

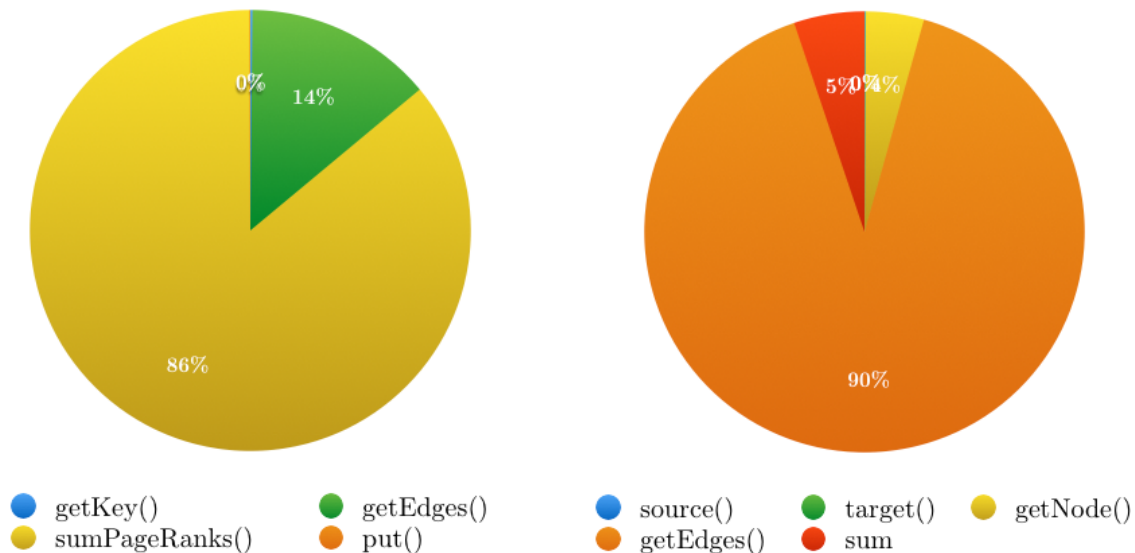
Scotland Yard

Julian Loscombe and Ben Milne

April 27, 2015

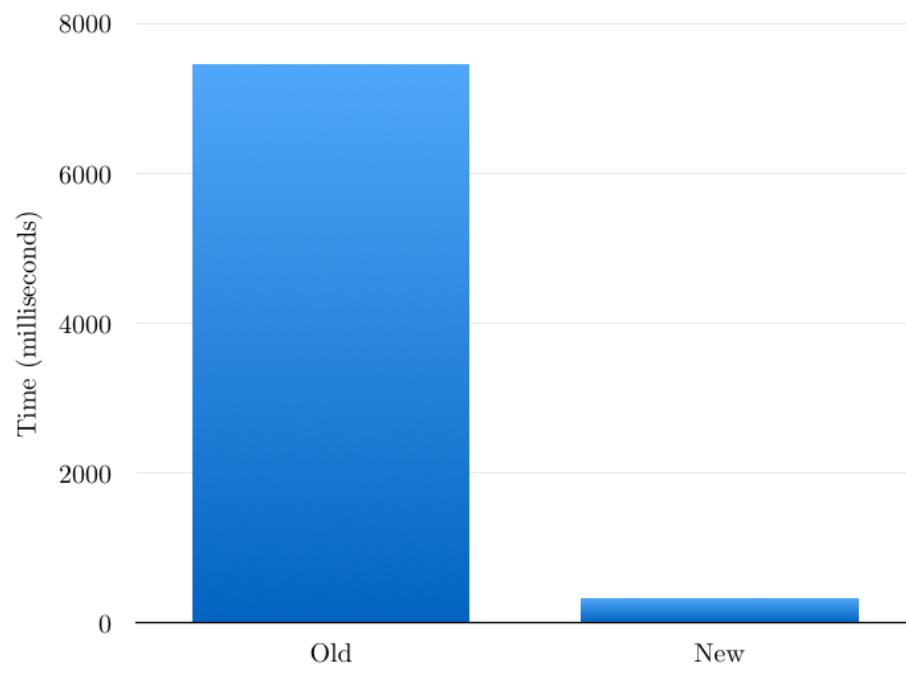
1 Transitional Issues

The update to the project files part-way through the project caused a significant lag (-10s) when starting a new game. We tracked this down to the Page Rank algorithm, specifically the `iterate()` method which is called 100 times at the beginning of a game. In order to see clearly where the problem was we used the `System.nanoTime()` method to find the time each line of the method took.

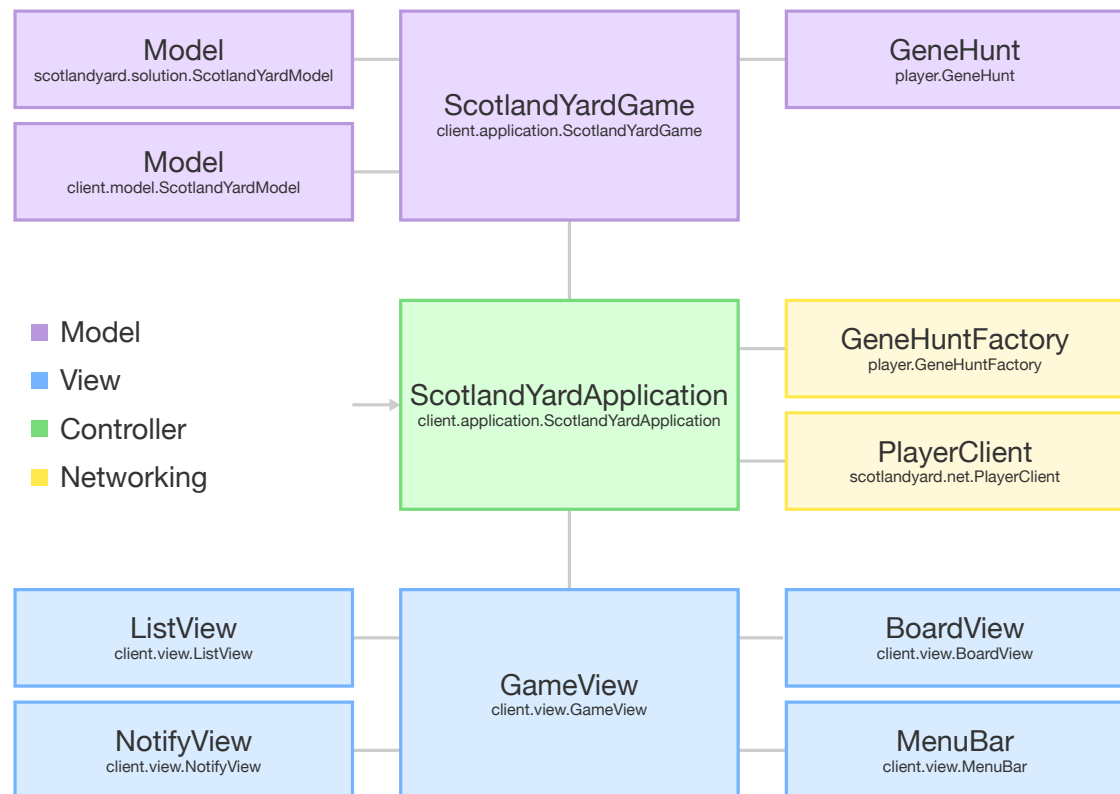


Clearly then, the issue is with the `getEdges()` method in the Graph file. We believe this is because of the change from using Lists to Sets. The cost of iterating over a Set, it seems, is far greater than doing the same for a List as illustrated at <http://javacodegeeks.com/2010/08/java-best-practices-vector-arraylist.html>.

To fix this we created a map on initialisation which maps a node number to its edges. This means we only need to iterate over the Set once and the result is a significant speed increase.



2 Project Schematic



As our project grew it became important to organise our code. To achieve this we came up with the above project structure. You can clearly see the separation of the model and the views.

3 Animations

In the first part of the project we did not have sufficient time to add animations to the game. We feel these are important as they guide your attention and allow you to keep track of what is happening more easily. In order to easily implement our animations in this part we created a class called `AnimatablePanel` which is documented separately.

4 Benefits of our GUI

We decided early on to use our own GUI, this is because we feel our GUI has four main advantages over the one provided. Our GUI:

1. Automatically make a random move when the time to take a move runs out. Therefore the game will not quit if you take too long.
2. Allows panning and zooming which especially useful for low resolution screens. The provided GUI does not resize and hence on some screens a portion of the board is cut off and unreachable.

3. Shows the suggested routes generated by the AI and all the valid moves you can take.
4. Shows the time remaining for you to make a move clearly, we found the timer on the provided guy was often off screen and impossible to see.

5 Disadvantages of our GUI

Although we are generally very pleased with our GUI, we have found some limitations of it:

1. It is difficult to see what moves Mr X has previously made as we have used a chat view to communicate game events. In hindsight this is not the best method and the provided GUI is much better for this. We decided to use this method before we knew the limitations of the networking, hoping to enable client to client chat.
2. It is very difficult to see what tickets other players are holding. Again we hoped to mitigate this issue using the chat.

6 The horrible bug with the networking

When connecting our GUI to the networking part of the project, we encountered a difficult to trace bug, as you may remember. The problem was that the `PlayerClient` class calls `model.start()` in the `run()` method, but we also called `model.start()` on another thread as we did not know that it had already been done. This meant that the game was effectively started twice on two Threads causing havoc with the updating of our GUI. To solve this problem we changed our code so only the `PlayerClient` called the `model.start()` method. It would have been helpful to either add the synchronised tag to the `model.start()` method or add some documentation about the `PlayerClient` class.

7 Documentation

We have fully commented our code to the JavaDoc specifications and have generated the html, you can find it in the `docs/Documentation/JavaDocs` folder.