The Intelligence Engine

Sunday, September 8, 2013

Decision Modeling and Optimization in Game Design, Part 7: Production Queues

This article is the seventh in a continuing series on the use of decision modeling and optimization techniques for game design. 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)*

The spreadsheet for this article can be downloaded here: link

Introduction

In the previous articles in this series, we introduced the concepts behind decision modeling and optimization and showed how they can be useful in framing and solving a surprising number of design decisions. We also discussed many of the limitations of these techniques.

In this post, we'll show how these techniques can be used to optimize an undeniably *hard* design problem. So, hang on to your hats: this example is going to be a beast.

Once we're done, though, it should be very clear how this approach can be useful for tackling some very difficult design problems that can't reasonably be solved any other way. Although it's a bit of work to set it up, the spreadsheet we end up with will serve as indisputable proof that decision models can answer some questions that are difficult or impossible to solve any other way.

Our example is based on a production queue for a colony in a classic "4X" strategy game. As designers, we want to know: what's the best order to build things in our colony?

This will give us all sorts of insights into whether there is a "dominant strategy" that allows players to most easily win the game, or whether there are problems with the balancing of our game's buildings or the game rules.

If we were to have some sort of system that allowed us to automatically see the best production queue for a given city or colony, that would be fantastically useful, wouldn't it? Then we could tinker with the costs and benefits of each building, or experiment with our game rules, and see how the optimal production queue changed.

In this installment, we're going to show you how to build exactly that kind of system.

Although this example is limited by a simplified game model and the limitations of Excel and Excel Solver, it should be clear to anyone with a programming background how you could easily extrapolate from this example to your own model in a high-level language such as C++, C#, or Java, and potentially integrate it with your own game as well.

Master of Orion

In 1993, a small studio in Texas released a brilliant classic "4X" strategy game called *Master of Orion*. Three years later, they followed it up with another classic, *Master of Orion II: Battle at Antares*. These games were two of the initial forerunners in the space-based "4X" strategy genre, and laid the groundwork for later franchises such as Stardock's successful *Galactic Civilizations* series. Our example in this case will use colony in a game broadly similar to *Master of Orion II*.

Contributors

- Paul Tozour
- Sensai Pose
- Unknown

Let's say you're designing a new game in the style of *MoO 2*. You want to know what the *optimal* initial build order is in the general case -- that is, all things being equal, what's the best sequence of actions for a player to build up a colony?

Being able to answer this question will be enormously useful to us in helping to balance the game and design all of the different building types.

Your population begins at 1 million people. It has a defined "maximum population" of 10 million people. It grows at a rate that depends on the ratio of the current population to the maximum population:

- At 10% of maximum population or less, it grows at 2.5% of the current population each turn
- At 20% of maximum population size, it grows at 3% of the current population each turn
- At 30% of maximum population size, it grows at 3.5% of the current population each turn
- At 40% of maximum population size, it grows at 4% of the current population each turn
- At 50% of maximum population size, it grows at 4.5% of the current population each turn
- At 60% of maximum population size, it grows at 4% of the current population each turn
- At 70% of maximum population size, it grows at 3.5% of the current population each turn
- At 80% of maximum population size, it grows at 3% of the current population each turn
- At 90% of maximum population size, it grows at 2.5% of the current population each turn

Also assume that the population is divided into "citizens," with each "citizen" representing 1 million people (rounded down). Each citizen requires 1 unit of "food." Each citizen can be dedicated to either farming or production. Each citizen designated as a farmer will grow 3 units of food, and the colony will automatically allocate as many farmers as needed to growing enough food to ensure all citizens are fed, with the remaining citizens dedicated to production (i.e. they are "workers").

Also assume that a colony produces 1 unit of "production," plus 2 additional units for each citizen designated as a "worker."

There are 5 initial building types in a colony:

- A **Hydroponic Farm** generates 2 additional food every turn. It costs 32 production units to build.
- The **Biospheres** improvement increases maximum population size by 3. It costs 56 production units to build.
- The **Cloning Center** improvement adds 100,000 additional population every turn. It costs 36 production units to build.
- The **Factory** increases the production of each worker citizen by 1. It costs 44 production units to build.
- The **Robotic Factory** adds a flat 5 production units plus an additional 1 production per worker. It costs 56 production units to build.

Additionally, there is a sixth thing we can build: **Housing**. We can build "Housing" whenever we like; unlike the others, it is not a discrete "building" but a target for us to allocate resources toward. Every turn that we build Housing, we generate an additional 10,000 population per unit of production.

This setup is a bit simplified compared to *Master of Orion 2,* but it's rich enough to prove the point we need to make here.

Our question is: given the above, what should the player build, and when should he build it, in order to get to the best possible colony as quickly as possible?

This is an even trickier problem than it first appears. If it were simply a matter of finding the best *ordering* of a Hydroponic Farm, Biospheres, Cloning Center, etc, to build then we could just look through all 5!=120 combinations.

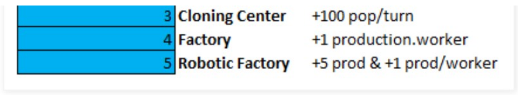
But in fact, the player can build Housing on any turn, and this has a huge effect on the colony's population.

So we're left with a much more complicated question: which of the 6 things (5 building types or Housing) should the player build for each of the N turns until all the buildings have been completed?

The Simulation

To solve this, we'll build a model conceptually similar to the one we used to solve for the tax rate in Part 2 of this series: we'll put the starting stats for the colony at the top of the attached spreadsheet and then show each new turn of the game on a subsequent row. You can think of this as "unrolling" the game simulation onto the spreadsheet, with the vertical axis representing time.

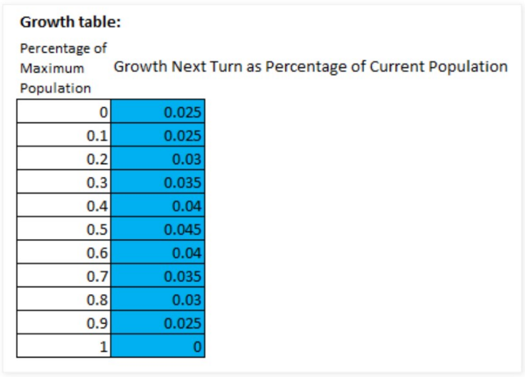
We'll start by making a table of "Production ID numbers." These are fixed numerical IDs we'll use to refer to each of the five building types, plus Housing. For example, '2' means "Biospheres." We'll show why this is useful in a moment.



3	**Cloning Center**	+100 pop/turn
4	**Factory**	+1 production.worker
5	**Robotic Factory**	+5 prod & +1 prod/worker

Since we're carefully following the formatting standards we laid out in Part 2, these cells are all blue, indicating that they're fixed values that are part of the problem definition.

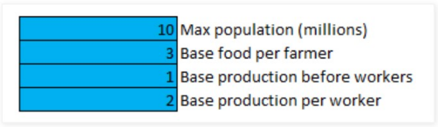
Next, we'll set up a table for the population growth rates we described earlier:



Growth table:

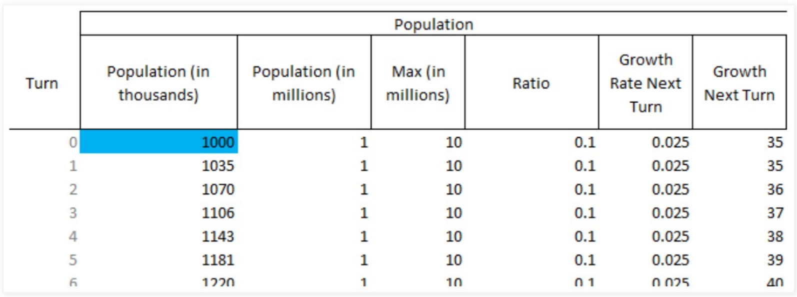
Percentage of Maximum Population	Growth Next Turn as Percentage of Current Population
0	0.025
0.1	0.025
0.2	0.03
0.3	0.035
0.4	0.04
0.5	0.045
0.6	0.04
0.7	0.035
0.8	0.03
0.9	0.025
1	0

We'll also put in a table with some of the constants we described for the colony in the previous section.



10	Max population (millions)
3	Base food per farmer
1	Base production before workers
2	Base production per worker

Now that we have these tables in place, let's build the simulation.



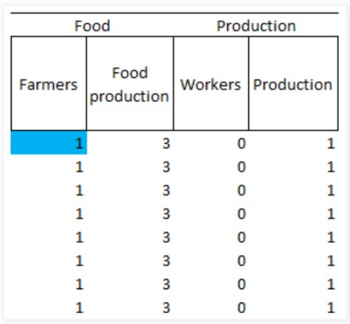
	Population					
Turn	Population (in thousands)	Population (in millions)	Max (in millions)	Ratio	Growth Rate Next Turn	Growth Next Turn
0	1000	1	10	0.1	0.025	35
1	1035	1	10	0.1	0.025	35
2	1070	1	10	0.1	0.025	36
3	1106	1	10	0.1	0.025	37
4	1143	1	10	0.1	0.025	38
5	1181	1	10	0.1	0.025	39
6	1220	1	10	0.1	0.025	40

We start by modeling population growth. This part of the spreadsheet lists the turns along the leftmost column, and in the subsequent columns, it lists the current population (in thousands), the number of "citizens" (population in millions), the current population maximum, the ratio of current to maximum population. Then, the "Growth Rate Next Turn" column looks up the current growth rate value in the "Growth table" above, while the "Growth Next Turn" multiplies this factor by the current population (since the growth rate is in terms of a percentage of current population).

You'll notice, however, that this is actually 35 instead of 25 for the first two turns. This is because we're actually building Housing for now, which adds another 10k population a turn. We'll get to that part of the spreadsheet in a moment.

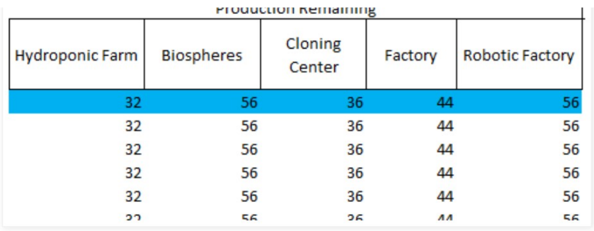
"Population (in thousands)" grows every turn by adding the previous population to the "Growth Next Turn." So, the 1000 population on turn 0 becomes 1000+35=1035 on turn 1, and so on.

Next, we'll model the citizens' job roles:



Food		Production	
Farmers	Food production	Workers	Production
1	3	0	1
1	3	0	1
1	3	0	1
1	3	0	1
1	3	0	1
1	3	0	1
1	3	0	1
1	3	0	1

We know that each farmer produces 1 "unit" of food. In this part of the spreadsheet, we determine the number of farmers and workers each turn. Each new "citizen" added to the city will become a worker by default. Any time the colony's food demand (= number of "citizens") exceeds the current food production (=3 x number of farmers), we'll reallocate one of these workers to a farming role instead. The production column lists current production according to the formulas stated at the top (1 unit of production + 2 per worker, plus any modifiers based on whether we've built things like a Factory or a Robotic Factory).

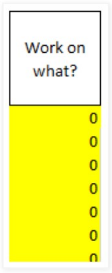


Production Remaining

Hydroponic Farm	Biospheres	Cloning Center	Factory	Robotic Factory
32	56	36	44	56
32	56	36	44	56
32	56	36	44	56
32	56	36	44	56
32	56	36	44	56

This gives us an easy way to track both the status of any building (whether it's been created or not), and our progress toward completing each of them.

Finally, in the rightmost column, we put in our *decision cells*. These cells will be integers that express our production decision for what to build each turn. Because we're following the formatting conventions laid out in Part 2, these cells are yellow, because they're the ones we're going to tweak with Solver.



Work on what?
0
0
0
0
0
0

The formulas in the spreadsheet are a bit too complex to go into in detail here; the reader is invited to download the attached spreadsheet. However, the important thing to note is that as we change the cells in this column to any value from 1 through 5 (the Production ID Numbers we listed in the table above), the spreadsheet allocates all our production resources that turn toward any of the five different buildings in the "Production Remaining" column (download the spreadsheet and tinker with the values in this column to see for yourself). If we keep them at 0, it represents Housing, and it adds to the colony's population instead.

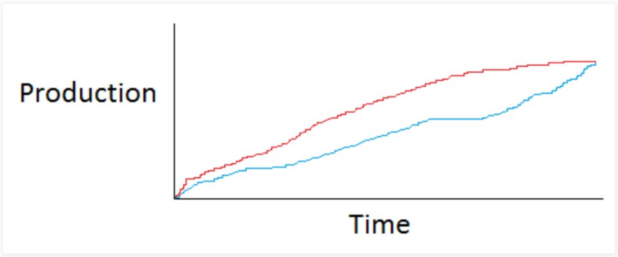
Optimizing

There's still one thing we're missing: an objective function to help guide Excel Solver to the right answer.

Our objective function should be something that expresses how powerful our city is -- its total population and the sum of all its capabilities. A highly skilled, "min-maxing" player will want to build the colony to its maximum capability level -- that is, maximum population, food production, and production output -- so that it can make the greatest possible contribution to the player's space empire.

Moreover, such a player will also want to achieve the maximum level of food and production output *as early as possible*, so that if the colony is interrupted by the need to do something else before this initial production queue is completed -- for example, it's attacked by an alien race and needs to retarget to military production -- it's likely to have a higher production capability at such a point to help it best address that challenge.

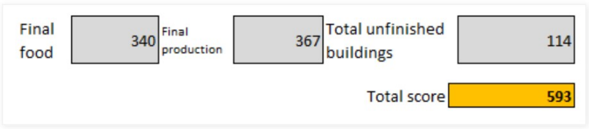
In other words, even though the blue curve and the red curve in the illustration below both get to the same endpoint of total production capability over time, we should prefer the red curve because it gets us to a higher level of total production capability earlier than the blue curve does. So we care about the total area under the curve, not just the endpoint of the curve.



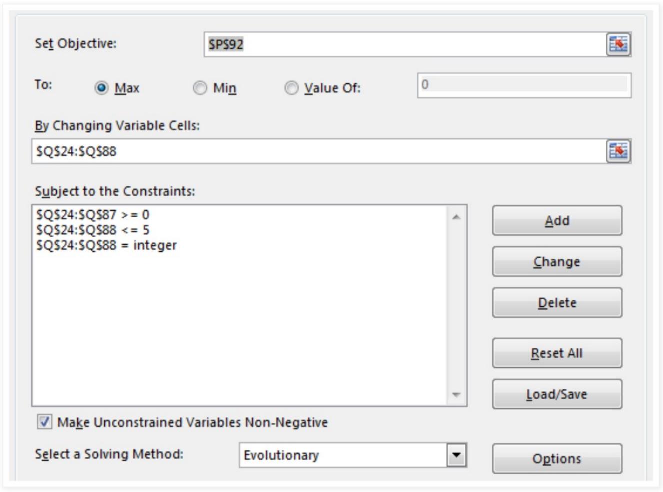
We'll build this as a customized function by adding three different values together:

- The total food production throughout all turns of the simulation. It might be enough for us to simply take the total value of food production on the last turn, but that might not differentiate between scenarios where we ramped up to maximum food production earlier in the simulation and those where we only ramped up later. All things being equal, we want to get to maximum food production as early as possible, so we'll take the sum of the food production over all turns.
- The total production over all turns. Just as with food production, it makes the most sense to add the sum of production over all turns to reward earlier production gains.
- An additional penalty for any buildings left unfinished. We want to ensure that all 5 building types are completed by the end, so we'll take the sum of all the remaining production units required on all 5 buildings and subtract it from the total. This should also help Solver find the best solutions more easily, since buildings don't have any effect on food or production until they're completed, and this factor helps reward Solver for *partially* completing buildings, whereas the previous two elements did not.

We then add those three values together, and we now have the following at the bottom of the spreadsheet.



Final food	340	Final production	367	Total unfinished buildings	114
				Total score	593


Set Objective: $P$92

To: (● Max) Min Value Of: 0

By Changing Variable Cells: $Q$24:$Q$88

Subject to the Constraints:

$Q$24:$Q$87 >= 0
$Q$24:$Q$88 <= 5
$Q$24:$Q$88 = integer

[✓] Make Unconstrained Variables Non-Negative

Select a Solving Method: Evolutionary

Now, we hit Solve.

Not So Good ...

When you run Solver, you'll notice that it takes quite a long time to solve. Since it's using the Evolutionary solving method (the drop-down box at the bottom of the dialog shown above), it can take a very long time to experiment with lots of different possible solutions and try to evolve the best solution.

In our first run, we get a total of 593 for our objective function after 64 turns of game time. This isn't very good; only the Hydroponic Farm and the Robotic Factory have been built. Repeated attempts to re-optimize don't seem to improve the solution at all.

The real problem here is that there are just too many parameters to optimize. We've asked Solver to solve 64 different parameters; there are a massive number of possible solutions. We've given Solver too big a haystack and too small a needle.

If you wanted to throw money at the problem, you could probably fix it by licensing a more powerful Solver engine from the folks who make Excel Solver.

But of course, that's *not* the approach I would recommend! There are inefficiencies in our decision model setup, and it would be far better for us to fix those first and see if fixing them won't let Solver find a solution.

As anyone who's done AI pathfinding knows, the best way to optimize your pathfinding algorithm usually isn't to tweak the algorithm itself, it's to simplify the space that it needs to search on. The simpler and coarser the pathfinding representation, the more quickly any search algorithm will run.

So we need to do the same thing: we need to take our 64 individual "What should I build this turn?" decisions and transform them into a much smaller set of decisions that Solver can handle using some kind of dimensionality reduction.

It Only *Looks* More Complicated ...

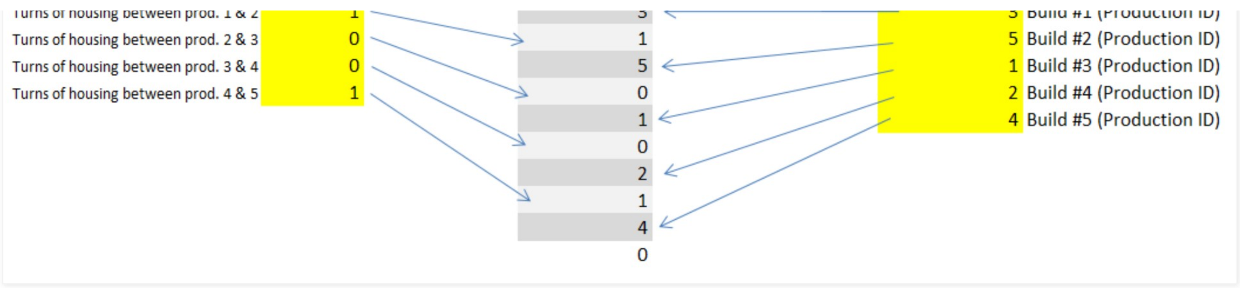
Let's rephrase our question. We can take advantage of the fact that any time we start building a particular building, we'll want to keep building it until we're finished. It seems safe to assume that the optimal production queue will build each of the five buildings in a focused stretch until they are done -- that is, there's nothing to be gained from stopping production on a building before it's finished to work on something else.

This assumption allows us to rephrase the question from "What should I build every single turn?" to a much simpler series of questions:

- How many turns should I build Housing before I create the first building? (0 or more)
- Which building should I build first? (Production ID 1 through 5)
- How many turns should I build Housing between the first and second buildings? (0 or more)
- Which building should I build second? (Production ID 1 through 5)
- How many turns should I build Housing between the second and third buildings? (0 or more)
- Which building should I build third? (Production ID 1 through 5)
- Etc ...

Then we continue that pattern until all 5 buildings have been built.

In order to do this, we first implement a table above the spreadsheet that looks like this:


Turns of housing between prod. 1 & 2 1
Turns of housing between prod. 2 & 3 0
Turns of housing between prod. 3 & 4 0
Turns of housing between prod. 4 & 5 1
3
1
5
0
1
0
2
1
4
0
3 Build #1 (Production ID)
5 Build #2 (Production ID)
1 Build #3 (Production ID)
2 Build #4 (Production ID)
4 Build #5 (Production ID)

The yellow decision cells on the left are the various "How many turns of housing?" questions we will ask Solver.

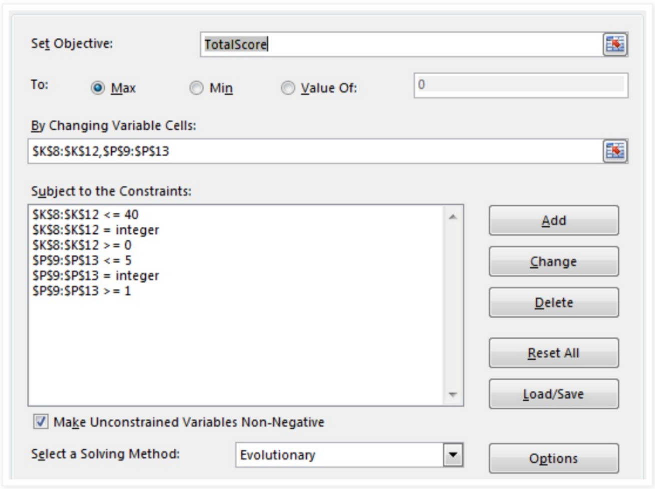
The yellow decision cells on the right are the five production IDs of the five different buildings.

In the center, we have our production table, which lists the sequence of events -- in this example, 18 turns of Housing, followed by building #3 (Cloning Center), then 1 more turn of Housing, then building #5 (Robotic Factory), then 0 turns of Housing, then building #1 (Hydroponic Farm), etc.

Finally, in order to implement this change, we take the entire spreadsheet and copy it to a new worksheet in Excel. Then, we replace all the cells in the "Work on what?" column with formulas that essentially express the build order as explained in the previous paragraph. This will cause the Production table above to essentially "program" the build order. We also change these cells from yellow to grey since they are no longer the decision cells.

The exact formula is very complicated, and I don't want to derail the article by going into all the minutiae of how it's expressed in Excel, but as always, you're welcome to download the spreadsheet and check it out for yourself if you're curious. (In short, we break it out into two separate columns, "Turns of housing in a row" that lists how many turns remain to build housing, and "Index in production table" that increments as we go further down the production table. The "Work on what?" column returns 0 (Housing) for even-numbered rows in the Production Table, and returns the corresponding Production ID for odd-numbered rows.)

We also have to change all of our Solver settings. The dialog now looks like this:


Set Objective: TotalScore

To: [✓ Max] Min Value Of: 0

By Changing Variable Cells:
$K$8:$K$12,$P$9:$P$13

Subject to the Constraints:
```
$K$8:$K$12 <= 40
$K$8:$K$12 = integer
$K$8:$K$12 >= 0
$P$9:$P$13 <= 5
$P$9:$P$13 = integer
$P$9:$P$13 >= 1
```

[✓] Make Unconstrained Variables Non-Negative

Select a Solving Method: Evolutionary

Here, the decision cells ("By Changing Variable Cells") are the yellow cells at the left and right sides of the new Production Table we showed above ("Turns of Housing" and "Production ordering"). The constraints simply state that the "Turns of Housing" cells need to be integers between 0 and 40, and the "Production ordering" values need to be integers between 1 and 5.

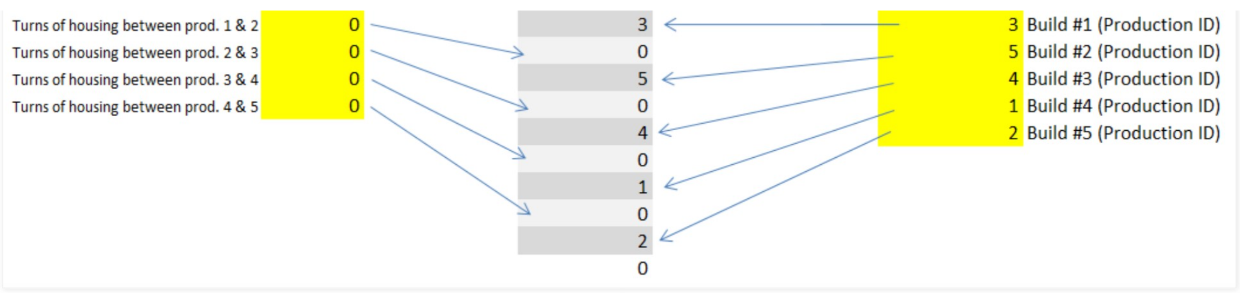
[If you've been reading this series diligently, you may be surprised at what's missing in this step. After all, the Production ordering table on the right side lists cells for the 5 buildings to create, each specified by a Production ID value from 1 to 5. Don't we need to add some sort of special constraint here to ensure that they end up with unique values, and that we don't cause Solver to try to build the same buildings twice?

The answer is no ... or at least, probably not. The objective function we've specified above should already reward Solver for building *all* the building types, and it won't benefit from trying to build a building more than once, so that should already naturally ensure that they're unique. So we'll leave off implementing this constraint until we actually need it (and as you'll see, we never will, so we can forget about it).]

Conclusion

Now, we can run Solver again, and in just a few minutes, Solver should give us our final answer, with our objective function having a value of 884 (if Solver doesn't reach this value for you, try re-running the Solver; you may also need to mess with the Solver's evolutionary optimization settings to help it attain this value).

This solution represents the following:


Turns of housing between prod. 1 & 2 0
Turns of housing between prod. 2 & 3 0
Turns of housing between prod. 3 & 4 0
Turns of housing between prod. 4 & 5 0
3
0
5
0
4
0
1
0
2
0
3 Build #1 (Production ID)
5 Build #2 (Production ID)
4 Build #3 (Production ID)
1 Build #4 (Production ID)
2 Build #5 (Production ID)

- Build Housing for 10 turns.
- Then build a Cloning Center.
- Then build a Robotic Factory.
- Then build a Factory.
- Then build a Hydroponic Farm
- Finally, build Biospheres.

That's it!

In a relatively short time frame, we've solved a problem that it would be extremely difficult to solve with guesswork or with any number of testers, and we've gained useful design insights in the process.

Not only that, we now have a tool that can allow us to quickly see how future design changes will affect the optimal build order. If we decide at some point down the road to lower the cost of Biospheres from 56 to 40, or change Hydroponic Farms to generate +3 food instead of +2, we can make that change and re-optimize the model to see how it changes the optimal initial build order.

From here, it's well worth looking at whether Cloning Centers and Robotic Factories are too powerful, too cheap, or too useful, or whether Hydroponic Farms and Biospheres might be too weak or too expensive. It might be a good idea to play around with the benefits and the production costs of each of these buildings and find the exact point at which Solver changes its answer. If we could find the point where Solver changed its mind about the ordering of two buildings, that would mean that we'd found the crossover point of those two buildings along a cost/benefit curve.

Ideally, we'd like to build a tool that would give us an instant sensitivity analysis illustrating how the value of each building changed as its cost (or any other parameter) changed -- though that's a topic for another day.

It's also worth noting that the buildings in this example lend themselves to a fairly well-defined objective function. This won't necessarily be the case in general, as many buildings could have more ambiguous or context-sensitive contributions to a colony's success. For example, the Marine Barracks in the example above generates marines to help with ground defense in case the planet is invaded. The value of a Marine Barracks will scale depending on the planet's likelihood of getting invaded, and it will naturally be much more useful closer to the empire's borders than it will be in a backwater planet in the farthest sectors of the map where enemies can't reach -- there's no point wasting resources on building defenses for a planet that will never be invaded.

It also depends to a certain extent on how many turns have elapsed in the game itself, since there is unlikely to be much planetary invasion in the early game, while space empires are just developing, whereas later in the game, enemies become much more aggressive and their technologies allow them to invade more distant planets.

In these cases, we're left to improvise when determining how such a building would contribute to the production queue's objective function. In the case of a Marine Barracks, we might have a contribution to the objective function that scales over time, or we might model multiple classes of colonies with different defensive needs (say, Low, Medium, and High), and give Marine Barracks and other such defensive buildings fixed weightings in each case, and see how the optimal production queue changes in each scenario.

Stay tuned! In future articles, we'll build decision models to help us design game levels, analyze game world layouts in a Metroid Prime style of game, and much more.

-Paul Tozour

This article was edited by Rob Zubek of SomaSim Games and Professor Christopher Thomas Ryan, Assistant Professor of Operations Management at the University of Chicago Booth School of Business.

Posted by Paul Tozour at 5:46 AM

Labels: Part 7

2 comments:

Pete September 10, 2013 at 3:00 PM

This series has been great, I'm REALLY looking forward to the Metroid analysis!

Reply

Sam February 16, 2020 at 11:43 PM

Our essay help online writing facility allows college and university undergraduates access to expert academic content writers who can take care of converting order instructions given by students into a skilfully customized academic paper.

Reply

Comment as: Ahmad Al-Kashef ((Sign out

Publish Preview Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)