Chapter 19

Time Estimates

The requirements analysis, game design, and technical design phases have us cover the most difficult part of project planning: identifying the tasks we must perform. Second to identifying the tasks is estimating how long it will take to complete them.

All great games are unique works of engineering and art carried out passionately by a team of game developers for an extended length of time. It is impossible at the start of the project to identify all the tasks that must be performed, and it is more than impossible to estimate exactly how long it will take to finish a creative effort that involves new bits of engineering and inspiration. Despite this challenge, if you walk into a publisher's office and announce at the end of your presentation that your project will take as long as it takes until the game is done, you will find yourself ushered out of the office after the publisher picks himself up off the ground. This calls for a story:

Recently Taldren required the services of an outside accounting firm; they quoted a price for their services that was a range, not a fixed bid. I dug a little deeper and it turned out they did not know if it was going to take them 20 or 40 hours to complete this accounting task for us. I gave it a moment's thought and realized that the accounting folks sure have their business model more mature than the typical game developer. I am sure it is true that he does not know if it will take 20 hours, 40, or somewhere in between to perform this accounting work. However, I could not help being offended since the business model we game developers use is a fixed bid. I told the accountant that we must agree to a fixed price for projects between 30,000 and 60,000 man-hours! It should be common sense that it is far easier to estimate a task that a single person will be charged with and is expected to last less than a week compared to a project requiring two dozen developers for nearly two years. Yet if I used the same range as the accountant, I would quote our publisher's advances (fees) like this:

Publisher: "Congratulations, I believe in your game, and we will pick it up. But first, how long will it take to complete and how much in terms of advances will you require?"

Erik: "We have given it considerable thought and planning and have settled on a $2.6 million budget and ready in 18 months at the low end and $5.2 million and three years at the upper end; we will let you know as we progress."

And that is when the publisher falls off his chair laughing and I am ushered out of the building.

The funny thing is that in reality many, many game projects are actually run as in the scenario above. More often than not developers find out mid-project that they can no longer ignore the underbid, and the release date is looming near and the game is not ready. It is not uncommon for some games to receive multiple new infusions of time and cash. These failures to accurately bid a project force publishers to be even more defensive in their positions and demand even more profit from future game development deals.

I now realize why independent game companies are so much more efficient than the military and TV and movie industries—we have to be! That is something I am genuinely proud of—how much work we all get done with relatively modest resources. That being said we do need to do a better job of estimating our projects and slowly weaning our publishers off of time and money budgets that estimate the fewest dollars and have a non-zero probability of creating the game.

Now let us get on with the estimating.

Two Ways to Estimate a Task

I have a very simple view when estimating the time required for a task to be completed; it is always one of two answers: How much time will it take to complete the task *or* how long do we have to complete the task.

Time Boxing

I find that in practice it is a lot easier to deal with tasks that simply have to be executed by a certain time. I first heard the term "time boxing" from a technical director at Electronic Arts Sports when I asked him how he estimates how long it will take to do something that is technically very challenging when there is not a standard reference for how long it will take to complete. He replied that when you simply do not know how long a task will take, spend your estimating time figuring out how long you could *afford* to be working on the problem. That becomes your time estimate that you later plug into your Gantt chart. If you run out of time and the task is still not complete and you intend to honor your time budget, you must abandon the task: Cut the feature, fall back to a less exciting version of the feature, or make some other cut to compensate for the loss. If you determine that you cannot perform a satisfactory cut and you are out of time, then you are stuck with going to your executive management team and advising them of your dilemma and requesting additional time and money, an activity you should avoid.

The elegant thing about time boxing is that you do not need to get bogged down in estimating something that is fundamentally unestimatable, and at the same time you have a powerful motivational tool for the developer(s) who must carry out the work. If someone knows that their work will simply be thrown away unless they complete it by a certain date, and that certain date is backed up by a rationally developed project plan, then they dig deep into themselves, concentrate, and usually find great satisfaction by pushing themselves to new levels of personal achievement to meet the deadline.

JARGON: *Time boxing*—placing a rigid time constraint around a task that is based on the project not being able to afford more time on that particular task.

Task Estimating

The core of the chapter is estimating how long it will take to implement some software or create some artwork. The artwork side is considerably more reliable in predicting than the software side.

Art

Typically your game's art assets can be broken down into a certain number of models, textures, animations, rooms, levels, sprites, tiles, and so on. Estimating is then very straightforward; create one or two of these assets and assume that the rest will take as long or even a bit shorter as the team develops more experience with the tools and the desired artistic direction.

Have your artists keep careful records of how long it is actually taking to create the art assets. After about one month of production, stop and hold a meeting and review their data on how long it is taking them. Also follow up on anything that appears to be taking longer than expected and ask if there is anything that could be done to make them more productive. Usually at this point I find that the estimates are trending downward. It is vitally important to task the artists, like all developers, at less than 100 percent capacity to allow for the usual time loss of illness, vacation, system upgrades, and unusual family events. In the artists' case, however, there is also the inevitable need to go back and iterate on specific areas of the game when an area is undergoing design drift or rapid technological development.

Design

My weakest area of task estimation is design. I have always led the design efforts at Taldren, and I am constantly undervaluing my time. As an entrepreneur I think it is perfectly normal to add another 10 hours of tasks to a 70-hour workweek. I say this glibly, without boasting, but it is the truth. This makes it difficult for the designers and producers below me, as I expect from them the same unflagging devotion to the company, and I feel I am not currently allocating enough time for design and production management. I recently returned from a visit to South Korea where I spent several days learning about how a game developer there, Makkoya, goes about its business of making games. One of the most striking impressions I had was that fully one-fourth of the company was devoted to game design! At Taldren, I am sure that one-fourth or more of the personnel enjoy significant responsibilities and authority in game design, but until this summer we never had a full-time employee exclusively devoted to game design. Game design is iterative and creative; this conspires to make it difficult to estimate how long it will take to *complete* a task. I generally allow approximately one man-day for the design of a screen or panel depending on its complexity. For larger systems such as how the combat system of an RPG system might work, a week might be appropriate to rough out the entire system and perhaps another two to three weeks to flesh out all of the details. In the end, you should have the designer responsible for the design task estimate how long it will take to complete the task. If game design is somewhat analogous to writing, then you should expect your designer to be able to generate three to ten pages of design a day. I feel uncomfortable attempting to distill the efforts of game design into too simplistic a metric. Please contact me if you have some better methods of estimating design tasks.

Programming

Programming tasks are notoriously difficult to estimate; in fact, it could be argued that the theme of this book revolves around the difficulty of planning software. There is no standardized method for predicting how long some programming task ought to take. There is no standard such as the number of lines of code per day per developer.

For example, if you create an incentive for programmers based on the number of lines of code, they will simply write more lines of code. This happened when Apple and IBM worked together on the Taligent operating system. The IBM engineers had labored under a number-of-lines-produced-per-day incentive program, while the Apple engineers were new to the system. The Apple engineers, being superb problem solvers and optimizers, realized they would be paid more money to write more lines of code, so they did—to the detriment to the project. Similar problems occur at the close of a project if management proposes bonuses based on the number of bugs closed per developer. Consciously or subconsciously, folks will realize that quality is not sought during development and they might as well be sloppy and collect the bug fixing awards at the end of the month.

Hire the best folks you can and avoid using incentive programs that motivate your programmers to go into another direction besides making the best game possible, on time and on budget.

There are roughly four categories of programming tasks:

1. Difficult, due to the design being vague, a time risk
2. Tedious but not a time risk
3. Simple and not a time risk
4. Difficult and a time risk

Category number one: tasks are difficult and time-consuming because they are vague. In my opinion, this is the number one reason why schedules break in my firm opinion. Schedules do not break because the developer is pushing the envelope too hard or because the developer has explicitly agreed to too many features. Rather, the schedule breaks when the developer agrees to perform a task at a high level without digging deep enough to find all of the required subtasks.

This is also by far the most difficult process to consistently master: task identification. That is why so much of this book focuses on raising the formalism of the game development process by involving a separate requirements gathering phase and Unified Modeling Language for specifying software requirements—tasks.

So how do you know if a task has been broken down enough? My simple rule is to ask the programmer, “How are you going do X?”

The response “I don’t know; when I get there I will figure it out,” is an easy red flag to spot. This task requires immediate attention to break it down into smaller tasks.

Better is this response: "Um, I will start with looking at Y for inspiration and then I will plug away for a while until I am confident that this is the best method for performing X in a separate test-bed framework. Then I will integrate the new code." That response may well be detailed enough to feel comfortable depending on that task's unique circumstances. You will need to make the call if it requires deeper analysis.

Of course the reply "Oh that's easy. I wrote up my attack plan earlier in the day and sent it to you in an email; didn't you get it?" makes me feel all warm and fuzzy inside like a good beer.

Often the real difficulty is that the project is not far enough along to break down a task into finer resolution. To address this I demand much more resolution for the tasks upcoming in the next 60 days or so than I require of tasks much later in the schedule. I highly recommend meeting at the beginning of each milestone to assess the quality and depth of your game and technical design material for the upcoming milestone and quickly assign someone to drive to the required resolution in advance of the rest of the development team. In this manner your technical directors and art directors should act like scouts for an army scouring the future and reporting back and suggesting course changes.

Category two tasks are tedious and time consuming but low risk. These tasks are not especially difficult to estimate nor do they create much worry that something unexpected will occur to everyone's unpleasant surprise. The danger that lies in these sorts of tasks is that due to their tedious, inglorious nature, the folks assigned to perform them will settle into a lower energy state, as their area is apparently not critical. This attitude could cause these tasks to go over budget, and again the resource assigned to the task may not understand the impact of running over schedule. There is not much trick in estimating these tasks; rather the challenge is maintaining a sense of importance and urgency in these tasks as the developer is working on them. You will need to show them what they will be working on next or how they could be helping out in other more exciting areas if they push through their slogful of tasks.

Category three tasks are simple and are not a time risk. Not much to be said here; these are straightforward. However, I do encourage you to load up your project with as many of these types of tasks as possible! As with long, tedious tasks, there is a minor danger of small, simple tasks seeming unimportant, and some time loss could occur here. However, I find this to be a relatively rare occurrence as most people derive pleasure from closing out their tasks, and the smaller tasks give them more apparent velocity on their task closure rate.

Category four tasks are the difficult, time-risk tasks that we touched on earlier with time boxing. These are the glory tasks usually assigned to your most senior programmers: create a new 3D engine, create a physics engine, reverse engineer something obscure, create a technique for doing anything no one has done before. The first thing to do with these types of tasks is be sure they are not masquerading as category one tasks, where the goals and design have been vaguely defined and that the current task appears difficult due to the breadth of the task. For example, "create a new 3D engine" is grossly vague as a task and could involve anywhere from the efforts of a few months to many man-years depending on the sophistication of the 3D engine requirements. This is clearly a candidate for breaking down into smaller steps. A better example of a category four task would be when John Carmack set out to put curved surfaces in Quake III. That would be an excellent task to wrap a time box around. (However, in John Carmack's case I would guess he just worked on it until he was satisfied with his efforts.)

At the end of the day you really want to eliminate as many of these types of tasks as possible from your game project. They act like festering boils on an otherwise healthy game project plan. Be sure each of these category four tasks that remain in your project are key features both in gameplay and in a marketing sense. If there is significant doubt that anyone will miss this particular feature, you should probably cut it and save yourself the schedule pressure.

Each Shall Estimate Thy Own Tasks

A key rule that I follow under all practical circumstances is to have my programmers estimate their own tasks. This has several powerful benefits. The most powerful is that you have full buy-in from the developer that they have a reasonable schedule to follow. Another benefit is that you are growing your employee's strength in project planning and management by having them participate or, even better, shape the final game development schedule.

How will they derive their own time estimates? At the end it will come down to a very subjective calculation that distinguishes humans from computers. We are able to soak in data from a myriad of sources—past performance, expected performance, level of interest, motivation, and guesses—and in a relatively short period of time estimate how long it will take to perform a task.

Yep, that's it; at the end of the day it will come down to just a gut estimate. Of course the simpler the item is, such as implementing a dialog box, the more straightforward the estimating process is. However, I do not know of anyone who has a software-project-estimator-o-matic device for coming up with estimates.

Save Your Plans and Compare

To improve your developers' skill at estimating, take care not to throw away their original estimates, and take the time to compare them with the actual results achieved during production. This should always be educational no matter how senior the programmer.

Making the Plan

Now that we have identified all of our tasks and have generated time estimates for them, it is time to flip to the next chapter and roll all of this data into a plan!

Chapter 20

Putting It All Together into a Plan

A lot of game companies use Microsoft Project to plan their game project schedules. MS Project is decidedly stronger at planning schedules than maintaining schedules. There are many annoying difficulties getting a workable schedule out of MS Project, and it seems that maintaining the tasks is certainly the most time-consuming chore a producer will face.

There are a host of other project planning products that you may use, but from running my "Real Methods of Game Production" roundtables at the Game Developers Conference in 1999 and 2002, I have found there is no clearly superior tool to MS Project.

Project planning and task tracking are two separate activities in my mind, and all of the project planning software packages including MS Project purport to do both tasks well. The real truth is these software packages do a decent job at planning a project, but when it comes time to update the schedule by closing tasks and inserting new tasks, the process is slow and tedious. Many times I have simply started new schedules to plan out from that current point to the end of the project. Mind you, it is not impossible; it just takes a lot of time.

The larger game development teams schedule maintenance that is so time consuming they have a dedicated human on their team updating the schedule full time! Part of the problem is that game projects with 100 to 800 man-months lie somewhere between the two classes of project management software: the dozen or so man-months of effort for a marketing campaign (which MS Project is excellent for) and the hundreds of man-year efforts for major construction projects (for which you need very expensive software such as Primavera and a small group of dedicated project managers).

Okay, now on with the overview of MS Project.

The goal of a schedule is to organize all of the project's tasks, illustrate the dependencies between tasks, track progress, level tasks, and assist in scenario planning.

Dependencies need special care as you do not want some of your development team to stall for lack of art, for example, nor do you want a critical feature to fail to be completed on time for a given milestone because the key component of this critical feature must be completed by just one programmer who task.

Tracking progress is simply marking off tasks that have been completed.

Scenario planning is using the software to analyze different "what-if" scenarios such as "What if we cut the map editor altogether?"

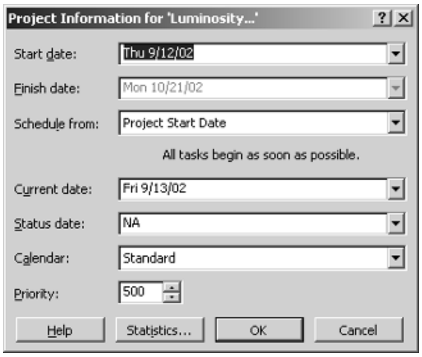
another critical

Let's Create a Schedule for FishFood!

Go ahead and fire up your copy of MS Project. A wizard tool will pop up suggesting that you take up the wizard's offer; decline the offer, close, and close the window.

Create a New Project File

A blank project will be staring at you; dismiss this project and select File | New to create a new project file. A project information pop-up dialog will solicit either a start date or end date. Choose a start date to schedule from rather than an end date to schedule back from.

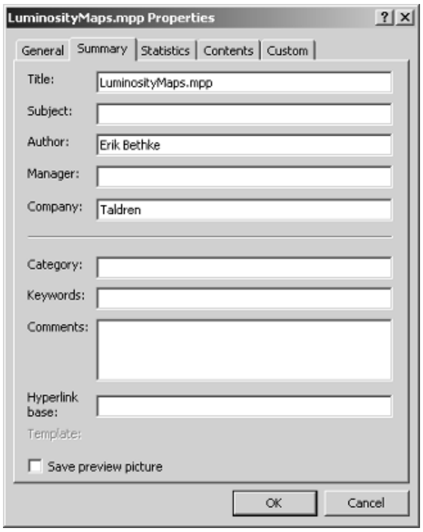


Project Information for 'Luminosity...'

Start date:	Thu 9/12/02
Finish date:	Mon 10/21/02
Schedule from:	Project Start Date
	All tasks begin as soon as possible.
Current date:	Fri 9/13/02
Status date:	NA
Calendar:	Standard
Priority:	500

Properties for an MS project

Project, like all Office products, offers a properties dialog that you may fill out with a bunch of dull details such as author name, manager, company, etc. If you feel the need to decorate your files with such details, choose File | Properties.



LuminosityMaps.mpp Properties

Title:	LuminosityMaps.mpp
Subject:	
Author:	Erik Bethke
Manager:	
Company:	Taldren
Category:	
Keywords:	
Comments:	
Hyperlink base:	
Template:	

☐ Save preview picture

MS Project's File | Properties dialog

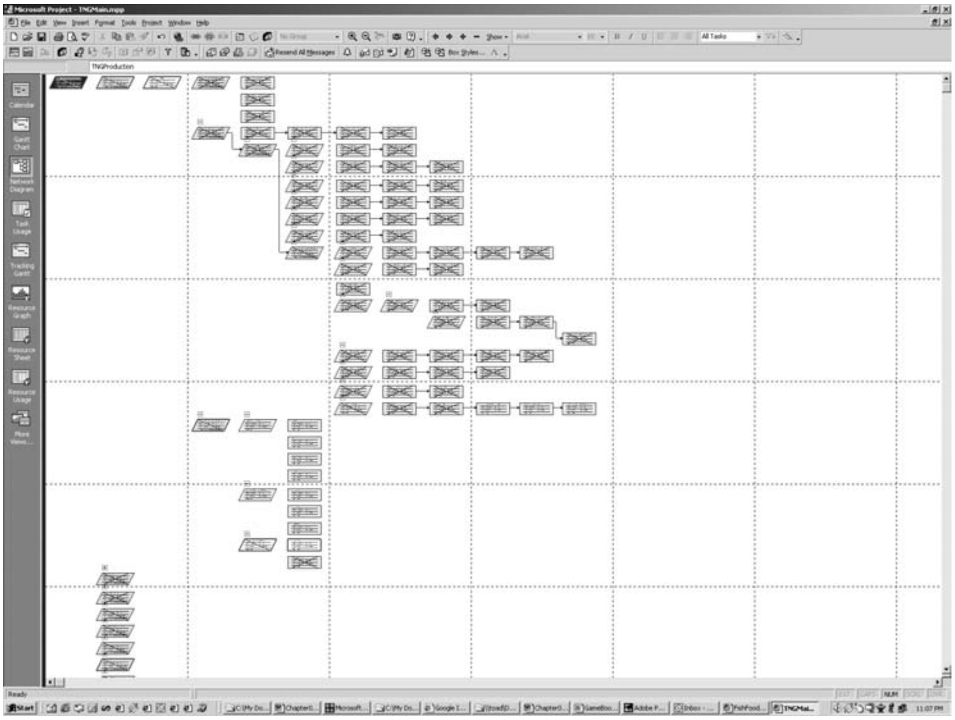
What Is a PERT/Gantt Chart Anyway?

There are a myriad of diagrams, charts, and reports you are able to generate with Project. A good-sized project will be composed of thousands of bits of information from task names to assigned resources, start dates, priorities, and dependencies. The view you choose will reflect what you are trying to get a good look at. The two most common types of charts are Gantt and PERT. These were introduced in Chapter 10.

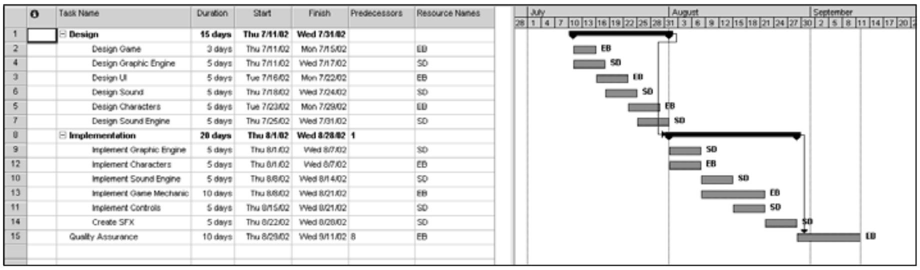
To review, the PERT chart is the visually simpler chart with boxes for tasks that are drawn left to right with dependency links between the boxes. The boxes may be detailed with duration and resource name.

The advantage of the PERT chart is that it displays the critical path of a project very well. It stands out like the trunk of a tree with non-critical path tasks stemming from the trunk as branches or sometimes as solitary boxes. PERT charts are fun to fill in as tasks are completed. The disadvantage to a PERT chart comes when you are charting more than just the high-level tasks, say fifty to a thousand tasks. When the number of tasks reaches that size, it takes a lot of paper to print out the chart, and the dependency lines may become too tangled to make much visual sense. Another minor disadvantage of the PERT chart is that since it is such a graphical layout, the project manager might get distracted for an inordinate amount of time fiddling with the boxes and getting the layout of the boxes to look good. (This apparently remains a minor AI problem to solve someday: well-laid-out PERT charts.) Bottom line: PERT is good for overviews and easily constructed from the Gantt chart.

The Gantt chart consists of a spreadsheet of data on the left-hand side such as task ID, task name, start date, end date, duration, and resource name (who is going to do the job). On the right-hand side are the tasks graphically portrayed as bars of varying length proportional to their duration laid out left to right underneath the project calendar displayed at the top of the chart.



A sample PERT chart



		Task Name	Duration	Start	Finish	Predecessors	Resource Names
1		**Design**	**15 days**	**Thu 7/11/02**	**Wed 7/31/02**		
2		Design Game	3 days	Thu 7/11/02	Mon 7/15/02		EB
4		Design Graphic Engine	5 days	Thu 7/11/02	Wed 7/17/02		SD
3		Design UI	5 days	Tue 7/16/02	Mon 7/22/02		EB
6		Design Sound	5 days	Thu 7/18/02	Wed 7/24/02		SD
5		Design Characters	5 days	Tue 7/23/02	Mon 7/29/02		EB
7		Design Sound Engine	5 days	Thu 7/25/02	Wed 7/31/02		SD
8		**Implementation**	**20 days**	**Thu 8/1/02**	**Wed 8/28/02**	**1**	
9		Implement Graphic Engine	5 days	Thu 8/1/02	Wed 8/7/02		SD
12		Implement Characters	5 days	Thu 8/1/02	Wed 8/7/02		EB
10		Implement Sound Engine	5 days	Thu 8/8/02	Wed 8/14/02		SD
13		Implement Game Mechanic	10 days	Thu 8/8/02	Wed 8/21/02		EB
11		Implement Controls	5 days	Thu 8/15/02	Wed 8/21/02		SD
14		Create SFX	5 days	Thu 8/22/02	Wed 8/28/02		SD
15		Quality Assurance	10 days	Thu 8/29/02	Wed 9/11/02	8	EB

A sample Gantt chart for a simple game called FishFood!

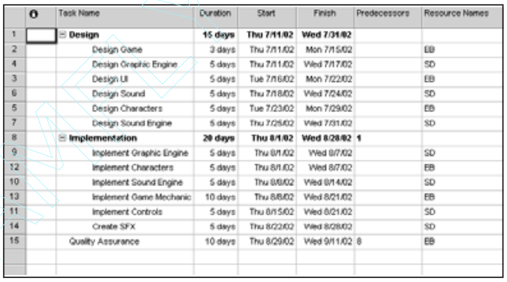
The main advantage of the Gantt chart is that it is good for displaying up to several hundred tasks resulting in a finer granulation in your schedule. (The finer the granulation in your schedule, the more likely you are planning all of the required tasks, and thus the more likely you will be on time.)

Dependencies between tasks are drawn as simple arrows between the tasks. The Gantt chart is easy to read both from top to bottom, with the convention of the earlier tasks at the top, and from left to right as time passes.

The main disadvantage to the Gantt chart is, of course, the key strength of the PERT chart: that it is difficult to see at a glance the critical path of the project. Fortunately, with Project it is simple to enter your task information under the Gantt chart and later choose to view your scheduling information from any number of views such as the PERT chart.

Start Entering Tasks

Entering task information in Project really could not be easier. Pick a row and start by simply typing in the name of the task in the Task Name column, and enter the estimated time for duration. Bam, you have entered a basic task.



		Task Name	Duration	Start	Finish	Predecessors	Resource Names
1		**Design**	**15 days**	**Thu 7/11/02**	**Wed 7/31/02**		
2		Design Game	3 days	Thu 7/11/02	Mon 7/15/02		EB
4		Design Graphic Engine	5 days	Thu 7/11/02	Wed 7/17/02		SD
3		Design UI	5 days	Tue 7/16/02	Mon 7/22/02		EB
6		Design Sound	5 days	Thu 7/18/02	Wed 7/24/02		SD
5		Design Characters	5 days	Tue 7/23/02	Mon 7/29/02		EB
7		Design Sound Engine	5 days	Thu 7/25/02	Wed 7/31/02		SD
8		**Implementation**	**20 days**	**Thu 8/1/02**	**Wed 8/28/02**	**1**	
9		Implement Graphic Engine	5 days	Thu 8/1/02	Wed 8/7/02		SD
12		Implement Characters	5 days	Thu 8/1/02	Wed 8/7/02		EB
10		Implement Sound Engine	5 days	Thu 8/8/02	Wed 8/14/02		SD
13		Implement Game Mechanic	10 days	Thu 8/8/02	Wed 8/21/02		EB
11		Implement Controls	5 days	Thu 8/15/02	Wed 8/21/02		SD
14		Create SFX	5 days	Thu 8/22/02	Wed 8/28/02		SD
15		Quality Assurance	10 days	Thu 8/29/02	Wed 9/11/02	8	EB

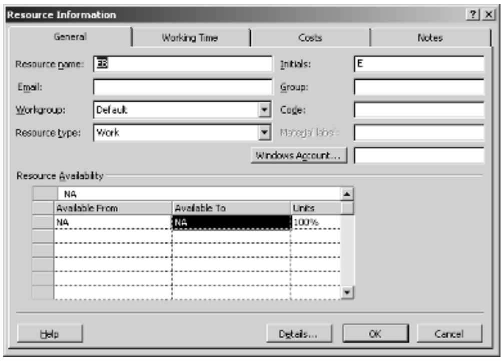
Focusing on a task name and duration

Now let's talk about task names. It is important to be sure the name of a task includes a strong verb like "purchase workstations" or "test logon protocol" or "implement save game" rather than the vague "workstations," "logon protocol," and "save game." The strong verb makes the difference between a task and a topic. I still make the mistake of using topic names rather than task names; this is usually a strong hint from my subconscious that this topic has not been thought out enough for me to feel comfortable articulating discrete tasks.

Another common mistake I see in game project schedules, including my own, is that the schedule is composed of only features to be implemented and assets to be created. You may be wondering what else there is to game production. Well, it does take actual time to test all of the deliverables in the milestone before you send it off to the publisher. It takes time to respond to the publisher's feedback, it takes time to go to E3, and it certainly takes loads of time creating the E3 build! It takes time to train developers on new tools such as when you switch from Character Studio to Maya. It takes time to create all of the documentation at the start of the project. It takes time to reconcile the schedule with reality. It takes time to submit a build to the license manager and get feedback. It takes time to plug in the sound effects and voice-overs. And it certainly takes time to balance and tweak your games.

Tasks Are Performed by Resources

The final key bit of information that you need to add to your task besides task name and duration is who will do the work—the resource. Enter the first name, last name, initials, job title, or alphanumeric string you want in the resource column.



Resource name: EB
Initials: E
Email:
Group:
Workgroup: Default
Code:
Resource type: Work
Material label:

Resource Availability

NA

Available From	Available To	Units
NA	NA	100%

Adding a human resource to a project

If you do not know at this time who will be performing the task, as you want to see how things will stack up before deciding, guess, flip a coin, or choose somebody at this point. We will use Project's task leveling tool later to help us sort out who should be doing what for maximum productivity.

Where Does All of This Task Information Come From?

An excellent question to ask at this point is where these task names, time estimates, and durations are coming from.

Experienced game developers who have led large portions of games and who are tasked with creating a schedule for a type of game project they are familiar with will be able to sit down with Project and immediately dash off a few dozen tasks before pausing to think. However, at some point both the experienced project manager and the less experienced project manager will need to come up with tasks in a more formal manner. By far the best (and only) way to come up with the tasks is to get them from the people who will be carrying out the work, or at the very least the leads of each of the portions of the game project. For example, your lead animator should come up with estimates for all of your animation. I would advise against your lead programmer, who might specialize in graphics, coming up with the online multiplayer tasks; those should come from the multiplayer programmer herself.

You may choose to collect these tasks from a rather informal brainstorming session, or you might send an email out to everyone to review the design documents (game and technical) and come up with the tasks for their area of the project. The size and scope of your project will determine what works best for your project. In general, if I am trying to execute a mini-project like getting a build ready for E3 or developing a demo for a brand-new game we will be pitching to publishers, the informal brainstorming approach works most efficiently for me. I reserve the more formal approach, where each resource is given perhaps a week to break down his area of the schedule into composite components, for the beginning of full production.

The reason it is so important to get the developers themselves to come up with the tasks is threefold: First, they are the experts in that field, and they will be better able to break the problem down into smaller pieces. Second, you want them to participate in the scheduling so that they understand better what they need to accomplish, why, and by when. Finally, by giving the developer the authority to set the time estimates you will achieve a far greater "buy-in" and sense of obligation to get the job done in a reasonable amount of time compared to when the schedule is passed down by a heavy (and often less knowledgeable) hand from above.

Organizing Tasks

I have to admit I like building MS Project Gantt charts (it is a good thing to like your job), and Project makes it easy to organize the tasks in your Gantt chart. There are nine levels of indentation to facilitate the logical grouping of tasks. As the tasks are coming in from your various team members, plug them into the chart, push them around, and indent them; have fun. Now is the time to make the schedule logical and clean. In fact, this aspect of project making is so easy I am able to do it in real time for small projects with half a dozen of my guys riding shotgun over my shoulder, shouting out tasks and time estimates, and am able to keep up and cook a schedule together. Some people might shudder at the apparent lack of thought put into a schedule crafted in that manner; however, I have found that all schedules are merely estimates of what needs to happen. Also, most people's guesstimates of how long a task will take to complete will not be far off from a more carefully crafted estimate (both of which are bound to differ more relative to the actual time it took to complete the task compared to the difference between the two tasks).

Draw dependencies between tasks with reluctance; do not think that the more lines you draw on the Gantt chart the more accurate your schedule will become. Rather, group related tasks under super-tasks and draw dependencies between these chunkier bits.

Task Granularity

How fine in time resolution should your task estimations be—a day or a week or some other time? I have been back and forth across the issue and yes, the finer the resolution the more accurate and reliable the project is likely to be. If you could measure every task down to a quarter of a day, you would have tremendous resolution to work with, and you would have a Gantt chart that would impress the most jaded of executive management teams. The problem with schedules with ultra-fine task resolution is that they invariably become wrong quite quickly and require a tremendous amount of producer time to fix: Delete these 10 tasks, add these 20 tasks, modify the duration of these two dozen tasks, and so on.

Thus, my new philosophy on task resolution for schedules is to cut the tasks into pieces as small as possible but no smaller than the producer has time to maintain. This is really just being honest with yourself and knowing what your time limitations are for maintaining the project plan. I would say that a schedule that has 15 developers working on a game for 15 months should have somewhere between 300 and 600 tasks in the project plan.

How to Account for Vacation and Sick Time

When creating your schedule you must account for vacation and sick time. I have to admit I was vexed for quite a while on how to best manage the planning for vacation and sick time. I mean, how would you know that your lead programmer would come down with bronchitis and lose seven days 13 months from now? If you try to stick tasks in the project plan called "vacation" or "sick days," you are creating a bunch of little falsehoods that will annoy you as you try to perform project leveling. Project will toss these tasks about all over the place, and you will start placing dependency lines or special instructions for the timing of each and every one of these tasks.

After thinking about the sick and vacation day problem for a long time, I have finally developed an elegant and easy solution: I modify the working calendar for all of the developers at Taldren and change Fridays to half-days. This effectively places two full days of fluff per month into the schedule, leaving 24 working days a year for sick and vacation time. Take whatever your company manual says about days off and adjust your Friday time off up and down to suit your tastes (I recommend going a little bit conservative).

I like this method for handling unschedulable tasks so much I might start writing off part of Mondays for project maintenance, system upgrades, interviews, dog and pony shows, and other unplanned tasks that tend to affect everyone in the company at one time or another. By keeping these tasks separated on Mondays vs. Fridays, I will be able to adjust either one up or down as I develop more accurate historical data.

Remember Odd Tasks

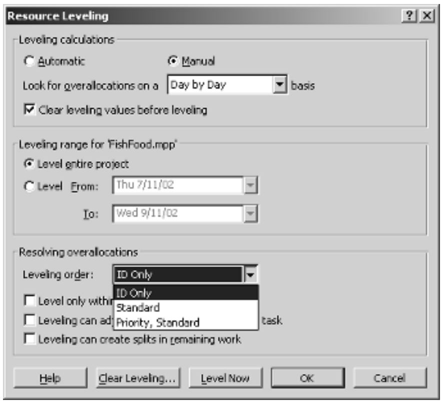
Scour your collective brains to identify weird or odd tasks like trade shows, submission to hardware manufacturers, the installer, the auto-patcher, customer service, fan interaction, and so on. This is one area where experienced organizations have an edge on start-ups; the start-ups generally only plan for the absolute minimum of tasks yet still have to complete all the tasks that everyone else does as well.

Time Leveling in Project

The main advantage of a project tracking package such as MS Project over a task tracking database is the ability to analyze the loads between the various team members and perform task leveling.

There are two principal tools for performing task leveling in MS Project that complement each other in your quest for a clean, balanced schedule across your team: the automated leveling tool and the resource usage view.

After you plug in all of the tasks with the required bits of info of who and how long, click on Tools | Resource Leveling | Level Now….



Resource Leveling

Leveling calculations
- ( ) Automatic (●) Manual
- Look for overallocations on a: Day by Day basis
- [✓] Clear leveling values before leveling

Leveling range for 'FishFood.mpp'
- (●) Level entire project
- ( ) Level From: Thu 7/11/02 To: Wed 9/11/02

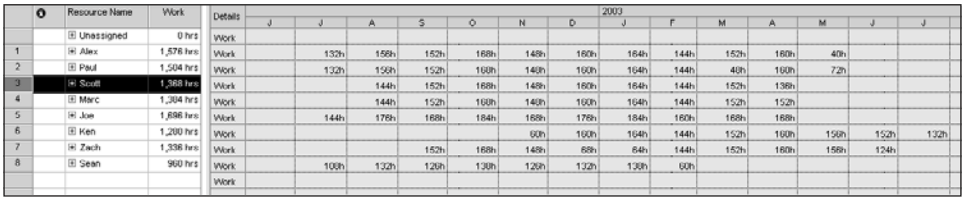
Resolving overallocations
- Leveling order: ID Only (ID Only / Standard / Priority, Standard)
- [ ] Level only within ... task
- [ ] Leveling can ad...
- [ ] Leveling can create splits in remaining work

The Resource Leveling dialog in MS Project

Despite the intimidating number of choices on this dialog box, there are really only two meaningful options: to level by ID or to level by Priority, Standard. For simple schedules with less than 300 tasks, I find that leveling by ID tends to work well as the Gantt chart will most likely be laid out with early tasks at the top of the chart and later tasks at the bottom of the chart. The Priority sort is useful when you have truly large project files and you have attached priority weighting to each of your tasks (if you do not weight individual tasks, then the leveling will behave as if you had chosen the ID sort).

What MS Project does during the leveling is look at all of the dependencies between the tasks across all resources and lay them out in time in order to best accommodate a smooth path to completion. As you know, computers are not intelligent; as such, MS Project will make a finite number of dumb placements of tasks. Your job is to look over these errors and correct them through adding dependency lines, priority weightings, or time constraints such as "start no earlier than X date." After iterating for a while you will end up with a schedule that makes sense.

All done? No. If you click on the Resource Usage View button, you will discover that your task assignments have caused an uneven allocation of time across your team, as shown in the following figure.

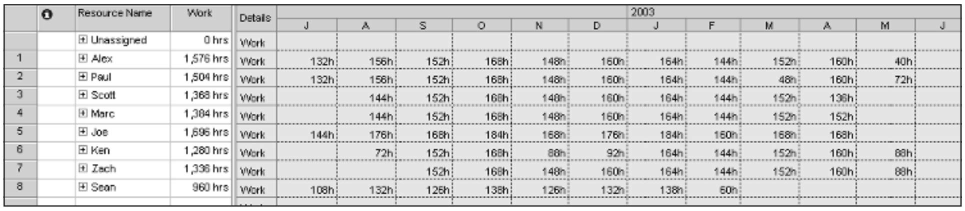


											2003						
	Resource Name	Work	Details	J	J	A	S	O	N	D	J	F	M	A	M	J	J
	⊞ Unassigned	0 hrs	Work														
1	⊞ Alex	1,576 hrs	Work		132h	156h	152h	168h	148h	160h	164h	144h	152h	160h	40h		
2	⊞ Paul	1,504 hrs	Work		132h	156h	152h	160h	140h	160h	164h	144h	40h	160h	72h		
3	⊞ Scott	1,368 hrs	Work			144h	152h	168h	148h	160h	164h	144h	152h	136h			
4	⊞ Marc	1,304 hrs	Work			144h	152h	160h	140h	160h	164h	144h	152h	152h			
5	⊞ Joe	1,696 hrs	Work		144h	176h	168h	184h	168h	176h	184h	160h	168h	168h			
6	⊞ Ken	1,200 hrs	Work						60h	160h	164h	144h	152h	160h	156h	152h	132h
7	⊞ Zach	1,336 hrs	Work				152h	168h	148h	68h	84h	144h	152h	160h	156h	124h	
8	⊞ Sean	960 hrs	Work		108h	132h	126h	130h	126h	132h	130h	60h					
			Work														

The resource usage view before leveling

You will see that some of your developers have large gaps of idle time in their schedule, and others are acting as the long pole and causing the game to sprawl out past the final delivery date. How do you fix this? You have to understand what MS Project is telling you. It is saying that the long pole folks have been assigned too many critical path tasks and the others with gaps in their schedules are twiddling their thumbs while waiting for the critical path folks to deliver the goods. The solution is to look for tasks belonging to the critical path folks that may be transferred to the people with gaps. This is very much an iterative process as you are looking for clever bits of reassignment that will neatly cover the gaps in some folks' schedules while eliminating the spikes in the critical path folks' schedules.

To solve the difference between developer A with a gap and developer B with a spike, you might have to rotate a subset of tasks through developer B and developer D to make it all work out. The goal is to massage your schedule until your resource usage view looks like a nice clean brick of time with all gaps filled and the whole team finishing up at the roughly the same time.



	❶	Resource Name	Work	Details							2003					
					J	A	S	O	N	D	J	F	M	A	M	J
		⊞ Unassigned	0 hrs	Work												
1		⊞ Alex	1,576 hrs	Work	132h	156h	152h	168h	148h	160h	164h	144h	152h	160h	40h	
2		⊞ Paul	1,504 hrs	Work	132h	156h	152h	168h	148h	160h	164h	144h	48h	160h	72h	
3		⊞ Scott	1,368 hrs	Work		144h	152h	168h	148h	160h	164h	144h	152h	136h		
4		⊞ Marc	1,384 hrs	Work		144h	152h	168h	148h	160h	164h	144h	152h	152h		
5		⊞ Joe	1,696 hrs	Work	144h	176h	168h	184h	168h	176h	184h	160h	168h	168h		
6		⊞ Ken	1,280 hrs	Work		72h	152h	168h	88h	92h	164h	144h	152h	160h	88h	
7		⊞ Zach	1,336 hrs	Work			152h	168h	148h	160h	164h	144h	152h	160h	88h	
8		⊞ Sean	960 hrs	Work	108h	132h	126h	138h	126h	132h	138h	60h				

After leveling

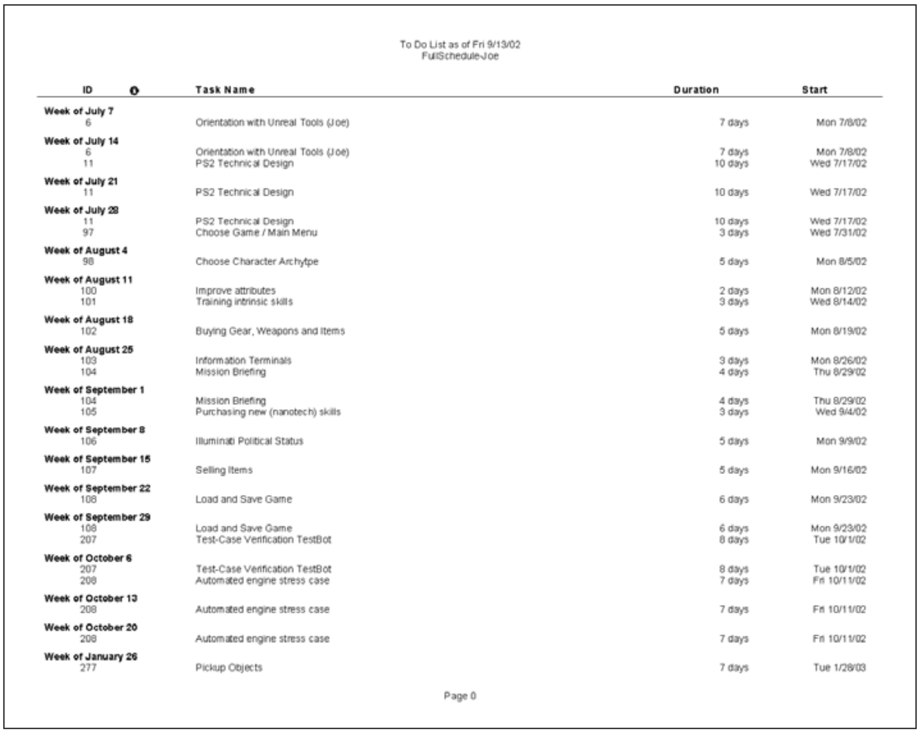
Let it Jell

All freshly minted project plans are full of errors, inconsistencies, and omissions. All of my project plans needed several passes to get into shape, and the difference from the first draft to the first revision is always the most dramatic. You will not be able to fix these flaws the same day that you create the schedule. Instead you must let it jell for at least a week and then come back and read through the schedule carefully with the leads of your team with a critical eye. Take the time to do this and you will make a schedule at least twice as strong as it was just a week previously. Novelists must do this with finished manuscripts; producers should also set aside their schedule for a time and revise.

How to Distribute the Schedule to the Team

A fine schedule that is locked up and kept in the oracle's tower is not very useful. A project plan must be a communication device used by the whole team. Every time I think of producers who keep the schedule information secret, I squint like Clint Eastwood and twitch my fingers looking for a gun. Managing a team is not a management vs. the developers contest! Take the schedule and paste it up on the wall! As the team members get tasks accomplished, have them go up and highlight completed tasks (more on measuring progress in the next chapter).

Take the time to create customized reports for each of your team members. MS Project boasts a number of reports including To Do Lists and a Who Does What list, as shown on the following page.



To Do List as of Fri 9/13/02
FullSchedule-Joe

ID	Task Name	Duration	Start
Week of July 7			
6	Orientation with Unreal Tools (Joe)	7 days	Mon 7/8/02
Week of July 14			
6	Orientation with Unreal Tools (Joe)	7 days	Mon 7/8/02
11	PS2 Technical Design	10 days	Wed 7/17/02
Week of July 21			
11	PS2 Technical Design	10 days	Wed 7/17/02
Week of July 28			
11	PS2 Technical Design	10 days	Wed 7/17/02
97	Choose Game / Main Menu	3 days	Wed 7/31/02
Week of August 4			
98	Choose Character Archytpe	5 days	Mon 8/5/02
Week of August 11			
100	Improve attributes	2 days	Mon 8/12/02
101	Training intrinsic skills	3 days	Wed 8/14/02
Week of August 18			
102	Buying Gear, Weapons and Items	5 days	Mon 8/19/02
Week of August 25			
103	Information Terminals	3 days	Mon 8/26/02
104	Mission Briefing	4 days	Thu 8/29/02
Week of September 1			
104	Mission Briefing	4 days	Thu 8/29/02
105	Purchasing new (nanotech) skills	3 days	Wed 9/4/02
Week of September 8			
106	Illuminati Political Status	5 days	Mon 9/9/02
Week of September 15			
107	Selling Items	5 days	Mon 9/16/02
Week of September 22			
108	Load and Save Game	6 days	Mon 9/23/02
Week of September 29			
108	Load and Save Game	6 days	Mon 9/23/02
207	Test-Case Verification TestBot	8 days	Tue 10/1/02
Week of October 6			
207	Test-Case Verification TestBot	8 days	Tue 10/1/02
208	Automated engine stress case	7 days	Fri 10/11/02
Week of October 13			
208	Automated engine stress case	7 days	Fri 10/11/02
Week of October 20			
208	Automated engine stress case	7 days	Fri 10/11/02
Week of January 26			
277	Pickup Objects	7 days	Tue 1/28/03

Page 0

A sample To Do List report for the Black9 project

Also, you can sort the main Gantt chart by resource name and print out just that section of the schedule. Print out mini Gantt charts for each team member to stick up on their own walls—they'll love it!