Lab 1: Semaphores
CMSC 442
Fall 2017

The dictionary.com definition of "semaphore" is "any of various devices for signaling by changing the position of a light, flag, etc" (http://dictionary.reference.com/browse/semaphore?s=t as of Sept. 2015). Semaphores were used by ships and trains as signals to avoid collisions. In computer science, we use the word "semaphore" to mean something different – a special data structure that prevents "collisions" in concurrent code.

Semaphores in computer science don't use lights, flags, and so forth (they typically usually use an integer variable and a "wait list" instead), but in many ways they are very similar to the semaphores used by trains. In this lab, we are going to use a train simulation game called OpenTTD (an open source close of transport tycoon deluxe) to build some intuition about semaphores and how they work.

Step 0. Getting Started

Make a folder named "Lab1". In that folder, download the following files from the course web site (Under "Class Resources"):

```
Activity1.sav
Activity2.sav
Activity3.sav
```

Launch openttd (either from the command line or the menu) and click on "Load Game". Then select the Activity1.sav file. When the game loads, you should see a map which contains two train tracks. One is a circular train track. The other is a linear track which crosses the circle. You should see an oil train operating on the linear track.

You can zoom in and out using the plus and minus keys on your keyboard. You can use the mouse or the arrow keys to scroll around and look at the various elements of the game. Sometimes you may find it difficult to see the track or the signals behind the trees and other decorations. You can make the trees invisible by holding down the mouse button on the "gear" icon at the top of the screen and selecting "transparency options". Click on the trees to hide them. Click again to show them. You can also make different kinds of buildings, roads, tracks, and signs transparent. If you want to make them invisible and not just transparent, you can click the little rectangle below the icon as well.

The game is currently paused, but when it resumes, the train will pick up oil at the oil well at one end of the track and delivers it to the refinery. Every successful transportation of cargo earns you money that you can use to build more trains, tracks, and so forth.

Near the middle of the track is a small brick building. This is the train depot. When trains break down, they go to the depot for repairs. You can also purchase new trains or change over to a new type of cargo at the depot.

The circular track has four train stations and a depot of its own. One station is near the coal mine. Another is at the sawmill. A third is at the power plant and a fourth is near the forest. There are two trains on the circular track. One picks up wood from the forest and delivers it to the sawmill. The other picks up coal from the coal mine and delivers it to the power plant.

Step 1. Race Conditions

To resume the game, you can click the "pause" button at the top-left of the screen. The oil and wood trains should narrowly miss each other.

If you watch the trains for a minute, you'll notice that when the oil train reaches the station next to the refinery, it stops, unloads, and then a little message with the word “Income” and a dollar amount floats up from the train. This indicates that the train has successfully delivered a cargo.

If you let the game run long enough, however, the wood and coal trains will collide. Obviously, this isn't a good thing. The problem is that the trains are using a shared resource – the segment of track in front of the depot. When they both try to use it, they crash.

This is very similar to something that can happen when two threads, which are running concurrently, try to write to a shared resource (such as a variable or data structure). When one process starts to write, the other process overwrites part of the data structure, which causes the program to misbehave or even crash. We call the part of the program which accesses the shared resource the “critical section”. Since the problem only happens if the two processes access the critical section at the same time, we say that the program has a “race condition”. If one of the programs “beats” the other one to the part of the code that accesses the resource, it will finish in time to avoid the problem, but if it “loses the race” they will both be in the critical section at the same time.

In software, we use a construct called a “semaphore” to prevent these problems. In OpenTTD (and with real trains), we use “signal semaphores” instead.

The simplest kind of signal semaphore is a simple light (or flag) that tells a train if it can proceed. When the signal is red, the train must stop. When it is green, the train can proceed. When two trains need to share a track, the track can be divided into “signal blocks” by placing semaphores along small sections of track. Whenever a train enters the signal block (the space between two semaphores), the semaphores at both ends turn red, preventing any other trains from entering that same section of track. When the train leaves the signal block, the semaphores turn green, which allows other trains through. Signal blocks are very like “critical sections” in a program and the way OpenTTD behaves is exactly like “mutual exclusion” code that protects a critical section in a program.

Reload the Activity1.sav file (by clicking on the “disk icon” and holding the mouse button down, then selecting “Load game” from the menu that comes up). Place semaphores on the track that will prevent the trains from crashing.

To place a semaphore, click on the “train track” icon in the toolbar. Then click on the icon that looks a little like a stoplight. A little window will come up that contains a set of signal lights and flags. Select the tiny one in the bottom left corner.

Note: When you place a signal, it can face either forward, back, or “both” directions. To change orientations, you can click on the signal until you get the orientation you want. Trains cannot pass signals from behind so, for now, make sure that all of your signals are either all point the same way or are set to the “both” orientation.

Try to use as few signals as possible – but be sure the trains cannot collide. If you've done everything correctly, your trains should start to make money and allow you to pay off your “debt” (you start the game with a loan which you can pay off by clicking on the “coins” icon in the toolbar” and clicking “Repay $20,000”).

When you get it working, click on the “disk icon” and save your game as “Solution1.sav”.

Hint: You can read more about signal blocks on the OpenTTD wiki:
https://wiki.openttd.org/Signals#Block_signals

Step 2. Deadlock

Now load the game named “Activity2.sav” (click on the “floppy disk” icon and hold down the mouse button, then go to “Load Game”).

In this scenario, I have created three tracks that cross each other in a sort of “triangle” shape. To prevent the trains from colliding, I have added semaphores that prevent any two trains from being in the triangle at the same time.

Go ahead and unpause the game. At first it may seem like nothing happens, but if you look closely at the bottom of the screen, you'll see that time is passing (the day and month in the status bar are incrementing slowly).

What's happened is that the trains are all stuck. Train 1 is blocking train 2, while train 2 is blocking train 3, which is in the way of train 1.

In programming, this is called a deadlock. Deadlocks occurs when two or more processes hold resources needed by the other processes. For example, one process may need the printer and the disk to finish. Another process may need the disk and the network to finish. A third may need the network and the printer to finish. If the first process has the printer, the second process has the disk, and the third process has the network, none of them can finish and give back their resources. So they get stuck in a deadlock.

In the case of your trains, the “resources” are the tracks that make up the legs of the triangle. Essentially what happens is this: one of the trains gets partway across the intersection, but then cannot enter the critical section on the far side of the intersection. Let's assume that it's the coal train. The reason the coal train cannot enter the critical section on the far side of the intersection is that the wood train is currently in that critical section. But the wood train is also stuck because it needs to get into the critical section on the far side of the sawmill – and it can't, because the coal train is in that critical section (in fact, the coal train is in TWO critical sections. It's a long train!)

You can temporarily solve the problem by making one of the trains give up the track. Click on train 1 (the oil train on the top right-hand side of the triangle). Then click the curved arrow. This tells the train to reverse directions. It will leave the triangle, allowing the other trains to escape.

This works for awhile – but eventually they will all get stuck again. The problem is that we have too MANY semaphores! Instead of keeping the trains out of the entire triangle, they only protect small pieces of track. This allows a train to get partway through the intersection, blocking other trains.

To fix this, you need to move some of the signals. To do this you will have to add new signals and delete the old ones. If you click on the railroad track icon, and then click on the “traffic light” icon, you can enable the “bulldozer tool”. Clicking on the bulldozer and then on some signals allows you to remove those signals.

See if you can figure out how to get the trains moving again **without allowing them to collide**. Don't forget you can always reload to start over from the beginning.

When you get it working, click on the “disk icon” and save your game as “Solution2.sav”.

Step 4. Starvation

Even when a system is not deadlocked, it is possible for a thread to become blocked for long periods of time (or even forever) while other threads take turns monopolizing a resource it needs. This is called “starvation”.

Load “Activity3.sav”. This scenario looks a lot like the first activity, but when you unpause the game, you will only see two trains: the oil train and the coal train. Where is the wood train? It's in the depot. If you click on the little red brick depot, you'll see that the wood train is patiently waiting there. If you click on it, you'll see it's running and wants to get to “Frindinghead Woods”, but it can't.

The reason for this is that the circular track is one critical section – and the coal train and oil train keep “taking it”. This means that the wood train can never get in. This looks a little like a deadlock, but if you wait long enough, the coal train will eventually go to the depot, which frees the wood train. Then it's the coal train's turn to starve!

Starvation often happens when a critical section is too large or when there are many processes running. When either of these happens, slower threads tend not to get the critical section very often and can “starve”.

Fix the problem by adding semaphores to split the critical section up into smaller pieces. When you get it working, click on the “disk icon” and save your game as “Solution3.sav”.

Step 5. Efficiency

The train system you have now should work without collisions or deadlocks – but it's pretty inefficient. If you pay really close attention to the wood train you'll see that it spends a lot of time waiting. See if you can improve the efficiency of the system. You can add (or remove) semaphores, add additional track, or even add additional trains. See if you can find a system that keeps everything moving except when it's actually loading or unloading cargo. Save your result as “Solution4.sav”

Step 6. Reflection

When you are done with OpenTTD, answer the reflection questions on the attached sheet. Be sure your name is at the top. If you'd like to continue fooling around with OpenTTD, feel free. In addition to building train systems, it lets you construct airports, shipping routes, and road networks. To submit, turn in your reflections as a hard copy.

NAME: _____

1. At the beginning of Activity1, the oil train narrowly missed the wood train. There were no semaphores protecting them. Why didn't they crash?

2. Figuring out how many semaphores to use in your code (or your train track) is something of an art. What can go wrong if you use too few semaphores? What can go wrong if you use too many semaphores?

3. Were you able to find ways to make the train system more efficient? What seemed to work well? What didn't?

4. Suppose a multi-threaded program contains a shared data structure (such as an array). If two threads try to write to the data structure simultaneously, a race condition can occur in which one thread overwrites a value added by another thread. Explain how this relates to what you did today in this lab.

5. Did you enjoy this project?

6. Do you feel you learned something from the project?