Data Structures for Problem Solving (20112)
CS 242 Week 7 Cable Car Project, Parts 3 and 4

When you finished part 2 of the CableCar project you had a player module that could play the game without breaking any of the rules, as long as yours was the only player. That last condition is now gone. Here is the small set of things you must change for part 3.

- Your player module must handle any number of players, from 1 to 6, including yourself.
- Your player may be in any position from first to sixth.
- When the engine calls your `game_over` you must report some game results.

As before, your player must not crash.

Here are the details.

For a multiplayer game, there are two main pieces of functionality you have to add. `move_info` has to be prepared to get information on multiple players' moves. Presumably your player data would store that information. Second, keep in mind that there are 32 stations on the board, so how they are assigned to players is a bit tricky. Consult the official game rules for how this is done.

You need to know which player owns which station so that your module can report the game results. Inside your `game_over` function, print directly to standard output the information in the format shown below.

```
Player 0: score = s0
Player 1: score = s1
:
:
Player n: score = sn

Player #p has won.
```

or

```
Players #p,#q, ...,#z have won.
```

if there is a tie.

Of course, at the end of the game, all paths are complete, so each player's score is the sum of the lengths of the paths emanating from that player's stations. The winner has the highest score. Since we count from zero, n is one less than the number of player modules specified in the configuration file.

You may assume that no player has been kicked out of the game.

Here are some small notes that affect your implementation.

- You may remove the "*if part 1*" section from your `game_over` function. It is no longer needed.
- There is a rule that does not allow completed routes of length one when there are alternatives. You should enforce that rule even at stations that are not owned by any player. This happens when the number of players is three, five, or six.

1 Problem-Solving Session (20% of part 3 grade)

Your instructor will group your team with another team in class today.

Answer the following questions.

1. Describe how you will keep track of which player is assigned to each station. Feel free to write descriptions in English or use diagrams or pseudo-code to answer this question.

2. `move_info` now gets called when other players execute moves.

   Imagine the following execution scenario.
   - A 3-player game has begun.
   - You are player ID #1 (second player).
   - All players' `init` functions have been called.

   What calls to your functions will be made during round 1? Make sure you list the calls in order, and for `move_info` calls, state which player ID will be specified in the `PlayerMove` parameter.

3. At this point you should separate your group into two teams. Fill out a separate sheet of paper with your thoughts on the following question.

   (a) What impact will multiple players have on your player design?

   (b) What are your initial ideas for a winning strategy?

2 Implementation for Part 3 (80% of part 3 grade)

Before you begin coding for this part of the project, go to `http://www.cs.rit.edu/~royale/downloads/` to download the latest version of the engine.

Refine the functions required for regular game. Those functions are `init`, `move_info`, `move`, and `game_over`. Again, remember that in part 3 your player still does not have to win games. It simply has to handle multiple players without crashing.

Zip your player subdirectory to a zip file (not rar, tar, 7zip, or any other format) called **part3.zip** and submit it to the proper dropbox.

No design document is required for part 3, but you are encouraged to maintain the one from part 2 with your latest design.

3 Implementation and Writeup for Part 4

In part 4 you add a strategy to your player module that you believe will help it win more games.

For full credit, your player module has to beat the built-in player called `BadComputer` at least 90 times out of 100 two-player-mode trials. How your player fares in games with more than two players does not influence your grade.

You have one additional week to complete part 4. There is no problem solving session.

Again, make sure that you now include all team members' names in all files; each member is responsible for all of the project code!

Create a plain-text design file called **design.txt** containing the following information.

- Your team name
- Each team member's name and account [aaa0000 format]
- A description of the strategies you employed to help your player module win the game
- Descriptions of changes to data structure design in your part 4 software
- Descriptions of changes to algorithms in your part 4 software

Place the **design.txt** file into your team's player directory, alongside the code.

When you want to submit (and you should do so several times in case something goes wrong near the end),

1. Choose which member of the team will be the one to submit the project deliverables. Or, if your instructor created "group drop boxes" in mycourses, make sure you can locate your team's dropbox.
2. Zip your player subdirectory to a zip file (not rar, tar, 7zip, or any other format) called **part4.zip** and submit it to the proper dropbox.

4 Retrospective

There is a report you must fill out about your project, its design, and your strategy. It is available on line in the Survey section of mycourses. It is due Friday of week 10.

NB:
Remember that you can recover functionality points for parts 2 and 3 as long as you had submitted something that demonstrated minimal effort by the original due date.