Project Report

Description of Problem

This project was to use PDDL to make an AI checkers player. PDDL is a language for planning. I thought this would be interesting since games like checkers and chess require the player to plan ahead. Many games use AI to act as a single player.

Description of Method

After building a checkers game and a PDDL planner using weighted A* using the hlits heuristic. I translated the state of the board to PDDL. The planner generated all plans with certain number of steps. They are then sorted by how many goal states were met in the end state. The planner will run when either there are not more moves that have been planed or the current planned move is invalid. The PDDL represented the computer's piece positions with a number of predicates matching the number of spaces from the board edge in each direction. Moves were actions that subtracted predicates from and added predicates to the state. Kinged pieces get a Kinged predicate added to the state. Opponent pieces are not represented themselves but are portraied as actions to capture those pieces, adding a Captured predicate when the action is taken. This means the planner may attempt to make some moves that are invalid like moving on top of the opponent's pieces.

Solution Performance and Other Solutions

There are other solutions to this problem, for example MiniMax. Most other solutions would probably be better. The primary drawback is the atomic nature of PDDL making it hard to represent complex systems. In this case the board's state was "dumbed down" when converted to PDDL. The issue with this solution is it takes a lot of time to make plans and space to store all the states and actions. This could be improved with a better PDDL representation of the board, caching for common plans (for example the first plan is always the same for a given board size), as well using better heuristics during the planning phase.

Extensions

While I would call the project a success as I did make a checkers player using PDDL it is not efficient or necessarily well put together. I would like to in the future consider either a different method , like MiniMax, or extending the PDDL language to allow for more complex types than the simple addition or removal of predicates.