Table Of Contents

1. Algorithm Simulator

1.1 Introduction

A **sorting algorithm** is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) which require input data to be in sorted lists; it is also often useful for canonicalizing data and for producing human-readable output. More formally, the output must satisfy two conditions:

1. The output is in nondecreasing order (each element is no smaller than the previous element according to the desired(total order);
2. The output is a permutation (reordering) of the input.

This simulator includes nine(9) sorting algorithms such as Bubble Sort,Insertion Sort,Selection Sort,Merge Sort,Quick Sort,Counting Sort,Radix Sort,Bucket Sort and Shell Sort. The learners will able able to :

- simulate each algorithm and analyze each step that they read in the book.
- differentiate one algorithm to another and decide which is better for what data.
- find the new idea to simulate these algorithms of his/her technique.
- also get a nice idea about how to design GUI based applications.

1.2 Objectives

This simulator is very helpful for CSE students. The main objectives of it are:

- to make better understanding of sorting algorithms.
- to design GUI based application.

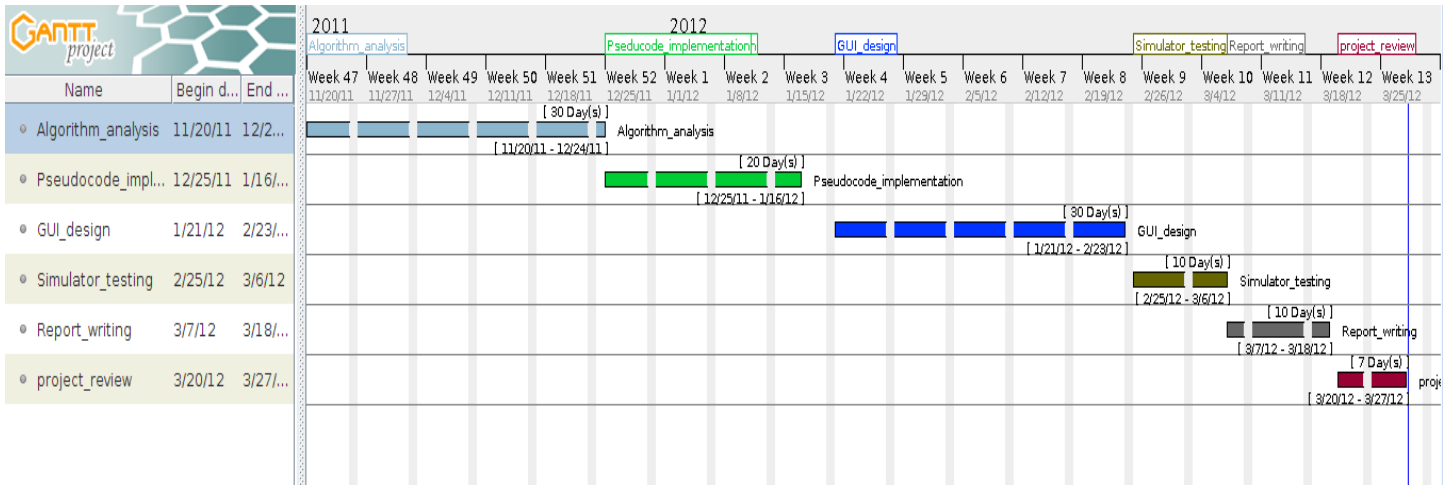
1.3 Analysis

The ideal of sorting algorithm would have the following properties:

- Stable: Equal keys aren't reordered.
- Operates in place, requiring O(1) extra space.
- Worst-case O(n·lg(n)) key comparisons.
- Worst-case O(n) swaps.
- Adaptive: Speeds up to O(n) when data is nearly sorted or when there are few unique keys.

There is no algorithm that has all of these properties, and so the choice of sorting algorithm depends on the application.

1.4 Project Scheduling



1.6 Design

The following figure shows the design prototype

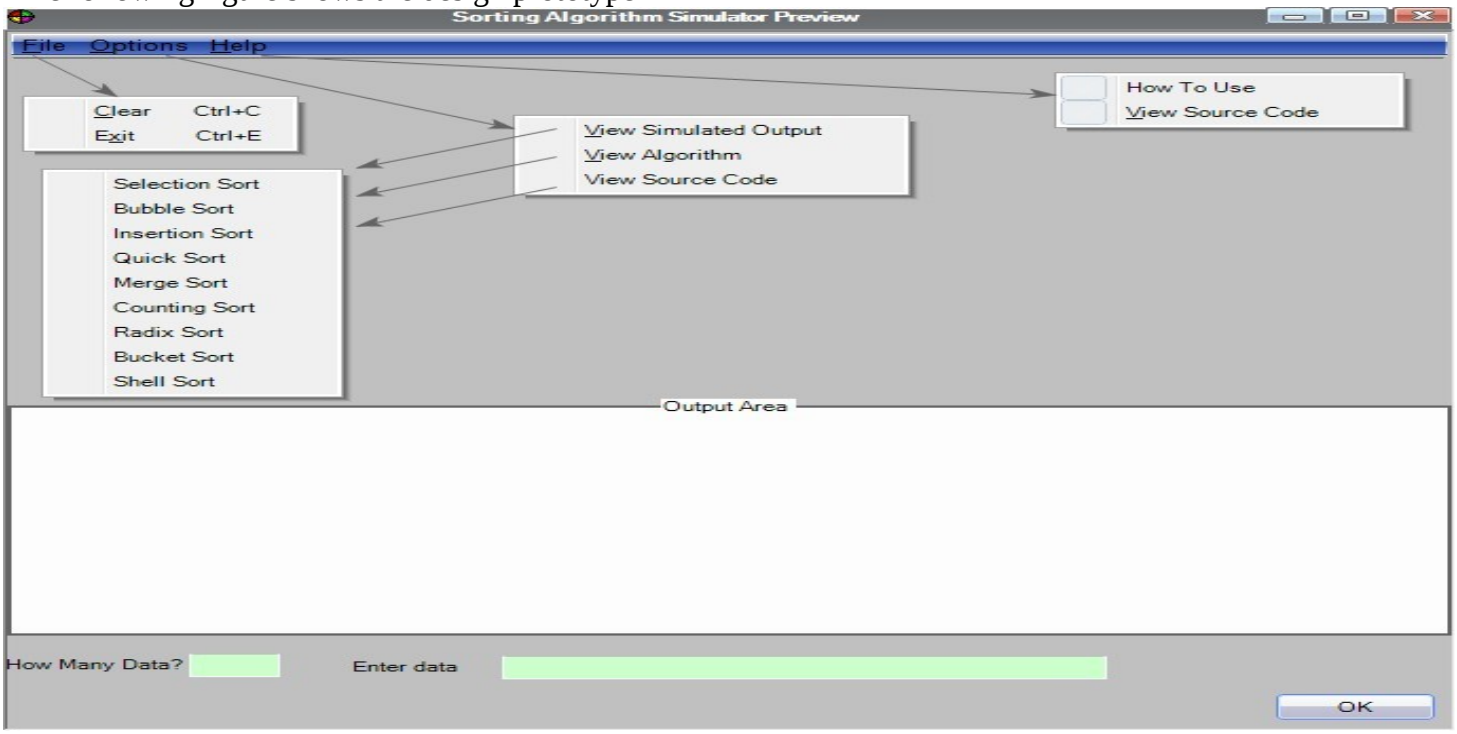
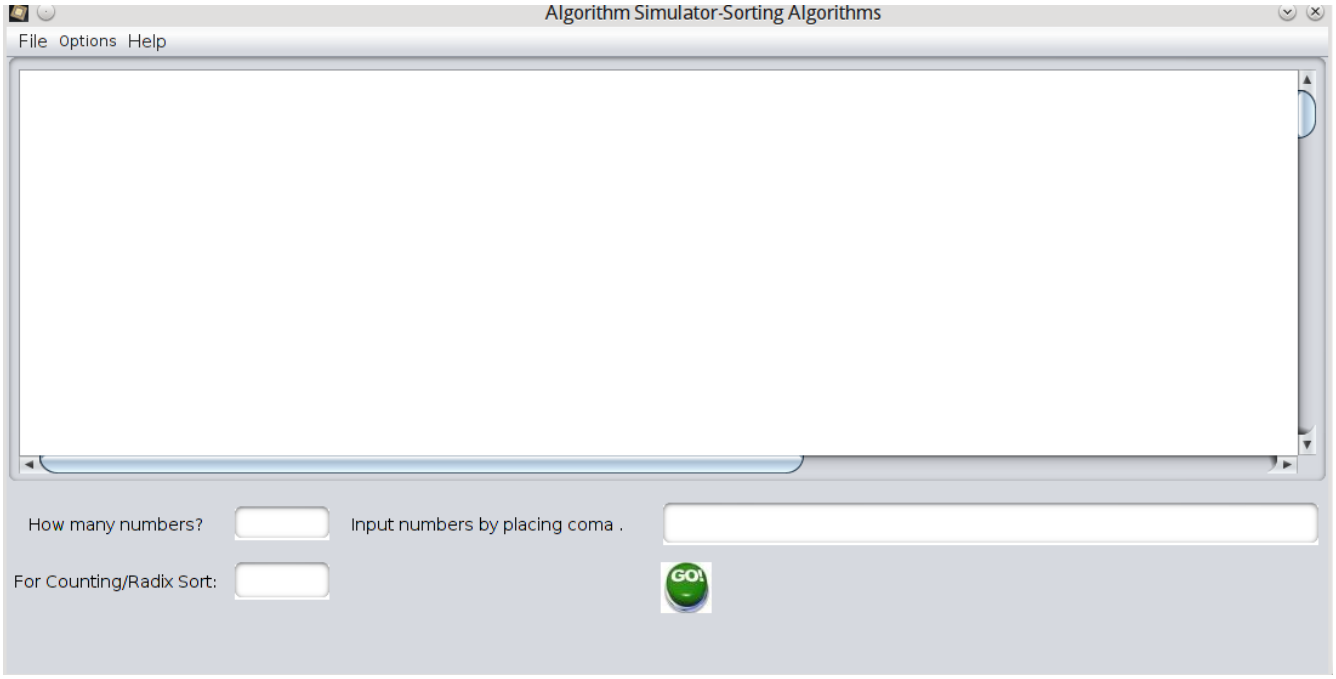


Fig: Design prototype

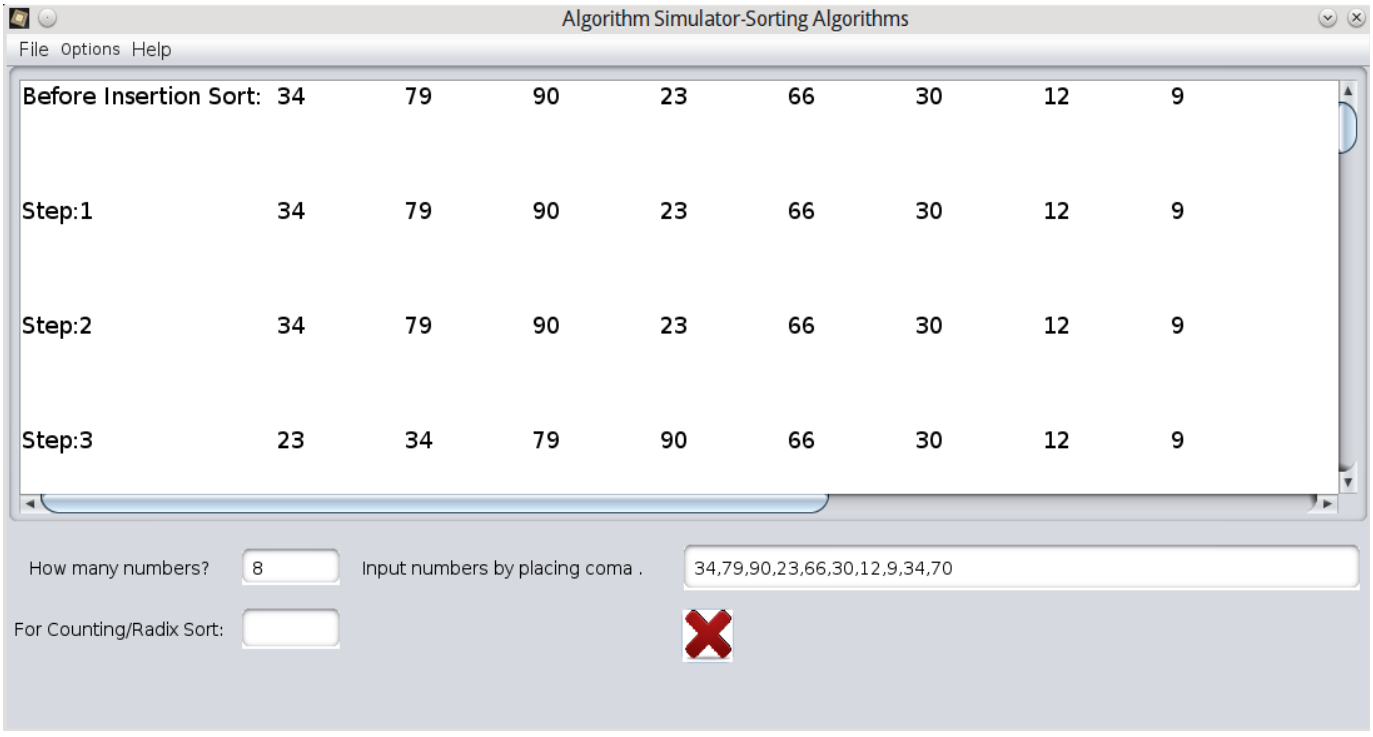
1.5 How to Use It

- System requirements: jdk1.7.0_03 version or greater version .
- Run the simulator by clicking the "SortingAlgorithms.jar" file.(For MS Windows)
- For Linux (type : java -jar SortingAlgorithms.jar in the terminal)

  -Then a window will open like this :



- **The File menu** contains: Clear and Exit sub-menu.
- **The Option menu** contains:
  1.**S**imulate Algorithms where 9 algorithms included and you should select one at a time to simulate .
  -After inputing you enter the button and enjoy simulated output as:



2. View Algorithms that contains all algorithms that you can view by clicking one of this and you will see a html page in a **JeditorPane** that contains the selected algorithm.

3. Similarly,**View Implemented Codes** Contains implemented source code that you can also view in a html page.

- **Help menu** contains about algorithm and designer which you can can view as a html page when you click one of it.

1.7 Tools Used

- Java Development Kit 1.7.0_03(jdk1.7.0_03) for java system environment.
- Netbeans 7.1.1 IDE for efficient program design.
- GanttProject for project scheduling and management.
- Gui-Design Studio for User Interface Designs and Instant Prototypes Without Coding.

1.8 Limitations and future development

- There is only one limitation that is , the simulation is text based not animated .
- This limitation should be developed by the learners.

1.9 Suggestion:

- Implement the sorting algorithm simulator including heap sort(which is not included here) and shows the heap sort as like as tree (it would tricky but not so difficult).
- Implement sorting algorithm simulator that shows you output step by step by clicking a single button once at a time and it will appear as an animated output.

1.10 Conclusion

- After all,this simulator is realy an important way to learn algorithms.
- If the learners follow it then they will able to do new thing.

1.11 References

- **"Introduction To Algorithms" Third Edition by Thomas Cormen , Charles E. Leiserson, Ronald L. Reivest ,Clifford Stein**
- **Java - How To Program, 6th Edition (2004) by Deital and Deital.**
- **https://netbeans.org/kb/docs/java/quickstart.htmla.**

****

2. CPU Scheduling Simulator

2.1 Introduction

- This simulator introduces CPU scheduling,which is the basic for multiprogrammed operating systems.
- By switching the CPU among processes,the operating system can make the computer more productive.
- CPU scheduling is the method by which threads ,process or data flows are given access to system resources.

2.2 Objectives

The objectives of this simulator are;

- to describe various CPU-scheduling algorithms.
- to analyze evaluation criteria for selecting a CPU-scheduling algorithms for a particular system.

2.3 Analysis

Almost all programs have some alternating cycle of CPU number crunching and waiting for I/O of some kind. (Even a simple fetch from memory takes a long time relative to CPU speeds.)

  - In a simple system running a single process, the time spent waiting for I/O is wasted, and those CPU cycles are lost forever.
  - A scheduling system allows one process to use the CPU while another is waiting for I/O, thereby making full use of otherwise lost CPU cycles.
  - The challenge is to make the overall system as "efficient" and "fair" as possible, subject to varying and often dynamic conditions, and where "efficient" and "fair" are somewhat subjective terms, often subject to shifting priority policies.
- Whenever the CPU becomes idle, it is the job of the CPU Scheduler (a.k.a. the short-term scheduler) to select another process from the ready queue to run next.
- The storage structure for the ready queue and the algorithm used to select the next process are not necessarily a FIFO queue. There are several alternatives to choose from, as well as numerous adjustable parameters for each algorithm, which is the basic subject of this Simulator.

2.4 Project Scheduling

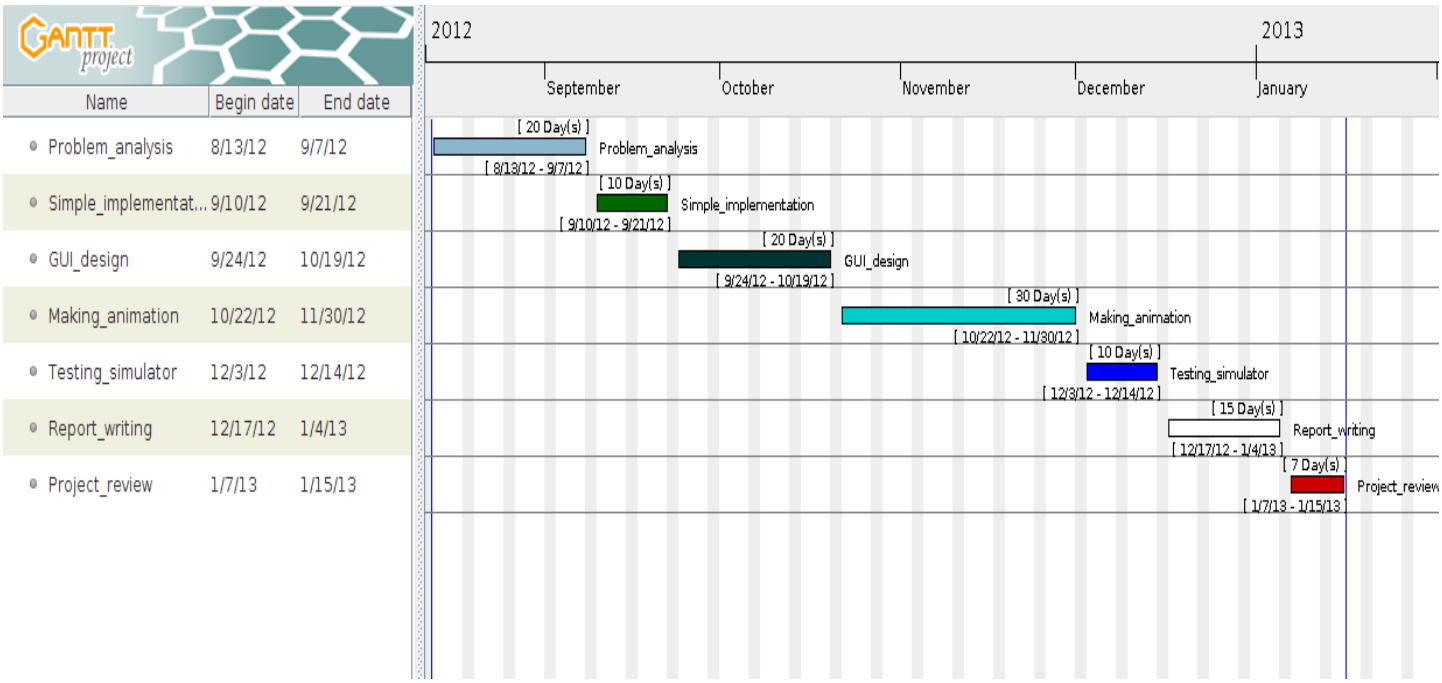
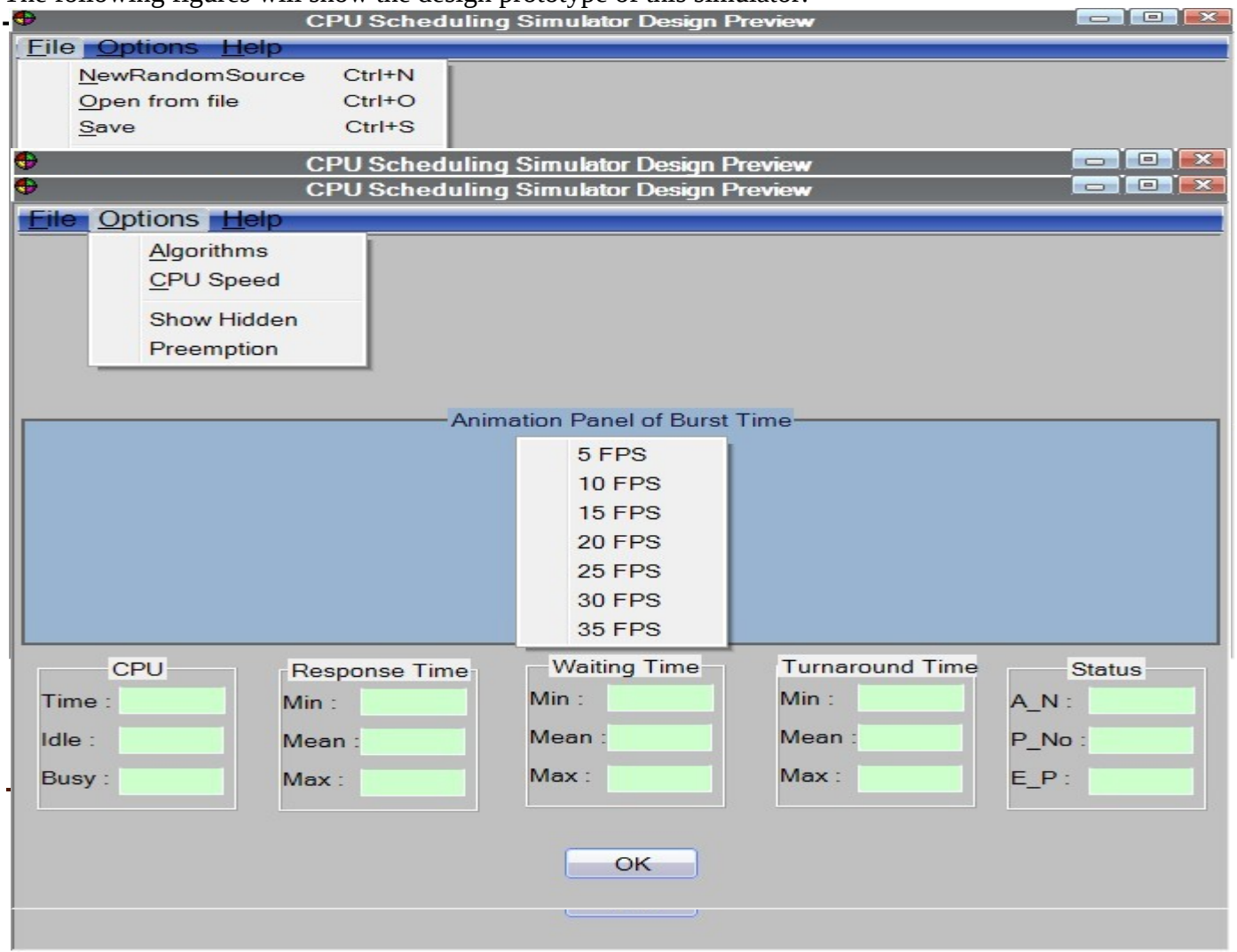
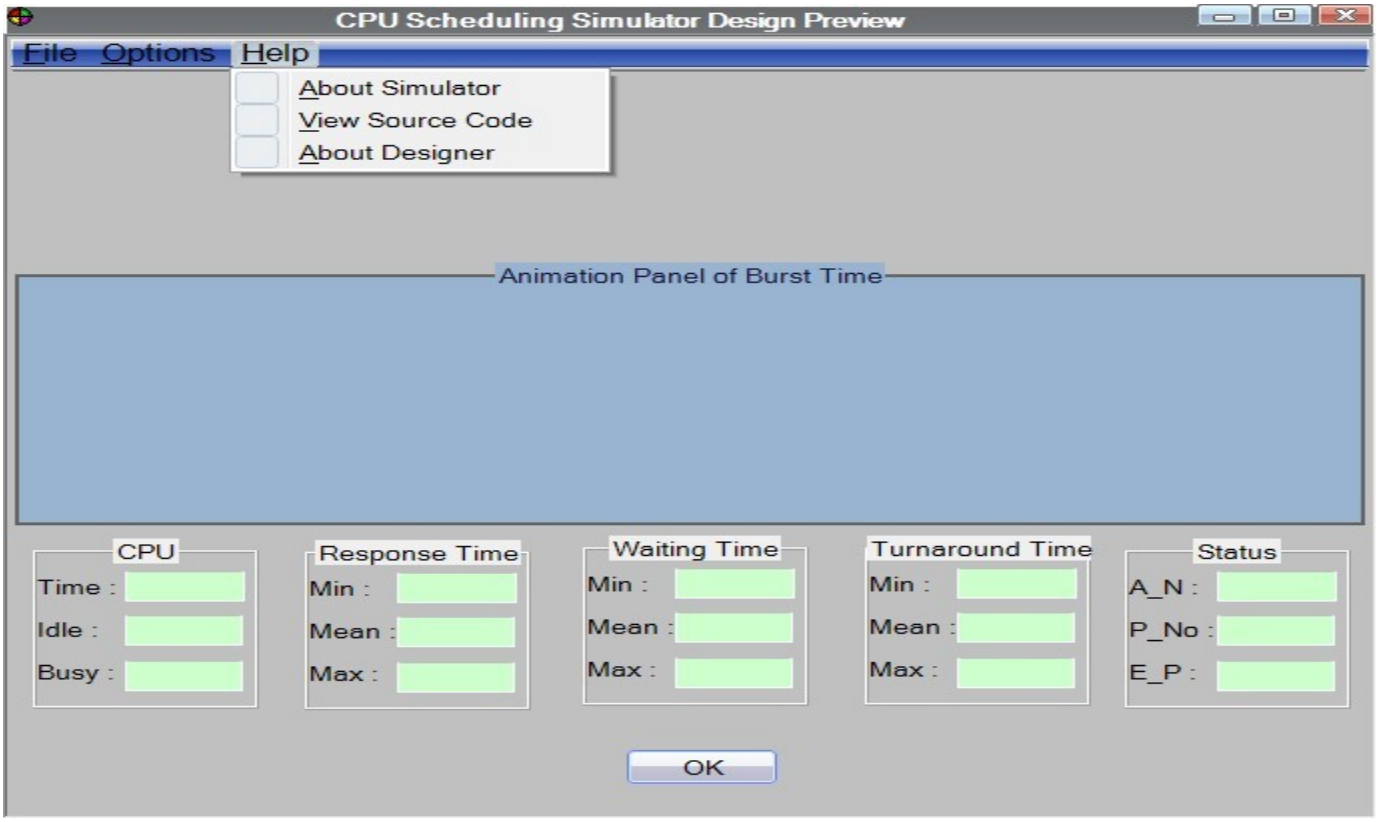


Fig: Gantt Chart

2.5 Design

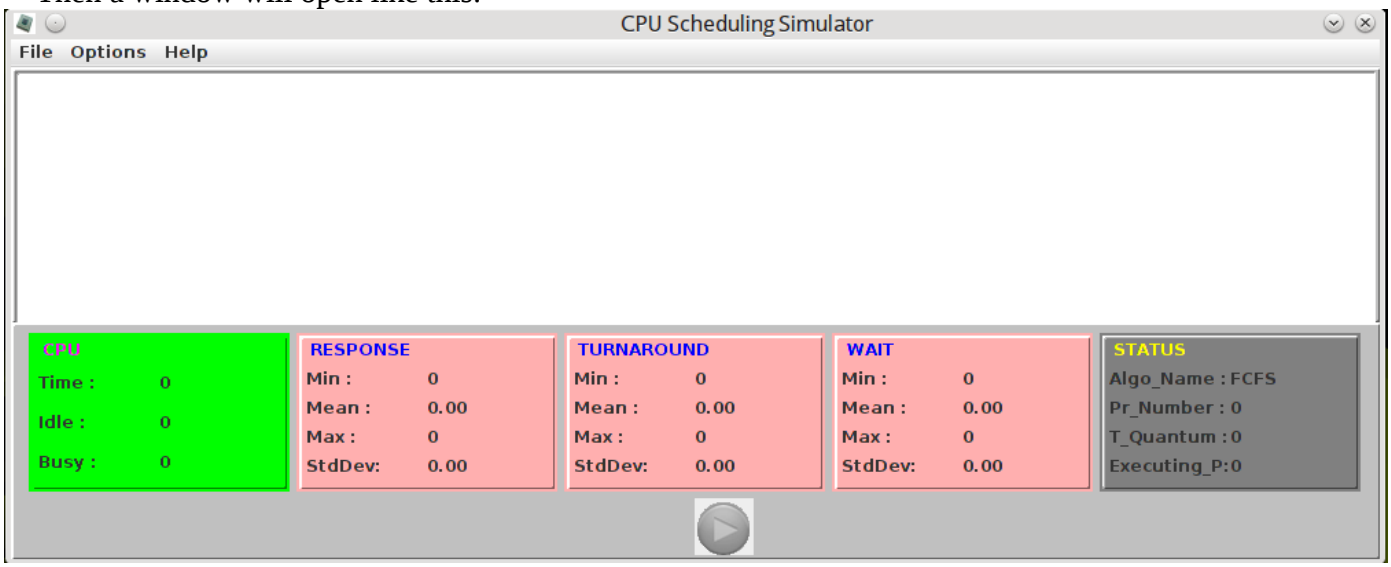
The following figures will show the design prototype of this simulator:





2.6 How To Use It

- System requirements: jdk1.7.0_03 version or greater version .
- Run the Simulator by clicking the ShedulingProject.jar file (for MS Windows)
- For Linux open the terminal and write : java -jar ShedulingProject.jar
  Then a window will open like this:



➢ **The File Menu contains**

---New Random Source for Randomly generated CPU burst time,Time Quantum,priority,and

process number.

---**Open Data Source** for CPU burst time,Time Quantum,priority,and process number from file in any disk drive. The File format like this:

```
3       // number of processes
5       // time quantum for Round Robin
24 0 0  // burst time , arrival time ,proirity for process-0
3  0 1   // burst time , arrival time ,proirity for process-2
3  0 2  // burst time , arrival time ,proirity for process-3
```

By writing this way save the file in .text format .

----**Save Statistics** for saving all calculations such as Response,Turnaround,Waiting time that is like this:

Algorithm Name: FCFS

PID	Burst	Priority		Arrival	Start	Finish	Wait	Response	Turnaround
1	24	0	0	0	23	0	0	24	
2	3	1	0	24	26	24	24	27	
3	3	2	0	27	29	27	27	30	

	MinTime	MeanTime	MaxTime	StdTime
Response :	0	17.0	27	14.798648586948742
Turnaround :	24	27.0	30	3.0
Waiting :	0	17.0	27	14.798648586948742

Save the output in any format such as .txt/.doc/.odt/.csv you like.

The process id (PID) is based on Job Queue not Ready queue. But the process execution is calculated according to Ready queue. So the learners should calculate waiting time ,turnaround time etc. based on Ready queue.

---**Reset Current Source** to reset previous state.

---**Quit** to close the window/simulator .

- **The Options Menu contains :**

---**Algorithms**: First Come First Serve,Shortest First Job,Priority,Round Robin Scheduling Algorithm .So select one to simulate

---**Speed** which indicates the CPU speed in Frame Per Second(FPS) .Select one …....

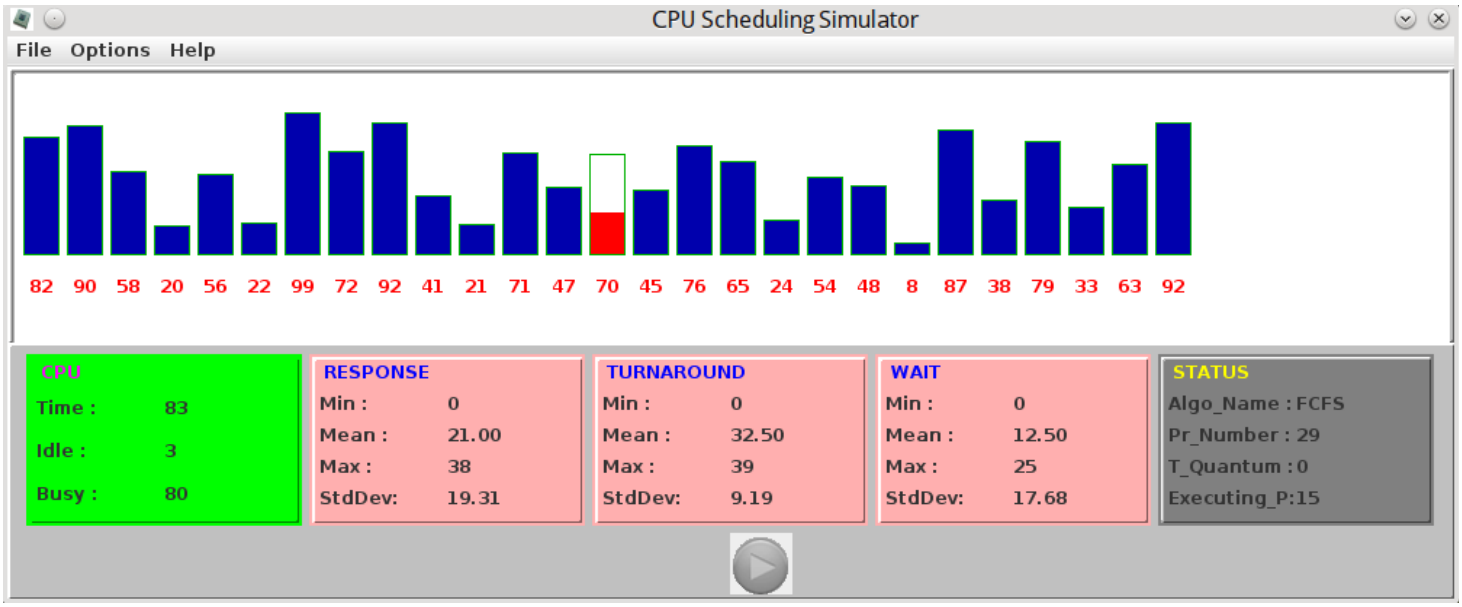
---**Show Hidden** to show the burst time in rectangular bars if it is selected.

---**Preemption** which shows the algorithm is preemptive or not .

- **The Help menu contains:**

---About Simulator(How to use),Source code,About CPUscheduling :by clicking one of them it will appear a Jeditorpane.

- **By maintaining above condition if we run the simulator then we can see the simulator like this:**



-The white color indicates that the process is in Job queue ,the blue color indicates that the process is in Ready queue and the red color indicates executing process.

- The five panel indicates various status information and the icon (below the panels) indicates run button.

After doing these lets have a fun with CPU Scheduling simulator !

2.7 Tools Used

- Java Development Kit 1.7.0_03(jdk1.7.0_03) for java system environment.
- Netbeans 7.1.1 IDE for efficient program design.
- GanttProject for project scheduling and management.
- Gui-Design Studio for User Interface Designs and Instant Prototypes Without Coding.

2.8 Limitations and future development

- It is a nice simulator that the learners do not have to face any limitation.
- The learners can take it as a research project in future as Operating system course.

2.9 Suggestion

1. Implement preemptive Shortest Job First and Priority Scheduling Algorithm. Modify the CPUScheduler,java class to do this
(it is tricky but not so difficult).

2.10 Conclusion

After all, this simulator is a nice one for the CSE students. If they follow the simulator they could realy understand about the CPU Scheduling different algorithms and can make decision which is better for what purposes.

2.11 References

- "Operating System Concept "8th Edition by Abraham Silberchatz ,Peter B.Galvin,Gerg Gagne
- Java - How To Program, 6th Edition (2004) by Deital and Deital.

3. Simulation of Classic Problems Of Synchronization

3.1 Introduction

- In this Simulator , it represents a number of Synchronization problems as example of a large class of concurrency-control problems.
- These problems are used for testing nearly every newly synchronized scheme.
- In my solution to the problem, I use semaphore and monitor for synchronization.

3.2 Objectives

- To understand the synchronization of multiple threads.
- To highlight the technique of Inter-process communication and synchronization problem between multiple operating system processes.

3.3 Analysis

Following are the classic problems of synchronization in operating system:

1. Bounded-Buffer Producer-Consumer Problem:

Suppose there are producer and consumer processes. Producer produces objects, which consumer uses for something. There is one Buffer object used to pass objects from producers to consumers. A Buffer is used to store production of producer.

The problem is to allow producers and consumers to access the Buffer while ensuring the following:

1. The shared Buffer should not be accessed by these processes simultaneously.
2. Consumers do not try to remove objects from Buffer when it is empty.
3. Producers do not try to add objects to the Buffer when it is full.

When condition 3 is dropped (the -Buffer can have infinite capacity), the problem is called the unbounded-buffer problem, or sometimes just the producer-consumer problem.

2. Readers-Writers Problem:

In this problem, a number of concurrent processes require access to some object (such as a file.) Some processes extract information from the object and are called readers. Some processes change or insert information in the object and are called writers. The Bernstein condition states that many readers may access the object concurrently, but if a writer is accessing the object, no other processes (readers or writers) may access the object.

There are two possible policies for doing this:

First Readers-Writers Problem: Readers have priority over writers. It means that unless a writer has permission to access the object, any reader requesting access to object will get it. This may result in a writer waiting indefinitely to access the object.

Second Readers-Writers Problem: Writers have priority over readers. It means that when a writer wishes to access the object, only readers, which have already obtained permission to access the object, are allowed to complete their access. Any readers that request access after the writer has done so must wait until the writer is done. This may result in readers waiting indefinitely to access the object.

3. The Dining Philosophers Problem:
There are five philosophers seated around a dining table, with a plate of food in front of each one. On the table between each pair of philosophers there is a single chopstick.

A philosopher needs to pick up both the left and right chopstick in order to eat. If either is in use by another philosopher, he must wait. The problem is to find a way to synchronize access to the chopsticks to make sure that every philosopher gets a chance to eat.
One solution is to control access to each chopstick with a separate semaphore. When a philosopher wants to pick up a chopslick, it performs P operation on the corresponding semaphore. When chopstick is put back on table, V is performed on the semaphore. Suppose the philosophers are numbered 0-4 and semaphores are also numbered from 0 to 4. For philosopher i, the left chopstick is chopstick i and the right chopstick is chopstick [(i+1)%5].

4. Sleeping Barber Problem

- Consists of a waiting room with s seats and a barber room with one barber chair for c customers and one barber.
- Customers alternate between growing hair and getting a haircut.
- The barber sleeps and cuts hair.
- If there are no customers to be served, the barber sleeps.
- If a customer wants a haircut and all chairs are occupied, then the customer leaves the shop and skips the haircut.
- If chairs are available but the barber is busy, then the customer waits in one of the available chairs until the barber is free.
- If the barber is asleep, the customer wakes up the barber.

5. The Cigarette-Smoker Problem
The cigarette smokers problem is a concurrency problem in computer science, originally described in 1971 by S. S. Patil.
Assume a cigarette requires three ingredients to smoke:
1.Tobacco
2.Smoking Paper
3.A match
Assume there are also three chain smokers around a table, each of whom has an infinite supply of one of the three ingredients — one smoker has an infinite supply of tobacco, another has an infinite supply of paper, and the third has an infinite supply of matches. Assume there is also a non-smoking arbiter. The arbiter enables the smokers to make their cigarettes by arbitrarily (non deterministically) selecting two of the smokers, taking one item out of each of their supplies, and placing the items on the table. He then notifies the third smoker that he has done this. The third smoker removes the two item from the table and uses them (along with his own supply) to make a cigarette, which he smokes for a while. Meanwhile, the arbiter, seeing the table empty, again chooses two smokers at random and places their items on the table. This process continues forever. The smokers do not hoard items from the table; a smoker only begins to roll a new cigarette once he is finished smoking the last one. If the arbiter places tobacco and paper on the table while the match man is smoking, the tobacco and paper will remain untouched on the table until the match man is finished with his cigarette and collects them.

3. 4 Project Scheduling

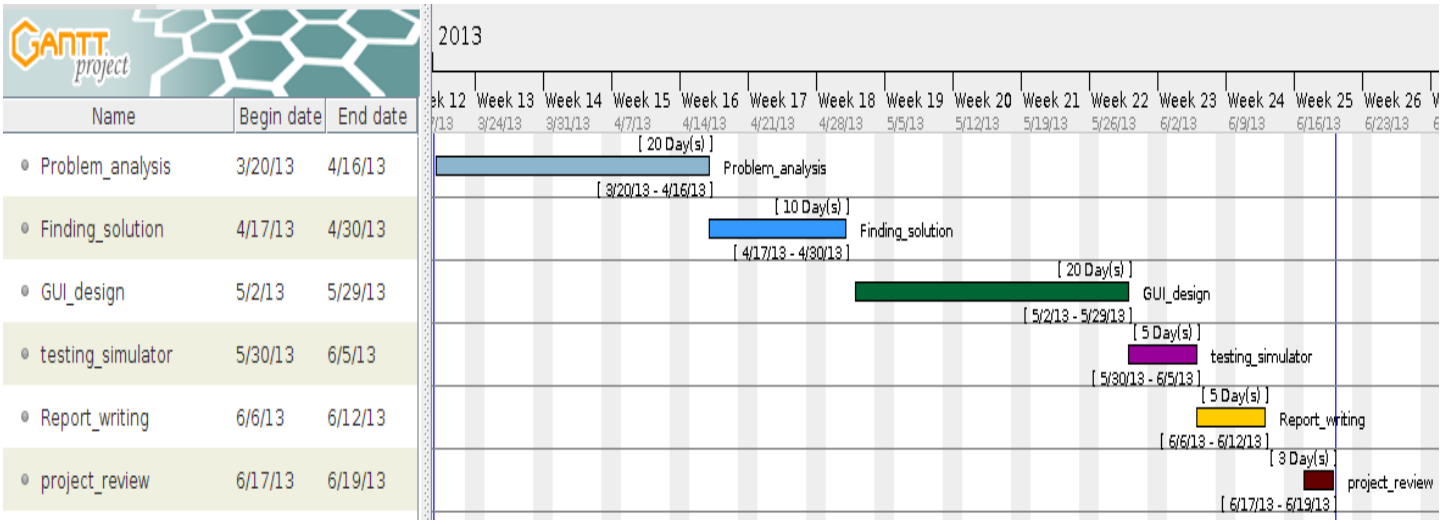
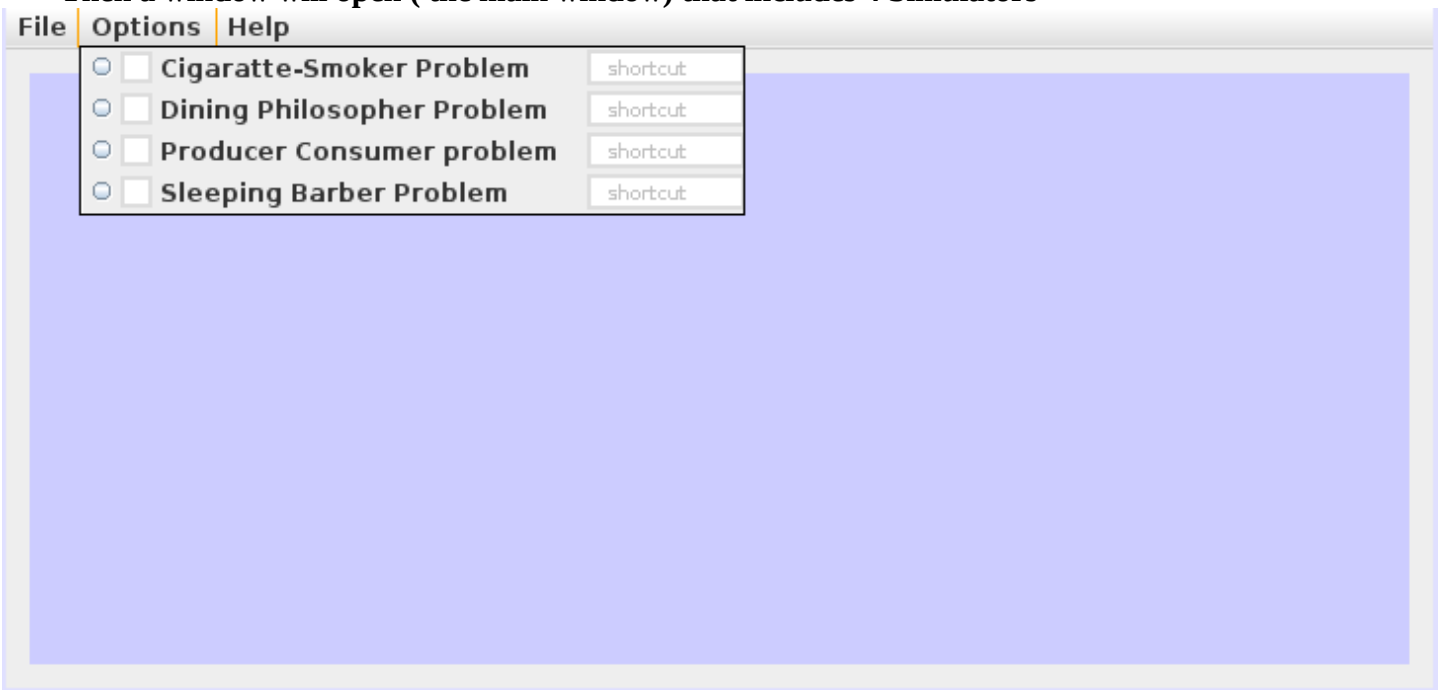


Fig: Gantt chart

3.5 How To Use It

- System requirements: jdk1.7.0_03 version or greater version .
- Run the Simulator by clicking the ClassicSynchronizationProblem.jar file (for MS Windows)
- For Linux open the terminal and write : java -jar ClassicSynchronizationProblem.jar

-Then a window will open (the main window) that includes 4 Simulators



- **The File menu** contains Exit option which indicates to close the simulator.
- **The Options menu** contains 4 Simulator : Dining Philosopher ,Cigarette-Smoker,Producer Consumer,and Sleeping Barber Problem .So, select One of them which you like.

-After selecting one of them you will get another window that is your required simulator. For example I have selected **Cigarette-Smoker Problem** and clicked the play check box and it shows simulation what you read in the book.

-The above procedure is similar to remaining 3 Problems.

- **The Help menu** contains Hot to Use it. If you want to view it then it will show you a text editor pane which contains user manual of this simulator.

3.6 Tools Used

- Java Development Kit 1.7.0_03(jdk1.7.0_03) for java system environment.
- Netbeans 7.1.1 IDE for efficient program design.
- GanttProject for project scheduling and management.

3.7 Limitations and Future Development

- One limitation of Producer Consumer problem is that it is not designed efficient way but in future it will be modified.
- Another limitation is that the Readers-Writers Problem is not included here. So it also be developed in future.

3.8 Suggestion

- Implement Readers-Writers problem together these 4 problems.
- Modify the Producer-Consumer problem **(for both Synchronized and Unsynchronized method)**.

3.9 Conclusion

After All, this simulator is realy helpful for CSE learners to their Operating System course. By using this simulator they properly capture the some important things such as Thread,Process,Synchronization,Interprocess Communication etc.

3.10 References

- "Operating System Concept "8th Edition by Abraham Silberchatz ,Peter B.Galvin,Gerg Gagne
- Java - How To Program, 6th Edition (2004) by Deital and Deital.

4. Ongoing Projects and Other Activities

4.1 Projects

- Controlling Computer Programs By Speech Recognition Using Java.

4.2 Research

- Add Hoc Networking

4.3 Others

- Designing Posters for ITRRC based on the Research Fellows projects.

The End