Parth Kalkar - 05.06.2000

The first practice on Game Theory

Duplicator-Spoiler Game:

This is a finite position game between Spoiler and Duplicator as in the theory test.

1. Positions of the game are integers in [1. . (*day* + *month* + *year*)], and the players alternate turns (Duplicator-Spoiler-Duplicator-Spoiler-etc.).
2. Moves (for both players) are (+*n*) where *n* ∈ [1. . (*day* + *month*)] (within the admissible range [1. . (*day* + *month* + *year*)]).
3. A player wins and the game stops as soon as the player moves to the final position (*day* + *month* + *year*).

To solve a game *G* means to compute sets of positions W_D (W_S) where Duplicator (Spoiler) has winning strategies.

Game Logic:

$D -> day, M -> month, Y -> year$

a. A player always wants to end up on a number where another player can't move to (D + M + Y) in one move, i.e $(D + M + Y)$ % $(D + M + 1)$.

b. If a player can move from their position to a number that's equivalent to $|D + M + 1|$, then the desired position cannot be reached in one move.

c. If a player can maintain a sequence of these numbers, then they win.

Winning Strategy:

We just compute a set of winning positions, and each player will try to step in a non-winning position for their opponent.

The Duplicator has a unique winning strategy that depends on the remainder of $(D + M + Y)$ % $(D + M + 1)$.

Manual:

The program is an interactive number game. It discards invalid inputs like a string or a number outside the expected range (the user will be notified of that and the invalid input will be ignored).

Initially, the program asks the user to choose whether to start from a random position or to specify one. In both cases, the program won't allow the starting position to be winning.

Then the user is asked to choose the playing mode for Spoiler. After that, the game will begin.

At the start of the game, the program prints out the current position (and advice in the *Advice* mode), and then asks the user to enter a new position. Then it states the move of the Spoiler.

Note that the input should be a position, *not* the number of steps. E.g. in order to move from 1040 to 1070 the user should enter 1070, *not* 30. Additionally, the user may input **-1** to stop the game and return to the set-up phase or **-10** to shut down the program.

This process continues until one of the players wins or you stop the game. If the user moves to the final position, the game will print that they won. If the Spoiler has moved into the final position, the game will print that the Spoiler won.

If one of the players wins, the game will save a history of the moves in a file called "log.txt" in the directory where the program was launched. If the program was shut down, the file won't be created.

Technical:

The program has two classes: *modes* and *Bot*. *Modes* are defined as enum containing *Smart*, *Random* and *Advisor* with the values 0, 1 and 2 respectively. Most of the functionality is implemented in the *Bot* class. A constructor is defined which is responsible for creating the winning array and assigning the value to the mode variable. Winning is a boolean array that tells the bot whether the i-th position is a winning one or not. This class has 3 functions.

a. *backward_induction()* function. It computes the winning array by calculating the answer to all the points that can reach the final position using limited number of moves. It works recursively - the function returns an integer representing the boundary where all the values of the winning array to the right of the boundary are calculated. The function then calculates the winning value for each point that can reach the points to the right of the boundary using 1 move.
b. *advisor()* function. It tells the user what is the winning move, if the bot is playing in the Advisor mode. In other cases, it does nothing. Therefore, it is called from the main function.
c. *get_move()* function. It plays a random move in the admissible range while playing in the Random mode or if there's no winning move for the bot. Otherwise, it will look for a losing position in the admissible range and choose that, so it would make the user lose.

get_num() function is used to take an integer input from the user and return it.

Demo Video:

Link