Making Half-Life Mods - Part 1

by Robin Walker

Originally appeared in VERC Collective in Feb 24, 2003 and archived in TWHL since August 1, 2008.

(this article is presented as written for and published at Stomped.com, March 16th, 2001)

Robin Walker is the co-creator of the original Team Fortress mod for Quake. He was hired by Valve Software in 1998 and helped to design the company's official Half-Life mod Team Fortress Classic. He is now helping to design Valve's next announced game, Team Fortress 2.

In part one of this article, Walker talked about how to set up a mod team, designing a mod, and his «five week» plan on actually finishing the mod and releasing it. In this segment, he goes over every aspect of his five week plan from the build process to playtesting to actually finishing and releasing the mod and what to expect afterwards.

Five Weeks Out

Centralize Ownership

You should designate a single member of your MOD team as the Shipping Leader (SL). This person will drive progress on the MOD for the next five weeks. All changes made to the MOD from now on should occur only at the request of the SL, and all requests for changes should be funneled through this person. No team member should make any changes, no matter how minor, to the MOD unless the SL has requested that they make a particular change. This doesn't mean the rest of the team are losing control of the MOD… the SL is still a part of the team, and will be listening to all feedback. The point of the SL is to ensure that all changes to the MOD are going through a single person. This avoids problems such as a mapmaker breaking the game by making a last minute change because he didn't realize something else had changed in the game code. The SL will know the state of every component in the game (code, maps, models, textures, etc) at all times throughout the next 5 weeks to ensure this never happens.

Choosing the SL isn't easy. Here are a few tips:

- Don't immediately assume the person who's currently running the MOD is the best choice for SL, especially if the MOD has been worked on for months and hasn't got any closer to being released.
- Game Coders are probably the best choice. As the shipping process comes to an end, most fixes will be made in the game code.
- The SL should be highly motivated, disciplined, organized, and as ego-less as possible. The SL will need to be able to commit five weeks of his or her life to this process.

- The SL should be able to make global decisions for the MOD. The SL should understand that this often requires cutting features and content in order to ship.

Establish a Build Process

You need to create a process by which you build your MOD. Building is the process of taking all your work and producing an installable, working version of the MOD (generally in the form of an install file). This should be done exclusively by the SL for the next five weeks, and the SL should have a strict process that is always followed. Creating a strict process for this will ensure you don't waste hours tracking down bugs that are simply a result of someone building in a different way than the previous person.

The SL should maintain the final/release candidate version of the MOD from now on. All changes should be sent to him, and he should incorporate them into his copy of the MOD one by one and with a full understanding of the impact of the changes on all parts of the MOD. Don't forget to backup your code and content regularly!

The SL should build the MOD every day for playtests. More on that later.

Feature Locking

Shipping is the process of locking down portions of your MOD. «Locked» means that the portion is not to be touched from then on. Bugs found in locked portions of your MOD should be thought about carefully. Unless the bug is really important (showstopper), just note it down and fix it in the next update of your MOD. Regardless of the temptation to make that one «easy fix», unlocking portions of your MOD should be avoided as much as possible.

At this point in the shipping process, five weeks out, you should also be feature locked. This means you shouldn't be adding any new features to your MOD whatsoever. If part of your team is not involved in shipping but wants to continue working on developing the MOD, they should be working in a separate content and code database. Most source code control packages allow for branching of code in this fashion. (Yes, we strongly recommend that you use some form of source control). Every change made to the MOD from now on should be a bug fix. The SL should ensure this. Even if a coder thinks of another cool feature that they say will only take them 10 minutes to code, do not let them add it in. Even if the coder sends the code, finished and bug-free, to the SL, do not add it to the MOD. Save it for the next version.

A healthy attitude for the SL to have is that every change to the MOD from now on will add two days to the release date.

Playtesting

From now on, you should be running playtests every day, or every second day if that's too much. Playtests should be based on installable versions of the game, built by the SL. Don't let team

members play from their personal versions of the game... everyone should be running a version of the MOD installed from a build sent out by the SL (that's what your viewing public will be installing and using, that's what you should be testing). You'll waste many hours on finding bugs caused by incompatible versions if you don't do this.

To make this easier, the MOD must be kept in a playable state at all times. Get very, very worried for every day the MOD isn't playable. If a coder or mapmaker makes a change that breaks the MOD, think about it carefully before incorporating it into the SL's build. How long will the game remain unplayable? How many playtests will you miss? How many team members won't be able to work because the MOD isn't running? Not breaking the MOD should be religion for the team.

When you do playtest, make sure as many of your team members are playing as possible. Everyone working on the game should be playing it regularly. Make sure you have some external players as well. Turn on server console logging (set «log» to «on» in the server.cfg file). This will dump all the output of the server into a file in the gamedir/logs directory (the name of the file will match the date). Whenever any player in the game spots a bug, have them use their «say» key to say «BUG: description of bug». Then, when the game is over, you can open up the log file and get all the bugs out of it by searching for the word «BUG.»

Bugs /Changes

The SL should maintain a complete list of all bugs and changes, and their current status. Preferably this should be done using some kind of true database. E-mail is insufficient for tracking bugs; it's just too easy for items to drop off the first page of a user's mailbox, etc. After each playtest, the bugs and necessary changes from the log file should be added to the list by the SL, and assigned to team members. When a team member has fixed a bug / change, they should submit the new content to the SL, who should verify that it is fixed and then update the status on the bug list.

The bug list is a fantastic tool to evaluate how well you are progressing. It can be used to find out who is overloaded with work, who is underloaded, who is not fixing his bugs, which area of your MOD is farthest from completion, and so on. Don't remove anything from the bug list, even when it has been fixed (though you should mark it as fixed in some way, of course). It's very useful to see what bugs have been fixed throughout the history of the project. Something might regress, re-creating a bug, and knowing who fixed it last time makes it easy to ask them what caused it. At the end of the project, you should be able to see every bug fixed and every change made in your MOD for the entire shipping process. The SL shouldn't allow any bug fix or change into the SL's master copy of the game unless the bug / change has been detailed in the bug list.

There is software that will help you create and maintain a bug list. Alternatively, a spreadsheet will work just fine. Again, e-mail is a bad choice.

Cut / Defer Broken Features

The hardest, nastiest, and unfortunately most necessary part of shipping is the act of being realistic and cutting features. We have a saying at Valve that everyone will have their favorite feature cut from the game. While it's not true, it does help everyone prepare themselves for the fact that they will have features they like - or that they spent some to a lot of time on–cut. Your game simply cannot

have every cool feature and still ship in a reasonable time frame. The SL should make decisions about what to finish and what to cut, based on how far along in the release process you are. The closer you get to releasing, the more you should think about each bug as you find it. Is the bug in a feature that absolutely must be in this version? How many days will it take to fix this feature? Can this feature be cut, or deferred to a later version?

Work Smart, Not Hard

As we've said over and over again, the shipping process is hard, and it's even harder if you don't think carefully about what to work on. Working a lot is no substitute for carefully choosing what to fix, what to defer, and what to cut out altogether. The SL should be extremely careful about which bugs / changes should be worked on, and by whom. Don't spend a week fixing a minor problem in a feature just because the feature is cool. Fix crash bugs (showstoppers). Fix bugs that utterly prevent you from shipping the game. Fix bugs that are preventing other team members from fixing their bugs. The SL should develop categories for bugs to aid in making the right decisions. A good level of granularity is Must Fix, Severe, Medium, Minor, Zero, Deferred.

As the project gets closer to shipping, the SL should be carefully evaluating every bug that shows up. Remember, every bug that's fixed creates more playtesting, and usually more bugs. If you are two weeks from your release date, and you've got a bug that will take someone three days and 500 lines of code to fix, you're not going to make that release date unless you cut or defer that portion of the game.

Three Weeks Out

Content Locked

By now you should aim to be content complete. This means that all content in the game is in a locked state, except for the game code itself. All the maps, models, textures, sounds, HUD art, launcher art, and so on should be finished and in the SL's master copy.

Shutting Down

This was mentioned at the five week mark, but it's even more important now. The SL is the only person who should be touching the master copy of the game, and he should simply be rolling in the bug fixes from the coders, who should be fixing only the bugs the SL hands them.

Playtesting

The MOD should be being playtested every day, for at least two hours. Between now and when you ship, you want as many people as possible hammering away at your MOD. It's too late now to make any major game design changes - don't even be tempted.

One Week Out

No last minute changes

The SL should be evaluating every change that has to be made, and deciding whether they should be deferred to the next version. Again, a healthy way to think about it is that every single change, even if it's a single line of code, will add two days to the release date.

2 day Safe period

Once every bug that is going to be fixed has been fixed, and everything else has been deferred, you're not done. Now you wait at least 48 hours, during which time you should playtest like crazy. Try to get everyone hammering away at the game for as much time as possible. If you find any more bugs that have to be fixed, fix them, and start the 48 hours again.

If your MOD passes 48 hours of heavy playtesting without finding any new issues, you're ready to release!

POST-RELEASE

So you've released, the players love it, and web pages everywhere are talking about how much fun your MOD is. Whether you're done now is up to you. From our experiences in the on-line multiplayer field, a MOD only stays popular as long as it's supported. No matter how great your MOD is, it's not going to garner really significant player numbers with its first version. Player numbers are grown over time through repeated releases of new content, bug fixes, and of course, community support. Both Counter-Strike and Team Fortress started out small and grew over time. Each time they released a new version, more players tried them out and started playing them.

Knowing what to fix, what to change, and how to listen to your community is a continual learning process. Good luck!

<Back to part 1