Controllable, Demographically-Guided Character Generation using Stochastic Logic Programming*

Ian Horswill[1]

[1] Northwestern University, 2233 Tech Drive, Evanston, Illinois, USA

Abstract

Many games involve story worlds whose possible character features (names, clans, ethnic/racial backgrounds, etc.), are entirely invented. In these worlds, designers can fiat the statistical distribution of features, and so often use independent, uniform distributions. In games set in approximations to the real world, however, values of features such as names are often conditioned on values of other features such as ethnicity and gender. In this paper, I describe a system that generates demographically-plausible names, ethnicities, genders, height, weight, hair color, and eye color for non-player characters in a table-top role-playing game set in present-day San Francisco. The system allows game masters to pre-specify desired features and samples other features based on US census data and other sources to generate statistically plausible values. The system is implemented in a stochastic version of logic programming that supports feature structure unification. I will discuss how the implementation works and why it is an appealing choice.

Keywords

Procedural content generation, logic programming, Bayes nets

1. Introduction

Characters in table-top role-playing games (TTRPGs) are typically defined by some set of features, such as name, gender, "race," class, and numerical stats such as intelligence and dexterity. Some games define characters by more complex attributes such as background history. Automatic generators for non-player characters (NPCs) allow a human "game-master" (GM) to specify the values of some features, then fill in sensible combinations of values for the other features, typically in some randomized manner.

In some cases, feature values are tightly constrained by game mechanics. For example, the mechanics might include a build-point system that assigns costs to different features values and limits the total cost summed across all such features. These can be generated using a constraint-solver such as *Z3* [1] or *CatSAT* [2].

Other features, such as name, are typically unconstrained by mechanics. For settings (story worlds) that are entirely invented, these can be generated by randomly picking from a stock of pre-authored names, possibly conditioned on other setting-appropriate attributes such as location of birth, religion, social class, or ethnic origin. They can even be randomly generated from context-free grammars, as in *Dwarf Fortress* [3].

However, when the setting is based on the real world, players have expectations of what combinations are common, uncommon, strange, or even impossible. While deliberate violations of those expectations can be a valid choice, accidental violations can be jarring or even offensive. They can also play into subtle prejudices if, for example, the system always generates thief characters who are Black or have stereotypically racialized names.

In this paper, I discuss the issues in building a system that generates NPCs using open-source demographic data. The system generates names, genders, ethnicities, ages, and "driver's-license appearance" (height, weight, hair, and eye color) for characters that match the actual population as

*Joint AIIDE Workshop on Experimental Artificial Intelligence in Games and Intelligent Narrative Technologies, November 10-11, 2025, Edmonton, Canada.

[1] ian@northwestern.edu (I. Horswill)

closely as possible. The GM can specify values for any of the features and the system will fill in values for the other features based that are statistically plausible given the specified features.

I will begin by describing the setting of the game and the use of the generator. Then, I will discuss its statistical model and its demographic sources, the implementation of the generator itself, and its problems and limitations. Finally, I will discuss related work and conclude.

2. Setting

The generator was built for a campaign of the table-top role-playing game *DELTA GREEN* [4], [5], a cosmic horror game in which players play are government agents fighting to delay humanity's eventual destruction by creatures of the Lovecraft mythos.

The campaign is set in the Tenderloin district of present-day San Francisco. The Tenderloin is an area rich in history. A historic center of illegal alcohol, gambling, and prostitution, Dashiell Hammet's *The Maltese Falcon* is set there [6], [7]. It was one of the city's first gay neighborhoods, predating The Castro. Today, it is known in large part for poverty, homelessness, and drug addiction. It also has the city's highest concentration of children, with an estimated 3000 children in an area little more than a third of a square kilometer (0.9km^2, 0.35mi^2).

NPCs in the campaign are drawn from a variety of organizations, most of them real:

- The Bohemian Club [8]
- Catholic Charities (NGO) [9]
- (US) Central Intelligence Agency
- (US) Centers for Disease Control
- (US) Federal Bureau of Investigation
- *Los reyes del azúcar* (fictional street gang)
- San Francisco Office of the Medical Examiner
- San Francisco Police Department (particularly the Tenderloin and Southern stations)
- San Francisco State University School of Medicine
- Urban Alchemy (NGO) [10]
- Various startup companies (fictional)
- The Wisdom Institute (fictional NGO and new religious movement)

NPCs are also drawn from other demographic groups, including both housed and unhoused residents of the Tenderloin, SoMA, and Nob Hill districts, shopkeepers in those districts, and various entrepreneurs and venture capitalists. It is therefore necessary to be able to invent arbitrary characters from any of these organizations or demographic groups on demand.

3. Examples of system use

To give a better sense of the use case of the system, let's look at some examples commands and system output. The user specifies what features they want to fix by placing them in { }'s. To generate a completely arbitrary character with no constraints on their features, we use the command:

```
NPC {}
```

This leaves all features unspecified. An example output is:

```
Eric Mendoza
•  Hispanic Latino male
•  Age: 20
•  Height: 5'8"
```

- Weight: 224lbs
- Hair: brown
- Eyes: brown

```
{ age:20 group:ca_resident  gender:male  ethnicity:hispanic_latino  cdcEth:Hispanic
  height:172.5393  bmi:34.05986  weight:101.39556  eyes:brown hair:brown
  firstName:Eric lastName:Mendoza }
```

The bracketed expression at the end gives all the features for the character. Numbers are different because the internal features are in SI units (kilograms and centimeters), while the bulleted list above is converted to Imperial units (pounds, feet, inches).

The command:

```
NPC { eyes:blue  height:190 }
```

Specifically generates tall (190cm) characters with blue eyes, for example:

Donald Cameron

- white male
- Age: 22
- Height: 6'3"
- Weight: 211lbs
- Hair: brown
- Eyes: blue

```
{ eyes:blue  height:190  age:22 group:ca_resident  gender:male  ethnicity:white cdcEth:white
  bmi:26.54374  weight:95.8229  hair:brown
  firstName:Donald  lastName:Cameron }
```

The GM can also specify particular demographic groups important for the campaign: general California resident, unhoused Tenderloin resident, San Francisco Police, or STEM worker (doctors, scientists tech workers, etc.). Thus, the query:

```
NPC { group:sfpd  age:38 }
```

to produce a 38-year-old police officer produces the character:

Randall Arnold

- White male
- Age: 38
- Height: 6'1"
- Weight: 230 lbs
- Hair: red
- Eyes: brown

```
{ group:sfpd  age:38  gender:male  ethnicity:white  cdcEth:white
  height:185.58723  bmi:30.281744  weight:104.29826  eyes:brown  hair:red
  firstName:Randall  lastName:Arnold }
```

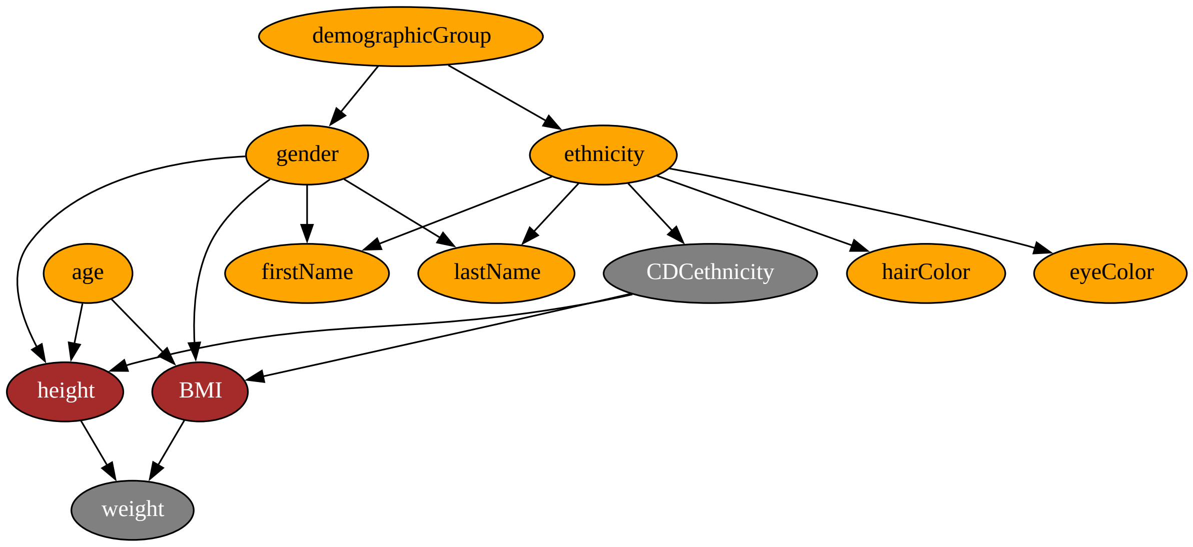

Figure 1: Dependency structure of statistical model. Nodes in orange are defined by discrete probability distributions. These form a standard Bayes net. Nodes in brown (height and BMI) are continuous values whose distributions are described by piecewise-linear functions derived-from CDC data. Nodes in grey (CDCethnicity and weight) are deterministic (non-stochastic) functions.

The underlying system is a logic program, so more complex queries are also possible by combining constraints. For example, a character of random age, but in their 20s could be generated using:

```
[NPC { age:?age }] [>= ?age 20] [< ?age 30]
```

Or a malnourished character could be generated using the query:

```
[NPC { bmi:?bmi }] [< ?bmi 18]
```

4. Statistical model

The structure of the model is shown in Figure 1. Nodes in orange are discrete random variables, although in practice demographic group is an input to the system and not stochastic. Nodes in brown (height and BMI) are continuous random variables. Those in grey are deterministic (non-stochastic) functions of their dependencies.

One weakness of the model is that height and body-mass index (BMI) are the only features that depend on age. Thus, although mortality rates vary with gender and ethnicity, both gender and ethnicity are treated as statistically independent of age. Similarly, since immigration from different groups varies over time, the distribution of surnames should vary with age, but is treated as independent in this system. And the distribution of baby names, and hence given names varies over time, meaning that first name should also depend on age, but does not in our system. I am not aware of any dependence of life expectancy on hair or eye color, so their independence is likely valid.

4.1. Ethnicity

Ethnicity is represented as discrete categories using two different systems taken from US Government sources. The primary representation is the pre-1997 US Office of Management and Budget (OMB) categories, which were used for surname data in the 2000 and 2010 Census, and consists of the categories:

- Hispanic or Latino
- Non-Hispanic White
- Non-Hispanic Black or African-American
- Non-Hispanic Asian or Native Hawaiian or other Pacific Islander
- Non-Hispanic American Indian or Alaska Native
- Non-Hispanic multi-race

However, data sourced from the US Centers for Disease Control uses a simpler categorization of:

- White
- Black
- Latino
- Asian
- All (data for the whole population)

We generate ethnicities using the Census (OMB) system and then map them to CDC categories to their closest CDC categories, save for mixed race and AIAN, which are mapped to All, meaning they are treated statistically as members of the general population rather than a specific group.

4.2. Gender

Given the paucity of statistical data on the non-binary population, the current system uses only the categories of male and female, although it does not assume those necessarily represent gender assigned at birth. The distribution of gender for the general population is taken to be 49% male, 51% female.

4.3. Last name (surname)

Frequencies of last names for different ethnicities were taken from the 2010 US Census Bureau data files [11]. This data is limited to names that occur at least 100 times in the census, so it has a shorter tail than normal.

The system does not currently support generation of more complicated names such as titles or other honorifics, middle names, surname-first order as is common in Asia, or Hispanic-style dual surnames. They would be trivial to add, given the appropriate statistical data.

4.4. First name (given name)

The Census Bureau does not release first name statistics broken down by ethnicity. However, specifically because of this, Tzioumis [12] compiled a list of first names appearing on mortgage applications, broken down by Census/OMB ethnicity category.

This data has a number of issues. Most obviously, mortgage applications are a biased sample of the population. They overrepresent high-SES (socio-economic status) member and wildly underrepresent the poor. They likely also overrepresent men, although not as much as they would have in 1950.

Another issue is that Tzioumis' data is not broken down by gender. So we must filter it using separate lists of common male and female given names. This means that names that *can* map to either gender, but only rarely do, are wildly overrepresented for the minority gender. An example is "Maria" which is 99.3% female in the world population and only 0.7% male nevertheless appears frequently as a male name in the generated data, requiring manual editing of the lists of male and female names.

4.5. Height, weight, and body-mass index

Height and body-mass index statistics are taken from the 2018 US National Health and Nutrition Examination Survey (NHANES), distributed by the US Centers for Disease Control [13]. These give 5th,10th,15th,25th,50th,75th,85th,90th, and 95th-percentile height and BMI for each decade of life for each gender and CDC ethnicity category. Values for height and BMI are generated by looking up the appropriate distribution given age and CDC ethnicity, then randomly choosing a percentile value in the range 5-95, and finally performing a linear interpolation of data to find the value that would correspond to the chosen percentile.

Weight is then given by:

$$W = 0.0001\,B\,H^2$$

where W is weight in kilograms, B is body-mass index, and H is height in centimeters [13].

4.6. Hair and eye color

I could find no reliable data on hair or eye color in the US. Many web sites list statistics, but they disagree, with some listing the prevalence of blond hair in the US as 33% and others at 2%. Few list sources, and those that did list sources were not supported by those sources; they appears to be AI-generated pages where an LLM hallucinated the data, but we cannot be sure. Percentages often didn't sum to 100. Deliberately generated AI queries had the same issues.

In the absence of reliable data, I adopted the following distributions:

- Hair: brown 49%, black 18%, blond 21%, red 12%
- Eyes: brown 45%, blue 17%, grey 10%, hazel 9%, amber 9%, green 9%

Note that the blond, red, blue, and green colors are artificially increased here because of the gating mechanism discussed below.

The hair- and eye-color systems have many serious limitations:

- It models only natural hair colors, no dyed colors.
- It does not attempt to model age-dependent graying of hair, hair coloring, or the use of color contact lenses to adjust eye color.
- It does not model the covariance of hair and eye color. For example, green eyes are much more common among red-haired people in reality, but in the system they are independent variables.
- It does not seriously model the covariance of hair or eye color with ethnicity. In the absence of good statistics, it simply gates certain colors (blond and red hair, blue and green eyes) on ethnicity. The color distributions above are treated as the color distributions for white people, and for non-whites, the blond/red/blue/green colors are simply disabled.

4.7. Demographic groups

Ground-truth demographic data is not available for most of the groups listed in section 2. However, data on gender and ethnicity for the following groups is available:

- *California Residents* [11]
- *San Francisco Police* [14]
- *STEM workers* (Science, Technology, Engineering, Math) [15]
- *Unhoused population* of San Francisco [16]
- *Senior tech executives* [17], [18]

The user can choose any of these demographic groups as the base for character generation. It determines the probability distributions over gender and ethnicity used for the rest of the generation process.

5. Generation

The system generates characters using rejection sampling [19]. This is the standard algorithm used for sampling Bayes nets [19] and is arguably the core of most Monte Carlo algorithms. In this case, rejection sampling is simply a loop that generates candidates based on the statistics of the net, then rejects them if any generated features disagree with the features specified by the user. A given sample is generated by starting with the roots of the DAG (directed acyclic-graph) in Figure 1, generating the feature value for each node based on its statistics, then generating values for its children based on their statistics and the values generated for their parents, and so on, until all features are generated. If the user has specified a value for a feature, and that value disagrees with the value generated, then the candidate is abandoned and a new candidate is generated starting from the beginning.

5.1. Implementation using stochastic logic programming

The system is implemented in the Step language [20], a classical logic programming language in the style of Prolog [21]. The current version uses one predicate (subroutine) per feature (age, name, etc.). Each predicate generates a random value for its associated feature, be it from a table of conditional probabilities, a parametric distribution, or a deterministic function.

Although the system could easily have been built in any language, Step's stochastic rules and feature structure unification make it very convenient for implementing heterogeneous Bayes nets.

5.1.1. Feature structures

Step supports two basic compound data structures: tuples (lists), notated with square brackets, e.g.:

```
[1 2 3 4]
```

and feature structures (dictionaries) notated with curly braces, for example:

```
{ group:sfpd  age:38  gender:male }
```

As in all logic programming languages, data structures may contain variables. In Step, those variables are notated with names beginning with `?`s. Thus, the feature structure:

```
{ gender:female  age:?age }
```

specifies an object whose `gender` feature is female and whose `age` feature is whatever the value of the variable `?age` may be. Step data structures read like JSON objects, albeit without commas between items. Unlike JSON, they can be matched using first-order unification.

5.1.2. Feature structure unification

Feature structures were originally developed for computational linguistics [22]. Although in one sense they're simply dictionaries, their appeal is based on the fact they can be matched using unification.

Unification is a technique for matching data structures for variables such as `?age`, above. Formally, it solves for what values must be assigned to the variables in the two data structures in order to make them be the same data structure. For atomic data, such as numbers, it's simple: two numbers unify if they're the same number, and fail to unify if they are not. A number and a variable unify by setting the variable equal to the number. For lists, the lists unify if they're the same length and their respective elements unify. So the lists:

[?x ?x] and [1 ?y]

unify by setting both ?x=?y=1. The first elements constrain ?x=1 and the second elements constrain ?x=?y.

Two feature structures unify (match) provided any features to which they both assign values, assign values to them that can be unified. The feature structures:

{ age: 15 } and { age: 22 }

do not unify because they assign incompatible values to the age feature, whereas the feature structures:

{ age: 15 } and { age: ?x }

do unify by setting ?x=15.

When feature structures contain different sets of features, their unification includes all the features from both structures. The unification of:

{ age: 15 } and { height: 150 }

is:

{ age: 15 height: 150 }

Unification in some sense pretends that each feature structure always had both features, albeit with unknown values (values that were variables).

5.1.3. Feature rules

In logic programs, a predicate (a subroutine) is defined by a series of rules of the form "the predicate is true of these arguments if these other things are true. For example, in Step the rule:

FirstNameGenderMatch ?name male: [MaleName ?name]

states that a call that matches the pattern [FirstNameGenderMatch ?name male] is true if the call [MaleName ?name] true, whatever ?name may be. If we call [FirstNameGenderMatch john male], the system matches (unifies) the arguments john and male with the rule's arguments, ?name and male, respectively. Unification yields ?name=john.

It then runs [MaleName ?name]. Since ?name=john, it really runs [MaleName john], which is true. So the original call [FirstNameGenderMatch john male] is also true.

Rules and calls may also contain complex expressions such as feature structures. The rule:

Eyes { ethnicity: ?eth eyes: ?color }: [EyeColor ?eth ?color]

takes whatever feature structure is passed as an argument to Eyes, and stores its ethnicity and eyes features in the variables ?eth and ?eyes, respectively. The argument can have other features too, and likely will, but they are ignored for the purpose of this rule. If the structure passed in does not have those features, unification *adds them* with unbound variables as their values. Those values are then bound by the subsequent call to EyeColor, thus filling in the values in the argument.

As a result, calling [Eyes ?npc] when ?npc={ ethnicity:black }, will have the effect of randomly choosing an eye color for the character, for example, brown, and extending the value of ?npc to be:

```
{ ethnicity:black  eyes:brown }
```

5.1.4. Weighted rules

A typical predicate is defined by multiple rules which are tried until one works, or until they are exhausted, in which case the call to that predicate fails, meaning it is false. Step allows the predicates to optionally run in random order and to place weights on the individual rules. The system then uses a weighted random shuffle of the rules, meaning that the probability of a given rule being tried first is its weight divided by the sum of weights for all the predicate's rules. This makes sampling of discrete distributions very easy to implement. The EyeColor predicate above is implemented by the rules:

```
[randomly] [fallible]
[45] EyeColor ? brown.
[17] EyeColor white blue.
[10] EyeColor white grey.
[9] EyeColor ? hazel.
[9] EyeColor ? amber.
[9] EyeColor white green.
```

Here the [randomly] annotation states that its rules should be shuffled for each call and the bracketed numbers before each rules give their probabilistic weights, that is, their relative probabilities of being chosen. We will discuss the [fallible] declaration below.

As discussed above, in the absence of data on the distribution of eye color for different ethnicities, this implementation only allows blue, grey, and green eyes for the "white" ethnicity. The other colors are allowed for all ethnicities and are treated as having equal distributions for other ethnicities, although this is likely inaccurate.

5.1.5. Rejection sampling

The rule given above for Eyes is logically correct but statistically incorrect:

```
Eyes { ethnicity: ?eth eyes: ?color }: [EyeColor ?eth ?color]
```

This works when the user has not pre-specified an eye color. However, when they have prespecified an eye color, it will narrow the rules for EyeColor to a single rule, making the choice of that color deterministic. Proper rejection sampling requires the eye color to be chosen randomly given the choice of ethnicity, then *compared* to any pre-specified value for the eye color. If they do not match, the character generation process must be restarted and repeated until it randomly generates a character with the desired eye color.

To understand why we need to restart the sampling process, consider the query:

```
NPC { eyes: brown }
```

in which the user specifies a brown eye color, but nothing else. The system will begin by randomly choosing ethnicity, gender, etc. given their statistics for the general population. If the above rule is used, it will then skip probabilistic generation of eye color and simply accept the color brown. That means the ethnicity has been generated using the statistics of the *general population*, not the statistics of the *brown-eyed population*, with the result that it would over-represent white people.

The solution is to generate the color randomly and then compare it to any value listed by the user, failing (restarting) if they cannot unify:

```
Eyes { ethnicity: ?eth eyes: ?color }: [EyeColor ?eth ?random] [= ?random ?color]
```

If the user did not specify a color, then ?color will simply be an unbound variable and the comparison at the end will assign it to ?random. If they did specify a color, then the comparison will only succeed if the randomly generated color was the same as the user-specified color.

The [fallible] annotation on the EyeColor predicate in the previous section tells it not to attempt to generate a second random color if the first one is rejected. As a result, the call to Eyes would also fail, and the generation process would restart.

The sampling process is driven by the driver loop:

```
[predicate]
NPC ?who:
   [CountAttempts ? 100]
   [Sample [Age ?who] [Gender ?who] [Ethnicity ?who]
             [Height ?who] [BMI ?who] [Weight ?who]
             [Eyes ?who] [Hair ?who] [Name ?who]]
   [PrintCharacter ?who]
   [NewLine] ?who/WriteVerbatim
[end]
```

of which, the calls to CountAttempts and Sample are the important parts. They form what is known in logic programming as a “failure-driven loop” [23]. The call to Sample runs the calls to Age through Name, each of which adds one feature to to the feature structure ?who. If any of those calls fails, then Sample fails immediately and the system backtracks to the CountAttempts call, which will retry the call to Sample up to 100 times.

Failure-driven loops are admittedly rather ugly, but are a common design pattern in logic programming.

5.1.6. Exceptions to rejection sampling

Not all features use the sample-then-test design pattern discussed above. There are two important exceptions.

First, features that are roots of Bayes net in Figure 1 are not statistically dependent on other features and so can simply be set by the user. They could be implemented with that pattern above, but there’s no particular need to do so.

The other features are the floating-point valued features. The reason being that the probability of any given floating-point number being randomly generated is on the order of 2^{-64}, which is nearly zero. As a result, it’s effectively impossible for the failure-driven loop to generate a character whose BMI is *exactly* some specific floating-point number. So if the user specifies a particular height or BMI, the system will simply accept it without trying to generate it through rejection sampling. That means that the statistical dependence of gender and ethnicity on height will be lost. That said, one can instead use a range query such as:

```
[NPC { bmi:?bmi}] [>= ?bmi 20] [< ?bmi 21]
```

to generate a character with a BMI in a given *range*. That will be correctly sampled.

6. Limitations

Although the system was quite serviceable for GMing a TTRPG, it has definite limitations.

6.1. False statistical independence

As discussed above, the system treats many things as being statistically independent that are not actually independent. For example, hair color and eye color. It wildly overrepresents certain combinations, such as black hair and green eyes that, while possible, are very rare. Also, most features are statistically dependent on age in real life, but most are treated here as independent of age.

6.2. Data granularity

Census data aggregates people into a small number of ethnic categories, much smaller than the number of semantically meaningful ethnic groups. While this is largely invisible for features such as hair/eye color or physical measurements, it is problematic for name generation.

For example, the “white” category includes both Irish and Russian ethnicities. Both of which have prenames and surnames that are generally associated with their ethnicities; the name Liam O’Reilly would be interpreted by most US players as more likely Irish than Russian, whereas Vladimir Ivonov would be read as more likely Russian. Unfortunately, none of that information is included in recent census data. As a result, the name generator is just as happy to generate names like Liam Ivanov and Vladimir O’Reilly as it is to generate the ethnically normative combinations. This is less of an issue for the “white” category, since the US has had enough intermarriage of many of the groups it comprises, that it’s less likely to be noticed or to offend.

It's more of a problem when generating characters for recent immigrant groups or members of the Native Americans population. For example, census data aggregates Pakinstani, Indian, Tibetan, Chinese, Japanese, Korean, Thai, Hawaiian and Indonesian people, to name but a few, in one category. As a result, the system *more likely* to generate names, such as “Kimoko Ghandi,” that combine prename and surname from distinct ethnic groups, than it is to generate prename and surname from the same group. In fact, it’s very *unlikely* to generate a prename of the same nationality as the surname.

This is not an algorithmic problem, but rather a data problem. If using this in a commercial digital game, one would likely want to give up on census data for these groups and hand-curate smaller lists of names for the different nationalities.

6.3. Rejection sampling

The main issue with rejection sampling is that its expected execution time is inversely proportional to the probability of the feature values specified by the user. If the user specifies no features (probability 1), then the first generated sample will always be accepted and the system is very efficient. If the user specifies gender (~50% probability for the selected gender in most cases), then it will generally only take one or two samples to find a solution. But if one specifies a very unlikely combination of features, say red hair and blue eyes, it might take many samples to generate one with those features. And as discussed above, if the user specifies a fixed value of a floating-point feature, it will effectively run forever. Fortunately, these use cases are very rare in table-top RPGs. And in any case, a GM is likely happy to wait a hundred milliseconds to generate a particularly unlikely character.

That said, if this system were to be used in a video game where execution time is critical, and if low probability queries were common, it might well be worthwhile to construct separate Bayes nets for common use cases. For example, one might have a separate set of statistics used to generate a boss versus a normal NPC. This is a case where the ability to define the system as a set of declarative rules conditioned on features is useful. One can simply add separate rules to detect these use cases and switch over appropriately.

7. Related work

While I have not found prior work on using Bayes nets to generate demographically accurate characters, this work nonetheless has many antecedents in the literature. Ryan [24] used census data as a name corpus, although he does not specify whether he used frequency information when choosing names or used a uniform distribution. *City of Gangsters* [25] used 1920 census data for the city of Chicago to generate NPC populations and names.[2]

Probabilistic methods have been heavily used for tile-based level generation. Somerville and Mateas used sampling of learned Bayes nets to produce levels for The Legend of Zelda [26]. Markov chains have also been very popular, for example the work of Snodgrass and S. Ontañón [27].

More recently, Abdelrahman et al. [28] have used Problog, another probabilistic logic programming language, to generate game levels. Problog is based on annotated disjunctive clauses and is designed primarily for inference, that is, for estimating probabilities given evidence. However, it can be used in sampling mode, where it, like other systems, uses rejection sampling.

This disadvantage of Problog for this application is that it is a grounded language, meaning it expands all rules into copies in which all variables are replaced with all possible values for those variables. Although grounders can be very clever in avoiding generating unnecessary ground instances, this still is an expensive process in both time and memory. For example, the grounded form of the name predicates used here would require around 30,000 ground instances, provided one re-coded the problem to make it easier to ground. A direct grounding of the name predicates as written here would have generated 48M ground instances, requiring at least 40 bytes per instance, or around 2GB of RAM. So far as I can determine from the Problog documentation, the floating-point features such as height, weight, and BMI would be impossible to implement directly in Problog, although they could easily be implemented in discretized form (e.g. short, medium, tall height).

Problog is a valuable tool for Abdelrahman's level-generation application. For this particular application, however, it is both overkill and insufficient. Lifted rules (i.e. ungrounded rules) in a more classical logic programming language are preferable.

A common question is why not use a large language model for this problem. While large language models are very useful, they are not designed to sample from a distribution. That is, they are not designed to accurately reproduce a given statistical distribution. As such, they're not well-suited to this particular problem.

As an example, Hispanic people are significantly underrepresented in STEM fields in the US. They make up 20% of the US population but only 8% of the STEM workforce. However, when Microsoft Copilot was given the prompt "generate 100 random names of people who could be scientists", it generated 33% Hispanic surnames. When given the prompt "generate 100 random names of people who could be scientists *in the US*", it generated 30% Hispanic surnames. In both cases, it significantly overrepresented the Hispanic population. This is a good thing if one intends the game to be aspirational: to present a world in which underrepresented groups are more evenly represented, or at least not underrepresented. It's counter-productive if the game seeks to draw attention to and problematize that underrepresentation. It's possible that with fine-tuning a model could better approximate a given body of statistics, but that would be slower and involve more programmer effort than simply sampling from tables of data directly.

8. Conclusion

When procedurally generating characters, it is highly desirable for them to accurately represent their worlds. For imaginary worlds, this can be largely a matter of fiat. However, it can be surprisingly difficult for games set in the real world. While not algorithmically difficult, it is limited by the availability of detailed demographic data. Indeed, part of the motivation for setting *City of*

[2] Matt Viglione, personal communication.

Gangsters [25] in the 1920s was the availability of more detailed census data for that period than for the modern era.[3]

The system presented here does as good a job as possible, given available open-source data. It has very definite limitations, as discussed above, but is quite sufficient for a human-in-the-loop application such as a gamemaster's tool. However, as discussed above, use in a digital game would likely require more care.

The system could easily be re-implemented in any language. However, the choice of a stochastic logic programming language with feature structures is very convenient. The whole system is on the order of 150 lines of code, not including the demographic data, which is 37K lines. Complex queries involving multiple constraints are built-in to the language itself and so are included for free in character generation once the basic sampler has been implemented. If reimplementing the system for a digital game, it would be attractive to implement it as a set of rules over feature structures, even if the Step language itself was not used.

Acknowledgements

I would like to thank Sam Hill for his comments and suggestions during the development of the system, and the reviewers for their helpful comments on the draft of the paper. Finally, I would like to thank the members of the Office of Special Amusements for their participation in the *DELTA GREEN* campaign.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] L. De Moura and N. Bjørner, "Z3: An efficient SMT Solver," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008. doi: 10.1007/978-3-540-78800-3_24.

[2] I. Horswill, "CatSAT: A Practical, Embedded, SAT Language for Runtime PCG," in *AIIDE-18*, AAAI Press, 2018.

[3] T. Adams and Z. Adams, *Slaves to Armok: God of Blood Chapter II: Dwarf Fortress.* (2006). Bay 12 Games.

[4] D. Detweiler, C. Gunning, S. Ivey, and G. Stolze, *DELTA GREEN: Agent's Handbook.* Chelsea, Alabama: Arc Dream Publishing, 2016.

[5] Hite, *The Fall of DELTA GREEN.* Pelgrane Press, 2018.

[6] D. Hammett, *The Maltese Falcon.* Alfred A. Knopf, 1930.

[7] J. Huston, *The Maltese Falcon*, (1941).

[8] "The Bohemian Club." [Online]. Available: https://www.bohemianclub.org/

[9] "Catholic Charities USA." Accessed: Aug. 28, 2025. [Online]. Available: www.catholiccharitiesusa.org

[10] "Urban Alchemy." Accessed: Aug. 28, 2025. [Online]. Available: www.urban-alchemy.us

[11] US Census Bureau, "Census Surname Data," Census Surname Data. Accessed: Aug. 28, 2025. [Online]. Available: https://www.census.gov/topics/population/genealogy/data.html

[12] K. Tzioumis, "Demographic aspects of first names," *Scientific Data*, vol. 5, no. 5, Mar. 2018, doi: 10.1038/sdata.2018.25.

[13] Centers for Disease Control, "NHANES Questionnaires, Datasets, and Related Documentation," NHANES Questionnaires, Datasets, and Related Documentation. Accessed: Aug. 28, 2025. [Online]. Available: https://wwwn.cdc.gov/nchs/nhanes/Default.aspx

[14] "Demographics," San Francisco Police Department, 2025. Accessed: Aug. 28, 2025. [Online]. Available: https://www.sanfranciscopolice.org/your-sfpd/published-reports/demographics

[15] R. Fry, B. Kennedy, and C. Funk, "STEM Jobs See Uneven Progress in Increasing Gender,

[3] Matt Viglione, personal communication.

Racial and Ethnic Diversity," Pew Research Center, Apr. 2021. Accessed: Aug. 28, 2025. [Online]. Available: https://www.pewresearch.org/science/2021/04/01/stem-jobs-see-uneven-progress-in-increasing-gender-racial-and-ethnic-diversity/

[16] D. Sjostedt, "Ask The Standard: Who is homeless in San Francisco?," *The San Francisco Standard*, Jun. 06, 2023. Accessed: Aug. 28, 2025. [Online]. Available: https://sfstandard.com/2023/06/06/san-francisco-homeless-population-demographics/

[17] Binder, Tanya, L. Stewart, A. Climent, and J. Hancock, "Why Is Racial Diversity So Low on U.S. Tech Boards?," Spencer Stuart, Jul. 2021. Accessed: Aug. 28, 2025. [Online]. Available: https://www.spencerstuart.com/leadership-matters/2021/july/why-is-racial-diversity-so-low-on-us-tech-boards

[18] D. Bell and D. Belt, "Gender Diversity Survey – 2020 Proxy Season Results," Fenwick & West LLP, Mar. 2021. Accessed: Aug. 28, 2025. [Online]. Available: https://www.fenwick.com/insights/publications/gender-diversity-survey-2020-proxy-season-results

[19] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2009.

[20] I. Horswill, "Step: A Highly Expressive Text Generation Language," in *Artificial Intelligence and Interactive Entertainment (AIIDE-22)*, Pomona, CA, 2022.

[21] D. H. D. Warren, L. M. Pereira, and F. Pereira, "PROLOG - The Language and its implementation compared with LISP," in *Symposium on AI and Programming Languages*, ACM, 1977, pp. 109–115. doi: 10.1145/800228.806939.

[22] S. Shieber, H. Uszkoreit, Pereira, J. Robinson, and M. Tyson, "The Formalism and Implementation of PATR-II," in *Research on Interactive Acquisition and Use of Knowledge (Final report of SRI Project 1894)*, Menlo Park, CA: SRI International.

[23] W. F. Clocksin and C. S. Mellish, *Programming in Prolog: Using the ISO Standard.* New York, NY: Springer, 2003.

[24] J. Ryan, "Curating Simulated Storyworlds," University of California Santa Crus, 2018.

[25] SomaSim, *City of Gangsters.* (2021). Chicago.

[26] A. Somerville and M. Mateas, "Sampling Hyrule: Multi-Technique Probabilistic Level Generation for Action Role Playing Games," in *2015 AIIDE Workshop on Experimental AI in Games*, AAAI Press, 2015. doi: https://doi.org/10.1609/aiide.v11i3.12817.

[27] S. Snodgrass and S. Ontañón, "Controllable Procedural Content Generation vis Constrained Multi-Dimensional Markov Chain Sampling," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*, 2016.

[28] M. Abdelrahman, C. Martens, S. Holtzen, C. Harteveld, and S. Marsella, "Probabilistic Logic Programming Semantics for Procedural Content Generation," in *Proceedings of the Nineteenth Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2023)*,