Exercise 2

General Game Playing

Prof. Michael Thielscher and Stephan Schiffel

International Masters Programme in Computational Logic — winter term 2009/10

09.11.2009

Exercise 2.1

Encode the following concepts as valid gdl rules:

- the initial setup of an 8x8 checkers board
- an action that moves one of the pieces one row diagonally forward (This includes the precondition and the effects of the action.)
- a successor relation for numbers from 0 to 12
- a ‘smaller than’ relation over above numbers
- counting the number of pieces of each color on the board
- the game ends if one of the two players lost all his pieces or after 60 steps
- the player with more pieces on the board wins

Exercise 2.2

Decrypt the scrambled game descriptions from the course web page by analyzing the rules and replacing the nonsensical relation symbols, function symbols, atoms, and variable names with meaningful ones.

- Do you know the name of the game? If not, try to explain the rules as simple as possible.
- How did you recognize the meaning of the rules?

Exercise 2.3

Encode your game from exercise 1.4 in GDL. Make sure that your description abides by the GDL specification. In particular check:

- correct syntax, safety of rules (Definition 6 in the GDL spec)
- well-formedness of the game description (Definitions 21 to 25 in the GDL spec)

Try your game description with the GameController and a player of your choice to see if it works. Hint: GameController will output parsing errors on the terminal.

Send me an email by **Thursday, Nov 19th 2009** with your game description and, for each player, a sequence of joint moves such that the player wins. Prepare to explain, why the game is well-formed: How do you ensure terminality, playability, and winnability?

Exercise 2.4

Implement exhaustive search for single player games. Your program should be able to solve small single player games like 'Maze', 'Blocks', 'Buttons' or 'Pancakes' within seconds or a few minutes (see table below). Solving a game means to find a sequenze of moves to win within the startclock. To actually play the game perfectly after that you have to remember the best move for each state you encountered during search such that you can reuse that information. For bigger games or smaller clock values your program won't be able to exhaustively search the game tree in the given timeframe but it should still play better than a random player. Make sure your program always returns a legal move in time, even if the search hasn't finished.

Game	Startclock	Playclock
Buttons	10	10
Pancakes	30	10
Asteroids	60	10
Hanoi	60	10
Aipsrovers01	120	10

Ideally, your program will be able to solve the following games in the given times. If you can not solve the games in the given times or just want to get better:

- Use a hash table to recognize repeated states. This can lead to substantial reduction of the size of the search space. It is also a good way for reusing information from a previous step search in the next step.
- Improve your algorithms, avoid unneccessary computations, object creations, memory allocations, etc.

Play a match of each of the games above with your player and send me an email with the gamecontroller logs or links to the matches on GGP Server.

Exercise 2.5

Test your program with more difficult single player games (e.g. 8puzzle or peg) and longer start clock ($\approx 15 - 30min$) and play clock ($\approx 3 - 5min$) to check that it is able to handle longer runtime and bigger memory requirements. Although your program may not be able to solve these games it should always return a legal move.