**CIS 163**

**Project 3 – Voting Booth Simulation Game**

**Group project (2 or 3 members per group)**

**Due Date**

• At the beginning of the lab; see the schedule, last page of the syllabus.

**Before Starting the Project**

• Review Inheritance, Polymorphism, and Interfaces (Chapters 8 and 9 in the textbook)

• Read this entire project description before starting

**Learning Objectives**

After completing this project you should be able to:

• design, implement, and test a small class hierarchy

• use two-dimensional arrays and enum types, and

• implement a GUI-based game

• Working with a group on a project.

**Project Description**

For this project, you will develop an application that simulates customer arrival and departure at a voting center. Statistics are generated for each simulation. This allows management to run various scenarios to determine how many staff / booths to have working to minimize waiting.

**Project Outline**

**Step 1:** A simple version of the Voting Booth Simulation program has been provided to you on BB.

Please download the code and install the code into your IDE.

**Step 2:** Examine the provided code and run the program until you fully understand how it works.

**Step 3:** Using the figure 1.0 below, write a program that simulates that voting booth station.

**Step 4:** Write code to satisfy requirements for a grade of a “C” (i.e., Max Score: 78%)

4a: To receive full credit of 78%, the Java Style code must be completed as well (.5 pts per error).

**Step 5:** Write code to satisfy requirements for a grade of a “B” (i.e., Max Score: 88%)

5a: To receive full credit of 88%, the Java Style code must be completed as well (1 pts per error).

**Step 6:** Write code to satisfy requirements for a grade of a “C” (i.e., Max Score: 100%)

6a: To receive full credit of 100%, the Java Style code must be completed as well (1.5 pts per error).

**Project specifics:**

The different types of voter are:

50% votes have last names that start with A-L

50% votes have last names that start with M-Z

**Heritance:**

Base abstract class: Voter

**protected** **double** votingBoothTime;

**protected** **int** leaveTime;

**protected** **double** checkInTime;

**public** **abstract** **void** setCheckInTime(**double** checkInTime);

**public** **abstract** **void** setLeaveTime(**int** leaveTime);

**public** **abstract** **void** setVotingBoothTime(**double** votingBoothTime);

**Random class:**

Random time (rt) is calculated using the desired mean (dm).

rt = dm\*0.1\*Random.nextGaussian() + dm

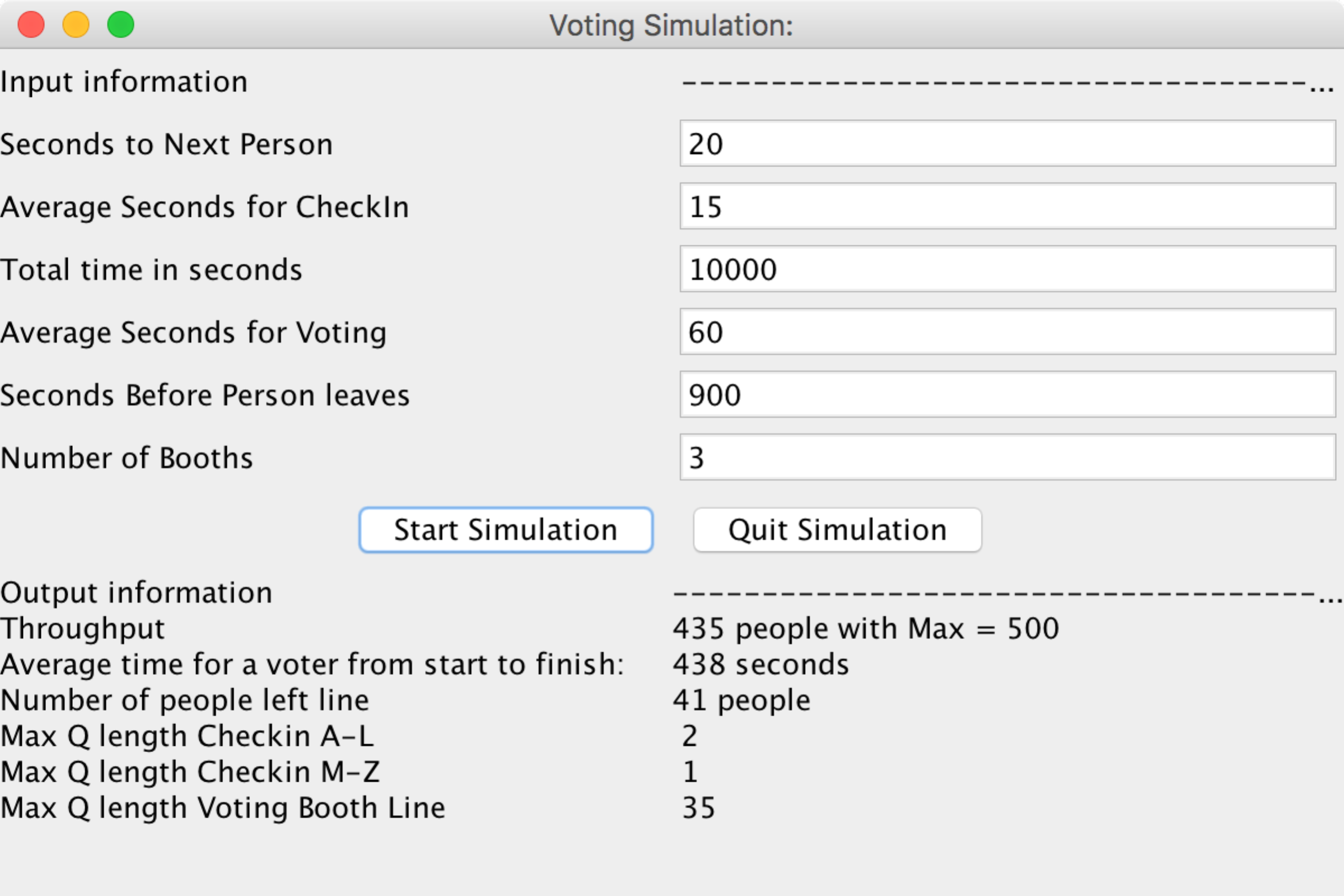
**Notes for project:**

The master clock ticks once per second.

Each time a person is generated, the next customer is scheduled to arrive in the near future based on the average arrival rate.

**Requirements for a grade of a “C” 75%:**

Produce an output that looks something like the following.



**Requirements for a grade of a “B+” 88% (**Do all of the requirements for a grade of a “C” first!)**:**

1. Create a simple GUI for showing your simulation.

- Your GUI should show people in the different lines (see the instructor for GUI requirements.)

2. Draw UML class diagrams that represent a collection of classes (1pt per error)

- Must use the Dia tool in EOS

3. Create a SpecialNeeds, Limited Time voters that extends Voter. Special needs requires more time to be serviced and Limited time leaves the line sooner.

- 10% are special Needs voters

- 20% are limited Time voters

- 70% are regular voters

Extended classes:

**public** **class** SpecialNeedsVoter **extends** Voter

checkInTime \* 1.5;

leaveTime \* 2;

votingBoothTime \* 3;

**public** **class** LimtedTimeVoter **extends** Voter

checkInTime (no change);

leaveTime \* 0.5

votingBoothTime \* 0.5;

**public** **class** RegularVoter **extends** Voter

checkInTime (no change);

leaveTime (no change);

votingBoothTime (no change);

* In your GUI, these SpecialNeeds, LimitedTime voters must be represented somehow.

4. Produce LOTS more statically analysis: For example. Average time at Checkin A-L for special needs

**Requirements for a grade of a “A” 100% (**Do all of the requirements for a grade of a “B” first!)**:**

1. Create a complex GUI for showing your simulation.

- Use Dots to represent people in line.

2. Draw UML class diagrams that represent a collection of classes (1.5pt per error)

- Must use the Dia tool in EOS

3. Create a SuperSpecialNeeds voter that extends SpecialNeedsVoter and requires more time at each station.

- In your GUI, this SuperSpecialNeeds voter must be represented somehow.

4. Produce LOTS more statically analysis the changes as your simulation runs.

5. Have the ability to modify the simulation.

- Remove one (or more) of the voting booths.

- Remove one of the initial tables

**Figure 1.**

1 person voting at any given moment.

Table for people that have last name A - L

People leave

1 person being serviced and rest in Q waiting.

Voting Booth 1

1 Q that services all voting booths, 'x' is the people

xxxxxx x

Voting Booth 2

VotersEnter Here

50% to A-L

50% to M-Z

xxxxxxxxxx

Voting Booth 3

You are done once you leave a voting booth.

xxxxxx x

Voting Booth nth

1 person being serviced and rest in Q waiting.

Table for people that have last name M - Z

Legend: x is a person

Blue boxes are comments

Red boxes are part of simulation

Flow of simulation