Reveling in Restrictions: Technical Mastery and Game Boy Advance Homebrew Software Development

Brett Camper
Comparative Media Studies, MIT
bcamper@mit.edu

ABSTRACT

Nintendo's Game Boy Advance handheld game system has attracted a sizeable online community of amateur or "homebrew" developers. These programmers create games outside of the official, professional context. With an emphasis on "mastering" the technical specifics of this small machine, homebrew games adhere to aesthetic guidelines influenced by but modified from those of the commercial market. This paper examines these homebrew games and the discourse around them to show how amateur developers use technical limitations to motivate and shape the production of their own work, and the ways in which they perceive the work of others. A loose set of categories for understanding Game Boy Advance homebrew programming is identified.

Keywords

Game development, amateur software, homebrew, demos, Nintendo, Game Boy Advance.

1. INTRODUCTION: THE GAME BOY ADVANCE HOMEBREW SCENE

Nintendo's Game Boy Advance (GBA) is one of the most popular handheld video game systems of all time, with over 65 million units sold through the end of 2004. In fact, continuing the benchmarking against Hollywood box office sales so common in journalism and scholarship about the game industry, one report placed Nintendo's total 2003 revenue from the GBA at $1.5 billion, higher than that of any single movie studio [11]. Given its prominence, it is unsurprising that the machine has attracted a substantial community of amateur, independent developers.

These amateurs refer to their hobby as "homebrew" development, and collectively have made available several hundred titles over the past two years through community websites gbadev.org.

The GBA hardware on which the games are played is proprietary and dedicated to games. This distinguishes GBA homebrew from much other amateur game development. A hardware platform is considered to be functionally "dedicated" if it is narrowly intended for a specific use or set of uses. For instance, a decade ago, a cell phone would have been considered a device dedicated to making phone calls, but with considerable diversification over the past few years into other forms of communications and information retrieval (text messages, news, sports and weather reports, calendars, games, etc.) we can no longer think of cell phones as primarily dedicated to voice communications in the way the landline telephone has maintained this primacy.

Developing for a dedicated games-specific console like the GBA as opposed to an open, multi-purpose platform like a personal computer introduces unique challenges of access that are pivotal in understanding the technological bounds against which the homebrew scene operates. Nintendo seeks to keep the GBA a "closed" system – they do not publicly provide detailed information on how to create software for the system. Instead, they reserve such information for commercial developers whom they approve. For the non-professional homebrew scene, the primary consequence of this closed technological and social system is a lack of access. Homebrew GBA development takes place unofficially, without Nintendo's consent.

The homebrew scene is based almost entirely online, and revolves around a small network of websites. Because the practice is not a "standard" use of Nintendo's video game console, hobbyists are involved in a substantial amount of reverse engineering and tinkering with the system's hardware and technical internals, such as its ROM layout and BIOS. Websites like gbadev.org, the hub of the scene, are therefore crucial for the exchange of "hacked" technical information, program source code, and tools for manipulating graphics and sound data. Social reputation within the community is governed by personal demonstrations of such technical mastery. GBA homebrew development is one of the latest in a long tradition of hands-on, young male-dominated past-times: early 20th century amateur "radio boys," the garage rock and hot-rodding movements of the 1950s and 60s, and the creation of the original Apple computer in the mid-70s via the Homebrew Computer Club (the terminological origin of today's "homebrewers").

2. LOW-LEVEL PROGRAMMING: SITUATING TECHNICAL MASTERY

Technical discussion topics on gbadev.org outnumber those relating to graphics, sound, game design, and narrative combined by a ratio of roughly five to one.[1] When asked why they program for the GBA, homebrewers will commonly mention technical appeal:

[1] This is based on my own count of forum topics and posts, conducted Jan. 5, 2005, at <http://forum.gbadev.org>.

Lord Graga: "…the hardware itself is extremely interesting to a lot of people. It's simple, yet you can do some rather advanced things with it... It's a certain satisfaction to sit down and start writing a game/demo/whatever for this machine, since you are so close to the hardware."

dagamer34: "You have complete control over the hardware."

notb4dinner: "I bought my GBA SP to kill time on the train and it was just such a cute little machine I couldn't help but have a go at programming it. Pleas[a]ntly it's just as cute on the inside as outside" [1].

GBA homebrewers are unusual game consumers in that they are primarily interested not in the intended, external function of the machine, but instead on its internal, "hidden" electronics.

Technical mastery can be distinguished from the more general category of programming skill by its insistence not simply on the behavior and construction of software (in this case computer or video games) but more importantly on the underlying electronics hardware that physically enables that software to run. This is an important distinction – the practice of software development is increasingly dominated by "high-level" tools that direct the computer's actions independent of any knowledge of specific hardware. Object-oriented programming languages such as C++ and Java, and software APIs (application protocol interfaces) are built on a principle of abstraction away from hardware, emphasizing *what* behaviors are possible, not *how* such behaviors are carried out by the machine.

Creating software for a machine like the GBA carries a prerequisite of technical mastery that necessitates a return to the focus on hardware more common in an earlier era of game programming. Today most common game development platforms available to hobbyists follow the trend of de-emphasis on hardware specificity in favor of portability; this is true of both commercial packages like Microsoft's DirectX, and the popular open source Simple DirectMedia Layer (SDL) library. There is much to recommend high-level software development tools and methodologies, from code sharing and reuse, to component-driven architectures that decrease dependency on individual programmers. But while these features of high-level design may be favorable to the goals of both academic computer science and commercial software development, some hobbyist game programmers find their motivations elsewhere. Efficient professional software management practices value the ability to capture past solutions and make them accessible to others within the company – homebrew GBA developers, by contrast, regularly "reinvent the wheel" by duplicating behaviors normally thought of as basic building blocks of programming.

Technical mastery emphasizes a process of tinkering and discovery over the "blind" reuse of abstraction. The label of "low-level" programming, often used to describe this kind of machine-focused activity, is particularly apt in its suggestion of diving deep down into the machine. Indeed GBA homebrew development involves exploration and mapping, as demonstrated by exhaustive documentation projects like GBATEK that uncover each and every capability of the GBA's hardware through lengthy trial and error. While a high-level approach uses textual representations to indicate or invoke behaviors, the low-level equivalent will trigger those same behaviors by directly tweaking numerical values within the computer's memory. For example, the GBA provides hardware support for what are known as sprites, rectangular blocks of pixels (usually representing an object or character within a game) that can be placed on screen at different positions, sizes, orientations, and depths in relation to one another. Sprites are essential to almost any 2D video game built for a dedicated game console. In a simple case, a programmer might wish to make all sprites visible or invisible at some point during gameplay (for instance, sprites might be turned off when the player pauses the game).

One simple way to do this would be via a high-level function call such as C++ would provide, by including the line "setSpriteVisibility(FALSE);" within our source code to make sprites invisible. But GBATEK specifications tell us that in order to accomplish the same task on the GBA, we must manipulate the "display control register" (known as DISPCNT for short) by changing its numerical value. A register is a kind of raw interface to the computer's processor, where pieces of data are stored and retrieved. First, we find that DISPCNT is located at address 04000000h inside the GBA's memory. DISPCNT is a register composed of 16 bits (each of which can be set to 1 or 0, true or false); together the 16 individual bits make up one large number stored in the register. To make all sprites on the GBA screen invisible, we must set bit 12 of the DISPCNT register to 0 – but we must be careful to avoid disturbing the other bits in the register, or we will inadvertently affect unrelated aspects of the screen's status, since altering one bit changes the entire number. So we must first read the register's current status, "mask" out the bit we want to change, and, after figuring out the new value for the entire register, write the new number to its location in memory. While the high-level equivalent at the start of this paragraph may still appear esoteric to the technically uninitiated, the general difference in "level" and effort between the two methods is likely clear.

The low-level nature of GBA programming means the community values the programming process equally to – if not more so than – the final product. Register manipulations and similar activities are error-prone, and in order to succeed the group must be willing to share not just information but also encouragement and appreciation. Consequently, small technical victories that might be unimpressive to non-homebrewers are praised by programmers who themselves had to make the same, painful initial steps toward competence. The scene must be tightly knit as a group of insiders. Yet it is also a hobby with transient membership, open to serious outsiders seeking entrance:

grumpycat: "What makes it cool is that we're all enthusiastic. Beginners welcome - we all start somewhere… and when you get

your first single color sprite on the display, moving up and down with the buttons, we'll all cheer you on" [5].

This communal urge of self-reinforcement, of "just wanting to give back to the community," is also a linchpin of many open source software development projects [8].

When GBA homebrewers speak of being "close to the hardware," or having "control" over the machine, they refer to this low-level, numerical programming. It is important to note that low-level operations are not necessarily more mathematically complex than high-level ones. Low-level manipulations of the GBA's CPU registers and memory can involve a great deal of math, but usually only arithmetic, and rarely as difficult as a high school trigonometry class. Nor does it involve hands-on physical alteration of electronics (though some crossover homebrewers with a special interest in electronics do modify the hardware itself). Instead, low-level GBA homebrew stubbornly refuses to go the way of "pure" software, occupying a mid position between program code and machine hardware, and reminding us of the roots of computer programming in electrical engineering. *Lord Graga*: "Working without any kind of API is raw geek satisfaction"[Email interview].

The technical focus of GBA homebrew programming, combined with this software's non-commercial status, creates aesthetics different from those of the professional games industry. GBA homebrew software references the history of games, both technically and culturally, but it has evolved into something quite its own. In the remainder of this paper I identify a set of trends in homebrew games that demonstrate this close relationship. In amateur games, technical mastery of the GBA is by turns an end in itself; a tool for mitigating the challenge of limited resources and single-person game development; and a means of revisiting older, nostalgic gaming styles. The purpose in constructing the categories below is to evaluate homebrew games beyond their external, "received effects," to find the internal logic that governs and fundamentally motivates their development. The technical specifics of the machine are viewed through their social application.

3. TECH DEMOS: PROCESS AND PRODUCT

The largest single category of ROMs released into the GBA homebrew community, most likely encompassing the majority of programs, is the "tech demo" group. Tech demos come in several flavors, but as the name implies, their primary purpose is to demonstrate technological skill. On the GBA this tends to equate with visual accomplishments of graphics programming, but tech demos may also pertain to other functional areas. They may be visually beautiful – or of the most mundane, barebones presentation. Not a coherent genre in and of themselves, tech demos are defined negatively, in relation to what they are not. Certainly not purposeless, they are neither playable games nor functional software intended to enable some other creative activity. Tech demos may be thought of as short form works, to be quickly experienced. Often they inspire contemplation: like a puzzle, a tech demo can stimulate the user to mentally re-construct its algorithm, to try to understand the programming behind the effect of the demo.

Many tech demos are interactive in the sense that they accept and react to input from the user. Sometimes the behaviors of tech demos may be similar to those typical of games. One common feature, for instance, is the ability to use the GBA's control pad to move an anthropomorphic character around a graphical environment. Often these demos contain only some kind of visually coherent static world or "landscape," and a single, user-controlled character. There may be minimal interaction possible with other objects or characters on-screen, but usually the user is left to simply wander aimlessly with no feedback, as in *Jetpack* and *Express* (pictured below). But even tech demos that exhibit "game-like" interactivity are probably not best understood as games. While they may contain an artificial world, and a systematic set of rules, the vast majority of tech demos lack other features crucial to games, such as win and loss conditions or a quantifiable outcome [13].

Figure 1. In *Express* (above), a female character walks through, but cannot interact with, an industrial landscape. *Jetpack* (below) also features a single user in a flat, vacant world.

Lord Graga's platform demo is a borderline case: the user moves a short human figure clad in red through a small world (slightly bigger than the GBA screen) built of green tiles; if the user manages to maneuver the red figure to a door in the upper right area of the world map, the demo resets and the character is placed back at its starting position, in the lower left of the map. There is a rudimentary win condition, and thus one might technically

consider this a game, if a very boring one. But it does not provide "meaningful play" in the sense described by Salen and Zimmerman. For play to be meaningful it must be both discernible (the user can see an immediate reaction to their manipulations) and integrated, meaning an action has not only present consequences, but "also affects the play experience at a later point in the game" [13]. The platform demo contains a discernible reaction – the player reaches the door and the level resets – but that result is not integrated. The difficulty is minimal, and the same play choices are simply repeated to the user over and over in a loop, without any change in the state of the world. In other words, the demo provides play, but it is functional rather than meaningful. The play is meant to demonstrate a technical achievement such as a graphics engine and control scheme, that might enable meaningful play were its rules and content further developed.

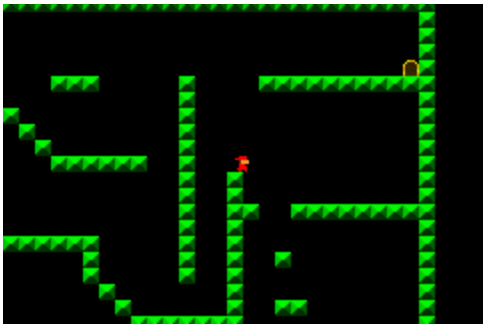

Figure 2. Lord Graga's platform demo.

Tech demos are more often even less oriented towards gameplay than the examples cited above. For instance, the two tech demos below are typical in their focus on a single graphical effect. One is a "waterfall" created using the pseudo-3D technique known as "mode 7," in which each scanline of the screen is scaled progressively larger to make a surface appear to be receding into the distance.[2] In the context of GBA development this is considered worth demonstrating because the effect is not directly provided by the machine's hardware. Instead, the programmer must write code to calculate the appropriate scaling factor for each line and then configure the GBA to apply those levels of zoom as the screen is drawn in real-time. The other demo is an image of a spinning, reflective 3D cube that is similarly difficult to program on the GBA, a device intended solely to produce 2D images. Such challenges make these demos attention-worthy to other members of the scene. The waterfall demo allows the user to adjust aspects of its size, speed, and viewing angle, while the cube is non-interactive.

A rarer audio-oriented achievement, *SPLAM!SID*, plays Commodore 64-based SID (Sound Interface Device) music files on the GBA. Commodore's SID chip was a popular tool for generating computer-based music in the 1980s, but its highly specific mix of analog and digital circuitry makes replicating its unique sound difficult on the GBA, which has sound hardware based on entirely different FM (frequency modulation) voice synthesis techniques. While *SPLAM!SID* plays a pre-selected set of well known SID songs, its real purpose is to demonstrate the coder's relative success (or failure to the ears of the user) in simulating the songs' original aural qualities, rather than to function as a limited jukebox.[3] Homebrew tech demos are closely tied to the priority of technical mastery within the scene. Many include source code, and are thus a valuable repository of applied knowledge. Releasing tech demos publicly is a way for programmers to both gain appreciation from and provide helpful tutorials to others in the community.

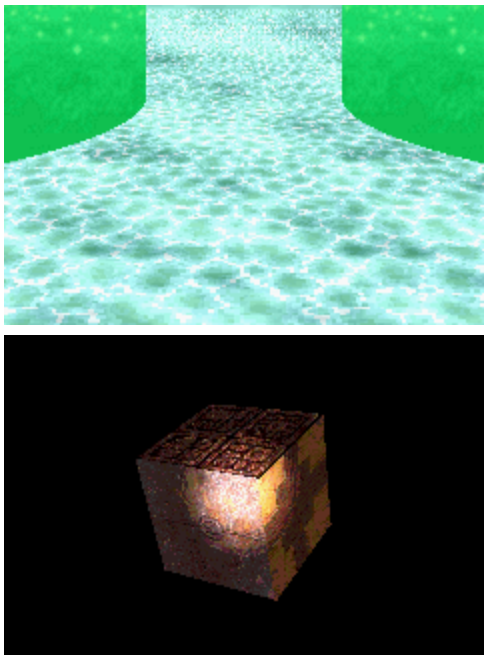

Figure 3. Tech demos usually focus on a single graphical effect, often one that is difficult to program, both showing off a coder's skills and serving as a guide for others.

Many tech demos, especially those such as *Express* that provide some form of interactive play are intermediate, incremental releases of longer works in progress. Developers exhibit these early versions to solicit creative and technical feedback from scene members. But as many in the scene have remarked, a great deal of tech demos are never turned into "finished" games:

[2] "Mode 7" is a phrase that originated in the gaming press in the run-up to the release of Nintendo's Super NES in 1991. The name does not actually refer to any official graphics mode on the GBA, but has come to stand for a particular technical algorithm used to depict a 2D bitmapped surface projected into a 3D space.

[3] Indeed, if a user (or programmer) wishes to play SID songs on modern hardware, there are options which are considerably more accessible than a GBA-specific application like *SPLAM!SID*. One popular example is the PC-based Sidplay 2.

blinky465: "...so many games appear to dissolve into the ethers…"

ScottLininger: "Projects going by the wayside is the standard for these sorts of efforts. It's no different than forming a garage band or a comedy troupe or a demo team. People grow up, get busy with other stuff, and move on to bigger, more exciting projects that never get done. I know, I've been there many times. ;)" [7].

When a programmer chooses to announce his tech demo to the scene, he often ignites longer technical discussions about the specific algorithms involved in the work. In an environment of highly individualized production, this can help to sustain motivation; such a forum can also serve as a recruitment tool to assemble a small development team beyond themselves. It also highlights how personal game development projects motivate situated learning of programming as a skill (where educational or professional contexts might not strongly appeal).

One such technical discussion involved *blinky465*'s puzzle game with an isometric viewpoint.[4] As a visual mode popular in 2D games of the 1980s and 90s, the basic algorithm for isometric projection is well understood. Discussion instead revolved around the details of implementing the graphics within the restrictions of the GBA hardware. While isometric "tiles" (e.g. depicting floors or walls) are diamond shaped, the GBA has a built-in system of square tiles. A range of creative solutions to this challenge were suggested, from finding all unique rectangular intersections between isometric tiles and re-rendering them into GBA-compatible tiles on the fly, to exploiting the availability of multiple layers of square tiles on the GBA to make the isometric tiles interlock smoothly by splitting them into alternating background layers of the GBA, like a checkerboard. Yet technical elegance was never far from sight:

DekuTree64: "But the 2-layer method is so inefficient, because not only does it take 2 layers, but you also have half your VRAM wasted on the transparent pixels between tiles. I like Keldon's idea better. It would pretty much require dynamic loading of chars [i.e. tiles]… but I think you could get it pretty darn fast by writing specialized assembly code for each of the possible angles to be copied" [6].

Despite the heterogeneity of tech demos, they are unified in their focus on the programming process over its product.

[4] In isometric projection, the perpendicular co-planar X-Y axes are projected diagonally, at 45-degree angles from the screen display axes, without perspective convergence.

Figure 4. *blinky465*'s on-going demo of his isometric action-puzzle game.

4. SMALL WORLDS: NEGOTIATING LIMITED TECHNICAL AND HUMAN RESOURCES

As noted above, a great deal of tech demos never become completed games. The most common cause of unfinished projects is an overly ambitious project scope – especially dangerous considering both the technical limitations of the GBA and the "part-time" commitment of many hobbyists with jobs and families.

LordGraga: "Some people are brave (or foolish) enough to start with an RPG in the beginning of their development career, which always ends up as a small demo, usually with some ripped GFX [graphics] and no finished feel" [Email interview].

Earlier I made a formal distinction between tech demos and games, based on the kinds of play they allow (open-ended and functional vs. goal-oriented and meaningful). But if there is one phrase which captures the additional, intuitive quality of a "game," it is the vernacular "finished feel." A game feels finished when it sustains interest over time, when it provides a variety of approaches and solutions to its challenges, when it surprises the player. By looking at those "finished" games released to the scene – games that offer meaningful play – we can see emergent trends.

I use the term "small worlds" to label a set of homebrew GBA games that, while spanning multiple genres, graphical styles, and technical implementations, are driven by an underlying design aim. Their "smallness" indicates no slight or negativity, but instead refers to both the spatial layout of their game worlds, and the algorithms of their technical approach. Small world games harness the built-in features of the GBA hardware that minimize the programming of game infrastructure; they resist the temptation towards indulgent programming to encourage a focus on the implementation of gameplay. Their design is also often cognizant of the on-going nature of homebrew development, in which additions are made in pockets of time snatched during late nights or vacations. In short, they are an effective response to homebrewers' technical and social restrictions.

Llamabooost is a top-down survival shooter in the tradition of the arcade classic *Robotron: 2084* by Eugene Jarvis, and a more

immediately thematic ancestor, Jeff Minter's *Llamatron: 2112*. As in those predecessors, *Llamabooost* restricts gameplay to a single screen. The player controls a character occupying a relatively small portion of the screen (roughly 1/16th of its width), able to fire projectiles in one of four directions. Gameplay is simple – navigate the screen, shoot enemies, and collect bonuses. But *Llamabooost* and its precedents continuously renew gameplay by injecting an ever-increasing multitude of enemy characters into the level. The tension of gameplay is built into its limited space, a frantic sense of enclosure as the player avoids surrounding adversaries.

Lord Graga: "I usually code smaller games which I can keep building on until I'm satisfied…. With games like Llamabooost I can add lots of stuff later if I wish… and I still have a finished product if I stop somewhere in the middle of the level/monster/powerup adding" [Email interview].

Lord Graga supports the argument that games allowing for *incremental* alteration are more suited to homebrew development. *Llamabooost* is designed from a sedimentary approach in which a bedrock foundation of simple rules and code provides the basis for further evolutionary additions. In early levels, the enemies are all of one type, but as the game progresses, new kinds of creatures appear, with additional capabilities. The walking flames, for instance, hurl fireballs at the player.

Beyond a cultural homage to 80s shooters, *Llamabooost* is technically well suited to the GBA, because its gameplay is entirely based around the movement of sprites. All mobile entities in the game (the player llama, the bonus llamas, and the enemies) as well as static items (e.g. power-ups that recover player health) are easily represented as 2D sprites. Because the GBA supports sprites as a basic feature, the coder does not need to focus on more sophisticated graphics programming techniques, like the planar scaling of the waterfall demo described above. To display a GBA sprite you simply the machine with an image and set its X and Y coordinates. And in *Llamabooost* each game level takes place on a single screen, without scrolling the background image, eliminating the need to position the sprites relative to a moving "camera" as would be necessary in many top-down 2D games. The gameplay is enabled by a large number of simultaneous sprites, advancing to create the feeling of being trapped. The GBA provides for up to 128 sprites at any one time, an ample provision for creating this sense of visual overload. *Llamabooost* simply but efficiently makes the most of the GBA hardware.

Figure 5. The first level of *Llamabooost*, in with one type of unarmed enemy, top, and a later level containing fire-throwing adversaries, bottom.

The puzzle game *ClacQ* demonstrates the small world aesthetic differently. Puzzle games, the 80s title *Tetris* being the most well-known, must be included in any broad consideration of amateur game development. A genre popular (possibly most popular) with non-professional developers of many stripes, from GBA homebrew, to indie PC, to web games sold as shareware, it lends itself to hobbyist development because gameplay is similarly often constrained to one screen. A rectangular grid of graphical tiles is commonly used as a playfield. In contrast to a game like *Llamabooost*, however, which moves in fast real-time action, puzzle titles like *ClacQ* depend on bursts of concentrated investment by the player to solve each self-contained level or puzzle. Will Wright often describes the act of game-playing as one of mentally reconstructing an internal model of the game's externally displayed mechanics. By observing and interacting with the game's behavior on screen, the player increases his or her skill at the game by iteratively improving that psychological model [15]. Puzzle games maximize this offloading of computational and entertainment burden from the game mechanics to the player by inducing relatively long periods of non-interactive contemplation. Players proceed by a process of trial and error, reconfiguring the same small set of level elements over and over.

In *ClacQ*, players must build a safe path for the traversal of an egg from one end of the screen to another. When gameplay begins, the player is presented with a field of disconnected gray blocks. By placing the appropriate "scaffolding" segments, a smooth route is paved. The levels quickly build in complexity by requiring such platforms to be chained together. *ClacQ* supplements this standard contemplative mode of play with additional, layered

challenges to the player, some explicit, others revealed progressively. The most basic challenge is a time limit for each game level. While time restrictions are a common game feature, the short fuse of those in *ClacQ* creates an unusually hectic atmosphere, and again emphasizes reuse of design and content by increasing the likelihood of multiple attempts at puzzle completion. Incremental challenges prolong play as the game advances. When the player fails a level, he may restart the; but after the completion of level 10, the player is told to go back and solve the previous levels again, consecutively and without faltering. Against the expectation of having succeeded, the player is further challenged – without requiring any new content. While such a strategy risks player frustration, in the tradition of perfectionist and "precision" play it is an acceptable trope, more likely to induce a sense of renewal.

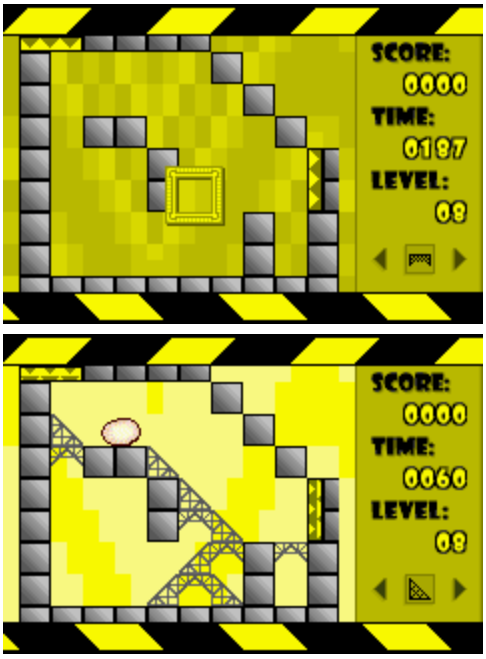

Figure 6. *ClacQ* begins with a set of unconnected gray blocks (above); by bridging the blocks with scaffolding, the player ensures safe passage for an egg that rolls through the level (below). The interdependency of scaffolding quickly reveals gameplay dependent on careful planning and mental projection of spatial layouts.

ClacQ's scaffolding theme takes maximal advantage of the GBA's tile maps, a feature common to dedicated game consoles designed for 2D graphics, but unusual on a more general purpose PC. In a tile map, unique square graphic units (such as the gray blocks and scaffolding segments below) are assigned numerical indexes, for reference. The programmer sets the values of each map cell using a tile index, and the GBA then automatically draws evenly spaced tiles. *ClacQ* uses tile maps extensively: besides the obvious use of a map to represent the level's playing field, the animated, bright yellow background is also a coarser map scaled up to fill the screen.

In *Megatroid*, another homebrew title dependent on tile maps, the play field exists simultaneously on two, overlaid tile maps, but only one map is displayed to the player at any given time. Combining action and puzzle play, the goal of each level is to find an exit, within a harsh time limit (ensuring a breakneck flurry of jumps through game space). The twist to *Megatroid* is that the player must switch between "dimensions," or tile maps, to slip past a constant series of dead ends on one map by finding a through-route on the companion map. This is a particularly subtle use of tile maps to technical advantage: while *Megatroid*'s space of play scrolls through an area larger than a single GBA screen (unlike *Llamabooost* and *ClacQ*), by using two maps (the GBA provides up to four) at the same time, the designer avoids having to chain multiple maps together. Where a role-playing game would typically string several tile maps side by side to create a feeling of a large, connected space, the levels of *Megatroid* are the size of a single map. Maintaining two separate maps internally is easier, but the technique still allows twice the total space to be modeled – *and* it does so by integrating the separation of the maps directly into the game's mechanics. With this simple tactic, *Megatroid* captures the "exploratory" gameplay in expansive RPGs. By limiting visibility in a condensed space by, it depends on the player to mentally provide the gaps in its visual presentation.

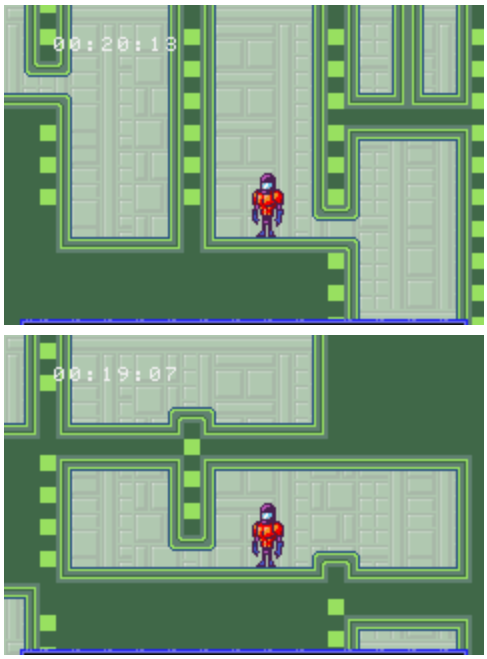

Figure 7. In *Megatroid,* the player alternates between two versions of the same space. Pressing a button toggles between the two spaces, without additional player movement (top and bottom).

These "small world" titles use methods of gameplay (some tried and true, some novel) along with specific techniques appropriate to the GBA hardware to reduce development time and prolong play. While there are occasionally completed "large" games released into the scene, the small world approach has yielded a

substantially higher percentage of success for those seeking the "finished feel." Many homebrew projects that adopt the role-playing genre, for instance, may demonstrate admirable technical and spatial engines, but lack the length of play, and replayability of their "smaller" siblings. They feel uninhabited, stretched thin across screen after screen of look-alike spaces.

5. GENRE REMEDIATION: INTERSECTING CULTURAL AND TECHNICAL STRATEGIES

Some homebrew projects more explicitly and reflexively mimic past commercial genres as a means of cultural commentary. As *Llamabooost* hints at, "retro" games and play mechanics often serve as inspiration for homebrew works (as they certainly do for commercial games). The PD ROMs website's Coding Compo 2.5 from 2004 is a primary example of such reference at the community level [12]. Competition entrants were asked to create original games in the aesthetic and technological style known as "Game & Watch," after the brand name of early 1980s handhelds marketed by Nintendo (and copied by competitors such as Tiger Electronics). Game & Watch machines were technically rudimentary devices that each ran only a single game (as opposed to running multiple games via cartridges). Using an earlier variant of liquid crystal display (LCD) technology, the screen itself was pre-fabricated to contain all of the graphics for the specific game it was manufactured to support. For instance, if a character was to be portrayed walking across the screen, several black and white LCD segments representing the figure were placed at incremental locations. Lighting on and off these segments achieved the illusion that the character was "moving" [3]. The stylized non-realism of Game & Watch's visuality shares more with 19th century pre-filmic inventions like the zoetrope than with the fluidity usually associated with digital animation.

The Game & Watch brand plays a critical cultural role in game history as one of the initiators of handheld gaming. A homebrew return to the style is not only a kind of homage to childhood memories, but also formally "appropriate" to the small form factor of the GBA. This connection is plain in the commercial sphere as well, as demonstrated by Nintendo's recently released (November 2004) DS (Dual Screen) handheld system (a successor to the GBA). Its design is not "new" or futuristic but instead patterned off of multi-screen Game & Watch units. In this regard both homebrew and commercial game products are "remediating" the Game & Watch genre. In coining the term remediation, Bolter and Grusin largely focused on cross-media repetitions of formal conventions, such as adoptions of painterly lighting and soft focus by early photographers, or backflows between media like overlay additions to cable news in reaction to the multi-windowed and constantly streamed nature of websites.

The PD ROMs Game & Watch compo is an intra-medial or local form of remediation. Replicating the older form on a more powerful and current piece of hardware also causes what Bolter and Grusin call hypermediacy, an intensified awareness of the artificiality of media. The compo entries re-create the entire look and feel of Game & Watch: grays and washed-out colors mimic the dull, monochromatic liquid-crystal displays, technological artifacts like the 'ghosting' of unlit graphical elements, and the illusion of non-electronic full-color printed backgrounds (a kind of miniature, bastardized transposition of the grand panoramas of the 18th and 19th centuries). It's worth noting that while the competition allowed entries designed for several console platforms, the GBA-based games as a group offer considerably more attention to these "realistic" details. By comparison, the entries for other console systems possess stark, high resolution black and white imagery that appears simply minimalist, without any of the recognizable technological genre markers discussed above. The GBA developers apparently feel a greater sensitivity to and appreciation of the Game & Watch aesthetic – perhaps an effect of Nintendo's cultural resonance as a brand, as well as the original parent of this LCD gaming style.

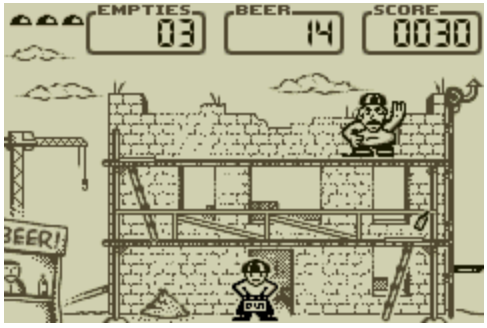

Figure 8. *Beer Belly Bill*, the top ranking entry from PD ROMs' Game & Watch-themed competition.

Game & Watch is a technical form of intense limitations, and its selection as a compo theme implicitly recognizes the role of the hardware platform in shaping the conditions of production and the aesthetics of games. In this sense it is metonymic shorthand for the "taking on" of technical restrictions purposefully tackled by homebrew GBA developers. In his theory of games, Bernard Suits describes the act of game-playing as the "selection of inefficient means." In Suits' view, a game is inherently inefficient because it involves intentional adherence to artificial constraints (the rules of the game in question) to achieve its designated goal. Suits uses the rather comical example of golf, in which the goal is ostensibly to get "a ball into a hole in the ground," yet this is accomplished not by simply placing the ball directly, but by using a golf club, standing at the proper distance (far off) from the hole, and following the protocol of the swing [14]. Extending this comparison to game-*making*, GBA programmers may be perfectly efficient at harnessing the machine's hardware, but their very decision to work with the platform in is the adoption of technical, and in consequence aesthetic, restrictions. The further circumscription of the Game & Watch's own constraints on the GBA externalizes this choice.

Why would GBA homebrewers want to re-create the "primitive" style of Game & Watch games? Certainly the form has nostalgic appeal, but it is also a strategy that shifts the frame of evaluative reference, allowing amateurs to avoid competition with the commercial, or rather, to compete on their own terms. By "stepping down" to this nostalgic genre, homebrew developers substantially increase the feasibility of their productions – it is a lot easier to code the non-animated graphics of the simple, iconic poses of a Game & Watch character. It also places homebrewers in a position of critique. It is difficult to discern the relative intention of Game & Watch look-alikes (and similar genre remediations) as homage or parody; many games offer a mix of both.

6. REMAKES: USING EXISTING GAMES AS A MOTIVATIONAL FRAMEWORK

Another popular focus of homebrew development is on game remakes. Here the goal is to replicate, to varying degrees of fidelity and accuracy, the graphics, sound, and gameplay of existing commercial games. Remakes should be distinguished from console emulators, programs that allow less sophisticated game hardware systems, such as the NES, to be simulated on the GBA. Once an underlying system has been emulated, *any* game originally designed for that machine can be played on the GBA. A remake works by a very different logic. Instead of executing the native binary code of the game on the GBA (as opposed to its initially intended target platform), the purpose is to independently reconstruct the *behavior* of the game. Writing an emulator is actually primarily software, rather than game, development – the playing of games is an indirect effect enabled by the process. Writing a remake is game development at the primary level, technically and inspirationally. Creating a remake is an intensely technical activity, an ideal target for perfectionism.

Contrasting emulators with remakes is useful because while both could be forms of preservation, they reflect qualitatively separate impulses. The function of preservation should not be understated. Indeed, as Eric Zimmerman notes, "playing older games [is] a hobbyist's trade…. [Games are] a medium without a history" [16]. There are several movements beyond the GBA community aimed at the archiving of games and supporting materials (e.g. digital scans of printed manuals and retail box art). Along with emulators, these databases are reflective of a "collector's" approach, a vernacular museum (as other prescient attempts to archive popular media have been) [4, 9, 10]. Remakes are ultimately less about preservation and more about nostalgia and homage to individual, personally cherished games. They are analogous to amateur practices including musical cover bands, the classic kit car movement, and hobbyist model airplanes.

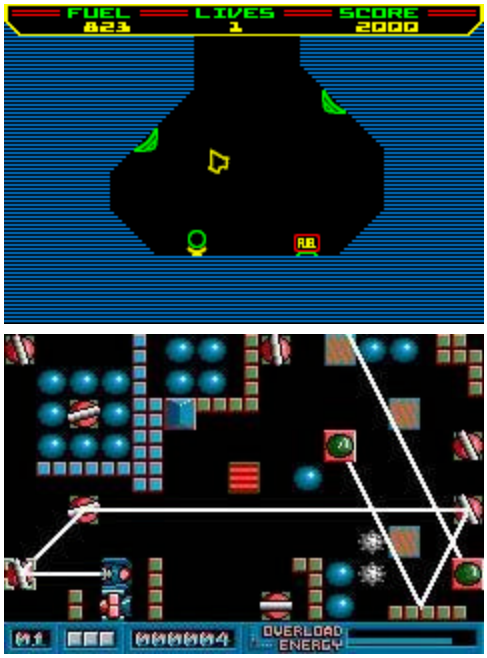

Figure 9. High accuracy remakes of classic Atari ST computer games: *Thrust*, top, and *Deflektor*, bottom.

One common style of remakes emphasizes the fidelity of the re-creation – developers will go to great pains to extract the original, pixel-accurate sprites and other graphics from the source game. Certain secondary details, like the unique behavior of a high score list, might be singled out for criticism by both the game's creator and audience. These remakes become an unusual reverse engineering challenge. Though replication of program functionality has a long history in software development, the highly interactive (often real-time) nature of games, coupled with the requirement of visual consistency sets the intent of the activity apart. At heart, the game has to look right and more importantly feel right, an intuitive tactile achievement that can't be faked. The master games these "faithful" remakes seek to model are by technical necessity for older machines, particularly early home computer systems like the Commodore 64, Atari ST, and Sinclair ZX Spectrum. Their graphics, like the Game & Watch, are more than a step down from the GBA's native capacity, and their screen resolution similarly close. But they also provide an ideal source material for homebrewers: both feasible to copy and, central to the gaming childhood of many GBA coders. Notable examples from several genres are pictured below.

These exacting remakes raise fundamental questions about originality and autonomy in acts of media replication. They do not seem impressive when judged on their contribution of "new" features, the traditional concept of innovation. However, we must recognize that the game industry as a whole has always been openly based on free borrowing and repetition. The first "amateur" computer-based game *Spacewar* became the first commercial arcade game, *Computer Space*. The scene itself grapples with these questions of value:

Matto154: "What is up with people wanting to make ports or remakes? It seems as though everyone who wants to make a game is just copying other games. What is wrong with people? It is not that hard to think of something original" [2].

One answer is that remakes can be fan games, too. Like playing in a cover band, or using the family camcorder to re-enact a scene from a favorite movie, creating a remake is an act of self-identification with a slice of our media sphere:

Gopher: "I never understand people who say they can't come up with any ideas…. However, there is another reason people like to port games. When you find a game you really love, you want to be a part of it, and porting it to another platform gives you a sense of that" [2].

But a fuller explanation requires a return to the technological privileging that is evident from tech demos onwards:

SmileyDude: "It really depends on what you are trying to accomplish -- if programming is what turns you on… porting an existing game is a great idea" [2].

A remake can focus technical interest and enthusiasm for a project, offering a highly structured framework and goal. *Mike Hawkins* describes the process of re-creating *Android 2*:

"To ach[ie]ve this I had to reverse engineer the orig[i]nal Speccy [e.g. Spectrum ZX] version, mainly to rip the map data and initial bot positions. The maze map turned out to be just 60 blocks wide and about 240 blocks high. The maze is wrap-around so appears much bigger than it actually is. The initial bot positions were a bit more difficult to find since they are not stored as a single table nor sequentially in RAM" [Unpublished game manual].

The production of technically accurate game remakes is a programming puzzle grounded in very specific cultural references.

7. CONCLUSION

The Game Boy Advance is not the first or the only console system to have widespread homebrew support: the original Atari 2600, launched in 1977, seems to have maintained the strongest amateur development community over time, and other popular systems of the past, such as the Nintendo Entertainment System (1985 U.S. release) and more recently the Sega Dreamcast (1999 U.S. release), have also garnered considerable attention. But as one of the most vibrant homebrew scenes, the GBA offers a representative glimpse into non-commercial game-making. And the particular technical status of the GBA makes it both accessible to a single programmer (the machine runs on an ARM7 processor common in many embedded devices), and suitable to mimicking older gaming styles of the 1980s and 90s (through its low resolution, 2D sprites, and tile maps).

Technical performance is an equally important criterion for the evaluation of commercial games, to be sure – the entire games industry has increasingly fetishized its technology and 3D graphics capabilities in particular. But in an amateur environment in which the programming process is considerably more transparent, where the audience is often most interested not just in interacting with but in *understanding* the technology underneath the gloss, technical mastery becomes more than a professional job credential, but instead a key personal and social motivator for everyday programmers. The GBA homebrew scene shows us how a focus on technology can actively shape, rather than limit, aesthetic design through shared cultural markers and expectations.

8. REFERENCES

[1] Everybody's Profession??? *gbadev.org*. Retrieved Jan. 5, 2005. http://forum.gbadev.org/viewtopic.php?t=3070.

[2] Flashback. *gbadev.org*. Retrieved Mar. 24, 2005. http://forum.gbadev.org/viewtopic.php?t=2203.

[3] Gielens , J. *Electronic Plastic*. Gestalten Verlag, Berlin, 2000.

[4] Home of the Underdogs. Retrieved Aug. 8, 2005. http://www.the-underdogs.org/.

[5] Homebrew games on the DS. *Warp Pipe Technologies*. Retrieved Feb. 23, 2005. http://forums.warppipe.com/viewtopic.php?t=13874&postdays=0&postorder=asc&start=15.

[6] isometric games. *gbadev.org*. Retrieved Mar. 7, 2005. http://forum.gbadev.org/viewtopic.php?t=4421.

[7] Just wondering… how many games ever got made? *gbadev.org*. Retrieved Jan. 7, 2005. http://forum.gbadev.org/viewtopic.php?t=4718.

[8] Kelty, Christopher. Hau to do things with words. Retrieved Nov. 16, 2003. http://www.kelty.org/or/index.html.

[9] Killer List of Videogames. Retrieved Aug. 8, 2005. http://klov.com/.

[10] MobyGames. Retrieved Aug. 8, 2005. http://www.mobygames.com/.

[11] Nintendo GBA Beats Hollywood Blockbusters. *NintendoHead.com*. Retrieved Apr. 5, 2005. http://www.nintendohead.com/news/29.

[12] PDRoms Coding Competition 2.5. *PD ROMs*. Retrieved Mar. 15, 2005. http://www.pdroms.de/pdrc2_5-submissions.php.

[13] Salen, K. and Zimmerman, E. *Rules of Play: Game Design Fundamentals*. MIT Press, Cambridge, MA, 2003.

[14] Suits, B. *The Grasshopper: Games, Life and Utopia*. University of Toronto Press, Toronto, 1978.

[15] Wright, W. Sims, BattleBots, Cellular Automata, God and *Go*: A conversation with Will Wright. *Game Studies*, 2.1 (2002).

[16] Zimmerman, E. Do Independent Games Exist? In *Game On: The History and Culture of Videogames*. Ed. Lucien King. Universe Publishing, London, 2003, 120-9.