12 Final Project

Project goals and logistics

The goal of this assignment is to give you the opportunity to use all the concepts we have learned in the design and implementation of a small project. The project will involve user interactions, but the core is the design of the functionality, following the design recipes, building abstractions, and using the existing libraries. Good design and a simple user interaction part is much more valuable than a flashy user interactions with poor design of the program that drives it.

You should finish the design and implementation of the *engine* for your project by Thursday, April 9th. You will then have five days to work on the user interactions and on improving the program design, with the final version due on Tuesday, April 14th.

The project presentation will be done in the lab in 212 WVH on Thursday, April 9th, Monday, April 13th during the regular lecture times and on Tuesday, April 14th during the regular lab times.

Project details

Choose one of the four options for the project. We give you fairly detailed specifications for the model part of the program. The requirements for the view are minimal. They only describe the basic functionality — you are free to enhance your presentation as you wish — and will be given credit for the work.

One part of the credit for this assignment will be given for a design document that describes the data, the organization of the program, the key program components, and the design of tests. Imagine you want someone to keep improving your program — provide a road map that explains what your program does and how does it do it. This document should complement the Javadoc generated web pages. A separate document will give you a more detailed guidelines for what we expect.

One part of the credit for this assignment is for the model part.

You will also get credit is for the user interactions (view) — grading both the design of the user views and the design of the program that drives it. A small bonus may be earned for exceptionally well designed display or interactions. It is better if the user interaction is done only through the console, but is well designed and documented, than if a fancy GUI display is driven by a code that another programmer cannot understand and maintain.

12.1 Project Presentation

You will present your project (both partners together) during the times given above. Each partner should be able to describe any part of the code in the project, regardless of *who wrote it*, as we expect that both partners work on the project together. More information about the presentations will be provided shortly.

12.2 Option: Connect Four

The Model

ConnectFour is a two-player game played using a vertical rack that has seven tracks of height six. Each track holds the game pieces that have been dropped into it. Once a game piece is dropped into the track, it cannot be moved or removed.

The players take turns dropping game pieces into the rack. Their goal is to line up four of their pieces in a row, in a column, or in a diagonal, and prevent the opponent from achieving this goal. A player cannot drop a piece into a track that is full (has seven pieces in it already).

The game ends when one player *connects four* or it ends in a tie if all slots in the rack are filled without either player winning the game.

If the player makes an illegal move, you may choose to disqualify the player, or allow the player to make another move.

The Interactions

The program should display the state of the game at the beginning, after each move, and when it declares the end of the game. The following:

```
o o o o o o o
b o o b o o o
r b o r b o b
r r b b b r r
r r r b r r b
b b r b r b r
Black wins
```

shows the state of the game after the black player won by placing four pieces on a diagonal.

The program should allow each player to make a move and report the end of the game.

12.3 Option: Dice Game

The Model

The *Dice Game* is played by two to five players by rolling five dice. You may limit your game to two players.

Players take turns rolling the dice (one or more times), accumulating points for the combination of dice thrown. The first player to reach 10000 points wins.

Once the player had a turn with a score of at least 750, the score is recorded and any additional score earned is added to the accumulated total. Until then, a score under 750 is discarded.

The score for one roll of dice is computed as follows:

Roll:	Score:
1 1 1	1000
2 2 2	200
3 3 3	300
4 4 4	400
5 5 5	500
6 6 6	600
1 alone	100
5 alone	50
12345	1000
23456	1000
123456	10000

If the player scores any points on the first roll, he may keep the score and roll again only the dice that did not count towards that score. The score from the roll of the remaining dice is added to the earlier one. If all dice had been used towards the score, the player may roll all dice again. If a roll of dice at any time during the player's turn results in a score 0, the player looses all score accumulated during this turn and his turn ends. The player may choose to end the turn after any roll.

Here is sample progress of the game:

```
Player 1: Score at the start of the round: 0
Roll:         Score for this roll:  Total for this round:
1 2 5 5 6          200                   200
keep 1 5 5 and roll again two dice
4 5                 50                   250
keep 1 5 5 5 and roll again one die
4                    0                   0
** no score on this roll ends the round with no score
Score at the end of the round: 0

Player 2: Score at the start of the round: 0
Roll:         Score for this roll:  Total for this round:
6 6 6 3 4          600                   600
keep 6 6 6 and roll again two dice
1 5                150                   750
** player chooses to end the round
Score at the end of the round: 750

Player 1: Score at the start of the round: 0
Roll:         Score for this roll:  Total for this round:
3 5 5 5 6          500                   500
keep 5 5 5 and roll again two dice
5 5                100                   600
roll again all five dice
1 1 3 4 6          200                   800
** player chooses to end the round
Score at the end of the round: 800

Player 2: Score at the start of the round: 750
Roll:         Score for this roll:  Total for this round:
2 2 2 4 6          200                   200
** player chooses to end the round
Score at the end of the round: 950
```

The Interactions

The program should show each player's score. It should be clear which player is taking his turn. During the player's turn, the program should show the dice values, the score for that roll, and the accumulated value for this turn.

At the end, the program should declare the winner and show the scores for all players.

12.4 Option: Traffic Simulation

The Model

The goal of your program is to simulate the traffic of cars traveling through an intersection controlled by traffic lights. The simulation measures the average wait time at the light, the maximum wait time at the light, the average length of the queue of cars waiting to move on, and the maximum length of the queue of cars waiting to move on.

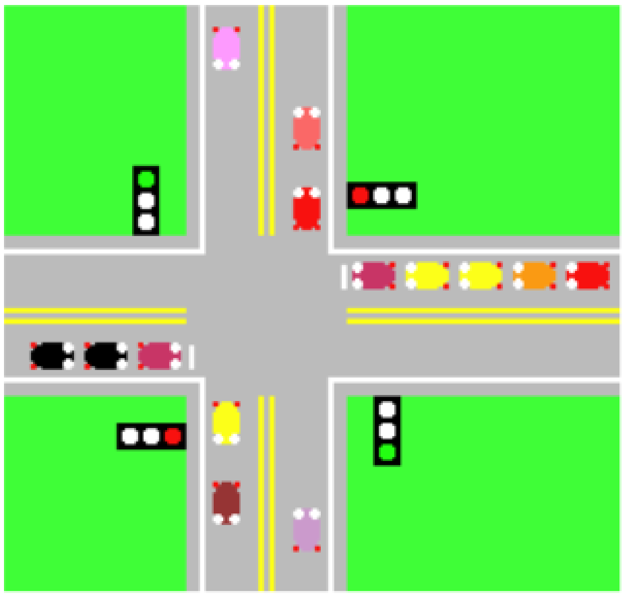
The intersection is a crossing of a two-way road that goes in the North-South direction with a two-way road in the East-West direction.

The user can set the duration of the red and green lights, as the duration of the orange light is the difference between the red and green lights (while the light is red in one direction, the other direction is green, then changes to orange, then to red while in the other direction the light changes to green).

The user can set the rate at which the cars arrive into the traffic lanes, as well as the duration of the simulation.

The cars move at a constant speed, though you are free to expand the simulation to allow each car to travel at its own speed, provided there are no collisions with the cars ahead.

Design the model with a fixed values for the user choices, then work on the user interactions.



The Interactions

At the minimum the user should be able to set the timing for the traffic light and the duration of the simulation. At the minimum the simulation should report the number of cars that passed through the intersection in each direction.

At each tick of the clock, the simulation should report the number of cars waiting in each direction, as well as the current light settings.

You may display the scene at the intersection as a scene on the Canvas, but are not required to do so.

You should add the reports listed above and the user's choice of the traffic load, i.e. the frequency at which the cars enter the traffic lane(s) once your program works.

12.5 Option: Graph Algorithms

The Model

Graph

Your program needs to represent a graph with nodes that represent capitals of the 48 US states. Each node has a name — the name of the state. For each node, record the information about the capital of that state. Each edge represents a bi-directional connection between two adjacent states. You may consider the *four corner states: Colorado Utah, Arizona and New Mexico* as connected to each other. Each edge has a value that represents the distance between the capitals of the two states. The distances between two cities are based on the geographic distance. (See a separate announcement for a shortcut you can use to compute this distance.)

Algorithms

Your model should implement three graph traversal algorithms:

- Depth-First Search: uses a *Stack* to record the *ToDo* information
- Breadth-First Search: uses a *Queue* to record the *ToDo* information
- Shortest Path Search: uses a *Priority Queue* to record the *ToDo* information

To implement the shortest path you need to represent a priority queue.

The detailed description of the algorithm appears in a separate document. You will encounter a significant penalty for repeating the code - one algorithm implementation should run all three variants, distinguishing between them by selecting the appropriate implementation of a common interface for dealing with the *ToDo* information.

Using Libraries

Furthermore, throughout the project you are encouraged to leverage as much as possible from the existing Java libraries (both the Java Collections Framework, and the JPT libraries). The designer should focus on the design of interfaces between tasks, between components, wrapper and adopter class that allow you to use an existing library class in a customized setting.

The View

The requirements

The view at the minimum should have the following functionality:

- User should be able to see a representation of the graph.

  This can be a graphical display, a text that lists the nodes and the edges (with their weights), a graphical display of the text that lists the nodes and the edges.

- User should be able to select which of the three algorithms is to be used for the subsequent task.

- User should be able to specify the origin and the destination of the desired path.

- The user should be able to see the resulting path.

  Again, this may be a graphical display, or just a text listing the nodes along the path.

The frills

Of course, the view can be much more elaborate. Here is a list of possible enhancements:

- Highlight the path is a different color in the graphics display.
- Display the steps in the search by highlighting in a different color the *v*isited nodes, the *f*ringe nodes (those currently in the queue or the stack), the *o*rigin, the *t*arget, and the *u*nseen nodes. Animate the process using either the timer, or a user advance triggered by a key press.
- Animate the reconstruction of the path by traversing from the found target back to the previous node, all the way up to the origin.
- Select the origin and the target in a graphics display using a mouse.
- Display in a GUI the path length and possibly the nodes along the path.
- Choose the algorithm through a GUI.
- Make the graphics look like a game — e.g. traversing a street map or a maze.

12.6 The Advice

The design part of each project typically takes the greatest amount of time. the more time you spend thinking things through, the easier it is to actually write the code.

Make sure you think the whole framework through before you start programming. Spend some time researching the Java libraries to see what tasks can be done using the existing tools. Write sample adapters to see how the existing class can be used in your setting.

Then design the key component by specifying their interfaces — the method headers, the interfaces that various classes must implement or use to get information from others.

For now, you have not learned about various tools and techniques to support such design process — other than class diagrams. Any description that you find helpful in clarifying the roles of the different classes and interfaces in your program is acceptable.

The design document you produce should include a brief user's guide, give a general overview of the project organization as well as describe all data definitions and the key methods. The Javadocs supplement this with detailed information about the actual implementation.