37

Alibi Generation

Fooling All the Players All the Time

Ben Sunshine-Hill

37.1 Introduction
37.2 Your Ideal World
37.3 Initial Generation
37.4 Identifying When an Alibi Is Necessary
37.5 Generating the Alibi
37.6 Options for Alibi Generation
37.7 Maintaining and Deleting Characters
37.8 Conclusion

37.1 Introduction

The sandbox just keeps growing. Recent open-world games let players roam freely over tens or hundreds of square miles. And it's not just raw size: the bar for density and variety of content keeps being raised. Of course, the bar keeps being raised! We don't just want to give players a *space*; we want to give them a *world*. A living, reacting, interacting world, where a player's actions can have far-reaching consequences, and where a new plot lies in wait behind every doorway, behind the eyes of every individual character going about his virtual life. Where things are happening, and the player—if she dares—can become a part of them.

That's the ideal, anyway. But while a combination of large content teams, efficient level building techniques, and careful use of procedural content generation have given us larger and more varied worlds, we're still trying to figure out how to populate and simulate them. We tend to spawn randomly generated characters in the area around the player (the "simulation bubble"), giving the impression of a fully populated world. But it's tricky to keep up the illusion. If the player blocks a just-spawned NPC's path, how will he replan? If the player picks his pockets, what will she find? If the player follows him, where will he turn out to be going? With a careless approach to character generation, the answers are likely to be "back the other way," "`rand(0,10)` money," and "endlessly wandering"… and that's just not good enough anymore. The bar's up *here* now.

Alibi generation is a more considered, consistent kind of character generation. NPCs are still generated randomly to fill the simulation bubble, but as necessary, they are given *alibis*: filled-out backstories, goals, and states of being; everything necessary to play the part of a living, breathing character in a living, breathing world. Done properly, it is *impossible* for the player to determine whether an NPC has always been around or whether they were just given an alibi a couple of seconds ago.

37.1.1 Background

The underpinnings of alibi generation rely heavily on probability theory. To get the most out of this article, you should be familiar with terminology such as *joint distribution*, with notation such as $\mathrm{E}[PQ]-\mathrm{E}[P]\mathrm{E}[Q]$ and $\mathrm{P}(\neg A|B=1)$, and with tools such as Bayesian networks. For an introduction to this field, I highly recommend the Khan Academy series of videos on probability.

37.2 Your Ideal World

Suppose your game was meant to execute on the ultimate hardware: An infinitely fast processor with unlimited memory. There would be no reason for the simulation bubble, of course; it is much more straightforward to simulate all the characters in the world, all the time. And no reason to start them *in media res*, either: Just start everyone at home and run them for a few in-game hours as part of the first frame of simulation.

That's not to say that you'd have no need for procedural generation. Since your content team *wouldn't* be infinite, you'd still need to use it to create your world's populace in the first place. But rather than generating them as a current state, you'd have the system generate the information *about* them: Where they live, where they work, how they dress, what sorts of activities they engage in; their AI rules would take care of the rest. As the content creator, you'd create the following items:

1. A list of immutable "fact" features about a character. This list might include "where the character lives" and "how tall the character is." Some of these things might be immediately visible ("what color the character's hair is"), but most will be only indirectly observable ("what food the character prefers").
2. A list of possible values for each fact and how likely each of those values is. (In the case of features like "where the character lives," this will probably be generated from information in the world.)
3. A list of mutable state features about a character. Most of these will be immediately visible things like "what the character is currently doing," but some may be less visible, like "how hungry the character is."
4. Rules for how a character chooses actions, based on their immutable facts and their current state, and for how those actions affect the character's state.

You'd probably spend much of your time on the second item, deciding which facts were correlated with which other facts, and how likely each value was. In contrast, you wouldn't bother coming up with information like "How likely is it that the character is in a Mexican restaurant *right now*," because there'd be no need for it. Each character would

go about his life, and the interaction between his immutable facts, his mutable state, and his rules of behavior would lead to a mutable state which either included "in a Mexican restaurant" or didn't. Put differently, a character's immutable facts would be the cause, and the character's mutable state would be the effect. A character whose facts included "prefers Mexican food" would be more likely to be in a Mexican restaurant at any given moment than a character whose facts didn't, without you needing to specifically hard-code that in.

Alibi generation turns this around. It starts by generating only the visible information about a character. Later, if necessary, it treats the visible information as the cause, and generates the invisible information (the alibi) as an effect. For instance, if a character is generated with visible information including "in a Mexican restaurant," and later the player gets into a conversation with the character and an alibi is needed, the alibi generated for the character would be more likely than average to include "prefers Mexican food."

But—and here's where it gets a bit tricky—how often *should* you generate people in Mexican restaurants, and exactly how much more likely *is* it that someone in a Mexican restaurant prefers Mexican food? The key feature that makes alibi generation useful is that you, as the content creator, are still responsible *only for coming up with the four original items*. All of the dependent information used to generate characters *in media res,* and to give them alibis later, is generated in a preprocessing step based on that original information.

Let's go into this a little more. There are three parts to the runtime component of alibi generation: Generating initial, alibi-less characters, identifying when an alibi is necessary, and generating an alibi for a character. We'll explore each of these in turn.

37.2.1 Heisenburgh

It's difficult to go too far into this "ideal world" stuff without an example application, so we'll present ours: Heisenburgh. Heisenburgh is a simple simulation of pedestrians in a city, about the size and population of Manhattan. (Its street layout is adapted from a region of Basra, Iraq.) There are thousands of potential points of interest in the city, such as homes, office buildings, restaurants, banks, and theaters. In the full simulation, a character has a home and a workplace. They pick a next goal (such as "go to work" or "eat at a nice restaurant") based on their current location, walk there via the shortest path, then stay there for a random amount of time dependent on the type of goal. Goals can be round-trip errands or one-way journeys. For some types of goals (e.g., getting a hot dog), characters pick the closest destination of its type; for others (e.g., visiting a friend) they pick a particular one in the world. Afterward, they either return to their previous location, or pick a new goal, depending on the goal type.

Heisenburgh's mechanics are simple, yet simulating millions of its characters in real time is beyond the capabilities of current-generation video game hardware.

We broke the world into sectors of about eight city blocks each and simulated only the sectors visible to the player, using alibi generation to manage characters. Each sector was connected to neighboring sectors by "portals," which tended to be in the middle of blocks. We precomputed the shortest path through a sector from any portal to any other portal, from any building within the sector to any other building within the same sector, and between any portal and any building in the sector. These precomputed paths were known as "path segments," and a character got from place to place by following a sequence of these path segments. A character's initial information consisted of their current path segment

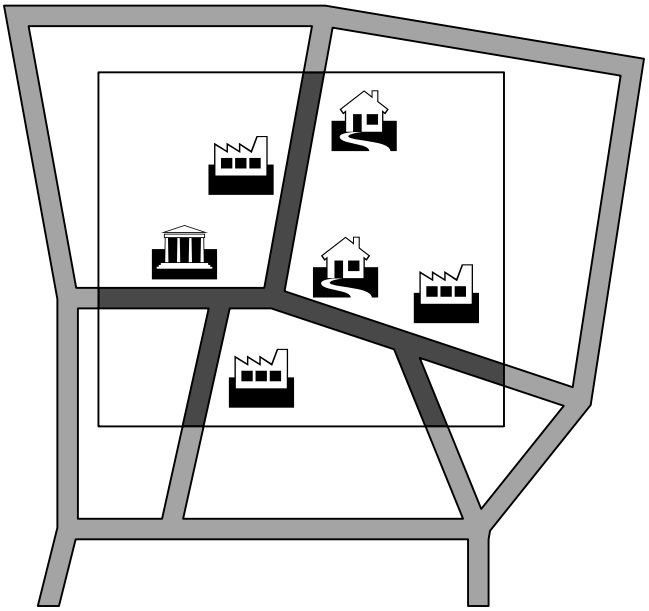

Figure 37.1

A single sector (rectangular outline) in Heisenburgh. This sector contains six buildings and five portals, and results in 110 path segments: 20 from portal to portal, 30 from portal to building, 30 from building to portal, and 30 from building to building.

(including direction); their alibi consisted of the building they had previously left and the building they were heading towards, as well as their reason for the trip (Figure 37.1).

37.3 Initial Generation

Initial generation of characters, of course, is something we already know how to do, because we've been doing it all along. It needs to be really quick to do, because we'll be doing it for every character, including unimportant background characters. From the point of view of alibi generation, the most important aspect of initial generation is that it has to be consistent with the distribution of characters in the full simulation. If you generate a character walking down an alley toward a dead end and there aren't any destinations at the end of the alley (or other AI reasons why a character would go there), it'll be impossible to generate an alibi for that character. More subtly, if there's only one destination in that direction, every character walking down it will get an alibi which includes going to that destination, so the alley had better be exactly as populated as that destination is popular.

37.3.1 Population Size

The first part of initial generation we need to look at is deciding how *many* characters to generate in a particular place (say, in a bookstore when the player first walks in), and how often to generate new ones (new customers walking through the door). The most important aspect of this is that the two processes match up. If the entry rate is too high,

the bookstore will start out deserted and quickly fill up; if it's too low, the bookstore will become deserted over time.

For Heisenburgh, after breaking the world into sectors, we identified all possible paths through each sector, each end of a path being either a sector boundary or a building entrance. We calculated and stored the average population of each path. (Note that a "path" may be several blocks long and a single block of sidewalk will have many "paths" which go down it.)

To generate the initial population when a player moves into view of a new sector, we used the *Poisson distribution*. This is a random distribution over populations, which takes the average population as its only parameter and assumes that characters' positions are independent of each other. Sampling from the Poisson distribution can be done using a simple iterative algorithm [Knuth 69].

To generate new characters entering a sector at the beginning of a sector path when the adjacent sector was not visible, we used the *exponential distribution* to generate the "time until next arrival." This distribution takes the average entry rate as its only parameter. If the average population of a path segment is p and the time to travel all the way along this path segment is t, the average entry rate is p/t. After sampling a time until next arrival, a delayed event is set for that future time; when the event fires, a new character enters that path segment, and a new next-arrival-time is generated. A global priority queue manages all arrival times. When two adjacent sectors are both visible, we didn't generate new entries from one into the other, as that would result in characters visibly popping into being. Instead, characters got into the sector the normal way: by previously being in the other sector.

The Poisson and exponential distributions are, mathematically, the "correct" distributions to use for these tasks, assuming simple independence and homogeneity properties. There's little reason to deviate from them. Moreover, since the distributions depend on only two numbers per path, this data can simply be precalculated and stored.

37.3.2 Position and Other Information

For each character starting on a path, it's necessary to define where they are on that path. That's done simply by sampling a random position on the path, then offsetting by a small random amount to bootstrap collision avoidance. A character's appearance—face, clothing, etc.—is also randomly generated as a set of meshes, which are then merged.

37.4 Identifying When an Alibi Is Necessary

When deciding whether to generate an alibi for a given character, you need to strike a balance between realism and performance. Simulating a character may become more expensive once they have an alibi, and the alibi generation process itself may also be expensive. However, waiting too long to generate an alibi may make it impossible to find one which is consistent with everything the player has already observed about the character.

Ideally, you want to generate an alibi at some point before the alibi-less and alibi-ful behavior of the character could diverge. If a character has just started walking down a block with no destinations on it, there's little uncertainty about what their short-term behavior will be. It's only once they get to the end of the block and need to decide which

way to turn that they would draw on their hidden, internal state. Interaction with the player can also necessitate an early alibi.

For Heisenburgh, splitting character-traveled regions into paths through sectors worked in our favor here. Since a character's initial state usually included several blocks of walking, it wasn't necessary to generate an alibi for them the first time they turned a corner. It was only once they approached a sector boundary—and the end of their path through the sector—that an alibi was needed. Moreover, if a character came into being almost at the end of their sector path, we did not immediately generate an alibi; instead, we picked a random path through the next sector for them, relying on the player's inability to reason about their path given the tiny portion of it they were able to see.

37.5 Generating the Alibi

Successfully generating an alibi starts with the groundwork laid out by the initial generation. Like initial generation, the goal is to come up with a set of data that is consistent with certain prior conditions. Storing the entire conditional probability table for alibi generation, however, would be utterly infeasible—it's much too large. The real trick, then, is finding a compact representation for the alibi distribution and a way to sample from it. There are a few options for this; we'll present one here, and others in Section 37.6.

Enter the Metropolis–Hastings algorithm. This is a technique for generating random samples from a distribution which is difficult to sample—or even compute—directly. It is a *random walk* technique, which starts from some initial condition and then incrementally modifies that condition over many iterations. Although the initial condition is not random, over time the distribution of the current state converges to the desired probability distribution. In conventional Metropolis–Hastings, the goal is usually to generate a large number of samples from the same distribution; the first few hundred iterations are known as the *burn-in*, and are discarded because they are likely to be correlated with the initial state. For alibi generation, however, we spend all of our iterations on burn-in; after that, we take the last state as the alibi.

The best part of Metropolis–Hastings is that we don't actually need to compute conditional probabilities. Given initial data D, to sample an alibi $A|D$, we need to compare the *relative* probability of two alibis, $\mathrm{P}(A_1 \mid D) / \mathrm{P}(A_2 \mid D)$. By Bayes' Rule, this is the same as $\mathrm{P}(A_1, D) / \mathrm{P}(A_2, D)$. So we need to provide a function $f(A, D) \propto \mathrm{P}(A_1, D)$. We also need to provide a transition function which randomly chooses a "nearby" candidate alibi A_2 given A_1 as input, and a function $q(A_1 \to A_2)$ which tells us the probability of that transition – the probability of choosing that candidate alibi A_2 when moving from A_1.

During each iteration, we perturb A_1 into A_2 using our random transition function. Then we compute the acceptance probability, $a = \min\left(\frac{f(A_2, D)}{f(A_1, D)} \frac{q(A_2 \to A_1)}{q(A_1 \to A_2)}, 1\right)$. We accept A_2 with probability a; if we do not accept it, our current alibi remains as A_1.

For Heisenburgh, we analytically generated a small set of tables that could be used to generate $f(A, D)$ [Sunshine-Hill 11]; these consisted of conditional probability tables and tables of per-destination probability distributions (like entry rate). Remember that an alibi consisted of source and destination buildings; to perturb an alibi, we perturbed the source building, then the destination building. Each building in the world held a table of nearby buildings (including the building itself), and perturbed source and destination

were sampled uniformly from the current source and destination buildings' tables. To avoid a division by zero, we removed unidirectional transitions from the tables, which often made them differently sized. That meant that $q(A_1 \rightarrow A_2)$ was the reciprocal of the product of the source and destination table sizes.

Alibi generation took place in two steps: First, determining *where* a character was going, and second, determining *why*. The "where" consisted of both their source and destination. The "why" consisted of whether their current destination is a one-way trip, going to a round-trip errand, or returning from a round-trip errand, and what their goal was. The second step was important because it helped determine what the character's *next* action would be and because it determined whether the source or destination had any particular significance to them. If the player were to see a character going home to a particular building, that character had better keep going home to that same building in the future.

37.6 Options for Alibi Generation

The most difficult part of alibi generation is designing and building the data which is used for the sampling. I've shown you the gold standard—the Metropolis–Hastings sampling from closed-form probability distributions—but there are a few options.

37.6.1 Exact Calculation

For Heisenburgh, we built a representation of the alibi distribution which was provably identical to the stationary distribution of the "ideal world" (the full simulation). I won't lie to you: Doing this is *very, very difficult*. Even for the simple character AI we used, solving the full set of equations took weeks and invoked deeply obscure areas of stochastic process theory. Doing things this way is great if you can pull it off, since its results are dependably correct; but if your AI rules are significantly complicated, it's just not feasible.

For more information on Metropolis–Hastings, I recommend [Chib and Greenberg 95].

37.6.2 Canned Alibis

A really, really simple approach to alibi generation is to store, for each unique set of initial conditions, a list of prerecorded alibis. For instance, you might have a list of alibis specifically for men who are wearing suits, carrying paper bags, and walking east on a particular block. A randomly (or sequentially) chosen alibi from the list is applied whenever one is needed.

The benefit of this approach, of course, is simplicity. There's no need for elaborate probability calculations, and it's by far the fastest way to "generate" an alibi. It's a one-size-fits-all solution. The drawback is the need to strike a compromise between space requirements and variety: The more alibis per initial condition set, the larger the space requirement; but the fewer alibis, the less variety that alibi generation can create.

The space requirement is dependent on both the world size and the variety of initial conditions. The former affects both disk space and RAM space; the latter affects only RAM space, since alibis will only need to be generated for characters who were created near the player's current position, so the lists can be paged in and out. It's especially important, therefore, to limit the dimensionality of initial conditions, at least those which delineate alibi lists. For instance, in the above example you might choose not to have separate lists for carrying paper bags versus not carrying paper bags.

The recording process is simple: Have an option to simulate the entire world fully populated, run it for a while, and take snapshots of people in different initial conditions. It's useful to disable graphics, sound, and any other subsystems with no bearing on the simulation. If your characters all start at deterministic locations, remember to give them time to wander for a while before you start sampling. Additionally, don't sample too frequently, or alibis for contiguous paths may end up overly correlated. This process can be informally parallelized; run it on several machines overnight and combine their alibi lists.

37.6.3 Hybrid Generation

There's a potential middle path for people who want variety and a large space of initial conditions, but still want to use recording to generate alibi lists. The basic idea is to start with canned alibis, then add variety to some aspects of the alibi—but not others—using random perturbation.

The most likely way to employ this strategy is to perturb the character's current destination, but only among destinations of the same type. As in the Metropolis–Hastings method, each building has a transition table of nearby buildings, but now it is only among buildings of the same type. And as in the canned alibis method, each set of initial data has a table of alibis to choose from. First, a canned alibi is chosen; then, that alibi's destination's transition table is used to move the alibi destination around. Only a few burn-in iterations, or even just one, need to be run, and $f(A_2,D)/f(A_1,D) \sim 1$ (as long as the current path is along the shortest path to A_2; otherwise, it is 0), so the acceptance probability is only the ratio of the table sizes. It's perhaps a stretch to even call this Metropolis–Hastings; it's basically just a random walk over the space of possible destinations.

37.7 Maintaining and Deleting Characters

An important question to ask about alibi generation, particularly if we're looking at it as an improvement to the "simulation bubble" approach, is when to *remove* characters. As a first approach, we can simply delete characters when their current sector becomes invisible. That's akin to the simulation bubble, but it creates obvious potential problems. Even if the player hasn't been watching a character for long, she still might notice if she goes around a corner, comes back, and he's gone. We can make this a little less likely by depopulating invisible sectors only if the player is more than a specified distance away or only once they've been invisible for a certain period of time; this way, the player would not be assured of being able to find the same characters even in the full simulation. In the case of interior locations where characters may remain for some time, a longer invisibility cutoff time should be used.

It's a good idea to extend the lifetimes of characters with alibis. A character with an alibi often has one because they were specifically important to the character in some way, so they may be memorable to the character for longer. This can create a situation where an invisible sector has a couple of characters with alibis in it, but it is otherwise unpopulated. When you initially populate a sector which already has characters in it—alibi-ful or otherwise—you should subtract this population from the "average population" parameter used for the Poisson distribution, but not from the parameter used for the exponential distribution.

As an alternative to these strategies, the LOD Trader (discussed in the chapter "Phenomenal AI Level-of-Detail Control with the LOD Trader") is an ideal tool for

managing alibis. Whether a character has an alibi or not can be treated as an LOD feature, with the no-alibi level given a ULTB penalty. Likewise, as discussed in the LOD Trader, rather than deleting characters based on thresholded distance or time, existence can be treated as a feature, with the transition to nonexistence given US and FD penalties. Creating initial characters is still done when a sector becomes visible, and as mentioned before, when generating the initial population, subtract any current population from the average population parameter.

37.8 Conclusion

Alibi generation, in its most fundamental form, is a simple idea: Generate details lazily. That's "lazy" in the computer science sense: When first required, not before, and not after. There's a variety of ways to do this, from the simple and informal (canned alibis) to the complex and theoretically precise (Metropolis–Hastings). The most important objective, regardless of approach, is to keep the concept of the "ideal game world" in mind and design your alibi generation system so as to seamlessly replicate the experience of that ideal game world.

References

[Chib and Greenberg 95] S. Chib and E. Greenberg. "Understanding the Metropolis-Hastings algorithm." *The American Statistician*, vol. 49, no. 4 (Nov. 1995). pp. 327–335, 1995. Available online (http://elsa.berkeley.edu/pub/reprints/misc/understanding.pdf).

[Sunshine-Hill 11] B. Sunshine-Hill. Perceptually Driven Simulation (Doctoral dissertation), 2011. Available online (http://repository.upenn.edu/edissertations/435).