Homework 4
Computer Science
Spring 2017
B351

Steven Myers

February 26, 2017

All the work herein is mine.

Answers

1. The difference between a general search strategy and a strategy we would use to search a game's state space is that games involve an adversary where general search does not. In the example of navigating a maze, we simply have nodes that we can travel to that have some kind of distance from the goal node and weight at each intermediate node. In a game, there are two or more adversaries with contingent strategies that manipulates all game state nodes in each move. To say that a *general search strategy* is *sound* for a game is false because it would suggest that whatever action yields the closest state to the goal state at the current state is truly the closest to winning the game. However, this may not always be true in the human world since in reality, an adversary may flip the odds against the current player at any moment, regardless of how good the previous move was. In conclusion, we must consider the adversary's strategy in addition to just a single player's strategy, or else our strategy is not sound - only complete.

2. In short, yes we can modify minimax to work with three players. Instead of a two-player minimax function where one player attempts to minimize the "score" or cost and other maximizes, we would need to make it so our minimax function attempts to maximize the current player's score, while minimizing both of the other player's scores. For instance, A will want to have a better score than both B and C. Here's a sample of what that might look like:

```
# Each curly bracket set is the scores of A, B, and C respectively.
Turn 1, A's move:
            {0,0,0} # initial state, no player has any advantage
    /          |           \
{4, 2, 1}  {0, 0, 1}  {4, 4, 4}

Turn 2, B's move:
             {4,2,1} # 'A' chose left branch due to obvious benefit
     /          |          \
{1, 0, 1}  {10, 1, 10}  {5, 0, 5}

*Turn 3, C's move:
         ?
```

*We would need some additional functionality to determine whether or not it's more beneficial to take a move that would result in a net gain in B's score, but would also result in a more disastrous gain for

our opponents. Turn 2 in the example above gives such an example. We could not simply choose the middle branch, despite that it maximizes B's score since the benefit for A and C is so great. A two player minimax function would have to consider possible alliances that would benefit both players to attack against the tertiary opponent.

3. My Gobblet files are located in *gobblet.py*. It can be played using the parameters given in the prompt. I was able to fully implement the game and an AI that will make moves. However, my alpha-beta-search does not yield the proper moves. I experimented with different evaluation functions, like rewarding points if there are lots of two-in-a-row's, three-in-a-row's, or subtracting a score based on how many pieces they have left in their external stacks. None of these methods seemed to yield proper moves, though. Additionally, I think that my pruning is not done properly. A depth N greater than 2 takes 60 seconds to complete. I know that one way to reduce the initial branching factor would be to remove duplicate moves from the external stacks that yield the same result. For example, there are 48 moves in the first move of the game because you can move one piece from the external stack in 16 ways; however, those 16 moves are equivalent in utility value initially regardless, so they can be pruned. I wasn't sure how to go about doing this.