PART I

About Game Testing

CHAPTER 1

Two Rules of Game Testing

Whenever I start a new test team or bring a new tester into the group, I give them these two rules:

Rule 1: Don't Panic

Rule 2: Trust No One

Don't Panic

In a game project, panic is a bad thing. The person panicking did not choose to panic, and may not realize it is happening. It is an irrational reaction to a set of circumstances, and it can lead a tester to cause harm to the project. When I sense that a tester is reacting inappropriately to some unreasonable request, I will indirectly remind him not to panic by asking "What's rule one?"

Scuba divers put themselves in a situation similar to what game testers might face: limited resources (the equipment you bring with you), time constraints (air supply), rules to follow (rate of descent/ascent), and other surprises (unexpected sea visitors). According to Dr. William Morgan, episodes of panic or near-panic may explain many recreational diving accidents and deaths. The panic attack was often spurred by something that a non-diver would deem serious—entanglement, an equipment malfunction, or the sight of a shark. But the attacks don't make things better, Morgan says—they can lead to irrational and dangerous behavior.[1] Even scuba divers with many years of experience sometimes experience panic for no apparent reason.[2]

1. "I'm Afraid We Must Talk About…Panic Underwater." The Why Files™ <http://whyfiles.org/sports/scuba> (January 21, 2005)

2. "More questions and answers about Panic Underwater…" The Why Files™ <http://whyfiles.org/sports/scubaq4.html> (January 21, 2005)

Testing the wrong build, failing to notice an important defect, or sending developers on a wild goose chase after a non-existent bug shouldn't end up getting you physically hurt, but there will be a price to pay in extra time, extra money spent, and/or loss of sales and reputation.

Game project panic happens when you are

- Unfamiliar
- Unprepared
- Under pressure
- Unrested
- Nearsighted

Unfamiliar

As a member of a game team, you might be asked to do something you've never had to do before. You might be given someone else's tests to run, be thrown into the middle of a different game project, or told to take someone else's place at the last minute to do a customer demo. In situations like these, rely on what you know, stick to basics, and pick up any new or different ways of doing things by watching the people who have already been doing it.

You may even be asked to accomplish something you've never done before, such as achieve 100% automation of the installation tests, or write a tool to verify the foreign language text in the game. Maybe *no one* has ever done this before. Don't make a commitment right away, don't make stuff up, and don't try to be a hero. If you are unfamiliar with a situation, you act based on your best judgment, but it still may not be right. This requires good "radar" on your part to know when to get help, and also a dose of humility so you don't feel like you have to take on everything yourself or say "yes" to every request. You don't need to lose any authority or "street cred." Find someone who's "been there, done that" and can steer you toward some working solutions. Stay away from responses that are known to fail. You can even search the Internet to see if anyone else has been through it and lived to tell about it.

Note

Chapter 8, "The Test Process," shows you how to define and follow a set of activities that will give you consistent test throughput and results, even when you're in unfamiliar territory.

01 GTIO ch01 1/28/05 1:30 PM Page 5

Unprepared

Nobody expects the Spanish Inquisition, and a lot of unexpected things will happen on your project. Expect the unexpected! Many parts of the game need to be tested at various points in the game's life cycle. Behind the scenes, many different technologies are at work—3D graphics, audio, user interfaces, multithreading, and file systems to name a few. If you are not ready for a variety of test assignments and don't have the skills needed to perform them successfully, then you will stumble rather than star.

Study, practice, and experience are ingredients for good preparation. During the course of the project, try to get to know more about the game code. Keep up with the industry so you are also aware of what the next generation of games and technologies will be like. Become an expert in the requirements and designs for the parts of the game you are responsible for testing, and then get familiar with the ones you *aren't* responsible for. When you least expect it, you may need to take on a different position, fill in for another tester, or grow into more responsibility. Be ready when it happens.

Note

The information in Chapter 5, "The Game Production Cycle," gives you a heads up on preparing yourself to succeed as a game tester, as well as covers what kinds of environments, projects, roles, and jobs you might find yourself in someday.

Under Pressure

Pressure can come from any of three directions:

- Schedule (calendar time to complete the project)
- Budget (money to spend on the project)
- Headcount (the quantity and types of people assigned to work on the game)

There's nothing to prevent one or more of these resources from shrinking at any time during the project. As a tester, these factors won't be under your control. Usually they are determined by business conditions or project managers. In any case, you will be impacted. Figure 1.1 shows the resources in balance with the scope of the project.

Moving in any one of these points on the triangle squeezes the project, creating pressure. Sometimes a game project starts out with one of these factors being too small, or they can get smaller anytime after the project has launched. For example, money can be diverted to another game, developers might leave to start their own company, or the schedule gets pulled in to release ahead of a newly announced game that competes with yours. Figure 1.2 shows how a budget reduction can cause pressure on the game project's schedule and headcount.

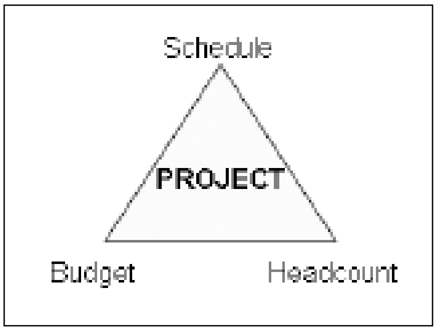

Figure 1.1 Resources balanced with project scope.

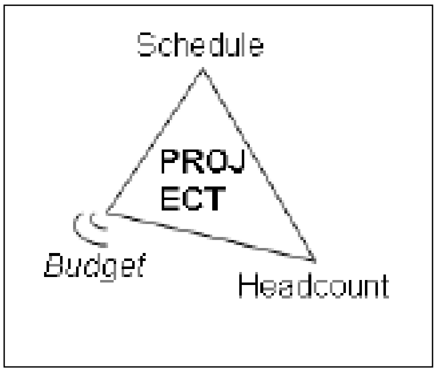

Figure 1.2 Budget reduction causes pressure.

Another way to cause pressure within this triangle is to try to stuff more into it than was originally planned for. This demand could be internally driven, such as adding more levels or characters, or scrapping the old graphics engine for a new one to take advantage of some newly announced hardware. Other unplanned changes might be made to support more game platforms than originally planned or to keep up with newly announced games in terms of number of levels, characters, online players supported, and so on. Figure 1.3 illustrates how increasing the scope of a project can put pressure on the budget and headcount if they are not increased.

When there is pressure on the project, you can expect it to get passed on. Someone demands something from you, and uses phrases like the following:

- I/we need … immediately
- I don't care…

- That was then, this is now
- Figure out how to do it
- Make it happen
- Deal with it
- We can’t afford to…
- Nothing else matters but…

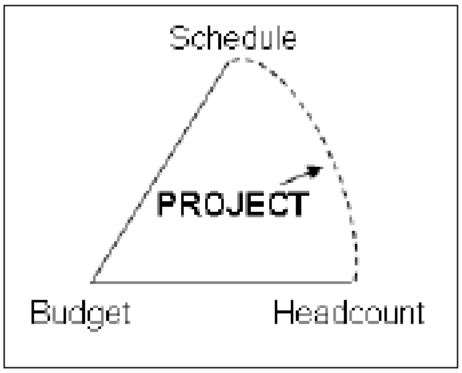

Figure 1.3 Budget and headcount pressure caused by project scope increase.

It’s likely that you will get more than one demand at a time, and from different people to boot. Examine the schedule, budget, and headcount available to you. Achieve the request by then scaling down what you would normally do so that it fits in your new triangle. Do the things that will have the most impact on meeting the request to the greatest extent possible. Then in the next release, take care of the stuff that didn’t fit this time around.

Note

Chapter 2, “Being a Game Tester,” introduces you to what’s expected of you in your role as a tester, and how to make an impact on the quality of the game.

Chapter 5, “The Game Production Cycle,” describes how goals and expectations change as the game progresses from being a concept to hitting the shelves, and how this affects the tester’s job along the way.

Unrested

Running long tests after staying up for 30 hours straight or working 100+ hours a week is not the best approach to finding defects. It is, however, a good way to introduce them! When developers do this, they keep testers in business, but it doesn't help get the game released. It's just as bad for the project when testers make mistakes.

Reporting a problem that doesn't really exist (for example, tested the wrong build, didn't do the setup or install properly, and so on) will send the developers on a wild goose chase and waste precious time. If you absolutely have to do testing late at night or at the end of a long week, make a checklist to use before and after the testing. If there's another tester around, have her check your stuff and you can check hers when she does her testing. Also, by writing down some of the information as you go along, you won't be prone to mistakes later on if you have to rely on your tired memory. It's kind of like a pre-launch checklist for the space shuttle. If something is wrong, stop the countdown. Go back and make it right—like it says in the test instructions. After testing is done, record pertinent results and facts. The next page includes an example checklist that you can start with and expand on to fit your own game projects.

In addition to putting practices into place for checking mistakes, look for ways to prevent them in the first place. Depending on your game platform and test environment, automation may be a viable option to make your work repeatable at any time of the day. Automated tasks can even go to work while you're at home resting.

Note

Chapter 16, "Game Test Automation," and Chapter 17 "Capture/Playback Testing," describe techniques and tools you can use to make the best use of your testing *and* resting time.

Nearsighted

Panic symptoms can include too much focus on the near term. Many game projects take months, so make that a factor in deciding what to work on today and how to do it. A question I will ask a tester to put him back in the right frame of mind is "Will this be our last chance to test this?" If the answer is "no," then we discuss how to approach the present situation in the context of an overall strategy of repeated testing, feedback from test results, budgeting resources, and so on.

Successful sports teams know how to avoid panic. When they are losing, they're confident that they can come back from behind and win the game because they are a) familiar with the situation, b) prepared to deal with it from practice, film study, and in-game experience, c) rested, and d) don't feel pressure to make up the deficit immediately. Teams that have a losing record often lack one or more of these ingredients.

Late Night Testing Checklist

Pre-Test

Do you have the right version of the test?

Test version: ________________

Are you using the right version of the build?

Build version: ________________

Are you using the right hardware configuration/settings?

Describe: ______________________________

Are you using the right game controller and settings?

Describe: ______________________________

Which installation options did you use (if any)?

Describe: ______________________________

Is the game in the right initial state before running the test case?

Describe: ______________________________

Post-Test

Did you complete all of the test steps in order?

Did you document the completion of the tests and the test results?

Did you record all of the problems you found?

If you reported a problem, did you fill in all of the required fields?

Note

Chapter 7, "Test Phases," shows you what kinds of testing should be done along the way as the game code matures. This helps you test appropriately for a particular situation and also know that you can rely on the additional testing you will do later in the game.

Trust No One

On the surface this sounds like a cynical approach, but the very fact that testing is built into the project means that something can't be trusted. You'll read more about this in Chapter 3, "Why Testing Is Important," and in Chapter 5. The very existence of testers on a game project is a result of trust issues, such as the following:

- The publisher doesn't trust that your game will release on time and with the promised features, so they write contracts that pay your team incrementally based on demonstrations and milestones.
- The press and public don't trust that your game will be as good and fun and exciting as you promise, so they demand to see screen shots and demos, write critiques, and discuss your work in progress on bulletin boards.
- Project managers don't trust that the game code can be developed without defects, so testing is planned, funded, and staffed. This can include testers from a third-party QA house and/or the team's own internal test department.
- The publisher can't trust the development house testers to find every defect, so they may employ their own testers or issue a beta release for the public to try it out and report the defects they find.

Don't take it personally. It's a matter of business, technology, and competition. Lots of money is on the line and investors don't want to lose it on your project. The technologies required to produce the game may not even have been available at the time development started, giving your team the opportunity to create the kind of game no one has ever done before. By trying to break the game, and failing, you establish confidence that it will work. Games that don't come out right fall victim to rants and complaints posted on the Internet. Don't let this happen to you!

Balancing Act

Evaluate the basis of your testing plans and decisions. Hearsay, opinions, and emotions are elements that can distract you from what you should really be doing. Using test methods and documenting both your work and results will contribute to an objective game testing environment.

Measuring and analyzing test results—even from past games—gives you data about your game's strengths and weaknesses. The parts that you trust the least—the weak ones—will need the most attention in terms of testing, retesting, and analysis. This relationship is illustrated in Figure 1.4.

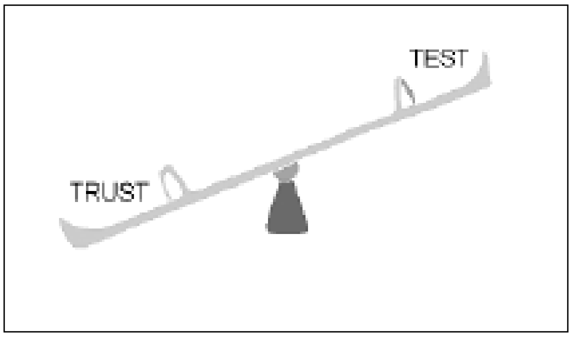

Figure 1.4 Low trust means more testing.

The parts you can trust the most—the strong ones—will require the least attention from you, as illustrated in Figure 1.5. These should still be retested from time to time to re-establish your trust. Chapter 5 helps you determine what kind of testing to do, and when to do it. Chapter 8 gives you specific strategies and criteria for planning, choosing, and revisiting testing.

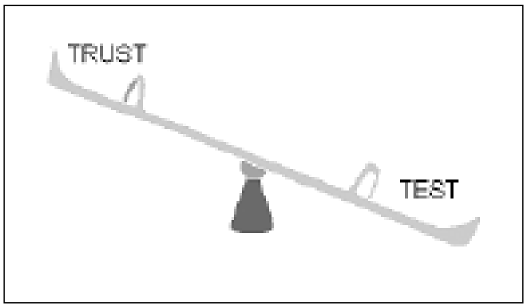

Figure 1.5 More trust leads to less testing.

Note

Chapter 6, "Software Quality," introduces you to some basic principles for evaluating the trustworthiness of your game code. Chapter 9, "Testing by the Numbers," describes measurements that you can compile from the test data you normally collect and explains how to analyze those measurements to zoom in on specific problem areas.

Word Games

It's useful to be wary of advice you get from outside the test team. Well-meaning people will suggest shortcuts so the game development can make better progress, but you won't remove bugs from the game by simply not finding them. Don't trust what these people are telling you. At the same time, don't cross the boundary from being distrustful to turning hostile. The whole team is working to deliver the best game it can, even when it doesn't seem that way to a tester.

A general form of statements to watch out for is "X happened, so (only/don't) do Y." Here are some examples:

- "Only a few lines of code have changed, so don't inspect any other lines."
- "The new audio subsystem works the same as the old one, so you only need to run your old tests."
- "We added foreign language strings for the dialogs, so just check a few of them in one of the languages and the rest should be okay too."

And some variants:

- "We only made small changes so don't worry about testing <insert feature name here>."
- "You can just run one or two tests on this and let me know if it works."
- "We've gotta get this out today so just …."

Tip

You'll be surprised how many bugs you will find by behaving opposite from the advice you get from other people about what should and should not be tested.

Don't equate a "trust no one" attitude with a "don't do anything you're asked to do" attitude. If a test lead or the project manager needs you to meet goals for certain kinds of testing to be done, be sure you fulfill your obligation to them before going off and working on the stuff you don't trust. The difference is between being a hero ("I finished the tests you wanted, and also managed to start looking at the tournament mode and found some problems there. We should do more testing on that next time around") or a zero ("I didn't have time to do the tests you wanted because I was getting some new tests to work for the tournament mode").

Note

In Chapter 4, "The Game Team," you learn something about what the other people on the game project are doing, and how your testing affects their work and impacts the progress of the game.

Last Chance

Examine your own tests and look for ways you can improve so you gain more trust in your own skills in finding defects. Just never let that trust turn into arrogance or the belief that you are perfect. Leave room to mistrust yourself just a little bit. Remain open to suggestions from managers, developers, other testers, and yourself. For example, if you're in the middle of running a test and you're not sure you are testing the right version—check it! You may have to go back and start over, but that's better than reporting the wrong results and wasting other people's time too.

As game development progresses, management and developers want to feel comfortable about the quality of the game and its readiness for the next milestone and, ultimately, final release. As a tester, you should not be lulled into complacency. I often re-energize my team by instructing them to "Treat this release like it's our last chance to find problems." Conflicts will arise about whether or not to introduce new tests, and you'll hear complaints about why important problems are found so late in the project. There are many reasons for late defects showing up that have nothing to do with incompetent testing. Here are some you will run into:

- The defects were introduced late, just before you found them.
- Bugs from earlier rounds of testing kept you from getting to the part of the game where the late defect was hiding.
- As you spend more time testing the game you become more familiar with where the defects are coming from, so it is perfectly natural that especially subtle problems might not be found until late in the project.

In any case, even if the bugs were there from the very first release, they were not put there by the testers. Somewhere there is an imaginary world where game testers get to hear "Thank you for finding this important defect right before we shipped it!" but don't count on that happening in our world (refer to Figure 1.6).

Note

Chapter 12, "Cleanroom Testing," and Chapter 14, "Play Testing and Ad Hoc Testing," give you methods to use for testing the game based on your testing intuition and insight.

Trust Fund

You can get a head start on knowing what not to trust in a couple of ways. Sometimes the developers will tell you, if you just ask…

Tester: "Hey Bill, is there anything you're particularly concerned about that I should focus on in my testing?"

Bill: "Well we just redid the logic for the *Fuzzy Sword* quest, so we definitely want that looked at."

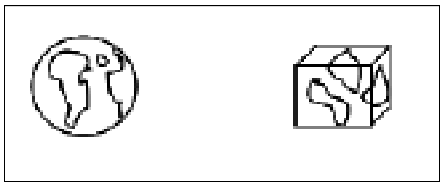

Figure 1.6 Our world (left) and the tester loving world (right).

You can get more clues by mentioning parts of the system and seeing how people react. Rolling eyes and pauses in response are giveaways that there is some doubt as to how good that new weapon will work or if multiplayer will work as well as it did before the latest changes.

Note

In Chapter 3, you find out why testing is important to the health of the game. It covers many factors that contribute to making things go wrong and how you, the game tester, should deal with them.

Give and Take

If you've been paying close attention up to this point—and you should as an aspiring or working game tester—you would have noticed an apparent contradiction between the testing approach to counteract panic ("don't treat this release like it's the last one"), and the "trust no one" approach of treating each release like it *is* the last one. A sports analogy might illustrate how these concepts can co-exist.

In baseball, one batter can't step up to the plate with bases empty and put six runs on the board. Instead, batter by batter and inning by inning, the team bats according to the situation, producing the most runs it can. The batters and base runners succeed by being patient, skilled, and committed to their manager's strategy. If every batter tries to hit a home run, the team will strike out a lot and leave the opposing pitcher fresh for the next inning.

At the same time, when each player is at bat or on base, he is aggressively trying to achieve the best possible outcome. He is fully analyzing the type and location of each pitch, executing his swing properly, and running as fast as he can once the ball is hit. He knows that it contributes to the team's comeback and that his one run or RBI could mean the difference between a win and a loss for the team.

So, as a tester, you can do *both at once* by following this advice:

- Know your role on the team based on the responsibilities assigned to you
- Execute your tasks aggressively and accurately
- Do the most important tests first
- Do the tests most likely to find defects often
- Make emotion-free and objective decisions to the best extent possible

Note

Chapter 15, "Defect Triggers," describes how testing causes defects to appear so you can cover those possibilities in your testing. These also help you decide which tests will be the most important ones to run and which ones should be run the most often.

The Rest of the Story

The rest of this book equips you to apply the two rules to your game testing. Don't feel like you have to incorporate everything at once to be successful. You may already consider yourself an effective tester. Use new insights from this book to refine your existing skills and add the techniques you learn in Chapters 10–17 where it makes the most sense for your projects.

You should also apply the two rules to what you read in this book. Don't trust that what you read here will work every time for everything you do. If you get results that don't make sense, find out why. Try something, and then measure or evaluate it to decide whether to go on using it and refining it, whether to try something new, or whether to go back to what you were doing in the first place. But do try it before passing judgment. I would just caution you not to trust yourself too much before you make sure you are applying the technique properly. Then you can decide if it works for you. These methods are good—trust me.

Remember, as a game tester, everyone is trusting in you to find problems before the game ships. Don't give them cause to panic!

Note

Chapter 10, "Combinatorial Testing," Chapter 11, "Test Flow Diagrams," and Chapter 13, "Test Trees," introduce you to three important game testing methods. Use them to understand the game software early in development and systematically explore the game's features and functions throughout the project.

Summary

In this chapter you learned two important rules for game testing, and how they relate to the remaining chapters of this book. Panic and trust are counter-productive to game testing. You can beat the system by remembering and applying the two rules.

Panic results in

- Poor judgment and decision making
- Unreliable test results
- Too much emphasis on the short-term

Panic costs the project in

- Unnecessary rework
- Wasted effort
- Loss of confidence and credibility

Avoid panic by

- Recognizing when you need help, and getting it
- Preparing for the unexpected
- Relying on procedures
- Getting sufficient rest

Don't trust

- Hearsay
- Opinions
- Emotions

Rely on

- Facts
- Results
- Experience

Test each game release as if

- It's not the last one
- It is the last one

Exercises

1. What is Rule 1?
2. Give two or three examples where you could have applied this rule but did not, and describe how results might have been different if you did apply it in one of those situations.
3. What is Rule 2?
4. Give two or three examples where you could have applied this rule but did not, and describe how results might have been different if you did apply it in one of those situations.
5. Which of the following puts pressure on a game project?

   a) New funding

   b) Taking people away to have them work on another game

   c) A requirement to add more levels to be comparable to a competitor's recently announced title

   d) b and c

6. EXTRA CREDIT: Name the two science fiction games that feature the phrases "Don't Panic" and "Trust No One."