COSC 3011

Planning Document for Group G

Program 01

Group G is composed of Jay Bishop, Kyle Bobak, Anna Carrigan, Evan Turner, and Debbie Kretzschmar.

The team has regularly scheduled meetings every Tuesday and Thursday with the option to have additional meetings as required. In-between times, we use email for communication and to share documents.

Our responsibilities for Project 01 have been distributed through-out the group. Anna took the lead on initially getting the team together for meetings. Jay wrote the code for the Play Area buttons, Kyle, the code for the Side Area buttons, and Evan, the “New, Reset, and Quit” buttons. Anna wrote the code for the Tile Class and reworked code in several other areas. Debbie wrote the planning document including the UML and TileArea abstract class. All members of the group collaborated as a team to come up with a concept of how we wanted to address the challenge of developing the game. A whiteboard in our meeting room was used to sketch different possible approaches. The team is working cohesively and each member will assume duties as required to accomplish future assignments.

We first analyzed the specifications. We were tasked to design a game board that contains a four by four playing area with sixteen slots that are initially empty. Along each side of the board, in a single column, there will eight game pieces marked with a line, which we will call drawn tiles, for a total of sixteen. A panel along the top of the game board needs to contain three buttons. The buttons will be “New”, “Reset”, and “Quit”. The New Button resets all the tiles back to the side areas of the board. The Reset Button takes a current set of tiles and puts them back in their original slots along the side of the game board. The Quit Button will exit the game.

The rules for the game are that the player may click on a drawn tile they wish to move from the side area, and then click on an empty location in the playing area where they wish to place the drawn tile. A player can change their mind about where they would like to place a drawn tile by clicking on that tile in the playing area, and then clicking on an empty spot in the side areas. The player can then move a different drawn tile from the side area to the location where the previous tile was located. The end goal of the game is to arrange all sixteen drawn tiles in the playing area such that each tile’s line exactly aligns with the adjacent tile’s line.

As the group analyzed the specifications, we also considered expansion or future requirements. The tiles must be able to rotate in the playing area, clicking on the “New” Button will cause the tiles to shuffle their locations and also rotate, and a player can also supply their own maze. We also considered that a future requirement may be to check the player’s solution i.e. do all the lines match up.

After considering the above requirements, and future expansion options, we developed the following initial design. (A UML diagram is at the end of this description) After game set up, a key step would be to determine how to move the tiles from the “side area” to the “playing area”. The mechanism we devised was to click on the tile we wished to move, “copy” its properties i.e. design, and then paste that design/image onto the second tile that was clicked. The second tile’s properties would be copied into the first tile’s properties. This would create the effect of moving the tile by switching the properties from one tile to another.

The classes we chose to implement this concept include Main, GameWindow, Tile, TileArea, GridButtons, and SideButtons. In addition, several Java library classes or interfaces such as JFrame, JToolBar, JButton, Image, and ActionListener were utilized. We will next describe what each of our classes represents, followed by its responsibilities and how it collaborates with the other classes.

Class GameWindow represents the entire game board. Class GridButtons represents the central four by four playing area that holds sixteen game pieces or tiles, and Class SideButtons are the two areas along the left and right side of the game board. Each one of these two areas holds eight tiles. Tile Class represents the game pieces or tiles. TileArea is an abstract class that contains common methods used by both SideButtons and GridButtons.

Class GameWindow is essentially the manager of the overall game. It has knowledge of the entire setup for the game board, including the game board’s initialization, tile moves, resetting a game, setting up a new game, or exiting. Because moving a tile involves “big picture” knowledge of where the first click occurs as well as the second, this method was placed in the GameWindow. However, GameWindow collaborates with Tile to provide information on whether or not the Tile “is drawn” and SideButtons/GridButtons can provide information on whether or not their positions are blank or filled with a drawn tile. This collaboration will be used to validate a move. Thus, if a player tries to click on a drawn tile in the SideButtons area and then clicks on a drawn tile in the GridButtons area, the move would be declared invalid due to the collaboration via dependency or association GameWindow has with Tile, GridButtons, and SideButtons.

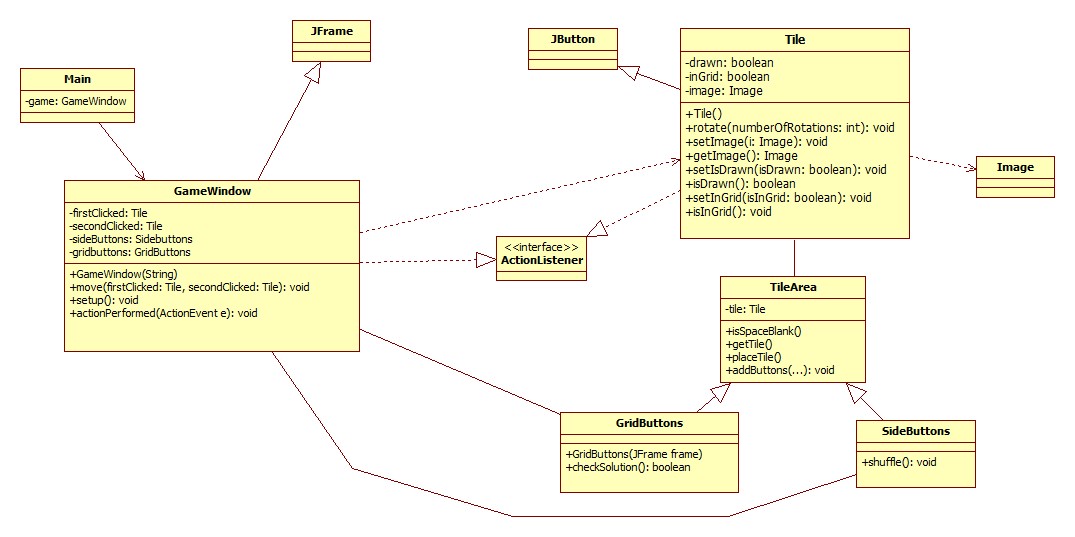
TileArea is an abstract class that contains common methods to both GridButtons and SideButtons. Its responsibility is to know what type of tiles are in its slots i.e. are they drawn or not drawn? It will also be involved in getting and placing tiles.

Both GridButtons and SideButtons are subclasses of TileArea. However, they have specialized behavior in that GridButtons class may check in the future on whether or not a solution is correct, and SideButtons will shuffle tile locations in order to have a new game have randomly placed tiles. SideButtons also needs to know what tile design was located in each of its slot’s at game start. This will allow the player to “reset” the tiles back to their original position or “shuffle” these locations in the future.

A Tile knows if it has an image drawn on it or it is blank. It also knows if it is in the GridButtons (playing area) or not in the GridButtons. It can also rotate itself. It has an association with the TileArea and as a result both the GridButtons and SideButtons.

In summary, in order to allow a user to play the above described game, our group will be using a “click” mechanism instead of “drag and drop”. We will have GameWindow oversee the game processes from setup to exiting. Players will move tiles by clicking on a tile and then clicking on a second tile at the location they wish to move to. The tiles will be moved by switching the property information between the two tiles. GameWindow with knowledge or collaboration from the helper classes of SideButtons, GridButtons, and Tile will determine if moves are valid and accomplish setting up new games, resetting them, or exiting. (Please see the UML diagram on the next page)

**UML FOR PROPOSED DESIGN**

****