Data Structures for Problem Solving (20112)
CS 242 Week 4 Cable Car Project, Part 2

Part 1 of the project was an assignment that got you familiar with the game and had you create program constructs you would need to build a legitimate player module. In this second part of the CableCar project you will put that work to the test by building a player that can act as a "good citizen" in the regular game, as long as it is the only player.

Your code will not have to be aware of other players. Therefore the requirements are limited to successfully finishing an entire game. "Successful" means

- All the tiles have been placed.
- Your player module did not crash, i.e. raise any exceptions that caused the engine to eliminate it from the game.
- Your player module did not attempt any moves that are against the rules of the game. This would also cause the engine to eliminate it.

1 Problem-Solving Session (20%)

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

Read the copy we printed for you of the Documentation at the Cable Car project web site for Part 2. It gives details on the "API" (application program interface) you must implement. It will also give you a few helpful tips.

You may want to review the game rules as well. Ask your instructor for a copy of them if necessary.

Then answer the following questions.

1. What type of data is the player ID?

2. In a one-player game, what will the ID of your player be?

For questions 3-7, note that the moves listed do not succeed one another. **Each move in each question starts with the same board mentioned in that question.** The figures are handed out separately.

3. Figure 1 shows an empty board. For each of the following `PlayerMove` objects state whether the it would be considered a legal move, and if not, why not.

   a. `PlayerMove(0,(0,0),'d',0)`
   b. `PlayerMove(0,(0,0),'g',0)`
   c. `PlayerMove(0,(0,0),'h',2)`

4. Figure 2 shows a board with only a few tiles on it. For each of the following `PlayerMove` objects state whether the it would be considered a legal move, and if not, why not.

   a. `PlayerMove(0,(5,6),'a',0)`
   b. `PlayerMove(0,(6,6),'a',0)`
   c. `PlayerMove(0,(7,6),'d',2)`

5. Figure 3 shows a board halfway through play. For each of the following `PlayerMove` objects state whether the it would be considered a legal move, and if not, why not.

   a. `PlayerMove(0,(0,0),'b',0)`
   b. `PlayerMove(0,(0,0),'b',1)`
   c. `PlayerMove(0,(1,1),'b',0)`

6. Figure 4 shows a board with only a few open spaces left. For each of the following `PlayerMove` objects state whether the it would be considered a legal move, and if not, why not.

   a. `PlayerMove(0,(0,2),'g',0)`
   b. `PlayerMove(0,(0,7),'h',0)`
   c. `PlayerMove(0,(5,3),'f',0)`

7. Figure 5 shows a board with very few spaces left on it. For each of the following `PlayerMove` objects state whether the it would be considered a legal move, and if not, why not.

   a. `PlayerMove(0,(7,5),'g',0)`
   b. `PlayerMove(0,(7,7),'h',2)`

8. Give a high-level ("English") description of the algorithm you intend to use to generate a legal move. Include a description of how you would scan, or iterate over, the board looking for a place to put the tile. Indicate how it would integrate with some kind of testing function that tells you whether or not a move choice is legal, but do not write details of that testing function.

Remember that your work in this question is independent of any strategy you may eventually devise to win the game.

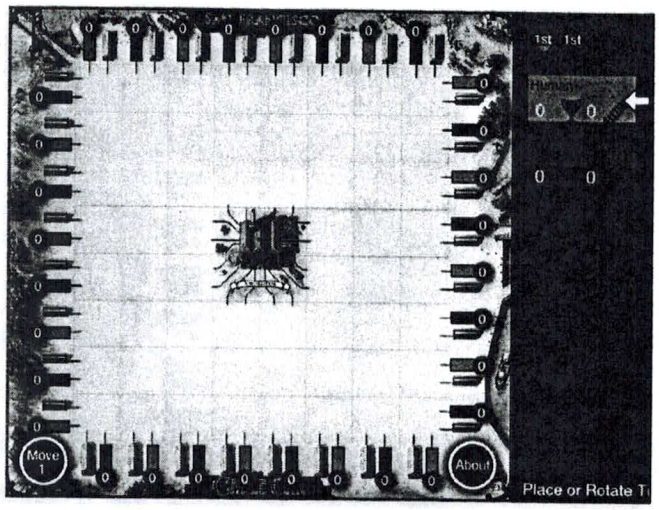

Figure 1: Question 3

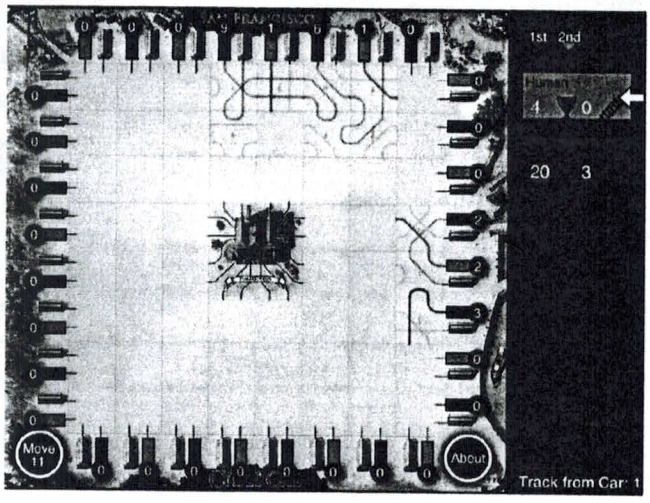

Figure 2: Question 4

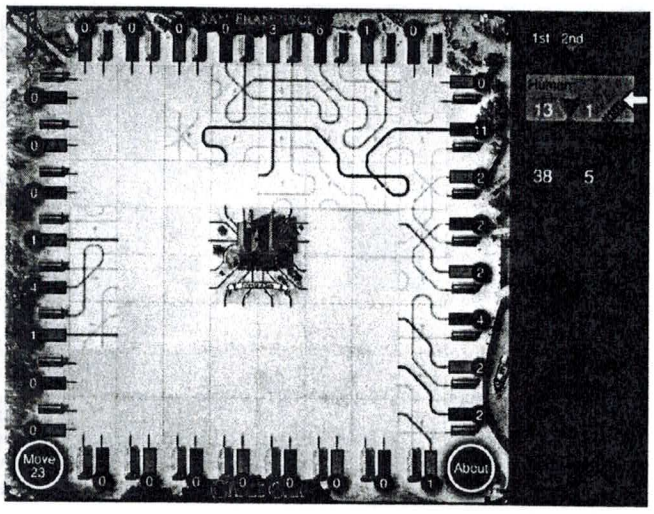

Figure 3: Question 5

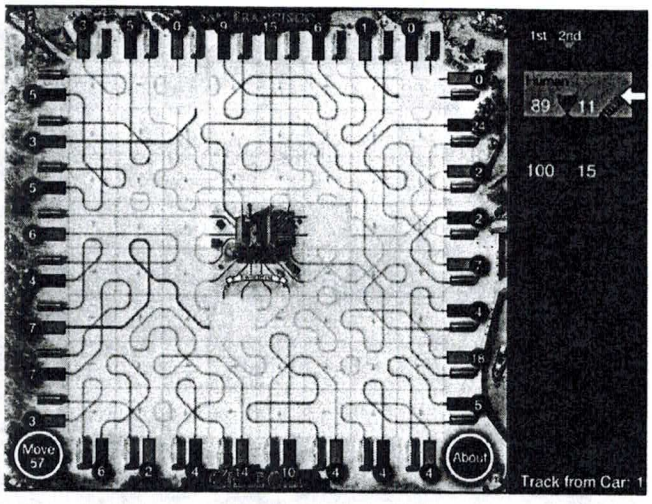

Figure 4: Question 6

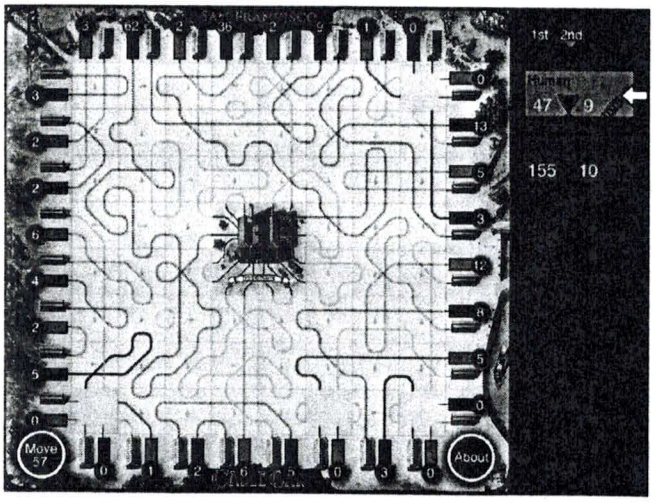

Figure 5: Question 7

2 Implementation and Writeup for Part 2 (80%)

Choose a name for your team. In your copy of the game code, create a new Player with the the directory's name being your team name. Your source code will be placed in this new directory.

Following the outline code you already found in the `Players.Student` package (**src/Players/Student** directory of the project), build the functions required for regular game. This includes enhancing `init` and `move_info`, and writing `move`. Again, you can assume in `init` that `numPlayers` is 1.

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]
- Which team member's design was used as a starting point (Or indicate that a new design was created.)
- Descriptions of state structure descriptions, including changes, for part 2
- Descriptions of algorithms used, include changes, for part 2
- A summary of the testing you did

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 from now on.
2. Zip your player subdirectory to a zip file (not rar, tar, 7zip, or any other format) called **part2.zip** and submit it to the proper dropbox.

3 Grading

- Problem Solving: 20%
- Design and Implementation: 80%
  - Submission instructions followed: 5%
  - Functionality: 55%
  - Coding Standards: 5%
  - Design document: 15%