Project 3

Due November 6, 2019 at 11:59 PM

This project specification is subject to change at any time for clarification. For this project you will be working with a partner. You are given the fix a given project that was developed by another developer. The project was a Six Men's Morris game, the rules will be defined below. The existing game will work, but it is extremely buggy. The developer would compile the project using manual calls to g++ instead of using a Makefile and didn't use test driven design. The developer also did not use revision control, so you will be adding revision control with git. You are tasked with creating a Makefile for the project that will run compile/link a test program and the project. You will develop a set of tests have check correctness for the game. Once you have created the tests, you will then fix the existing code. You have been provided with the source files, and a skeleton test file.

You will find https://www.tutorialspoint.com/makefile/ helpful in developing your Makefile. The requirements for you Makefile are:

- Only used Conventional Macros should be declared (e.g. CXX, but not AR)
- The tests must be successfully run prior to building of the game
- For each target all true dependencies must be listed
- The Makefile must have a clean to remove binaries, object files, core dumps, etc.
- A conventional naming scheme must be used to organize the project files (source files, object files, etc.)
- The Makefile must build and run the tests and Six Men's Morris game on the CSIF without modification

You need to create a git repository for your project. You should initialize it on github and then after cloning add the project files using the directory structure you have chosen. The following video https://www.youtube.com/watch?v=HVsySz-h9r4 has a very helpful tutorial on using git command line tools. You should commit progress as you make it and use branches so that you and your partner can develop in concurrently. You will include your .git directory in your submission. If you have not already gotten a github student developer pack, you should follow the directions here https://help.github.com/en/articles/applying-for-a-student-developer-pack. This will allow you to develop using a private repositories.

Six Men's Morris is derivative of Three, Nine, and Twelve Men's Morris, also known as Merelles or Mills. This game that dates back to the Roman Empire and has some variations of rules. The following rules and board setup will be assumed for this project:

- The board has sixteen positions in which pieces can be placed. The positions are arranged in two squares of eight positions with an inner and outer square connected at the midpoints of the lines (see Figure 1)
- Each player begins with six unplaced pieces. The game consists of two phases, a placement phase, and then a movement phase.
- Placement Phase
  - Players take turns placing pieces on empty positions.

- If a player is able place three pieces in a row (either horizontal or vertical), they have formed a *mill*.
- When a player forms a *mill* they then remove one of the opponents pieces. In this version any of the opponent's pieces can be removed.

- Movement Phase
  - Players take turns moving pieces to adjacent empty positions
  - Pieces may not "jump" other pieces
  - If a player forms a *mill* they then remove one of the opponents pieces. See below for example.
  - A player wins if their opponent is reduced to two pieces, or if their opponent has no legal moves (see Figure 4).

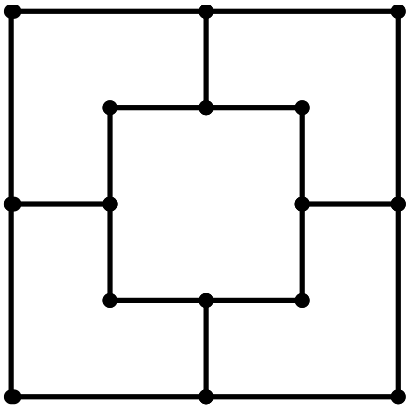

Figure 1. Six Men's Morris Board

Below shows moving of the dark piece from outer left center (see Figure 2) to the upper left corner (Figure 3) in which a mill is formed. After this the dark player would remove one of the lighter player's pieces.

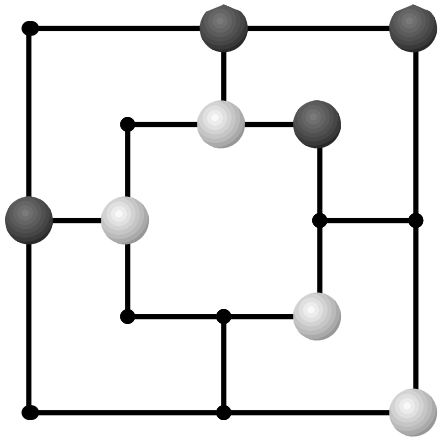

Figure 2. Board Before Move

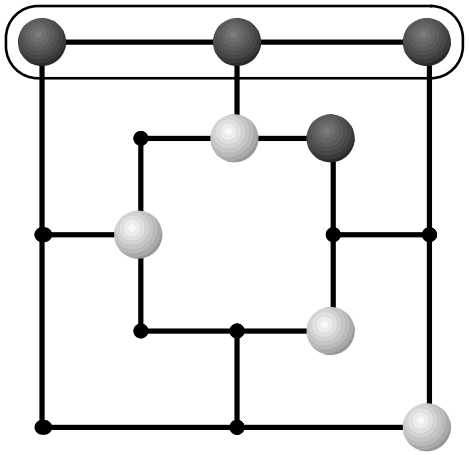

Figure 3. Board After Move with Mill Formed

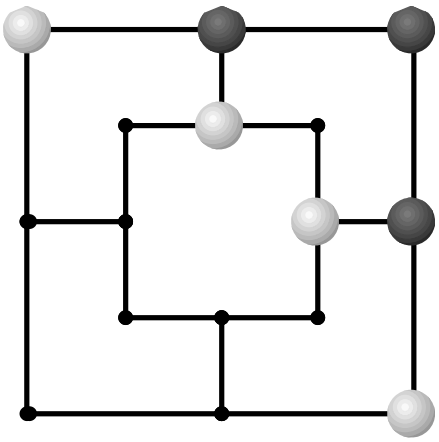

Figure 4. Dark is Blocked by Lighter Pieces and Has no Valid Moves

The Six Men's Morris game is implemented almost entirely in the `CSixMensMorrisBoard` class. The has `CSixMensMorrisBoard` the following member functions:

```
// Default constructor should initialize the board.
CSixMensMorrisBoard();

// Constructor to specify a board setup with the piece at each
// location, the player turn, the number of unplaced pieces,
// and the previous board setup.
CSixMensMorrisBoard(char turn,
const int unplaced[SIX_MENS_MORRIS_PLAYERS],
const char positions[SIX_MENS_MORRIS_POSITIONS],
const char previous[SIX_MENS_MORRIS_POSITIONS]);

// Resets the board back to initial setup.
void ResetBoard();

// Returns which player's turn it is.
char PlayerTurn() const;
```

```
// Returns the player at the particular position, '\0' should be
// returned on bad parameter.
char PlayerAtPosition(int position) const;

// Returns the number of pieces left for the player to place,
// returns -1 on bad parameter.
int UnplacedPieces(char player) const;

// Returns true if the game is over.
bool GameOver() const;

// >RU:6 RC:0  WU:6 WC:0
// o---------o---------o      0---1---2
// |         |         |      | 3-4-5 |
// |         |         |      6-7   8-9
// |    o----o----o    |      | A-B-C |
// |    |         |    |      D---E---F
// |    |         |    |        LEGEND
// o----o         o----o
// |    |         |    |
// |    |         |    |
// |    o----o----o    |
// |         |         |
// |         |         |
// o---------o---------o
std::string ToString() const;
operator std::string() const;

// Places a piece at the location specified by where. If the
// placement failed, or it is not the player's turn Place should
// return false. If the placement succeeds true should be
// returned.
bool Place(char player, int where);

// Returns true if the player can remove an opponents piece,
// otherwise returns false. Returns false on invalid parameter.
bool CanRemove(char player);

// Returns true if the player can move their piece located at
// position specified by where, otherwise returns false. Returns
// false on invalid parameter.
bool CanMove(char player, int where);

// Does a move for the player, by taking the piece from the
// from specified position and moving it to the to specified
// position. If the move failed, or it is not the player's
```

```
// turn Move should return false. If the move succeeds true
// should be returned.
bool Move(char player, int from, int to);

// Removes an opponent's piece from the position specified by
// from. Upon successful removal of the opponent's piece true is
// returned. If the player cannot remove the opponent's piece,
// or bad parameters are entered, false should be returned.
bool Remove(char player, int from);
```

You can unzip the given tgz file with utilities on your local machine, or if you upload the file to the CSIF, you can unzip it with the command:
`tar -xzvf proj3given.tgz`

You **must** submit the source file(s), the Makefile, .git fiels, and README.txt file, in a tgz archive. Do a `make clean` prior to zipping up your files so the size will be smaller. You can tar gzip a directory with the command:
`tar -zcvf archive-name.tgz directory-name`

Provide your interactive grading timeslot csv file in the tgz. The directions for filling it out have been posted.

You should avoid using existing source code as a primer that is currently available on the Internet. You **must** specify in your readme file any sources of code that you have viewed to help you complete this project. All class projects will be submitted to MOSS to determine if students have excessively collaborated. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.

Helpful Hints

- Read through the guides that are provided on Canvas
- See http://www.cplusplus.com/reference/, it is a good reference for C++ built in functions and classes
- You may find the following line helpful for debugging your code:
  `std::cout<<"In "<<__FILE__<<" @ "<<__LINE__<<std::endl;`
  It will output the line string "`In F @ line: X`" where F is the name of the file and X is the line number the code is on. You can copy and paste it in multiple places and it will output that particular line number when it is on it.
- Make sure to use a tab and not spaces with the Makefile commands for the target
- `make` will not warn about undefined variables by default, you may find the `--warn-undefined-variables` option very helpful
- The debug option for make can be helpful in understanding which targets need to be built, and which are not. The basic debugging can be turned on with the `--debug=b` option. All debugging can be turned on with the `--debug=a` option.
- Use a `.gitignore` file to ignore your object files, and output binaries

- Do not wait to the end to merge with you partner. You should merge your work together on a somewhat regular basis.