LUDO simulator
Tools of Artificial Intelligence

This is a quick introduction to the LUDO simulator used in the course.

JavaDoc

The following java files are included in the simulator:

Interface Summary	
LUDOPlayer	Interface which any automatic ludo player must implement.

Class Summary	
AggressiveLUDOPlayer	Example of automatic LUDO player
FifoLUDOPlayer	Example of automatic LUDO player
LUDO	Main class the LUDO simulator - "controls" the game.
LUDOBoard	The LUDOBoard class is the core class of the LUDO simulator.
ManualLUDOPlayer	Example of automatic LUDO player
PacifisticLUDOPlayer	Example of automatic LUDO player
RandomLUDOPlayer	Example of automatic LUDO player
SemiSmartLUDOPlayer	Example of automatic LUDO player

To make an automatic LUDO player you need to write a java class implementing the Interface **LUDOPlayer**. The class contains just one method: play() which is called each time your player should decide which brick to move.

Method Summary	
void	play() The play() method is called each time it is the players turn to roll the dice and play.

The **LUDOBoard** class constrains a representation of the actual LUDO board and methods to acquire information's about the state of the game. See the source code and the JavaDoc included in the simulator for further details.

Method Summary

boolean	**almostHome**(int index, int color) If a given index corresponding to color are in colored(safe) area close to home.
boolean	**atField**(int index) if a given index is at the field(white) area.
boolean	**atHome**(int index, int color) If index corresponding to color are in home area(brick completed game).
int[][]	**getBoardState**() get Bricks positions(board state)
int	**getDice**() The current value of the dice.
int[]	**getMyBricks**() Get index-positions of your bricks.
int	**getMyColor**() Get your color.
int[][]	**getNewBoardState**(int nr, int color, int dice2) Get the index-positions of the bricks if a particular brick, of a given number and color is moved a given a dice value.
int[]	**getPoints**() Get points for completed game indexed as: points[color] points are given from 3 to 0, 3 for a win, 0 for a loose.
int	**getTurns**() Number of turns left of the current player
boolean	**inStartArea**(int index, int color) If brick corresponding to color and nr are in starting area.
boolean	**isDone**(int color) If all bricks of a particular color is home(game completed)
boolean	**isGlobe**(int index) if index is a globe
boolean	**isStar**(int index) if index is a star
void	**kill**() Kill the current game
boolean	**moveable**(int nr) If a particular brick may be moved.

boolean	**moveBrick**(int nr) Move one of your bricks numbered from 0-3.
boolean	**nothingToDo**() If any of your bricks may be moved.
void	**paint**(java.awt.Graphics graphics)
void	**play**() Start a game between four players
void	**print**(java.lang.String str) Print something to the graphical interface.
void	**reset**() Resets the ludo board, the dice, the brick positions and points.
int	**rollDice**() Roll the dice to get a random number between 1 and 6.
void	**setPlayer**(LUDOPlayer player, int color) Let a specific color of bricks be controlled by a given ludo-player
static void	**trap**()

As an example of how to use, the class see some of the included LUDOPlayers.
The play() method from RandomLUDOPlayer is shown below, "board" is of the type LUDOBoard.

```
public void play() {
    board.print("Random player playing");
    board.rollDice();
    int nr=-1;
    double best = 0;
    for(int i=0;i<4;i++) // find a random moveable brick
    {
        if(board.moveable(i)) {
            double temp = rand.nextDouble();
             if(temp>best) {
                 best = temp;
                 nr = i;
             }
        }
    }
    if(nr!=-1) board.moveBrick(nr);
    //else nothing to do - no moveable bricks
}
```

Index of board

In the simulator, many methods require the use and understanding of indexes, the image below shows how the indexes are mapped onto the LUDO board.