CHAPTER 5

Small Single Player Games

5.1 Introduction

We start our tour of general game playing by looking at single player games. In the game-playing community, these are often called *puzzles* rather than games; and the process of solving such puzzles is often called *problem-solving* rather than game-playing.

Puzzles are simpler than multiple-player games because everything is under the control of the single player. The world is static, except when the player acts; and changes to the world are determined entirely by the current state and the actions of the player.

In this chapter, as in most of this book, we assume that the player has complete information about the game. We assume that it knows the initial state; it knows all of its legal actions in every state; it knows the effects of its actions in every state; for every state, it knows its reward; and, for every state, it knows whether or not it is terminal.

In this chapter, we also assume the games are small, i.e. they are small enough so that there is sufficient time for the player to search the entire game tree. This guarantees that the player can find optimal actions to perform. That said, as we shall see, it is sometimes possible to find optimal actions even without searching the entire game tree.

Despite these strong assumptions (just one player, complete information, and the availability of adequate time to search the game tree), the study of single player games is a good place to start our look at general game playing. First of all, many real world problems can be cast as single player games with these same restrictions. More importantly for us, as we shall see, the techniques we examine later can be viewed as more elaborate versions of the basic techniques introduced here.

We begin this chapter with an example of a single-player game that we use throughout the chapter. We then look at two different approaches to single-player game playing.

5.2 8-Puzzle

The 8-puzzle is a sliding tile puzzle. The game board is a 3x3 square with numbered tiles in all but one of the cells. See the example shown below.

	1	3
4	2	5
7	8	6

The state of the game is modified by sliding numbered tiles into the empty space from adjacent cells, thus moving the empty space to a new location. There are four possible moves - moving the empty space up, down, left, or right. Obviously, not all moves are possible in all states. The states shown on the left and right below illustrate the possible moves from the state shown in the center.

4	1	3	down ⇐		1	3	right ⇒	1		3

	2	5
7	8	6

4	2	5
7	8	6

4	2	5
7	8	6

The ultimate object of the game is to place the tiles in order and position the empty square in the lower right cell. See below.

1	2	3
4	5	6
7	8	

The game terminates after 8 moves or when all of the tiles are in the right positions, whichever comes first. Partial credit is given for states that approximate the ultimate goal, with 10 points being allocated for each numbered tile in the correct position and 20 points being allocated for having the empty tile in the correct position. For example, the initial state shown above, the one in the middle, is worth 40 points; and the goal state is worth 100 points.

5.3 Compulsive Deliberation

Compulsive Deliberation is a particularly simple approach to game playing. On each step, the player examines the then-current game tree to determine its best move for that step; and it makes this move. It repeats this process on the next step and so forth until the end of the game.

In pure compulsive deliberation, each step of the computation is independent of every other step. No data computed during any step is accessible on subsequent steps. The player treats each step as if it were a new game. This is obviously wasteful, but it does not hurt so long as there is enough time to do the repeated calculations. We start with this method because it is simple to understand and at the same time serves as a template for the more sophisticated, less wasteful methods to come.

The following procedure is a simple implementation of compulsive deliberation. On receipt of a play message, the player simulates the specified move as usual. It then computes all of its legal actions in this new state and iterates through these actions comparing the score of each to the best score found so far. If it ever finds an action that guarantees a reward of 100, it stops and returns that action. Otherwise, it retains the highest score and action and returns the corresponding action when it is done.

```
function play (id,move)
 {state = simulate(move,state);
  return bestmove(role,state)}

function bestmove (role,state)
 {var actions = findlegals(role,state,game);
  var action = actions[0];
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var result = maxscore(role,simulate([actions[i]],state));
       if (result==100) {return actions[i]};
       if (result>score) {score = result; action = actions[i]}};
  return action}
```

Recall that every game has a unique reward for each player in each state. The reward is a perfect measure of utility for terminal states. Unfortunately, in general, the rewards for non-terminal states do not always correlate with the rewards for terminal states; and so they are not necessarily useful in defining the scores for non-terminal states.

State utility is a measure of utility that is relevant for both terminal states and non-terminal states. By definition, the utility of a state for a player is defined as being the best reward the player can guarantee for itself by any sequence of legal moves starting in the given state.

One way to determine the utility of a state is by computing all sequences of legal actions that lead from the given state to a terminal state and taking the maximum reward for the terminal state resulting from each such sequence.

The following procedure computes the same score in a slightly simpler manner. The procedure takes a player and a state as arguments and returns the corresponding utility as score. The procedure computes the score via a recursive exploration of the game tree. If the state supplied as argument is terminal, the output is just the player's reward for that state. Otherwise, the output is the maximum of the utilities of the states resulting from executing any of the player's legal actions in the given state.

```
function maxscore (role,state)
 {if (findterminalp(state,game))
     {return findreward(role,state,game)};
  var actions = findlegals(role,state,game);
  var score = 0;
  for (var i=0; i<actions.length; i++)
      {var result = maxscore(role,simulate([actions[i]],state));
       if (result>score) {score = result}};
  return score}
```

Since all games in GGP competitions terminate, this procedure always halts; and it is easy to see that it produces the utility for the specified role in the specified state.

Exercises

Exercise 5.1: What is the reward associated with each of the 8-Puzzle states shown below?

(*a*)

	1	2
3	4	5
6	7	8

(*b*)

1	2	3
4		5
6	7	8

(*c*)

	1	3
4	2	5
7	8	6

(*d*)

1	2	3
4	5	6
7	8	

Exercise 5.2: Consider the 8-puzzle with the initial state shown below, and assume that there are exactly two steps left in the game, i.e. the game ends after exactly two additional actions.

1	2	3
4	5	6
7		8

(*a*) What is the player's reward in this state?
(*b*) What is the utility of this state?
(*c*) How many nodes are in the game tree?
(*d*) How many distinct states are in the game tree?
(*e*) What is the maximum number of nodes examined by compulsive deliberation?

Exercise 5.3: Consider the 8-puzzle introduced in the notes. The initial state of the game is shown below on the left, and the ideal arrangement is shown on the right.

	1	3
4	2	5
7	8	6

1	2	3
4	5	6
7	8	

Recall that there are 4 feasible actions - up, down, left, right - and any of these actions is legal so long as the empty tile remains within the 3x3 grid. The reward for a state is the sum of rewards for each tile, with 10 points being awarded for each numbered tile in its ideal position and 20 points being awarded if the empty cell is in its proper position.

(*a*) Which of the following sequential plans are legal?

```
[right,right,down,down]
[right,right,right,down]
[right,down,right,down,left,up,right,down]
[right,down,right,left,right,left,right,down]
```

(*b*) Which of the following sequential plans are complete?

```
[right,right,down,down]
[right,down,right,left,right,left,right,left]
[right,down,right,left,right,left,right,down]
[right,down,right,down,left,up,right,down]
```

(*c*) Which of the following sequential plans are minimal?

```
[right,right,down,down]
[right,down,right,left,right,left,right,left]
[right,down,right,left,right,left,right,down]
[right,down,right,down,left,up,right,down]
```

(*d*) Which of the following sequential plans are optimal?

```
[right,down,right,down]
[right,down,right,left,right,left,right,left]
[right,down,right,left,right,left,right,down]
```