A continuación os dejo algunos consejos para el desarrollo la segunda practica. Antes de pasar al siguiente punto asegurarse de que el actual funciona bien.

- Implementar la primera parte, la de modificar el modelo y el controlador. Es poco código, tardáis una hora en hacerlo.

  Para comprobar que todo funciona bien, implementar una "vista" sencilla, como la de consola del SlidePuzzle, y usarla en el batch mode para ver que recibe las notificaciones bien.

  Quitar esta vista del startBatchMode después.

- Modificar la clase Main para que tenga el parámetro -m con valor por defecto "gui" (ver la última parte de enunciado)

  Clonar el método startBatchMode a startGUIMode y quitar la llamada a run. Esta llamada la remplazamos con la construcción del MainWindow (ver la última parte de enunciado).

  Más adelante tenéis que hacer la opción -i opcional en el caso de GUI, pero no lo hagáis ahora.

- Implementar MainWindow, sin ningún componente, simplemente que abre la ventana. Y modificar startGUIMode para que crea la ventana.

- Implementar la primera versión del ControlPanel, que incluye sólo un botón RUN que llama a _ctrl.run(10) -- además del botón quit que ya lo tenéis en el enunciado. Añadir el ControlPanel al MainWindow.

- Implementar un componente (una clase) SimpleView que extiende JPanel y implementa el SimulatorObserver. Este componente tendrá un JTextArea txt donde vamos escribir el contenido del mapa al recibir las notificaciones onAdvance y onRegister. Por ejemplo usando txt.setText(map.toString()) o txt.append(...)

  Es como la primera vista swing del SlidePuzzle, algo muy sencillo para ver que todo está bien conectado.

  Añadir este componente al MainWindow y ver que todo funciona bien al pulsar el botón RUN.

  Hay que quitar este componente más adelante, cuando implementes las tablas, etc.

- Implementar el botón LOAD para cargar nuevo archivo de entrada, y hacer la opción -i opcional en el modo GUI.

- Cambiar el comportamiento del botón RUN para que use el método run que aparece en el enunciado, y llamarlo con un valor fijo, p.ej. 100.

- Añadir el botón STOP al ControlPanel y implementa su funcionalidad como se indica en el enunciado.

- Añadir al ControlPanel un JTextField para el delta-time y usarlo cuando llames a run (es decir llamar a _ctrl.setDeltaTime antes de empezar la simulación).

- Añadir al ControlPanel un JSpinner para el número de pasos y usarlo cuando llames a run (es decir llamar run con el número de pasos correspondiente). Como primera versión podéis usar un JTextField en lugar del JSpinner.

- Implementar la clase InfoTable, pero como primera versión que

muestro información fija (es decir pasar la información como array String[][] a la constructora del JTable, no conectarla a ningún modelo).

Añadir un componente de InfoTable al MainWindow y ver si se bien bien la tabla, etc.

- Implementar el modelo de tabla para los grupos (GroupsTableModel) y cambiar InfoTable para que use un modelo (en lugar de información fija). Añadir la tabla de grupos al MainWindow.

  IMPORTANTE: Estudia los ejemplos en extra.jtable antes de empezar.

- Implementar el modelo de tabla para los cuerpos (BodiesTableModel) y añadir la tabla de cuerpos al MainWindow.

- Implementar el StatusBar y añadirlo al MainWindow. Ese componente es muy sencillo, usar el StatusBar del SlidePuzzle como ejemplo.

- Añadir el componente Viewer en ViewerWindow y añadir el botón correspondiente al ControlPanel -- sin hacer ningún cambio en Viewer de momento. Comprobar que la ventana abre correctamente, etc.

- Modificar la clase Viewer para gestionar la tecla 'h', es decir gestionar la tecla en keyPressed y implementar showHelp.

- Modificar la clase Viewer para que dibuje todos los cuerpos (es decir implementar drawBodies) y modificar ViewerWindow para que llame a _viewer.addBody, _viewer.addGroup y _viewer.update cuando sea necesario (ver el enunciado). Ahora al ejecutar la simulación tenéis que ver la simulación en la ventana del ViewerWindow (lo cuerpos tienen que mover como en el visor HTML que habéis usado en la primera práctica).

  Como versión inicial, simplemente dibujar todos los cuerpos con el mismo color y no dibujar los vectores.

- Modificar drawBody para dibujar los vectores de los cuerpos, y usar un color distinto para cada cuerpo.

- Modificar la clase Viewer para gestionar de la tecla 'v', es decir gestionar la tecla en keyPressed y cambiar drawBodies para que muestre/oculte lo vectores.

- Modificar la clase Viewer para gestionar la tecla 'g', es decir gestionar la tecla 'g' en keyPressed y cambiar drawBodies para que muestre solo los cuerpos del grupo seleccionado.

- Modificar la clase Viewer para gestionar las teclas l, j, m, i y k.

- Implementar el dialogo y el añadir el botón correspondiente al ControlPanel.

  Empezar con una versión sencilla, sin la tabla de parámetros, simplemente al seleccionar un grupo y unas leyes, construimos el JSON correspondiente sin la sección data y lo pasamos al método _ctrl.setForceLaws (el builder correspondiente va a usar valores por defecto si es necesario, como g,c,etc).

  Una vez la primera versión funciona bien, ya podéis añadir la tabla de parámetros al dialogo.

  IMPORTANTE: estudiar lo ejemplos en extra.dialog antes de empezar.

- En ControlPanel, desactivar los botones (menos el STOP) antes de empezar la simulación y activarlos cuando acabe.

- comprobar que estáis capturando los errores que puede lanzar el controlador/modelo (en ControlPanel y en el dialogo de leyes de fuerza) y mostrar los mensajes de error correspondientes.