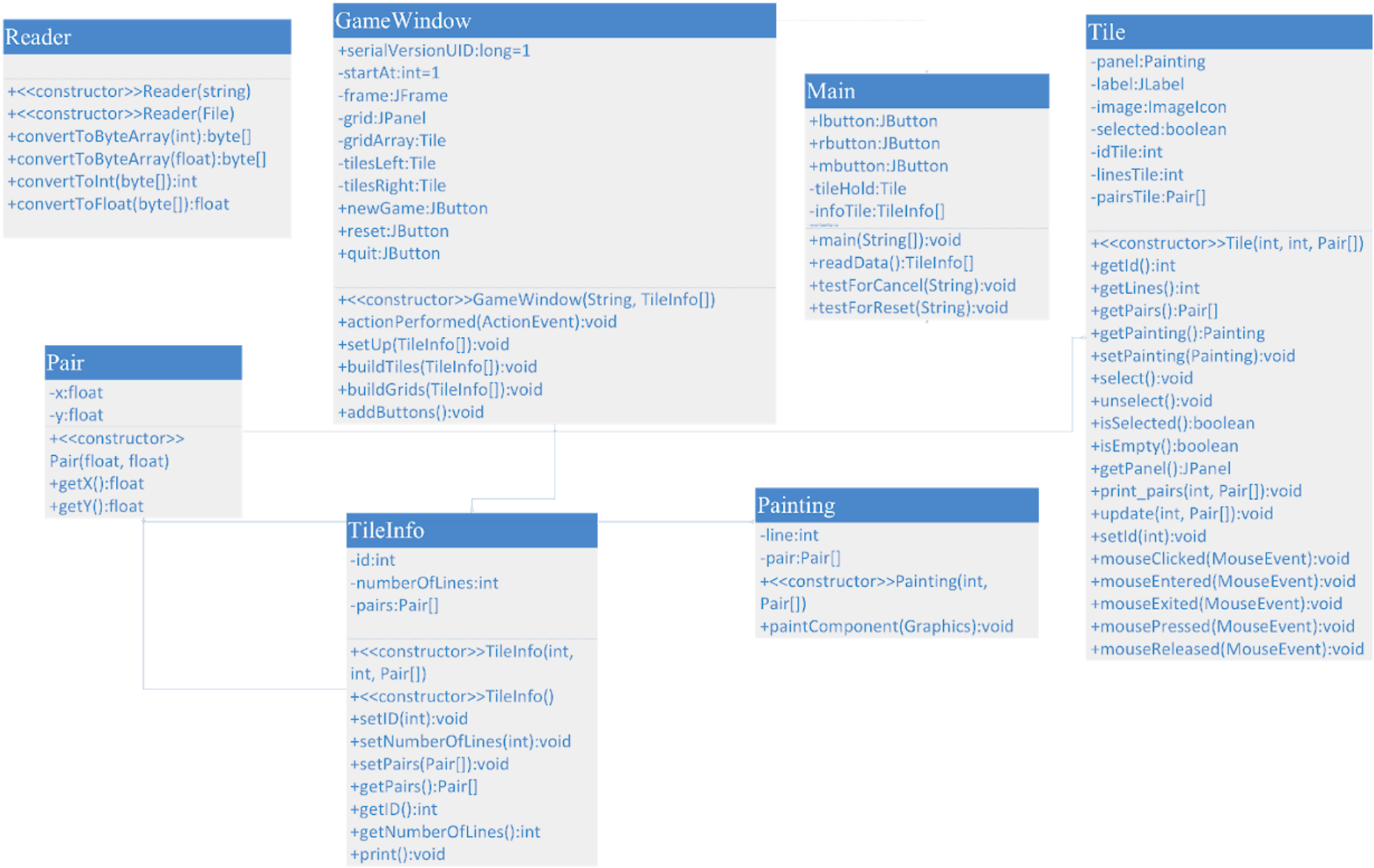
Group J
Damir , Oyedola, Omar, Max, Ana
May 11, 2018
Software Design

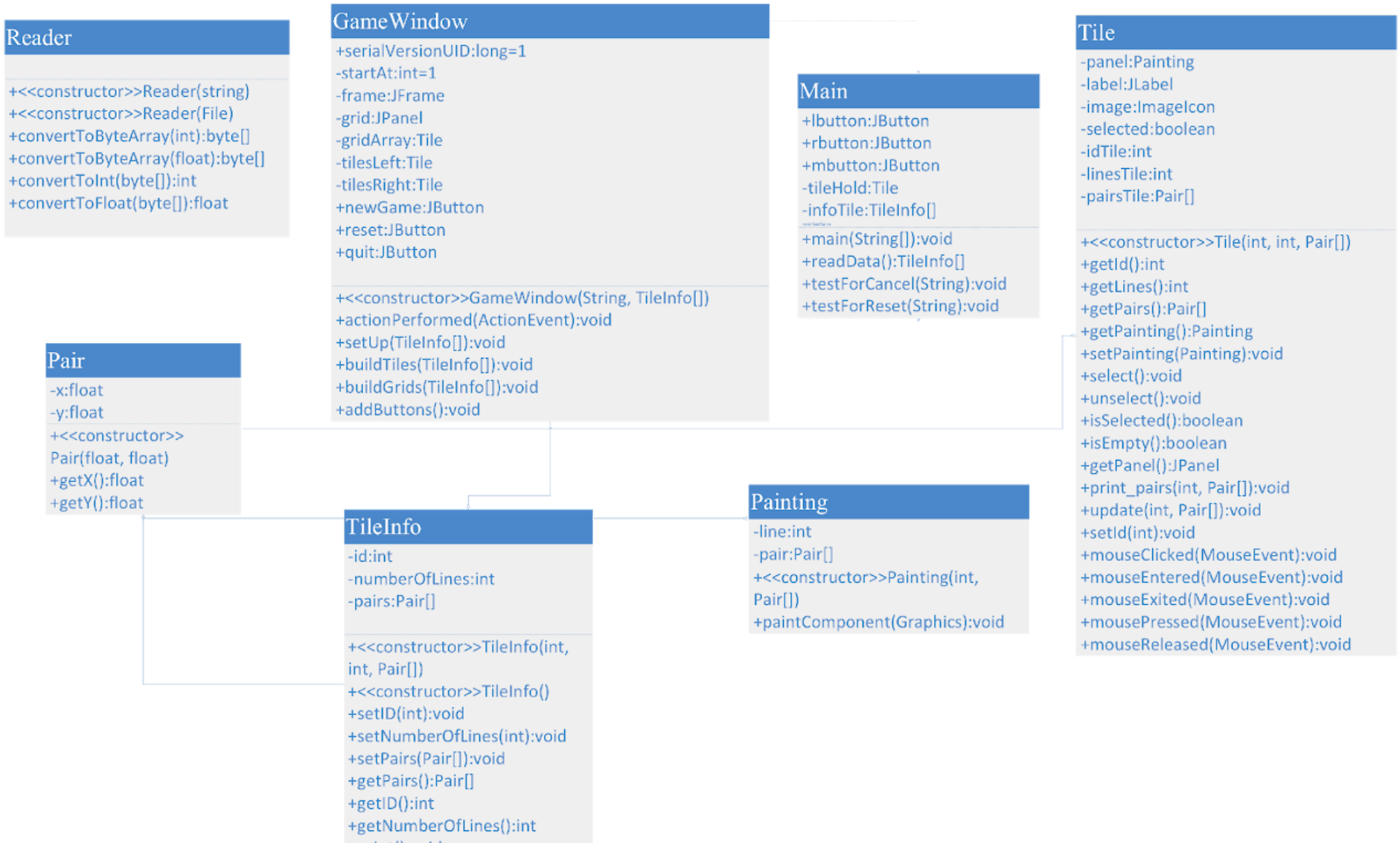
The first step of this project was planning. In order to create exactly what we wanted and needed, we had to figure out as a group how the game was played, and what the end-goal was. As a result, the first program consisted of plenty of planning, and setting up the very basics of the game board, including some planning documents as to how we would go about this as well as a simple UML. We created a layout consisting of three buttons, “New Game,” “Reset,” and “Quit,” a 4x4 grid in the center of the screen, and 16 place-holding squares on either side of the grid. It was purely aesthetic, but made clear what was coming in future programming assignments. For ours, the side squares turned out more rectangular (issues with sizing), but the idea was there, and we failed to add the lines in the middle grid to show the user where the tiles are supposed to go.

Program 2 added some functionality to the game board. First, we fixed the rectangular tile spaces and made them squares. We also planned how to create a Tile class, and implemented the actual tiles to take the place of the empty spaces. We also made the “Quit” button exit the game window. However, with this iteration, we had some trouble making the tiles moveable. We had decided beforehand to move based on clicks as opposed to drag-and-drop, but we had difficulty actually bringing it to fruition. Instead, the tiles ended up disappearing when selected. This was fixed in the next iteration, with the aid and guidance of Dr. Buckner.

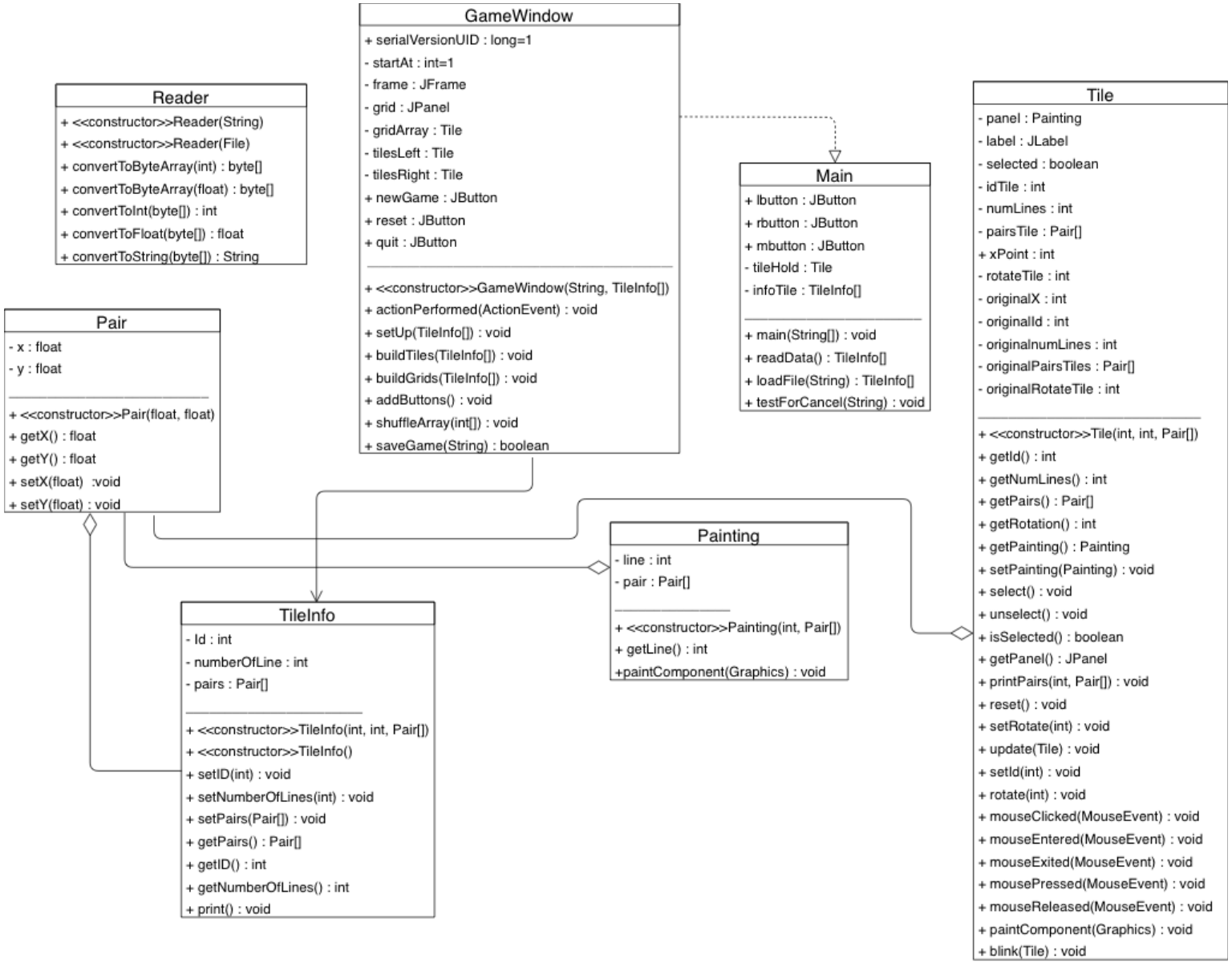
The third program involved reading in raw data files and converting them to a format that can read to be printed on the tiles (to create the maze image). As with the other iterations, our first meeting involved planning what needed to be finished from the previous assignments, and who would take which tasks, ranking them in order of importance. First, we created a reader class that would read in the data and convert it to the correct format. Then, the information was actually be drawn on each of the tiles, with the use of the Painting class. Each of the tiles had a corresponding TileInfo aspect that contained an ID, number of lines, and coordinates for the lines to make it easier to draw. Also, the "Reset" button needed to be set up so that the side tiles that were moved to the center grid would return to their original positions on the sides, and double-clicking the tiles would also send them back. Unfortunately, we were unable to get this functionality working in this iteration of the project. Also, at this point in the project, we decided to abandon the GridBagLayout that we had attempted to set up, and instead used setBounds to position everything on the board. The UML for program 3 (the first one that was saved individually, unfortunately) is on the following page.


Reader
+<<constructor>>Reader(string)
+<<constructor>>Reader(File)
+convertToByteArray(int):byte[]
+convertToByteArray(float):byte[]
+convertToInt(byte[]):int
+convertToFloat(byte[]):float
GameWindow
+serialVersionUID:long=1
-startAt:int=1
-frame:JFrame
-grid:JPanel
-gridArray:Tile
-tilesLeft:Tile
-tilesRight:Tile
+newGame:JButton
+reset:JButton
+quit:JButton
+<<constructor>>GameWindow(String, TileInfo[])
+actionPerformed(ActionEvent):void
+setUp(TileInfo[]):void
+buildTiles(TileInfo[]):void
+buildGrids(TileInfo[]):void
+addButtons():void
Main
+lbutton:JButton
+rbutton:JButton
+mbutton:JButton
-tileHold:Tile
-infoTile:TileInfo[]
+main(String[]):void
+readData():TileInfo[]
+testForCancel(String):void
+testForReset(String):void
Tile
-panel:Painting
-label:JLabel
-image:ImageIcon
-selected:boolean
-idTile:int
-linesTile:int
-pairsTile:Pair[]
+<<constructor>>Tile(int, int, Pair[])
+getId():int
+getLines():int
+getPairs():Pair[]
+getPainting():Painting
+setPainting(Painting):void
+select():void
+unselect():void
+isSelected():boolean
+isEmpty():boolean
+getPanel():JPanel
+print_pairs(int, Pair[]):void
+update(int, Pair[]):void
+setId(int):void
+mouseClicked(MouseEvent):void
+mouseEntered(MouseEvent):void
+mouseExited(MouseEvent):void
+mousePressed(MouseEvent):void
+mouseReleased(MouseEvent):void
Pair
-x:float
-y:float
+<<constructor>>
Pair(float, float)
+getX():float
+getY():float
TileInfo
-id:int
-numberOfLines:int
-pairs:Pair[]
+<<constructor>>TileInfo(int, int, Pair[])
+<<constructor>>TileInfo()
+setID(int):void
+setNumberOfLines(int):void
+setPairs(Pair[]):void
+getPairs():Pair[]
+getID():int
+getNumberOfLines():int
+print():void
Painting
-line:int
-pair:Pair[]
+<<constructor>>Painting(int, Pair[])
+paintComponent(Graphics):void

Program 4, like all the previous iterations, began with planning what needed to be done and divvying up the work. The assignment involved randomizing the placement of the tiles after the data has been read and printed onto them. To do this, we randomized the placement of the side tiles based on their ID. We also had to implement a rotate that would turn some of the tiles when they were randomly placed along the sides. "Reset" would now, ideally, also return the rotation to its original rotation. Our reset button was able to return the tiles, but not the rotation. Also, because we used the same seed in our random number generator for placement, the "random" placement would stay the same. We added an inability to swap tiles, so a maze tile could not be placed on top of another maze tile. The UML for Program 4 is on the following page.


Reader
+<<constructor>>Reader(string)
+<<constructor>>Reader(File)
+convertToByteArray(int):byte[]
+convertToByteArray(float):byte[]
+convertToInt(byte[]):int
+convertToFloat(byte[]):float
GameWindow
+serialVersionUID:long=1
-startAt:int=1
-frame:JFrame
-grid:JPanel
-gridArray:Tile
-tilesLeft:Tile
-tilesRight:Tile
+newGame:JButton
+reset:JButton
+quit:JButton
+<<constructor>>GameWindow(String, TileInfo[])
+actionPerformed(ActionEvent):void
+setUp(TileInfo[]):void
+buildTiles(TileInfo[]):void
+buildGrids(TileInfo[]):void
+addButtons():void
Main
+lbutton:JButton
+rbutton:JButton
+mbutton:JButton
-tileHold:Tile
-infoTile:TileInfo[]
+main(String[]):void
+readData():TileInfo[]
+testForCancel(String):void
+testForReset(String):void
Tile
-panel:Painting
-label:JLabel
-image:ImageIcon
-selected:boolean
-idTile:int
-linesTile:int
-pairsTile:Pair[]
+<<constructor>>Tile(int, int, Pair[])
+getId():int
+getLines():int
+getPairs():Pair[]
+getPainting():Painting
+setPainting(Painting):void
+select():void
+unselect():void
+isSelected():boolean
+isEmpty():boolean
+getPanel():JPanel
+print_pairs(int, Pair[]):void
+update(int, Pair[]):void
+setId(int):void
+mouseClicked(MouseEvent):void
+mouseEntered(MouseEvent):void
+mouseExited(MouseEvent):void
+mousePressed(MouseEvent):void
+mouseReleased(MouseEvent):void
Pair
-x:float
-y:float
+<<constructor>>
Pair(float, float)
+getX():float
+getY():float
TileInfo
-id:int
-numberOfLines:int
-pairs:Pair[]
+<<constructor>>TileInfo(int, int, Pair[])
+<<constructor>>TileInfo()
+setID(int):void
+setNumberOfLines(int):void
+setPairs(Pair[]):void
+getPairs():Pair[]
+getID():int
+getNumberOfLines():int
Painting
-line:int
-pair:Pair[]
+<<constructor>>Painting(int, Pair[])
+paintComponent(Graphics):void

The next step of the project was to add “Load” and “Save” button under “File” (what used to be “New Game”). If the user tried to load a file that did not exist, an error was returned, and if the user tried to save a game with the same name as another file, it prompts the user to either overwrite or change the name. Unfortunately, we did not add a pop-up window prompting the user to save the game before quitting, and had difficulty getting the game to correctly load saved games. Also, we were given new file formats to read in and print based on the first four byte values. If the first four bytes were 0xca 0xfe 0xbe 0xef then the tiles were to be randomly placed and rotated (as before). If they were 0xca 0xfe 0xde 0xed, the tiles were to be placed exactly as was specified in the file. At this point, we had the reset button working correctly, as well as the options for loading, saving, and quitting the game. The UML for Program 5 is on the following page.


GameWindow
+ serialVersionUID : long=1
- startAt : int=1
- frame : JFrame
- grid : JPanel
- gridArray : Tile
- tilesLeft : Tile
- tilesRight : Tile
+ newGame : JButton
+ reset : JButton
+ quit : JButton
+ <<constructor>>GameWindow(String, TileInfo[])
+ actionPerformed(ActionEvent) : void
+ setUp(TileInfo[]) : void
+ buildTiles(TileInfo[]) : void
+ buildGrids(TileInfo[]) : void
+ addButtons() : void
+ shuffleArray(int[]) : void
+ saveGame(String) : boolean
Reader
+ <<constructor>>Reader(String)
+ <<constructor>>Reader(File)
+ convertToByteArray(int) : byte[]
+ convertToByteArray(float) : byte[]
+ convertToInt(byte[]) : int
+ convertToFloat(byte[]) : float
+ convertToString(byte[]) : String
Main
+ lbutton : JButton
+ rbutton : JButton
+ mbutton : JButton
- tileHold : Tile
- infoTile : TileInfo[]
+ main(String[]) : void
+ readData() : TileInfo[]
+ loadFile(String) : TileInfo[]
+ testForCancel(String) : void
Tile
- panel : Painting
- label : JLabel
- selected : boolean
- idTile : int
- numLines : int
- pairsTile : Pair[]
+ xPoint : int
- rotateTile : int
- originalX : int
- originalId : int
- originalnumLines : int
- originalPairsTiles : Pair[]
- originalRotateTile : int
+ <<constructor>>Tile(int, int, Pair[])
+ getId() : int
+ getNumLines() : int
+ getPairs() : Pair[]
+ getRotation() : int
+ getPainting() : Painting
+ setPainting(Painting) : void
+ select() : void
+ unselect() : void
+ isSelected() : boolean
+ getPanel() : JPanel
+ printPairs(int, Pair[]) : void
+ reset() : void
+ setRotate(int) : void
+ update(Tile) : void
+ setId(int) : void
+ rotate(int) : void
+ mouseClicked(MouseEvent) : void
+ mouseEntered(MouseEvent) : void
+ mouseExited(MouseEvent) : void
+ mousePressed(MouseEvent) : void
+ mouseReleased(MouseEvent) : void
+ paintComponent(Graphics) : void
+ blink(Tile) : void
Pair
- x : float
- y : float
+ <<constructor>>Pair(float, float)
+ getX() : float
+ getY() : float
+ setX(float) :void
+ setY(float) : void
Painting
- line : int
- pair : Pair[]
+ <<constructor>>Painting(int, Pair[])
+ getLine() : int
+paintComponent(Graphics) : void
TileInfo
- Id : int
- numberOfLine : int
- pairs : Pair[]
+ <<constructor>>TileInfo(int, int, Pair[])
+ <<constructor>>TileInfo()
+ setID(int) : void
+ setNumberOfLines(int) : void
+ setPairs(Pair[]) : void
+ getPairs() : Pair[]
+ getID() : int
+ getNumberOfLines() : int
+ print() : void

Finally, for this last iteration of the project, we were tasked with programming the win condition, adding a timer to the game, and fixing all previous tasks that were either incomplete or not functioning properly. With regards to our group (J), we added proper functionality to “Load” and “Save” this time around. The issue was that we would occasionally load the testing files that were given to us. In turn, this also affected our save functionality. The main issue that we fixed was with regards to us allocating more size to our byte array than we needed. Furthermore, we were also reading and writing 32 tiles, but we were only required to do 16 tiles. Although, these were some of many issues, they were the biggest contributors to getting our “Load” and “Save” to work. Moreover, we also managed to integrate the timer functionality without much trouble. Additionally, we also managed to integrate our win condition with little to no trouble. The UML for program 6 is on the following page.