A Systematic Approach to Level Design

Source Engine Case Study

By Will2k

I-Introduction

Level design is still a relatively new form of artistic expression considering it only broke into mainstream in the mid 90's with the release of modding tools for Doom and Quake.

However, level design quickly evolved as the gaming industry itself matured and transformed into multi-billion dollars corporations. Games became more complex, gamers became more demanding and the competition became fiercer to attract more players thus generating more revenues.

This change forced level design to move from the chaotic hobby it started as, to a discipline that is taught in universities where one can now graduate with a degree in game design. Gone are the early days where a hobbyist mapper could slap together a couple of rooms connected with some corridors and expect players to be amazed. The level designer nowadays, on top of being an architect, has to be also a 2D artist for materials manipulation, an entry level programmer for basic scripting and AI manipulation, a photographer for colors and lighting and an engineer for the quantitative capabilities of the engine he/she's working with. That seems like a lot to do but with a systematic approach everything will be put in place with minimal hassle and headaches.

In this paper, I'll try to depict my way of approaching level design. I'm not saying that I know better than someone else nor I'm saying that this approach is the only one valid; on the contrary, I, myself, still have a lot to learn. I will not be teaching how to use the level editor as many excellent tutorials are available online. I'm only presenting my way of doing business; an approach that served me well since I started level design some 14 years ago.

I started playing games as a kid in the mid 80's with the original Atari and started tinkering with computers as a teenager in the mid 90's. I was comfortable working under DOS and writing code in GWBASIC and QBASIC. I was a fan of shooters, from the 2D side-scrollers on the Atari and Sega to the early First-Person Shooters on the PC (Doom, Quake, Duke Nukem…). However, it was in 1998 that my gaming life changed and the level design bug bit me after playing 2 titles: Half-Life and Delta Force.

Along the years, I dabbled with most game engines and some maps were never released. Most of my early work was with the GoldSrc engine (HL1) and recently the Source engine (HL2) but in between, I fiddled and created levels with Voxel engine (Delta Force series),

IdTech3 engine (Quake 3), UE1 engine (Unreal Tournament) with most of the maps either lost on the internet or lost on my former hard drives.

No matter what the game or the engine are, the essence of level design remains the same and I'll try to illustrate what I accumulated in those long years in the hope of laying solid ground for aspiring beginners as well as offer a good read for experts and veterans.

II-Project Life Cycle

II.1-Planning

This is by far the most overlooked and underrated aspect of level design. Beginners will say: I'm doing this for fun; why would I need to plan and do the serious stuff? Here's a paradox for you: Gaming is a serious business. Make no mistake that while gaming is fun and is a form of entertainment for the gamer, it is a serious business for the developer and requires discipline, talent and solid management systems to deliver quality products that are satisfactory to you, the player.

It's a tedious thing to do at first but once you get used to it, it becomes second nature and you won't look back. Without proper planning, your project will stall after a rush start and either it will slowly die because of the lack of interest and ideas from your side or it will be finished after prolonged phases and cycles of development and the end result will be incoherent on many levels (gameplay, design…). This is why many beginners steer away from level design after trying it for the first time.

First thing you need to do is to brainstorm ideas: Gather as much ideas for maps as you can from different sources (movies, TV shows, books, trips you took, city you live in…). At this stage nothing is silly and everything is allowed. Once you have enough ideas, start the screening process and eliminate ideas that are pointless or vague. You need to also eliminate ideas that don't translate well into a map: for example, you might want to recreate your neighborhood or a river that flows near your town; These maps might be visually appealing but they won't fare well in gameplay due to several reasons (open-ended area, complex layout, too much detail, ridiculous amount of textures and models to be implemented….)

Once you narrow it down to 2-3 viable ideas, it's up to you to choose one for the upcoming map. After choosing the idea and theme of the map, you need to start drawing a sketch of the layout. You heard me right: you can use plain old pencil and paper or if you are a fancy graphic designer, digital pen and touchpad. You don't need to put every detail on the paper, rather just put the building blocks that define the routes and write down every design note that comes to your mind on this same paper (width of road, height of building, tree type, lighting intensity and color, time of day, textures and building shapes...)

Please note that this layout might become final or might be heavily modified at later stages, but it is a very important tool to keep you on the right track during your project life cycle. It is also extremely important not to start with a flawed or unplayable layout as it will be very difficult to adjust later on once the building phase has started. Your best bet is to carefully study the layouts of official maps as well as popular custom maps to find out what works and what doesn't for each scenario (hostage rescue, bomb/defuse, capture the flag, deathmatch…)

The planning phase can take from 1 day to 1 week. Keep in mind that it's more cost-effective to spend 1 week planning and changing ideas and layouts than to rush your project, run out of ideas, find it difficult to implement late changes then let the project die.

II.2-Assets Preparation

Up to this stage, you have not touched your level editor yet and you won't until after you finish this second phase of the project.

While many level designers advocate the method of building the level using "dev" textures at a first stage, I find it more natural and streamlined to start building using the actual textures, not to mention the motivation boost you get after seeing your textures come to life in the early builds of your level.

In many cases, the texture itself can dictate the shape of the building. A composite or an asymmetric texture can be a catalyst for your creativity as the brush having this texture will start taking shape in your head even before creating the actual brush in Hammer editor. As you can imagine, this won't be the case with a monochrome, flat "dev" texture.

Now that you know the theme and approximate layout and shape of your level, it's time to start gathering reference photos. These photos will be a tremendous help for creating or searching for the adequate textures and models and for outlining the shape of the geometry in the level.

After you get all the reference photos needed, it's time to start building your materials and models. The best option here is to make your own textures and models: If you are proficient in 2D photo editors (i.e. Photoshop) or 3D editors (i.e. 3DS max), then by all means create your own assets for your level since you are the most qualified person to know what you actually want in your map.

However, if you are not skilled in 2D/3D editors, you are left with 2 options that are not bad after all: use the standard assets that come with the game or get custom content from dedicated communities' websites. Obviously, you will have to do some compromises with the last 2 options since the assets might not fit exactly with your map but, it's a fair trade-off for the hassle of creating your own textures/models.

I need to emphasize one important point: do not waste your time by creating new textures/models if something very similar can be found in the standard game assets or is freely available on communities' websites. Only resort to creating new content for your map if you are pretty sure that nothing similar exists, as this phase can be time-consuming and should not exceed 2-3 weeks. If you feel that this part is dragging then you need to rethink your map and switch some of its parts to use standard or existing assets.

One last thing to keep in mind about custom content is that too much of it will noticeably increase the size of your finished level. Many players will not download a huge map file unless they are sure it's a quality map and client computers might have problems downloading the map from servers.

II.3-Level Building

Now that all the tools and assets are ready, you can open Hammer editor for the first time. You need to keep the reference photos open and readily available and the layout sketch and notes laying on your work desk for continuous consultation.

The strategy I use is to divide the map into small parts, then build and test each part before moving to the next part. This strategy originates from my own background in industrial engineering and project & operations management. It is easier to manage and control a project in small steps and increments than to be overwhelmed by it as a whole.

There are also many benefits from building the level in this way:

- You will build faster as the focus is on one particular area and you won't be distracted by other aspects and areas of the map.
- You can assign realistic and short time tables for building each small part of the map. Short deadlines will boost your productivity and give you something to look forward.
- Building and compiling each part alone will give you an early sign of what's working and what's not in your level (models, textures, lighting, skybox, architecture…) instead of being surprised after months and months of building the map without compiling.
- Building and compiling each part alone will allow you to sort and fix bugs and issues with the particular map part before moving to the next part. This could be a real bonus as you know that your map is gradually growing and is bug-free. There is nothing more frustrating than building the full map for a long time, compiling in the end only to find a gazillion bugs and error messages and then feeling helpless and depressed not knowing where to start. This can be serious as many mappers will give up at this stage and release the map as is only to get negative reviews and flaming, further aggravating the situation for the beginner level designer.
- You will get the necessary boost and self-motivation to work on the next part after compiling and play-testing each part at a time. You won't be able to wipe the grin off

your face after seeing your early level builds in action; this will open your appetite for further level building and anticipation for the next build play-test.

To illustrate this method, I shall use my map cs_east_borough as an example (download link at the end of this paper).

First, I created the basketball court (CT spawn) and the area surrounding it, then I concentrated on building the roads network to define the boundaries for the map. Next was the central area surrounding the back alley followed by the garage (T spawn) and the surrounding area. I moved on to the lateral streets buildings and ended things up with the 3D skybox geometry.

After you finish each area, seal it with a final skybox for the parts where the geometry is done and with a temporary skybox for the part that still needs to be connected to future unfinished parts. It's important not to wrap the map up with a skybox cube since it will conceal potential leaks and will give erroneous readings for visleaves and frame rate. You need to compile your completed and sealed part and get ready for a play-test and bug hunt. A good thing to do at this stage is to keep a copy of the compile log (text file in notepad); do this for each map part you compile even if you end up with 20-30 log files (you can delete them after releasing the map). This is important to keep track of the map progress and its cost on the game engine (entities, brushes, lighting data…) and to pinpoint bugs locations and trace back their origins to the corresponding part of the map.

As I mentioned earlier, play-test your build: check the frame rate, visleaves' cuts, lighting, models, entities and error messages in the compile log and game console. Only move to working on the next part after you have sorted out every issue with the current part. Take a moment to admire what you accomplished in this build and give yourself a pat on the back. This motivation is needed to get you started on the next part of your level.

I already said that I work directly with the final textures and not with "dev" textures, but it is also very important not to go too much into details at this stage. Start building the current map part using the actual textures for the major building blocks, add important props like trees, lamp posts, dumpsters, cars; also add major detail that contribute to gameplay like walls, ramps, stairs and tweak the lighting to its final state but refrain from going on a detail spree yet. Don't start adding decals and overlays, small static and physics props, foliage detail and practically anything that doesn't contribute to gameplay but rather to visual appeal. You will be losing precious stamina and time that should be used towards building the next part and getting closer to finishing the map. Don't worry about the details and nailing the perfect look as you will have enough time during the final art pass after you complete the core of the level.

During this phase, the layout can be altered and changed to better suit the growing map or to remedy a flaw or bug found during the playtest. We are talking here about minor changes to enhance an already sound and working layout: widen a road, add a connecting hallway, add access to second floor, etc. And as I stressed earlier in the planning section, it is very important

not to start with an unbalanced and flawed layout, as the change in this stage becomes very difficult and would eventually lead to scraping the whole map and rebuilding again with a new layout.

It is also worth mentioning that in case you need to test some concept that you are not sure of its outcome, it is wiser to build a small test map and implement this concept than to directly integrate it in your map. The test map will compile in 4-5 seconds which gives you lots of time and freedom to toy around and tweak your concept as opposed to your main map that will need a noticeable time to compile and will lead to lost time and productivity, not to mention potential unintentional damage to your map.

It also goes without saying that you need to back up your map source file (the vmf) after each partial compile in case you screw things up and want to revert to an earlier version.

If for any reason you have a creativity block (and let's face it, it's likely to happen), save your work, leave your workstation and have a brain reboot. This can be done in several ways depending on the type of person you are: have a conversation with the wife/girlfriend, call your parents/relatives, pay a friend a short visit, take your car for a quick drive, take your camera for an impromptu photo-shoot, watch some TV/DVDs, listen to some music…It really depends on you but the goal is to let your brain have a cold reboot and get away from the map. It will surprise you that creativity will flow back without notice while you are performing one of the above tasks that relieve your brain.

The timeframe for this phase depends on your motivation and available free time and can't be determined realistically. It's not uncommon for small maps to be done in less than 2 weeks while large projects might take 6 months to 1 year to finish.

II.4-Final Art Pass

Now that the map is almost done, it's time to get your reward. Go ahead and unleash your creativity at this stage and choose which detail goes where. This stage is mainly cosmetic but can make or break your level as a nicely decorated map can make all the difference.

Take your time to refine and maybe replace textures, add small details, decals and overlays, special effects and special lights, etc.

I can spend 15 minutes just to find the right decal to place on a certain wall furiously swapping different decal models; then I can spend another 5 minutes to find the optimum place on that wall (true story ☺…I'm a perfectionist and attention to details is second nature)

You don't need to spend this much time but you get my point of carefully choosing your detail level during this phase which shouldn't take you more than 2-3 days.

II.5-Optimization

My favorite topic and another overlooked aspect of level design.

At this stage, your map looks cool, plays well and runs at a decent frame rate. However, this map is not destined for you to play on your PC alone; it will be released and shared by thousands if not millions of players. Keep in mind that different people have different specs, so you need to make sure that your level runs satisfactorily on a wide array of configurations.

If your level is for a multiplayer game then you really need to make sure that your map is ready for the lag that will eventually be caused by high ping, client-server network and the number of players on the server.

There are plenty of technical tutorials describing in detail the process of optimization in Hammer and the Source engine, so what I will do is describe how to approach this process in a systematic way that works for each map.

As surprising as it may seem, optimization starts with the map layout itself. A poor layout can be a hell to optimize while a well-thought layout can pass through with minimal optimization. If possible, try to avoid unnecessary open areas with unbroken line of sight (LOS), implement corners to break LOS, add solid structures like buildings or walls to also break LOS.

Always build your level following the grid in Hammer and whenever possible, make your building blocks of standard sizes. As an example, a building of 512 units high can be made of 4 blocks of 128 units each. This will allow the engine to cut the visleaf horizontally and will give you greater flexibility for texture application.

Your next step would be to lower the skybox as much as possible without hiding geometry or models and without hindering physics objects and being unrealistic. You will also want to fill the top of buildings with skybox brushes to reduce LOS as well as the number of visleaves generated by the engine.

The brush faces that touch the "void" outside your map will be culled by the engine whereas the faces that are inside your level but can't be seen by the player will still be drawn and rendered thus putting additional load on the engine. I can't stress this enough: apply the "nodraw" texture to any surface that the player won't see during normal gameplay (I'm not including observer cam or by using noclip). Don't forget faces from the big roofs to the tiny rails on the balcony as every nodraw will help improve performance.

Use func details as much as possible. Anything that does not constitute the backbone of your level like buildings and floor and anything that is not used to break LOS should be switched to func detail. Don't get alarmed if large portions of the map become func details as long as the level is properly sealed with regular brushes (func details and props don't seal the level and will cause a leak). Func details won't create new visleaves thus reducing the load on the engine; in

addition, func details are internal entities that get "consumed" during compilation. This means that they don't exist as an entity during gameplay further reducing engine overhead.

Props can be optimized by setting their fade distance in their properties. Care must be taken not to fade critical props though as this can impede gameplay. My map cs_east_borough is again used as an example: there are cars and dumpsters on the streets that will be used by players for cover, so it's important for these props not to disappear from a distance otherwise the player taking cover behind them will be exposed to the enemy players. Needless to say that the element of surprise will be lost and the player will be really pissed off. On the other hand, small props are a prime candidate for fading such as trash bins, light bulbs, fire hydrants, lamp posts since most of them are non-critical to the gameplay and probably won't be noticed by the player when he is at certain distance from them.

The next step in your optimization process is to use hint brushes. These brushes are regular world brushes with the "hint" texture on 1 or more faces and "skip" texture on the remaining faces. It's called hint because you are actually giving the engine a hint to cut the visleaf exactly along the "hint" face. It is always wise to add at least 2 horizontal hint brushes for open-ended maps (one at 128 units, another at 256 units and more can be added as necessary) to prevent having tall visleaves that "see" each other from across the map ("see" here means having a direct LOS).

Hallways, doors and windows should have a hint brush at their edge to prevent visleaves from poking from one side to the other. Corners should have a triangular-shaped hint brush, typically at 45°, to prevent the visleaves on both sides of the corner from having a direct LOS. Keep in mind that all this is done to control the visleaves' cut and decide what the engine renders at any location in the map. Your ultimate goal here is to make a specific visleaf "see" the least amount of adjacent leaves thus preventing the engine from rendering the content of these "unseen" leaves which will reduce engine overhead and increase frame rate.

To see hints in action and rendered visleaves in-game (red boxes in space), open your game console, enable cheats (sv_cheats 1) then type mat_leafvis 1 (to see individual visleaves) or mat_leafvis 3 (to check how many visleaves the visleaf you are standing in is seeing). This will help you decide whether your hint brushes are doing their job or you need to alter some and add some others.

Sometimes hint brushes are not enough to control visibility and that's where areaportals come in handy. Unlike hint brushes, areaportals are brush entities that have the areaportal texture on all faces. They are usually placed on doors and windows, but can exist anywhere in the map. Typically, I use them at the end of hallways, corridors and even in open areas for separating streets. Areaportals need to seal an area completely or they will cause a leak.

Areaportals will hide both world geometry and models; they are a wonderful tool but they need to be carefully manipulated. In the case of closed doors, the linked areaportal state is set to

“closed” and nothing behind it gets rendered by the engine until the door is opened. In the case of hallways, windows or open streets, you can’t set the areaportal state to “closed” obviously as the player will see the void behind it which is totally unrealistic and level-breaking. Nothing is lost however, as you can set the state to “open”. Despite not hiding the world behind it, the open areaportal will present a very nice effect called culling. Instead of hiding everything behind it, the areaportal will selectively hide the world that you, the player, can’t see from your position. An example in cs_east_borough is the garage where the terrorists spawn. The main entrance has an open-state areaportal. If you place yourself in front of the repair shop, everything inside will get rendered (and that’s normal because you actually see the inside). Now if you start moving sideways, you won’t be able to see some parts inside because of the side wall of the garage. Because of the areaportal and its culling effect, the engine also won’t render these parts that you don’t see. Without the areaportal, the engine would still render the inside even though you can’t see it.

You can clearly see the tremendous benefits of areaportals in controlling visibility and forcing the engine not to render parts of your level which will translate in higher frame rate and happier players.

To check areaportals in-game, type r_drawportals 1 in console and all areaportals will be highlighted in green. To see their effect on visibility, type mat_wireframe 1 and wander around near areaportals. You will start noticing parts of the world appearing/disappearing as you move around and that’s good news; your areaportals are working. Using mat_wireframe 1 is also useful to check the effect of hint brushes in the same way too.

The last resort in optimization is by using occluders. Like areaportals, they are brush entities that have the occluder texture on the occluding face and the nodraw texture on all other faces. Occluders are used to hide models only and not world geometry. They are very costly for the engine as they work in real-time to decide what models get rendered and what don’t based on your relative location to the occluder brush. I usually don’t use occluders since I manage my optimization by using all the above techniques, but sometimes you will be forced to use them when everything else fails to cut visibility and improve frame rate. A good example of using occluders is found in cs_assault. The map has several occluder brushes mainly inside the containers and inside the warehouse roof. The last one is interesting because when you are on the roof, the occluder will prevent every model inside from rendering. Without the occluder, the engine will render all the models inside even though you don’t see them and that would be a big waste of resources and will inevitably lower your frame rate.

As I said, occluders are costly for the engine. Keep trying with other optimization techniques over and over again until you are sick and exhausted and can’t improve the frame rate anymore; then, and only then, use occluders.

To check them in-game, type r_visocclusion 1 in console and all occluders will be highlighted in white. The hidden models will be green and the unhidden will be red. To check the effect of occluders on your frame rate, type r_occlusion 0 to turn off occlusion. If there is no noticeable drop in your fps, then your occluder is useless and eating up unnecessary resources and is better removed. However, if after turning off occlusion you notice a severe drop in fps, then your occluder was really needed and was doing a good job cutting visibility and maintaining high frame rate.

To check how is your optimization effort faring, you can type +showbudget in console (-showbudget to deactivate). This command will display your real time frames per second, your map ping in addition to a detailed graph showing the engine cost of each of your level's components (world brushes, props, physics, sounds, vgui, etc.). This can prove very useful if you are experiencing a drop in frame rate at certain locations in the map and you are not sure why it is happening. The highest bar in the graph will let you pinpoint which component is the culprit in sucking engine resources.

The optimization phase, depending on the level size and complexity, can take anywhere from 1 day to 2 weeks.

II.6-Fine Tuning and Release

Your level is almost finished and now comes the boring and tedious part of quality assurance and testing. At this stage you will probably be both exhausted and excited but don't give in to the temptation of releasing the map yet.

Fire up the map and start wandering around as if it was your first time seeing it. Admire your accomplished work and feel proud that you actually finished the project within a reasonable amount of time.

Now it's time to put the warm feelings aside and switch to "objectivity-mode". Inspect every little area in the map for visual glitches, error messages in console, bugs that were overlooked in earlier builds, new bugs in the final build. Do not let anything slip by even if that means that you have to do several runs in the maps. If you followed my approach exactly, then this step won't take too much time since most of the glitches and bugs would have been polished earlier during the incremental builds but still, this inspection is needed to make sure everything is perfect.

Once satisfied with the results, you have to pack your map with all the custom content for pre-release and early playtests. You need to include the custom materials, models, sounds, radar overviews, bots navigation file and possibly anything you modified for the level. Use a packing utility (i.e. Pakrat) to embed all the resources in your bsp file for neater and easier download (or vpk in other cases).

It is nice to include a readme file to let people know a couple of things about the level or if they need to follow special instructions.

For the playtests, it is best if you have access to a public server where people you know/trust can privately test your map and give you feedback. That would be the closest simulation to what will happen in the real world once you publicly release the map. However, if such access is not available, then you can still revert to good old LAN playtest with as much friends as you can gather.

Collect every bit of feedback from all players and start the screening process. Some points might be subjective or dumb while others might be very useful and helpful. You will decide what points to take into consideration and what to discard.

Focus first on the points that are gameplay-breakers then move on to the other points that are cosmetic. Once you implement the changes, go for a second playtest and even a third if necessary until everything is polished and everyone is happy.

Don't forget to test the map with bots (if supported) to make sure that the provided navigation file is bug free.

This phase can go between 1 week and 1 month depending on the playtests and bugs in your level.

With all that work behind you, you can now release your finished level and upload it to online communities' sites to share with other players. Keep an open mind and be ready for constructive criticism and feedback especially from veteran level designers who will give you useful advice regarding your map. And since it's the internet, be prepared for some trolls who will flame you. Be polite, ignore them and don't let them depress or frustrate you. Your journey into level design has just begun.

III-Conclusion

As I stated in the introduction, my aim was not to teach the technical details of level design or how to use the level editor but rather to present a systematic approach that I use myself to help beginners to have a firmer grip on their level design endeavor.

I hope that by following this standard operating procedure, rookies will have something to build upon and will be able to finish their projects and improve the quality of their released levels.

If any of the terminology seems awkward for some beginners, then I strongly advise you to start reading articles from the links at the end of this paper to familiarize yourself with the technical words and concepts.

Level design is still half-art and half-science, and I hope I have shed some light in this article on the science and methodology part.

Level design is about perseverance so my last advice to beginners: don't give up; keep trying and you will eventually reach your goals.

Will2k

March 12, 2012

Contact: **will2k@hot-shot.com**

Website: **http://will8k.tripod.com**

Useful Links

http://developer.valvesoftware.com/wiki/Main_Page Valve Developer Community

Official site of the Source engine wiki and tutorials. Start here

http://www.gamebanana.com/ GameBANANA

Community site with custom content, maps upload and tutorials

http://www.interlopers.net/ Interlopers

Community site with tutorials

http://www.gamebanana.com/css/maps/164363

Direct download link for my map cs_east_borough