COSC 3011
Introduction to Software Design
Spring 2019

Course Assessment

1 Introduction:

Course is has unlimited enrollment and is currently begin taught every Spring.

A 3 credit course, which meets 3 days a week. The basic course is lecture with two exams, a final exam, and programming project(s). The size of the enrollment means that group projects are reasonable if only from a grading point of view.

Texts:

- Dale Skrien, *Object-Oriented Design Using Java*. 1ed. 2009, McGraw-Hill, New York, NY.
- Object-Oriented Analysis, Design and Implementation, 2nd ed., Brahma Dathan and Sarnath Ramnath
- Java Design Patterns, Vaskaran Sarcar

References:

- Github git cheatsheet, training.github.com
- Git Pocket Guide, Richard E. Silverman (epub)
- Pro Git, Scott Chacon and Ben Straub (epub or mobi)
- Git Markdown cheatsheet, Anon

2 Instructors Evaluation of Course:

After teaching this for several years I an finally getting comfortable with the course.

Topics covered were: basic design principles, UML, CRC, Use Cases, and Design Patterns, refactoring. I used the same text as that used by my predecessors and have added two

more. All these are availble as PDF so it is no burden on the students. The Java Design Patterns adds enough on patterns to not need the Gang of Four text.

There were 68 students, and I divided them into groups of 4 or 5 for programming projects. This works well for the programming portion. Now if I can only get all the students to actually be group members instead of assuming someone else will cover for them it would be an awesome course. I have yet to have to step in an "fix" a group. Mostly I advise them to handle it and they step and shoulder the responsibility. For each of the six project milestone submissions, I had the students fill out an online *quiz* on group dynamics. The results were not especially interesting but did give me a feel for how they were getting along. One or two complaints about a specific individual, but never by all, or even most, group members.

Strangely, I did not get anyone asking to be allowed to opt out of group work for a solo operation. However, it appeared that for a few of the groups, there were one or two members who did all the programming, one who tried to keep things coordinated (meetings, etc.) and the others did bits and pieces, documentation like UML and basically tried to feel included. Other groups seemed to really click. Maybe because of one or two members, maybe just luck of the draw. Speaking of which, group membership was assigned by me and there was no changing allowed. I did insist that the groups completely isolate themselves from their fellows, but there did not seem to be much discussion about how they were doing or how something in particular was accomplished.

I started the course in the traditional lecture manner. I have limited the exams in course to just one. This is about the basics from the introductory classes and then we concentrate on management methodologies.

The exam results were good. I believe that I finally have figured out how to tell them what I want. And the exam is more about generalities.

From the first time I taught this course I kept the idea of assigning each group a topic for an in-class presentation (lecture). I allowed slides but severely limited the number they could use. I gave each group seven days to get it together, only giving them their topic just in time. All members of a group had to be present and give part of the presentation. Several groups did an outstanding job, most were barely adequate, but it did get them to actually investigate a specific area of design or OOP. I also had a brief (10 point) quiz, online, on each topic. This last was more on the order of homework (that was not very strictly graded). These quizzes were more for my edification that anything else. The students were also required to attend their compatriots lectures. I actually took attendance through the use of a slip of paper with four "rate the presentation, 1-5" questions, and a place for the rater's name. I compiled the results and let each group know what their fellows thought of their work.

The presentation was 55 points and its bibliography was 45. If each got up and talked they got full credit. They were also given 5 points per presentation attended. This served to act

as the second "exam." The students seemed to think this was not unreasonable. In fact a few said this (the presentations) was one of the better elements of the course.

The programming project was a semester-long affair. I gave them a really rough first pass at a format for a game GUI. Then through six iterations they had to add elements as I specified. This project had several goals:

- They needed to work in arbitrarily assigned teams to create a software project with specific deadlines.
- The groups were required to provide documentation of their design.
- The software, written in Java, had to work.
- The group design had to be changeable. For instance the game started with three required buttons, New/Reset/Quit. Part way though, New became File and they had to load and save files.
- The groups had to be able to learn, not just copy it from some online source. The source file format for the game was invented by me and was raw bytes, not textual. Most had never processed a file whose format was that specific and could not be examined with a regular text editor.

I liked the results in term of basic software. I made very few restrictions on what or how it would be written. I am still trying to make it perfect but it is working.

I need to be more specific about what design documents I want them to generate. The students seems to only have two modes, minimalism and "War and Peace." I need to get somewhere in the middle.

The last iteration of the game counted as their take-home final exam. And that seems to be working well for the course. There is a real problem with grading an exam for 68 students where the questions are open-ended synthesis and analysis type questions. I have to grade them all. I have to try to be objective. I have to be able to actually read them. If I could throw an ACT style exam with a bubble sheet at them and feel like I made a real evaluation, that would be different. But I do not fell that would be fair to either me or the students

3 Performance Indicators to Assess:

Perf. Ind. b.2: Analyze at least two or more proposed solutions to given problem and select the best solution for the given problem.

This was assessed through the program design project. The teams were required to plan, design, and program a game. This was in five steps over the majority of the semester with the sixth, wrap-up due as the final.

I have combined the results of the first five steps of the programming project as I did for the course grade.

68 students were assessed.

Excelled: 30 students
Mastered: 39 students
Partially Mastered: 2 students
Below Expectations: 7

Perf. Ind. c.3: Design the selected solution for a given problem.

This was assessed with the same set of projects as (b.2).

4 Attachments:

- Program 01 assignment
- Program 02 assignment
- Program 03 assignment
- Program 04 assignment
- Program 05 assignment

The following pages contain the assignments and quizzes used for the assessments. The Programs were used for assessing **b.2**, **c.3**, and .

For Performance Indicators b.2, c.3 Back

COSC 3011
Software Design
Program 01

5 Introduction

This assignment is the beginning of a programming project that will consist of several stages. We will start by getting the teams together, getting the software products accessed, doing a little programming, and, hopefully, a lot of planning.

The project is a game. Better than some lame web interface for something we did not want to do anyway. The game will be simplistic take on a non-electronic game, IZZI, original marketed by a company named Binary Arts (now Thinkfun). And no, we are not going to violate any copyrights. IZZI, consists of a set of 64 square tiles that are divided into eight sections. Each section connects to an edge, dividing each edge into two pieces. Each section is either black or white. The goal is to place every piece into a larger 8x8 square. The only rule is that white must touch white and black must touch black.

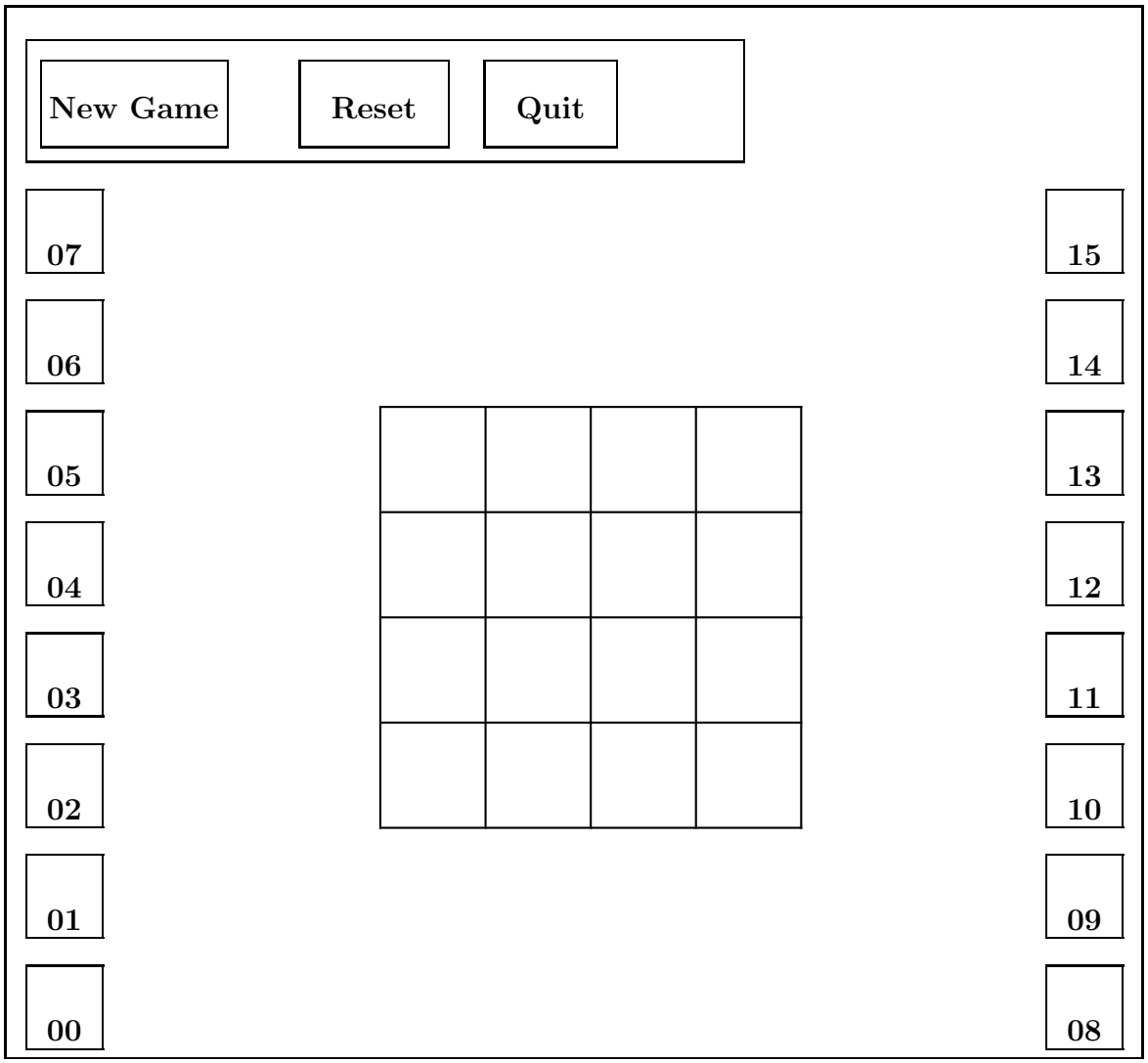
We are going to make a 4x4 square. Instead of colored sections, we are going to start with a maze. The maze will just be a set of lines. When placed, the lines in each tile must line up with those in the touching tiles.

Initially, there will be an empty game area and the tiles will be arranged along the left and right sides of the window (one column on each side). The player will use the mouse to drag-and-drop a tile somewhere in the game area. If the player decides to place that tile some where else, she can drag-and-drop again. If the player wants to use a different tile instead, he can drag a tile off the game area to an empty spot on the edge and choose another tile. Eventually, your program will allow the player to rotate a tile in the game area. And, to keep it interesting, the final version will have the tiles placed randomly in the holding areas and each will be randomly rotated, and (hopefully), the user can supply her own mazes.

On the next page is an example of what the initial game might look like. In fact, it would do for a first pass at the game. The numbers are just placeholders I put in for the example. These sites would be the original positions of the game tiles. Of course, the tiles would look like sections of a maze.

The option buttons are pretty self-explanatory. The *New Game* resets all the

tiles, and when the random factor described above is in place, there will be a "new" set of tiles. The *Reset* button, takes the current set of tiles and replaces them in their starting positions. *Quit* exits the game. Initially, only the *Quit* button works.



6 What to do

You need to get your teams together, plan, and do a little programming. I want you to develop an initial planning document. It needs to outline the project, as you see it. As part of this, create a basic UML diagram that will give you a start on the program. All this should describe the program as well as specify who will do what.

You should think about how an effective team works together. Set some rules

that you can all agree on. For this to be meaningful, you have to meet more than once for fifteen minutes. You might want to plan a meeting schedule, place, that sort of thing. Do not forget, participation in the project will have a significant impact on your project/program grade. I will post a "quiz" for each milestone. Each of you will complete that quiz, grading

the other members of your team. If you cannot put in the effort to complete the quiz, you must not be involved and your participation grade will be 0.

I will give you a starting point for this project. Basically a couple of files that you can modify and make additions to. Add files, directories, classes, interfaces, whatever you think you need to solve the problem. A hint, do not think that your first pass at this will have any resemblance to the final product. Do not get so attached to a design decision that, unless there is no time left, you are not willing scrap it and start again.

7 What to turn in

Create a zip (or tar/gzip) file of the directory structure that contains of all the ".java" files. DO NOT include ANY ".class" files. DO NOT create a jar file. By the way, I will be looking at your source code. FIX THE IDE!!!! Lines 120 characters long, not wrapping text reasonably, indenting with tabs instead of spaces, ALL BAD.

Make sure that your planning document(s) are included in this. Please put them someplace reasonable like in a "docs" directory. You will submit this on WyoCourses on the Program01 assignment.

For Performance Indicators b.2, c.3 Back

COSC 3011
Software Design
Program 02

8 Introduction

I really do not want you 'working ahead." That is not because I think you need to follow instructions, although that is large part of it, but more because you need to concentrate on making the alterations to your program as "generic" as possible so that when I tell you what follows this, you will not have to tear about a large amount of work and maybe have to rewrite all you have done.

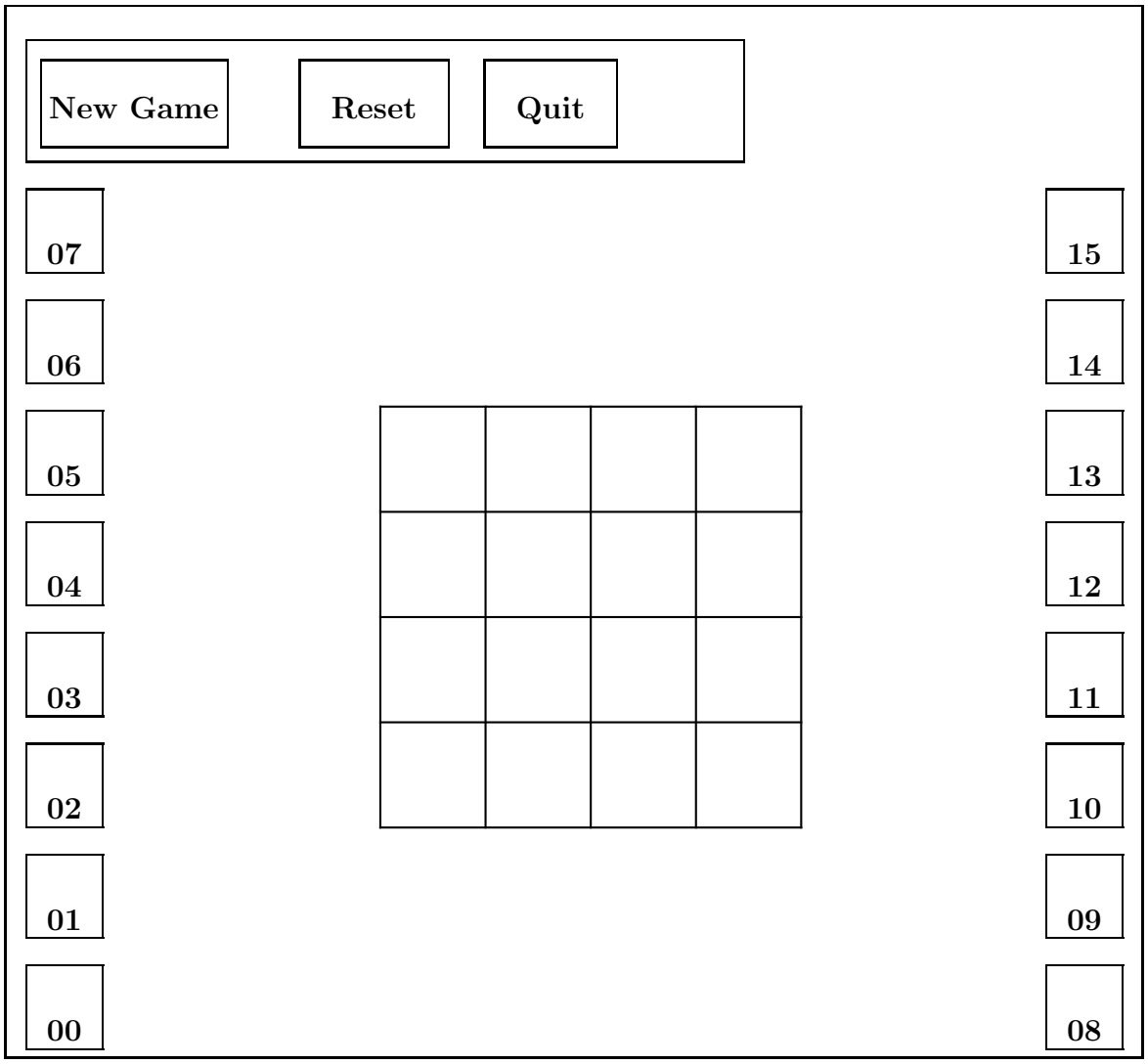
9 New Requirements

That said, now we are going to add some more functionality. First, I am assuming that did I did not ask you to make any major changes to your previous version. Make sure you review the comments on the previous program. That being said, if Quit button works, forget about the other buttons for this iteration of the project.

1. The possibility exists that you made the program window by simply drawing lines on a background. I hope you went farther than that but if you did not, then now, you need to add movable "tiles" in addition to those plain lines.
2. Make your tiles movable. Of course, for those over-achievers, you may already have that done.
    - I have no preference on which of the following two methods you choose. I would prefer that you do NOT do both, just too confusing.
    - You can drag/drop them from the holding areas onto the square.
    - You can make some movement based on clicks. Maybe double-click to select and single-click to place, or right then left, or center-paste, or...

- **Regardless which method you use, they must "snap" either to a position in the square or a position in the holding area. They cannot just placed anywhere on the board. That really means if they are "close" to either a position on the side or in the square they go to the "close" position. If they are too far away, pick some reasonable distance, they go back where they started.**

3. **Give your tiles some identity. Do not get carried away with this, but I want the tiles to be differentiated from the background of the rest of the playing surface, by each having a solid colored background. That does NOT mean, a different color for each, but you could do that if you want (again, over-achieving). I also want each visibly numbered. Like put a number in the center, bottom left corner, something consistent of each tile.**



10 What to do

I do not really want anything extraordinary this time, and maybe you think this is baby steps, but that is the way of it. You will again need to get your teams together, plan, and do a little programming. I want you to update your planning document(s) to reflect any design changes and/or additions to your project plan. Make sure that you also update the UML diagram to reflect the changes. The more work you do this time, to a degree, the less I will have to complain about. Of course, there is such a thing as overkill.

Do not forget, participation in the project will have a significant impact on your
project/program grade. I will post a "quiz" for each milestone. Each of you will complete that quiz, grading the other members of your team. If you cannot put in the effort to complete the quiz, you must not be involved and your participation grade will be 0.

You already have your first attempt at the program. Add on to that as needed. And again, do not get so attached to a design decision that, unless there is no time left, you are not willing scrap it and start again.

11 What to turn in

Create a zip (or tar/gzip) file of the directory structure that contains of all the ".java" files. DO NOT include ANY ".class" files. DO NOT create a jar file.

Make sure that your planning document(s) are included in this. Please put them someplace reasonable like in a "docs" directory. You will submit this on WyoCourses on the Program02 assignment.

For Performance Indicators b.2, c.3 Back

COSC 3011
Software Design
Program 03

12 Introduction

Now, you should be able to "place" tiles. So lets put something on them. You will read the information from a file and create tiles that can generate a maze.

13 New Requirements

I need to discuss the format of the file with the maze information. The format is probably not going to be something that you are used to. That just means I expect you figure out how to handle a "standard" format that everyone's versions of this game will use.

1. The file extension will ".mze".
2. The file will NOT be strings of text, it will be raw data. The values in the file will all be either *integers* or *floats* (not doubles).
3. When I say raw, I mean that an integer is stored as four bytes, the same with a float.
4. You will have to figure out how to convert values from their native format to byte arrays, and back.
5. File layout.
   - An integer number of tiles, N.
   - For each tile
     (a) An integer that is the tile "number," a value between 0 and N-1.
     (b) An integer that is the number of lines on the tile, M.
     (c) For each line, two pairs (four values) of floats that correspond to the $\{x_0, y_0, x_1, y_1\}$ endpoints of the the line.

6. Strangely, the filename of the default file (the one everyone starts with) will be default.mze. You can hard-code that for now.

And, the Reset button now has a job.

14 What to do

When your program starts, it will attempt to open and read all the data from the file default.mze. If it cannot open or read the file, then the program should exit.

Once the file data has been read, you will draw lines on the tiles that correspond to data that is in the file. No, you cannot change the data. If absolutely necessary, you can scale the data. You really should not have to do that.

It does not matter what order you draw the tiles in, just do not modify the layout. I should be able to compare the tiles from two different programs and they will be the same.

Now, the tiles that you have waiting on the sides of the window should each look like a maze tile piece. I should be able to place them on the game board and move them around as needed.

If I press the Reset button, any moved tiles should go back to their original position *before* any moves were made.

15 What to turn in

Create a zip (or tar/gzip) file of the directory structure that contains of all the ".java" files. DO NOT include ANY ".class" files. DO NOT create a jar file. If you have created a "package" make sure that include that directory in the zip file.

Make sure that your planning document(s) are included in this. Please put them someplace reasonable like in a "docs" directory. You will submit this on WyoCourses on the Program03 assignment.

There will be another quiz like that for Program 01.

For Performance Indicators b.2, c.3 Back

COSC 3011
Software Design
Program 04

16 Introduction

Now that the maze has been created and we can actually start "playing" the game, you need to add some *features*.

17 New Requirements

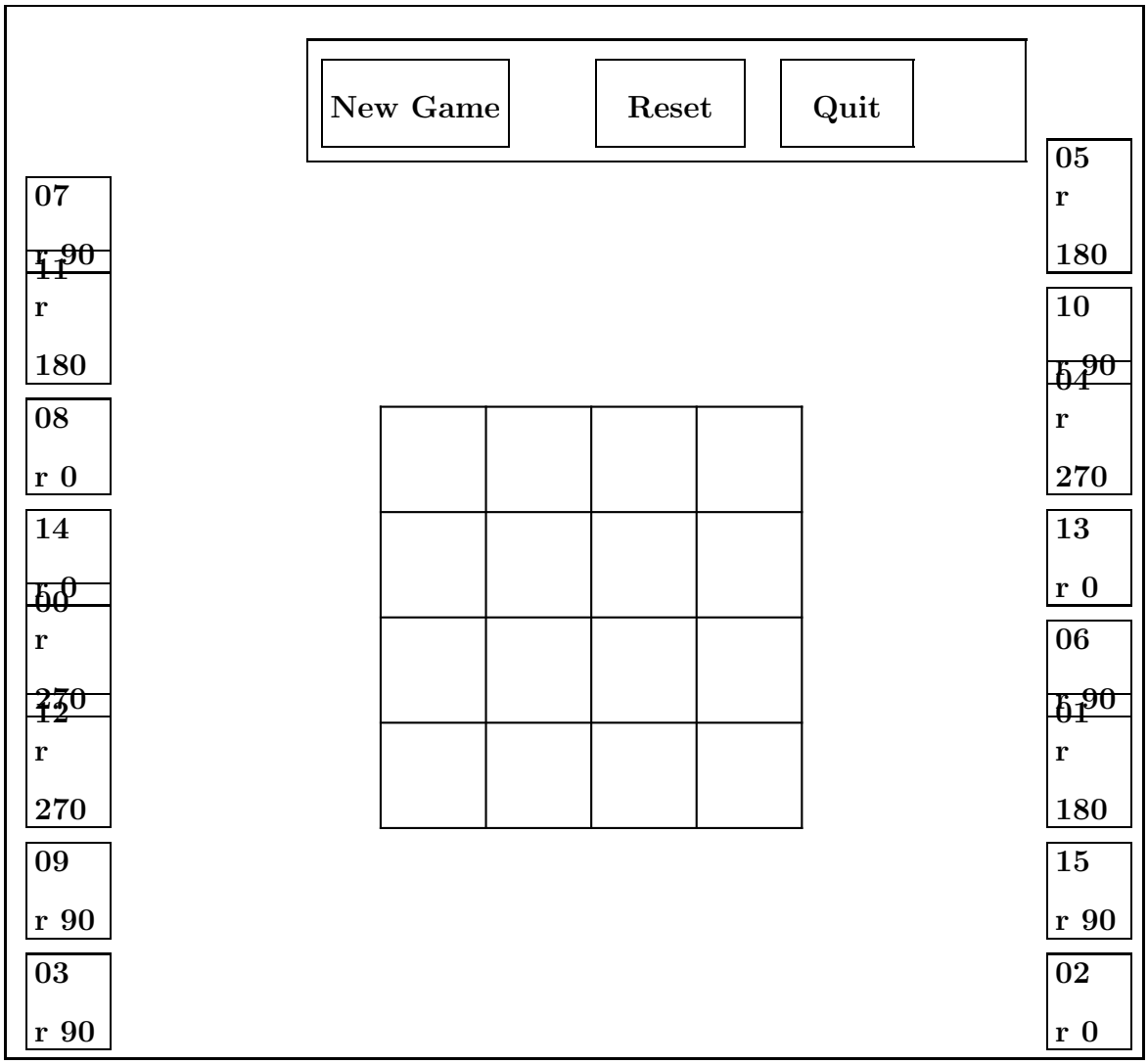
- Instead of just creating the tiles one after the other from the file data, you need to randomize their placement. That is, the player can no longer simply place the first tile in the upper left corner, the second tile to the right of that and so on until the maze is completed.

  1. If we number the tiles as specified in the data file, then the "first" tile displayed may or may not be 0, the second should not be 1, and so on.
  2. If we assume that each tile, as the data is given, has a rotation of $0°$ then no more than four of the tiles will initially be displayed with that rotation. The rest will be rotated one of $90°$, $180°$, or $270°$. Make sure that at least one of the remaining tiles is rotated each one of these.

- Then if you are randomly generating them and randomly rotating them, you have to give the player the ability to rotate tiles as well as place them. The rotation could be initiated by something as simple as a right-click on the tile causing it to rotate clockwise $90°$ every click. Whatever you choose it should be simple (and documented) and not require keyboard events. Also, this must work wherever the tile is currently placed and be "sticky," once a tile is rotated, it keeps that rotation until it is rotated again, regardless of where it is moved.

- The initial position and rotation of the tiles must be saved, because that Reset has to still work and put all the tiles back to their original position and starting rotation.

- **The picture on the next page gives you an idea of the "random" placement and rotation (r 90 is rotate $90°$).**



18 What to do

- **When your program starts, it will attempt to open and read all the data from the file default.mze. If it cannot open or read the file, then the program should exit.**
- **Once the file data has been read, you will draw lines on the tiles that correspond to data that is in the file, placing them randomly in the holding areas and rotating at least 75% of tiles as specified above. The same requirements exist on the modification of the file data (do not change it).**

- **Every time the user starts a new game, she should expect to get a new initial layout. That means you need to dynamically determine each tiles position and rotation in the starting spaces.**
- **In addition to the ability to place the tiles on the gameboard the following must be true.**
  1. **No two tiles can occupy the same place. An attempt to place one tile into a position that is occupied will cause the "incoming" tile to be returned to its last position. There needs to be some indication (preferably not a beep) that this was an invalid move. The indication cannot be a pop-up window nor can it be output to the terminal. Something simple like a "flash" (tile background temporarily changing) or whatever you can come up with. You have played this type of game, if you cannot be creative, be a copy cat.**
  2. **Once rotated, the tiles will retain their current rotation until rotated again. There is no limit on the number of times a tile can be rotated, moved, placed, re-moved, re-rotated, re-placed, etc.**
  3. **The player must be able to take tiles off the game board as well as put them on the game board. In fact, any tile, can be placed in any open position, at any time.**
- **If the Reset button is pressed, any moved tiles should go back to their original position and original rotation *before* any moves were made.**
- **The buttons will be named exactly New Game, Reset, and Quit. And there will only be those three buttons. They will be placed near the top of the window approximately where they are shown shown in the picture on the previous page.**
- **The window for the game will be layed out as was specified in that original document. If you cannot display the window as originally given because you have absolutely no access to any computer anywhere that has a display resolution large enough for a 900x1000 window, then scale it all. That means it will still look like the originally given example, just smaller. AND DOCUMENT IT.**

  You do not have to figure out how to dynamically resize the window. But I warn you, if it is less than 600 high, I will assume that you are jerking my chain and grade accordingly. Tiny is not good.
- ***All submissions* will have a Main class, inside a Main.java file and that class will contain the *main*() method. Changing the name of that class is not an option. Everything else, not really an issue.**

SEE NEXT PAGE!!!!

19 What to turn in

Create a zip (or tar/gzip) file of the directory structure that contains of all the ".java" files. DO NOT include ANY ".class" files. DO NOT include any of the Eclipse files. DO NOT create a jar file. If you have created a "package" make sure that include that directory in the zip file AND document it.

Make sure that your planning document(s) are included in this. Please put them someplace reasonable like in a "docs" directory. You will submit this on WyoCourses on the Program04 assignment.

There will be another quiz like that for Programs 01 and 03. I will even put the link in the right place in the module.

For Performance Indicators b.2, c.3 Back

COSC 3011
Software Design
Program 05

20 Introduction

This time we add something that should not be too strenuous. We have a game that should be playable, and hopefully interesting. But it may be difficult to solve so the user (because the boss is coming) quickly saves the current game state so that it can be reloaded "at lunch" (whenever the boss is not looking).

21 New Requirements

- The program will still start by loading the default input file, the same as the last version of the program. By the way, test this from the command line and make sure that you provide documentation that indicates which directory the default file must be found in.
- That "New Game" button is now relabeled File.

  1. The File button will give a menu that ONLY has Load and Save.
  2. At a minimum, Load and Save will provide a text entry area for a file name to load or save. If the file is not found (Load) you will give some nice error pop-up. If the file name already exists for Save, you will provide a pop-up that allows the user to continue (which completely overwrites the existing file) or goes back to the text entry field.

     If you want these options to pop-up file selector windows that is fine AS LONG AS they include the option of typing in a filename. Filenames MUST allow for either full paths or relative paths as well as the current directory. File selectors MUST include a way to navigate up/down the directory structure.

     Remember that this may not run on Windows. On other operating systems path element separators are the (forward) slash, /, not the back slash, \. That means do not decide what those characters, let the user (and Java) do the heavy lifting.

- **Assuming that you choose to load a file, the program will replace the currently loaded maze with the chosen new maze. If the current maze has been modified, you must give the user the opportunity to save it. Make sure that you test this! If the current maze has not been changed since loading, then simply load the new maze from the file.**
- **There will have to be a new file format to make all this work. It will still be raw bytes but it needs some changes.**
  - **The file extension will default to ".mze" but the user can use any extension. That is not a real issue. We will DECIDE NOTHING based on file extension.**
  - **When the first four bytes of maze files contain the hexadecimal values `0xca, 0xfe, 0xbe, 0xef`, the maze is an *original*. By *original* I mean that, like the default maze, it is designed to be loaded as a never-before-played game. That means when it is loaded, it is randomized as for Program 4.**
  - **When the first four bytes of maze files contain the hexadecimal values `0xca, 0xfe, 0xde, 0xed`, the maze has been *played* and all the positions/rotations are expected to be retained as-is.**
  - **Previously the tile numbers were in the 0-15 (0 to N-1) range. We have to expand that to 0-31. The values 0-15 will indicate that the tiles are in the holding areas, 0 is the upper-left space, 1 is next down on the left, 8 is the upper-right, and 15 the lower-right. The values 16-31 will indicate the tile is in the game area. The upper-left corner is 16, the next to the right is 17, and so on until the bottom-right corner is 31.**
  - **There will have to be another value stored with each tile, the tile's *rotation*. This will be an integer from 0-3, with 0 being no rotation, 1 is $90°$ clockwise, 2 is $180°$, and 3 is $270°$ clockwise.**
  - **To summarize**
    1. **Bytes 0-3: `0xca, 0xfe, 0xbe, 0xef` if unplayed for randomized initial layout (in starting areas) or `0xca, 0xfe, 0xde, 0xed` if *played* and tile layout is fixed.**
    2. **Bytes 4-7: an integer number of tiles, N.**
    3. **Next four bytes: an integer tile number, range 0-31, defines exact placement if this is a *played* maze. Ignored otherwise.**
    4. **Next four bytes: an integer tile rotation, range 0-3, defines exact rotation if this is a *played* maze. Ignored otherwise.**
    5. **Next four bytes: an integer number of lines, M, on the tile.**
    6. **Next sixteen bytes: four floats that are the $(x_0, y_0), (x_1, y_1)$ endpoints of each line.**
    7. **Repeat item 6. M times.**
    8. **Repeat from 3. for each N-1 remaining tiles.**

22 What to do

- When your program starts, it will attempt to open and read all the data from the file default.mze. If it cannot open or read the file, then it should prompt the user for a filename. Easiest here to leverage the Load menu option.
- If any file that is being loaded does not have the correct first four bytes, give a simple (pop-up) error about invalid file format. When the pop-up is acknowledged, close it and present the game window with no maze loaded. This gives the user the choice to load a different file or quit.
- Once the file data has been read, you will draw lines on the tiles that correspond to data that is in the file.
- If the first four bytes are `0xca 0xfe 0xbe 0xef`, place the tiles randomly in the holding areas and rotate at least 75% of tiles as specified before. The same requirements exist on the modification of the file data (do not change it).
- If the first four bytes are `0xca 0xfe 0xde 0xed`, place the tiles exactly where specified by the file data with the specified rotation.
- Every time the user starts a new game, she should expect to get a new initial layout (from the default or *original* maze files). That means you need to dynamically determine each tile's position and rotation in the starting spaces.
- In addition to the ability to place the tiles on the gameboard the following must be true.

    1. No two tiles can occupy the same place. An attempt to place one tile into a position that is occupied will cause the "incoming" tile to be returned to its last position. There needs to be some indication (preferably not a beep) that this was an invalid move. The indication cannot be a pop-up window nor can it be output to the terminal. Something simple like a "flash" (tile background temporarily changing) or whatever you can come up with. You have played this type of game, if you cannot be creative, be a copy cat.
    2. Once rotated, the tiles will retain their current rotation until rotated again. There is no limit on the number of times a tile can be rotated, moved, placed, re-moved, re-rotated, re-placed, etc.
    3. The player must be able to take tiles off the game board as well as put them on the game board. In fact, any tile, can be placed in any open position, at any time.

- **If the Reset button is pressed, any moved tiles should go back to their original position and original rotation *before* any moves were made. If the Reset button is pressed, assuming that there had previously been any move in the game, and the next thing the user does is try to Load a new file (or Quit), the user will NOT get a warning about saving the file. That means that Reset, resets the *modified* attribute.**
- **The File button will bring up a menu with exactly two selections, Load and Save. See the preceding text about what these two options should do. If you have any questions about this please ask.**
- **The buttons will NOW be named exactly File, Reset, and Quit. And there will only be those three buttons. They will be placed near the top of the window approximately where they are shown shown in the picture from Program 4.**
- **If the user selects Quit and there has been any move in the game, the user will be asked (again, use a pop-up) if they want to save their changes. If Yes, then give your "standard" Save option, otherwise exit.**
- **Window layout should not change from the previous assignment.**
- ***All submissions* will have a Main class, inside a Main.java file and that class will contain the *main*() method. Changing the name of that class is not an option. Everything else, not really an issue.**

SEE NEXT PAGE!!!!

23 What to turn in

Create a zip (or tar/gzip) file of the directory structure that contains of all the ".java" files. DO NOT include ANY ".class" files. DO NOT include any of the Eclipse files. DO NOT create a jar file. If you have created a "package" make sure that include that directory in the zip file AND document it.

Make sure that your planning document(s) are included in this. Please put them someplace reasonable like in a "docs" directory. You will submit this on WyoCourses on the Program04 assignment.

There will be another quiz like that for Program 4.