Work-Term Report
Microsimulation, LifePaths, and Source Control at Statistics Canada

Mohammed A. Chamma

Contents

Abstract

I describe my work term as a researcher in the Modeling Division at Statistics Canada from January to April of 2014. Specifically I give an overview of the work environment at the government, an introduction to the field of microsimulation, a description of a programming language called Modgen used for creating models, and a report of the work I took part in during my term. The work revolved around a microsimulation model called LifePaths, which simulates the lives of Canadian Citizens and allows researchers to explore policy and social questions. During my term the division was contracted by Health Canada to study the costs of future senior care using LifePaths. Most of the division's remaining time was focused on merging together different versions of LifePaths and preparing a clean and publicly releasable version of LifePaths that would serve as the basis of future work. During the merge project, I worked on getting the team to use source control software as a way of improving the team's workflow.

Introduction

My name is Mohammed. I'm a third-year student in Physics-Mathematics. For my first co-op work term I was part of the Modeling Division at Statistics Canada in Tunney's Pasture, Ottawa. This report is a description of my time there, the environment I was a part of, and the projects I was involved in. I will also give background on microsimulation, LifePaths, and the Modgen programming language to provide context for the projects I participated in.

The work environment of Statistics Canada is structured, orderly, and often slow-paced. I was located on the 24th floor of the 26-story R.H. Coats building at Tunney's Pasture. The Modeling Division I was a part of is made up of about ten people, and included PhDs, researchers, and another coop student (also from uOttawa). We each had our own cubicle with high walls and a work computer. None of the regular work computers are connected to the internet –occasionally we had access to confidential data and having these computers connected to the internet would be a security risk. Each floor has a set of shared computers (one or two at each corner of the floor) that are connected to the internet and can be used for browsing. The hours, at least in the Modeling division, were flexible and the focus was more on completing tasks than on completing a set amount of hours.

The Modeling Division is responsible for creating mathematical and computer simulation models of Canadian citizens. We use real statistical data collected by StatCan to make these models representative of real society. I worked on a model called LifePaths, which is categorized as a *microsimulation* model. LifePaths is a computer program that aims to simulate many aspects of Canadian society, including education, fertility, mortality, wages, taxes and many more. When you start LifePaths, you specify the size of the population you want to simulate, and you have the option of changing some parameters, for example, life expectancy. When you start the simulation, LifePaths creates a virtual Canadian and starts simulating this virtual Canadian's life, including basic details like gender and province of residence, and more complicated details like whether or not this person finished highschool, how many children they have, and so on. LifePaths simulates this person's entire life and when that virtual Canadian dies, it simulates the next Canadian's life, until it reaches the population size you chose at the beginning. These virtual Canadian's are statistically similar to real Canadians. When LifePaths decides, for example, what program of study a person enters, it knows what the statistical distribution of programs of study of real Canadians are, and draws from this distribution. Though none of the simulated Canadians in LifePaths exist in reality, the entire population of virtual Canadians are statistically similar to the actual population. When the LifePaths model finishes simulating a population, it surveys the virtual population and outputs tables specified by a researcher. These tables can be compared with real data, and in this way we can validate whether the virtual population matches the real population.

Why would you want to be able to create a virtual population that somehow matches the real population? Having a virtual population lets you make a change and observe the impact on the population, without harming anyone or wasting lots of money. LifePaths is programmed with a set of *parameters.* These parameters form a *scenario*. The parameters encompass a multitude of things, like migration rates into and out of the country, average wages for different jobs, fertility rates, probabilities of becoming disabled, the percentage of taxes one owes, and so on. By changing these parameters we can create new scenarios, run the simulation, and observe the impact on the population. How would populations distributions change if the migration rate were to increase by so-and-so amount? Who would benefit the most if we cut taxes? LifePaths is a tool that lets you pose these questions and attempt to answer them with statistical arguments.

The LifePaths model was programmed in a language called *Modgen*. Modgen was developped a few years ago by Statistics Canada and is used (as the name suggests) to generate socio-economic models. Modgen is a language that compiles to the more standard C++ programming language, used worldwide in all sorts of software. The benefit of Modgen is that it is simpler to write than C++. Modgen introduces many constructs and general ideas that are useful in creating simulation models that C++ doesn't have, making it easier for modelers and statisticians to focus solely on their model, rather than the underlying technical details. Since Modgen code compiles into C++, we can retain the performance and speed that C++ offers. This is important as large simulations can take a very long time.

During my work term, the team was hired by Health Canada for a project with LifePaths. Health Canada wanted to understand the future costs of healthcare for seniors. LifePaths can simulate people's lives in the future based on population projections and so create a projection of what the costs would be. Health Canada also asked the Modeling Division to investigate different scenarios: what would it be like if the government paid for 100% of the costs? What would it be like if the government paid for 80% of the costs and seniors the rest? LifePaths also allowed us to investigate how many people would even be able to afford their care if they were required to pay.

The Health Canada project involved programming new modules into the simulation to simulate the different types of homecare a person could receive, and assigning homecare to people with a set probability. To get these probabilities (so that they reflected reality) we analysed data in the National Population Health Survey (NPHS), which asks respondents, amongst other things, if they received homecare and what kind of homecare they received.

While most of the team was kept busy with the Health Canada project, there was also a project to renew and create a new release version of LifePaths. LifePaths was in a state where there were hundreds of tables that few understood, modules that were never used, and multiple different versions from people adding their own features and then never merging them all together again. I was given the task of merging the different versions together and to create a clean, releasable version of LifePaths. This clean version of LifePaths would also serve as a starting point for future LifePaths projects.

Microsimulation and LifePaths

LifePaths is referred to as a *microsimulation* model. The *micro* in *microsimulation* refers to the fact that the simulation models the behaviour of single individuals, as opposed to modelling aggregate (or macro) statistics. For example, one can collect data on a population's size every year, and then construct a model (or function) that fits the data. This function however does not give you any information about the individual units of the population, for example the birth and death year of each person in the population. In this example the macro data is the size of the population and the micro data is the birth and death year of each person. Microsimulation (LifePaths specifically) tries to recreate the macrodata (population size) by simulating the microdata (birth and death years) of individuals. When a simulation is complete LifePaths aggregates the data (in effect by conducting a survey) of the simulated individuals and outputs tables of statistics that can be compared with real statistics.

A consequence of microsimulation is that it is possible to examine in details the lives of each simulated individual. Included with Modgen is a tool called the BioBrowser. BioBrowser graphically displays how a person's attributes change in time. For example, we can use BioBrowser to see a person's disability state (no disability, light disability, severe disability) throughout their life. This lets us verify that the lives of the people we're simulating make sense in reality.

Microsimulation is a relatively new field. It was first proposed by an Englishman named Guy Orcutt in 1957 in a paper titled "A new type of socio-economic system".[1] Over time more and more statistical agencies have started their own microsimulation models to address a whole host of issues. Aside from LifePaths, Statistics Canada has microsimulation models for the tax and transfers system, health care analysis, and population projections.

Tasks Completed

While I was at StatCan I worked on the following things:

- I and another co-op student created a prototype version of a simulation module that simulated homecare.
- I merged two versions of LifePaths and produced a clean and releasable version of LifePaths that will serve as the starting point for future projects.

[1]Orcutt G (1957) 'A new type of socio-economic system', *Review of Economics and Statistics*, 39(2), 116-132.

- I introduced and used source control software to carry out the merge and track changes. This let me build a history of changes that served as documentation.
- I presented to the division about the source control software I used (called 'git') to the team. I explained about how it worked, how to use it, and how I was using it for LifePaths.
- I validated that my merged version of LifePaths still behaved as expected by comparing it's output with previous versions of LifePaths and with real statistical data. This involved a suite of graphs that plotted various statistics. These statistics included population by province, unemployment rate since 1971, earnings by province, births by year, and many more.

The Modgen Programming Language

The LifePaths model is written in the Modgen language. Modgen allows you to use language constructs that are tailored for writing socio-economic models. It also abstracts away some lower-level details of how the simulation runs and creates an executable file with it's own visual interface for running the simulation model. This allows statisticians to develop the model without the need of a programmer and lets the researcher focus more on the details of the model and less on the implementation of an interface. In short, Modgen is a language for specifying *actors* and simulating their lives by specifying the *events* that can happen to them. In LifePaths, the actor is a Person and the events are life events. Persons can get married, suffer an accident, go to university, have children, and so on.

Actors

Let's say we were working on a Modgen microsimulation model and wanted to create a Person actor. We want to simulate the number of children this person has throughout their life. We can declare the actor in a file for Modgen like this:

```
actor Person {
    int numOfChildren = {0};
};
```

Modgen automatically tracks each actor's age (**int** age) and the current year of the simulation. The variables numOfChildren, age, and year are called actor *states.* We'll use the variable numOfChildren to track the number of children each simulated Person has.

Events

What we need now is a way for a Person "to have a child", and then to increase the numOfChildren variable when that happens. Modgen provides an event construct where we can define what the event of "having a child" means. Modgen events require two functions. The first function is called the time function. The time function determines *when* the event happens. The second function is the event function and it gets run when the event happens.

Time in LifePaths (and most Modgen simulations) is considered continuous. This is possible because instead of using a clock that ticks regularly to schedule events, very precise *waiting times* for events are generated and the simulation just jumps to the event that has the smallest waiting time. Thus, the time function of an event must generate a waiting time for the event. These time functions can depend on all the other states of the Person. For example, if we have a "death" event, and your age is young, we might calculate a very large waiting time, as younger people are less likely to die. If your age is very old, we would calculate a much shorter waiting time. This means that every time the age changes, we want to recalculate the waiting time of the event, to capture this new information.

The general flow of the simulation of an actor goes like this:

1. Call all the time functions of all the events. This gives you a list of waiting times.
2. Call the event function for the event with the smallest waiting time. This is the event that happens "next".

3. Recalculate all the waiting times of all the events.
4. Repeat.

This loop repeats until one of the events ends the simulation for the actor (like a death event). Modgen takes care of all the details of the loop and all the researcher has to do is define what the event functions are. If a researcher wants some kind of timekeeping to, for example, know what month an event happens, then the researcher creates a “month” event, whose waiting time is always 1/12 of a year. The year state is updated automatically by Modgen.

To create a “have a child” event, you declare the event in the actor block of your file:

```
actor Person {
    int numOfChildren = {0};
    event timeChildEvent, ChildEvent;
};
```

In this case, the time function is called `timeChildEvent` and the event function is called `ChildEvent`. The time function will come up with a waiting time based on the likelihood of having a child and the event function will simply increment `numOfChildren` by one every time the Person has a child.

As an example, let’s construct an oversimplified probability model of the likelihood of having a child based on age. Let’s say a person has zero chance of having a child at their age if their age is under 16, a 60% chance if they are between 16 and 30, a 40% chance if they are between 30 and 60, and no chance after that. If we had a function $P(x)$ that represented the *probability of having a child during age x* with x as an *integer*, this could be written as

$$P(x) = \begin{cases} 0 & x \le 16 \\ \frac{0.6}{30-16} & 16 < x \le 30 \\ \frac{0.4}{60-30} & 30 < x \le 60 \\ 0 & otherwise \end{cases} = \begin{cases} 0 & x \le 16 \\ \frac{0.6}{14} & 16 < x \le 30 \\ \frac{0.4}{30} & 30 < x \le 60 \\ 0 & otherwise \end{cases}$$

We divide by the width of the age interval so that $P(x)$ is normalized. We would implement this model in Modgen with this function:

```
TIME Person::timeChildEvent()
{
    TIME waitingTime; //declare variable
    if(age <= 16){
        waitingTime = TIME_INFINITE;
    } else if(16 < age && age <= 30) {
        if(RandUniform() < 0.6/(30-16)) {
            waitingTime = 0; //0 means this is the next event
        }
    } else if(30 < age && age <= 60) {
        if(RandUniform() < 0.4/(60-30)){
            waitingTime = 0; //0 means this is the next event
        }
    } else {
        waitingTime = TIME_INFINITE;
    }
    return waitingTime;
}
```

The function `RandUniform()` returns a *uniformly* distributed random number between 0 and 1. This means that boolean expressions in the form `RandUniform() < p` have a probability `p` of being true. Now all that's left is to implement the event function. The event function simply increments the state of `numOfChildren`.

```
void Person::ChildEvent()
{
    numOfChildren = numOfChildren + 1;
}
```

The two event functions are all that's needed to implement this simple model. Modgen takes care of the details including scheduling the event and keeping track of the age and time of each actor. From here you compile the model and Modgen creates an executable file that has a visual interface to the model. In the interface the user can specify the size of the population to simulate. To create some useful output, users can create tables with Modgen's tabling language using the variable `numOfChildren`. These tables act as surveys and Modgen fills in the table as it simulates the actors. At the end, a spreadsheet table is displayed with output that can be copied to other tools like Microsoft Excel or Stata so that graphs can be made. In this way we can validate that the simulation is giving output that agrees with the model we implemented, or that the simulated data matches with some real data.

Parameters

Another feature of Modgen is the ability to specify *parameters* that can be changed from scenario to scenario by a user through the visual interface. Let's say we wanted to investigate a scenario in the above example where 16 year olds had a very high chance of having a child. One way to do this would be to simply change the code of the model. Changing the code however isn't very flexible when a researcher wants to investigate a wide variety of scenarios. What could be done instead is to specify a parameter in the code that represents the probability of having a child during age x. The time function would access this parameter when calculating waiting times, and the parameter can be changed through the visual interface by the researcher. The visual interface displays a spreadsheet like table (in this case it would be a table of probabilities by age) and values can be entered or pasted in from some external program. These modifications to the parameter don't require you to recompile the code, or even to have access to the code, and can be saved as a seperate scenario file with it's own output. This way, a researcher can create a set of "what-if" scenarios when investigating an issue.

To declare a parameter in Modgen, you specify the type of the parameter (a real number), its name, and its dimensions (in this case, age). Since we're using age and age is treated as a continuous variable, we also create a *partition* to divide up a person's age into discrete groups of our choosing:

```
partition AGE_GROUPS {
    5 , 10, 15, 20, 25, 30, 35, 40, 45, 50,
    55, 60, 65, 70, 75, 80, 85, 90, 95
};

parameters {
    double ChildBirthProbability[AGE_GROUPS];
};
```

This tells Modgen to create a parameter table in the visual interface for the parameter `ChildBirthProbability` and the number of entries to have based on the number of divisions in the `AGE_GROUPS` partition. Default values for the parameter are specified by another file that Modgen makes a copy of whenever a new scenario is created.

Having a parameter simplifies the implementation of our time function. We can replace what we had before with the following function:

```
TIME Person::timeChildEvent()
{
    TIME waitingTime; //declare variable

    //take the age and figure out which age group it falls into
    int ageGroup = SPLIT(age, AGE_GROUPS);

    if(RandUniform() < ChildBirthProbability[ageGroup])
        waitingTime = 0; // make this the next event
    else
        waitingTime = TIME_INFINITE;
    return waitingTime;
}
```

With that we've created a basic model that allows a researcher to enter values for the `ChildBirthProbability` parameter, perhaps based on some real data or perhaps in investigating a "what-if" scenario (eg. "what if everyone over 30 stopped having children?"). This model can be compiled and the executable can be distributed online to anyone interested in these kinds of problems.

Visual Interface

Modgen automatically creates an interface for running, modifying, and saving simulations.

The workflow for using a Modgen model is usually something like this:

1. Specify or Modify the scenario's parameters.
2. Choose a population size.
3. Run the simulation (this generates an output database).
4. View or Export the output tables.

Figure 1 shows an example of the interface that Modgen generates for any model:

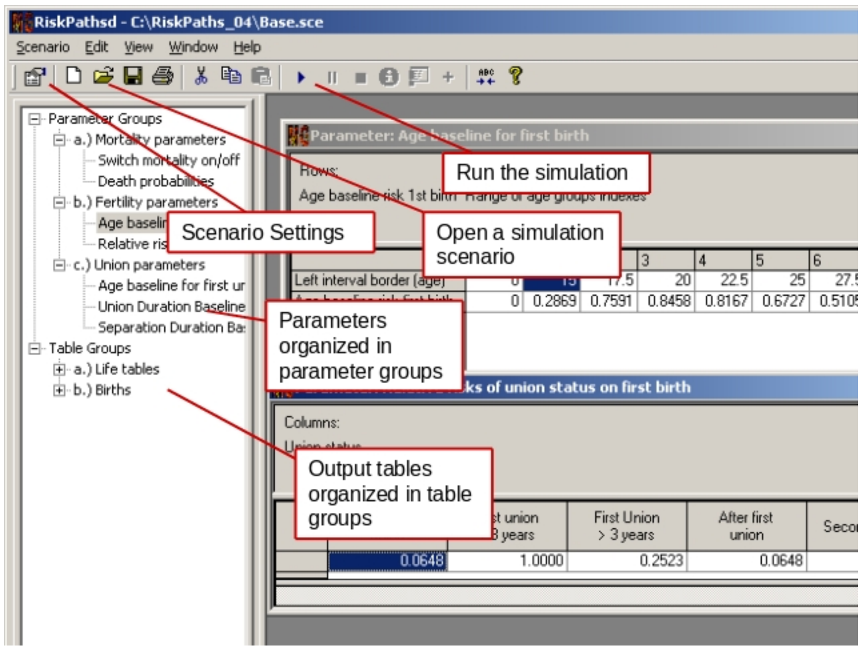

RiskPathsd - C:\RiskPaths_04\Base.sce
Scenario Edit View Window Help
Run the simulation
Scenario Settings
Open a simulation scenario
Parameters organized in parameter groups
Output tables organized in table groups
Parameter Groups
a.) Mortality parameters
Switch mortality on/off
Death probabilities
b.) Fertility parameters
c.) Union parameters
Age baseline for first union
Union Duration Baseline
Separation Duration Baseline
Table Groups
a.) Life tables
b.) Births
Parameter: Age baseline for first birth

Figure 1: Modgen's Interface

Projects

Before starting on any projects, the beginning of the work term was spent learning about Modgen and microsimulation and practicing with Modgen to create some simple models. I then used Modgen to participate in two projects. The first project was a contract job that came from Health Canada to study the costs of healthcare for seniors in the future. This required adding a new module for LifePaths written in Modgen. The second project was to merge the codebase of different versions of LifePaths and to produce a 'clean' and releasable version of LifePaths. The merge project involved reorganizing hundreds of lines of Modgen code.

Homecare Project

Around the beginning of the work term, the Modeling Division was approached by Health Canada. They wanted to use LifePaths to get an idea of what the costs of senior care would be like in the future. There was a few weeks of meetings as the project was fleshed out and around February it became clear that at least one aspect of the project was to add a module to LifePaths that simulated people receiving homecare.

The first concern was how to define homecare, and where the data would come from. The plan involved multiple aspects. First we had to find some real data on the receipt of homecare to understand the amount of detail we could realistically simulate. Is there data on different *kinds* of homecare? Could we find out what the proportion of people receiving homecare was in the different provinces? How about the proportion by different ages? The next aspect of the project was to take this data and construct a mathematical model. With a model, we could write some Modgen code and create a module that realistically simulated homecare.

I participated in finding the data on homecare and in prototyping the early stages of the homecare module. Other members of the team created the mathematical model and implemented it in Modgen.

For our data source we used a survey conducted by StatCan called the National Population Health Survey (NPHS). This survey asks respondents a multitude of questions about their health and lifestyle. This included questions about health issues and the type of care they received. The National Population Health Survey started in 1994 and tries to survey the same respondents every two years (this is called a longitudinal survey). In addition to the longitudinal component, the NPHS has large data files of respondents who were only surveyed once (this is called a cross-sectional survey). The respondents of the NPHS were from all across Canada and the cross-sectional component had a much larger sample size than the longitudinal component so we used the cross-sectional data of 1996 and 1998 to get the information we needed about homecare.

Of the tens of thousands of people surveyed there were only about 2500 who had received some type of homecare. The concept of homecare in the NPHS was broken into seven different categories, which the team simplified to just three: Nursing homecare, personal homecare, and “other” homecare. This simplification was made because of the very low numbers there were for some of the seven categories. Nursing homecare is when a nurse visits the home to provide medical care. Personal homecare is care related to day-to-day tasks (meal preparation, hygiene).

From this data the simplest model that we could create was one that assigned a probability based on your province of residence, your age, your gender, and, since LifePaths already modeled disability for people, your disability level. To make this model you just count the number of people receiving each type of homecare grouping them by age, gender, province, and disability level and then divide by the total number of people receiving homecare. To know if a person receives homecare at all, the proportion of people who receive homecare out of the entire population (or sample) is calculated and used as the likelihood of receiving homecare.

To prototype the simulation module, I and another coop student implemented the above model and created some output tables to show how it worked. We presented the prototype and this got the rest of the team discussing various aspects of the problem, and they worked on creating and implementing a more complete model.

After that point I spent most of my time on the merge project. The division presented to Health Canada their analysis of different scenarios around the end of April.

Merge Project

Most of my time during the work-term was spent working on the merge project. While I was doing this, I started investigating how the team's workflow could be improved. The goal of the merge project was to produce a 'clean' and publicly releasable version of LifePaths that others could use or modify. The other goal of the project was to have a version of LifePaths that would serve as a good starting point for future projects. Inefficiencies in the workflow and code organization motivated this project.

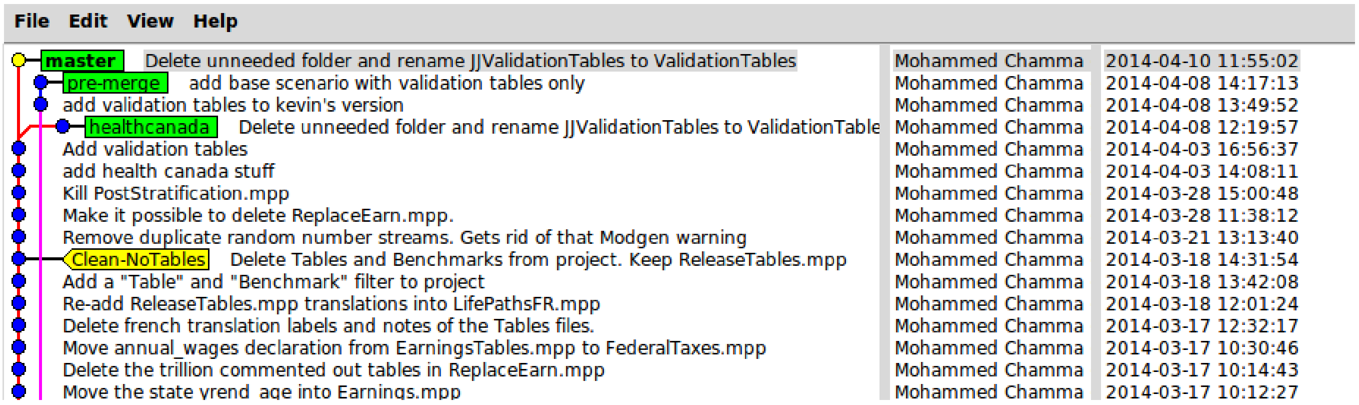
Team Workflow and Source Control

The team uses a shared folder to work on things together. In the case of LifePaths, all the source code sat in a folder and if someone wanted to work on it they would make a copy of the folder and rename it (adding their initials or some other identifier). The result of this was that the shared folder was filled with multiple versions of LifePaths and no clear indication of what was different between them (especially when people just added their initials to things). Over time this led to a problem where no one really knew which version of LifePaths should be used, and which versions had the desired features.

It was also a problem when people wanted to collaborate together. It was hard to coordinate who made what changes and when two sets of changes from different people were put together, things often stopped working without a clear idea of which change was responsible.

To address some of these workflow problems I suggested using a tool that enables source control. Source control tracks history and changes made to a project, line-by-line. It would allow the ability to easily see who made what changes, and the time that those changes were made. Source control also lets you manage different "branches" of code. The copied folders with initials would be replaced with branches. These branches can be compared with one another to explicitly see what their differences are, and can be merged together using the source control tool.

While doing the merge I worked on convincing the team that source control was a solution to a lot of the problems and I presented about two pieces of software that do source control: the first was called Git and was an open-source tool I was familiar with from before, and the other was called Team Foundation, which is the standard at StatCan. I presented to the team about the two tools and how they would be used internally for LifePaths. While I was told progress on these kinds of things (getting new software and changing workflows) was slow at StatCan, I was allowed to install Git on my machine and a few of my coworkers' machines so that we could use it for our own work, and I ended up using Git to create the merged version of LifePaths.



master Delete unneeded folder and rename JJValidationTables to ValidationTables	Mohammed Chamma	2014-04-10 11:55:02
pre-merge add base scenario with validation tables only	Mohammed Chamma	2014-04-08 14:17:13
add validation tables to kevin's version	Mohammed Chamma	2014-04-08 13:49:52
healthcanada Delete unneeded folder and rename JJValidationTables to ValidationTable	Mohammed Chamma	2014-04-08 12:19:57
Add validation tables	Mohammed Chamma	2014-04-03 16:56:37
add health canada stuff	Mohammed Chamma	2014-04-03 14:08:11
Kill PostStratification.mpp	Mohammed Chamma	2014-03-28 15:00:48
Make it possible to delete ReplaceEarn.mpp.	Mohammed Chamma	2014-03-28 11:38:12
Remove duplicate random number streams. Gets rid of that Modgen warning	Mohammed Chamma	2014-03-21 13:13:40
Clean-NoTables Delete Tables and Benchmarks from project. Keep ReleaseTables.mpp	Mohammed Chamma	2014-03-18 14:31:54
Add a "Table" and "Benchmark" filter to project	Mohammed Chamma	2014-03-18 13:42:08
Re-add ReleaseTables.mpp translations into LifePathsFR.mpp	Mohammed Chamma	2014-03-18 12:01:24
Delete french translation labels and notes of the Tables files.	Mohammed Chamma	2014-03-17 12:32:17
Move annual_wages declaration from EarningsTables.mpp to FederalTaxes.mpp	Mohammed Chamma	2014-03-17 10:30:46
Delete the trillion commented out tables in ReplaceEarn.mpp	Mohammed Chamma	2014-03-17 10:14:43
Move the state yrend_age into Earnings.mpp	Mohammed Chamma	2014-03-17 10:12:27

Figure 2: The Git history of the merged version. Each little bubble represents a set of changes. The most recent change is at the top.

My work-term ended before any decision was made on which source control software to use and before I left I made a few guides on how to use Git and how it could be used specifically for LifePaths.

The Merge

There were two principal versions of LifePaths that needed to be merged. The first was a few years old and added modules that simulated the payment and transfer of *provincial* taxes in Canada. This capability, while conceivably useful, had just sat around in it's own folder seperated from the rest of LifePaths for years, and was never used. The other version was just a few months old and was considered to be the most recent 'release' version. It had up to date data on population projections and mortality, and had a very large module written by one of the researchers in the division for a conference paper on earnings.

Aside from there being two versions to merge, the previous release version was filled with code for hundreds and hundreds of output tables that few really understood. One of our goals was to pick just a few tables that would serve as example of the capability of the Modgen tabling language and to delete the rest, releasing only these hand-picked tables. This ended up reducing the codebase by *thousands* of lines.

At first it seemed like one of the biggest challenges of merging the two versions of LifePaths –both with thousands of lines of code– was how to find what the differences were between the two. Part of the Git source control tool was that it can help you merge two "branches" of code together by automatically merging the parts of a file that are the same in both versions and telling you explicitly where both versions of a file differ. This lets you open the file where the 'conflict' was found, see both versions of the different lines of code, and pick the version you want to keep. Sometimes you have to rewrite that section of the file to have both versions and still have it compile.

Since the two versions of LifePaths dealt with different concepts (one with taxes and the other with earnings) most of the differences were easily resolvable and it was just a matter of picking the right version to keep. Since the taxes version was a lot older than the earnings version, there were also a lot of conflicts arising from things that were out of date in the taxes version. These are simply resolved by picking the version that was newer.

There were lots of little differences everywhere and most of the task was manual labor. For fun, I used Git to ouput a file with a list of all the changes (line-by-line) made to merge the two together. This file ended up being 600 pages long and made it seem like I did a lot more work than I did!

I also ended up merging the simulation modules created for the homecare project into this ’clean’ version.

By the end of the work term I had produced a version of LifePaths that was a lot slimmer with a lot of unneeded things removed. It also had all the major work of past years combined into a single model, and will hopefully be a good starting point for future work. Perhaps it will one day be released for public use as well.

Conclusion

Overall my work term at StatCan was educational. I learned about the rules surrounding the government workplace and about how StatCan is structured. I enjoyed working with my coworkers and I felt like my contribution to their work was useful to them. I learned about the field of microsimulation and about how the LifePaths microsimulation model works. I also learned about the Modgen programming language and used it extensively in my work while at StatCan. I participated in the division’s work with Health Canada and helped renew LifePaths by creating a new, clean version of if that can be released to the public.

I learned about workflows and how they can be improved with the right tools. I also learned about how much resistance and lethargy there can be to implementing new things (like git) in large organizations.

The work I participated in at StatCan taught me about simulation in general, which is used for many applications in Physics. I also got to practice programming, and debugging when things didn't work. My experiences at StatCan are experiences that I can draw on when working on more physics-related programming tasks.