1. Introduction
It seems that the hyper-intelligent pan-dimensional beings have taken to ripping off 90s era steampunk sci-fi stories for the purposes of human testing. The reasons they do this can only be described as sinister from our humble, puny, earthly perspective but it must be another step in the quest for the secret to the life, the universe, and everything. You have been tossed into this unforgiving environment and now your only goal is to beat the adversary, a computer program named Arthur. But you’re also still pretty lazy, so you want to do it without expending a lot of effort. To do this, you will implement various adversarial search algorithms to find optimal policy with which to arrive victorious, so you can go back to doing absolutely (hopefully) nothing.

2. The Game
You will be writing programs to control adversarial players in TRON. TRON is a two player game played on a grid, in which once a turn, both players move in one of the four cardinal directions (north, south, east, and west) and leave behind a barrier in the position where they were. A player loses by colliding with a barrier. An example game board is below:

```
###############
#1 # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
# # # # # #
#2 #
###############
```

This game board is empty and is the default game board that you will get upon using the command `python tronEngine.py`. Player 1 starts in the upper left hand corner (the 1),
and player 2 starts in the lower right hand corner. A player cannot move beyond a wall. It is important to note that players move *simultaneously*, so games can end in a tie if they are on a collision path. You can see two players playing on this empty board with the wall following policy by running the command `python tronEngine.py -AIs wall wall`. Here’s an example board a few moves after the start position given above:

```
#################
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##   #1       ##
##            ##
##            ##
##            ##
##            ##
##            ##
#   2##########
#################
```

In this case, player 1 has moved repeatedly down and player 2 has repeatedly moved to the left. If they continue to do so, they reach this end position (a tie, since they both hit an obstacle at the same time):

```
#################
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
##            ##
#################
###
```

This game can be seen as a two player version of the Atari game Snake. More game boards are located in the `maps/` folder. Further command options (including visualizing boards with a GUI) can be seen by executing `python tronEngine.py -h`. 
3. The Code

The stencil code this assignment is located in /course/cs141/stencil/adversarial_search. To get the code to your current directory, use the command:

\[ \text{cp -r /course/cs141/stencil/adversarial_search .} \]

**tronEngine.py** This contains the main code for running the actual game. Try python tronEngine.py -h to see the commands relevant to this file. You should not need to execute any other Python file. It also contains several very useful utility functions you’ll most likely need when coding. Relevant functions:

- **move_player** takes a board object, a player object, and a direction and moves them in that direction, if possible.
- **simulate_board_positions** takes a board object and a list of actions and removes an intermediate list of board states from a given action. It does not mutate the original board object.
- **game_status** takes a board and returns the game status, which indicates whether the game has ended, and if so, whether it has ended with a tie or a win (see comments in the function for more details on return types)

**tron.py** Utility functions the main objects for the TRON game. It might be useful to directly use some objects and constants but that’s optional. They are as follows:

**DIRECTIONS** A constant containing a four tuple of the directions used in this game, NORTH, SOUTH, EAST, and WEST.

**Player** The player object, which has three variables: coords, which is the coordinates in (x, y) in a two-element array; is_alive, which is a boolean value denoting whether or not the player is alive; and symbol, which is the character used to display the player.

**Board** The main board object. The relevant methods are moves, which returns the moves that are safe to make (important) from a specific coordinate, and print_board which can be useful for debugging.

**support.py** Contains support code. You should not need to use functions or variables in this file.

**randbot.py** Contains an implementation of a bot with a random policy.

**wallbot.py** Contains an implementation of a bot with a wall-hugging policy.

**studentbot.py** This is where you should implement the majority of your code, i.e., your adversarial search algorithms. By default contains the wall following bot.
4. What You Need To Code

This time around you only need to code one (external) function in studentbot.py: which_move. This function takes a board state and a player and returns what move the player should make. Since you need to implement Alpha-Beta pruning, it is recommended you also implement separate functions for your scoring heuristic, mini-max, and the actual pruning.

5. Rubric

This assignment is designed to get you to think harder about developing heuristics, and to play around with adversarial pruning algorithms.

The difference between this game and what you did last week is that there are other players making decisions at the same time as you, and you don’t know what their decisions will be. This means that it is much harder to accurately predict what the map will look like after each of move, and this has some implications for how to implement an AI to win this game. You will be graded on the following things:

First, your bot will need to reliably beat the random bot and wall bot provided. This will be tested in the following environments:

- The empty default board.
- An empty room where they start next to each other in the center.
- A large loop with lots of jagged walls of obstacles (like in maps/apocalypse.txt).
- Two separate rooms for each player (to test how efficient their movement is when unimpeded by another player).

You can make new maps in the maps folder to test these functionalities. This will be worth 80% of your grade. The rest of your grade will come from your implementation of alpha-beta pruning, which will be graded by number of expanded nodes.

6. Handin and Grading

You can install the assignment just like assignment1, except the code is now called Tron. There’s no written part to this assignment. When you have finished the code, go to the main directory (Tron/) and use the command cs1410_handin Tron.

If you see “no command found,” this means the command is not in your path, in which case use the absolute path /course/cs141/bin/cs1410_handin Tron. If you still get an error which says you’re not enrolled, please email the TA list (mailing address on website) with your CS login and banner ID with the subject header “CS1410: Not Enrolled [CS login] [Banner ID].”