Final Escape Project

Due Mar 2 by 8am **Points** 150 **Submitting** a file upload **File Types** zip
Available until Mar 4 at 8am

This assignment was locked Mar 4 at 8am.

Make sure that you read the complete assignment specification.

This assignment is the culmination of the term. If you had success in the previous assignments, you should be in good shape for this final one. The grade for this assignment is 60% of the final grade as described in the syllabus. There are three levels of tests that we will run for this assignment submission. Each one corresponds to a letter grade. You must pass 75% of a level's test scores in order to "pass" that level and be eligible for the next level. The levels are described later in this assignment specification. Each level is specified by the features of the game that you must implement.

C-level requirements

Implement a game that can be played (i.e. moves made by the client) that has the following characteristics and features:

- Only boards with SQUARE locations.
- Boards may be infinite or finite on either or both axes.
- All pieces have the default value of 1. The VALUE may be omitted as described in the game Escape Manual's *Piece type attributes* section (p.9).
- Piece attributes that may appear in the configuration are VALUE (optional) and DISTANCE (required).
- Only OMNI and ORTHOGONAL movement patterns will be used in the game configurations.
- Locations will only be CLEAR or EXIT.
- A move may not be along a path that goes over an existing piece or EXIT location.
- All calls to **move()** will be valid moves.
- The only rule in effect will be TURN_LIMIT. If the game has not been won after the second player make the last move, the winner will be the player who has removed the most points from the board. If both players have removed the same amount of points the the game is a DRAW.
- If a player removes all of the pieces before the time limit, that player wins the game immediately when the last piece escapes.
- There should be no exceptions thrown from the **move()** method.
- The result of a call to **move()** is a **GameStatus** object with the following characteristics:
  - **isValidMove()** always returns true
  - **isMoreInformatin()** always returns false
  - **getMoveResult()** will return the status of the game after the move has been made.

    - **finalLocation()** always returns null. If the piece exited, then it is up to the client to know that it exited.

Your code will be reviewed for readability, intentionality, organization, and application of principles and patterns. We will also run your tests for code coverage. You should have at least 90% code coverage (branch and line) on code you write. You will also maintain the TODO list indicating your development using TDD. Organization is about how you have grouped classes and packages to show the organization of your design, i.e., where you think the major components of the game manager are. For example, you might have a package named **rules** where you have all classes that are used to apply the rules and check to see if they have been violated.

B-level requirements

In addition to all of the C-level requirements (you must pass 75% of the test scores before we run the B-level tests), you must also implement the following. Some of these may override or add to previous requirements.

- Add HEX locations to the game.
- HEX locations will have OMNI movement pattern only.
- Add DIAGONAL movement pattern for the SQUARE
- Pieces may have any positive VALUE. (default 1)
- Piece type attributes will be VALUE, DISTANCE, FLY, and UNBLOCK.
- Locations will be CLEAR, EXIT, or BLOCK.
- Invalid moves of the follow type must be checked for:
    - Trying to move from an empty location.
    - Trying to move onto a location containing another piece.
    - Trying to move onto a BLOCK location.
- A piece may not move across BLOCK locations unless its type has the UNBLOCK attribute.
- TURN_LIMIT and SCORE rules will be implemented. If there is no TURN_LIMIT, the game goes on indefinitely until one player wins. If SCORE is not in effect, then removing all players as described in the C-level requirements is applied.
- If a player cannot move, and still has at least one piece on the board, then that player loses. This should be determined after the end of the previous player's move.
- FLY may move from one location to another, regardless of any obstacles, as described in the manual. To clarify, the "valid path" would be a valid path on an empty board with the appropriate location type.
- The result of a call to **move()** is a **GameStatus** object with the following characteristics:
    - **isValidMove()** false only if an invalid move is attempted.
    - **isMoreInformatin()** always returns false
    - **getMoveResult()** will return the status of the game after the move has been made or an invalid move was attempted. If an invalid move is attempted then the opposing player wins.

    - **finalLocation()** always returns null. If the piece exited, then it is up to the client to know that it exited.

A-level requirements

In addition to the C and B-level requirements (you must pass at least 75% of the test scores in each of these, not just one of them, the following will be implemented. Some of these may override or add to previous requirements.

- All movement patterns will be implemented. See the table in the *Movement Patterns* section of the manual (p. 9)
- All piece type attributes will be implemented. See the table in the *Piece type attributes* section of the manual (p. 9).
- All rules will be implemented as described in the manual. Note: it is possible for all three rules to be used in one configuration. See **the manual for the updated description of the rules (https://canvas.wpi.edu/courses/43466/modules/items/893788)** . If SCORE is not in effect, then removing all players as described in the C-level requirements is applied.
- You will add the **addObserver()** and **removeObserver()** methods in your game manager implementation. The **GameObserver** interface is in the **required** package of the starting code. You do not have to implement any concrete observer except for your own testing. The client (test) will add or remove observers.
- The result of a call to **move()** is a **GameStatus** object with the following characteristics:
    - **isValidMove()** false only if an invalid move is attempted.
    - **isMoreInformatin()** If an invalid move is attempted, or a conflict occurs, or the game is over (WIN, LOSE, or DRAW), along with populating the GameStatus object, you will send one or more messages to observers describing the status. For example if player Chris wins, you might say "Chris wins by removing all pieces."
    - **getMoveResult()** will return the status of the game after the move has been made or an invalid move was attempted. If an invalid move is attempted then the opposing player wins.
    - **finalLocation()** always returns null. If the piece exited, then it is up to the client to know that it exited.

Grading

We will run the tests on your code to determine your correctness and implementation level. As long as you pass 75% of the tests scores, you have achieved that level, which means that is the grade ***that you could possibly get*** for the assignment. Out of the 150 points possible, 120 are for correctness. We calculate using a multiplier for each grade to calculate the correctness points these multipliers are A = 1.0, B = 0.9, C = 0.8, NR (not passing 75% of the C-level tests) = 0.65.

Here is an example that might help you understand how we will grade the correctness portion. Assume that there 100 test score points for each level.

> Student S gets 96 points for C-Level, 80 points for B-Level, and 50 points for A-level. We keep the C and B-level scores and take the average (96 + 80) / 2 = 88. We take the 88 points and multiply it by 1.2 (120/100) to get the points without factoring the multiplier. 88 * 1.2 = 105.6. Now we adjust using the multiplier. Student S achieved the B-level so we have 105.6 * 0.9 = 95.04.

Student S will get a grade of 95 + the points for coverage, readability etc., and TDD for the final project grade. If S received the full 30 points, the the grade would be 125.04 points for the project. This is approximately 83.4%, or a B.

Note: I reserve the right to raise the multipliers, but I will never lower them.

Submission

If you use an IDE, submit the complete project if you wish. However, we expect that there will be a **src**, **test**, **configurations**, and **TODO.md** file that is easily located. All of your work should be in these directories and file. We will take these and put them in an IntelliJ project, replacing the three directories and the TODO.md file. Your code should run with the libraries that were provided to you with the starting code. We are using JUnit5 testing framework. Do not use Windows specific path names. All paths to your test configuration files should be relative to your project and start with "configurations/" followed by the path to the specific configuration file in the configurations directory.

The submission should be zipped archive.

Project rubric

Criteria	Ratings	Pts
Code coverage At least 90% on the code you write (not counting enums, interfaces, and builder package code.		10 pts
Correctness Points earned by running the tests as described in the assignment.		120 pts
Radability, intentionality, organization, design You will receive 0, 5, or 10 points based upon the grader's assessment o your project.		10 pts
TDD You will receive 0, 5, or 10 points based upon the grader's assessment of how well you followed the TDD process This is based upon your tests, and the **TODO.md** **(http://TODO.md)** Links to an external site. file..		10 pts
		Total Points: 150