Woody Tigerbaum's Interplanetary Package Delivery

Frequently Asked Questions List (FAQ)

Author: Dave LeCompte
Contact: pyweek at bigdicegames dot com

Contents

How much of the week did you actually spend on the game?

Did you think about doing music?

How did you do feature X?

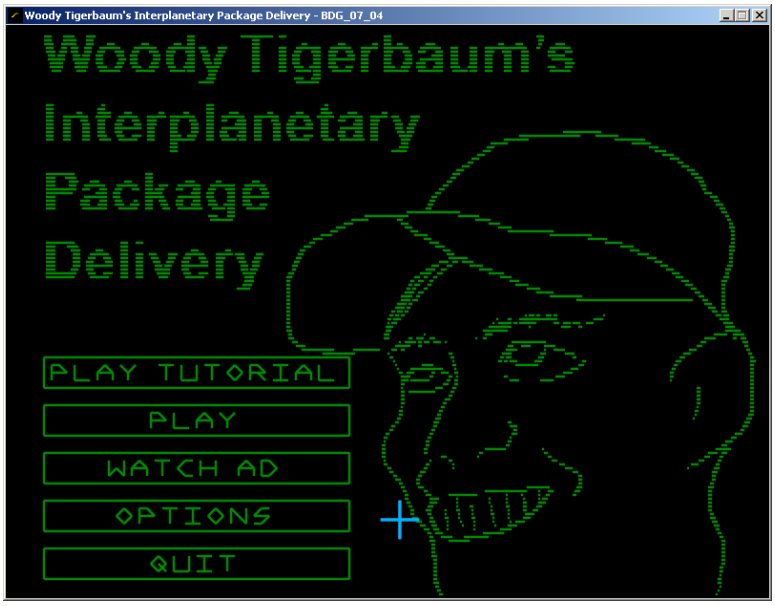
- Raster art intro
- Gravitational simulation
- Game states
- Options menu
- Computer player
  - Any humans playing have finished the current level
  - On a moon, orbiting a gas giant
  - On a stationary planet, with a gas giant somewhere in the level
  - The other 80%

Are there features that didn't make it into the game?

- Smarter AI
- More accurate physics
- More stable physics
- Save/Play replays
- Attract mode
- New objects:
- Slingshot Art
- More event sounds
- Random levels / more levels
- Mac binary build
- Online gameplay

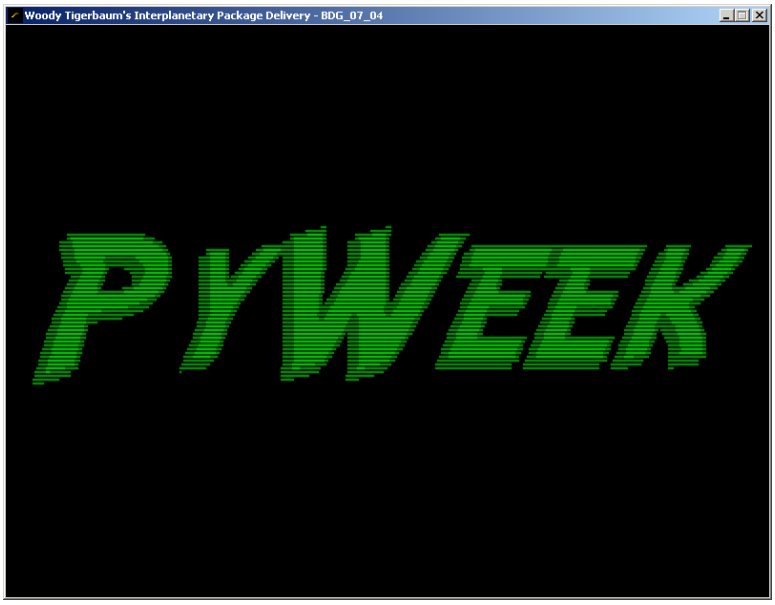
What are your plans for future versions?

This FAQ rocks - how is it made?



Are these questions really frequently asked?

For the most part, no. But a FAQ seems marginally more presentable than calling it an outline.



You really made a game in a week?

Yes, I did. I started with an empty directory at 5pm Saturday, March 31st, and by Noon Saturday, April 7th, I submitted the contest entry.

Did you work alone, or as a team?

I worked alone. I mentioned the challenge to several Python programming friends, and some of them expressed mild interest, but nobody really wanted to help out.

I think that the next time I do one of these things, I might talk to a buddy who has musical talent or who can whip out 2d character art quickly.

What tools did you use?

I used a PC running Windows XP. My editor was emacs. I didn't use IDLE or any other integrated development environment.

I was concerned that I might lose my work, or that I might make a change I wanted to back out, so I set up a Subversion repository on my Linux box. I didn't check my work in all that frequently, so I still would have had a fair amount of ground to recover if my main hard drive went out.

For graphics, I used Adobe Photoshop CS2 to manipulate images, with a Wacom Pen Partner tablet for all the intro art.

All the sound was edited and manipulated in GoldWave. I saved the sounds out in WAV format, and then used oggdropXPd to encode them into OGG for distribution.

I grabbed a screenshot and turned it into an icon using png2ico.

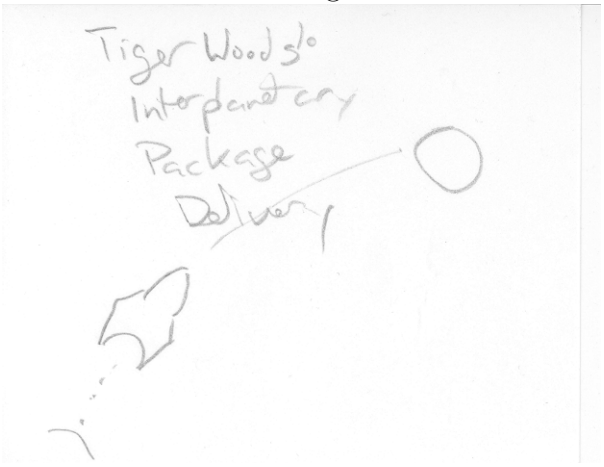
Did you have the idea picked out ahead of time?

Before the challenge started, there were 5 potential themes announced:

- The only way is up
- Underneath the radar
- One way or another
- Don't stop 'til you get enough
- The final countdown

I spent some time before the challenge brainstorming possible directions to go with each of those themes. Some themes I liked more than others (the radar theme was suggesting a flight sim to me, in particular), but I wanted to get a few options for each potential theme, so I'd be ready when the selection was announced.

Now, I have a bulletin board where I keep little cards with ideas for games I'd like to screw around with someday. Some of them are interesting for gameplay reasons, some of them are technologies I want to experiment with. Some of the cards on the board just represent technologies - most of which lead into games.



One of the cards on the board was a space-golf hybrid, which has been there for at least a year. I looked over the board, trying to figure out other game ideas that would fit both the theme and the time constraint, but the space-golf game was the only one that worked at all.

Early Brainstorming Ideas

These were some of the ideas I sketched out that fit the candidate themes:

Jumping Game

One of my favorite games from the mid-eighties was "Jumpman". I really liked the feel of the jumping mechanics - if you could collide with a platform, you could scramble up onto it. I was also very impressed with the how different each level felt from one another, by only using a small number of game mechanics.

So, I considered making a jumping game where the platforms were slowly moving downward, forcing you to scramble up the screen to stay alive.

Flight Sim Photo Capture

This could fit with "Don't stop 'til you get enough" or "Underneath the radar". The player would fly around a (pretty much static) landscape. I'd have to make a simple dynamic model that felt good for an arcade-y feeling airplane.

If the theme was "Don't stop", the gameplay could be to go out in the world and take pictures of certain landmarks (don't come back home until your photo album is full). If the theme was "radar", the idea would be to get to certain places on the map without being seen by fixed radar emplacements. I imagined the radar being up on the top of canyon walls, so the only way to get to your destination would be tricky flying through some tight canyons. (Very much like Rebel Assault.)

I spent a little time right before the challenge started experimenting with Blender to remind myself how to use it, in case I needed to make some landscapes.

Side scroller shoot-em-up

Under the theme of "One way or another", a side-scroller where the direction of the scrolling changed over time could be kind of fun.

This actually ties back to a game idea I toyed with back in the 80s, but never did anything with. If I recall correctly, back then, my idea was that there'd be "black boxes" which gave you unpredictable powerups, including turning a corner and flying in a new direction.

I think if I were to tackle this today, the scrolling would be completely scripted beforehand, perhaps with some "fork points", allowing you to select which path to go down (a little like Spy Hunter, or perhaps OutRun).

Bomb Defuser

The only thing I could come up with for "The final countdown" was a literal take on the countdown - a mad bomber would drop dynamite bundles with easy-to-read LED countdown timers. Your job would be to defuse the bombs by running over them before they counted down to zero.

The only interesting bit would be that the bombs wouldn't all count down from the same starting value - some would start at 30 seconds, others at two minutes. This way, you'd have to run around the screen based on the bomb timers, not just follow the bomber around.

Pipe Race Game

This game would be a fairly straightforward race game inside a pipe. Depending on the theme that was announced, there could be a number of different directions to go. For "One way or another", the world could rotate periodically, causing the player to slide around the pipe.

For "Underneath the radar", there could be periodic radar pulses coming down the pipe at the player, and obstacles that the player could hide behind. You wouldn't want to be hit by the radar pulse, but at the same time, you wouldn't want to run into the solid obstacles, either.

Ideas Within the Theme

When "There's No Way to Go But Up" was announced, I jotted down the golf idea, but I wanted to brainstorm a little longer before working on code. I considered several alternate game ideas:

"Hot Air Balloon Raiden"

This would be some sort of scrolling action game, perhaps with some sort of shoot-em-up gameplay style. I couldn't figure out how to make a fast paced game around a hot air balloon. And I couldn't figure out how to make it interesting or fun.

"Hot Air Balloon Lunar Lander"

There was a game I typed in out of the old Creative Computing magazine years ago (anybody remember typing in programs out of magazines?) that had a sprite coded in Apple 2 assembly of a hot air balloon. Your goal was to land on one of the safe landing pads, while being blown around by (unmarked) wind layers. There was a 2d terrain that you didn't want to blow into. Again, I couldn't figure out how to make this fun.

Some sort of block stacking game

Sort of a Jenga-ish kind of game, but with more interesting pieces. I think this could be fun, but I'd need to be more comfortable with physics code (whether it be full-on ODE or something simpler, like the "Hitman Physics" constraint stuff I keep tinkering with).

I considered adding time pressure to this game by making the structure you were building slowly sink into a swamp.

Rock Climbing

Several years ago, I saw Chris Hecker show off a demo of a rock climbing game he had been working on at the time. He was using it as a context in which to explore inverse kinematic solvers. I looked at it and said that the Hitman Physics stuff might do a fine job of the math he needed to do. I think it's still possible, but I suspect that I'd have to figure out a good control system in order to make it fun. How do you let feet dangle? How do you grab and release the rock? Would there be ledges? Chimneys? How do you handle fatigue? Is it something you could meaningfully track per limb? Would that be fun?

Satori Maze

Playing on "Way", my mind went to "Tao", and thinking of a more meditative game. In the end, the only thing I came up with was a sort of psychedelic shooter-ish game. I imagine it'd really clearly indicate how little I understand of Eastern beliefs. I guess the idea of the game was going to be that you had to move your soul up the screen, but the screen was a brilliantly colored kaleidoscope path, and you had to navigate a fairly narrow trough of goodness.

Settling Down

I had told myself that I wanted to come up with three ideas that I felt good about before I'd really get started, and I don't think I really did. The rock climbing idea was OK - I could see how it'd be fun to play and fun to code, but I could also see it hit unexpected design problems. All the other ideas weren't really even as good as that.

So, I went with my first idea, and this time that worked for me. It's probably easy to take the wrong lesson from that - to take the first idea that comes along. The last time I made a game for one of these challenges, I think I really benefitted from taking the time to brainstorm different possibilities, because my first idea that time was pretty bad.

What game libraries did you use?

I used PyGame 2.4.2 and PyOpenGL 2.0.2.01. There were versions that were more recent than either of those, but I didn't want to disturb the installation that I had on my

development machine.

I had looked at other libraries, like PGU and PyODE, but I wasn't really familiar enough with them to feel comfortable structuring my game idea around their requirements.

I considered using cgkit to get some basic 3d math functionality (how much fun is it to reimplement dot and cross procucts?), but I didn't get around to it.

The nice thing is that there were very few dependencies for my game, which should make the folks running from source happy.

I did find myself griping about writing text rendering features for the umpteenth time - so maybe it's about time to think about extracting pieces of this game to use as an engine.

How much of the week did you actually spend on the game?

I made an estimate of the total time I spent, and it was around 60 hours. I find that to be a pretty impressive number, considering that I tried to maintain a normal work and sleep schedule during that same week.

There were times when I deliberately distanced myself from the game to get some fresh air and perspective, but as a rule, if I was awake, I was working on the "to do" list.

Having that sort of focus was certainly productive, but by the end of the week, I was very ready to relax. I didn't find that I had been sleeping as well as I wanted, and I wasn't getting as much done at work as I should have. During that week, I didn't watch much TV or get much exercise. Now that the week is up, both of those habits are back.

Did you think about doing music?

I downloaded quite a lot of "Creative Commons"-licensed music that I was considering using in my game. I downloaded so much, in fact, that I didn't have enough time to find things that I liked.

I also told myself that it was probably better to have no music than to have bad music, and whatever music I chose, most players would turn it off pretty quickly and replace it with their own music, if they even wanted music at all.

I considered making an iTunes playlist to go with the game, but I didn't get around to it during the challenge.

How did you do feature X?

Some of these techniques are more impressive than others, I imagine:

Raster art intro

I'm pretty pleased with how this turned out, in spite of it being a clash - the raster scanlines of the intro represent a completely different technology from the vector art of the game. My original plan was to have a vector art intro that would be consistent with the ingame artwork, and I spent some time on that. I managed to convert a sketch of Woody into vector art that pleased me, but most of the rest of my slides ended up looking bad.

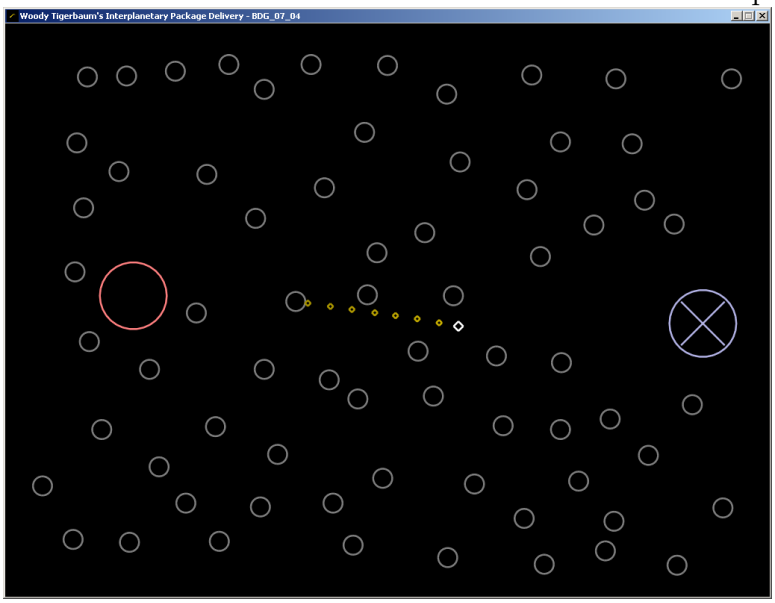
I spent some time thinking about how to improve my vector scanning tool, and I think I might have been able to make something that worked, but it probably would have taken more time to get working to my satisfaction.

So, instead, I decided on the old low-res raster approach. I wrote a script that took an image and converted it to greyscale. I then traced each scanline, converting to black and white and recording where the transitions were. Each scanline had a list of transitions (an even number, as it turns out, in every case) of black to white or vice versa. I created a dictionary where the keys were the vertical positions of each scanline, and the values were the horizontal positions of each of the transitions on that scanline.

Once the dictionary was constructed, I wrote the dictionary out in the same syntax as normal Python dictionaries. This allowed me to include it in the game as just another python source file. I could have loaded the data in from a text file, but I liked the fact that I wasn't keeping track of data or doing loading and parsing during the intro. Also, if you were playing a binary version of the game, there were no data files sitting around, tempting you to mess with them.

At runtime, each frame, I iterated through the table, and considered whether the scanline should be drawn or not, and if it should be drawn, how brightly (based on how recently the fictional raster beam had passed by). If the scanline needed to be drawn, I pulled tran-

sitions off the scanline transition list two at a time and used them as endpoints of a line.



Gravitational simulation

I used a pretty naive approache to this, using a straightforward Euler integration of all the gravitational accelerations from every planet, asteroid, and moon:

newPos = oldPos + oldVel * deltaT

newVel = oldVel + acc * deltaT

I did a cursory bit of research into quick-but-accurate-enough integrators, and I thought that the above formulation was a Symplectic Euler Integrator, but research after the challenge was over indicates that what an actual Symplectic Euler requires using the new velocity to calculate the new position.

The following is copied down from Wikipedia's page on the Euler-Cromer algorithm:

dx/dt = f(t,v)

dv/dt = g(t,x)

v(n+1) = v(n) + f(t(n), x(n)) * deltaT

x(n+1) = x(n) + g(t(n), v(n+1)) * deltaT

So, what I should have been doing is:

newVel = oldVel + acc * deltaT

newPos = oldPos + newVel * deltaT

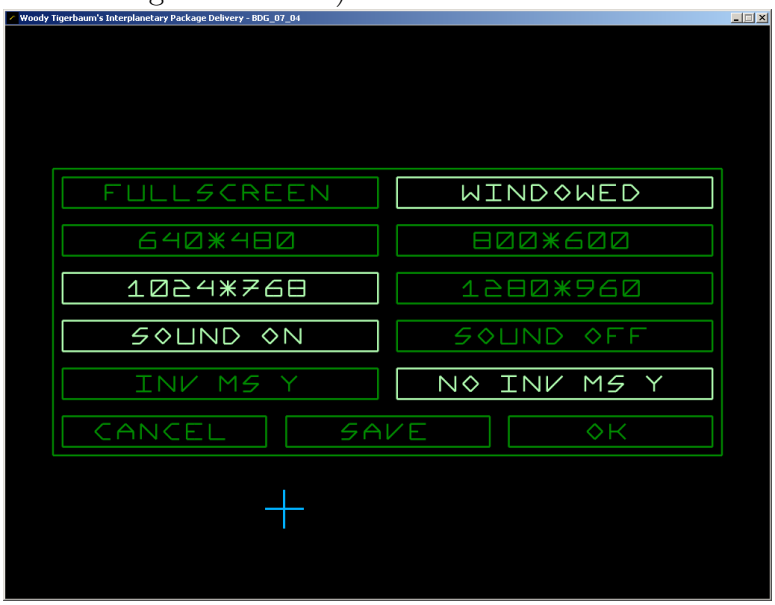
Game states

Each of the following is a discrete "state" that the game can be in:

- each splash screen
- main menu
- options menu
- new game menu
- level intro (or tutorial) dialog
- aim shot
- set power
- determine accuracy
- flight
- level score display
- overall score display
- pause menu

although some states could overlap - you could have a dialog box pop up while something else is displaying, too.

I implemented this with a Hierarchical Finite State Machine approach. There's a single FSMMgr object, which maintains a stack of State objects. The main loop of the program calls the FSMMgr's update() and draw() methods, and the FSMMgr in turn calls the update() and draw() methods of the top State on the stack. Some states know that they only draw part of the screen, so they call their parents' draw() method before they draw their own. (This is how dialog boxes work.)



Options menu

All of my options are stored in globals, most in the options.py file. I didn't really feel comfortable creating a new display if the resolution changed - my gut (or experience?) told me that it was better to shut down PyGame all the way and start over.

So, if the user changed the resolution or fullscreen/windowed setting, I threw an error that got caught all the way up in the Skellington code. When the Skellington code caught that exception, the new flags were arguments to the exception, so Skellington updated the arguments that were passed in to the main() function, and it re-launched the game (with a flag to suppress the opening intro).

Even now, I'm not sure all of this is necessary, and it happens so smoothly, you don't really even notice that anything's going on, which is how it should be.

Computer player

There are a number of different cases that the computer player can be in:

Any humans playing have finished the current level

In this case, the human players are probably anxious to get on with the game, so there's little point in the computer players making bad shots and missing the target.

I let the game play itself Friday night (starting 12 hours from when I was scheduled to submit the game), recording the angle and power of each shot it made. When I got up the next morning, I processed this file, and turned it into code. If you look at book.py, you can see that, given a level number and the name of a planet, there's a sequence of power, angle pairs. These were the values that the unattended session found as shots that landed on the target.

On a moon, orbiting a gas giant

It was important to resist the urge to launch towards the target planet, even if there was an obstacle in between. I added in a heuristic something like this:

- calculate Vp, the vector to the gas giant planet
- calculate Vt, the vector to the target
- calculate a, the angle between Vp and Vt
- if a is less than 90 degrees, there's a good chance you'll hit the gas giant instead of the target. In that case, just wait for a bit.

On a stationary planet, with a gas giant somewhere in the level

Shooting into a gas giant is a bad idea from a stationary planet, too. In this case, I flipped a coin to decide whether to go left or right of the planet, and then I picked a random angle (I think it was between .2 and .7 radians) off the angle to the planet.

The other 80%

The "normal" case would just be to line up the angle with the target, pick a random power (within a small window around 50%), and pick a random accuracy (within a small window around 95%).

I know the computer player can be made substantially smarter. Right now, I think playing against the computer is not very satisfying because it has such a wide difference in perceived behavior between the normal case and the "no more humans" case. I'd like the performance to be more consistent, even in weird cases like gas giants.

Are there features that didn't make it into the game?

Sure, a bunch of them.

Smarter AI

The AI could certainly learn from its mistakes and adjust its aim and power to home in on a good target. Think of the old "artillery" games, where the AI took a blind shot, and adjusted.

More accurate physics

If you're running at a low frame rate, it's possible for the projectile to pass through objects. That's not hard to fix - especially when all my collision geometry is circles. Calculate the line segment between the old position and the new position, figure out the closest point on that line segment to the center of the circle, and if it's inside the radius, there was a collision.

More stable physics

Some machines can't reliably run at 75 frames per second - they have bursts of high capacity intermixed with low capacity. My main loop suffers in conditions like that. It seems like it wouldn't be hard to handle a variable frame rate, while doing a constant-time physics update. I tried that once before, and for some reason, I had trouble.

Save/Play replays

It wouldn't be hard to add this feature - it just would require a little UI to make it work. As it was, the UI was slapped together pretty quickly.

Attract mode

The AI could play itself pretty easily if you left the main menu up for too long. I suspect that this feature wouldn't have even been noticed by most challenge judges, so I left it out.

New objects:

I had some ideas for things that could add more variety to the levels.

- **worm holes** paired objects on the map that act like tunnels - go in on one, come out the other one. It occured to me that getting the collision right on these would be a little touchy. You'd have to make sure that you don't collide with the exit wormhole on the way out.

- **black holes** the idea for these would be that you can't (hardly?) see them until you hit them. Like navigating a maze blind. Is that fun? I wasn't convinced. Maybe used very sparingly.

- **space stations** you could make a space station with four solar panel "wings" look an awful lot like a windmill. That would have been perfect.
- **space dust** regions of the map could be impassable - if the package entered one of these areas, it'd be lost, the same as if it went into deep space or a gas giant.

Slingshot Art

Right now, the gameplay art is abstract and minimal. I considered adding a drawing of a slingshot onto the planet you were launching from.

More event sounds

Perhaps sounds for getting lost in space, landing on a (non-target) planet, and getting lost in a gas giant.

For that matter, I was hoping to get an effect like the Jetson's car as the package flew through space. I did spend some time on this, but it didn't turn out as I had hoped.

Random levels / more levels

I scrambled to get a full 18 "hole" course into the game, and at one point, I was thinking I'd be able to do 36 handmade levels (counting a few tutorial levels that would do double duty). Random levels would extend gameplay (at the cost of weakening the AI further). It wouldn't be hard to create "level packs" of whole new courses to extend gameplay, too.

Mac binary build

I tried to use py2app in the same way I used py2exe to make a binary build. I left this for the last three hours I had to work on the challenge, so it didn't come together.

Online gameplay

This could be a can of worms, but it'd be kind of fun to be able to share your favorite courses with a buddy, and upload high scores (with replays!) to a central server.

What are your plans for future versions?

My current plans are to go over the feedback from the PyWeek judging, and correct the glaring mistakes - small things that can make a big difference in people's enjoyment of the game. I'll post that 1.01 version on the PyWeek site.

After that, I'm thinking about adding in a few of the features that didn't make it into the game, and making that version available for purchase.

This FAQ rocks - how is it made?

Thanks. I'm using The DocUtils/ReST package.

- to generate HTML

  rst2html.py faq.txt faq.html

- to generate PDF

  rst2latex --documentoptions=12pt,letterpaper faq.txt faq.tex

  pdflatex faq.tex