from actual states.

5.2 Forward State-Space Search

Since greedy algorithm does not guarantee to return a solution for the problem, we also evaluate the performance of the Depth-First Forward Search (DFFS). However, given such a huge search space (each bird has 360 possible angles to be shot) and the uncertainty of the game engine, the DFFS also does not guarantee a solution given a fixed amount of time.

5.3 A*

DFFS will suffer from time cost and it will have to enumerate over all feasible plan to find the optimal plan in the worst case. Whereas A* is much better in performance when especially in the context of this problem when time is limited. A* guarantees to return the optimal solution when there is one. It requires more space complexity but will work well when the plan space is not big.

6 Experiments

We have carried out a number of experiments to evaluate the performances with or without SeRPE using different search algorithms.

6.1 Datasets and constrains

We adopt the first chapter *Poached Egges* of the chrome version of Angry Birds game. The time limitation is set to 10 minutes for each level. We evaluate the highest score achieved by different algorithms as well as average time needed to pass a level. Table 2 shows some basic statistics about each level, including number of birds, pigs, woods, stones, ices and total number of obstacles. The experiment was conducted in Google Chrome (Version 40.0.2214.115 m) with Nvidia Quadro K6000, Intel Xeon CPU E5-2667 2.90GHz and 36GB RAM in Windows 8 workstation.

Table 2: Basic Statistics of Each Level

Level	Birds	Pigs	Woods	Stones	Ices	Tot Obstacles
1	3	1	12	1	2	15
2	5	4	4	4	0	8
3	4	2	4	3	0	7
4	4	1	5	3	5	13
5	4	5	8	13	11	32
6	4	2	6	6	6	18
7	4	3	7	4	18	29
8	4	4	2	5	2	9
9	4	3	23	6	4	33

6.2 Experimental Results

We measure 6 methods in the first 10 levels of Angry Bird in aspect of highest scores obtained and average time cost to pass a level.

6.3 Highest Scores

We firstly measure the highest scores achieved by different methods. The results are shown in Table 3 and Figure 5. The results show that SeRPE-Greedy, SeRPE-DFFS and SeRPE-A* significantly outperform their counter parts which are simply planed with Greedy, DFFS or A*. In specific, SeRPE-A* works the best in almost all levels.

6.4 Average Time Cost

For all 6 methods, we also measure the average time cost for each level. We set the maximum time limit 600000 milliseconds (10 minutes) and will terminate the process at this limit. As shown in Table 4 and Figure 6, methods incorporating SeRPE engines outperform their counterparts in most cases. The difference at the beginning levels are quite small, but grows dramatically at the last a few levels. Just as we have expected, SeRPE-A* outperform all other methods in most levels.

6.5 Discussion

In this section, we discuss lessons that learnt from the experimental results.

Table 3: Highest Score for Each Level

	1-1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9
Greedy	29290	33790	31660	10990	40700	24000	15640	11480	9600
DFFS	29330	52240	34380	34380	41970	34380	34380	27150	20590
A*	30130	52500	41910	29770	61110	63770	32070	34380	39640
SeRPE-Greedy	29290	33840	33490	25670	56960	33620	20060	11780	18510
SeRPE-DFFS	30470	52390	40870	40700	62030	39880	31910	28480	39700
SeRPE-A*	32460	61500	41960	42530	64440	63770	46280	46720	42380

Table 4: Average Time Cost to Pass Each Level (ms)

	1-1	1-2	1-3	1-4	1-5	1-6	1-7	1-8	1-9
Greedy	50004	76937	58662	73220	59974	600000	600000	600000	600000
DFFS	43114	93699	57586	133267	113281	363596	600000	600000	600000
A*	42099	86056	58140	77596	161638	185486	168320	193087	187021
SeRPE-Greedy	49604	56412	58499	74077	59637	94921	151850	600000	600000
SeRPE-DFFS	37507	54632	48246	114650	108311	168320	259924	600000	600000
SeRPE-A*	34371	46056	34383	71690	49652	75946	84274	82887	93856

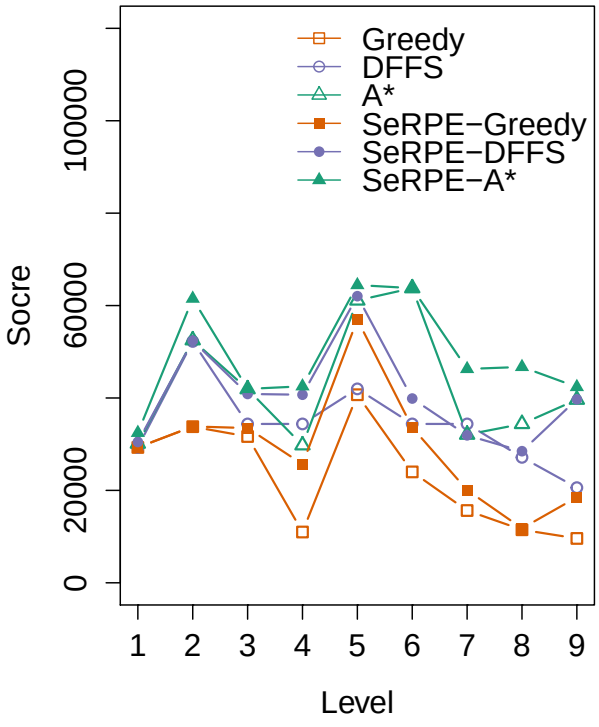

Figure 5: The Highest Scores for Each Level

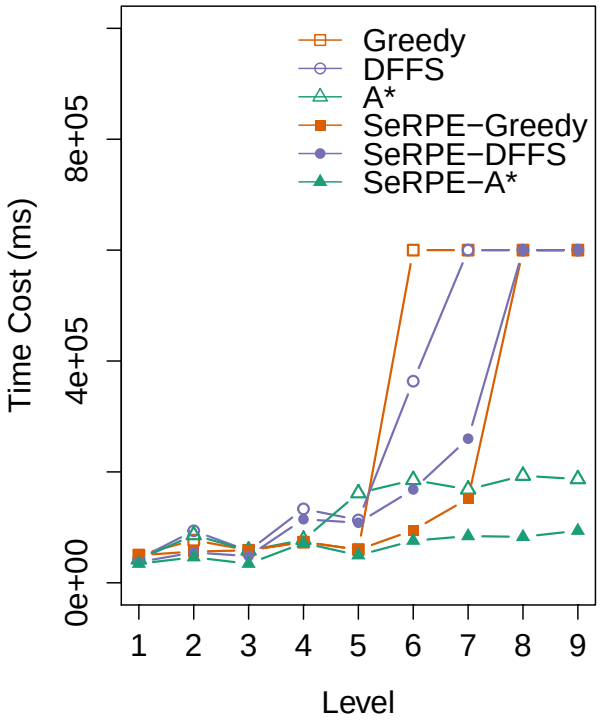

Figure 6: Average Time Cost to Pass Each Level

In a deterministic model where the planning is 100% accurate, an automatic agent does not necessarily need refinements methods to modify the initial plan. An offline plan can ideally solve the problem. Nevertheless, in a real-world problem like Angry Birds, the deterministic model even with A^* search cannot guarantee a solution. This is caused by the following reasons:

1. **Inaccurate Simulation** It is a challenging task to derive an exact model of a physical world. The knowledge base of the physical world is built upon the prior exploration with trials and errors. Even in a virtual world like Angry Birds, it is almost impossible to model the world without reading its source code.
2. **Changes in Time** In practical scenarios like Angry Birds, the global states of any object may be subject to change form time to time. For example, it is very hard to know whether the scene is static or not after a shot. The structure may be unstable and moving in a very slow pace. As a result, the subtle changes in the global states may lead to failure of the current plan.
3. **Erroneous Actions** As for human beings, it is hard to perform an action such as *move left hand 5 centimeters forward exactly*. Similarly, for a video game like Angry Birds that involves human interactions, the acting engine hardly points to the perfect position in pixel-level. Consequently, the actor may produce different states than the planner predicted.

Hence, deliberately planning and acting with refinement methods outperforms the deterministic models. Though only partial plan is conducted each time, the engine itself observes the current state and makes changes to future plans. For some other cases, it may be impossible to generate a complete plan in limited time. Then partial plan can be generated and performed first. With A^* algorithm, whenever the current state cannot exceed the highest score achieved previously using the maximal heuristic function value, the game can be restarted to save time.

7 Conclusion

In this paper, we present a deliberately planning and acting system with refinement methods for the popular Angry Birds video game. We design a RAE based on ARP-interleave with SeRPE. Some algorithms, greedy, DFFS and A* are incorporated to search for plans. Our experimental results show that ARP-interleave outperforms their counterparts which purely rely on the search algorithms.

Acknowledgments

The authors would like to thank Prof. Dana S. Nau and Dr. Vikas Shivashankar for teaching CMSC 722 and advising the team for this paper.

References

[Calimeri *et al.*, 2013] Francesco Calimeri, Michael Fink, Stefano Germano, Giovambattista Ianni, Christoph Redl, and Anton Wimmer. AngryHEX: An Artificial Player for Angry Birds Based on Declarative Knowledge Bases. In *Proceedings of the Workshop Popularize Artificial Intelligence co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013)*, pages 29–35, 2013.

[Ferreira *et al.*, 2013] Leonardo Anjoletto Ferreira, Guilherme Alberto, Wachs Lopes, Paulo Eduardo Santos, Universidade Metodista, De São Paulo, São Bernardo, São Paulo, São Bernardo, and São Paulo. Combining Qualitative Spatial Representation Utility Function and Decision Making Under Uncertainty on the Angry Birds Domain. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[Ge and Renz, 2013] Xiaoyu Ge and Jochen Renz. Representation and reasoning about general solid rectangles. *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 905–911, 2013.

[Ge *et al.*, 2014] XiaoYu Ge, Stephen Gould, Jochen Renz, and Andrew Wang Peng Zhang Sahan Abeyasinghe, Jim Keys. Angry birds game playing software version 1.32, 2014.

[Ghallab *et al.*, 2015] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning and Acting*. preprint, 2015.

[AI birds, 2013] AI birds. Angry Birds AI Competition 2013 accepted papers. `http://goo.gl/GM7eQT`, 2013.

[AI birds, 2014] AI birds. Angry Birds AI Competition 2014 accepted papers. `http://goo.gl/GkBF75`, 2014.

[AI birds, 2015] AI birds. IJCAI 2015 Angry Birds AI Competition. `http://goo.gl/OEkd97`, 2015.

[Jutzeler *et al.*, 2013] Arnaud Jutzeler, Mirko Katanic, and Jason Jingshi Li. Managing Luck : A Multi-Armed Bandits Meta-Agent for the Angry Birds Competition. 2013.

[Narayan-chen *et al.*, 2013] Anjali Narayan-chen, Liqi Xu, and Jude Shavlik. An Empirical Evaluation of Machine Learning Approaches for Angry Birds . 2013.

[Polceanu, 2014] Mihai Polceanu. Towards A Theory-Of-Mind-Inspired Generic Decision-Making Framework. 2014.

[Shah, 2004] Nimish Shah. Knowledge Representation, Reasoning and Declarative Problem Solving by C. Baral, Cambridge University Press, 2003, 2004.

[Tziortziotis *et al.*, 2014] Nikolaos Tziortziotis, Georgios Papagiannis, and Konstantinos Blekas. A Bayesian Ensemble Regression Framework on the Angry Birds Game. pages 1–12, 2014.

[Wa, 2014] Przemysaw Wa. Qualitative Physics in Angry Birds : First Results, 2014.

[Zhang and Renz, 2014] Peng Zhang and Jochen Renz. Qualitative Spatial Representation and Reasoning in Angry Birds: The Extended Rectangle Algebra, 2014.

A Appendix: Refinement method

$m{-}destroyAllPigs()$
$task: destroyAllPigs()$
$pre: P\ is\ not\ \emptyset$
$body:$
$P = getAllPigs()$
$if\ P\ is\ not\ \emptyset:$
$nondeterministically\ choose\ p \in P:$
$destroy(p)$

$m1{-}chooseTarget()$
$task: chooseTarget()$
$pre: P\ is\ not\ \emptyset$
$body:$
$TNT = getTNTs()$
$if\ TNT\ is\ not\ \emptyset$
$for(t \in TNT)$
$if\ isReachable(t)$
$return\ t$

$m2{-}chooseTarget()$
$task: chooseTarget()$
$pre: P\ is\ not\ \emptyset$
$body:$
$P = getallPigs()$
$if\ P\ is\ not\emptyset$
$get\ P' \subset P\ that\ is\ not\ blocked$
$nonderministicallychoose\ p$

$m1{-}destroyPig(p)$
$task: destroyPig(p)$
$pre: exist(p) = F$
$body:$

$m2{-}destroyPig(p)$
$task: destropyPig(p)$
$pre: exist(p) = T$
$body:$
$A = getAllBirds()$
$if\ A\ is\ not\ \emptyset:$
$a = getCurrentBird():$
$destroy(p, a)$

$m1{-}destroyPig(p, a)$
$task: destropyPig(p, a)$
$pre: exist(p) = T, exist(a) = T$
$body:$
$isBlocked(p)$
$hit(a, p)$

$m1{-}hit(a, obj)$
$task: hit(a, obj)$
$pre: exist(obj) = T, exist(a) = T$
$body:$
$tgtPnt = getHitPoint(obj)$
$sttPnt = estimateLauchPoint(sl, tgtPnt)$
$shot(sttPnt, time)$

$m2{-}hit(a, obj)$
$task: hit(a, obj)$
$pre: exist(obj) = T, exist(a) = T, blocked(obj) = T$
$body:$
$obs = getObstacles(obj)$
$nondeterministicallychooseob \in obs:$
$hit(a, ob)$

$m3{-}hit(a, obj)$
$task: hit(a, obj)$
$pre: exist(p) = T, exist(a) = T$
$body:$
$spObjs = getSupporters(p):$
$nondeterministicallychoosesp \in spObjs:$
$hit(a, sp)$

$m4{-}hit(a, obj)$
$task: hit(a, obj)$
$pre: exist(p) = T, exist(a) = T$
$body:$
$tnts = getTNTs():$
$nondeterministicallychoosetnt \in tnts:$
$hit(a, tnt)$

B Appendix: Code

Our code is available on Github: `https://github.com/ruofeidu/CMSC722_AngryBirds`

C Appendix: Sample Log

A sample log output illustrating how our system runs.

```
0       [System]
Program starts with configuration -d
32      [System]
Server waiting on port: 9000
1051    [System]
Client connected
1934    [System]
Running A* agent without IRPE
8309    [Vision]
Detecting objects in the Angry Birds using vision module
10068   [Vision]
Detect TNTs 0
10069   [Vision]
Detect pigs and rolling 1, glass 5, stone 1, wood 27, birds 8
10069   [Vision]
Statistics: 1, 34, 0
10213   [Planner]
m1-destroyAllPigs: 1 pigs
10216   [Planner]
m2-destroyPig ab.vision.ABObject[x=536,y=290,width=14,height=10]
10216   [Planner]
Estimate launch point from 2
10217   [Planner]
launch point java.awt.Point[x=9,y=961], angle 73.79194556203768
17742   [Actor]
Shooting starts 193, 328, -184, 633
27765   [Actor]
Shooting completes
28096   [Vision]
Scale factor = 1.0006048459391437
28267   [Vision]
Detecting objects in the Angry Birds using vision module
28943   [Vision]
Detect TNTs 0
28944   [Vision]
Detect pigs and rolling 0, glass 2, stone 1, wood 10, birds2
28944   [Vision]
Statistics: 0, 13, 0
29116   [Planner]
m1-destroyAllPigs: 1 pigs
29116   [Planner]
m2-destroyPig ab.vision.ABObject[x=536,y=290,width=14,height=10]
29117   [Planner]
Estimate launch point from 2
29117   [Planner]
launch point java.awt.Point[x=0,y=959], angle 72.99302990476194
36778   [Actor]
no sling detected, can not execute the shot, will re-segement the image
37049   [Vision]
Detecting objects in the Angry Birds using vision module
38198   [Vision]
Detect TNTs 0
38198   [Vision]
Detect pigs and rolling 0, glass 0, stone 4, wood 5, birds8
38198   [Vision]
Statistics: 0, 9, 1
43113   [Result]
Level 1 gets 29290
43114   [Result]
Total Score: 29290
```