The Intelligence Engine

Sunday, July 7, 2013

Decision Modeling and Optimization in Game Design, Part 1: Introduction

This article is the first in an ongoing series on the application of decision modeling and optimization techniques to game design problems. The full list of articles includes:

- *Part 1: Introduction (Blogspot) (Gamasutra)*
- *Part 2: Optimization Basics and Unrolling a Simulation (Blogspot) (Gamasutra)*
- *Part 3: Allocation and Facility Location Problems (Blogspot) (Gamasutra)*
- *Part 4: Competitive Balancing Problems (Blogspot) (Gamasutra)*
- *Part 5: Class Assignment Problems (Blogspot) (Gamasutra)*
- *Part 6: Parametric Design Techniques (Blogspot) (Gamasutra)*
- *Part 7: Production Queues (Blogspot) (Gamasutra)*
- *Part 8: Graph Search (Blogspot) (Gamasutra)*
- *Part 9: Modular Level Design (Blogspot) (Gamasutra)*

We're Searching, Not Iterating

Most of game design is a process of *search*. When we design, we are evaluating many different possible design configurations to solve a given design problem, whether it be the way the rooms in a dungeon are connected, the set of features and capabilities that different types of game agents will possess, the specific "magic numbers" that govern unit effectiveness in a combat system, or even the combination of features our game will include in the first place.

Just as an AI-driven character will use a pathfinding system in a game to navigate through the game world, design involves navigating through a very-high-level space of possible configurations by taking some initial configuration and iteratively modifying it. We look carefully at the state of some aspect of our design - whether it be our combat system, one of the parts of our game world, a technology tree in a strategy game, or what have you - and attempt to find a way to improve it by changing that configuration.

Designers like to use the term "iteration" to describe this process, but "search" would be a more appropriate description. The truth is that when we "iterate" on a design, we're experimenting with a game in development. We are making educated guesses about small sets of modifications that will change the current design configuration into a new design configuration that we believe will better meet our design criteria.

These "iterations" don't resemble the generally linear changes that typically occur in "iterations" of computer code; they much more closely resemble a search through a maze, with lots of sharp turns and occasional backtracking. They often move us forward closer to the goal, but many times it's unclear whether they've improved the game or not, and we sometimes discover that design changes we assumed would improve the game have unforeseen flaws and we need to back them out or try again.

Game design is an incredibly difficult discipline. Design is like a dark room full of sharp objects, extraordinarily difficult to navigate safely once we stray from the beaten path. There are nearly always some painful injuries along the way, especially if we move too quickly. And we have relatively few tools to light up that dark room, and few well-defined and disciplined techniques for carrying out this process of design search.

This dark room is is the reason we "iterate" -- we don't always know what the ramifications of our design decisions will be until we try them. In other words, we are ***searching*** (Will Wright even directly referred to it as "searching the solution space" in his 2004 GDC Talk).

As a result, design is very often a productivity bottleneck, a major source of defects, and the biggest source of risk in game development. Countless teams have found themselves hamstrung by ill-conceived design decisions, creative thrashing, feature creep, misperception of the target market, or other design problems that resulted in product quality problems.

Given all of the dangers involved in experimenting with design, it's no wonder that so many publishers and large developers are so risk-averse, preferring to hew closely to established and well-explored genres, licenses, and genre conventions rather than embracing the well-known risks of design innovation in return for relatively unknown payoffs. Exploring the dark room is just too risky.

We would like to find ways to change that attitude. Rather than simply *avoiding* innovation, it would be better to find ways to improve our design skills and extend our capabilities, and build power tools to make design innovation safer and more efficient.

This Series

This article is the first in a series that will introduce *decision modeling*, a set of tools for decomposing decisions into formal models that can then be searched to find the most desirable output.

Decision modeling and optimization are frequently used in management, finance, advanced project planning, and many other fields to improve the decision-making process and solve difficult decision problems and optimization problems by searching through the possible alternatives much more quickly than humans can do by hand.

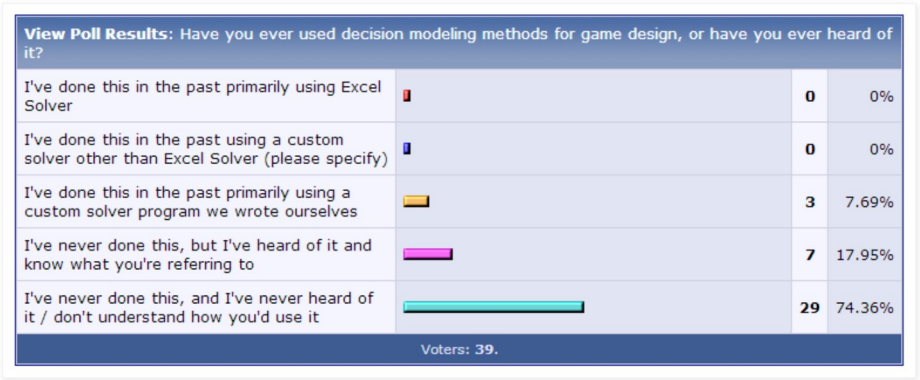
Blog Archive

► 2015 (5)
► 2014 (3)
▼ 2013 (9)
► December (1)
► September (2)
► August (2)
▼ July (4)
Decision Modeling and Optimization in Game Design,...
Decision Modeling and Optimization in Game Design,...
Decision Modeling and Optimization in Game Design,...
Decision Modeling and Optimization in Game Design,...

Contributors

- Paul Tozour
- Sensai Pose
- Unknown

Despite all of the potential benefits, decision modeling and optimization seem to be relatively unknown among designers in the game industry. A recent survey of professional designers on a popular developer forum indicated that only 25% respondents had even heard of decision modeling, and only 8% had used it in practice. A similar web-based survey passed directly to designers via Facebook had nearly identical results with a similar number of respondents.



View Poll Results: Have you ever used decision modeling methods for game design, or have you ever heard of it?

I've done this in the past primarily using Excel Solver	0	0%
I've done this in the past using a custom solver other than Excel Solver (please specify)	0	0%
I've done this in the past primarily using a custom solver program we wrote ourselves	3	7.69%
I've never done this, but I've heard of it and know what you're referring to	7	17.95%
I've never done this, and I've never heard of it / don't understand how you'd use it	29	74.36%

Voters: 39.

If used properly, decision modeling can significantly enhance many aspects of the design process:

- It can help you optimize the configuration of specific design systems or the optimal values of your game's parameters.
- It can help shed light on decisions as to what combination of features to include in your game.
- It can help you model the decisions a player might make, particularly in terms of helping you to identify dominant strategies or ways that players might "game the system."

In this series, we'll provide examples of all three of these usage categories.

A Definition

So what is decision modeling?

Simply put:

> **Decision modeling is the process of simulating a decision and then automating the search for its solution.**

We start by defining some sort of decision, attempt to pick out all the relevant factors that go into that decision, build those into a model that accurately represents the decision, and specify a set of input variables and a single output variable. Then we *search* for the optimal values of a set of decision variables (or input variables) that produces the best possible output.

If all goes well, we should be able to search through a much larger number of possible solutions than we could do by hand or with our imaginations. Although we can't apply it to everything, for the problems where it's appropriate, we can often get better results, get results faster, and in some cases, we can even solve problems that simply *can't* be solved any other way.

Along the way, we also specify a set of one or more *constraints* that act as boundaries to ensure that our model is valid. These constraints can limit the range or the type of our input variables or any aspects of our model.

Why Build Models?

Have you ever found yourself playing *Sid Meier's Civilization* and found yourself wondering, "Hey, wait a minute – what's the *right* way to start off my city? Should I build a Monument first, then a Granary? Or should the Granary come first? Or maybe the Temple first, then a Granary? What's the *best* decision? Is there even a way to answer that question?"

Also consider combat mechanics in a real-time strategy game. Balancing the parameters of multiple units in RTS games is a notoriously challenging problem. What if we had a system that could allow us to speed up the balancing problem by answering questions about our game's combat balancing without having to playtest every single time? What if we could ask the system questions like "How many Swordsmen does it take to defeat two Pikemen and three Archers?" Or, "What's the cheapest combination of Archers and Catapults that can defeat an enemy Guard Tower?"

In fact, maybe we can!

If we can model these design problems in the right way, we might be able to use automated optimization tools to search through the possible answers to find the one that best meets our criteria, *without* having to play through the game thousands of times.

Here's an example of a similar kind of problem – an example that we'll solve in a future episode of this series.

Let's say we have a game called SuperTank. In SuperTank, we drive a gigantic sci-fi tank into battle against other SuperTanks. Before each battle, we get to pick the exact combination of weapons on our tank.

You have 100 credits to spend on your weapon loadout. Your SuperTank can carry 50 tons worth of weapons, and it also has 3 “critical slots” for use by special high-powered weapons.

You have the following five weapon types, and you can use as many of each weapon type as you like, or skip any weapons entirely:

	Damage	Weight	Cost	Critical Slots
Machine Gun	2	1	5	0
Rockets	8	3	12	0
MegaRockets	15	10	16	1
Laser	7	4	9	0
UltraLaser	20	16	18	1

Assume that you want your SuperTank to have the highest possible total Damage value (assume that this represents damage per second, so it properly represents damage applied over time regardless of how quickly the weapon fires). Also assume that all weapons have the same range, firing arc, accuracy, and rate of fire, so they’re identical in all ways except for the ones shown on the chart above.

Quick! How many machine guns, rockets, lasers, etc should you equip on your SuperTank? What combination of 0ne or more of each weapon gives us the most damage without exceeding our budgets for Weight, Cost, and Critical Slots?

See if you can solve it by hand, or with a calculator.

Can you do it?

If you try it, you'll quickly see that it’s a surprisingly difficult problem.

There’s probably a way to solve this with complicated math equations. But we’re designers, and math just isn’t our thing.

Also think about how the answer would change if the parameters were different. Would the answer change if our SuperTank could hold 60 tons instead of 50? What if instead of having 100 credits to spend, we had 110, or 90 – how would the optimal weapon loadout change? What if it had only 2 Critical Slots, or 4?

Now imagine that we had a system that could instantly calculate the highest-damage weapon loadout for any given set of (Weight, Cost, Critical Slots) parameters. Type in the weapon parameters from the table above, then type in the SuperTank's parameters (50 tons, 100 credits, 3 critical slots), and ***BOOM!***, we can see the best possible loadout.

Wouldn't that be awesome?

We could use this to instantly give us answers to all sorts of useful questions:

- How does the optimal loadout change as we modify the parameters for a SuperTank?
- How does the optimal loadout change as we modify any of the weapon parameters?
- How much maximum damage will a SuperTank do at any given (Weight, Cost, Critical Slots) setting?
- Are the four weapon parameters (Damage, Weight, Cost, Critical Slots) appropriate and properly balanced for each weapon?
- Are there any weapons that are overly powerful, and are used too frequently? If any weapon is so useful that the correct decision is *always* to use it, then using it is always the optimal decision, and there really is no meaningful decision there. In that case, we should probably either remove the weapon from the game or rebalance it so that there are some circumstances where the weapon is *not* useful.
- Are there any weapons that are underutilized, and are rarely or never used? Similar to the above, if any weapon is so useless that the correct decision is to *never* use it, then there’s no meaningful decision there. In that case, we should either remove the weapon from the game or rebalance it so that there are some circumstances where it makes sense to use the weapon.

All of these are very important design questions that any designer should want to know the answers to. Knowing them will be enormously helpful to us in balancing *SuperTank*.

In just a few paragraphs, we’ve described a problem that’s remarkably difficult for us to solve manually, but which is trivially solvable with tools already built into Microsoft Excel.

In a future episode, we'll actually build a decision model for this that can answer all of these questions.

You’ll be able to see clearly that you can set up a model like this in minutes can allow you to solve this otherwise ferociously difficult problem. With just a bit of work, we’ll create a power tool to let us quickly and safely explore the design space.

Roadmap

Throughout this series, we’ll illustrate quite a few more sophisticated examples, and we’ll provide reference spreadsheets so that you can do all of

these examples yourself with nothing more than a copy of Excel. Our examples will include, among others:

- A simple combat example for a strategy game
- A model for optimizing the coordinates of several wormhole teleporters in a space-based massively-multiplayer game relative to each other and a number of population hubs
- A model for determining what tax rate to use for a simplified model of a city in order to balance citizen happiness against tax income in a 4X strategy game such as *Sid Meier's Civilization*
- A model for choosing how to assign spells and traits to character classes in a massively-multiplayer game
- An optimization model for determining the optimal build order for a planetary colony in a '4X' strategy game similar to the classic *Master of Orion*
- An example of a team trying to pick the right combination of features to include in a game, and using a decision model to help them make the appropriate trade-offs

In general, this series will build up from simple examples of finding optimal player strategies in specific game subsystems and then progress toward using decision models to help optimize parameters for game systems and optimizing feature set combinations.

In each of these cases, we'll describe the problem and show how to model it in Excel and solve it using Excel's built-in Solver tool. In each case, you'll see that we can do it more easily, quickly, and robustly than you could likely do without using Solver or an equivalent tool. We'll also provide the spreadsheets for each example so that you can download it and try it for yourself, reproduce the results, and experiment with each of the models.

Also, remember that the underlying representation, whether it be a spreadsheet or a program in a high-level language, or something else – is *irrelevant*. The important thing isn't whether we do this in Excel & Solver, Java/C++/C#, or anything else, but the way we model the problem and try to solve it.

Why Use Decision Models?

Some readers may be incredulous at this point. Building decision models probably seems like a lot of work. Why go through all the effort when instead, we can use user testing in the form of focus group testing and beta testing?

Let us state up front, on the record, that decision modeling isn't applicable to every problem. Some problems are too complicated or too difficult to model with these techniques, and there are many aspects of the design (such as aesthetic considerations, entertainment value, and the "feel" of the game) that are difficult or impossible to model with numbers. And decision modeling certainly does *not* eliminate the need for group testing, beta testing, or doing your job by playing your own game continuously throughout development on a daily basis.

But having said all that, it should become clear by the end of this series that decision modeling and optimization methods also give us a unique and remarkably powerful set of tools. They can fully or partially solve many kinds of problems that can't reasonably be solved any other way, and they can help provide answers and insights to all sorts of design questions that would be difficult to answer otherwise.
As with any tool, it's up to the practitioner to decide when their use is appropriate.

There are any number of cases where decision models may be inappropriate or too unwieldy to be useful. But as you'll see in this series, it's also surprisingly useful, and the more we can design properly at the earliest stages and get the bugs out of our design decisions before we even get into the user testing stage, the more likely that we'll be able to design systems that are solid, entertaining, and bug-free.

Think about the tools available to a typical programmer. Programmers have a very difficult job, but they're also blessed with many tools to help them find bugs before they ever go into testing. They have compilers that constantly scream at them the moment they make a typo; they use defensive programming practices to expose software defects; they have code reviews to help them identify one another's defects or call out poor programming practices; and they have many profiling and static analysis tools to help them find all kinds of performance bugs and other defects.

But designers don't have any tools like that. Our job is arguably just as difficult, but we have no compiler to tell us when we've made a syntax error. We have no profiler, no debugging tools, and no static analysis tools. We have no way to do code reviews since we don't have any 'code.' We write specifications and design docs and that's about it; we can share design documents and feature specifications among the team and hope they give us good feedback, but for the most part we have to actually get it in the game before we can see if it works or not.

That makes design incredibly risky, time-consuming, and expensive.

And just as with programming, human error is a natural and inevitable part of the process, and we *need* as many high-quality tools as possible to protect ourselves and our projects.

We are a very long way from having design tools that will support designers' exploration of the design space at anywhere near the level the way that our compilers, debuggers, profilers, and static analysis tools support programmers' exploration of the engineering space. But we're beginning to see the rise of a few custom game solvers and design tools, including a recently-developed playability checker for a Cut the Rope variant called "Cut the Rope: Play Forever" (link); the abstract game design system Ludi, which generated the board game Yavalath (link); and my own Evolver automated game balancing assistant for the mobile game City Conquest (link).

Decision modeling can help us move a few more steps toward that level of support and begin to augment and extend designers' own intelligence with some automated tools. And given the choice of having the tools or not having them, why wouldn't we choose to have them?

It's Not About Spreadsheets – It's About Models

This series is written for designers -- and we mean *all* designers, whether they come from an artistic, programming, storytelling, or board gaming background. So we're going to keep it simple, and stick to these promises:

- ***No code***. We'll keep the articles 100% free of any code and will illustrate all of our examples in Microsoft Excel using the built-in "Solver" utility. However, it's important to note that this series is *not* about spreadsheets or Excel – it's about decision modeling and optimization. Every single thing we do in this series can be done just as easily (and sometimes more so) in any high-level programming language.
- ***No math*** (or at least, not anything complicated). We're going to keep this series mostly math-free, and we won't use anything other than basic arithmetic here: addition, subtraction, multiplication, division, and occasionally a square root. Greek letters will be strictly forbidden.
- ***No four-dimensional spreadsheets***; we'll stick to the two-dimensional kind.

If you're a designer, this series should give you all the tools you need create decision models yourself, with no need to try to write code or rely on programmers to write any code for you. If you're a programmer, this should give you a fairly straightforward guide toward programming your own decision models in any high-level programming language, so that you can then build decision models of your own, either from scratch or by building off of a template that already uses Solver and Excel.

These articles are intended to be nothing more than starting point, so that you can take the concepts presented here and choose whether to build them out in Excel, pick another optimization tool, or try to build a solver of your own in a high-level language. Spreadsheets are a good start, but these decision models are most likely to be useful as a springboard for richer and more sophisticated models that can be integrated with your game architecture.

Disclaimers

Before we get too far into the thick of decision modeling, a few disclaimers are in order. Decision modeling and optimization don't provide any kind of complete system for game design, and we won't be making any claims to that effect. It's helpful to view it as a *tool* to help with some aspects of

the design process, and like any tool, it has plenty of limitations.

Here are some of the limitations you should be aware of:

- **It can be easily misused.** Like any tool, decision models can be used inappropriately or incorrectly, and an incomplete or buggy decision model can lead you to incorrect conclusions. Just as with software, the larger your decision model becomes, the more likely it is to contain bugs. It's also very easy to misinterpret what a model is telling you or to build an incomplete model that doesn't accurately model the decision you're trying to make.
- **It's complicated (sometimes).** Some design problems are just too complex to be usefully modeled with these approaches. Many problems have too many moving parts or are too closely integrated with other aspects of the game to be usefully represented in a standalone Excel spreadsheet. In these cases, you're left to decide whether to model only part of the system (which may leave you with an invalid / inaccurate model), build a complete model integrated into the game itself (which could be a lot of work), or forgo decision modeling entirely.
- **Not everything can be modeled**. Decision models can't tell you whether something is fun, whether it's aesthetically pleasing, whether it "feels" right, or whether it presents the player with a usable and accessible interface. There's generally no way to represent these kinds of subjective and aesthetic concerns in a discrete model. This means that there are clear limits on where decision modeling can and should be used, and it will be much more useful for systems design and optimizing mechanics and dynamics rather than aesthetics.
- **It has limits**. All optimizers have their limits, including the Excel Solver that we will use, and it's entirely possible to create decision models that have valid solutions but are so complex that no optimization tool can find them. For large enough numbers of unconstrained inputs, the problem can grow beyond Solver's ability to search every possible combination of inputs, and instead it must rely on various optimization methods. As we'll see throughout this series, we can simplify the expression of our models to make them easier for Solver to handle, and the developer of Solver (Frontline) offers a more powerful solver for larger problems, but it's definitely possible to create models that Solver cannot solve.
- **It doesn't guarantee optimality**. On account of that, when we're dealing with complex models, we can't always be 100% sure that we've found the *optimal* decision. We sometimes have to settle for second best: we spend more time optimizing, or start from scratch and re-optimize, so that we can say that the solution we've found is either optimal or very close to optimal with a reasonable level of confidence.

Finally, and most importantly:

- **We have to make sure we model the right things**. Not all problems are important enough that they need this kind of effort, and we have to make sure we know our priorities and avoid getting overly-focused on optimizing irrelevant problems while ignoring other, bigger problems that might be much more important.

Broadly speaking, there are certain things that need to hold true for decision modeling to be useful. The decision in question has to be something we can encapsulate within some sort of discrete model, and map the result of the decision into a single value. In other words, we must be able to map a finite set of inputs through a decision model and onto a single output in such a way that either minimizing or maximizing the output value gives us a better decision.

In cases where there are subjective concerns that can't be encapsulated in the model – for example, aesthetic considerations or usability / playability concerns – we will need to either cleanly separate those out from the decision model, use decision modeling only as an initial pass, or just abandon the decision modeling approach entirely.

In order for us to model decisions in a spreadsheet, there's also a limit on how complex the model can be. If our game does something very complex, we may not be able to replicate that complexity in Excel. It's important to keep in mind, though, that this is only a constraint on the power of the models we can build in Excel, and *not* on decision models themselves. You can build vastly more powerful solvers in your own game engine than you can build in a separate spreadsheet, and I hope that this series inspires you to do exactly that.

On the other side of the coin, all of these limitations hardly make decision modeling useless. Even in a case where a problem is too complex for decision modeling to tune completely, it can still help you get many components of your design much closer to a correct configuration, and it can help you find and debug a number of basic problems early in development.

And even when a decision model can't find the optimal solution to a given problem, either because the problem is too complex or because it requires aesthetic concerns or other subjective human considerations, it can still help you narrow down the solution, helping you rule out dead ends and otherwise reducing the complexity of the problem.

Finally, even if you choose not use decision modeling and never attempt to optimize any spreadsheets or build your own solvers, an understanding of decision modeling can still help you by changing the way you frame and think about design decisions.

This series is an exploration. We will look at many examples of game design problems and explore ways to model and optimize them in ways that offer us powerful design tools. You may be skeptical, or you may feel more comfortable not using any optimization at all. But I hope you will bear with me as we explore and see where we end up by the end of the series.

Conclusion

In the end, we should want to design *correctly*.

Many design questions are subjective, with no "right" or "wrong" answers. But in some cases – many more than you might think – there undeniably *are*. And in those cases, we should want to know how to get the right answer, or at least understand how we would go about defining the "right" answer and searching for that solution if it exists.

Decision modeling and optimization are powerful tools that can help us accomplish exactly that in many cases. I believe these tools should be part of every designer's toolbox. With a little discipline, it should become clear that these tools have untapped potential to help us explore the dark room of game design more quickly and more safely, and we will show you how with many applications throughout the rest of this series.

-Paul Tozour

Part 2 in this series is now available here.

This article is the first in a series of 10-15 on Decision Modeling and Optimization in Game Design that will launch roughly once a week starting in July 2013. We welcome any questions, comments, and feedback, or any special requests for decision models to tackle.

The author would like to thank Robert Zubek and Jurie Horneman for their feedback on this article.

Posted by Paul Tozour at 7:30 AM

Labels: Part 1

6 comments:

Unknown August 25, 2013 at 9:54 AM

Wow, awesome. Very well and clearly written. Thank you very much.

Reply

j4ur14 November 19, 2014 at 9:45 AM

I've been gaming since early childhood. Now on my earlies 30 I've been wondering about game designing. Being as engineer I have formation on decision modelling and I have found this article quite interesting. I think I'll read the whole blog and maybe I'll try to make a game of my own. Best regards, Pablo L.

Reply

Replies

Paul Tozour November 19, 2014 at 9:46 AM

Glad you found it helpful!

j4ur14 November 19, 2014 at 2:29 PM

Since I know about Operations Research, this kind of language is pretty familiar to me. The example of the tank remembered me of some tower defense game that I played and managed to do some spreasheets for maximizing damage output.

I'd bet you are a teacher. If not, you should be, because you are very good explaining ideas.

Best regards,

Paul Tozour November 19, 2014 at 2:33 PM

Nope, I'm not, but I appreciate the compliment!

Reply

sylvia June 1, 2020 at 1:27 AM

Very useful post and I think it is rather easy to see from the other comments as well that this post is well written and useful.Keep up the good work
software testing companies
QA testing services
software testing and quality assurance services
Automation testing services

Reply

Enter your comment...

Comment as: Ahmad Al-Kashef ((Sign out

Publish Preview Notify me

Newer Post Home

Subscribe to: Post Comments (Atom)

Simple theme. Powered by Blogger.