CSE 116 Stage 3: Code Review/Selecting Candidate.

Design of the Code (Modularity)
Readability
Code Documentation (JavaDoc)
JUnit Tests <Efficiency>

A: Dragon Team
B: MasterLabyrinth (Hard to play--inconvenience of action)
C: CSE group
D: Master Labyrinth
E: Stage_2

[Dragon Team]
Class packaging is fairly organized:

- No player class to hold the informations of player (tokens, magic wands). (i.e. Board class manages all informations that “Player” class should have) But the game is perfectly working.
- Can be felt like too much of codes, but fairly able to track the code.
- Some classes are misplaced in package (i.e. “gui.Player” class’ better package location should be “code” package.).

Class and Method modularity is easy to edit:

- Each functions are divided, and are organized well.

Code Documentation (comments) are well explained.

- Comments in important methods are well described to easily understand

JUnit Tests:

- Fairly covered the types of classes, but more number of tests are needed.

[MasterLabyrinth]
Design of the code:

- The design of the code is really unsatisfying, especially the control of the game is very complicated and inconvenient. The class organization is hard to understand that all of our team couldn’t understand of it.

Readability:

- The code looks readable but when it comes to the class GameApplet, the whole code becomes difficult to understand.

- What really good about the code is the way the Tile class is written. The method definitions look neat and are highly functional.
- However as one comes across the GUI, the whole code looks scattered. There is no differentiation in the code.
- The one main reason why we didn't choose this respective team is because we like to package all the related classes as one.

Documentation:

- There is hardly any solid Java documentation in the code. There are comments added at the start of each method.

JUnit Tests:

- The Board tests, examine every step of the process of populating the board, checking the tile position and making legal moves.
- The Card test works by checking if the cards are distinct. The whole condition is set under conditional statements.
- The MenuStateTest checks by adding different number of players and checking if the command line arguments work as expected.
- The Pawn Test checks if the pawn made legal moves or not.
- Finally the Token Test checks for the tokens and if they have the required respective icons.

[CSE group]

Design of the code:

- GUI: Designed a tile as 3*3 JButtons. No illegal functionality, but players may feel uncomfortable to play the game.
- Tile: Designed a tile class as abstract class, and its children classes. Convenient for calling a tile on the board. But defining rotation, and location of a tile is complicate to understand.
- Model: Code is not agile (i.e. the moveTile() method can be much simplified by creating sub-methods that support moveTile().). Therefore, the class is hard to edit, and follow.
- Pawn: By using arrow keys, a pawn move to adjacent tile one by one to reach the destination where player wants (i.e. no need to compute legal pawn movement to the destination tile.

Readability:

- A tile structure (3*3) and a board structure (7*7) in code are confusing to read.
- Board structure codes are hard to read.

Documentation:

- Most of classes contain documentation of their methods and constructors.

JUnit Tests:

- Board Test mostly covered a tile movement on the board about “rotation,” “move,” and “staying position.”
- More tests about tile movement of edge tiles are needed. Test that pushed-out tile sends pawn and token information to opposite side of board.
- Playermovement Test needs more test for pawn movement on the edges of board.
- TileTest tests the rotation of extra tile. This test is enough.
- TileMovementTest test that moved tile keep its types and rotation before it was pushed.

[Master Labyrinth]
Functionality:

- The program doesn’t interact with actions (no action listening, no key listening) when there are less than 4 players in command line argument.
- The game doesn’t have information panel of current state of the game when there are less than 4 players in command line argument.
- During pawn movement, if a pawn is overlapped on a tile where another pawn is standing, all pawns that had overlapped move together. (Illegal pawn movement)
- No token pick-up restriction control. (Illegal token pick-up)

Design of the code:

- The design of the code is very similar to the approach we had traditionally taken when we grouped similar or related code into packages.
- The use of ImageIcon was very neatly done and use of container component ratio was good. However at one glance the GUI looks very unorganized.
- The model on the other hand used principles that we traditionally had been using in our code for Stage 2.
- The board uses an Array based implementation which is what we decided to use for our code.
- What was interesting about this code was that the Pawn was visible as a dot on the screen.

Readability:

- What really set this code apart from others was it was very readable. The Board and Pawn classes were really easy to understand and the logic that was used was similar to ours.

Documentation:

- This code certainly lacks proper documentation. One reason why we didn’t choose this code was because we couldn’t figure out what each method did.

JUnit Tests:

- The tests written in the code covered every base. Testing each different component was done very efficiently.
- Testing the board , pawns and legal movements was all done in one single class. That made it a lot more easy to read.

[Stage2]
Design of Code:

- The code was designed were precisely. Each method was carefully written to perform specified

Readability:

- The code was very readable and the methods that were defined were very logical.

Documentation:

- This doc was very well written and documented.

JUnit Tests:

- What really set this code apart from every other code was the fact that it had somewhere close to 6 thousand tests.
- The tests very automatically generated and what was even more interesting was the fact the tests automatically ran the code and GUI.
- So with every test the code's behavior to legal and illegal moves can be tested and seen on screen.