How To Build a Game In A Week From Scratch With No Budget

by Jay Barnson, Rampant Games

An RPG in a week, starting from scratch?

How hard could it be?

I did it because of a dare.

There are many great resources available to small, budget-conscious independent game developers today. On a public forum, in a counter-rant, I expressed this fact by bragging that if you gave me a week, a fresh install of Windows, and a good Internet connection, I could build a halfway decent game with no budget whatsoever. No, it wouldn't be able to compete with Halo 2 or anything (if I could create that kind of game in a week, I'd quit my day job), but it would be reasonably amusing and playable.

Tom Bampton, who runs the monthly Game-In-A-Day 'competition' (www.gameinaday.com), said "You're on!" He then added an extra contingency – I had to do it without the benefit of one of the (free) game engines out there. I could only use a basic library / API.

At first, I dismissed the idea. I didn't have time to take a week off of work and my current game development project to do something like this. But then I thought: What is a week? Unless you work for EA, a work-week is 40 hours. How about taking 40 hours to create a game? I was intrigued – but I didn't want to just create a space-invaders clone. How about a role-playing game – one of the most complicated genres to create games for? Would it be possible?

I didn't know. I knew it would be extremely difficult. But I accepted the challenge.

On top of that, I documented what I was doing as I worked, which I expected would be a little like going through an entire development cycle on fast-forward. I thought it might be interesting to game developers – or at least an entertaining record of how I fell on my face if I failed. The end result was a long, rambling, stream-of-consciousness record of my hourly activities. I've tried to edit it down to something a little less yawn-inducing here.

So here is how I created a game in a single week from scratch, with no budget. If you want to skip to the end and see what the final product looked like, in all it's buggy, imperfectly-realized glory, you can download the Windows version of the game at: http://www.rampantgames.com/hackenslash.html

The Plan

The Goal

Create an "old-school" RPG in the style of the old, early 80's "top-down" RPGs like The Temple of Apshai, Ultima III, and Telengard. The player will move through rooms in a stereotypical dungeon, doing battle with various monsters with magic and combat. Along the way he'll improve his capabilities through gaining "levels" of experience, and magical equipment.

It won't all be about combat, however. The player will also have the ability to sneak past or negotiate with monsters. There will be locked and trapped doors and chests, and unique dungeon features that may have strange effects. The game will not be long on plot, characterization, or dialog – it's mainly a hack & slash affair. You go up the level treadmill until you are powerful enough to face the final boss, retrieve a great quest item, and bring it back safely home (your 'starting room').

The Rules

Rule #1: A limit of one work-week (defined as 40 hours)
Game Development time should be restricted to 40 total hours. These will be actual game development or research hours. Breaks of longer than ten minutes won't count towards the total time. This will be an "ideal" workweek of 40 highly productive hours.

The 40 hours only includes development to a feature-complete "alpha test" stage. Debugging and packaging the game for distribution and won't count towards the development time, but no new features should be implemented. Documentation of the process doesn't count.

Rule #2: All Free Tools
Except for the software that comes with a Windows install, only free / open-source software tools are used. The point of this whole exercise is to show how you don't need expensive (or even not-so-expensive) tools to develop a game. Hardware such as a scanner, microphone, and digital camera are exempted from this rule – if you don't have these, you can probably borrow them from someone.

Rule #3: No Engines, only basic libraries / APIs
The game must be created "from scratch" without the benefit of a fully-featured Game Engine. No cheating and creating a game using some kind of "click-and-play" game-maker software to throw together a game.

The Tools

Code:

- Python 2.3 (http://www.python.org/)
- PythonWin
- PyGame (http://www.pygame.org/)
- Py2EXE to compile this into an executable for distribution (http://starship.python.net/crew/theller/py2exe/)

Art:

- Gimp 2.0 (http://gimp-win.sourceforge.net/)
- MS Paint (which comes with Windows) – for pasting up screen shots grabbed by hitting the PrintScreen key (GiMP doesn't like these for some reason)
- Free textures available from places like Toob's Tiled Textures (http://www.textureartist.net/textures/index.htm) and Mayang's Free Textures (http://www.mayang.com/textures/)

Sound:

- Audacity (http://audacity.sourceforge.net/) plus my microphone or free sound samples

The (Intended) Schedule

Schedules are made to be broken, but it's important to have them as a baseline to compare your progress with and make corrections as necessary.

Hour 1-10: Basic Architecture
Design the "engine" and the main components. Get the world displaying on the screen. I should be able to move a 'test player' around the world to look at things. In fact, I should allow the "test player" to be turned into a full-on editing tool if I can swing it.

Hour 11-20: Player Interaction
Implement all core interactivity for the player – moving around, attacking things, opening doors, dying, picking up and using inventory. Bare-Bones representative objects in the environment will be created to test the interactivity

Hour 21-30: Making the World Active
Add the AI, game "events", traps, and special effects. By the end of this period, the game should be a pretty complete tech-demo of all of the game's major features.

Hour 31-40: Adding Content and Rules
Take the project from "tech demo" to game. Add all additional content. Complete and balance the game-play mechanics. Apply polish where time permits – adding special effects, animation, etc.

Post-Hour 40: Testing and Game Release
Fix bugs (no adding features!) Package up the game and release it. Finish documentation.

The Development Diary of Hackenslash: A Game In A Week

Hour 1 – Wild Freeform Design and Base Classes

I spend this hour creating some basic classes for the game – and using these to help guide my design. The world is represented as a series of rooms, connected by portals. Everything in the world is room-relative, similar to how old Adventures and

MUDs are designed. Most objects in the game are represented by a "GameObject," which has a position and contents (which include other objects – a map might contain rooms, a room contains a box, a box contains a sword… and I guess the sword could contain more rooms, but we won't go there.)

- I create a "creature" and "player" object
- I generate a set of "attributes" off the top of my head for creatures, and put this in a class. Apparently I'm a game geek who has played way too many RPGs. I don't know exactly how the game mechanics will work, yet. This really is seat-of-the-pants game development!
- I make a "room" object, derived from GameObject. Rooms have width, height, and walls – and not much else right now.

I figure out how things will work and make corrections as I go. I don't even have PyGame linked in at this point – I don't even have anything other than a console for output. But I feel like I've made great progress!

Hour 2 – PyGame 101

The goal this hour is to initialize PyGame, and start putting things on the screen. Actually, I spend most of my time going through the PyGame documentation and figuring out how to do things, since I have almost no experience with PyGame or SDL.

I end the hour bringing up a blank screen filled with black. So far, it's not very impressive. Actually, there's quite a bit going on behind that black window – sort of like the Black Triangle story. There's a functional game-loop, page-flipping, the calling of several classes, and a lot of stubbed behavior. But that doesn't make the black screen any more impressive.

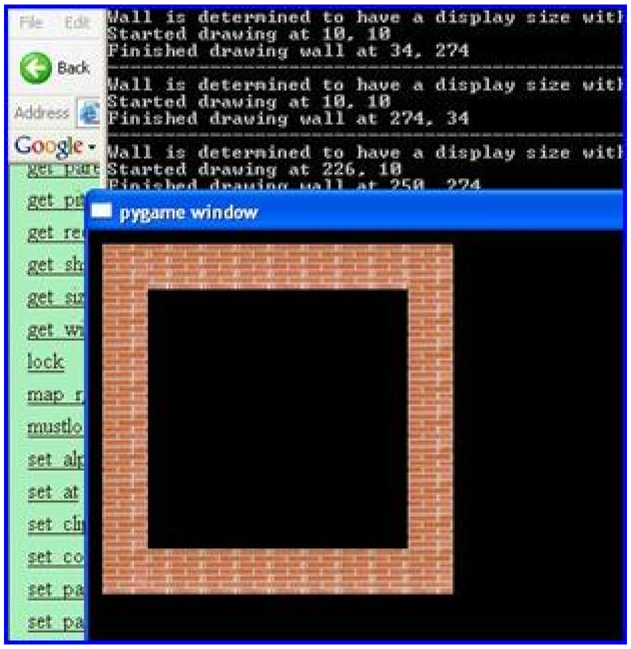
Hour 3 – If the Walls had Ears, I'd be cussing at them.

This hour's goal is to get a room's walls displaying on that black screen. To make this happen, I need a room, and I need graphics. I spend a lot of time in GiMP touching up some textures downloaded from Mayang's Free Textures, so that they tile. I create a texture manager class. And I fill out a sample room structure. I also spend a bit more time looking through PyGame's docs to see if there's anything else I can use that might make the job easier.

At the end of the hour, I still don't have walls on the screen.

Hour 4 – The Inn Now Has Room

After fighting some syntax errors, I finally get the walls to appear on screen. Not displaying *correctly* – they are in the wrong positions, with gaps between segments. It's horrible. But with a little bit of tweaking and bug-fixing, I have something resembling a 10-square by 10-square room on the screen.



```
Wall is determined to have a display size wit
Started drawing at 10, 10
Finished drawing wall at 34, 274
---------------------------------------------
Wall is determined to have a display size wit
Started drawing at 10, 10
Finished drawing wall at 274, 34
---------------------------------------------
Wall is determined to have a display size wit
Started drawing at 226, 10
Finished drawing wall at 250, 274
```

Without a really detailed project plan, it's really easy to get lost when you get to one of those points where you wonder, "What next?" It's very easy to get stuck chrome-plating what you've already done without moving on to the next task. I decide that if drawing one room is good, drawing two rooms is better, and move forward towards this goal.

- I create a "minidungeon" file to handle the generation and storage of these rooms.
- I start adding logic for "portals" – holes in the wall that lead to other rooms (and provide all the offset information necessary to draw the adjoining rooms).

Hour 5 – Hackenslash Gains More Rooms

- I Change the title of the window to be "Hackenslash!" Just because it was cool.
- I create a Map object to contain rooms, and a MapMaster class to hold multiple maps.
- I add a second room connected to the first via a portal.
- Neighboring rooms connected to the current room via portals are now displayed.
- I fix some clipping bugs where the walls aren't displaying correctly when partially outside the viewport.

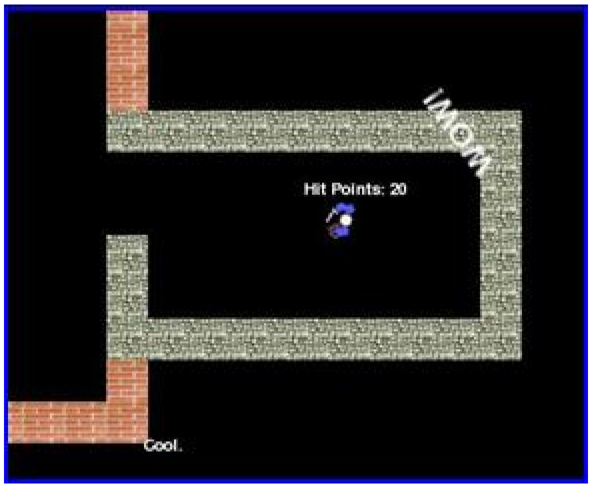
Hour 6 – Wherein We Practice Our Mad Drawing Skillz

- I add a door class, and set up the maps to accommodate doors (since these need to be shared by two rooms). *(Editorial Note: Too bad I never got to actually use these!)*
- I create 3 more wall tiles, combined them together into one "sheet"

- Walls graphic changes based on type.
- I make a simple top-down player-graphic.

Hour 7-8 – Rotations and Exclamations!

- I figure out how to have PyGame rotate bitmaps.
- I have the test-player spin slowly in a circle. Lots of tweaks are necessary to correct for size changes as he rotates.
- I learn how to use fonts in PyGame, and I build some classes to display and animate text.
- I add a manager class to automatically handle all this animated text. Having them be "fire and forget" means they'll be much easier to maintain, and I'll be more likely to use them in the future.

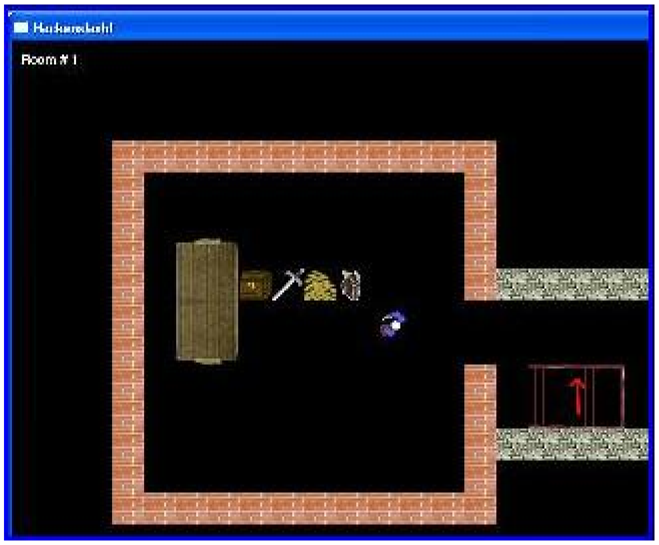


Hour 9-11 – Feature-ific!

Once again, I face a troublesome "What next?" decision.

Rooms need more interesting features, so a "feature" element goes onto the list. I don't know what sort of behaviors features should have, so I decide to go from specific to general. I decide on three static features that could be found in a dungeon room: A rug (visual only), a pillar (blocks movement and line-of-sight, like a wall), and a staircase (blocks movement from some directions along some tiles, and "teleports" you to a new location at the end.)

I determine that features can be larger than a single square, and should be able to be rotated any ninety-degree step. *(Editorial Note: In retrospect, a very dumb decision – I spent way too much time on this feature and derived very little benefit from it.)*



I end up spending about three hours working on "features," between graphics and fighting some poorly designed code.

Hour 12 – 13 – We Want Loot!

I create art and code for items. It's amazing how much time can get sucked away doing artwork. It's particularly annoying when it still ends up looking like dumb programmer art no matter how hard you work on it.

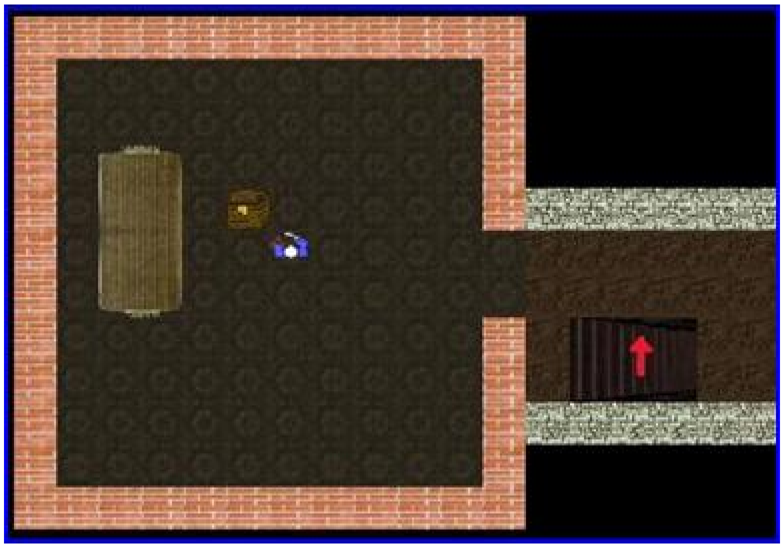
I add a lot of details for items, including their value, size, equipment slot, graohics image data, and so forth. They aren't interactive yet, but at least they are being drawn in their correct locations in the room.

Hour 14 – Carpets

I'm clearly behind schedule, and what do I do? I decide that the black background is too ugly, and I put in floors.

After working on the floor graphics, I discover that I'd forgotten to put a transparent background on the player and item graphics. So I spend more time fixing them.

But the floors look cool. Cooler than black, at least.



Hour 15-16 – Click! Click!

- I implement mouse control and event handling.
- Add player responses and movement to mouse events. The movement is blocky, not yet smooth-scrolling.
- The player isn't yet leaving the rooms or checking for collisions<
- I fix several bugs
- I create a prettier-looking staircase in GiMP.

By now, I've already crossed the threshold of Hour 17, so I start to get a little bit nervous. I should now be 2/5ths of the way done with this game – the second "working day" of development. While what I've got up and running is pretty impressive thus far, I recognize just how much I've got left to do. I now have 4 more working hours to finish up basic player interaction according to the schedule. It's going to be tight… but I still don't regret putting those floors in!

Hour 17 – Smooth Move, Until You Hit the Wall

- More time is spent cleaning up graphics and fixing bugs.
- Add collision detection and smooth-scrolling to player movement.
- Player can now move multiple steps (turns) on a single mouse-clock.

Hour 18 – Crossing the Threshold

- The player now goes through portals into other rooms.
- This exposes a cosmetic bug with overlapping walls and floors between adjoining rooms that is really jarring.
- More bug-fixing on rotated portals not allowing / prohibiting movement.

Hour 19 – Stairway to Heaven, Menu Hell

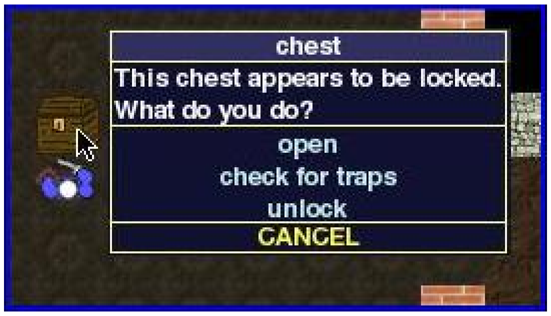
My brother volunteered to do some music for the game. He did the music for Void War, and is really good. This reminds me that I need to do sound (and now music) for the game. This looks pretty simple in PyGame, so it shouldn't take too long *(Editorial Note: I never did get around to this, sadly. Hackenslash ended development as a silent game.)*

My next goal is to handle interactions with creatures and items in Hackenslash. I'm really fond of how The Sims and Neverwinter Nights did context-sensitive pop-up menus that appeared when you wanted to interact with a game object. I'm thinking of doing something very similar here.

- I get stairways to properly 'teleport' the player to a new room.
- I do some research on the Internet and through PyGame docs to see if anyone already has some kind of menu system in PyGame already done with an open license that I can use. I don't find anything.
- I start work on the menu system.

Hour 20 – 21 – What's On the Menu?

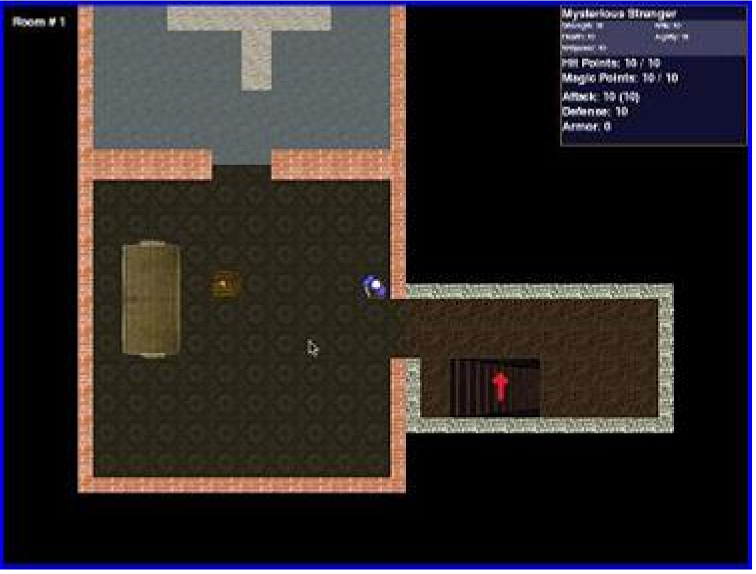
- I continue work on menus. Menus can be associated easily with the object that spawned them, making it easy to 'call back' to the originating object with a callback to handle the player's selection.
- I start work on item menus. They simply pop up and allow a selection at this point – clicking on an option does nothing but close down the menu.


chest
This chest appears to be locked. What do you do?

Hour 22 – Falling Asleep at the Wheel

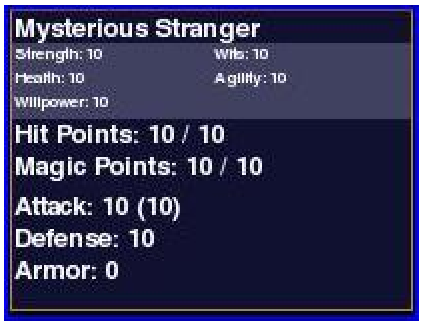
- I continue work on items – trying to make them functional and have them respond to menu commands – this includes putting more contextual information about the menu in the player's "action" queue. Right now there's still not much that the item actually does, except create an animated 'bark' reflecting command information.
- I improve how movement is calculated for performing an action, which allows more flexibility.

It's getting very late – I'm finding myself "zoning out" while working on this. If I wasn't paying attention to the total time – sure, I'd work through it. Since I'm on a limited number of "work-hours" for the development of this game, having a less-productive hour is really bad news. It's interesting how your priorities shift when you consider time a limited resource. So I go to bed.



Hour 23 – Stat Attack!

- I modify (and actually start USING) some of the attributes class that I created in Hour 1.
- I create a window in the upper-right corner to actually display those stats.
- I optimize said window so that it is just a bitmap that quickly blits on screen, rather than writing out font instructions every frame. I update the bitmap only when I detect a change in the stats.


Mysterious Stranger
Strength: 10 Wits: 10
Health: 10 Agility: 10
Willpower: 10
Hit Points: 10 / 10
Magic Points: 10 / 10
Attack: 10 (10)
Defense: 10
Armor: 0

Hour 24 – Player Menus

- I wrap up optimization on the stats window.
- I create a pop-up menu that appears when the player clicks on his avatar.
- I create sub-menus for potion usage, casting spells, etc.
- I fix some bugs in the menu response handling.

Hour 25 – I Take A Saw To Floors and Walls

This morning I had an idea in the shower (why is it that the shower is such a great place to have ideas?) for eliminating the overlap problem for walls shared by rooms (see the Hour 18 entry). What if I only draw half of the exterior walls (walls 0-3) in a room – the half facing into the room? That way there'll be no overlap with walls of neighboring rooms, and I don't have to add some complicated logic to detect and resolve overlaps.

I begin work (yet more engine / foundation work) on this "quick fix." Unfortunately, it really complicates my room drawing routing (for floors as well), and turns out to not be as quick a fix as I had hoped. It takes about an hour to create and debug this system to make rooms more seamless. But the results are much nicer.

While debugging the code, I discover a few more movement bugs related to crossing portal thresholds with negative offsets, and fix that.

Intermission – Crisis Management!

At this point I realize I'm more than 3/5ths of the way through my schedule, and that I have less then 15 hours left to finish the game. I go through my list of Things To Do, and assuming one hour per feature, it'll take around twenty-five hours to finish it all. That's ten hours over my limit I'm halfway through the time I'd allocated to getting the environment active, and I've not really started on it yet. **The project is officially in jeopardy.**

Going overtime is really not an option I'm allowing myself. Hiring additional help or buying some more code / resources isn't an option, either. Since I'm only counting real "development" time for my 40 hours, I'm already pretty maxed out on productivity – I can't think of any way to be more efficient. Aside from spending a lot of time on the Internet looking for magical free solutions to my problems, I really have no choice here but to cut features, and see what I can do to simplify my design.

- Doors: **CUT!** I *REALLY* want doors in this game. This is the most painful feature to cut – especially since I've already spent some time working on them already. But there is too much work to be done on them - especially considering the AI has to deal with them. I probably need 2-3 hours to get them working, and I don't have the time.
- Inventory: **SIMPLIFIED!** Forget having a "back inventory" of usable items you don't currently have equipped. Anything you aren't equipped with gets converted into money immediately.
- Traps: **SIMPLIFIED!** I wanted to have all kinds of nasty traps with interesting, debilitating effects. Not gonna happen. Traps will have a simple visual when they go off, and just do damage and temporarily increase the chance of running into a random ("wandering") monster
- Bows (Missile Weapons): **CUT!** It's going to be just melee and spells in this game.
- Saving / Loading the Game: **SIMPLIFIED!** Only your character will be saved and loaded, not the state of the world. *(Editorial Note: I didn't even do this!)*
- Particle Effects: **BACK-BURNERED!** These are getting dropped to the bottom of the priority list. I really doubt I'll get to them. I wanted some cool particles for animations for spell effects… but that is unlikely to make it.
- Spells: **SIMPLIFIED!** I had a concept for spells where you could "find" spells in the game via scrolls, and that there'd be over a dozen spells you could use. Well, as much as it pains me… I don't see that happening. I'm going to have only a handful of spells available now: Heal, Damage, Debilitate, Buff, and Recall. To deal with increased level, I'm going to allow the player to "beef up" the spells by increasing the number of magic points that go into the spells
- Monster and Player Animations: **CUT!** I don't have the artistic talent to do a great job, anyway.

While deciding what I won't do (or what I'm dropping down in priority to highly-unlikely status), it's equally important to decide what I absolutely must do – what needs to be made top priority.

There are a lot of elements to the planned game that I consider very important – like searching for traps, finding secret doors (well, secret portals now), and unlocking chests. But the core of the game is combat. If that's not there, nothing else matters. So I decide I must focus on that, and getting combat working is top priority. I set a goal that by hour 30, monsters will be working – at least enough that the player can kill them.

With my adjusted priorities in place, I move on to continue development.

Hour 26 – The Roll of the Dice

I work out the core "dice" mechanic, something I've had in mind for about the last week --- how the random chances come into play. Since we're not limited to using real dice, we can make a random number of any range we want. Like 1-33, or 6-17. So what I do for a "standard" roll is a weighted, random roll against the total of the attacker's score plus the defender's score. If the number is above the defender's score, the attacker wins.

For example, let's say I have a total attack rating of 15. I'm attacking a monster with a total defense rating of 10. My odds are 15:25 (25 is 15 + 10), or 3:5. So the game will generate a number between 1 and 25, and if it's above a ten, I win.

Damage uses a slightly different type of roll. I add the defender's "armor" rating together with the attack's "damage" rating. I generate a random number between 1 and this total, and then subtract the armor rating. If the total is less than one, the defender takes no damage. Otherwise, he takes the amount indicated. So if a monster with a damage rating of 10 attacks the player with an armor rating of 5, the game will roll a number between 1 and 15, and then subtract 5 for the damage.

This explanation took more time to document here than it did to write. The dice-rolling code looks like this:

```
from random import randint
def StandardRoll(attackerScore,defenderScore):
    if (attackerScore<1): # If the attack rating is 0, the attack
always fails
        return 0
    if (defenderScore<1): # Otherwise, automatic success on defense
rating of 0
        return 1
    roll = randint(1,attackerScore+defenderScore)
    if (roll>defenderScore):
        return 1
    return 0


def DamageRoll(attackAmount, defenseAmount):
    if (attackAmount<1): # 0 attack rating? No damage will be done
        return 0
    if (defenseAmount < 1): # Man, don't screw with us with negative
numbers
        defenseAmount= 0

    total = randint(1,attackAmount + defenseAmount)
    total -= defenseAmount
    if (total<1):
        return 0
    return total
```

Filling out the hour, I decrease the window size for drawing the dungeon to get a slight framerate increase --- the section to the right will now be entirely User Interface stuff. I also make sure the player's movement is corrected for frame- rate.

Hour 27 – Building A Monster

I have a lot to worry about to get monsters working. I have to change the game update system to make it turn-based. The player needs a variety of ways to interact with creatures (bribery, combat, spell-casting). The monster has to interact back. There's AI and pathfinding considerations. And graphics! DO I display the monster with the same top-down perspective?

I can't worry about all these at once – I need to start small. Just putting a monster in the room and displaying in the correct position on the screen is a good start.

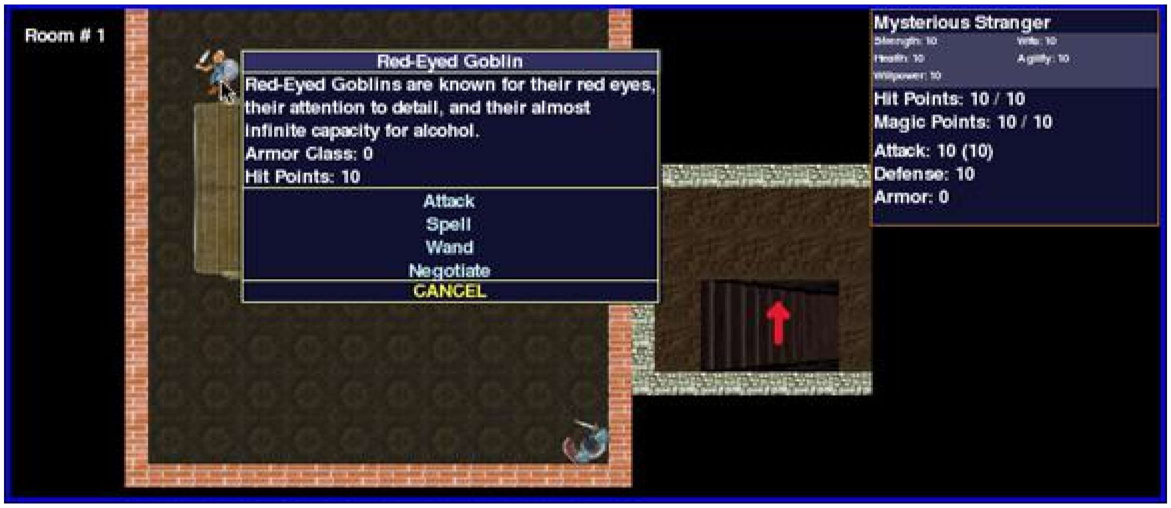
- I create a "monster" class, derived from creature.
- I create an "ActiveAI" list for the main game loop to deal with activated monsters.
- I work on art for a monster – which takes me the rest of the hour (and it still doesn't look very good- I should have just used a smiley-face).

Hour 28 – The Monster Appears, and the Player Has to Change His Armor

- I get the display routines working for monsters
- I make the monster block the player's movement.
- The perspective on the monster is totally different from the player, but I don't care. But the player graphic looks more terrible by comparison, so I re-do the player graphics.

Hour 29 – Time to Come Out Swinging

- I create a menu and menu-responder for the mnster.
- Player attacks are working
- Main loop responds to the monster dying.
- Experience points are awarded for killing the monster.



Red-Eyed Goblin
Red-Eyed Goblins are known for their red eyes, their attention to detail, and their almost infinite capacity for alcohol.
Armor Class: 0
Hit Points: 10

Mysterious Stranger
Strength: 10 Wits: 10
Health: 10 Agility: 10
Willpower: 10
Hit Points: 10 / 10
Magic Points: 10 / 10
Attack: 10 (10)
Defense: 10
Armor: 0

Hour 30 – With this Sword, I Thee Slash

- I create the first of true equippable "items" (a '+1 sword')
- Monsters drop their items when they die.
- Game loop update gets broken into turn-based and non-turn-based updates.
- Massive changes to player movement / actions based on the turn-based architecture.

Hour 31 – Entering the Final Stretch

- Monsters get emotional – I give them three 'attitudes' towards the player, which can be adjusted by player actions.
- Money is added to the game, in the form of silver
- I get the "negotiate" action working, where you can bribe the monsters to leave you alone if you haven't attacked them yet.
- Monsters attack a player if the player is right next to them (and they've not yet been bribed).
- I fix a bug where the player decides to treat a monster like a chest, and runs up to them to begin the negotiation. This gives the monster a free attack before the negotiation happens. That kinda ruins the point, but it's funny.

Hour 32 – Why Can't We All Just Get Along?

- Finish negotiation logic
- Add monster hunt / chase routines
- Monster chooses between spellcasting attacks (stubbed out) or melee combat

In order to save time, I cheat on monsters chasing players through a portal. I just assign a random chance of the monster going through the portal each round. This gives the player a better chance to escape, anyway.

Hour 33 – Wandering Artwork

- Monsters now wander around if they aren't actively attacking the player.
- Artwork for some items and potions

- I add potions as items – they don't work yet, but they appear

Hour 34 – 35 – Drink Me!

- Potions get added to the player's potion inventory
- Potions have the appropriate effect on the player's stats. Healing Potions restore hit points, Essence potions restore mana points.
- I add pure treasure (which gets converted immediately into silver) as an item.
- I start working on equipment items.

Equipment items are a little tricky. Since I don't have an "inventory" of unused equipment, you can't carry a spare sword around with you. If you pick up an item and you already have another item in the "slot", the game asks you if you want to replace the old item, or cash in the new item for treasure. For items where only a single slot is permitted (like a weapon, or armor), that's not too hard. Wands and rings are more complicated, since those can be in one of several slots.

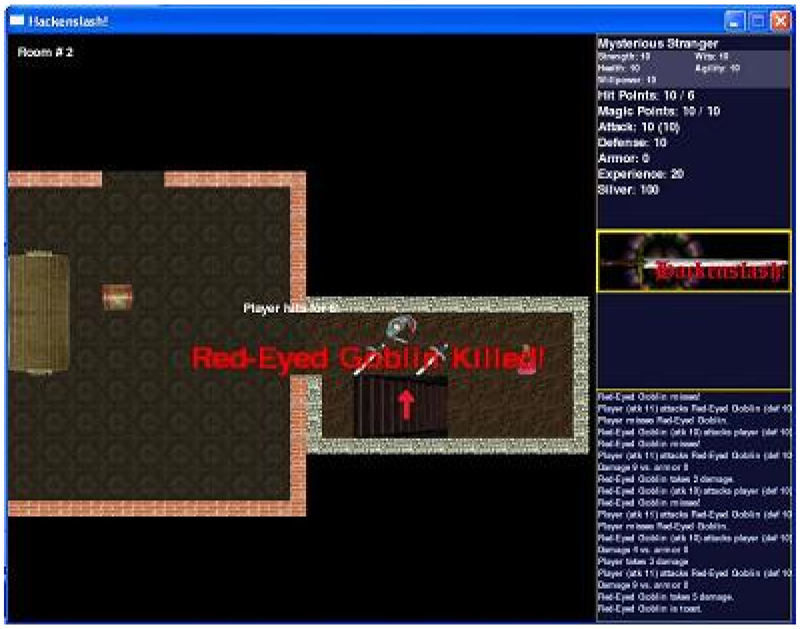
After almost an hour of working on this, I realize that I've got the player running through two menus (one to pick up an item, and another to choose what to replace) for items is really ugly. It's far better to have the game be "smart" about it and reflect these options in a single menu. I end up overhauling some of what was already done, throwing away a little bit of code, to make this change to a simpler system.

Hour 36 – HACKENSLASH!

- I create a "Hackenslash" title panel with a sword graphic, and link this bitmap into the right UI panel display
- Add the logic for the "monster display" panel to display the information on whatever monster last had the mouse pass over it.

Hour 37 – We Don't Need No Stinkin' Word-Wrap!

I create a scrolling "text box" class to display game mechanics information. This is useful both for player feedback and for debugging purposes. Without any user controls (like scroll bars), development goes extremely quickly. It actually takes more time to send it lines of text from various events in the game (picking up items, attack results, etc).



```
Mysterious Stranger
Strength: 10          Wits: 10
Health: 10            Agility: 10
Willpower: 10
Hit Points: 10 / 6
Magic Points: 10 / 10
Attack: 10 (10)
Defense: 10
Armor: 0
Experience: 20
Silver: 100
```

```
Red-Eyed Goblin missed!
Player (atk 11) attacks Red-Eyed Goblin (def 10
Player misses Red-Eyed Goblin.
Red-Eyed Goblin (atk 13) attacks player (def 10
Red-Eyed Goblin missed!
Player (atk 11) attacks Red-Eyed Goblin (def 10
Damage 9 vs. armor 0
Red-Eyed Goblin takes 3 damage.
Red-Eyed Goblin (atk 13) attacks player (def 10
Red-Eyed Goblin missed!
Player (atk 11) attacks Red-Eyed Goblin (def 10
Player misses Red-Eyed Goblin.
Red-Eyed Goblin (atk 13) attacks player (def 10
Damage 4 vs. armor 0
Player takes 3 damage
Player (atk 11) attacks Red-Eyed Goblin (def 10
Damage 9 vs. armor 0
Red-Eyed Goblin takes 5 damage.
Red-Eyed Goblin is toast.
```

Some of my preliminary work back in the first 20 hours is FINALLY paying off - there's a good framework in place for calculating the bonuses for skills based on equipment. All items have a 'dictionary' (a great Python construct) of bonuses and penalties to SOMETHING (the name of the roll). So when I get ready to make a roll, say, an attack roll, I just run through the equipment list and apply all adjustments with an "attack" key. I'll do something similar for spell effects.

Next, I work on the menu for opening chests – and decide to simplify it significantly. Why not just assume the player is ALWAYS going to check it for traps, and will ALWAYS attempt to unlock it if it's locked? The extra steps in there are tedious. No sense in having trivial choices floating around.

Hour 38 – A Little Bit of Magic

I'm frantic now. Three hours left – I have to work on ONLY what MUST be in the final game.

- I generate five types of spells – three that are cast on the player, two on monsters.
- I add extra "levels" for the spells to give them slightly more variety. The levels vastly increase the power (and cost) of the spell, and are only available as the player's "magic" skill increases.
- I make the player-menu non-static, so I can add menu options that become available only under certain circumstances.
- I add the ability to "level up" through a set of menus that become available from the player menu when the player's experience points exceed a certain level-based threshold.

Hour 39 – Crisis Management, Part II

What I'm missing:

- Searchable secret doors and items
- Quest Items – and a place to return the quest item to win the game
- Sound Effects
- Music
- A merchant from whom to purchase equipment
- A place to rest
- Random equipment with different bonuses and names
- Random monsters
- Graphics for the new equipment, monsters, and merchant.
- Random chest traps & locks
- Creatures casting spells
- Wands
- A larger dungeon with a quest item (and maybe boss monster) at the end.
- The ability to sneak around (?) to avoid being attacked by the monsters.

There is no way I can get these all in with a little over 100 minutes to go. I can handle maybe three or four of these if I'm fast. What is the minimal set of items needed to make a playable game?

I decide on the following

- The player must be able to rest to regain health and magic points
- Monsters and treasure should respawn while the player is resting
- Random equipment types and abilities should be generated with the treasure, and should gradually scale up by player level
- Random monsters should be created with abilities and numbers that scale up to the player level.

The initial room (the one up the stairs) becomes the the "rest" room (pay no attention to the awful pun – aw, heck, it's part of the charm). There will be no monsters in this room, ever. I add a new dynamic entry into the player menu when he's in a rest area – the "rest option." Then I create a routine to "re-initialize" the dungeon while the player is resting. Right now it's a stub.

I add the remaining equipment graphics in GiMP. I have no time to make them look any good. I simply draw them in 32 x 32, blur them in areas to hide my awful artwork, and then sparingly add some highlighting so they don't just look like a blurry mess. Now they look like a touched-up blurry mess.

For the random equipment generator, I have the game pick one or two random abilities and slap them onto the item, with a possible range based on the player's "level" *(Editorial note: Unfortunately, the player's level is never actually displayed on the screen anywhere. D'oh!)*. Certain items have mandatory effects – a sword has to improve either your attack or your damage rating, armor must improve your armor rating, and a shield improves your defense rating. I add these constraints… and we're now already well into hour 40.

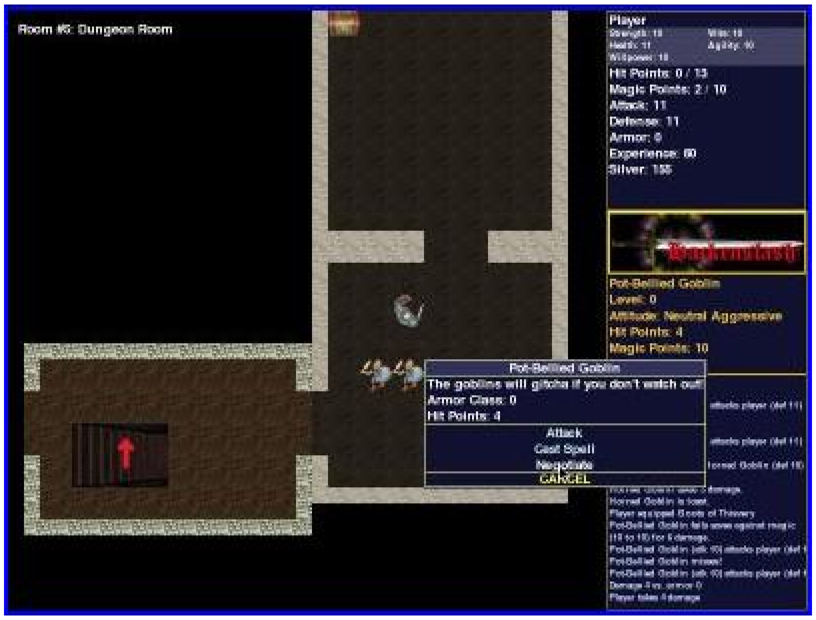
Hour 40 – The End Is Near

I have no time to create actual unique monster types – so I frantically work on just changing the names and stats of the one monster I have – the goblin. I throw some silly-sounding adjectives in front of the goblin names, and I have them "level up" similarly to the player based on the player's level. I finish up the "re-initialization" routine to repopulate the dungeon while the player rests. I finish up some routines to deactivate monsters that are far away from the player, and to make them become active when the player enters the room.

I add a couple more rooms to the dungeon, and then I'm out of time.

Post-Development Work: Wrapping It Up With a Bow

After talking with a friend, I'm convinced that I shouldn't work on fixing any bugs except the ones that crash the game. Since the purpose of this project is to show off what can be done in 40 hours, I leave it in that state. There are lots of non-critical bugs: stats aren't being displayed properly in the right-hand panel; I don't think goblin hit points are increasing with level; I'm not convinced monsters are properly de-activating; monsters are ignoring collision detection; treasures are appearing inside interior walls; monsters are ignoring magic attacks from long range...



Hit Points: 0 / 13
Magic Points: 2 / 10
Attack: 11
Defense: 11
Armor: 0
Experience: 80
Silver: 155

Pot-Bellied Goblin
Level: 0
Attitude: Neutral Aggressive
Hit Points: 4
Magic Points: 10

Pot-Bellied Goblin
The goblins will gitcha if you don't watch out!
Armor Class: 0
Hit Points: 4

Horned Goblin is toast.
Player equipped Boots of Thievery
Pot-Bellied Goblin [illegible]
Pot-Bellied Goblin misses!
Damage 4 vs. armor 0
Player takes 4 damage

I leave them, and focus on the crash bugs. This means playing the game quite a bit. It's actually pretty fun. Not play-this-game-for-hours fun, but certainly an amusing diversion. Once the crash bugs appear to be mostly quashed, I work on the distribution.

Py2exe is supposed to make things ridiculously easy, allowing you to package a Python program as a native Windows executable, including everything you need so that users don't need to have Python installed to run the program. It's not quite so easy as all that. First, it tells me there are missing modules, and the executable crashes in the font code. I check on the Internet, and find that the 'missing modules' are red herrings, and what I'm missing is a font file that needs to be copied over manually.

After I do this, I get another crash in the executable. This is much harder to track down. It turns out that I need an icon for the window – something that didn't seem required in the non-compiled version. Once I have that in place, everything works. I run a few more tests – and I seem golden. WHEW! I create a zip file of the distribution (I'm not going to bother with an installer / setup program), and I'm DONE.

My week-long project actually took two-and-a-half weeks of real-world, part-time effort. It's definitely not everything I hoped it would be, but I'm pretty proud of what I accomplished:

http://www.rampantgames.com/hackenslash.html

Aftermath: The Post-Mortem

Interestingly enough, the lessons I "learned" from working on a game with such impossible constraints are very applicable to the development of any game, on any budget and schedule. A lot of the lessons Hackenslash taught me weren't completely new to me – but these sorts of things often bear repeating.

Without further ado, here are the Top Ten Lessons Hackenslash taught me:

Lesson 10: Doing something like this really was worthwhile

I didn't think I'd have time to do something like this when I first found myself challenged. But now that it's done, I found it not only taught me a few things, but it increased my enthusiasm for continuing work on the project I put on hold for this. You wouldn't think that working on Yet Another Game would feel like a vacation, but it did. And after all, I didn't lose that much time – only forty hours. This was a fun experience – one I'd be happy to repeat again in the future.

Lesson 9: Cutting features isn't always free

Some of the last-minute changes to Hackenslash really blew the game balance out of whack. The inability of monsters to cast spells, and the lack of need to for the player to 'conserve resources' as he pushes deeper into the dungeon trivialized some of the challenge to the game. If those features were going to stay 'gone,' the game needed another design pass to re-balance it and improve the modified gameplay. In other words, cutting features introduced an additional cost to the development of the game. This made me wonder how many retail games were released in a terrible state because the development team didn't have time to re-visit the game design after features were cut to meet schedule.

Lesson 8: Do the important stuff first

I found I tended to be more productive, efficient, and make more progress on the game when I was in 'crisis mode," realizing how tight my deadline was and making a conscious decision to (usually) only work on the pieces that made the biggest difference in the game.

I think I may run through a similar exercise with all of my future projects: I'm going to try to break my development time into, say, 8-hour segments, and play a little game with myself: If I pretend that I only have those 8 hours to 'finish' the game, what could I do that would make the biggest difference in those 8 hours? I don't know if it would pay off as well at the end of the project as the beginning, but it's worth trying.

Lesson 7: Scope will expand to exceed your budget and schedule

Every programmer I've ever met tends to underestimate the time required for him or her to complete a feature. Add to that the dreaded 'feature creep,' and you can guarantee your project is going to be way over schedule. I'm not one of those guys who believed that "feature creep" is always a bad thing. I think some of the most killer features – the ones that turned games into hits – often came about as a form of "feature creep." But new features are seldom free. You will have to make room for them in your budget – and that often means cutting other, less worthy features. This project taught me a little bit about being ruthless in cutting features. In truth – if I had the option, I should have added an extra ten hours to make the game truly "work" – but only after cutting all the fat (like doors, magic wands, etc.)

Lesson 6: Get the Game Playable as Fast As Possible

The sooner you are able to get things on screen and start playing around with it, the sooner you can revise and improve your design, weed out the things that don't work, and come up with great ideas for making the actual game better. It also helps you catch bugs (especially those ugly design bugs) earlier, which makes them much easier (cheaper) to fix. It also aids you in sorting through priorities.

Lesson 5: It's sometimes much faster to throw away old code and start over

I only ended up completely throwing away some code and starting over once in this project. While I can't know with a certainty if I really saved time by doing this, I suspect I would have been fighting the design flaws of the original method all the way to the end of the project. On the flip side – throwing away the old rotating feature code and starting over with a better design might have saved me some trouble.

Lesson 4: Python Rules!

I can't believe how quickly many features came together using Python as opposed to, say, C++ or even Java. Things like typeless variables, dictionaries, and extremely easy-to-declare lists (allowing a mixing and matching of object types) made it very easy to implement content lists, attribute handling, spell effects, and so forth. I was already a fan of the language, but now the prospect of using Python, tied into a high-level 3D engine, has become extremely appealing to me.

Lesson 3: Don't underestimate the art requirements

Looking up source art, drawing, tweaking, testing, and re-tweaking artwork… even little 32 x 32 bitmaps – consumed a great deal of my time – and the results weren't nearly as satisfying as what I'd have gotten if I had devoted those hours to programming.

Would an experienced artist taken less time? Undoubtably – though the difference probably wouldn't have been too extreme. Would their results have been better? Absolutely. Be careful not to trivialize the effort it takes to generate art for your game, whether you are doing it yourself or getting someone else to do it for you. It can suck up a great deal of time if you aren't careful.

Lesson 2: I need to be more efficient in my use of time.

A night where I'd devoted four hours to working on the game often ended with only two hours (or less) of actual development time taking place. Some of the time went to documenting what I was doing, but I also found myself losing focus, taking extended breaks, not immediately returning to work after a minor interruption, surfing the 'Net, playing (quick) games, or whatever. Now, taking breaks during long stretches of development is a good and healthy thing. But by recording my actual productivity, I was pretty surprised at how inefficient I was with my usage of my development time.

I'm not getting paid by the hour. Better use of my time means I can get more done AND have more 'free time' to do other things. So I'm going to be making a concentrated effort to improve my use of time when I am "on the clock."

Lesson 1: *IT CAN BE DONE*

While Hackenslash in its current, 40-hour incarnation is hardly a poster-child for high production values, I think it demonstrates how much can be done – on a fairly complex game – with no budget and very little time by a single developer. Given more time – and even a fairly insignificant budget – or the help of a few friends – who knows how much better it could become?

The bottom line is this: If you want to develop games, nothing is stopping you. You can find the time. You don't need a big budget or fancy tools. You don't need a team of specialists. You don't need years of training. All you need is the will to make it happen.

And that's the most important lesson of all.

About the Author

Jay Barnson has been writing computer and console games professionally and as a hobby for over twenty years, starting with his initial struggles to put graphics on the screen with the Sinclair ZX80 and Commodore 64. Past credits include Twisted Metal, Warhawk, Jet Moto, Animorphs, and Extreme Championship Wrestling. More recently, he has been developing games as a part-time independent developer for Rampant Games. His first independent game release, Void War, recently won Independent Multiplayer Game of the Year for 2004.

Discuss this article in the forums

Date this article was posted to GameDev.net: **7/6/2005**
(Note that this date does not necessarily correspond to the date the article was written)