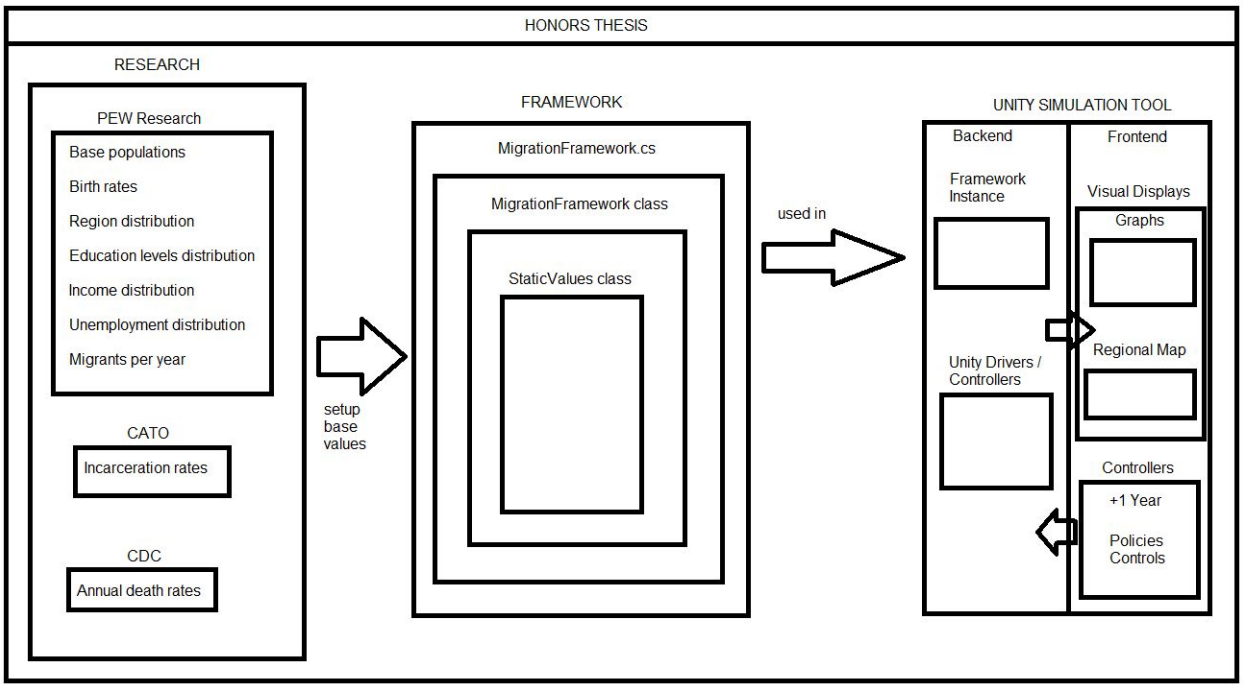
# *Simulating US Immigration for an Educational Game*

Joseph Larsen and Braydon Spangler

| | |
|---|---|
| Thesis Director: | Yoshihiro Kobayashi |
| Secondary Director: | Brian Nelson |

# Table of Contents

# Abstract




The Migration Framework and Simulator is a combination of C# framework / library and Unity simulation tool used for studying basic migration patterns across the US. Users interact with the Unity simulation tool by implementing political policies or adjusting values via sliders, buttons, etc., which will alter the values in the framework. The user can then use the simulation interface to view different estimated population values for categories of people, such as regional differences, education levels, and more.

# Initial Planning

Our initial plan was to build the framework based off of the simulation described in *Impact of immigration on the Japanese economy: A multi-country simulation model*. The paper described a multi-country model that took into account economic factors and economic policy changes as well as population growth and migratory push-pull factors. Our plan was to implement the framework described in the Unity3D game engine and create a front end interface to vary some of the values in the simulation.

However we were unable to implement this as the mathematics were beyond our grasp and the paper was missing some details regarding implementation of the simulation. Because of these difficulties, we adjusted our plans. We took the basic structure of the simulation described in the initial paper and used much simpler mathematics that were derived from our own research.

While these calculations are nowhere near as accurate as those described in the initial paper, they provide a proof of concept framework that can be modified to provide more accurate simulation in the future.

# Research

One of the most important things regarding this framework is getting correct statistics to be used for the simulation. The statistics used for this simulation have come from 3 primary sources: PEW Research, CATO Institute and CDC. Most of our statistics come from PEW Research. This includes the rates for: educational attainment, region of residence, birth rates and income levels. We got the incarceration rates from CATO Institute research and the death rates from the CDC. All of our sources had these statistics available for both native born and foreign born populations with the exception of the death rate which was for all residents of the US regardless of place of birth. Most of these statistics are from the year 2017 with the exception of the incarceration rate which is based on data from 2014. For all of these statistics we wanted to use the most recent data available from reliable sources.

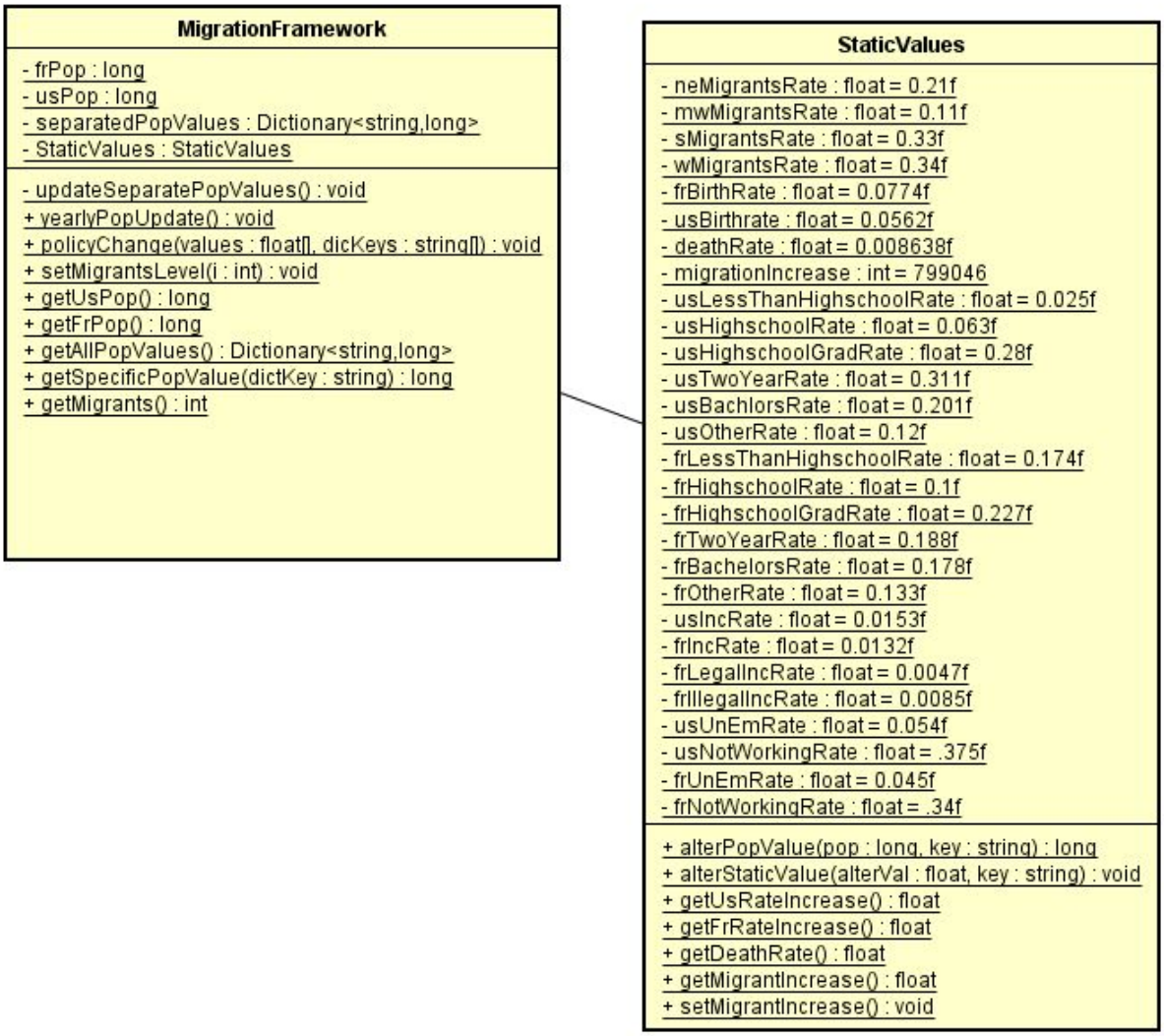
## Specific Citations for Statistics

All the data we used is publically available for examination on the websites of the organizations that reported and analysed it. Below are links to our specific citations:

- Starting Population
    - Values In Framework: frPop, usPop
    - Found in PEW Research article “Facts on U.S. Immigrants, 2017” in the Nativity of US Immigrants spreadsheet
        - https://www.pewresearch.org/hispanic/2019/06/03/facts-on-u-s-immigrants-current-data/
- Birth Rate
    - Values were given in births per thousand women. We calculated this into a rate by dividing the given births per thousand by 1000 and multiplying the number by .5 as we estimated that 50% of the population are women
    - Values In Framework: usBirthRate, frBirthRate
    - Found in PEW Research article “Hispanic Women No Longer Account for the Majority of Immigrant Births in the U.S.”
        - https://www.pewresearch.org/fact-tank/2019/08/08/hispanic-women-no-longer-account-for-the-majority-of-immigrant-births-in-the-u-s/
- Proportion of Migrants per Region
    - Values In Framework: neMigrants, mwMigrants, sMigrants, wMigrants, neMigrantsRate, mwMigrantsRate, sMigrantsRate, wMigrantsRate

    - Found in PEW Research article “Facts on U.S. Immigrants, 2017” in the Region and top states of residence of U.S. immigrants spreadsheet
        - https://www.pewresearch.org/hispanic/2019/06/03/facts-on-u-s-immigrants-current-data/
- Education Levels
    - Values In Framework: usLessThanHighschool, usHighschool, usHighschoolGrad, usTwoYear, usBachelors, usOther, frLessThanHighschool, frHighschool, frHighschoolGrad, frTwoYear, frBachelors, frOther, usOtherRate, usLessThanHighschoolRate, usBachelorsRate, usTwoYearRate, usHighschoolGradRate, usHighschoolRate, frOtherRate, frBachelorsRate, frTwoYearRate, frHighschoolGradRate, frHighschoolRate, frLessThanHighschoolRate
    - Found in PEW Research article “Facts on U.S. Immigrants, 2017” in the Education of U.S immigrants spreadsheet
        - https://www.pewresearch.org/hispanic/2019/06/03/facts-on-u-s-immigrants-current-data/
- Unemployment
    - Values In Framework: usUnEm, usNotWorking, frUnEm, frNotWorking, usUnEmRate, usNotWorkingRate, frUnEmRate, frNotWorkingRate
    - Found in PEW Research article “Facts on U.S. Immigrants, 2017” in the Work status and occupations of U.S. immigrants spreadsheet
        - https://www.pewresearch.org/hispanic/2019/06/03/facts-on-u-s-immigrants-current-data/
- Incarceration Rate
    - Values In Framework: usInc, frInc, frLegalInc, frIllegalInc, usIncRate, frIllegalIncRate, frLegalIncRate, frIncRate
    - Found in the CATO Institute article “Criminal Immigrants: Their Numbers, Demographics, and Countries of Origin”
        - https://www.cato.org/publications/immigration-reform-bulletin/criminal-immigrants-their-numbers-demographics-countries
            - Specific Rates from Figure 1
- New Migrants per year
    - Values In Framework: migrationIncrease
    - Found in the article “Origins and Destinations of the World’s Migrants, 1990-2017” using their data for 2017
        - https://www.pewresearch.org/global/interactives/global-migrant-stocks-map/
- Death Rate
    - This was given in deaths per thousand people and was calculated into a single rate float by dividing the given number of deaths per thousand by 1000.
    - Values In Framework: deathRate
    - Found in the CDC article “FastStats - Deaths and Mortality.”
        - https://www.cdc.gov/nchs/fastats/deaths.htm

# Framework




The simulation framework is built off of the research data. It consists of a main class that holds values for the US-born population and the foreign-born population, a dictionary of other sub-populations, and functions for altering populations, and a subclass that holds the percentages of subpopulations and the functions used by the main class for altering subpopulations. Both classes are static; the subclass is static since it is only used for referencing specific numbers by the main class, and the main class is static so all references to it throughout the simulation pull values from the same class instance.

# MigrationFramework Main Class

## Variables

```
private static long frPop = 44406371;
private static long usPop = 281312807;
```

Base Populations: These values are based on research and are used as the default starting population values.

```
private static Dictionary<string, long> separatedPopValues = new Dictionary<string, long>
{
    { "neMigrants", 9325338 },
    { "mwMigrants", 4884701 },
    { "sMigrants", 14654103 },
    { "wMigrants", 15098167 },
    { "usLessThanHighschool", 7032821 },
    { "usHighschool", 17722707 },
    { "usHighschoolGrad", 78767586 },
    { "usTwoYear", 87488283 },
    { "usBachelors", 56543875 },
    { "usOther", 33787537 },
    { "frLessThanHighschool", 7726709 },
    { "frHighschool", 4440638 },
    { "frHighschoolGrad", 10802647 },
    { "frTwoYear", 8348398 },
    { "frBachelors", 7904335 },
    { "frOther", 5906048 },
    { "usInc", 4304086 },
    { "frInc", 293083 },
    { "frLegalInc", 208710 },
    { "frIllegalInc", 377545 },
    { "usUnEm", 15190892 },
    { "usNotWorking", 105492303 },
    { "frUnEm", 1998287 },
    { "frNotWorking", 15098167 }
};
```

Sub Populations: This dictionary is used for storing all the subpopulations displayed on the UI. These values are based on multiplying the base populations by the percentages stored in the StaticValues class.

## Functions

```
private static void updateSeparatePopValues()
{
    Dictionary<string, long> newValues = new Dictionary<string, long>();
    foreach(KeyValuePair<string, long> kvp in separatedPopValues)
    {
        if (kvp.Key.Substring(0, 2) == "us")
            newValues.Add(kvp.Key, StaticValues.alterPopValue(usPop, kvp.Key));
        else
            newValues.Add(kvp.Key, StaticValues.alterPopValue(frPop, kvp.Key));
    }
    separatedPopValues = newValues;
}
```

updateSeparatePopValues(): This function is called internally every time a change to the base populations are made, and it updates all subpopulations in the dictionary.

Dictionary values are hard to change individually once set, so the most efficient way to update all the dictionary values is to create a new dictionary and then add in each KeyValuePair from the original dictionary with the original key but a different value.

```
public static void yearlyPopUpdate()
{
    Debug.Log("US Pop Begin" + usPop);
    long usPopGrowth = (long)(((usPop / 2) * StaticValues.getUsRateIncrease()) + ((frPop / 2) * StaticValues.getFrRateIncrease()));
    Debug.Log("usPopGrowth " + usPopGrowth);
    long frPopGrowth = (long) StaticValues.getMigrantIncrease();
    Debug.Log("frPopGrowth " + frPopGrowth);
    long usPopDec = (long)(usPop * StaticValues.getDeathRate());
    Debug.Log("usPopDec " + usPopDec);
    long frPopDec =  (long)(frPop * StaticValues.getDeathRate());
    Debug.Log("frPopDec " + frPopDec);
    usPop = usPop + usPopGrowth - usPopDec;
    Debug.Log("US Pop "+ usPop);
    frPop = frPop + frPopGrowth - frPopDec;
    Debug.Log("FR Pop " + frPop);
    updateSeparatePopValues();
}
```

yearlyPopUpdate(): This function is used for incrementing the year.  It uses annual birth and death rates to add and subtract from the native population, and annual migrant increase and death rates to add and subtract from the foreign population.

This function caused us some trouble because the native population was growing at a higher rate than expected, so Unity debug statements were used to find where the problem was occurring within the function.  This allowed us to easily locate where the bug was and remove it.

```
public static void policyChange(float[] valueChanges, string[] dicKeys)
{
    for (int i = 0; i < valueChanges.Length; i++)
        StaticValues.alterStaticValue(valueChanges[i], dicKeys[i]);
    updateSeparatePopValues();
}
```

policyChange(): This function is used for tying the policies implemented through the Unity simulation tool to the StaticValues variables that are changed. A list of the values' increases / decreases and the keys of the percentages to change are passed in, which are then passed back to the StaticValues class. Once the percentages are altered, the subpopulations can be changed.

```
public static void setMigrantsLevels(int i)
{
    StaticValues.setMigrantIncrease(i);
}
```

setMigrantsLevels(): There is only one bread-and-butter setter function in this class, and it is for setting migrant increases. It simply passes the value back to the StaticValues class where the migrants increase is stored.

```
// Getters
public static long getUsPop() { return usPop; }
public static long getFrPop() { return frPop; }
public static Dictionary<string, long> getAllPopValues() { return separatedPopValues; }
public static long getSpecificPopValue(string dictKey)
{
    if (separatedPopValues.ContainsKey(dictKey))
        return separatedPopValues[dictKey];
    else
        return -1;
}
public static int getMigrants() { return (int)StaticValues.getMigrantIncrease(); }
```

A list of get functions for the class. Get functions are used to access private variables from outside the class and are essential for the Unity tool to access values within the class.

# StaticValues Subclass

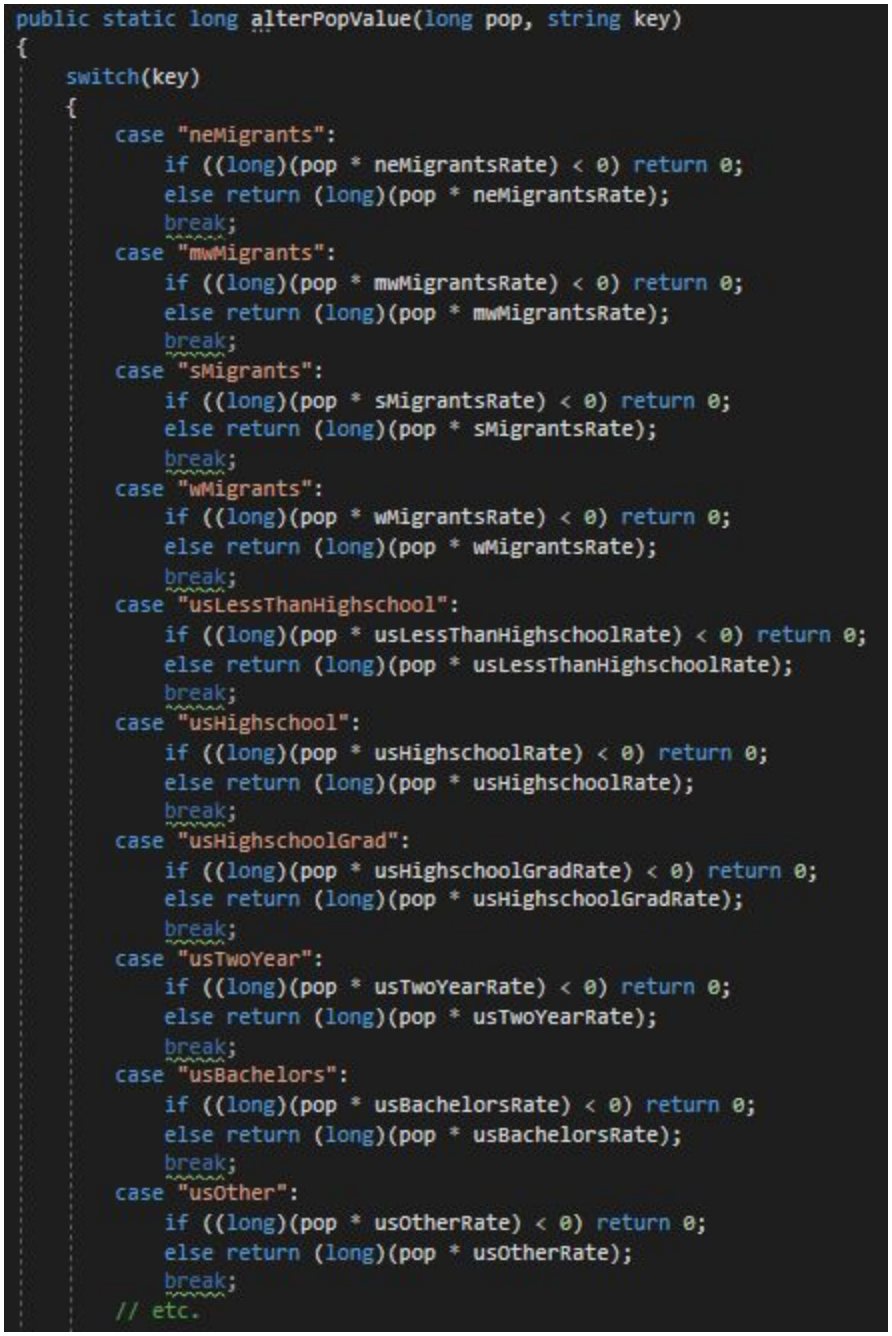
## Variables

```
// Proportions of Migrants per region
private static float neMigrantsRate = 0.21f; //0
private static float mwMigrantsRate = 0.11f; //1
private static float sMigrantsRate = 0.33f; //2
private static float wMigrantsRate = 0.34f; //3

// Birth rates per year
private static float frBirthRate = 0.0774f;
private static float usBirthRate = 0.0562f;

// Death rates per year
private static float deathRate = 0.008638f;

// Migration rate per year
private static int migrationIncrease = 799046;

// Education levels
private static float usLessThanHighschoolRate = 0.025f; //4
private static float usHighschoolRate = 0.063f; //5
private static float usHighschoolGradRate = 0.28f; //6
private static float usTwoYearRate = 0.311f; //7
private static float usBachelorsRate = 0.201f; //8
private static float usOtherRate = 0.12f; //9

private static float frLessThanHighschoolRate = 0.174f; //10
private static float frHighschoolRate = 0.1f; //11
private static float frHighschoolGradRate = 0.227f; //12
private static float frTwoYearRate = 0.188f; //13
private static float frBachelorsRate = 0.178f; //14
private static float frOtherRate = 0.133f; //15

// Incarceration Rate
private static float usIncRate = 0.0153f; //16
private static float frIncRate = 0.0132f; //17
private static float frLegalIncRate = 0.0047f; //18
private static float frIllegalIncRate = 0.0085f; //19

// Unemployment Rate
private static float usUnEmRate = 0.054f; //20
private static float usNotWorkingRate = .375f; //21

private static float frUnEmRate = 0.045f; //22
private static float frNotWorkingRate = .34f; //23
```

All variables from the StaticValues class are percentages of a whole represented by float values. Variables are divided by comments to depict their groups; for example, every value under the “Education levels” comment is used in calculating the subpopulations for education levels until the next comment.

There are three types of variables: native percentages, foreign percentages, and population increases / decreases. The first two are used for calculating subpopulations; percentages of native subpopulations start with “us___”, while percentages of foreign populations start with “fr___”. Population increases / decreases are used for changing the base populations whenever the year increments.

## Functions



```
public static long alterPopValue(long pop, string key)
{
    switch(key)
    {
        case "neMigrants":
            if ((long)(pop * neMigrantsRate) < 0) return 0;
            else return (long)(pop * neMigrantsRate);
            break;
        case "mwMigrants":
            if ((long)(pop * mwMigrantsRate) < 0) return 0;
            else return (long)(pop * mwMigrantsRate);
            break;
        case "sMigrants":
            if ((long)(pop * sMigrantsRate) < 0) return 0;
            else return (long)(pop * sMigrantsRate);
            break;
        case "wMigrants":
            if ((long)(pop * wMigrantsRate) < 0) return 0;
            else return (long)(pop * wMigrantsRate);
            break;
        case "usLessThanHighschool":
            if ((long)(pop * usLessThanHighschoolRate) < 0) return 0;
            else return (long)(pop * usLessThanHighschoolRate);
            break;
        case "usHighschool":
            if ((long)(pop * usHighschoolRate) < 0) return 0;
            else return (long)(pop * usHighschoolRate);
            break;
        case "usHighschoolGrad":
            if ((long)(pop * usHighschoolGradRate) < 0) return 0;
            else return (long)(pop * usHighschoolGradRate);
            break;
        case "usTwoYear":
            if ((long)(pop * usTwoYearRate) < 0) return 0;
            else return (long)(pop * usTwoYearRate);
            break;
        case "usBachelors":
            if ((long)(pop * usBachelorsRate) < 0) return 0;
            else return (long)(pop * usBachelorsRate);
            break;
        case "usOther":
            if ((long)(pop * usOtherRate) < 0) return 0;
            else return (long)(pop * usOtherRate);
            break;
        // etc.
```

alterPopValue(): This function is used by the MigrationFramework class to update individual subpopulations. The key and value of a subpopulation are passed in from the main class, and a

long switch statement identifies which percentage to use. The calculations for the new subpopulation are then done and the new value is sent back as the return value.

There are some policies that, with enough consecutive implementations, result in a negative percentage and therefore a negative subpopulation. To combat this, a non-negative check is performed before returning the value to make sure the subpopulation doesn't dip below 0.

```
public static void alterStaticValue(float alterVal, string key)
{
    switch (key)
    {
        case "neMigrants":
            neMigrantsRate += alterVal;
            break;
        case "mwMigrants":
            mwMigrantsRate += alterVal;
            break;
        case "sMigrants":
            sMigrantsRate += alterVal;
            break;
        case "wMigrants":
            wMigrantsRate += alterVal;
            break;
        case "usLessThanHighschool":
            usLessThanHighschoolRate += alterVal;
            break;
        case "usHighschool":
            usHighschoolRate += alterVal;
            break;
        case "usHighschoolGrad":
            usHighschoolGradRate += alterVal;
            break;
        case "usTwoYear":
            usTwoYearRate += alterVal;
            break;
        case "usBachelors":
            usBachelorsRate += alterVal;
            break;
        case "usOther":
            usOtherRate += alterVal;
            break;
        case "frLessThanHighschool":
            frLessThanHighschoolRate += alterVal;
            break;
        case "frHighschool":
            frHighschoolRate += alterVal;
            break;
        case "frHighschoolGrad":
            frHighschoolGradRate += alterVal;
            break;
        case "frTwoYear":
            frTwoYearRate += alterVal;
            break;
        // etc.
```

alterStaticValue(): This function is used in conjunction with policy changes to alter specific subpopulation percentages.  Similar to altering population values, it received the flat change in percentage and the key for the value to change and does the math to change the value.

```
// Getters
public static float getUsRateIncrease() { return usBirthRate; }
public static float getFrRateIncrease() { return frBirthRate; }
public static float getDeathRate() { return deathRate; }
public static float getMigrantIncrease() { return migrationIncrease; }

// Setters
public static void setMigrantIncrease(int x) { migrationIncrease = x; }
```

The list of get and set functions.  Each get function is used in incrementing the year, and the set function is used for setting the flat migrant increase per year.

# Unity Simulation Tool

The Unity simulation tool is the interface that the user would use to interact with the framework. The front end consists of a two-dimensional map of the United States for visualizing migration patterns across the US, a display for seeing the population and subpopulation numbers in the current situation, and the policy implementation system.  The back end consists of the migration framework and a driver tool for accessing values.

## Front End

The front end of the simulation is done within Unity UI. It consists of a map of the US, several information panels with associated buttons that activate those panels, a policy panel with associated buttons and a button to increment the year forward by 1. All of these are linked to the backend framework that processes, calculates and stores all the values. The numerical information for each of the categories is also displayed in bar graph format to allow for easy comparison of magnitude. This was done using a Unity chart and graph package available on the Unity Asset Store. The package can be found here. The US Map was a Unity Store bought asset. It can be found here.

### Regional Map

The regional map is used for visualizing the migration across different regions and major cities within the US.  The map is divided into different colors to divide the map into its four main regions: the northeast, the midwest, the south, and the west.

In addition to the regions, the top 20 US cities are shown as black dots on the map. These cities include: New York City, Los Angeles, Chicago, Houston, Philadelphia, Phoenix, San Diego, Dallas, San Antonio, Detroit, San Jose, Indianapolis, San Francisco, Jacksonville, Columbus, Austin, Memphis, Baltimore, Milwaukee, and Boston. The size of the city varies based on the starting population of each city; for example, the city with the highest population, New York City, starting at a population of roughly 8 million, starts at twice the size of the second highest population, Los Angeles.

Throughout the simulation runtime, 'migrants' in the form of agents will sometimes spawn on the map. These migrants spawn in a city and 'move' to another city. When a migrant spawns in a city, they are effectively leaving their home in that city, and the city will shrink in size to indicate a migrant leaving. When the migrant arrives at their new city, that city will grow in size to indicate a migrant arriving.

## Graphs

The regional map is only shown when viewing regional migrant data. The other three data panels - those being educational populations, incarceration populations, and unemployment populations - display bar graphs to visualize their respective data. The graphs will dynamically react to elements being changed in the policies panel.

## Policies Panel

The policies panel is used for the majority of adapting the simulation. It contains a few different user controls: a slider for increasing the flat migrant increase per year and a series of policy 'levels' that will increase or decrease the proportion of populations based on the level of the policy.

The migrant slider is fairly self-explanatory. The simulation will always start with a base value for migrants per year, and changing the value of the slider will increase or decrease the number of migrants per year.

The policy levels all tie to a specific set of population proportions; the education policies tie to education populations, the unemployment policies tie to unemployment populations, and the incarceration policies tie to incarceration populations. Each policy level starts at level 0 and can be increased or decreased by hitting the 'Level Up' or 'Level Down' buttons.

All of the current value adjustments were picked arbitrarily to serve as a proof-of-concept for the system. They are not based on real world values and the only criteria when they were being chosen was that all the rates must still add up to 1.0f in order to not mistakenly boost the total population.

## Miscellaneous

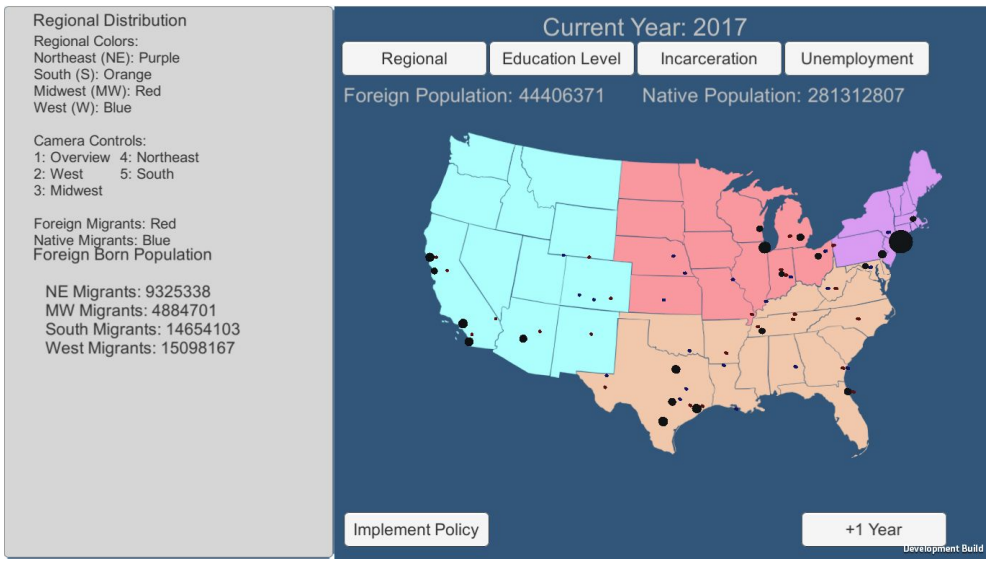
The simulator UI also contains a few extra displays and controls. These are as follows:

1. The current year is displayed at the top of the display. The simulator will always start at the year 2017.
2. The panel switching buttons are below the year. They change the left side panel and visual representation to display the information each button describes.
3. The current native born and foreign born populations are displayed below the panel switching buttons. They will always update to show the populations the label describes.
4. The +1 Year button in the bottom right of the display. Pressing the button will increment the year and apply any annual birth rates, death rates, and migrant increases.
5. The policy panel button is in the bottom left of the display. Pressing the button will display the policies screen.

## Screenshots

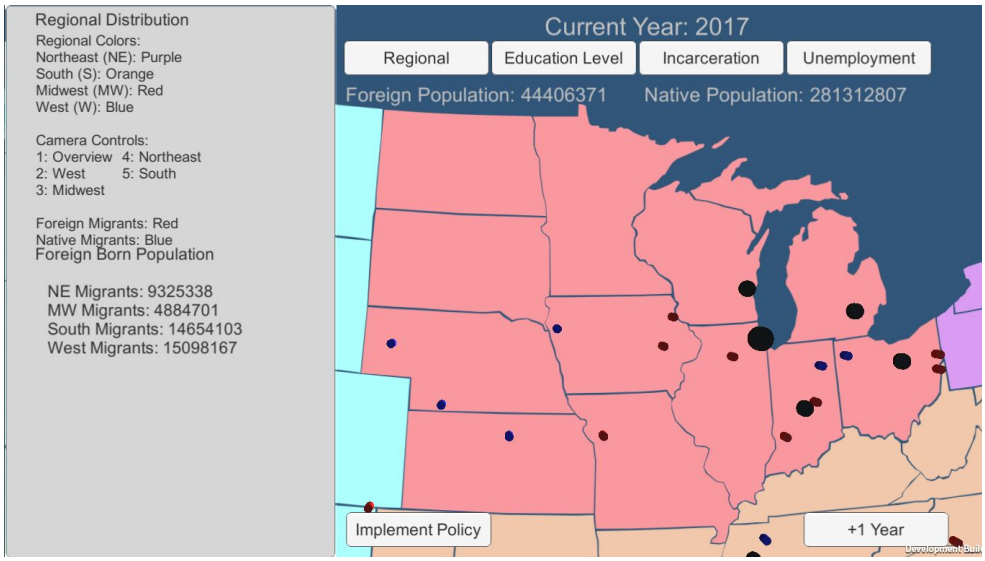
The screens available in the game are shown below. Each screen contains a short description of its purpose, as well as how to access the screen from within the simulator.

### Regional Map



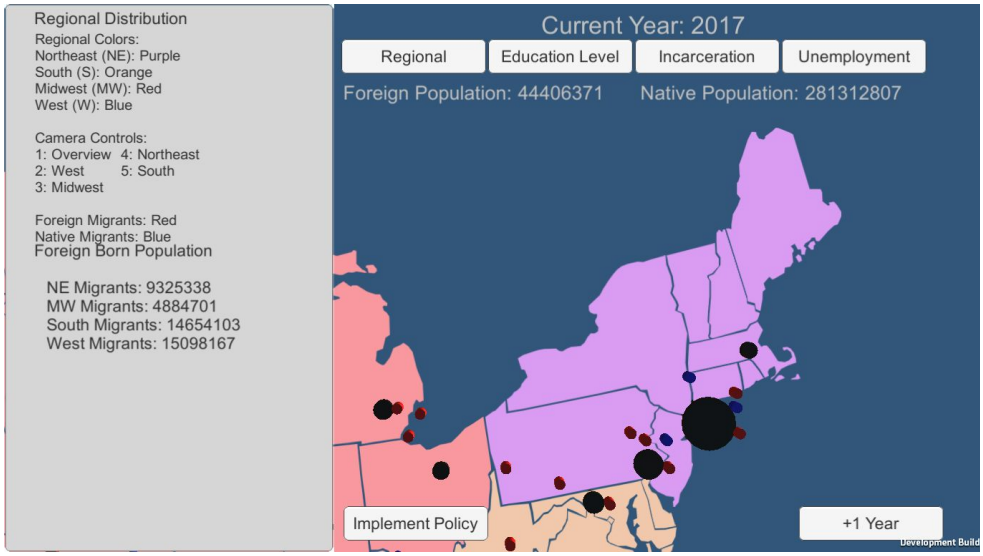

This is the starting screen of the game. It displays the full overview of the US map as well as the agent system in addition to the Number of foreign born residents of each region. It also contains a running total of Foreign Born Population and Native Born populations as well as the count of the current year. It contains buttons to increment the year by 1 as well as open the policy menu and all the other informational windows.

## Midwest Screen



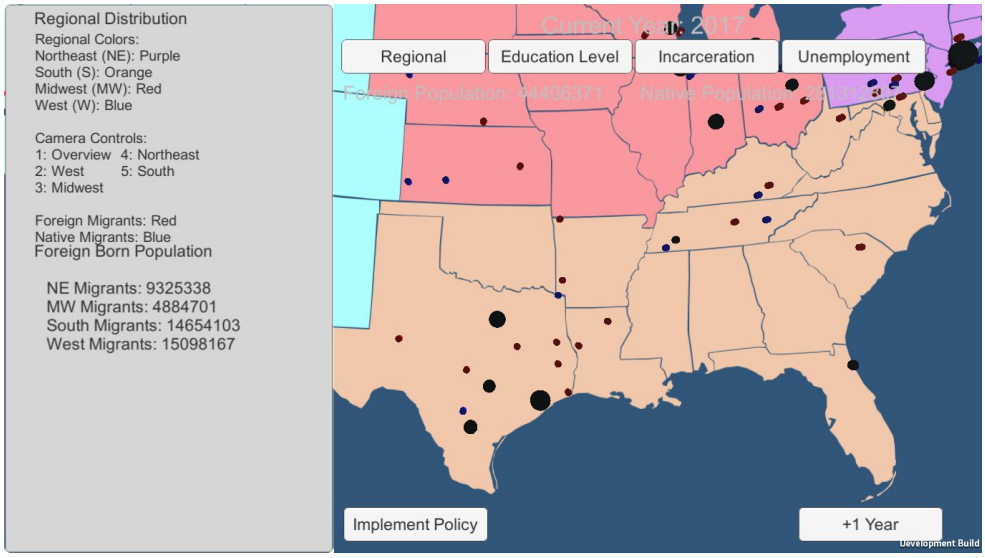

This screen zooms in on the Midwest region of the US to show the agent system in that region specifically. It is accessed by pressing the 3 key.

## Northeast Screen



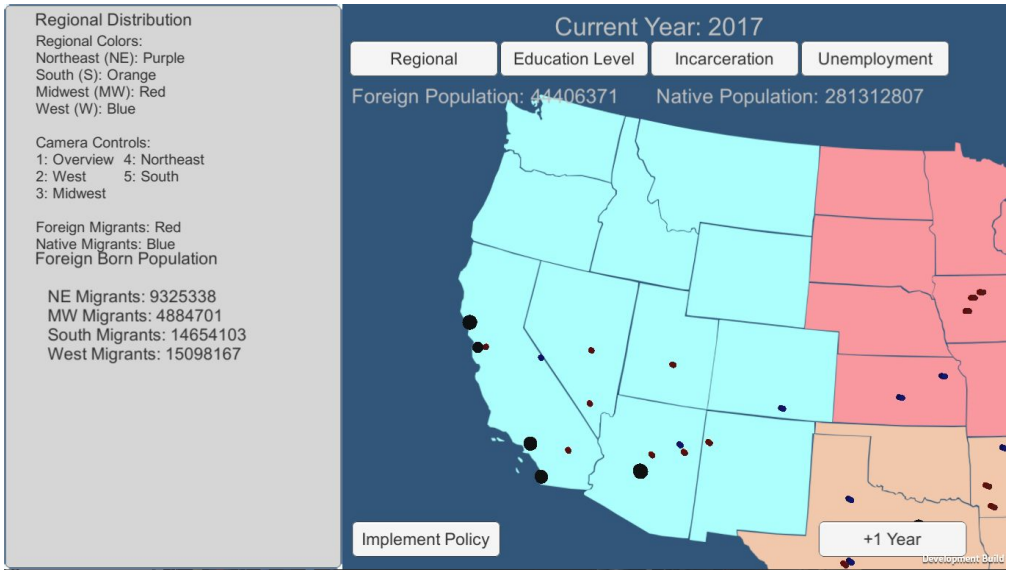

This screen zooms in on the Northeast region of the US to show the agent system in that region specifically. It is accessed by pressing the 4 key.

## South Screen



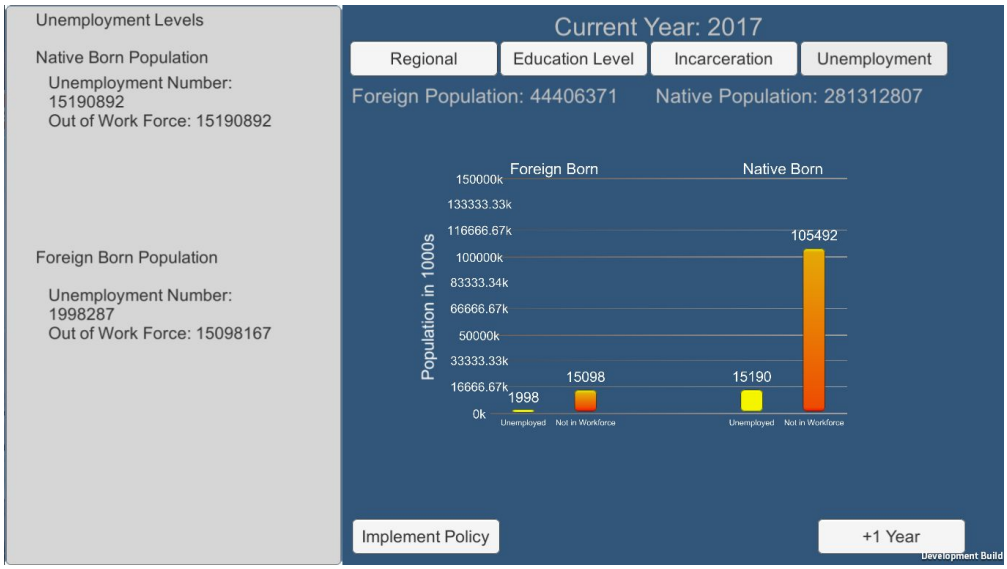

This screen zooms in on the South region of the US to show the agent system in that region specifically. It is accessed by pressing the 5 key.

## West Screen



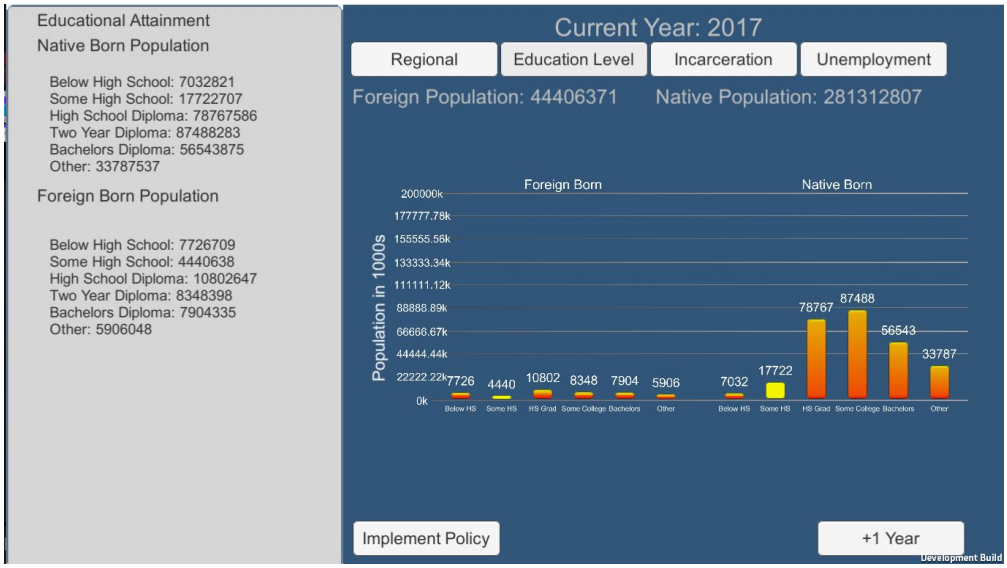

This screen zooms in on the West region of the US to show the agent system in that region specifically. It is accessed by pressing the 2 key.

## Unemployment Screen



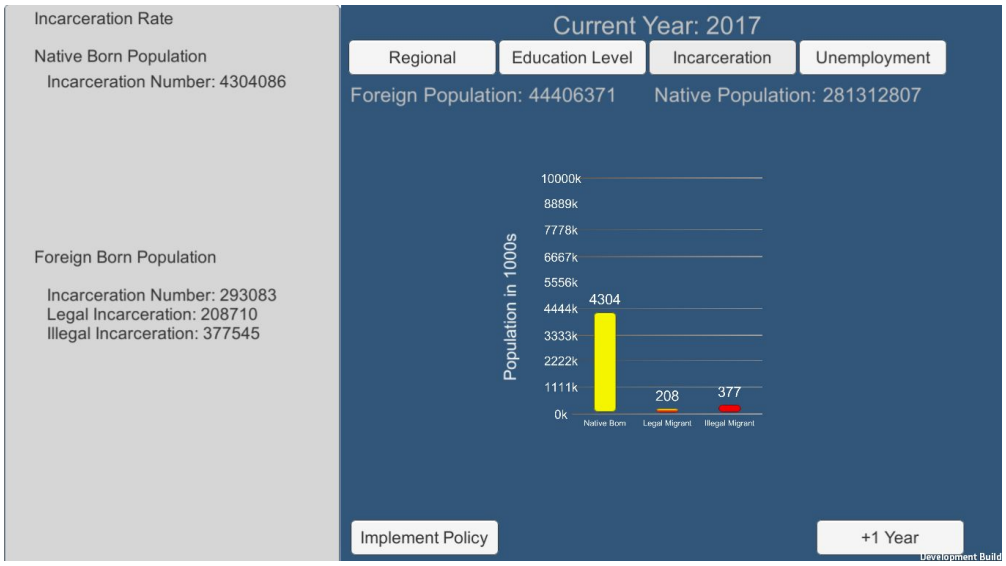

This screen shows the number of unemployed individuals and individuals not in the workforce for both Foreign Born and Native born populations. It is updated every time the year is advanced. It is accessed by pressing the ‘Unemployment’ button.

## Education Screen




This screen shows the number of people in each level of education for both Foreign Born and Native born populations.The data is updated every time the year is advanced.. It is accessed by pressing the Education Level button.

## Incarceration Screen




This screen shows the number of incarcerated individuals in 3 populations, Native Born, Legal migrant and Illegal migrant It is updated every time the year is advanced. It is accessed by pressing the ‘Incarceration’ button

## Policy Window


This is the policy panel, where population parameters can be changed via abstracted policies. There are currently 4: Migrants per Year, Jobs Program, Education Program and Crime program. The migrants per year are changed via a slider and the rest are leveled up or down to a minimum of 0. It is accessed by clicking the ‘Implement Policy’ button on the main screen.

# Back End

The back end of the simulation tool is fairly simple.  Most of the work is done by the migration framework.  Any communication from the front end interface to the framework is handled by a driver class, since Unity has some odd problems directly communicating with a static script.  The driver class is attached to an empty game object, and referencing that game object allows access to the driver script, and it uses standard public functions to access the main framework class.

There are also smaller drivers and controller scripts for the camera, map, and graphs.  These controllers are somewhat independent from the framework, with the exception of a few variable accesses, and are used for controlling the Unity tool's dynamic components.

## FrameworkDriver

### Variables

```
public Text currentYearText;
int currentYear = 2017;
public Text usPopText;
public Text frPopText;

public Text migrantsText;

public Text usEdText;
public Text frEdText;

public Text usIncText;
public Text frIncText;

public Text usUnemText;
public Text frUnemText;

public Text edPolicyText;
int edPolicyLevel = 0;
public Text jobPolicyText;
int jobPolicyLevel = 0;
public Text jailPolicyText;
int jailPolicyLevel = 0;

public Slider migrantSlider;
public Text sliderText;
```

The variables for the driver are used for identifying specific UI elements that the driver changes. It also contains the values for policy levels and the year.

### Functions

```
void Start ()
{
    updateAllData();
    sliderText.text = migrantSlider.value.ToString();
}
```

Start(): Run on tool startup.  It sets up all the data to make sure framework values are pulled immediately, and sets the migrant slider text to display its current value.

```
private void updateAllData()
{
    currentYearText.text = "Current Year: " + currentYear;
    frPopText.text = "Foreign Population: " + MigrationFramework.getFrPop();
    usPopText.text = "Native Population: " + MigrationFramework.getUsPop();

    migrantsText.text = "NE Migrants: " + MigrationFramework.getSpecificPopValue("neMigrants")
        + "\nMW Migrants: " + MigrationFramework.getSpecificPopValue("mwMigrants")
        + "\nSouth Migrants: " + MigrationFramework.getSpecificPopValue("sMigrants")
        + "\nWest Migrants: " + MigrationFramework.getSpecificPopValue("wMigrants");

    usEdText.text = "Below High School: " + MigrationFramework.getSpecificPopValue("usLessThanHighschool")
        + "\nSome High School: " + MigrationFramework.getSpecificPopValue("usHighschool")
        + "\nHigh School Diploma: " + MigrationFramework.getSpecificPopValue("usHighschoolGrad")
        + "\nTwo Year Diploma: " + MigrationFramework.getSpecificPopValue("usTwoYear")
        + "\nBachelors Diploma: " + MigrationFramework.getSpecificPopValue("usBachelors")
        + "\nOther: " + MigrationFramework.getSpecificPopValue("usOther");

    frEdText.text = "Below High School: " + MigrationFramework.getSpecificPopValue("frLessThanHighschool")
        + "\nSome High School: " + MigrationFramework.getSpecificPopValue("frHighschool")
        + "\nHigh School Diploma: " + MigrationFramework.getSpecificPopValue("frHighschoolGrad")
        + "\nTwo Year Diploma: " + MigrationFramework.getSpecificPopValue("frTwoYear")
        + "\nBachelors Diploma: " + MigrationFramework.getSpecificPopValue("frBachelors")
        + "\nOther: " + MigrationFramework.getSpecificPopValue("frOther");

    usIncText.text = "Incarceration Number: " + MigrationFramework.getSpecificPopValue("usInc");

    frIncText.text = "Incarceration Number: " + MigrationFramework.getSpecificPopValue("frInc")
        + "\nLegal Incarceration: " + MigrationFramework.getSpecificPopValue("frLegalInc")
        + "\nIllegal Incarceration: " + MigrationFramework.getSpecificPopValue("frIllegalInc");

    usUnemText.text = "Unemployment Number: " + MigrationFramework.getSpecificPopValue("usUnEm")
        + "\nOut of Work Force: " + MigrationFramework.getSpecificPopValue("usUnEm");

    frUnemText.text = "Unemployment Number: " + MigrationFramework.getSpecificPopValue("frUnEm")
        + "\nOut of Work Force: " + MigrationFramework.getSpecificPopValue("frNotWorking");
}
```

updateAllData(): This function is used for updating all UI display text with the proper values.  It is called after any population changes are made.

```
public void incrementYear()
{
    MigrationFramework.yearlyPopUpdate();
    currentYear++;
    updateAllData();
}
```

incrementYear(): Tied to the +1 Year UI button.  Calls the framework's year update function, then updates all UI values.  It also increments the current year for the UI.

```
public void educationPolicy(bool up)
{
    float[] values = { -0.011f, 0.004f, 0.003f, 0.002f, 0.001f, 0.001f, -0.011f, 0.004f, 0.003f, 0.002f, 0.001f, 0.001f };
    if (!up)
        values = Array.ConvertAll(values, f => f * -1);

    string[] keys = { "usLessThanHighschool", "usHighschool", "usHighschoolGrad",
        "usTwoYear", "usBachelors", "usOther", "frLessThanHighschool",
        "frHighschool", "frHighschoolGrad", "frTwoYear", "frBachelors", "frOther" };

    MigrationFramework.policyChange(values, keys);
    updateAllData();
    if (up) edPolicyLevel++;
    else edPolicyLevel--;
    edPolicyText.text = "Level: " + edPolicyLevel;
}

public void jobPolicy(bool up)
{
    float[] values = { -0.004f, -0.004f };
    if (!up)
        values = Array.ConvertAll(values, f => f * -1);

    string[] keys = { "usUnEm", "frUnEm" };

    MigrationFramework.policyChange(values, keys);
    updateAllData();
    if (up) jobPolicyLevel++;
    else jobPolicyLevel--;
    jobPolicyText.text = "Level: " + jobPolicyLevel;
}

public void jailPolicy(bool up)
{
    float[] values = { -0.001f, -0.001f, -0.001f, -0.001f };
    if (!up)
        values = Array.ConvertAll(values, f => f * -1);

    string[] keys = { "usInc", "frInc", "frLegalInc", "frIllegalInc" };

    MigrationFramework.policyChange(values, keys);
    updateAllData();
    if (up) jailPolicyLevel++;
    else jailPolicyLevel--;
    jailPolicyText.text = "Level: " + jailPolicyLevel;
}
```

____Policy(): These three functions are used for tying the front end of the Unity tool to the simulation framework.  They create an array of value changes for each changed subpopulation percentage and an array of the keys to change, then send those arrays to the framework.

The function also takes a boolean parameter that determines if the policy is a level up or a level down; all percentage changes are defaulted to a policy level up, but the input boolean is false, this indicates a level down, and all percentage changes have their sign flipped.

These functions also take care of the policy level.  If the policy is leveled up, the level value is changed accordingly, and vice versa for a level down.

All of the current value adjustments were picked arbitrarily and are not based on research. The only criteria when they were being chosen was that all the rates must still add up to 1.0f in order to not mistakenly boost the total population.

```
public void setMigrantsPerYear()
{
    MigrationFramework.setMigrantsLevels((int)migrantSlider.value);
    sliderText.text = migrantSlider.value.ToString();
}
```

setMigrantsPerYear(): This function is used on the migrants per year slider, which alters the base number of migrants that is added every year increment.  Whenever the slider value changes, this function is called.  It takes the value of the migrant slider and passes it to the framework, then also sets the UI display text to the new value.

## CityBehavior

```
public class CityBehavior : MonoBehaviour {

    float scaler = 1;
    float baseXScale;
    float baseZScale;

    // Use this for initialization
    void Start ()
    {
        baseXScale = this.gameObject.transform.localScale.x;
        baseZScale = this.gameObject.transform.localScale.z;
    }

    public void ScaleCity()
    {
        scaler += 0.05f;
        UpdateScale();
    }

    public void DescaleCity()
    {
        scaler -= 0.05f;
        UpdateScale();
    }

    private  void UpdateScale()
    {
        this.gameObject.transform.localScale = new Vector3(baseXScale * scaler,
            1, baseZScale * scaler);
    }
}
```

This class is used for controlling the size of a city.  It starts with a base X and Z scaling, and it also contains functions to increase and decrease the scale of the city.  It also has a private function that will update the city's scale every time the scale changes.

## AgentBehavior

```
public class AgentBehavior : MonoBehaviour {

    float timer = 0;
    string target = "";
    Vector3 prevPosition;
    bool flag = false;

    GameObject agentDriver;
    // Use this for initialization
    void Start () {
        prevPosition = this.gameObject.transform.position;
    }

    // Update is called once per frame
    void Update ()
    {
        if (true)
        {
            if (prevPosition == this.gameObject.transform.position)
                Destroy(this.gameObject);
        }
        flag = true;
        prevPosition = this.gameObject.transform.position;
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.name == target)
        {
            collision.gameObject.GetComponent<CityBehavior>().ScaleCity();
            Destroy(this.gameObject);
        }
    }

    public void SetTarget(string t) { target = t; }
}
```

This class is used for controlling the “migrant” agents moving around the regional map.

- There is a check for if the agent happens to freeze, get stuck, or any other reason it cannot move. If the agent doesn’t move at all between game tics, it will be destroyed so the map doesn’t get clogged up with stuck agents.
- When an agent arrives at its goal city, it is destroyed and the city it arrived at scales to show the migrant has entered the city.

## AgentDriver

```
public class AgentDriver : MonoBehaviour {

    public GameObject foreignPrefab;
    public GameObject nativePrefab;

    GameObject[] cities;
    float timer = 1;

    // Use this for initialization
    void Start ()
    {
        cities = GameObject.FindGameObjectsWithTag("City");
    }

    // Update is called once per frame
    void Update ()
    {
        timer += Time.deltaTime;
        if (timer >= 1)
        {
            timer = 0;
            int r = (int)Random.Range(1, 6);
            for (int i = 0; i < r; i++)
            {
                int choice = (int)Random.Range(0, 2);
                GameObject chosenPrefab = null;
                if (choice == 0)
                    chosenPrefab = foreignPrefab;
                else if (choice == 1)
                    chosenPrefab = nativePrefab;

                GameObject temp = Instantiate<GameObject>(chosenPrefab,
                    cities[Random.Range(0, cities.Length)].transform.position, new Quaternion(90, 0, 0, 0));
                GameObject targetCity = cities[Random.Range(0, cities.Length)];
                targetCity.GetComponent<CityBehavior>().DescaleCity();
                temp.GetComponent<AgentBehavior>().SetTarget(targetCity.name);
                temp.GetComponent<NavMeshAgent>().SetDestination(targetCity.transform.position);
            }
        }
    }
}
```

This class is used for spawning agents onto the map. Every second, somewhere between 1 and 6 agents spawn on the map as either a foreign or native migrant. Migrants spawn out of a city, and the city it spawns out of is descaled to show someone has left the city. The migrant then is given a new target city, which they will move to.

## CameraController

```
public class CameraController : MonoBehaviour {

    public GameObject ri;

    public Camera cam;
    bool regionOn = true;
    // Use this for initialization
    void Start ()
    {
        SwitchOnRegionCamera();
    }

    // Update is called once per frame
    void Update()
    {
        if (regionOn)
        {
            if (Input.GetKeyDown("1")) cam.transform.localPosition = this.gameObject.transform.GetChild(0).transform.localPosition;
            if (Input.GetKeyDown("2")) cam.transform.localPosition = this.gameObject.transform.GetChild(1).transform.localPosition;
            if (Input.GetKeyDown("3")) cam.transform.localPosition = this.gameObject.transform.GetChild(2).transform.localPosition;
            if (Input.GetKeyDown("4")) cam.transform.localPosition = this.gameObject.transform.GetChild(3).transform.localPosition;
            if (Input.GetKeyDown("5")) cam.transform.localPosition = this.gameObject.transform.GetChild(4).transform.localPosition;
        }
    }

    public void SwitchOffRegionCamera()
    {
        cam.transform.localPosition = this.gameObject.transform.GetChild(5).transform.localPosition;
    }

    public void SwitchOnRegionCamera()
    {
        regionOn = true;
        cam.transform.localPosition = this.gameObject.transform.GetChild(0).transform.localPosition;
    }
}
```

The camera controller class is used for moving the camera around to different parts of the region map. A boolean is used to indicate if the region map is being viewed or not, and when the region toggle is on, the default camera position will be an overview of the whole map. The camera can then move to focus on specific parts of the map via key inputs. When the region map is not being shown, the camera moves away from the region map so it is not shown.

## Graph Updaters

Each of the graphs have separate scripts to get the updated values from the framework and reflect those in the different graphs. All of them use methods from the ChartandGraph framework used for implementations of the graphs as well as getters from the migration framework class. It also divides the numbers by 1000 to make the numbers smaller for handling by the graph and charts framework.

UpdateUnEmp class

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ChartAndGraph;

public class UpdateUnEmpGraph : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        BarChart barChart = GetComponent<BarChart>();

        barChart.DataSource.SetValue("Unemployed", "Native Born", MigrationFramework.getSpecificPopValue("usUnEm") / 1000);
        barChart.DataSource.SetValue("Not in Workforce", "Native Born", MigrationFramework.getSpecificPopValue("usNotWorking") / 1000);
        barChart.DataSource.SetValue("Unemployed", "Foreign Born", MigrationFramework.getSpecificPopValue("frUnEm") / 1000);
        barChart.DataSource.SetValue("Not in Workforce", "Foreign Born", MigrationFramework.getSpecificPopValue("frNotWorking") / 1000);
    }
}

```

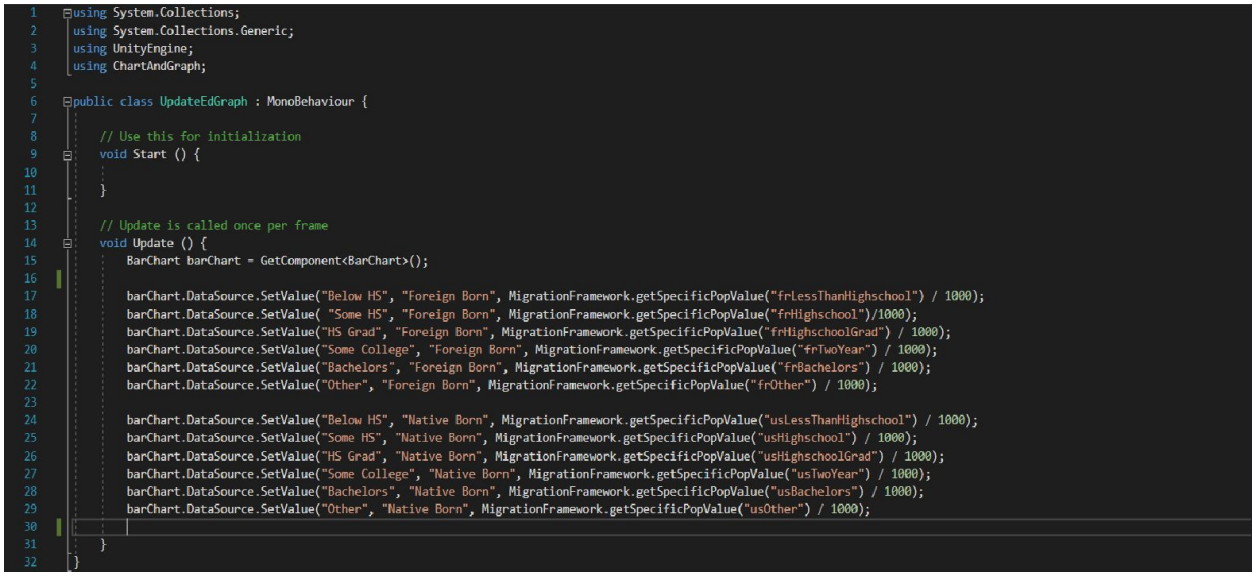
This script updates the data for the unemployment graph.

UpdateIncGraph

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ChartAndGraph;


public class UpdateIncGraph : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        BarChart barChart = GetComponent<BarChart>();
        //if (barChart != null) {
        //barChart.DataSource.SlideValue( "Some HS", "Native Born", 20, 4f);
        barChart.DataSource.SetValue("Native Born", "All", MigrationFramework.getSpecificPopValue("usInc") / 1000);
        barChart.DataSource.SetValue("Legal Migrant", "All", MigrationFramework.getSpecificPopValue("frLegalInc") / 1000);
        barChart.DataSource.SetValue("Illegal Migrant", "All", MigrationFramework.getSpecificPopValue("frIllegalInc") / 1000);

        //}
    }
}
```

This script updates the data for the Incarceration graph.

UpdateEdGraph



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using ChartAndGraph;

public class UpdateEdGraph : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {
        BarChart barChart = GetComponent<BarChart>();

        barChart.DataSource.SetValue("Below HS", "Foreign Born", MigrationFramework.getSpecificPopValue("frLessThanHighschool") / 1000);
        barChart.DataSource.SetValue( "Some HS", "Foreign Born", MigrationFramework.getSpecificPopValue("frHighschool")/1000);
        barChart.DataSource.SetValue("HS Grad", "Foreign Born", MigrationFramework.getSpecificPopValue("frHighschoolGrad") / 1000);
        barChart.DataSource.SetValue("Some College", "Foreign Born", MigrationFramework.getSpecificPopValue("frTwoYear") / 1000);
        barChart.DataSource.SetValue("Bachelors", "Foreign Born", MigrationFramework.getSpecificPopValue("frBachelors") / 1000);
        barChart.DataSource.SetValue("Other", "Foreign Born", MigrationFramework.getSpecificPopValue("frOther") / 1000);

        barChart.DataSource.SetValue("Below HS", "Native Born", MigrationFramework.getSpecificPopValue("usLessThanHighschool") / 1000);
        barChart.DataSource.SetValue("Some HS", "Native Born", MigrationFramework.getSpecificPopValue("usHighschool") / 1000);
        barChart.DataSource.SetValue("HS Grad", "Native Born", MigrationFramework.getSpecificPopValue("usHighschoolGrad") / 1000);
        barChart.DataSource.SetValue("Some College", "Native Born", MigrationFramework.getSpecificPopValue("usTwoYear") / 1000);
        barChart.DataSource.SetValue("Bachelors", "Native Born", MigrationFramework.getSpecificPopValue("usBachelors") / 1000);
        barChart.DataSource.SetValue("Other", "Native Born", MigrationFramework.getSpecificPopValue("usOther") / 1000);

    }
}
```

This script updates the data for the Educational Attainment graph.

# Future Improvements

Transitioning our project from algorithmic-based calculations to percentage-based resulted in changing a few major parts of the project, typically taking parts of the original process and simplifying them. This resulted in an application that was still accurate, but lacked some of the original content of our work. This content turned into stretch goals and future improvements, and includes the following:

- Country select: The original project allowed a user to select between 5-10 countries to simulate, each with their own unique population starting values, growths, and subpopulation values.
- Migrant map tie-ins: A lot of the region map is mostly cosmetic. There were plans to tie the number of spawning migrants to different numbers in the framework, but due to time constraints, this was unable to be done. While the user can still see migrants moving around regions naturally through the tool, the ability to alter spawn rates with the framework values would provide more depth to that tool.
- Multi-country outside system: Currently, there are two systems in the simulation: the US and everything else. Our original plan was to have different major countries or regions of the world represent different systems, and have their migrants all distinguishable so users could see where migrants were coming from. When the initial project was reworked, it became too difficult to keep this function due to the numbers we were using and the way we implemented policies. With more time, extensive research could be taken to provide accurate migration alterations between different countries or regions, and then this idea could be implemented.

- More accurate population growth simulation: Currently our framework does entirely linear calculations for the population growth. While this is useful for showing how our framework displays information, it is not accurate to the real world.
- Well Researched, realistic policies: The current policies are placeholders with estimated and arbitrary placeholder values. In the future the policy effects should be well-researched and verified like the base parameters.
- UI Updates: There are many similar map visuals used by simulators or visualizers with nicer UIs, such as the corona virus map shown here: https://coronavirus.jhu.edu/map.html. These could serve as an inspiration for improving our UI.

# Github

Below is the Github link to the repository that contains all the code and unbuild project files for this Thesis.
https://github.com/jMoneee/MigrationSimulator

# Final Build

Below is the link to a download for the final build of the project. You can download it from the link, extract it and launch Migration Simulator.exe to run it.
https://drive.google.com/file/d/1oidAuSgz_sNXL4u7Oi0UB0ELPXnRsd81/view?usp=sharing

# Works Cited

"Criminal Immigrants: Their Numbers, Demographics, and Countries of Origin." Cato Institute, 12 Apr. 2017, www.cato.org/publications/immigration-reform-bulletin/criminal-immigrants-their-numbers-demographics-countries.

"FastStats - Deaths and Mortality." Centers for Disease Control and Prevention, Centers for Disease Control and Prevention, 3 May 2017, www.cdc.gov/nchs/fastats/deaths.htm.

Livingston, Gretchen. "Hispanic Women No Longer Account for the Majority of Immigrant Births in the U.S." Pew Research Center, Pew Research Center, 8 Aug. 2019, www.pewresearch.org/fact-tank/2019/08/08/hispanic-women-no-longer-account-for-the-majority-of-immigrant-births-in-the-u-s/.

Major Cities in the USA - EnchantedLearning.com, www.enchantedlearning.com/usa/cities/.

Radford, Jynnah, and Luis Noe-Bustamante. “Immigrants in America: Current Data and Demographics.” Pew Research Center's Hispanic Trends Project, Pew Research Center, 3 Jan. 2020, www.pewresearch.org/hispanic/2019/06/03/facts-on-u-s-immigrants-current-data/.

Shimasawa, Manabu, and Kazumasa Oguro. “Impact of Immigration on the Japanese Economy: A Multi-Country Simulation Model.” Journal of the Japanese and International Economies, Academic Press, 21 May 2010, www.sciencedirect.com/science/article/pii/S0889158310000195.