postmortem

John Asmuth and Timothy Gerstner and Brian Poppy

Engine, AI, Sound code

Lead, Art code

Graphics, Levels code

4/29/2010

Overview

Cursed is a 2D fantasy RPG in which a druid and his familiar do their best to survive in a hostile world populated by demons, antagonist druids and the perils of his own mysterious past. The hero himself is unable to defend himself beyond casting a simple wood spell to block the path of his enemies, and must rely on the powerful, yet finicky, familiar for protection.

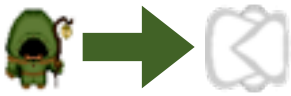
The player must fight his way through a dead world, drained of its energy during some cataclysmic event immediately prior to the game. Over the course of his travels, he is able to piece together events from the past and eventually discover the cause of this disaster, as well as its remedy.

Technology

Cursed was developed with Visual Studio 2008 using XNA as its general framework and Box2D as its physics engine. Source control was managed through the parallel use of Dropbox and Git. The artwork was drawn first on a Wacom Intuos 3 tablet and then edited with Gimp. Audio samples were played first on an Alhambra classical guitar using an Edirol USB Audio interface and Audacity, and edited with Adobe Audition. Color schemes were derived using http://colorschemedesigner.com as a starting point. Pidgin/AIM, Google Wave and IRC were used to facilitate communication. Google Code was used as an issue tracker.

Technology used for content that did not make it into the final game includes Maya 7 for 3D models, and Noteabilty Pro for music composition and generation.

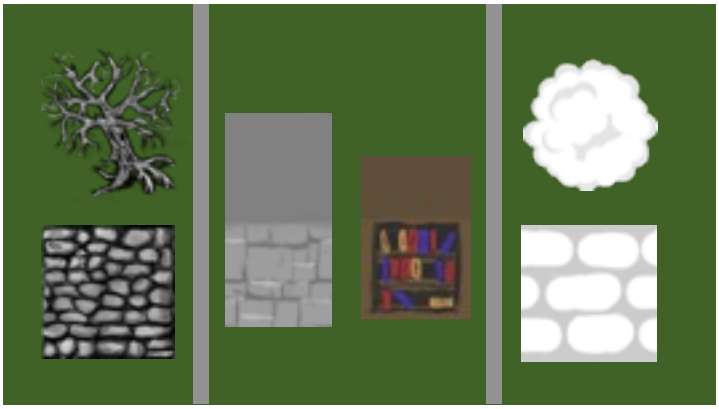
Things that went right



1 The final art style

was the result of a lengthy progression, beginning with (from left to right) the prototypical "circles of various colors" art (1), to a isometric-view and hand-drawn feral sprite (2), to a demonic and dangerous looking beast drawn entirely on computer (3), followed by a simpler (and quicker to draw) sketch on top of a pre-rendered model (4), and eventually finalized with a bird's eye (and easy to rotate) marshmallow ghost(5).

The tile art went through a similar progression, beginning with "squares of various colors" (not pictured), and transitioning to the very detailed and high frequency hand-drawn line art (left), followed by a slightly not-as-high frequency isometric tile set (center), to the very low frequency and two-toned bird's eye view tiles (right).



The final art style is very consistent and low frequency, allowing the player to relax and be comfortable as the druid uncovered more of the world.

As important to the final look of the game as the art is the color post-processing step. When all the artwork was drawn in full color (using various color masks on top of the original grayscale images), the color saturation was overwhelming. Mitigating this is the fact that a much smaller portion of the screen was drawn in color, indicating areas to which the druid's familiar was more attracted. Coloring only part of the screen helped to both focus the player's attention and avoid the dreaded Willy-Wonka look.

2 Source control, using the combination of Git and Dropbox, proved an effective way to share and archive code. Using Git allowed the developers to work on different aspects of the game independently, and merge with each other's branches as desired. There was no single "master" copy - only the three copies owned by each of the developers. The entirety of the repository and working code was kept in Dropbox and shared, allowing the making the request to "run my code" common, as it could happen without interrupting any work or taking more time than required to double-click the relevant executable.

3 The method of narration

was an unplanned strong point for the game. Having glimpses of the story appear as text on the ground, written in the third person, gave the game a storybook feel and provided motivation to explore and find new memories. This style is very similar to childrens' picture books, where the narration and artwork are often mixed.

4 Communication was not a problem.

The primary method of communication was Instant Messenger, combined with an ability to instantly (via Dropbox) examine another developer's code or demo, speeding up bug diagnosis and resolution. The ease of using direct communication to quickly fix bugs and solve other problems caused the issue tracker to fall out of use early in development.

5 Level editing could have been a disaster, except for the in-game level editor. By pressing a key, any developer could quickly drop into editor mode and paint new levels in very little time. The appearance of game objects was specified by the use of "presets", or string identifiers to linked to visuals defined elsewhere. Using presets, a level could created using one art style could be quickly converted to a new one with a minimum of effort and hassle.

Things that went wrong

1 Scoping proved to be a challenge to the inexperienced. Giddy with enthusiasm and the over-extrapolating the success-to-effort ratio observed in the prototyping phase of the course, the cursed developers (who also happened to be the "Cursed" developers) were more ambitious than abilities and the time frame allowed. Achieving the desired gameplay took longer than anticipated, leaving not enough time for level design and story.

2 The constantly changing art styles provided a challenge to level design and detracted from the output of the team artist in other equally important parts of the project, like making sure the engine programmer focused on enabling good gameplay more than maintaining pretty code structure, and generally keeping the game on track. Although the final art style was successful, it came late in the project's lifetime.

3 Roll-your-own physics engines can be fun to work on, but are extremely time consuming and not germane to the primary focus of the course: game design (as opposed to game development). The physics and collision engine used in the beginning suffered initial efficiency issues, delaying other aspects of development. Even when the engine speed was brought to manageable levels (using quad-tree body indexing), the limits of the engine programmer's physics knowledge were reached when many different bodies pushed into a corner, causing them to jitter confusingly and often pop through walls entirely.

Although the transition to Box2D happened quickly (basic functionality over the course of one day), the franken-code base never quite recovered from the shock and remains littered with a mix of two-and-one-half different geometry libraries and many subroutines that are not done in quite the right way for Box2D.

4 Choosing XNA had some benefits, but also some severe disadvantages. One of the developers had access only to a single desktop machine with Windows, causing him to be chained to his apartment for much of the semester. Beyond the obvious cabin-fever issues is the fact that when this machine kicked the bucket (almost literally - a coolant reservoir developed a leak, shorting out some components on the motherboard), the developer in question lost most of his ability to contribute in serious ways to the project.

Writing code optimized for C# and XNA also proved challenging - minimizing allocations turns out to be key (Box2D has a special XNA version beyond the C# version that minimizes allocations). Much of the early engine work was spent optimizing in this way.

XNA was chosen as the target platform in part because of the theoretical ease of introducing "cool new graphics stuff". The only significant aspect that was made potentially easier is the color post-processor. The rest of the visuals were vanilla sprite placement and alpha-blending.

The combination of a language that compiles on both Windows and Mac OS X, such as Java or C++, and OpenGL would have provided more flexibility. The option was weighed, and the ease of getting things working quickly with XNA, the lack of experience with C++ for two of the three team members, and the difficulty of developing a multi-platform Java/OpenGL application outweighed the benefits.

5 Indoor levels proved to be an outstanding failure in playtesting sessions during class. The tight corridors and nature of combat in “Cursed” made indoor, obstacle-heavy levels frustrating and distinctly not-fun. Fortunately the class and instructor stepped in to correct this error early on, but time was wasted.

Conclusion

Clearly, the development of Cursed was a learning experience. Three important lessons were learned. First, focus on what is important. As discussed earlier, game engine design and game design are two different things. Second, focus on what is possible. Simply knowing what is possible is impossible without prior experience. Third, the importance of testing cannot be overstated, even though every effort was made. These lessons, and others described earlier, are important for any sort of project, especially when software is involved.