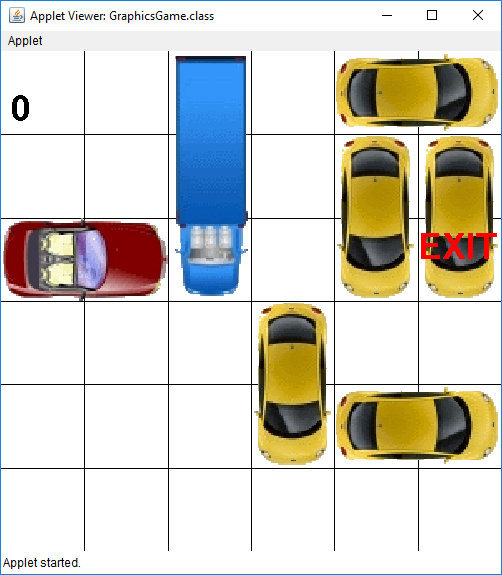
# COMP 55 – Traffic Jam Graphics

Now that you have the awesome Traffic Jam game, your job is just to use what you’ve learned this week to create the Graphics version of this awesome game.



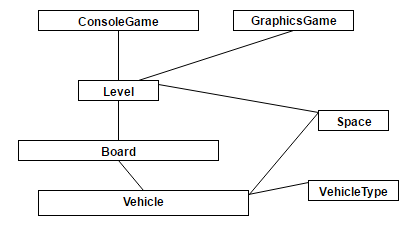
## The Graphics-Based Version

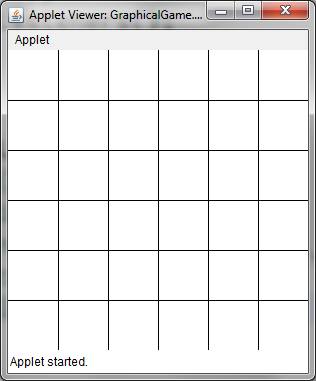
Now that you have the text-based version up and running (hopefully), you can work on the Graphics version of this game. If you don’t have a board that you are confident in, you should look into the additional document where you can use a helper jar instead of your own code that has a solution to the board for you. (More details can be found in the “Adding the Helper Jar” document on canvas. If you decide to use this instead of your own code, your submission will receive a small penalty, so you should only use this if you aren’t confident in your board. Your version of the game will ultimately look something like the version given above, which has the car (represented as the red car), some trucks and autos and the exit (as a red label)

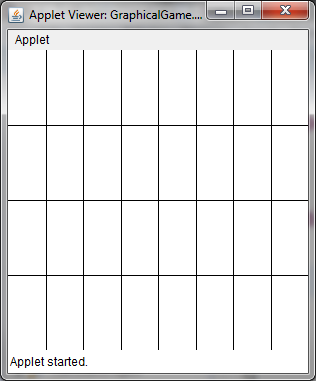
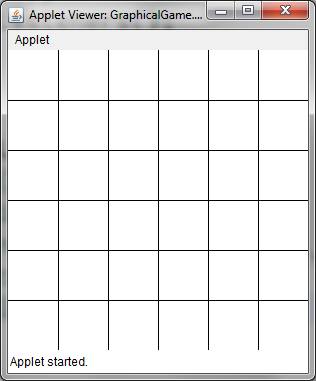
Luckily for us, we already designed the Vehicle, Board, Level and Space classes and will reuse all of them. This means that if we designed our classes right, we should only have to implement a class called GraphicsGame.java and not change anything that we’ve done before. The starter GraphicsGame is already in your traffic jam project.

## The plan of attack

Here is how the GraphicsGame interfaces with the rest of the classes. Notice that it just links up to Level, just like we did with ConsoleGame, and then we let Level be the intermediary to the actual game and its rules. Here are some methods that you’ll want to write (in some sequence of steps)



1. spaceWidth()/spaceHeight() – based on the number of rows and columns for a particular level, return the width/height of such a space. This isn’t a long method to implement (it could be written in one line), and it is meant to help you translate between row and columns and what is on screen. (see right) 
2. drawGridLines()- Draw the lines to make a grid that represents the spaces on that level. Your code should be general enough that it works for any size level with any number of rows and columns in any width. Here are two different grids: a 6x6 and a 4x8, that both take up the same space. Make sure to test this out before you continue!



1. convertXYToSpace(double x, double y) – convert the x, y coordinates given to you into a Space object that you can then use to access a row and column on the board. **Make sure to test this by overriding mouseClicked, and then having a GLabel or simply printing out the space that corresponds to the location of where someone pressed the mouse button. Do this multiple times. Thus, if you click inside the square that should be row 1 column 1, verify that is what gets printed out. As part of your submission you must leave this code in but commented out.**
2. drawWinningTile() – Create a label named exit on the level’s goal tile and draw it appropriately. You’ll notice that there is a constant for the Font that you’ll use here (named LABEL\_FONT). The label should appear in the goal space but does not have to be centered.
3. getVehiclesOnBoard() – this method is one that you want to introduce in to both Level and Board, but what you’ll want is simply to return a list of all the Vehicles that have been added to the game. If we can get a list of all the Vehicles, it will make it easier to work with the mouse events in step 8. Rather than try to construct a list of vehicles on the fly, it will be easier if you declare and setup a new ArrayList as an instance variable, which turns getVehiclesOnBoard into a simple getter function. Then incrementally add the new vehicle into the list each time addVehicle is called.
4. drawCar(Vehicle v) – Given a Vehicle object (named v), draw that vehicle on the screen based on its information. In order to draw a vehicle, you’ll have to create a GImage object. GImages take in a filename, x, and a y. In this case your starter project already has PNG images for you in the images folder, so to use them, you have to just specify the path “images/” followed by the name of the file that you want “car.png”. Notice that rather than having you rotate images, to do the vertical images, I have given you separate image files that have the same name as their counterparts, except they have a “\_vert” added to the end of the filename, so “car.png” becomes “car\_vert.png”. Just call the right one at the right position and you should be good to go. There are constants at the top of the GraphicsProgram that have some of these labels done for you. If you notice, the VehicleType’s toString method returns the same name as the filename you’ll want to call. You also want to make sure to set the size of the images so that they take up the appropriate number of spaces on the grid. Make sure to run this too and check to see that it works. The images should resize based on the number of rows and columns they take up on the grid.
5. drawLevel() - use all of your cool helper methods to get a screen that looks like the first image on page 1. Notice the 0 at the top left is the number of moves so far performed, so you’ll have to create and add that label too. You can either make a helper method for that or just make it in here.
6. Create one additional method called calculateSpacesMoved(), which will determine how many spaces a user wanted to move from their original location based on the user’s dragging of the mouse when they first started clicking on the mouse (will you need to store that information?). That is great to have when you ask the level to move a particular vehicle. For this assignment, you should not make the vehicle stop at a particular location if they can’t move any further. Wait until they release the mouse, and then use calculateSpacesMoved to figure out how many spaces you are supposed to move. Remember to use the logic from the interactivity lab to help you figure out the number of spaces, using the change in distance between where the mouse was pressed and then released. Be sure to use the x and y coordinates in both cases. To help make sure that you have this done, **You will be required to store the original location when a user clicks on a space, and then when the mouse button is released, to print out the result from calculateSpacesMoved based on where they are and where you released the mouse.** Use this small interaction to test that your calculateSpacesMoved is indeed working.**You must also print out the car that was clicked on initially and verify it is in fact the right car.**
7. Use the methods above to finish implementing all of the mouse events and event handlers needed for the game. The five mouse event handlers were mousePressed, mouseDragged, mouseReleased, mouseClicked, and mouseMoved, with each passing in a MouseEvent e. You don’t have to implement all of them, but you’ll have to decide how you want the code to behave. After the user has attempted to click, drag, and **then let go of an object** is *the only time* you should ask the board if it is possible to move to that location. You can ask by sending the request to move, just like you did in the ConsoleGame, using calculateSpacesMoved to help you know how far the vehicle should move. Then once the move happens, just update the car’s location. Since the car won’t move if it can’t, you can just ask the Vehicle where it currently is on the board, and just set its image location to that spot. If it moved, the Vehicle will give you a new row and column. If it didn’t, then it will give you back the original. Regardless you can set the location using that space and the calculations you have done with space width and height.

Oh and if you get a ClassCastException when trying to drag a Vehicle, it may be because you actually clicked on a GLabel instead of a GImage (ie vehicle). You don’t need to worry about it, however, if you do want to fix it, it is most likely because you are using getElementAt and then directly casting that as a a GImage when it is not, which raises the exception. To get around it, you can create an if statement that checks the type of getElementAt, making sure it is a GImage before proceeding. Introducing something early in mousePressed like:

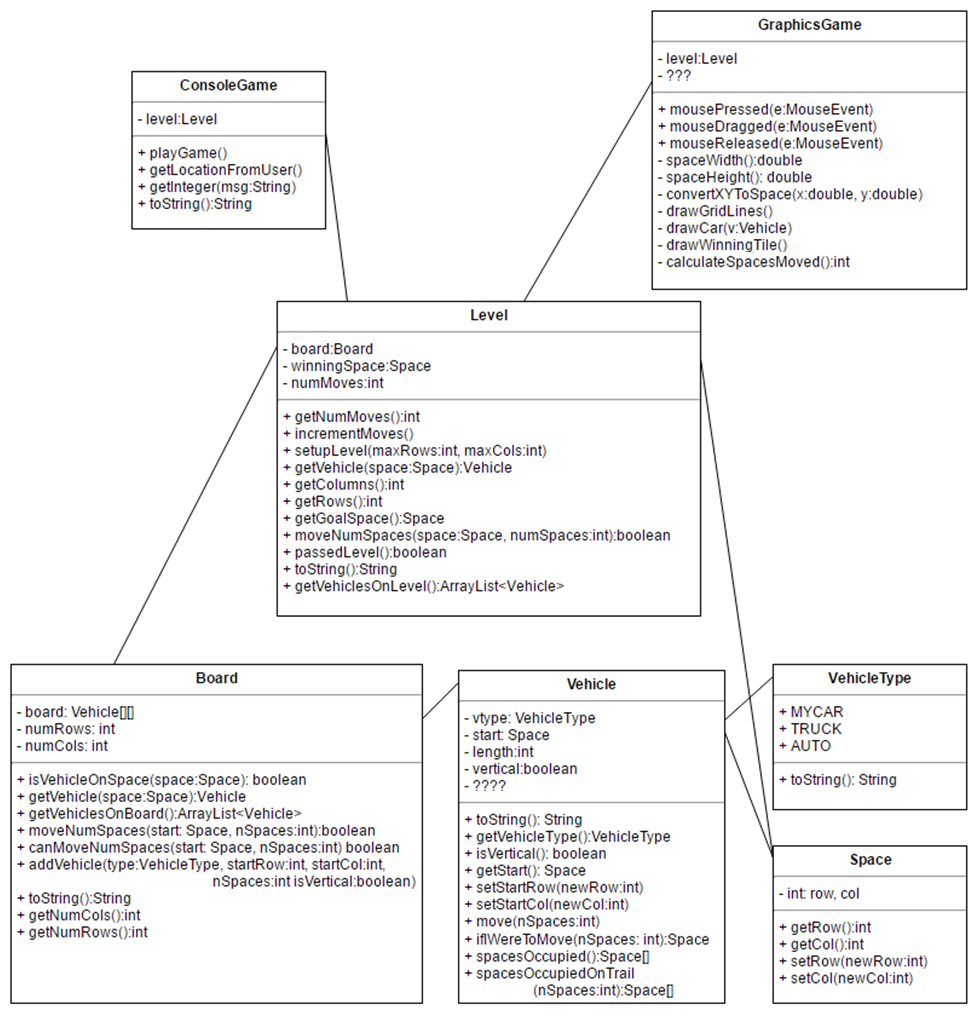
obj = getElementAt(e.getX(), e.getY());

if(!(obj instanceof GImage))) return;

should help to make sure you clicked a GImage.

1. Implement how to win. After each movement that a user makes, you should add the winning condition to check to see if the board is solved. If It is, then you can call removeAll() from the screen and then wipe it out, and add a new label that gives them a congratulatory message.

In order to give you a better handle on everything that you need to write, I’ve revised the UML model I gave, which shows in more detail the methods you should write for GraphicsGame. The challenge in this project is not the amount of code you need to write – If you write it correctly, the amount of code can be quite small, which makes it easier to debug. The real challenge then is in figuring out the math that is necessary to translate the x and y coordinates into movements on the row and column for the board, and leveraging everything you have already written so far and applying your finished labs towards this project. Please Please PLEASE, start right away. You don’t have much time.



Other than getVehiclesOnBoard, all of your work for this assignment should go into GraphicsGame. Nothing in the UML diagram has changed.

The resources for this assignment are provided to you via teams, but here they are just in case.

|  |  |  |
| --- | --- | --- |
| Traffic Jam Graphics Help | ["Narration Graphics Help"](https://www.dropbox.com/s/qjederptsbz5558/Narration%20-%20Traffic%20Jam%20Graphics%20Help.pptx?dl=0) | Narrated powerpoint describing some of the issues you may encounter with graphics |
| Graphics Troubleshooting | ["SOS Website for Graphics"](http://appdevhelp.surge.sh/tjgraphics) | Handy checklist to troubleshoot your graphicsgame |
| Traffic Jam Graphics Assignment | ["Assignment Description for Traffic Jam Graphics"](https://www.dropbox.com/s/t50ap8n5ms5fus8/COMP%2055%20-%20Intro%20Project%20Graphics.docx?dl=0) | Most up-to-date version of the assignment |
| Board Helper Jar | ["Helper Jar"](https://www.dropbox.com/s/17ry4569tric3w8/BoardHelper.jar?dl=0) | Latest version of the jar needed if you would like to use it for the graphics portion (only if you really need it) |
| How to Add Board Helper | ["How To Add Board Helper"](https://www.dropbox.com/s/peh6tn3s8xi0gfu/Adding%20Helper%20Jar%20%28only%20if%20Board%27s%20broken%29.docx?dl=0) | Latest instructions for how to Add the Board Helper if you need it |
| Diagram of how Graphics will work | ["Sequence Diagram"](https://www.dropbox.com/s/gooez0ulgcmgy3w/Traffic%20Jam%20Sequence%20Diagram%20Examples%20only.pptx?dl=0) | Latest version of the sequence diagrams which provides a little bit of understanding for how the game will work and interact |