LIQUID: 2-D FLUID DYNAMICS SIMULATOR

Software Developers Guide

Version 1.0

12/14/2015

PRYMM
Pavan Kumar Gade, Radhika Panchal, Yashraj Sinha, Minghua Liu and Manoj Mathe

Prepared for
Principles of Software Engineering
Fall 2015

Revision History

Date	Description	Author	Comments
12/14/15	Initial draft version	Yashraj and Minghua	Version 1.0 – Complete document is prepared by Yashraj and Minghua.

Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

Signature	Printed Name	Title	Date

Table of Contents

1. Introduction

This section provides an overview of the Software Development Guide for the project called LIQUID which is a 2-D Fluid Dynamics Simulator. The purpose of this document is to outline the important code structures of the code to any developer who wants to refer and do the modifications to the code of the project.

1.1 Purpose

The purpose of the Software Development Guide is to give outline of the way the code is arranged for the project. This document provides information for someone who is new to this projects and would like to modify the code with ease.

1.2 Scope

This document provides information for someone who is new to this projects and want to modify the code or integrate their code to this project. The GUI and rendering engine of the project is designed as two separate modular entities. This provides opportunity to replace and integrate either of GUI or rendering engine of their own.

1.3 Definitions, Acronyms, and Abbreviations

Here in this section it describes about the definitions and acronyms and abbreviations that we are using all over the Software Development Guide document. This is done for better understanding of the reader about the terms that is used throughout the document.

- GUI: Graphical User Interface
- DFD: Data Flow Diagram
- STD: State Transition Diagram

1.4 References

- https://nerget.com/fluidSim/
- http://physics.weber.edu/schroeder/fluids/

1.5 Overview

The complete code of the project is arranged in a structured manner to support maintainability. As can be seen from the figure below that the complete source code of the project is divided into six major packages as follow:

- prymm.controller
- prymm.databean

- prymm.gui
- prymm.model
- prymm.test
- prymm.util.

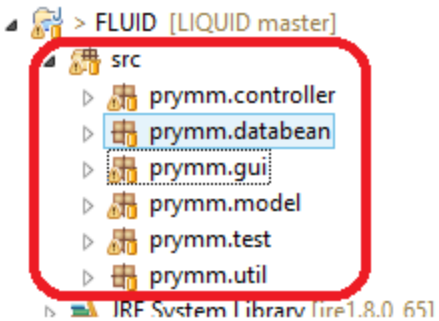

Figure-1 List of packages in source code of project

And in the following sections of document individual package with its list of files will be elaborated.

2. Package prymm.controller

This package in the project is responsible for making the project modular. This package act as an interface between prymm.gui package and prymm.model packages. Making developers and integrators to replace the GUI package or model package with any other project with minor changes needed.

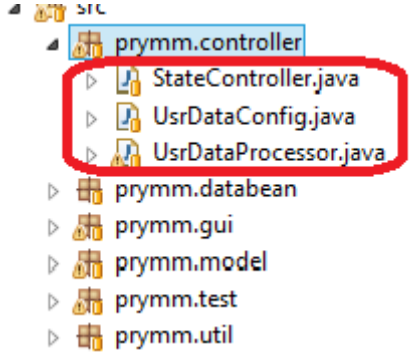

Figure-2 List of files in prymm.controller package

Prymm.controller package has three important files as can be seen from the figure namely StateController,UsrDataConfig and UsrDataProcessor.

- **StateController File:** This file is used to manage the present state of the application based on state transition diagram of the project such as initial, idle, running, pause and terminal

```
 */
public class StateController {

    /**
     * When updating the state, use below defined ones
     */
    public static final int INITIAL = 0;
    public static final int IDLE = 1;
    public static final int RUNNING = 2;
    public static final int PAUSE = 3;
    public static final int TERMINAL = 4;

    private static int CURRENT = 0;

    public static int getCurrentState()
    {
        return CURRENT;
    }

    /**
     * Used to set current state, below are the state available
     * StateController.INITIAL
     * StateController.IDLE
     * StateController.RUNNING
     * StateController.PAUSE
     * StateController.TERMINAL
     * @param state
     */
    public static void setCurrentState(int state)
    {
        CURRENT = state;
    }
}
```

Figure-3 Code snippet of StateController class

- **UsrDataConfig File:** This file provides the interface for user configurable parameters which gets set by prymm.gui package and used by prymm.model package for rendering engine. This file has all the user configurable parameters provided to the user at GUI like fluidType, viscosity etc. as shown in the figure below.

```
public class UsrDataConfig
{
    private static int totalNumber = 0;
    private static UsrDataConfig usrdata = new UsrDataConfig();

    private String fluidType;
    private String viscosity;
    private String temperature;
    private String barrierShape;
    private String initialForce;
    private String initialSpeed;
    private String containerSize;
    private boolean isEntryAdded;
    private boolean isExitAdded;
    private int length;
    private int width;

    private String replayPath;
```

Figure-4 Code snippet of UsrDataConfig class

- **UsrDataProcessor File**: This file implements methods used by prymm.gui package for the transition from one state of the system to another such as idle to running. This file also consist method to generate and retrieve log used for providing logging and replay functionality.

3. Package prymm.databean

This second package in source code folder is prymm.databean. This package in the project contains data specific to each type of fluid and the data structure of a single dot which represents nine matrices in a lattice. As can be seen from the figure this package contains files water and glycerin which contains data which are specific to these fluid such as formula to calculate viscosity based on temperature. The third file in this package is a SingleDot which represents nine matrices in a lattice which represents four pixels in simulation canvas.

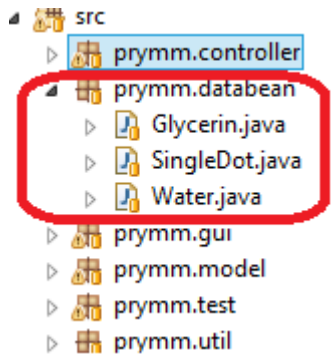

Figure-5 List of files in prymm.databean package

4. Package prymm.gui

This third package in source code folder is prymm.gui. As the name suggests this package in the project contains code for handling GUI related functionality such as default page and welcome page as shown in image below. The two major files in this package are WelcomePage and FluidDefaultPage.

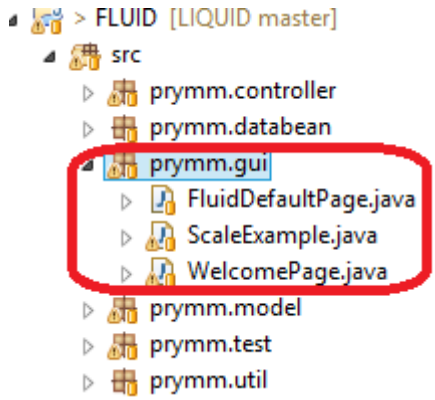

Figure-6 List of files in prymm.gui package

- **FluidDefaultPage File:** This file has major classes which are inherited by welcome page and contains window, display and canvas as can be seen in the figure below.

```
public class FluidDefaultPage
{
    /**
     * Should be configured here
     */
    protected static boolean isReplay = false;
    protected static String replayFileName = "";
    protected static String logFileName = "";

    private static FluidDefaultPage window;
    protected Canvas canvas;
    protected Display display;

    public Display getDisplay() {
        return display;
    }

    /**
     * get current canvas
     * @return
     */
    public Canvas getCanvas()
    {
        return canvas;
    }

    /**
     * create shell and display object and all the widgets
     */
    public void open() {
        // TODO Auto-generated method stub

    }
```

Figure-7 Code snippet of FluidDefaultPage class

- **WelcomePage File:** This file has the code for the GUI window visible to the user. It also contain all the user action handlers such as selection and configurations from the user. And contains all the text displayed on the GUI as illustrated in a code snippet below.

```
*/
public class WelcomePage extends FluidDefaultPage{

    protected Shell shlFluidDynamicSimulation;
//  private Display display;

    private Button runButton, stopButton, resetButton, btnAddPipeEntry, btnAddPipeExit, getLogButton

    private Combo comboFluidType, barrierShape, initialForceDirection, containerSize;

    private Scale viscosityScale, tempScale;
    private Text tempText;
    private Text speedText;
    private Text viscoText;
    private Text fileText;

    UsrDataConfig usrcDataConfig = UsrDataConfig.getUsrDataConfig();
    /**
     * @wbp.parser.entryPoint
```

Figure-8 Code snippet of class WelcomePage

5. Package prymm.model

This fourth package in source code folder is prymm.model. This package contains models used for fluid simulation and rendering engine. This package contain all the configurations and calculation used to simulate the behavior of the fluid. This package in Driver file has the main function which invokes threads to perform simulation related tasks.

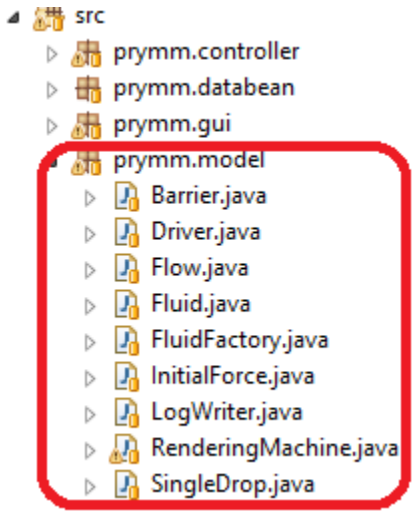

Figure-9 List of files in prymm.gui package

- **Barrier File**: This file implements the disable operation of lattice to form the shape of the barrier on the canvas.
- **Driver File**: This file implements the main function and invokes threads that performs simulation.
- **Barrier File**: This file implements the disable operation of lattice to form the shape of the barrier on the canvas.
- **Flow File:** This file has the class and methods to initialize the flow of fluid based on user configuration
- **Fluid File:** This has the class which inherited by fluid type class such as water.This can be seen in code snippet below in figure 10.

```
public class Fluid {

    /**
     * Determine the viscosity of the chozen fluid
     * Can be changed by type of the fluid chozen by user
     * Or by viscosity value set by user
     */
    protected double viscosity;

    /**
     * Type of the fluid
     */
    protected String type;

    /**
     * Constructor by Fluid type
     */
    protected double temperature;
```

```
public class Water extends Fluid

{

    public Water(double temperature)
    {
        super(temperature);
        type = "Water";
        viscosity = 1.790*Math.exp((-1230-temperature)*temperature/(36100+360*temper
    }

}
```

Figure-10 Code snippet showing the class Fluid and Water

- **LogWriter File:** This file implements the logging functionality and is invoked by UsrDataProcessor file when simulation is started.
- **InitialForce File:** This file is used to setup the initial force of the particles on the extreme end of container based on the flow of direction.

```
public class InitialForce {

    /**
     * Set initial force as per the direction and update all single drops within
     * @param allDrops2
     * @param direction
     */
    public static void initialForce(SingleDrop[][] allDrops2, int direction)

    /**
     * update all drops by adding pipe entry
     * @param width
     * @param length
     * @param allDrops2
     * @param direction
     */
    public static void updatePipeEntry(int width, int length,

    /**
     * update all drops by adding pipe exit
     * @param width
     * @param length
     * @param allDrops2
     * @param direction
     */
    public static void updatePipeExit(int width, int length,
```

Figure-11 Code snippet showing the class InitialForce

- **SingleDrop File:** This has the class which is used to cleate objects which acts as a point/pixel on the canvas.Each particle on the canvas is a SingleDrop and has properties such as velocity and viscosity for each SingleDrop object.Each SingleDrop object also has nine SingleDot associated to them.

```
 * @author minghua
 *
 */
public class SingleDrop
{
    /**
     * Velocity for a single drop of liquid
     */
    private double xVel;
    private double yVel;

    /**
     * Density for the current drop
     */
    private double density;

    /**
     * Used for checking if a single drop is a barrier or not
     */
    private boolean isEnable;

    /**
     * For nine dots needed for calculating each lattice(single drop)
     */
```

Figure-10 Code snippet showing the class SingleDrop

- **RenderingMachine File:** This file has the calculation and engine which simulates movement of particles in container. Methods collision, stream and bounce generates data to be used by the rendering machine to assign colors to the pixel in the canvas.

```
public class RenderingMachine implements Runnable{


    Flow currentFlow = null;
    double four9ths = 4.0 / 9;
    double one9th = 1.0 / 9;
    double one36th = 1.0 / 36;

    private static double[][] density = new double[UsrDataConfig.getUsrDataConf

     * for thread manangement minghua
    private static boolean isRunning = true;

    public RenderingMachine(Flow initialFlow) {


    private void canvasInit() {
    /**
     * calculate all the drops in canvas
     */
    private void doingCalculation() {
```

Figure-11 Code snippet showing the class RenderingMachine

6. Package prymm.test

This fifth package in just a collection of test files used throughout the development of the project and is recommended to use before integrating changes in any of the other project specific packages. As can be seen from the image below it was mainly used to test multithreading concepts.

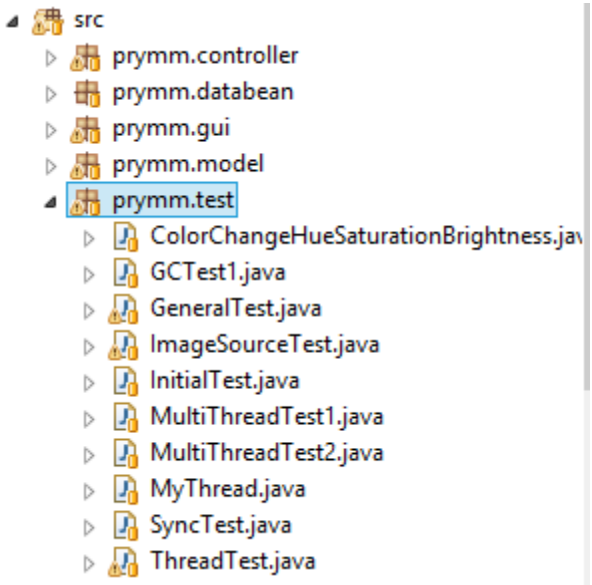

Figure-12 List of files in prymm.test package