Objectives

This is stage #1 of the course project. The main concepts covered are:

- software development in a small team setting,
- development of a design for a medium-sized software project, and
- building a suite of unit tests.

Overview

You have been employed by a small game manufacturer named *KnowSys*. This company has more programming talent than game-creation talent. Its developers create on-line versions of popular board games. Since the games this company develops cannot be exactly the same as the games on which they're based, the games produced are always extended and changed in interesting ways (from a programming perspective).

The game you will write is a variant of a game called Metro. Metro was designed by Dirk Henn, and was published by Queen Games. A description of the original game can be found at
http://www.queen-games.de/index.php?id=16760----2
If you click on the downloads link at the bottom of the page, you will be taken to a download page from which you can download the rules in your choice of language.

The requirements will be revealed to you in several stages. This document describes the first of two stages. You should not make any assumptions about what the second stage might require you to do. Also, if you find that the requirements are vague or underspecified, you should make no assumptions about how to resolve such vagueness or under-specification. Instead, you should ask your customer, who prefers to take questions via e-mail to <alphonce@buffalo.edu>.

Description

You and your teammates must produce a text-based game (no graphical elements allowed) implementing the simplified rules spelled out below. The code must have a clean separation between the game logic and the text-based user interface. The code must be unit tested, but the unit tests must be of the game logic only, not of the user interface.

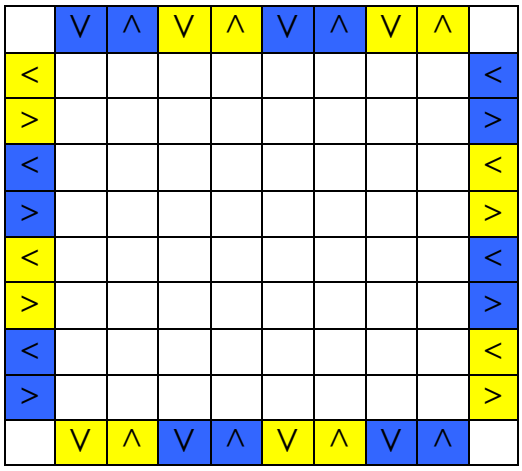
Rules

The rules are as follows.

The game is strictly a two-player game.

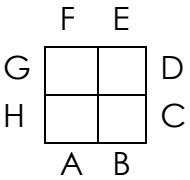
The game is played on a 4 by 4 board.

The board looks like this:

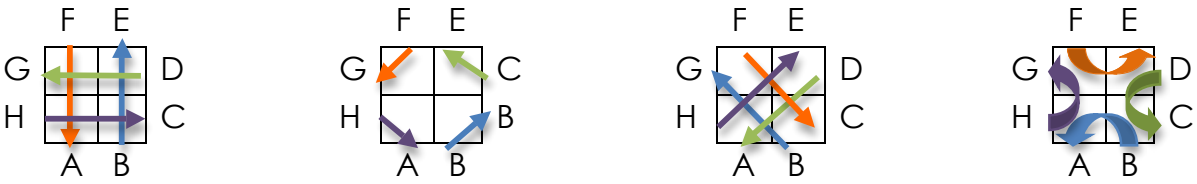


The arrows indicate the direction in which paths can be travelled. The colors indicate the station ownership, blue for player 1 and yellow for player 2.

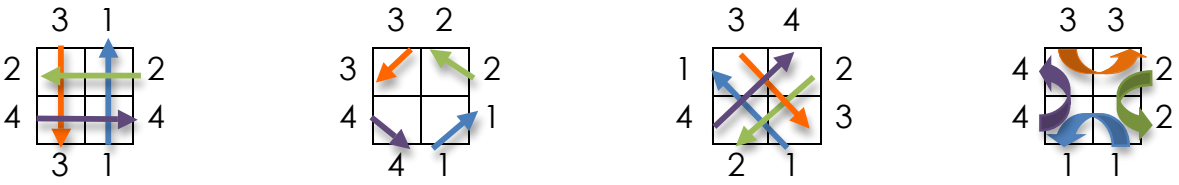
The game has 16 square tiles. On each tile there are four track segments. Each tile in the game has eight "connection points", points where tracks can connect. We can depict a tile in a textual way as follows, where the letters uniquely identify a connection point.



The following set of tiles will be used, four of each for a total of 16 tiles:



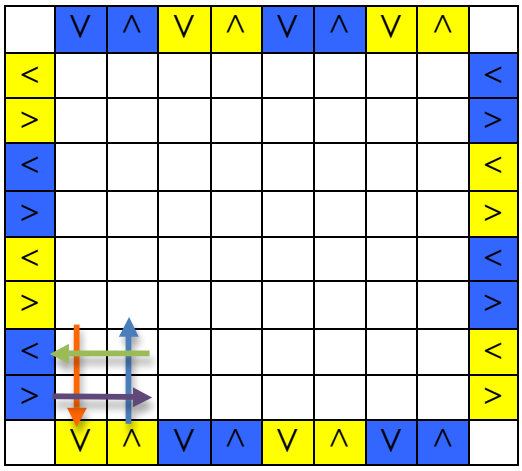
The arrows indicate the track segments on each tile. Textually we can indicate the paths by digits (1, 2, 3 and 4), and can show how the paths are connected to the connection points like this:



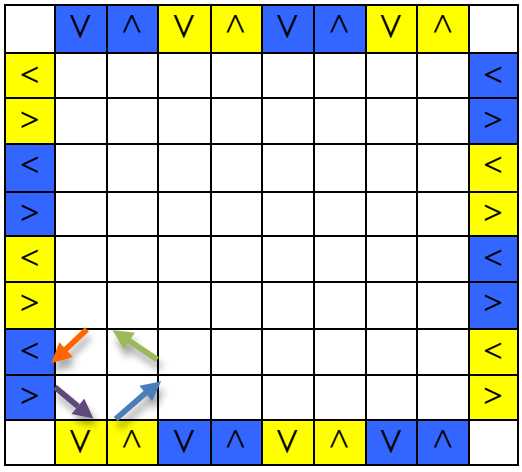
Paths are formed by connected track segments. A path is closed when it connects two stations. A path which is not closed is open. A player gets points for each closed path that starts from a station they own. The number of points for a closed path is one per track segment in the closed path. Open paths are worth no points. The only placement constraints are that closed paths cannot consist solely one track segment, *unless there is no alternate placement which does not violate this constraint*.

Sample placements

Here's an example of a legal placement:



Here's an example of an illegal placement (it is illegal because it creates a closed path consisting of just one track segment, along the purple track):



Players alternate turns. On a turn a player places a randomly drawn tile from the still unplayed tiles. Play continues until the last tile is placed. Players' scores are updated after each turn. The player with the highest score wins.

Textual User Interface

In a textual user interface a user types commands which are interpreted by the program. Exactly the following commands must be understood by your program; if any other command is entered, your program must display "I'm sorry, I didn't quite catch that. Try again!" and then accept another command:

resign – End the game and exit the program.

place(column, row) – Place drawn tile on board in the indicated column and row. If the tile is placed in an illegal way the user must be so notified, but the tile must be left of the board. See description of illegal placements above.

remove – Remove the tile placed on this turn. Remove is the inverse of place, and can only be done by a player after placing a tile on their turn, but before committing.

commit – End the current player's turn. This command must be permitted only if the player has indeed placed their tile in a legal way on the board.

display score – Display each player's current score, as in:

```
Player 1: 3 points       Player 2: 5 points
```

display player – Display the current player, as in:

```
It is player 1's turn.
```

display tile – Display the tile drawn on this turn. The tile display must show how the eight connection points (two on each side of the tile) connect to each other, as shown with the digits in the tile samples above. Thus, the four tiles must be displayed as follows:

```
 31        32        34        33
2  2      3  2      1  2      4  2
4  4      4  1      4  3      4  2
 31        41        21        11
```

display board – Display the whole board, including the borders, with tiles displayed as in the display tile section. An empty board must be display like this:

```
  AB  AB  AB  AB

G                 D
H                 C


G                 D
H                 C


G                 D
H                 C


G                 D
H                 C

  FE  FE  FE  FE
```

The edges are treated as though there were tiles there, but showing only the edge that touches the board. Along the edges the connection points are shown. As another example, the board sample with the valid tile placement above must be displayed as follows:

```
  AB  AB  AB  AB

G                 D
H                 C


G                 D
H                 C


G                 D
H                 C


G                 D
H                 C

  31
G2  2             D
H4  4             C
  31
  FE  FE  FE  FE
```

Gameplay and turn start

At the start of a player's turn the following information must be displayed:

- the current board configuration (same information as given by 'display board')
- who the current player is (player 1 or 2) (same information as given by 'display player')
- each player's current score (same information as given by 'display score')
- the currently drawn tile, which is drawn by the program from the currently available inventory of tiles at random (same information as given by 'display tile')

Gameplay continues until all the tiles are placed on the board, or one of the players resign. At the end of the game the scores of both players must be displayed, and the winner must be announced. A player who resigns automatically gets zero points, and is considered the loser (even if the other player also has zero points). Otherwise the winner is the player with the higher score.

Grading

Your team's project submission will be graded on its merits as follows:

Functionality: 60%

> As described above. Do not add functionality that is not listed, unless instructed otherwise.

Documentation: 10%

> You must have a brief user's manual which described how to start and how to play your game (including the commands/rules AS THEY WORK IN YOUR GAME). In other words, if your game does not support the resign command, say so!) Name your user's manual User.txt (it must be a plain text file).
>
> You must also have javadoc comments for each class and each public method which says WHAT the class or method does, not HOW it does it. For methods, the role and expected value for each parameter must be explained, and the return value must be documented too (describe the significance of possible return values).

Testing: 30%

> You are expected to have JUnit tests for the functionality of each public method you write. We are not aiming for perfection at this point, but if you write your code in a test-driven fashion you will have some tests for each such method.

Your grade is based on the overall team grade for the submission, and your peer grade. Peer grading will be discussed in detail in lecture. Basic point: if you do not contribute as much as others on your team you will not receive as much credit as your teammates.

Team meetings

To help you manage your time, here are some do's and don'ts for recitation:

• Do discuss accomplishments (goals met) since last meeting problems (goals not met) that have come up since last meeting goals for next meeting schedule for the next week for pairs to meet to pair program.

• **Do NOT even consider writing code during your recitation time unless you have done the above.** Remember, your recitation time is the only time during the week when you are guaranteed to be able to get together as a team. This time is precious - don't waste it.

• Do have someone take notes each meeting, and keep notes in your code repository. Each team **MUST** submit one set of meeting notes per week. For the first week of stage 1 your repository may not be set up yet. Once repositories are set up, put a copy in your repository, in a folder called "Minutes".

You must also e-mail me a copy of your team meeting notes. The subject line of the e-mail must be

[CSE116-T<xxx>] Minutes from meeting on <DATE>

where <xxx> is the team number, and <DATE> is the date in mm-dd-yyyy format.

• Do compromise.

• Do NOT exclude team members - use all your team's resources. Each team has people with different skill levels and talents. Coding prowess is just one of many skills needed to complete this project. **When pairing, strong and weak coders should work together, and the weaker coder should drive more often than the stronger coder (to help them build their skills with the help of the stronger coder, who navigates for them).** Doing this will pay off big-time in stage two of the project! Likewise, not doing this will cost you big-time in stage two!