

This project is worth 15 percent of your final grade. There are three problems available to be solved. You may work either individually or in teams of two. If you work individually, you must submit two of the three problems (7.5 percent each) and if you work in a team, you must submit all three (5 percent each). Grades for each subproject will be based on 75% for the write up, implementation and the number of AI techniques you used; the other 25% will be based on the performance in the competition.

Each project will be pitted in a competition with other classmates' implementations to see which is the best. The winners and some runner-ups of the competitions will be awarded some extra credit.

For Tetris and TSP, your solutions will be filtered through a domain-checker that checks your output and calculates its score. More information on these will be available when they are released. For the lightcycle game, a server program will be released.

You must provide a write-up that explains your program in depth. It must talk about what AI techniques you used. You must cite your references. Also, you should talk about any special optimizations you made. I expect that each write up should be at least two pages in length (single spaces, 10pt font). If it is not, you probably didn't do enough. If you would like to discuss with the instructor your strategy and if it is "enough," feel free to go to his office hours. From your write up, anyone should be able to read it and reproduce your work.

It is your responsibility to make sure your program works with the checker programs. If for some reason you think your work doesn't work with the checker because of a bug on the instructor's part, submit a bug report via email. It will be unacceptable if your program doesn't work flawlessly with the checker programs on presentation day.

1 Tetris

In this domain, you are to decide how to place 4-block pieces in a Tetris world that is 8-blocks wide. For each piece, you will output how many times to rotate it clockwise and which position to drop it. You are giving all pieces in order, in advance.

The pieces are as follows:

I	J	L	O	S	Z	T
	J	L	OO	SS	ZZ	T
IIII	JJJ	LLL	OO	SS	ZZ	TTT

The goal of your AI program is to plan the positions of the pieces such that you cleared the most lines. Remember, a cleared line is when an entire row is filled. When a line is cleared, the entire board is shifted down. Note, there is no *recursive gravity* that makes blocks fall into open spots below. In the case of a tie, the resulting height will be used as a tie-breaker.

The position of any piece, no matter how many times rotated is determined by the bottom left corner. The position is given by a number 0-7 to specify the index of the column. For example, dropping piece I after rotating once at position 1 will yield a board looking like:

```
|.....  
|.I.....  
|.I.....  
|.I.....  
|.I.....  
|.I.....  
+-----
```

The input will have two data items: the number of pieces that you must deal with as well as the list of pieces in the order they are given. For example:

```
20  
IJJLOSZTTSIJJLOSZTTS
```

For each piece, you are to output to standard out (or a file) the number of 90 degree clockwise rotations (0, 1, 2 or 3) and the column in which to drop it, separated by a single space. Drops must be legal (e.g., you cannot drop a T in column 7). Each piece drop is to be separated by a newline. Note, that you can use standard error or other means of output if you would like to show any other information about your algorithm, as long as you don't use standard out. Example output:

```
2 5  
3 7  
0 7  
1 0
```

As a general rule, your program should not run longer than .25 seconds multiplied by the number of pieces. For example, if the run has 100 pieces, your program should run in less than 25 seconds. This will not be strictly enforced, but should be followed as well as possible. You should expect problems up to 2000 pieces (but should be able to handle more).

2 Tron lightbike game

In the game, you move around in a 30x30 two-dimensional grid world. Your lightbike will either start at (4,4) or (25,25). You move around this space and the previous block your bike was in is “blocked”. If you ever cross the path of your opponent or your previous path, you lose. The same for your opponent. Also, if you ever move outside the bounds of the board, for example (30,29) or (-1, 0), you lose. In the case that both players crash into something in the same time step, the game results in a tie.

For each time step, you and your opponent will select as a move:

- NORTH ($y += 1$)
- SOUTH ($y -= 1$)
- EAST ($x += 1$)
- WEST ($x -= 1$)

For example, if your lightbike is at (5,5) and you move West, your lightbike will be at (4,5).

Your program should retrieve a IP address and a port number that represents the game server to connect to. Your client then connects to this server and sends the server your team name as a string with no spaces. Once both players connect, the server will send down two x,y tuples: the first is your position and the second is the other player’s position. Then, you will send your move as the string NORTH, SOUTH, EAST, or WEST. Once both players have submitted their moves, the server will again send down two x,y tuples: your position and the opponent’s position. Once the game is over, the server will let you know if you won or lost (WIN or LOSE or TIE). An example of a conversation between a client and a server, from the perspective of the player:

```
Player: TeamDonMiner
Server: 0 0 24 24
Player: NORTH
Server: 0 1 24 23
Player: NORTH
Server: 0 2 23 23
Player: SOUTH
Server: LOSE
```

It is very important you follow the protocol, as your agent wont interface with the server if it does not. You must not make any assumptions about the start position.

The competition for this project will be ran as a tournament.

Also see: <http://www.youtube.com/watch?v=-3ODe9mqoDE>

3 Traveling Salesman Problem

Your task in this project is to find a single short path from city to city so that each city in a 2-dimensional space are visited exactly once no matter where you start. Better solutions are shorter. Distance is measured by the standard Euclidean distance between two points. Therefore, the length of your path is the sum of the euclidean distances between cities in your path. More information about TSP is available from various resources (it is a very popular problem).

As input, your program will receive a list of city locations and a number specifying how many cities will be given. You may assume the points are positive and under 30,000. For example:

```
5
23.4 109.73
0.4 192.1243
1.1 193.13
10343.13 0.00001
111.3 41.013
```

Your program will give the path in terms of the index of which you saw the city during input. Separate them by spaces. For example:

```
3 2 1 4 0
```

You are expected to tweak your algorithm or try several different options to figure out what works best.

Your program will have two minutes to run and should handle up to 1,000 nodes. No matter the number of nodes, your program will have 2 minutes to run.

The winner will be determined by several different runs on different sets of cities. The shortest paths will be considered the winners.