Being A Better Creator

by Michael Heron (Drakkos) and the Discworld MUD creator team

Table of Contents

1. Welcome To The Learning Zone

1.1. Introduction

In this material we're going to look at the ancillary skills that go with being a creator. We won't talk much about the technical skills of coding; instead, we're going to focus on the other skills you'll need to master. These are the "softer" skills, which may not require knowledge of coding but which do need a considerable understanding of how everything fits together and the impact one decision is likely to have on another.

They also require a comprehensive understanding of how to put together games. Make no mistake — although the Discworld MUD is a volunteer-run project, and the game interface is implemented entirely in text, you are still a games developer. Hardly any games have the depth, complexity, and sheer size of Discworld MUD. There may not be any flashy graphics, but you're still going to need a considerably broad set of skills to add to and improve the gaming experience.

1.2. The Plan For This Course

It is expected that you will progress onto this volume once you have completed *LPC For Dummies 1*. You should by now have your own version of Learnville in your `/w/` drive, and the more adventurous among you might even have written your own variations on it.

In this volume, we'll plan out a second village — but not in the fairly simplistic way that we planned out Learnville (which was, after all, purely based on a map). This time, we're going to consider all of the things that our new village will need to have in order to support the wider MUD context. We didn't even begin to consider that for Learnville because it just wasn't important — Learnville was a demonstration of syntax, not a playable area.

Our second village is going to be called Betterville. It will have quests, infrastructure, and all sorts of associated goodness. It's our job in this volume to work out exactly what that goodness is going to consist of, and more importantly how that goodness can contribute to an overall plan for the area.

What we're not going to do, though, is code any of it. None of it. Not a single line of code, not a skeleton of a room, nothing. Nada. Zip. That's for *LPC For Dummies 2*. Breaking the coding from the design is a good strategy, because it means we have to think about what we're doing before we start coding.

First of all, let's talk a little bit about what we're actually trying to achieve with a new area for Discworld. As an opening gambit, we'll consider what defines a good feature of a game, and then look at the implications this has for designing an interesting area for the MUD.

1.3. What's In A Game?

What is it that makes one game more appealing than another game? What makes Game A better than Game B?

Obviously the answer to this will change from person to person — different people enjoy different elements to different degrees. A good game will provide something that appeals to everyone, at all levels of the game. Or, more realistically in a world of constraints, a good game will provide something that appeals to specific genres of gamers. Within Discworld, though, we have many genres of gamers, and we'll talk about that in a later section of this material.

Essentially what all game design boils down to is the provision of interesting game decisions. All games, at their fundamental level, are about providing an individual with a set of choices between different possible outcomes. These choices become interesting, from a psychological perspective, when their outcomes are sufficiently clouded in uncertainty that at the time of choosing there isn't an obviously correct choice.

Imagine the choice between the following options:

- Die
- Stay alive

Providing a player with those options is not providing them with the chance to choose between interesting courses of action. There's an obvious right answer and an obvious wrong answer. On the other hand, let's consider a second pair of decisions:

- Leap the chasm in the hope you'll manage to reach the other side
- Go back down the track and find a new way to go

Now the decision becomes more interesting: the player has to balance the likely difficulty of the jump against the capability of their character, as well as against the likely rewards to come from safely making that jump. On the other hand, if they take the safe option, they risk having to tediously retrace their steps if it turns out there *is* no other way to go. The important thing here is that the player does not know what the outcome of their choice will be at the time of making that choice. It may be obvious in hindsight, but it wasn't obvious at the time.

An element of this uncertainty comes from the black box of the outcome. In traditional board games, this is often incorporated as the roll of the dice, but the same basic feature is present even in entirely deterministic games such as chess. In chess, there is no randomness, but you still don't know until your opponent has made their move (and, usually, a good few moves after that) whether you made the right decision when playing your pieces.

However, randomness alone does not make a decision interesting — it just makes the outcome uncertain. To progress to an interesting decision, it has to be possible for people to use logical deduction to reduce the uncertainty implicit in their options. A player should never be able to entirely remove uncertainty (which is why randomness helps), but they should be able to reduce it to a level that allows them to make an informed, albeit uncertain, choice between competing alternatives. This can be quite subtle; for example, imagine a game of Russian Roulette coded with two schemes:

- When you pull the trigger, a die is rolled to determine whether you died. If you roll a 1 in 6, then you die.
- When you pull the trigger, the chance of your death depends on how many times the trigger has previously been pulled. Your chance of killing yourself thus increases every time the gun doesn't fire a shot.

The first case is equivalent to the barrel being spun after every pull of the trigger — in this scheme, players have to just trust to the odds. Randomness by itself is neither fun nor interesting, and it's certainly not challenging. The only decision here is whether the 1 in 6 chance of death is worth whatever reward is presented.

The second case is the equivalent of the barrel being spun once at the start and then successive pulls of the trigger advancing the chamber onwards. In this scheme, by paying close attention to how other people interact with the game, a player can decrease the size of the window of uncertainty and thus make strategic decisions about when to play and when to walk away from the table.

If randomness reduces how interesting a decision is, one thing that removes interestingness entirely is arbitrariness. Instant death as an unexpected and unknowable consequence of making a decision is an easy way to completely undermine the process of informed decision making. For example, a player standing before a door has two choices:

- Go through
- Stay where you are

If the door leads to certain death, without any hint, suggestion, or possible way of knowing in advance, then this is bad game design. We're going to return to the idea of insta-death later, since it's a sufficiently common transgression that we should spend some time talking about why it's bad from a game development perspective.

Whatever type of game you're developing, your decisions will eventually resolve down to the choices you offer the player and the space of time they have to decide between those choices. In a first-person shooter, the player's choices may be to fire their gun, switch their weapon, leap out of range, or run behind a wall — the fact that they only have a few to make those decisions doesn't mean that there are no decisions being made. On the other hand, decisions in a game of chess can take minutes to evaluate, but it's still the same core process.

What we are trying to do, then, as game developers, is to provide opportunities for people to make interesting decisions. We will have cause to return to this topic throughout this material, as it is the basis of good game design.

1.4. Design Philosophy of Areas

There are several types of area that can be developed in an online environment such as ours. Almost every "real" area is actually a blend of these, but will emphasise two or three types over the rest. The following is not an exhaustive list, but covers the common foci that are present:

- Killing zone

- Wealth zone
- Questing zone
- Immersion zone
- Infrastructure zone
- Exploration zone

Killing zones are self-explanatory — they contain (ideally, interesting) NPCs who exist only to be slaughtered for their high experience point (XP) yield. Killing zones need not cater purely to higher level players; there is scope for killing zones to support almost any level range. However, there is more "mileage" to be gained out of medium and higher level killing zones, since progress through the levels slows dramatically as an individual progresses, meaning that a killing zone is viable for much longer for higher level players.

Wealth zones are almost identical to killing zones, except that rather than being killed for XP, the NPCs are killed for the cash they provide. It's a good idea to keep killing zones and wealth zones as separated as possible, so that a choice has to be made between pursuing wealth or XP. If one zone provides both, then it obsoletes those zones that provide only one or the other — it's not an interesting game decision to kill in an area that rewards both wealth and XP as opposed to an area that rewards just wealth or just XP.

Questing zones exist to provide opportunities for problem solving. Quests on Discworld are far more idiosyncratic than in most online games, and the type of player who enjoys solving puzzles is often greatly attracted to our conception of quest design. Players expect that a new area will introduce new quests into the game, and meeting that expectation is beneficial to everyone. Obviously our system is not perfect, and badly designed and developed quests enhance the experience for no-one. We will spend quite some time discussing quest design.

Immersion zones cater primarily to roleplayers and social gamers. These areas provide interaction but no tangible in-game benefits. An immersion zone may be characterised by a particular kind of ambiance, opportunities for social interaction, the availability of a range of equipment suitable for roleplaying (RP) purposes, or sustainable opportunities for NPC interaction. They may also stitch into larger and more encompassing systems, such as our player housing system or player-run newspapers.

Infrastructure zones exist to provide the tools and equipment that people need to function in the game. They consist of things like the guild areas, smithies, weapon shops, armour shops, and shops or NPCs that sell spell components and other necessities.

Finally, *exploration zones* reward players for paying attention. Areas replete with hidden items, secret areas and rewards for observation fall into this category.

Every good area will have elements of each of these — even designated "killing" areas come with immersive room descriptions and occasionally even quests. Larger areas, like cities, will be defined by the blend of their sub-areas — Genua, for example, has its quest zones (such as the Murder Mystery Hotel), its killing zones (the Casino), its infrastructure zones (the market and the various shops and guilds), its immersion zones (the clothing district and the many bars) and its exploration areas (the sewers and various other secrets revealed only by close examination). Blended together, Genua has a particular feel based on the relative emphasis of its various constituent parts — its relative sparseness of killing zones combined with a heavy emphasis on immersion mark it out as a city for roleplayers and social gamers as opposed to a city for those looking for character advancement.

The first step, then, in planning out your development is to decide on what kind of features you want to emphasise. This will usually be decided on the basis of what's needed in the larger context of your development — if it's a street or district within a city, what features are currently not present that should be? If it's an area within a larger geographic zone, what does it need to provide in contrast to what's already available?

You only rarely have complete autonomy over this decision — there is always a process of give and take between what you'd like to develop and what needs to be developed. Discussions with the leader of the domain in which you'd like to develop are the first step to gaining an appreciation for how the different developments within a domain are linked together.

1.5. Betterville

So, with that in mind, let's decide on a focus for Betterville. This is simplified by the fact that we don't need to fit our development into a larger domain plan, and so we're free to choose which focus best fits our particular needs. Specifically, our main need is for the village to be sufficiently complicated to serve as a driver for the new bits of LPC syntax that we will encounter in *LPC For Dummies 2*. We have no playerbase to satisfy here, after all.

Killing zones don't give a lot of scope for this, although there's certainly much to be said for designing interesting and challenging NPCs. We'll leave that off as a focus, though, and return to it in a later text, because making interesting NPCs involves not so much new syntax as an introduction to the basics of a new concept — artificial intelligence.

Likewise, Betterville isn't going to be a wealth zone — we're not going to learn much by filling it full of NPCs laden with expensive items. We also have no need to provide an infrastructure for anything else, so we won't have that as a focus either. We'll still talk about infrastructure in the course of this material, but we won't add any of it to Betterville.

Primarily, Betterville is going to be a questing zone — we're going to have a number of quests in this area, so as to explore some deeper issues of game development and NPC interaction. We also want to have the area be immersive and responsive to player exploration, so as to provide us an opportunity for incorporating some deeper concepts of LPC such as handlers and inherits. Thus, we pick three focuses for our area: quests, immersion, and exploration.

Having decided on our focuses, we can now start to consider what we're going to include in the area to meet the needs of our development. That's where the real planning begins. It's a lot of fun to let your imagination run wild before coming face to face with the crushing reality of actually having to implement it all!

1.6. Conclusion

We're easing ourselves in gently here. Deciding a focus for our area means we can ensure we include the right kind of things to meet our goal. What that focus should be is usually a function of all the other areas in the domain (and indeed, the MUD as a whole). All of these areas appeal to different people in different ways, and a good game environment has examples of all of them in proportions that fit the makeup of the playerbase.

Good games are engines for generating branching points between interesting choices. Bad games are just engines for generating choices. For a choice to be interesting, each of the outcomes must have an element of uncertainty, and that uncertainty should be manageable (but not eliminated) by player strategising. Bad games introduce randomness rather than challenge, and arbitrariness instead of strategy. Over the course of this document, it's my hope that together we can explore how to do Good rather than Bad in the code we develop.

2. Player Demographics

2.1. Introduction

There are many varied opportunities for gameplay to be found within Discworld, and as such our playerbase has a blend of individuals who seek enjoyment in different spheres of the game. Before we can hope to provide opportunities for interesting decisions, we have to first know what kind of decisions our players find interesting.

While there currently exists no MUD-wide survey of playing styles, the work of Richard Bartle can help clarify the generic subgroups that are usually found within the demographics of an online MUD or MMORPG community.*

2.2. Bartle's Definitions

Bartle defined four key sub-groupings of activities. While all players will have their own blend of preferences for these, many individuals will tend to favour one over the others, and will seek out those activities on a more sustained basis. These sub-groupings are:

- Achievement
- Exploration
- Socialisation
- Imposition

Achievers are interested in setting and reaching in-game goals. These goals may be influenced by the game environment, or may be entirely personal. Achievers are often (though not always) highly motivated by opportunities to accrue reputation within their gaming environment. They like to be known as people who have achieved things.

Explorers are interested in defining the boundaries of the game world, whether these are topologically related (mapping, for example), or concerned with how the game world models its own physics and reality. Gaining an understanding of the environment in which they function is what most defines an explorer.

Socialisers play the game for its social context. The game itself is largely an engine for generating interaction opportunities between themselves and other players. Talker channels, internal bulletin boards and in-game social environments are the stamping grounds of a socialiser.

* Bartle's types are widely used in game design, but are not the only way of categorising players. For example, research by Nick Yee and others has led to alternative categorisations such as the nine Quantic gamer types.

Finally, there are the *imposers*, who play the game for the thrill of direct player to player conflict. These are the player killers who get a thrill from beating another human being in the way that they do not from beating a computer algorithm. Opportunities that increase the conflict between individuals work well for an imposer. More colloquially, imposers are best described by the label “killers” — the game world in general is a vehicle for attaining the necessary skills needed to be able to survive in the world of player killing.

I must stress again that very few players are the caricatures presented here, and everyone combines aspects of these categories to a particular degree. However, the categories do provide a useful framework for planning out the target group for a new development.

2.3. Your Zone’s Players

Understanding the focus of your area makes it easier to understand the kind of players who are going to be most attracted to it:

Type of Zone	Player Type
Killing zone	Achievers, Killers
Wealth zone	Achievers, Killers
Questing zone	Achievers, Explorers
Immersion zone	Socialisers
Exploration zone	Explorers

It may seem like socialisers are ill served by our zone categories, but that’s misleading — socialisers mostly gain their in-game fulfilment from other people rather than from what we provide as creators. We provide the environment in which their own interactions can set the parameters of the game.

Likewise with killers; although they will often frequent killing and wealth generation areas, it’s usually as a means to an end — to gain the necessary power and wealth to be able to pursue and prevail in conflict with other killers.

Knowing what kinds of players are going to be drawn to our area gives us the necessary information to decide what features our area needs in order to best service their expectations. Every decision you make in putting together your design can either enhance the experience for your player groupings, or detract from it.

2.4. Catering For Achievers

Achievers like to have goals that they can set for themselves, as well as opportunities for developing a reputation within the game. Your task then becomes to provide mechanisms to support those goals. Achievers are looking to progress through the levels to become a Person of Power, and also a Person of Repute. Systems that facilitate this goal are highly popular.

Providing a mechanism by which points can be accrued is a useful way of facilitating achievers, especially if those points are available for inspection by anyone else. Discworld uses guild and specialisation "Top Ten Tables" which are available to anyone who may be interested. On these tables, players can compare their standing against the standing of others in their guild or their specific specialisation. We also have a "Top Twenty Table" for achievement points, which fills the same niche. Essentially, anything that adds a permanent or semi-permanent mention of an individual is likely to appeal to an achiever. The "First achieved by" tag on an achievement, for example, is a way of gaining a measure of MUD immortality.

Additionally, since achievers often look to excel within specific sphere of the game, and since our game is still based on invested XP, achievers are drawn to areas which offer significant opportunity for strengthening individual power. NPCs are an obvious example of this — both high XP and high wealth yields will draw achievers to a zone.

More than this, though, a degree of challenge is an important motivating factor. Achievers set goals for themselves, and often feel less satisfied when those goals are easy to achieve; this is partially because easy-to-achieve goals are available to more individuals. If an achiever is to stand out, their actions must separate them from less capable players. There is a substantial difference between providing ten easy NPCs that offer a small XP yield and providing one difficult NPC that has a large XP yield. Killing the basilisk, for example, is a mark of achievement that stands out.

As a corollary to this, though, achievers are often highly focused on the cost/benefit ratio of their actions. While more powerful NPCs may give a welcome degree of challenge, they will often be ignored in favour of the more lucrative opportunities that come from killing many weaker NPCs. In the end, it's all about what gives the most benefit to the player.

Opportunities for advancing seldom-developed skills can draw in achievers looking to maximise their potential, especially if those skills are operationally useful. While individuals may not flock to an area that offers a chance to increase their lute playing skills, having a TM chance for an important skill like health or perception can be a draw. Of course, this has to be balanced with the MUD policy on how often such skills should be available for a taskmaster increase (not very), but the initial point remains.

Overall, achievers are largely concerned with how things improve their characters or their personal reputation, rather than the satisfaction of doing the thing itself. As such, achievers tend to prefer high-yield, straightforward quests to intricate, involved quests. For an achiever, choosing to do a quest is often a decision reached after weighing the potential reward against the possible XP/money per hour of another activity.

Achievers are essentially pitting themselves against the game. Their competition against other players is indirect, and is based on accumulated wealth, or number of quest points, or level in a skill, or teaching bonuses. Thus, in order to appease an achiever, these are the things that your development should provide ways to improve.

2.5. Catering for Explorers

Explorers are the cartographers and researchers, motivated by the chance to learn obscure information about their game environment. They may be methodical and highly analytical, or they may be led by spontaneous curiosity. Their basic features remain the same.

Explorers like to categorise and demystify the black box systems that work in the background of Discworld. They are responsible for teasing out necessary skill bonuses for success with commands, for working out which weapons or armours are best in which situations, and what relationship exists between complementary internal systems. They are also responsible for the creation of the many superbly detailed maps that the rest of us have available. In short, they are the scientists of a gaming environment.

As can be imagined, explorers like to know things that are not easily known. They like to be able to bridge the gap between action and outcome, and detail the larger context of how things fit together. Unfortunately, such things — assuming they are not changed often — are fleeting achievements. Once mapped, an area remains mapped. Once known, a spell bonus remains known. Changing things around reintroduces mystery, but greatly reduces the satisfaction explorers get from making the effort in the first place, and has a knock-on effect with regard to creating opportunities for players to make informed decisions — if the goalposts keep changing, where do you kick the ball?

Although secrets do not remain secret for long, we can still accommodate explorers in our developments. Large areas appeal to the compulsive mappers, but mapping is a One Time Thing. Secret areas and hidden interaction options are more effective — while mapping is a task of methodically exploring an area, finding secrets is a more intellectual pursuit involving delving into the provided clues and untangling them in such a way that a worthwhile result is obtained.

Alternatively, a development that includes a mysterious but ultimately knowable system can provide a great deal of lasting interest. An interesting crime system or gossip system can provide an incentive for explorers to extract as much information as they can from experimentation. However, such systems must come with appropriate rewards — it’s not simply knowing something that inspires an explorer, it’s knowing something worth knowing.

Defining something as being worth knowing is a tricky task. Ideally it will have some tangible benefit in the game — knowing the information makes your own gaming experience more pleasant. However, there's only so often you can provide such a thing, and certainly within a local development the cost of learning a system is usually far greater than the reward that comes from obtaining a little bit of situational knowledge.

One way around this is to provide information that helps explorers to find other pieces of information. Hinting at quests as part of the information teased out of a system can be a reward in itself, especially if knowledge of the quest is otherwise unavailable. Of course, since we have full solutions for all our quests listed on the MUD website, it might seem difficult or impossible to obscure this information, but the design of the quest itself can help there. We’ll talk more on that later.

One mechanism for drawing an explorer to your area is in the provision of tools to help measure other things, which is always surprisingly popular. I wrote a thaumometer for a player once, purely because he asked for one and I didn't see any reason why not. That turned out to be such a useful tool for mapping the impact of background magic on spell-casting that the explorers spent many happy weeks documenting and experimenting. An hour or so of idle development turned into a huge amount of enjoyment for the game's explorers, purely because it hit the sweet spot of what they wanted.

2.6. Catering for Socialisers

Socialisers are a difficult group to explicitly cater for, because so much of their enjoyment derives from interaction with other players. While you can't provide all that much, you can incorporate some elements to enrich interaction opportunities.

One method for this is to introduce multiplayer activities such as games or quests that require cooperation. Porting text versions of popular board or card games can be an excellent way to get people to socialise — Exclusive Possession, poker, and the various other games available to players are all perfect examples of catering for socialisers.

Roleplaying is one particular expression of socialiser tendencies, and providing tools to support that is an important part of keeping your socialisers happy. Providing options for people to tailor their gaming experience in such a way that the changes are cosmetic rather than functional, such as with custom clothing, is a draw for many players. Failing that, creating a shop with new and unique items or roleplaying aids can be a boon to those seeking to increase their sense of personal immersion. For socialisers, it's often more a case of "being" than "doing".

More than this, though, what you're looking to provide is things for people to actually interact about. Examples of this are the player-run newspapers and the player shops — while working within these provides real game world benefits, they are primarily interaction aids. They don't make it possible to do anything that couldn't already be done — instead, they facilitate an interaction in such a way that the infrastructure deals with the busywork, leaving players free to deal with the people part of the system.

There is a danger in this, though, that by simplifying the busywork you will actually remove a lot of the satisfactions socialisers get from interacting. Getting people to work together towards a common goal is one of the motivating factors socialisers have for socialising, whether that goal be an interesting conversation or a successful newspaper. When too much of that interaction is automated, then the challenge is gone and so too is the incentive to socialise.

2.7. Catering for Killers

Killers, like socialisers, play the game to interact with other people. However, that interaction is primarily conflict rather than cooperation. What many killers look for out of a development are the same things that achievers seek — more personal power within the game. However, the motivation is different — killers seek power to remain competitive against other killers.

High XP yield is a significant draw for killers, whether this be in the form of NPCs or low complexity quests. Skill increase opportunities, especially for Killer Viable skills, are also a draw.

However, one thing that can appeal specifically to killers is a framework for interesting player versus player conflict, or ways to widen the pool of individuals who can be part of the fun. An obvious example of this is the Capture the Flag arena, where even those players who have not opted into the world of playerkilling can participate temporarily in playerkilling events.

Many of the player-run guild structures have strong elements of supporting Killer Interaction — certainly in guilds where playerkilling is an important element (such as the thieves and the assassins), conflict between these guilds is a driver for a rich interaction environment.

Within a lone development, though, it becomes harder to specifically cater to killers. As with socialisers, the primary lever by which we can manipulate their game experience is the set of Other People. Sadly, no matter how much we wish otherwise, Other People are outwith our control.

2.8. Conclusion

This has been a fairly brief discussion of the kind of game elements that can support drawing the right kind of people to the right kinds of developments. Bartle and others have published work that is far more detailed than I have outlined here, but some of it is more difficult to relate to our specific gaming environment due to certain assumptions that are made throughout (for example, that killers can prey on non-killers, which isn't the case for us).

Knowing our audience is the first step in knowing what our development should consist of, and we at least now know what direction our planning should be heading. With that knowledge, we can move on to the next step of deciding what specific elements our development is going to have!

3. Urban Planning

3.1. Introduction

Our last two chapters have discussed the motivation for an area, and the kind of people who are likely to be interested in visiting it. In this chapter, we are going to lay out out the specifics of what Betterville is going to be in a way that lets us know, long before we write a single line of code, how everything relates to everything else.

It's important to note at this point that having a plan is not the same thing as having no choice when it comes time to develop — your plan is contingent on real world constraints such as the time you have available, your coding confidence, and the overall requirements of the domain in which your development will live. Plans must be flexible, but you can get an awful lot of benefit out of thinking through exactly what it is you are hoping to achieve.

3.2. An Interesting Area

What makes an interesting area? There are many answers to this, but that one that has always resonated most with me is a definition given by one of our old playtesters, a fellow by the name of Griffin. It was a very useful benchmark and one that I used throughout the planning of Genua. His full posting is available on the Playtesters' wiki ("Feature Density" page), and although the specifics are hugely out of date (both AM and KLK have been remodeled since then) in essence it breaks down into three measures:

- Count the number of rooms in a street or town, discarding inside rooms.
- Count the number of rooms in that street or town that have a special feature.
- Divide the number of features by the total number of rooms to get your feature density.

This is an inherently ambiguous metric, because some of these things are difficult to count properly (to which street does a crossroads belong, for example), and features are hard to define and difficult to quantify — is a room with a hugely complex gaming system as interesting as a shop selling common items, for example? It also ignores things like the quality of the writing, the richness of the immersion, and the overall thematic correctness of the development.

At a basic level, features are the things that you go to an area for — they could be quests, shops, interesting NPCs, or anything else that would serve as a draw for players. While people may come for interesting writing or mapping, they'll only do that once.

When putting together your area, it's an excellent idea to keep the feature density in mind, and aim for something reasonably high. If your development has anything lower than a feature density of 0.5, then you should consider either reducing its size or increasing the number of features. Large, empty areas are of dubious benefit to the MUD in general, except when the sheer size of an area is part of its theme — mazes, for example.

The measure is imperfect, but it has a fairly close match with how “interesting” an area feels — the higher the density, the more interesting an area it is likely to be. The more sustainable the feature density, the longer that area is going to hold its interest factor.

3.3. The Refined Feature Density Model (Experimental)

For the purposes of this document, we are going to look at a refined model for calculating feature density. The refined model is identical in structure to that of Griffin’s, but has some modifications to make it a better guide for creator development, although this comes at a cost of increased subjectivity and increased time to calculate. Such are the trade-offs we make in life.

We are acknowledging a difference of feature values to different groups of players, so it is illogical for us to use a single measure as a mark of interest to all players. Instead, we will use weighted measurements in a table to work out roughly how appealing our area will be to particular demographics.

In the Refined Model, we do not mark a feature as a single point — a hugely entertaining quest isn’t the same thing as a shop selling a random collection of useless items, although in the Traditional Model this would be the case. This is where our increased subjectivity comes into play — we assign each feature a value from 1 to 10 indicating how entertaining we believe the feature to be. We’re not always going to be right, but as budding game developers we should be able to make that kind of rough calculation of interest — it’s part of how we decide what to code. An average feature, such as a shop with unique stock, should be a 5 on the scale. A score of 5 indicates a “take it or leave it” feature. Anything less than a 5 is something that will fail to draw people to an area, while less than a 3 is likely to actively put people off: 5 is “Meh, I’ll take a look,” whereas less is moving towards “Oh, that’s really of no interest to me.”

Additionally, we rate a feature as to how interesting it is likely to be for our particular player groups — killers will not find quests as appealing as achievers, who are likewise going to find shops less interesting than socialisers. A true metric is going to reflect the difference of taste. As a further refinement of the Traditional Model, I would also recommend separating out sustainable features from those that are more temporary — a quest, for example, is a draw for a player until it is completed, and then irrelevant afterwards, whereas a shop holds its feature value over the long term.

Features in the Refined System are broader too — they include not only things to do, but things that happen. They include crime handlers and other immersion possibilities. Essentially, you list everything about your development that you think is likely to be noticed by players, and assign that a value between 1 and 10 for each category of player.

For a worked example of this, I am going to use one of my own streets, since I know I won’t be offended by my own analysis. The street is Liquor Alley in Genua, which is the Discword’s equivalent of Bourbon Street in New Orleans, designed to cater to socialisers and explorers. It has a number of features which will be listed in the table below. First of all, we set the table up with the features and set their interest factor to 0.

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	0	0	0	0
Daiquiri Delights Pub	0	0	0	0
Absinthe House Pub	0	0	0	0
Bayou Bites shop	0	0	0	0
Tattoo You Parlour	0	0	0	0
Talula's Titillating Toys	0	0	0	0
Lady Brook Liquour	0	0	0	0
Salut! (hidden pub)	0	0	0	0
Madame Delight (hidden area)	0	0	0	0

Liquor Alley has a total of eight street rooms, plus an alleyway and a downstairs area, giving us 10 "corridor" rooms. According to the Traditional Model, the 9 pubs and shops listed in the table above would provide a feature density of 0.9, which is very high. However, the refined model accepts that different players will give different values to each of the features present.

Here is where the subjectivity of our Refined Measure comes in — how do we rate each of the features? You are likely to disagree with these, and that's fine — it's the thinking that matters rather than the answers, especially since everyone is going to feel differently about what they like doing. As a baseline measure, I am going to rate every feature as 5 for each player group:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	5	5	5	5
Daiquiri Delights Pub	5	5	5	5
Absinthe House Pub	5	5	5	5
Bayou Bites shop	5	5	5	5
Tattoo You Parlour	5	5	5	5
Talula's Titillating Toys	5	5	5	5
Lady Brook Liquour	5	5	5	5
Salut! (hidden pub)	5	5	5	5
Madame Delight (hidden area)	5	5	5	5

Now, most of these features are pubs — these appeal to socialisers more than killers and achievers. We'll adjust the calculation accordingly:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	5	3	7	3
Daiquiri Delights Pub	5	3	7	3
Absinthe House Pub	5	3	7	3
Bayou Bites shop	5	5	5	5
Tattoo You Parlour	5	5	5	5
Talula's Titillating Toys	5	5	5	5
Lady Brook Liquour	5	3	7	3
Salut! (hidden pub)	5	3	7	3
Madame Delight (hidden area)	5	5	5	5

Both Salut! and Madame Delight are hidden, and thus of more interest to explorers:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	5	3	7	3
Daiquiri Delights Pub	5	3	7	3
Absinthe House Pub	5	3	7	3
Bayou Bites shop	5	5	5	5
Tattoo You Parlour	5	5	5	5
Talula's Titillating Toys	5	5	5	5
Lady Brook Liquour	5	3	7	3
Salut! (hidden pub)	7	3	7	3
Madame Delight (hidden area)	7	5	5	5

The two shops (The Tattoo Parlour and the Erotic Aids shop) are likely to again appeal to socialisers particularly — the Erotic Aids one perhaps a little more than the other:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	5	3	7	3
Daiquiri Delights Pub	5	3	7	3
Absinthe House Pub	5	3	7	3
Bayou Bites shop	5	5	5	5

Tattoo You Parlour	5	5	7	5
Talula's Titillating Toys	5	5	8	5
Lady Brook Liquour	5	3	7	3
Salut! (hidden pub)	7	3	7	3
Madame Delight (hidden area)	7	5	5	5

Each of the pubs has a Pub Toy in it — a game, or object with which the player can interact. This increases their interest to explorers and also to socialisers:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	7	3	8	3
Daiquiri Delights Pub	7	3	8	3
Absinthe House Pub	7	3	8	3
Bayou Bites shop	5	5	5	5
Tattoo You Parlour	5	5	7	5
Talula's Titillating Toys	5	5	8	5
Lady Brook Liquour	7	3	8	3
Salut! (hidden pub)	8	3	8	3
Madame Delight (hidden area)	7	5	5	5

Salut! in particular has a complex story-telling architecture in place, meaning that there is much for an explorer to find out — it thus has an additional degree of attraction for explorers:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	7	3	8	3
Daiquiri Delights Pub	7	3	8	3
Absinthe House Pub	7	3	8	3
Bayou Bites shop	5	5	5	5
Tattoo You Parlour	5	5	7	5
Talula's Titillating Toys	5	5	8	5
Lady Brook Liquour	7	3	8	3
Salut! (hidden pub)	9	3	8	3
Madame Delight (hidden area)	7	5	5	5

Madame Delight is hidden, and also has a small puzzle (although not a quest) to get past the doorman. Solving this puzzle and getting access to Madame Delight has appeal to explorers, and also to achievers who do not like to feel there are bits of the game over which they do not have mastery. Madame Delight herself has a huge range of possible responses for players, giving explorers something more to do and socialisers something to talk about:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	7	3	8	3
Daiquiri Delights Pub	7	3	8	3
Absinthe House Pub	7	3	8	3
Bayou Bites shop	5	5	5	5
Tattoo You Parlour	5	5	7	5
Talula's Titillating Toys	5	5	8	5
Lady Brook Liquour	7	3	8	3
Salut! (hidden pub)	9	3	8	3
Madame Delight (hidden area)	8	7	6	5

Tallying these up and dividing by our corridor rooms gives us the rough feature density by player type:

Feature	Explorers	Achievers	Socialisers	Killers
Bayou Bourbon Pub	7	3	8	3
Daiquiri Delights Pub	7	3	8	3
Absinthe House Pub	7	3	8	3
Bayou Bites shop	5	5	5	5
Tattoo You Parlour	5	5	7	5
Talula's Titillating Toys	5	5	8	5
Lady Brook Liquour	7	3	8	3
Salut! (hidden pub)	9	3	8	3
Madame Delight (hidden area)	8	7	6	5
Total	*60*	*37*	*66*	*35*
Corridor rooms	*10*	*10*	*10*	*10*
Feature density	**6**	**3.7**	**6.6**	**3.5**

This in itself is a simplified worked example, because we're ignoring things like Genua's crime system and rumour mill, both of which are germane to this area — but it's difficult to separate them out from the larger context of the city itself. Additionally, we're not paying attention to the sustainability of these features — a more complex analysis including sustainability is left as an exercise for the reader.

What this analysis tells us, though, is that Liquor Alley has substantial appeal for explorers, slightly better appeal for socialisers, and not much appeal at all for achievers or killers. This fits in with what the design goals were (although this is a "post-mortem"analysis — the only measure used to build Genua was the Traditional Model).

This shows an analysis of an area already in play, but we're designing an area of our own — so we need to perform this exercise from the other side of the fence, as a model to help us plan our development.

3.4. Feature Planning

There are two ways to begin the planning — one is to concentrate on the features first and then work out the area to fit around them, and the other is to plan out the area and then work out the features to fill it. We're going to go with the "features first" plan, because it helps us decide in advance how big our area can realistically be to keep a reasonable density.

To recap on our design goals, we are catering to two sets of players — explorers and socialisers. That means that we have several choices of the "flavour" of zone we're going to develop:

- Questing zone
- Immersion zone
- Exploration zone

Let's plan out a first rough feature-set that is likely to appeal to these particular groups. No details, just a vague idea of what the scope of our development will be.

- One obvious quest
- Two secret quests
- Three hidden areas
- A shop selling unique clothes
- A library that with careful research will yield hints about the quests and some info about other players

Where do we get these ideas from? Just by thinking about what fits into the theme of the area we're developing. We're not actually building Betterville into an in-game zone, which means that we don't have the benefit of thematic constraints and inspirations. We'll talk about the importance of the theme in the next chapter, because it will help us outline what form our village will actually take.

We don't need to know what we're actually going to do here, we're just speccing out what we'll need to do in order to provide a draw to the right kind of people. Even without knowing the specifics, we can start putting together our feature density chart. A quest is more interesting, generally, than any average feature — so it starts off a bit higher. It also appeals across the board:

Feature	Explorers	Achievers	Socialisers	Killers
Obvious quest	7	7	7	7

A secret quest appeals more to explorers and achievers than anyone else, but the amount it appeals to an achiever will be based on how much of a yield the quest will have. Killers are looking to maximise their own skills, and anything that requires intricacy is likely to be less profitable on that score than other available opportunities:

Feature	Explorers	Achievers	Socialisers	Killers
Secret quest 1	9	8	7	5
Secret quest 2	9	8	7	5

Hidden areas appeal to explorers, and do not especially appeal to others. Assuming they have no other draw than the fact that they're hidden, I'd rate them as follows:

Feature	Explorers	Achievers	Socialisers	Killers
Hidden area 1	7	3	3	3
Hidden area 2	7	3	3	3
Hidden area 3	7	3	3	3

The shop will appeal to socialisers over any other group:

Feature	Explorers	Achievers	Socialisers	Killers
Clothes shop	5	3	7	3

And the library will appeal to explorers who want to know about secrets, and socialisers who want to know about people:

Feature	Explorers	Achievers	Socialisers	Killers
Library	8	3	8	3

This is without any real planning — we can adjust the values up and down as we get a firmer idea for what course the development is taking.

Now, the only two groups we're actually interested in are the explorers and the socialisers. Summing up the feature densities gives us the following:

Feature	Explorers	Socialisers
Obvious quest	7	7
Secret quest 1	9	7
Secret quest 2	9	7
Hidden area 1	7	3
Hidden area 2	7	3
Hidden area 3	7	3
Clothes shop	5	7
Library	8	8
Total	59	45

This gives us a measure for what size our area is going to have to be in order to appeal to our chosen demographics — anything more than 9 rooms is going to drop the Feature Density score below 5 for socialisers, and keeping that score above 5 is a generally good rule to which we should adhere. Thus, we're going to give ourselves a budget of 9 corridor rooms to link all of these up.

3.5. Conclusion

The refined feature density model proposed in this chapter is experimental, and at best only a rough measure for guiding your development. It is also not enough to guarantee people will spend time in your area — especially if you are catering to demographics that are not especially well represented amongst the Discworld MUD playerbase. All the refined feature density model allows for is a structured framework for thinking about what your area will need to provide. Use it if you think it'll be helpful, ignore it if you don't.

A good game keeps your players happy, and spending this time at the start of the development before you start writing any code helps ensure that your plans are aligned with what is likely to appeal to the right kinds of players.

Of course, we are also restricted (and supported) by the thematic constraints of the Discworld generally, and in the next chapter we'll look at how thematic awareness can inform our developments.

4. Thematic Considerations

4.1. Introduction

We are both blessed and cursed to be working within a well-defined external theme. We are blessed because Mister Pratchett was tremendously inventive, and his world is both hugely detailed and full of unexplored corners. We are cursed in that Pratchett has defined our universe for us, and if we wish things to happen differently on our Discworld to the one in his books, we need a Damn Fine Reason as to why that should be the case. Gameplay always trumps thematic considerations, but if you (for whatever reason) want a steam powered robot shooting lasers from its eyes, you're not going to be able to do that unless you can develop it in such a way as to make it thematically consistent.

However, we have a huge advantage in the Discworld setting in that there are dozens of excellent books about our game world, each of them rich with possibilities for inclusion in the game. The detail in the books far outstrips our ability to implement it, which is why after over 30 years of development we are still missing massive swathes of the Discworld as defined in the novels. In this chapter, we're going to talk about the thematic considerations that go hand in hand with developing a well-defined game world.

4.2. The Restrictions

We have made several substantial digressions from Discworld Canon in the MUD, and these are almost always for gameplay reasons. We have a system of faith-based magic that is not supported by the books, for example. We have magical spells that permit things explicitly forbidden by Discworld Physics. We have a traditional "north, south, east and west" navigation system rather than a "hubwards, rimwards" system. Decisions to change these things are always based on making the game better to play, and they are not taken lightly.

That's almost never going to be a valid excuse for a development you are doing as a new creator. The Discworld setting is, for you, inviolable. So It Is Written, So Shall It Be. The leader of the domain in which you are developing may have their own guidelines on this, but the general rule is that you don't go against Canon.

The first step then is learning what that Canon is. In general, it is the list of Officially Published Discworld Novels. Things such as the Discworld Map, the Discworld Diaries, and the art of Paul Kidby are all non-canon — if the novels say something different to what is said in the non-canon material, the novel is always definitive. Unfortunately that sometimes means that a newer book will discount the material of an older book — that, alas, is probably quantum.

To know the setting, you need to read the books. Ideally you will read all of the books, because each and every one contributes to the overall context of the Discworld. It's more important though that you read the books related to the domain(s) you wish to develop in, and to read them well. If you are developing an area based on a location in one or more of the books, read those books obsessively and take notes. If details are listed, make sure you have plans to incorporate them. EVERY LAST DETAIL, as Death would say.

An excellent book every creator should own a copy of is the *Discworld Companion*. While not canonical, it is a very detailed reference work to the Discworld universe, including references to the novels in which concepts are discussed. In fact, stop reading now and go buy it. Come back when you're done. It's okay, I'll wait.

...

...

...

Do you have it? Good, let's continue. That book is going to be your Best Friend for development.

Your first step is to plan your constraints — the things that you absolutely have to mention. If you are lucky (and Terry Pratchett being a hugely entertaining author, you are likely to be), what's mentioned is going to be an interesting enough possibility to base a number of your features upon. If layouts are mentioned, then think how those are going to be incorporated. Make notes of names, and people — all of these are going to be present in your development.

The next step is mapping these constraints to your feature plans — this is why we leave the details as vague as possible. If you're planning quests, think what kind of quests fit into the constraints you've been given. Try to plan at least some of your development around the known elements — one of the reasons why people play the MUD is to visit some of the places they've read about, and it's disappointing if nothing exciting happens at these locations.

4.3. The Opportunities

Beyond that, you have free reign (within the constraints of the theme) to incorporate what you like. Just because it was never mentioned in a book doesn't mean it doesn't exist. We are, after all, creating a game rather than a simulation of the books, and a game needs things that a novel doesn't. The books rarely mention things like general shops, pubs and other staples of area development, because they don't fit into the narrative. There's always the possibility for adding shops and people that were never even mentioned. That's when we get a chance to be as original as we like.

Moreover, being as how our work is interpretative, we can infer connections. My first development was the village of Brass Neck in the Ramtops, and that had very few references in the books — the only real thing that was mentioned was that it was the birthplace of a previous Archchancellor of the Unseen University. Cutangle Inn was written to incorporate that link into the development in a way that would have been impossible otherwise. Cutangle was long dead after all, and he wouldn't have a direct presence. This tiny snippet of information allowed for a village that was otherwise not especially "Discworldly" to be tied into the larger context of the theme.

Opportunities to refer to the setting come from incidental references in the books, but general references to particular concepts peculiar to Discworld can bring a great degree of richness and add to the sense that everything is part of one massive world. Many MUDs are comprised of discrete and largely unrelated zones — there may be a zone full of vampires, and another full of robots. Everything in our game is connected to everything else, and making passing references to that is part of where the overall cohesion of our game comes from. Every so often you see the odd room chat reference to THE WAY DEATH TALKS for example, and it reminds you forcefully of the world in which you are adventuring. Tiny little things like that are worth their weight in gold.

When thinking how to fill in your feature list, consider the setting first and foremost — try to make your quests and such relate, as far as is possible, to the distinctive elements of the location in the book. If there is no mention in the books (other than a passing nod, which is quite common), consider how the larger setting of your domain helps contextualise things. Brass Neck is in the Ramtops, for example, and so windy mountain trails and small, tight-knit communities are the norm. This is why the trail to Brass Neck is deadly for those who are not careful; the mountains are a treacherous place. There is nothing in the books specifically stating that Brass Neck is a village wound around a mountain path, but that's what the domain setting suggested.

Discworld is based on a skewed version of Roundworld — the elements of Narrativium mean that stories repeat themselves, and ideas that are strong in our world can leak through to other worlds. When the books fail to lend inspiration, look to our own world for things you could incorporate and adapt to the setting.

These are where your opportunities lie — to fill in the blanks of the books and to exercise your imagination in tying the ideas you come up with into the larger environment of the Discworld. It's the challenge of this that makes creating so fun!

4.4. Data-Mining the Books

There is, no doubt, great joy to be had from owning physical copies of the books. A bookshelf filled with Discworld novels is a beautiful sight.

However, there is also much to be gained from having access to searchable electronic copies. References crop up in the most unexpected places, and an automated search over a collection of ebooks will glean so much information that you would otherwise miss — every incidental reference to even the most obscure locations will be brought to your attention.

All of the Discworld novels are available to buy in digital format, and many local libraries will also have them available to borrow via their websites. Moreover, Google Books allows you to search for specific phrases in specific books, and although it will not always show you the full text of the page on which the search results are found, it does at least tell you where you should be looking in your paper copies.

4.5. Player Input

It's often difficult to think up good ideas for what you'd like to include. It's all very well to say "I'm going to write a quest," but you're going to need a formal plan for that quest before too long. Sometimes the books don't give inspiration and you just can't think what to do. In such cases, the playerbase can be a useful source of ideas.

It's sad but true that many of the ideas submitted by the playerbase are unusable. They are either so exhaustively detailed that no creator wants to code them (half the fun is to plan the plan, as Mrs Lovett has said), so unbalanced that they would break the game entirely, or so vague as to be completely worthless. However, every now and again, there is an idea that is just perfect. The Druid Circle in Ram was coded because I read a player idea that said "Hey, we should have a druid circle," and I thought "Oh, that would be cool." Sometimes that's all it takes — the right idea gets to the right person at the right time, and it flicks the right switches.

In order to avoid good ideas getting lost in the pile of open bugreps, over the past few years we have been making an effort to collect them in our Rainy Day File, which lives on the creator wiki. It's important to note that inclusion in the Rainy Day File does *not* mean that an idea has been given formal approval for implementation; it just means that at least one creator thought it was a good idea worth further consideration.

Playtesters are usually a source of ideas that are more considered than that of normal players, and reading the Playtesters' wiki and the Ptforum board can be a source of inspiration. The player boards too, although a more mixed bag, are worth keeping an eye on to see what people are enjoying or complaining about.

We encourage player feedback for the occasional diamond in the rough — but you have to work to find those ideas. However, even reading bad reports can trigger off that little connection in your mind that leads you down a path of inspiration. Whenever you are lost for ideas, just spend time reading the ideas of other people. Make a note of ideas that appeal to you for your own personal Rainy Day File — you can do this on your userpage on the creator wiki if you like. Not every day is going to be a day in which you come up with great ideas, and other days will be full of more ideas than you can hope to code. Keep track of them as you go along so that you have a fallback list.

4.6. Back to Betterville

So, now that we've spoken a little about thematic awareness, let's return to our plan for Betterville. We have the luxury here of deciding on a theme of our own, because we aren't limited by a domain plan. We could set it in Muntab, or Brindisi, or any number of other places. Let's not be too adventurous though — let's set it in an area in which a lot of the theme is already defined. Certainly for our first development, having constraints will help us produce an area that fits into the rest of the MUD.

We're going to set this village in Genua — a land ravaged by fairytales gone wild. Genua City bears little resemblance to the books, because the time-frame of the MUD is considerably later than the time-frame of Witches Abroad, but a number of design decisions were made in the building of the city regarding the lasting effects of rampant fairy tales. Bois in Genua is a fine example of a village written without guidance from the books, and it is a credit indeed to its coder that it fits so tightly into the thematic context.

We're not going to be as adventurous as all of that — our village is going to be quite small (9 rooms, as we decided in the last chapter), but knowing where the village is set gives us some clues as to where our thoughts should coalesce. Genua is about broken fairy tales, so our quests should reference that in some way. We've decided to create a library in our development, so if we could find a fairy tale that references a library, we can start to build a cohesive theme. In this way, we can incorporate real world elements into our development, but modify them so they fit.

I am a big fan of the Disney adaptation of Beauty and the Beast, and a beautiful library in that is a significant part of the setting. Thus, the pieces start to fall into place — we want a fairy tale development incorporating a library... what if this village were in fact the village where the Discworld version of Beauty and the Beast took place? Lily Weatherwax is said to have experimented a lot with the magic of fairy tales in her journeys around Genua, and so it is entirely consistent that a version of Beauty and the Beast could have been acted out.

The challenge here, then, is to tailor this in such a way that it is not a pure copy of the story, but a parody — much as how Wyrd Sisters is a parody of Macbeth, and Soul Music is a parody of the explosion of Rock 'n' Roll. It'll be the Real World, but mutated.

So how about this... rather than this particular version of Beauty and the Beast having a happy ending, it was a failed experiment in which Beast ate or abandoned Beauty and then locked himself into the secret reaches of his library. That gives us a sufficient theme in which we can work a development — our quests will revolve around Beauty and the Beast, our hidden areas will be secrets in the library, and the library itself will explain to the player how to unlock the secrets contained within. We could even include the Beast as an interesting and powerful NPC to make it appeal a little to the achievers and killers.

How quickly a vague plan can mutate into an interesting area, with only reference to our game world! Simply by knowing where we are located and what we actually want, we can tie together Discworld elements of history with real world inspiration into an area that is entirely original (in that it wasn't mentioned in the books), unique, and with all the features needed to draw our desired demographics.

It is experience with the setting and how the “narrative rules” of Discworld function that helps in making these kind of connections. This is something that will come with practice and familiarity.

4.7. Conclusion

When working within a world with a well-defined theme, you have to know what that theme is. More importantly, you have to know what you can do and what you cannot. Those constraints are not problems you need to work around; they are the very meat and potatoes of making your area seem like something that fits into both the books and the larger game world of the MUD.

The best investment you can make in yourself as a Discworld creator is to read the books. It may be a bit much to ask you to love them (there are actually some Discworld creators who are not especially fans of Discworld), but a love of the Discworld setting will help you make the jump from abstract plans to more concrete, thematically consistent locations. And that’s very exciting!

5. The Best Laid Schemes

5.1. Introduction

Each of our chapters so far has taken us one step closer to defining exactly what we're going to have in our area. At this point, we know the demographics to which we are going to appeal, the features that we are going to provide to ensure that the right kind of players are supported, and a theme for our development that is both original and consistent with the Discworld setting.

In this chapter we're going to finalise these plans and decide on exactly what form our development will take. We'll decide what each of the quests are going to be, and what hidden areas will be present. We'll also sketch out the map of the region according to the constraints we have set ourselves (nine corridor rooms).

5.2. Making the Map

A good first step to help bridge the gap between vague plans and specific plans is to develop a map of how things will link to other things. In *LPC For Dummies 1*, we looked at putting a skeleton area together based on such a map, and that's always a worthwhile thing to do.

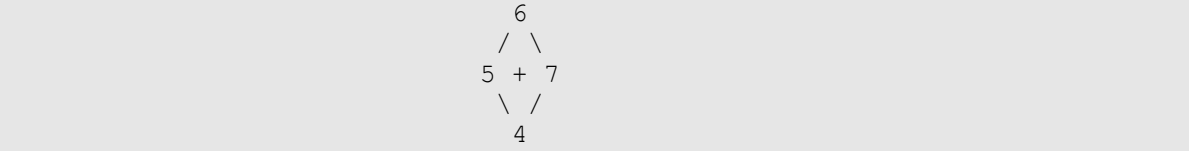
Trying to put together an interesting layout for an area is part and parcel of a development — street rooms are not likely to be "interesting" in the technical sense we have previously defined, but they can very easily be boring.

We have nine rooms, which isn't a lot, but it's enough for a small village square and five further connecting rooms. How we link them up is entirely our choice. We could be boring and put them all along a single line.

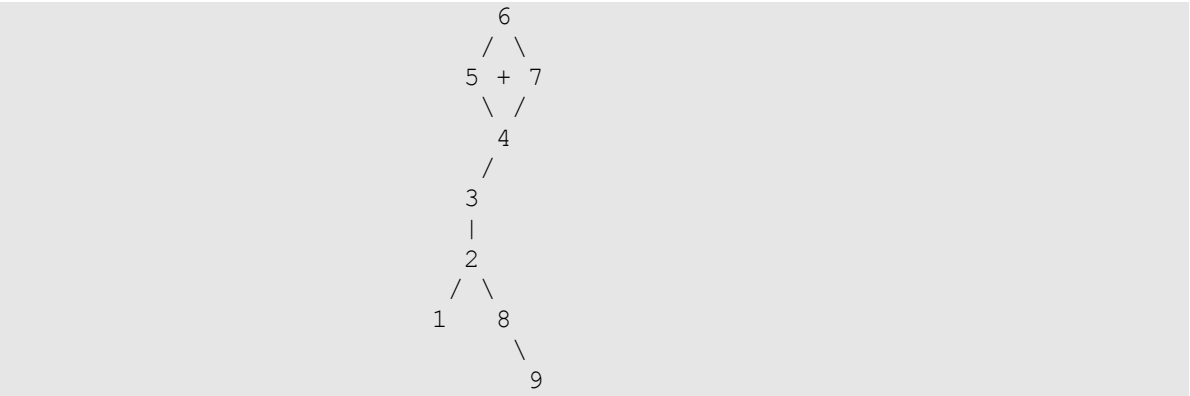
```
                        5 — 6
                        | X |
            1 — 2 — 3 — 4 — 7 — 8 — 9
```

Yawn. Although having to change direction when travelling is not an *interesting* decision, it's a decision nonetheless. All you're doing to get from the end of this village to the other end is travelling in one direction. A drinking bird toy could handle that for you.

We could make the layout a little more inspiring by mixing it up. Almost every village square is a "north, south, east, and west" affair. You could easily rotate that by 45 degrees to make one that is all diagonals:



It's a very small change, but one that instantly changes the topology of the village from the traditional “Ho hum, another village square” to something a little more interesting. Likewise with the path — there’s no reason why it has to go in a straight line. We can make it meander a bit. Also, we could have a branch off the main path leading elsewhere, rather than having a second path leading off the village square. All of these are more interesting choices than having a simple, linear progression.



We have exactly the same number of rooms — all that has changed is that we have laid them out a bit more exotically. An interesting layout will lead to easier descriptions, too. For the first map, it's all going to be variations on the theme of “The path leads east and west”, whereas with our second map we have many more interesting possibilities. So we’ll be using Map Two as the basis for Betterville when it comes time to code it.

5.3. Incorporating Features

The easiest thing to do to begin with is work out what our secret rooms are going to be. We have a number of choices:

- We can make a key thematic feature of our area hidden
- We can make the library itself visible, but rooms within it secret
- We can dot hidden areas around the village
- We can have a blend of these.

We need to decide which of these is likely to be a more interesting scenario.

If we make the library hidden, we're instantly making it more difficult for people to enjoy the area since a key feature is already missing. It'll be of appeal to explorers, but everyone else is going to be disadvantaged. People won't necessarily even know there is a library unless we hint at it elsewhere in the area — perhaps in the chats of villagers, for example.

If we make the library visible, but rooms within it secret, then players will be able to find our key feature. People always pay more attention when they have come to a location that screams "This is not mere scenery!", and so they're likely to investigate an obvious library in a way that they won't investigate a street room. Having a visible library hints that there may be further mysteries within.

We can dot hidden rooms around the village, and have the library open throughout. This will have the effect of making the village itself more interesting (but only to explorers), but the library less interesting. People in general are going to be coming to our village to investigate the mysterious library, because its very presence is a distraction from the rest of the village.

Our final option is to have a combination — we don't need to put all of our features in a single subarea of the development. We could have some secret rooms in the library, and some elsewhere in the village.

Personally, I favour having the library being a trove of hidden secrets, since that's where people are going to expect them to be. Partially we want to provide a challenge and a reward for people who pay close attention, but on the other hand we're writing for an audience, and our audience is reduced if people don't get to explore our creations. As an extra bonus though, we'll make the path leading to the library be a secret, albeit one that is fairly well signposted.

The library itself can be more than one room — when we talk about "corridor rooms", we mean rooms that link features to other features. A feature can be a small zone all of its own, but bear in mind that beyond a certain point you should do a feature density analysis of the subzone itself to ensure you haven't made it boring by virtue of sheer size.

Let's have three rooms for our library — an opening chamber, the bottom floor, and the second floor. Careful exploration of the library will reveal that each of these rooms has a secret to tell.

Where is the library going to be, in our development? Well, we've created a branching path off of the main road, and we're going to make it a secret path. There should be a reward for discovering a hidden path, so let's put the library at the end of it.

The shop is a simple one to place — it'll go in the village, and it doesn't really matter where.

We'll worry about where the quests will go once we have placed the physical features.

5.4. The Library

Thematically, a library in the middle of the wilderness does not make sense. Rural libraries, where they exist, are most often part of a larger structure. They may belong to a castle, or a church, or a monastery, or the private collection of a wizard, and so on. They don't exist alone.

We need to embed our library in a sensible context to make it ring true. Beauty and the Beast, our inspiration fairy tale, takes place in a large and beautiful palace. Again, those don't just exist randomly in the world — they have to have a reason for being there, and a palace doesn't naturally fit with a random village. So we can't really locate this library in a palace.

Remember, though, that we're using the original fairy tale as an inspiration, not as a straitjacket. Spending a little time considering the thematic elements can present a solution — wizards and their towers, while not mentioned often in the books, are certainly canonical (consider *Sourcery*, for example). In the fairy tale, the young prince of the palace is cursed by a witch into becoming the Beast... we could easily cast this as a young wizard being cursed by Lily Weatherwax. It fits in with the canonical representation of wizards, is narratively consistent (wizards being turned into animals is supported by the example of the Librarian, and having a second such example in another library is a bit of a thematic joke), and also ties Lily Weatherwax into the story in an entirely coherent manner.

What would happen after this event? It's likely the wizard would grow increasing bitter at his punishment and neglect his home. It would be overgrown with moss and ivy, and full of crumbling masonry. It may have subsided, causing structural damage. Perhaps all that is accessible now is the library itself, and higher levels of the tower seem impossible to reach.

That seems like a nice idea — that the library is only one part of a proper wizard's tower, but the long, harsh years have rendered it architecturally unsound. It also instantly suggests a series of quests that permit access to the higher levels. Perhaps each level brings with it a quest to get to the next floor, until the absolute pinnacle is reached and the Beast is encountered. Each level above the library, then, is one of our hidden areas.

We'll talk about the design of quests in a later chapter,. For now, we're firming up the plan to the point where we can actually talk about such things from the perspective of providing interesting areas.

5.5. The Shop

We mustn't neglect the shop — we could just slap one down and fill it full of clothes that are already available elsewhere, but that barely counts as a feature. Instead, we want this to be a source of things people would actually want. As with anything else we develop, we have to be sure to keep our thematic constraints in mind.

What kind of clothes are likely to be sold in a small, rural village? Well, real life experience shows that experts can be found in the most unexpected places, so in terms of the realistic ability of the tailors in question, that's not a limiting factor. What we do need to consider though is clientele. Who would come to this village to buy things? Again, this doesn't have to be hugely limiting — people will travel a long way for a genuine genius, but it's best if such geniuses are drawn from the books in some way wherever possible.

A local village store isn't going to sell anything especially interesting — hard-wearing, functional clothes rather than interesting and exotic outfits. However, remember we are in Genua, and people living in and around a fairy tale village are likely to have some rather unusual needs. Think of ways in which you could link the Beast into the larger context of the village — he's going to be a big feature for people around here, so what kind of influence is that likely to have?

Alternatively, think of something the village could be known for aside from its star attraction — is it near any especially noteworthy local features? Is it perhaps known for creating threads or wool or silk of unusual quality? Is it a draw for adventurers seeking the glory of vanquishing the beast? Or romantic young women who've decided that they will be the One True Love who breaks the curse? These are all rich possibilities to explore for ways in which you can provide something interesting for your players while remaining within your theme.

Let's go with the last of these, since it strikes me as rather amusing — a village shop that caters to the young women who have arrived in the village looking for the opportunity to Step Up In The World. Our shop will provide them with all the "fairy tale princess" clothing they need to look the part, as part of a cynical ploy to cash in on their dreams. Indeed, this also opens up another opportunity to enhance the theme — traditionally Beauty and the Beast is about a prince, not a wizard, and these young women may have arrived thinking of royalty, not some arrogant rural mage. As as result, they are likely to be somewhat vocal about their disappointment when wandering around the Spartan village we've provided.

5.6. It All Adds Up

Every decision we make about the theme opens up a new opportunity for making it all fit together. So far we have, for our village, the following defining characteristics:

- A Beauty and the Beast parody.
- A clothes shop catering to impressionable young women.
- A ruined wizard's tower with quests opening up successive levels.
- Grumbling young women wandering around, possibly dressed in exotic and romantic fairy tale gowns.
- A fearsome Beast at the top of the tower, with either rewards or rending for those who reach the highest heights.

All of that has been spawned from coming up with a simple list of features and considering how our theme influences those features. Considering what the implications of our design decisions are likely to be also influences further development.

One thing that you may have noticed from our plan is that our village is tremendously sparse — it only has one shop, and all the other features are concentrated in the library. This isn't necessarily wrong, but it does suggest that if we want to get the best out of our development then the village would be the best place to put any other features that we decide to add. A village pub and a smithy would go a long way towards filling things out. We can worry about that later, though, once the core of our design is in place.

5.7. Conclusion

This chapter is a worked example of turning vague plans into solid plans, and it is by no means the only direction this development could have taken. Interested readers can consider working up their own plans based on the thematic constraints we identified in the last chapter — you may be surprised how different your results can be with only a little imagination.

6. Quest Design

6.1. Introduction

In this chapter we're going to look at some of the design decisions that go into creating a quest for Discworld. Unlike games such as World of Warcraft where quests are mass produced and number in the thousands, Discworld adopts a more hand-crafted approach to the quests we put into the game.

Quests are usually the most challenging thing that a new creator will attempt to write — they are non-trivial to plan, and non-trivial to implement. Nowhere is the need for planning more apparent than in the design of even a fairly simple quest.

Before we get to that though, let's talk a little bit about different kinds of quest design philosophies.

6.2. Why Quests?

Why do we actually have quests on Discworld? Partially it's a legacy from the old text adventures that inspired multiuser dungeons in the first place. Text adventures back then were all about progressing through a story, and in order to progress you had to solve puzzles.† There were rarely any "hack and slash" elements to these adventures; it was all about figuring your way through the game quests.

Within these adventure games, each puzzle that you solved took you a step closer to the conclusion of the story, and if you didn't solve the puzzle then your progress through the game was halted. Puzzles — or quests — thus had a narrative payoff associated with them, in that they advanced the personal story of the player character.

With the exception of a period in which quests had their XP rewards removed (we'll talk more on that later), quests are also about providing alternate models of advancement through the game. Quests give substantial XP rewards, and occasionally also reward money or skill levels — so those that require only brainpower to solve (as opposed to character skills) can help new players to gain guild levels in a way they would otherwise find somewhat difficult. Quests therefore carry with them advancement possibilities.‡

† People are still writing these forms of old-school text adventures today, but the genre — now more commonly known as "interactive fiction" — has expanded massively over the decades. Today, some interactive fiction pieces are heavily puzzle-based, but some have no puzzles at all.

‡ We've moved away from the idea of quests rewarding powerful items on their completion. In previous years, quests gave away unique items like the Wyrm Sword, the Holy Blade of Soyin and stat reward items, but these days they give an item with no direct game advantage (the creator collector cards) and at most some useful but otherwise balanced tangible reward.

Since our quests are puzzles, and since a player needs to be observant to find them and to solve them (assuming they are not making use of the quest listings on our website), they also have an inherent value in terms of providing an additional challenge for players. In the same way that a crossword or a sudoko puzzle is enjoyable in and of itself, a good quest should give enjoyment by challenging in a new and interesting way.

Finally, there is a sense of satisfaction that comes from having completed a large number of quests, especially when at least some of those quests can be construed as a mark of honour. For example, beating Hin-Lop-Heds is not at all easy, so having completed that quest is a real status symbol within the game. Although it is not much used, the Hall of Heroes in Ankh-Morpork makes visible the quest achievements of all players in a public way. The social status of achievers in particular can be enhanced by having a large number of quests to their name.

6.3. Hand-Crafted and Memorable

Discworld MUD's approach to quests is very much a case of "quality over quantity". The quality of quests will vary from place to place, from coder to coder, and indeed in the perceptions of player to player — but the fact is that each of our quests is more or less unique. This puts a tremendous amount of pressure on a creator, because the "off the shelf" quest designs so prevalent in large MMORPGs are usually those that we try to avoid.

If we think about most of the standard quests in such games, they tend to break down into variations on a handful of themes:

- Killing quests
- Drop quests
- Escort quests
- Courier quests

A *killing quest* is one in which the objective is to kill a certain number of a certain type of NPC — for example, "Kill ten green pigs." A *drop quest* is one in which you have to collect a certain number of items — "Bring me four left-handed screwdrivers." Sometimes these get combined into quests that involve you collecting items that have a percentage chance of dropping from certain NPCs — "Bring me six wolf teeth." One feature of many of these hybrid quests is that they are extremely jarring from an immersion perspective. If you are asked to collect ten yeti spleens, and there's only a 30% chance of a spleen actually dropping, you have to wonder how all these other yetis can walk around without one.

An *escort quest* requires you to protect an NPC as it moves from one place to another, usually as it is attacked at several points by multiple attackers.

Finally, a *courier quest* involves taking an item from one person to another, potentially with a middle stage of having to first collect the item — "Go and pick up the ruby nipple from the temple of Zog, and then take it to my friend Stabby Joe over in Learnville."

All of these can be automated in the codebase, meaning that the task of adding a new quest is as simple as deciding on some descriptive text to go with it, a reward, and a target. Such quests can offer a narrative payoff through the descriptive text, are often a source of wealth and XP, and often provide useable items as a reward. What they do not offer, though, is a sense of inherent enjoyment — each one is a means to an end. Killing ten boars because you have a quest is exactly the same thing as killing ten boars when you don't have a quest — the quests themselves just provide a context for rewarding such actions in a consistent way.

While we do have examples of each of these types of quest on Discworld, even in terms of their execution they are unique. A killing quest on Discworld will be slightly different from another killing quest because we offer no common code library for developing them. We very much favour hand-crafted quests from our creators. We end up with far fewer quests, but each of those quests has all of the right qualities.[§]

By making our quests hand-crafted, we have a vast wealth of possibilities in terms of what a quest actually looks like. We are limited only by our imaginations, and that is tremendously exciting from the perspective of a developer. There's more satisfaction in hand-coding an interesting quest than there is in writing up a configuration file to be slotted into a handler along with a thousand virtually identical files. And likewise, there's more satisfaction to be had from the players who enjoy the quest you wrote.

People don't talk about identikit quests in the way they talk about hand-coded quests. I have never had a conversation in which someone has said "Oh yeah, remember that quest where you had to kill twenty plagued ocelots? Yeah, that was awesome." Quests that stick in the mind are the ones that go above and beyond the call to provide something unique and interesting as to their structure. Part of the reward for such quests is simply the process of completing it.

6.4. The Anatomy of a Discworld Quest

So, we've identified that we're going to have three quests in our library — our next step is to decide what those quests are actually going to be. We won't do that in this chapter, but we will first talk about the features of quest design that are particular to Discworld. Knowing what the conventions are makes it easier to guide your thinking. These aren't inviolable rules, but if you want to bend (or break) them you need to get approval from your supervisor.

So, in no particular order, the ten commandments of Discworld Quests! Except, as previously mentioned, they're more guidelines than actual commandments:

§ That's not to say that games employing the "quantity over quality" mechanism have no interesting or satisfying quests — indeed, there are often several of these in even the most ardent hack and slash MMORPG. They are very much the exception, though, and stand out precisely because they are so unusual.

6.4.1. Quests Do Not Reward “Must-Have” Items

The rewards that come from quests should be proportionate to the effort that was invested. Very rarely does that effort reach the level that would warrant a must-have item. Consider it as a balancing effect — if player X can get $500 in an hour from grinding high-risk enemies, then that represents a reasonable metric for how valuable an hour of time in the game should be. If your quest rewards an item that someone would pay $1500 for in the T-Shop, then it should take three hours of high-risk effort to obtain.

In practice, it’s impossible to judge the amount of effort invested into a quest, especially since some players will simply read through the full solution instead of figuring it out for themselves. Most quests can be resolved down into a set of linear steps (but again, more on this later), and the quest that you thought would probably take three hours to puzzle out may be possible to solve in five minutes with the help of a solution.

“Must-have” items throw the balance between risk and reward out of whack, and tend to disproportionately reward those who invest the least amount of effort. As such, as a rule we do not use them as quest rewards. Instead, we reward either flavour items (such as the creator collector cards) or items with only moderate utility (such as Vimes’ boots).

6.4.2. Quests Do Not Reward Stat Increases

In the old days there were some quests that granted permanent stat increases as part of their reward structure. Killing the basilisk granted a point of constitution, and completing the Radiant Adventurer quest gave a point of intellect (along with other horribly disproportionate rewards). Those have since been fixed.

Most of the balance of our skill-stat model comes from the fact that we know how many stat points individuals have to invest. If extra points are then made available, it becomes less important for people to consider where their points will be invested, and also possible to “power-game” the system in ways we have not intended.

An additional stat point is a tremendously valuable artifact, and no amount of invested effort can really justify it as a reward.

6.4.3. Quests Do Not Reward Skill Increases

Skill increases should come as part of action. It’s perfectly okay (and indeed, encouraged) to have quests that involve skill checks, and so the TM opportunities of a quest are a good way to make it possible to give a few bonus points of a skill to a player. Our taskmaster is what regulates such rewards, and it makes sure that levels awarded are a function of how experienced a player already is. The 5 points of `adventuring.health` that you decided to give out in a quest may be worth only 2000 XP to a complete newbie, but tens of millions to a higher level player.

There is a slight relaxation to this rule for “flavour” skills — those that do not have a direct and measurable impact on a player’s ability to advance within the game. Some of the skills in the `crafts`, `adventuring`, and `people` trees rarely have much of an impact on the game, and where appropriate you could possibly give out a few levels of these with no great worry about game balance. However, always check this with your supervisor before working it into your plan for a quest. It’s also worth remembering that a currently “useless” skill might become very useful in the future when new developments are put into game.**

6.4.4. Quests Do Not Restrict Access to Other Parts of the Game

Participation in the game shouldn’t usually be based on whether or not you have completed a particular quest. It's okay to make a few rooms accessible only as part of a quest, but it’s less okay to restrict larger areas — the larger the area, the less okay it is.

The reason for this is pretty simple — we don’t want quests to be something that people *have* to do, but rather something people *want* to do. When it’s mandatory, it becomes a chore rather than anything else. It forces people to play the game in the way we have chosen for them, rather than in the way they have chosen for themselves.

6.4.5. Quests Shouldn’t Require Disproportionately Large Levels of Useless Skills

There's little more frustrating than a quest requiring a hundred levels of `crafts.baskets.weaving` and that skill being of use nowhere except for that particular quest. It’s a tremendously easy way to make sure that the cost of doing the quest dramatically exceeds the reward. If your quest requires a “flavour” skill, then set the absolute limit at something that can be achieved with an investment of a few hundred thousand XP. You don’t want to discourage people from doing your quest purely because the net result will be to actually cost them XP.

On the other hand, you can be much more restrictive with “operational” skills — you can ask more from core trees (`faith`, `magic`, `fighting`, `covert`) because those have utility elsewhere in the game.

In both cases, though, keep in mind the ceiling that a guild can teach, and remember that the cost dramatically increases beyond the guild maximum.

6.4.6. Quests Do Not Have Lethal Consequences

Quests are about fun and experimentation, not lethal consequences. Players shouldn't be killed as a result of trying to work out your puzzle unless it’s adequately signposted. Having a big button that says “If you press this you will die” is an example of something being signposted — people won’t necessarily believe it, but you can’t help some people. On the other hand, having to choose between three identical-looking potions, two of which will kill you instantly, is not a good design choice.

** `adventuring.movement.sailing` is one example — useless before we had a sailing mission, but now a highly popular skill to advance.

This is not to say that your quests cannot involve risk — that risk just has to be proportional, and it has to be opt-in. Players have to know that what they're doing has potentially deadly consequences. Challenging a high level NPC to a fight is something that has an obviously potentially deadly consequence, and players can opt out of that fight when they feel it is going too badly. Dropping someone into a pit of fire with no escape because they pressed the wrong unmarked button on a dashboard is unfair and frustrating for the player.

6.4.7. Quests Should be Easy to Find, For a Given Value of Easy

By this I don't mean that a quest NPC should prod everyone who passes and say "Hey, I've got a quest for you." World of Warcraft signposts quest NPCs so that you don't have to look for quests; you get told exactly where they are. The opposite extreme is quests that are obliquely hinted to in an `add_item` three or four references down. It should be possible for an averagely observant player to get a hint that a quest is somewhere nearby.

This is in your best interests too — quests that are too hard to find will make a large proportion of the playerbase just give up and read the solution. It takes a lot of time and effort to code a quest, and you want people to be able to gain enjoyment from that effort.

6.4.8. Quests Should Use Obvious Syntax, For a Given Value of Obvious

It's really hard to predict what commands people are going to use to solve your quest, and even harder for you to make sure that everyone is going to be happy with the syntax. The most frustrating quests are games of "guess the syntax" in which the only puzzle is what arcane, unintuitive command will unlock the tasty candy of the reward. We've all done it — it's one of the consequences of working within a codebase that lets you do pretty much whatever you would like.

It's difficult to know in advance what is going to be good syntax, but try to make it something obvious. If you want someone to pull a lever, then make the syntax "pull lever", don't make it "tug lever". Use simple, clear syntax whenever you can. If necessary, give the player more than one option; for example, if they can "press button" then it would be generous of you to also allow them to "push button".

6.4.9. Quests Should Not Involve Hard-to-Get or Rare Items

Once upon a time, one of the hardest items to get in the game was a simple oil-can. It was available for a few pence from a shop on the Plaza of Broken Moons, but only one was available per reboot. If you were the first person to the shop after a reboot you could buy it and sell it for a hugely distorted profit, or use it to complete the simple quest to which it was linked.

When the only thing stopping a player from completing a quest is getting hold of some hard-to-obtain item, then the quest becomes a waiting game rather than something they can actively try to solve. This, presumably, is not what your intention was in writing the quest.

An item requirement is especially problematic if your item is available in another area of the MUD, because it's often the case that areas get moved around, temporarily (or permanently) shut down, or remodeled. You may occasionally hear people making reference to a "green gem" — this was an item that was available at one point, but stopped being available because the area in which it was found was removed from game. A few of them lingered around in private hands, usually in the vaults of players who kept them "just in case". This wouldn't be such a big deal if it wasn't for the fact that the gem was required for the Radiant Adventurer quest — the value of these gems was quite staggering, because this was a quest with stupidly overpowered rewards.

Making quest items rare distorts their value, and creates a small economy around their procurement. Obtaining the item becomes a matter for speculators rather than questers, and that is hardly ever a good situation.

6.4.10. Quests Should Have Logical Behaviour and an Obvious Goal

If trying to guess the syntax of something you want to do is frustrating, imagine if you were trying to guess the syntax when you weren't actually sure what it was you were doing. All quests should have an obvious goal that leads the player (with thought) to the solution. A quest in which you have to flick a piece of cheese through an open window needs you to explain why this is something the player should try to engineer. You can provide mention of the cheese in the room, and the open window, but there is no logical reason why the presence of both implies the cheese should be launched airborne.

On the other hand, a quest in which you attempt to ping a ball of cheese to a hungry but timid mouse in a mouse hole has an obvious goal — there's a hungry mouse, and you have some cheese. It won't come out of the mouse hole, so you must provide the cheese to the mouse.

Additionally, the behaviour of your quest should make sense — not only should you know what it is you are supposed to do, you should be able to rationally plan out how to reach the goal.

The canonical example of a quest in which this was not the case is the Babel Fish puzzle from the Infocom text adventure adaptation of *Hitchhiker's Guide to the Galaxy*. The Babel Fish is a fish that translates any language when it is placed in an individual's ear, and the game has a puzzle whereby the player must engineer such a desirable state of affairs. The solution breaks down as follows:

> *Pressing a button dispenses a Babel Fish, but it shoots out at such speed that it flies across the room and into a hole. The player must put a dressing gown on a hook above the hole. This causes the fish to drop down a drain which must be blocked by a towel. When the fish drops onto the towel, it is cleaned away by a small cleaning robot that instantly darts into the room, cleans, and disappears via a small panel. This panel is thus blocked with a satchel so that when the robot darts in, it cannot escape. Instead, it throws the fish into the air to the attention of a second cleaning robot responsible for the upper half of the room. This second robot is dealt with by placing some junk mail on the satchel so that when it is sent flying into the air the robot is sufficiently busy so as to miss the fish.*

There is no way a player can plan this out in advance! To be fair, it does actually break down into several sub-puzzles, each with a fairly well defined goal:

1. Stop the fish flying into the hole.
2. Stop the fish falling down the drain.
3. Stop the robot escaping with the fish.
4. Stop the second robot grabbing the fish when it is thrown in the air.

Solving each of these leads to the next in the chain, but the problem is that the last step of the puzzle violates the principle of having an obvious goal, and also having logical behaviour. This is made especially unforgivable in that you can continue with the game without solving this problem, but it makes the game impossible to complete — and you don't find this out until you are almost at the end of the game.

When planning out the stages that people must go through in your own quest, make sure that they progress naturally from one to the other, and that the required behaviour to bring about a goal from a starting state is something that can be discovered by more than trial and error.

6.5. Conclusion

Quests are going to be the most intricate things you code as a new creator, and also one of the things that are easiest to do wrong. In our next chapter we will continue to talk about the principles of good quest design, because we're not even close to being done with this topic. All we've really done here is discuss the general principles upon which we try to build Discworld quests... we haven't talked about actually designing them yet.

7. Let's Talk About Quests, Baby

7.1. Introduction

Now that we've spent a little time discussing the anatomy of a Discworld quest, let's bring it back around to focus on our own development and the specifics of how our own quests are going to work. As usual, there's thinking that needs to be done before we get to that point — so far we've only really spoken about constraints.

We're planning to have three quests — one secret quest to reward explorers, and two more obvious quests that should be easier to locate for everyone. We've also decided that our quests will take place in our ruined wizard's tower. All of that has emerged naturally from our thematic considerations. Now we need to make actual quests emerge!

7.2. The Intention of a Quest

First of all, let's consider why we're putting these quests in our development at all. We've spoken about this in general terms in the last chapter, but let's look at it in terms of what's specific to our own area. In short, what is it that we want our players to get out of the quest?

Do we want them to get:

- A challenge?
- A tangible reward?
- Access to a feature?
- Nothing beyond the fun of the quest itself?

The design of our quests will vary according to what they are supposed to be generating for the player. Most quests are simply "for the fun of it" — you find them, you do them, you get some XP and then move on. That's absolutely fine, but in certain circumstances there is much to be said for quests that give something beyond that. That's the first thing we need to consider, because it will have implications for the quest we write.

Additionally, when developing a suite of quests we need to consider the connections between them. These connections can be as simple as quests being part of a chain (for example, you can't do the Remote quest until you've done the Emote quest) or a more complex set of dependencies (doing certain quests in Bois will disqualify you from doing other quests). We can set up whatever connections we like between the quests, or no connection at all if we see fit. Our next consideration, then, is whether our quests are going to be linked to each other, or linked to other quests elsewhere. Prerequisites for quests can be more than simply requiring that other quests have been completed — items required, skills needed, necessary alignment — these are all things you should consider in advance. Ideally you'll have some formalised way of representing this, so that other creators can see at a glance how everything fits together.

Linking up quests makes a development seem more integrated, but it also frustrates those players who hit a stumbling block — failing to solve one quest locks them out of attempting the next. This is not necessarily a bad thing, but it's something to bear in mind when planning things out.

Ideally quests that are linked will have a sensible progression. Quest A may involve a door with an obvious lock, but no key, whereas Quest B rewards the player with a key, but no door in which to use it. There is a natural link here that leads from one quest to the other. If there is no obvious implied link, there should be a direct hint provided when solving the prerequisite quest.

Imagine, as an example for this, a pair of quests. Quest B involves finding a secret passage, and Quest A involves fixing the mechanism that opens the secret passage. If the secret passage only becomes available after Quest A is completed, then Quest B should hint at that. If it does not, it is very unlikely the player will thoroughly re-explore an already explored area. The mechanism in Quest A should thus be an obvious triggering mechanism for a secret passage, or fixing it should give a hint that something may have changed in the room. Something as simple as echoing a message to the room would suffice for this.

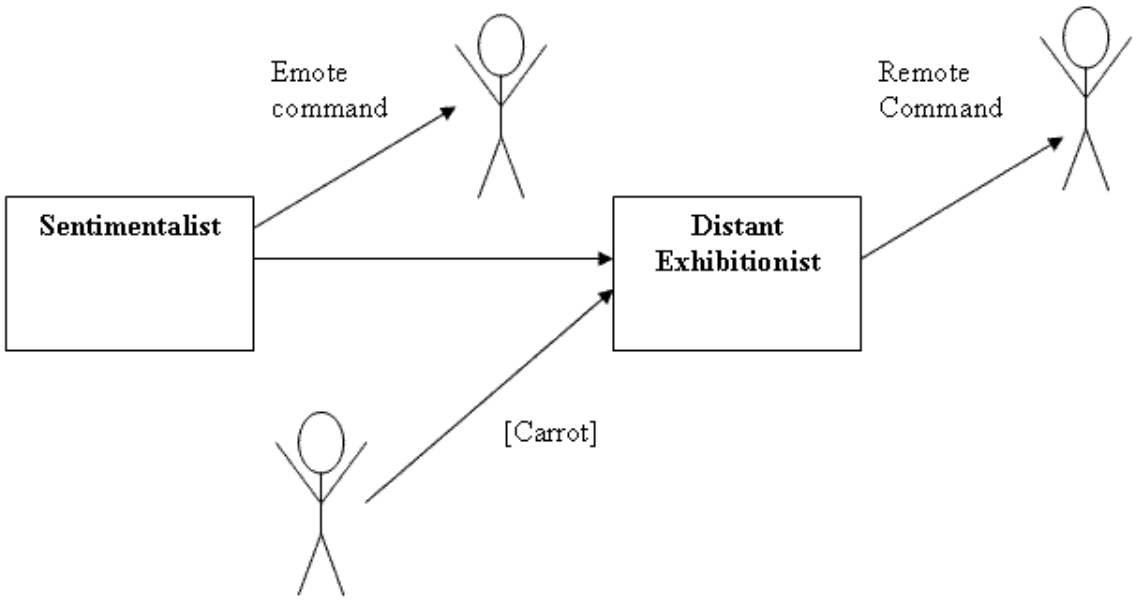
The combination of prerequisites and an understanding of why each quest is in place allows you to decide on what the exact steps of each quest should be. Before we get to that though, let's talk about how to represent the relationship between each of your quests.

7.3. The Quest Relationship Diagram

It can be difficult to communicate to others easily how your quests fit together, and so I am going to outline a diagramming standard that can be used to communicate this to your supervisor and other creators. It is made up of prerequisites (everything a player must have in order to complete the quest) and outcomes (everything a player gets from solving the quest).

The prerequisites are represented as arrows going into a box with the quest name. If a prerequisite is surrounded by parentheses, it means it is consumed in the process of completing the quest. Outcomes are arrows exiting the box… if one of these arrows goes directly into another box, it means that the first quest must be completed before the next can be started. If this line is unlabelled, it means it's a direct requirement (the second quest will simply check to see if the first quest has been completed. If it is labeled, it means that one of the outcomes of the first quest is a prerequisite of the second. If there is a cross through that arrow, it means it is something that restricts the second quest from being completed. The player is represented as a simple stick figure, and can be a source of outputs or inputs.

This is probably easier to show in an example than explain, so let's look at the simple quest relationship diagram for the Emote and Remote quests:



Sentimentalist can be done by itself with no prerequisites, so it has no arrows going into it. As a reward it gives the `emote` command, and that is not a reward that feeds into another quest, so its arrow is shown heading into the player.

However, in order to do the Distant Exhibitionist quest, a player must have first done Sentimentalist. The prerequisite for Distant Exhibitionist isn't actually having the `emote` command (you can test this yourself by giving yourself this command and then attempting to start the quest), it's that you have completed the Sentimentalist quest. As such, Sentimentalist has an unlabelled arrow that goes straight into Distant Exhibitionist to represent this.

Distant Exhibitionist requires, as an additional prerequisite, a carrot from the player. There is no restriction on where this carrot comes from, and so it is up to the player to obtain it (we don't worry about how). Both of these feed into the quest, and Distant Exhibitionist has a reward of the `remote` command.

It's a simple diagramming standard, but one that shows at a glance what is going where and where you are requiring the player to go outside the constraints of our scenarios to find necessary objects. It means your supervisor can come along and say "You're asking the player to find a blue meanie? You have to provide that somewhere else in your quests", as well as providing an at-a-glance reference for liaisons.

Let's use this diagram when we talk about the relationship within our Betterville quests.

7.4. The Betterville Quests

We already know the rough theme for our quests — each one allows you access to a higher level of the ruined tower. Now we need to work out exactly how that's going to work. We also want one of these quests to be harder to find than the others, so as to reward explorers. An easy way to do that is to have one quest as a prerequisite of the others.

Our task is to ascend the tower, which instantly suggests the kind of activities that would be appropriate for the quest — repairing staircases, finding hidden doors, clearing rubble, climbing bookcases — essentially the whole gamut of activities that involve people going from one level of a building to another. We haven't written a single line of description yet, so we are entirely free to decide on whatever we need to be in place so we can write our rooms around these requirements.

If we want one quest to be linked to another one, we also need to bear this in mind — in what ways can a quest be revealed by the completion of another set of tasks? Unfortunately there isn't a quick way to come up with ideas like this; you just need to trust to your imagination. As such, it's necessary for me to simply leap over the thinking part of that to a firm plan for what we'll do. This is the equivalent of me saying "Here's one I made earlier" — the process you need to go through to generate ideas yourself is to sit and think, talk over ideas with people, and imagine what you'd find fun if you were presented with the tower as a player. That takes time, and it's not something that can be sped up. Think back to what you're actually hoping to accomplish with your quests, and consider what can be put in place to support that.

Let's pretend you've already done that, and that we're ready to outline the result of our thinking. Here's what we're going to do for the first quest: upon entering this room for the first time, the player is confronted with a horrible mess with books strewn everywhere. The bookshelves will be in a fairly sad state of repairs, but marked with category headings. The first quest is for the player to organise the library once more by picking up books, reading the title, and putting them into the right category. This is a quest that is obvious — and it doesn't need to fit into any larger master plan.

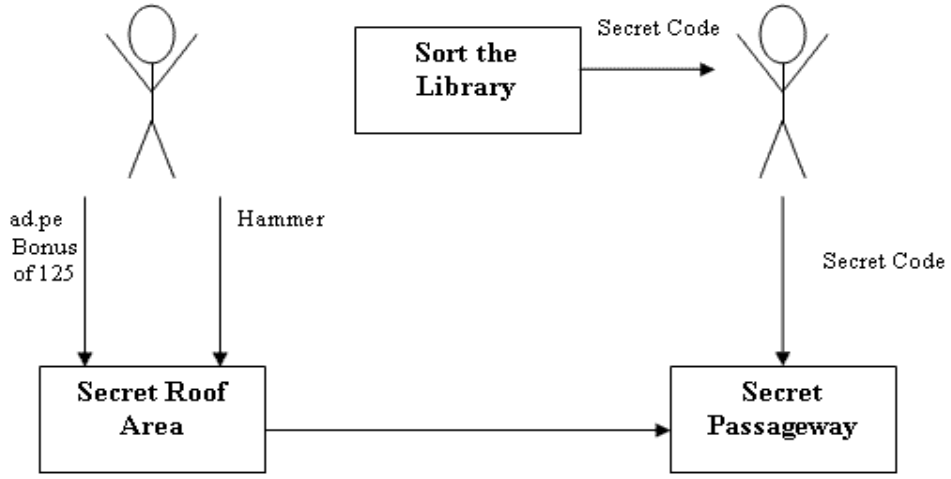
Now the other quests. Since this is a wizard's tower, we have a lot of scope for putting in Freaky Weird Shit, especially considering the nature of libraries in the Discworld setting. Really, our only limit is our imagination. However, we've got hidden areas we want to unlock in this library, and so we have an option of making access to the higher reaches based on the completion of some kind of quest. Perhaps there's a weak area in the ceiling that's visible only to an observant player, and reaching that weak area is based on climbing a particular bookcase. While not, strictly speaking, a secret quest, it's one that players will need to pay particularly close attention to find. We could base it on a certain level of `adventuring.perception`, and require some kind of tool for the player to actually break through the weak area — a hammer, for example.

Upon reaching the next level of the tower, the last quest becomes to clear the rubble that is blocking the staircase to the higher reaches. This, too, is an obvious quest because an obstacle implies a resolution, which implies a quest. An alternative is to have the rubble as a red herring and the actual quest as something slightly different. A nice way to do this is to provide an `add_item` saying something like "Seriously, this rubble is immovable. Don't even try. Honestly." because people will try anyway, and you at least warned them. If we are especially cruel, we can add some commands that allow people to attempt to use various tools on the rubble to try and clear it — but that provides nothing except the cold, hollow laughter of the damned.

Whether this is what you want will depend on your own vindictive nature. I, for example, am full of hate — and so the rubble will be a distraction and the real quest will be to activate the secret panel underneath a painting on the walls. The code to unlock the door will be found in the library itself, provided the library has been put in order. However, because we don't want the player to have to sort the library every time they want to attempt this quest, we will provide the secret code to them as an output of the quest. How that code works doesn't matter right now; we can decide on that later.

The secret panel will lead to a secret passageway that leads to a staircase, and at the top of our staircase is the Beast. We will return to him at a later date.

So that's our three quests, and they're all related. Let's show their relationship on a diagram:



We can see at a glance here how our quests inter-relate in a way that is not obvious in a description which is tremendously useful when assessing exactly what players are going to need to bring with them and where they are going to find the necessary items or information. It's not a binding contract — we can easily change it around when it comes time to do the code if we find things that we have written that are infeasible. What it does, though, is focus our thinking in the right way.

Next, we need to work out the precise steps that are going to be taken to actually solve the quests, and what systems will need to be in place to support them. Before we get to talking about that that, though, we need to discuss the difference between linear quests and dynamic quests. This is an important topic because it directly relates to how likely it is that someone can solve your quest simply by reading a solution.

7.5. Conclusion

In this chapter we've introduced a new diagramming format for showing quest relationships, and discussed the intentions behind quests. We've also refined our plan even further, and identified roughly what our quests are about. Our next step is to outline the steps that must be taken to actually complete the quests. We'll do this without any reference to the code we need — we *are* actually going to code all of this, just not right now. At the moment, our only constraint is on our imagination.

8. Dynamic Quest Design

8.1. Introduction

Although the full solutions to all our quests are available on the MUD website, this is not because we expect players to interact with these quests simply by following the instructions; rather, it's an acknowledgement that it can be very frustrating to get most of the way toward the solution of a puzzle and then get stuck.

Moreover, it is fully possible to develop quests that still require some work and thinking from players even when they know exactly what they need to do — and that's the topic of this chapter.

As a gesture of full disclosure, I will point out that much of this chapter is based on the article "Unlisting The Listable", available on the Discworld MUD website and also published in the articles section of the website *Top MUD Sites*.

8.2. Dynamic and Linear Quests

Linear quests are those that can be solved by issuing a set of commands in the right order, where none of those commands changes from individual to individual, completion to completion. Many of Discworld's quests fall into this category.

Dynamic quests, on the other hand, are those in which some elements of the quest are randomised or otherwise changed based on the player who is attempting the quest. This means that the full solution is unique to a particular player. The exact level of dynamism will vary from quest to quest. Strictly speaking, a quest such as the one in Slice where you count Muckloe's toes is dynamic, even though the only thing that changes is the number of toes. It's very much at the shallow end of dynamism, though.

At the other extreme is a quest like the murder mystery in Genua, where absolutely every part of it is dynamically generated when the quest is started. The only thing that the fully-spelled-out solution to this can provide is the instructions as to how to interact with the quest. Solving the quest is therefore still the work of a good few hours of sleuthing.

Let's talk about the murder mystery, and see an example of the architecture needed for an entirely dynamic quest:

The quest is set in a hotel — a murder has been committed — murder most foul! It's your job to find out who the murderer is. The hotel is a suite of eight or so rooms, each of which has numerous hiding places. The murderer is one of seven NPCs. The murder victim is the eighth NPC. NPCs are randomly set up so that one is the victim and one is the murderer. Instantly, this adds a level of dynamic configuration to the quest.

The murderer is furnished with a murder weapon (randomly), and each NPC is set up with an occupation. Each occupation is associated with a given murder weapon; for example, Captain Beefy, head of the city guard, perhaps trained as a butcher before becoming a soldier, so he'd be familiar with swords (from his soldiering), and meat cleavers (from his butcher days). Again, all of these occupations are set up randomly.

This immediately narrows down the suspects to those who are familiar with the murder weapon — examining the corpse reveals the weapon used, as well as a number of other clues (generated randomly). These clues are scattered about the hotel rooms, in various hiding places. Each NPC has a randomly generated whereabouts list, and the murderer will be the only one who was in all the rooms containing clues after the murder was committed. As an extra constraint, the murderer will have been seen by another NPC every hour after the murder.

The quest works by questioning NPCs — asking them where they were, who they saw, and who they know. Through thorough questioning, you can build up a profile of the movements of each NPC. From this and the list of murder weapons, you can work out which NPC was in which rooms after the murder, and then you accuse them — if you were right, you get the reward. If you were wrong, you get nothing.

This quest took weeks to code and test — but it stands as one that remains a challenge to complete no matter how many times a player has already done it on other characters, since there's so much randomness in how the initial setups are generated. There are billions of possible permutations — the only bits that a player can know in advance are the instructions, and those are the easy part. The hard part is solving the puzzle.

8.3. Which are Best?

There's not a lot of dissent on this score — quests that give substantial rewards should require a substantial investment of effort. There is generally no great effort involved in following the instructions from the full quest solution, and so linear quests disproportionately reward players who decide to skip the stage of figuring things out for themselves. It may take a normally inquisitive player an hour to work out how to solve a quest, while someone who just reads the solution can type in the instructions, get the reward, and then move on to the next. This completely inverts the principle of invested effort influencing reward.

Every level of dynamics that you put into a quest adds a level of work that an individual has to do in order to complete the quest. As such, whenever possible, you should be thinking of ways in which your quests can be randomly set up.

8.4. Some Guiding Principles For Dynamic Quest Design

The process of making a dynamic quest is the process of creating an implied contract with your players — that their effort is going to be fairly rewarded. As such, there are some design principles that go along with putting such quests in the game.

8.4.1. Principle One: Be Transparent

Make all of the information needed to find and solve your quest freely available within the game. Making a quest so obscure that players are almost forced to go "out of character" to read the solution on the website not only reduces immersion, but is also unfair on players who want to figure things out for themselves. This may seem to contradict our discussions about the things that motivate explorers, but everything in game design is a trade-off, especially when the thing in question has broader appeal — everyone does quests, so in general they can't cater purely for explorers.

Be up-front about the syntax — make it obvious, and make it clear. Stick it in a help file, if necessary. Let everyone know how to find it.

8.4.2. Principle Two: Identify Areas Where There is Room for Randomness

Consider the standard "courier" quest: You are a brave hero, yadda yadda yadda, great risk, peril, danger, take my item X, give it to NPC Y.

The bits around the edges may be different from quest to quest, but the structure is the same. There is great scope for randomness — generate item X from a list of possibilities, and likewise generate item Y from a list of possibilities. The same quest engine can generate dozens of different quests: "Take this rock to Maggie May", "Take this pencil to the Consumer of Souls", "Take this sword to Mrs. Miggins".

That's a simple example, but one that will work for a range of different quests provided they have a similar theme that can be expanded to include different elements.

For example, one of the engines we have on Discworld is not a quest, merely a series of odd jobs that people can do to earn money. Odd jobs are generated of the form <place> <broken item>. Inherits allow for the command syntax to be available in hundreds of rooms, so the same engine can generate thousands of combinations: "Fix the broken window in Annie's shop", "Fix the door handle in Old Wilson's mill", and so on.

Most quests have a solid object and a solid target — if you can randomly generate these from a list of possibilities, then you greatly increase the range.

As another example, we have a quest that involves contacting a spy, who then sends you on a mission to intercept a coded message. Obviously this message could be chosen from a list, but we did something slightly different — the message is sent as cryptographic symbols, which are generated randomly.

The player looks up each symbol in a book, and then gets a message (chosen from a random list) that is encoded according to a simple mono-alphabetic substitution cipher (the cipher is generated randomly). The quest is then to decipher the message and tell it to the original spy. The principle of deciphering such a message is fairly standard, and a description is available on many websites — but it's still a complex thought exercise to put that theory into practise.

Again, this is a quest in which knowing the full solution only gives you the easy part.

8.4.3. Principle Three: Provide In-Character Means of Finding Quest Hints

Our quest listings on the MUD website offer players the option of getting hints instead of the full solution, but this doesn't have to be the only way of providing a helping hand. An in-character quest hint system can be a lot more fun, and is certainly more immersive. Does your area have bars and taverns? That's an excellent place to drop the occasional hint. "I hear there have been some murders up at the old hotel", or "I saw a shady looking guy hanging around in the bar the other night". You can liberally include useless trivia and funny observations in this, too; it doesn't have to be All Business. Make it funny, make it cute, and people will like it regardless.

Simply giving people an indication of where quests can be found removes a lot of the frustration that might drive players to "look up the answer" on the website. If we make quests easy to find, and provide hints as to where people should be looking, then those who want to find and solve the quests themselves are able to do so without being entirely disadvantaged.

8.4.4. Principle Four: Make it Interesting

Obvious, I know... but hear me out.

Remember why people do quests in the first place — sometimes it's for the reward, but often, it's just for the thrill of solving a puzzle. Unlistable quests are a quantum leap more difficult to code, but they offer the potential to be so much more interesting. Their very difficulty inspires us to be more creative.

The more interesting you make the quest, in terms of the tasks involved and the storyline, the more people will be willing to do it.

8.4.5. Principle Five: Amply Reward Effort

If it takes two hours to solve a quest, then the quest should give a substantial reward that reflects the time taken. It doesn't have to be equal to what would have been gained farming NPCs for XP, but it should be enough to justify the expenditure of time. If people feel that they are getting enough of a reward for their hard-earned quest, there is less of an incentive to just bash their way through like a robot executing the instructions.

The investment in coding effort means that you want to get maximum utility out of your quest — a fair reward is also a good draw of interest. If you make it worthwhile, then your development time will not have been in vain.

8.5. Dynamic Quests — The Book Sorting Quest

So, how does all of this help us design our Betterville quests? Well, for one thing it gives us another design principle to bear in mind — make it all as random as possible.

Let's go over each of our quests in turn and look at how we could make them dynamic. The first is the sorting exercise in the library.

The traditional, linear version of this would be to have a set of books that fit into a set of bookshelves. The first book is "Stuff for people", which fits into the bookcase "Things about stuff"... if it's always that, then it can be solved entirely by following the quest solution.

A better solution, then, is to generate books either randomly, or from a large list. This is also an opportunity to add in a few jokes along the way, in the vein of the old standard of "Book Title" by Punnily Named Author. These two examples are taken from a development in Genua:

- "How I Crossed the Klatchian Desert", by Rhoda Camel
- "How I Crossed The Sto Plains", by Randy Hoelway

It's little touches like this that make our game so special, and worth exploring. If we have a long list of books, and a set of randomly selected book cases, then we're a long way towards producing real dynamism in our quest design.

To imagine this in practice, let's say we have four bookshelves — "Biographies", "Fiction", "Magic" and "Erotica". Those four bookshelves then set up the possible set of books to be found in the library. The player picks up a book, and reads the cover. The book is randomly chosen from the list of valid options:

- "How I Like It: The Tale of a Morporkian Seamstress", by Harden Fast

The topic is erotica, and so the book is slotted into the erotica bookshelf. Repeat this a certain number of times, and the library is organised.

The bigger the list of books, the less chance there is that any player is going to make a big list of them all and share it with their friends. You don't even have to have every one of them being a joke; you can have the book titles completely randomly generated if you so choose. Indeed, we're going to go with a variation of this when we move on to actually writing the quest.

So, let's formalise what this quest will involve:

1. Every reset, the library subsides and the books spill out of their bookshelves.
2. At this point, the bookshelves in the room are randomised.
3. The random bookshelves are used to generate random book titles.
4. When a player picks up a book, they have to put the book in the appropriate shelf.
5. When they've done this ten times, the quest is completed.
6. Upon sorting the last book, a small slip of paper falls out of it, containing a randomly generated secret code. This code is the code to the secret door in the chamber above.

While we don't need to come up with all the quest details now, we should probably think of a quest title and quest story in advance. For this one, how about something like:

Visiting Librarian, in which you obeyed the categorical imperative.

Your quest story should ideally be a little bit funny. It's not, here, but then I am an Old Man with no measurable sense of humour.

8.6. Dynamic Quests — The Secret Hole Quest

It's a little harder to figure out the dynamics of the hole quest because it seems like something you either do or do not. However, there are Ways and Means.

First of all, we could link the location of the hole to random bookshelves, so you need to look in the right place before the hole is seen. Rather than "look ceiling", you would "look ceiling above erotica bookshelf". In this way at least you have to do some work yourself.

The second option is to base the tool required to break through on some kind of random algorithm — sometimes it'll look as if a hammer is needed, sometimes a crowbar, maybe other times it'll need a knife of some kind. This should remain constant as soon as a player has discovered the hole, because it's unfair to make players go to fetch a crowbar and discover when they get back that the room's reset while they were away and now they need a shovel.

So, our formal steps for the quest are as follows:

1. Randomly generate a location for the hole
2. If the player has a certain `adventuring.perception` bonus, show the hole when they look in the right place.
3. When a player looks at the hole, give them a clue as to what they need to break through it.
4. The hole is reached by the player climbing the appropriate bookshelf.
5. The player uses the hints given as to the tool needed to break through the hole.
6. Upon doing so, the quest is granted and the player is moved into the secret room above.

For a quest story, how about this?

Roof Inspector, in which you discovered the hole truth.

I know, it's terrible. I'll get my coat...

8.7. Dynamic Quests — The Secret Room Quest

The randomness in the secret room quest is already in place — if the secret code generated from Visiting Librarian is random, then every player will need to complete that quest in order to find the code. We can add extra randomness by having more paintings in the room, and changing which painting the panel is behind. You could even work this as a simple puzzle in the vein of the zebra problem[††] if you were feeling adventurous. We'll be slightly less adventurous and have the paintings be of people with particular facial features and algorithmically generated names. We can generate them entirely randomly so as to end up with, for example:

1. The huge portrait of a man with a big nose and blue eyes.
2. The small portrait of a woman with small lips and red teeth
3. The medium portrait of a bear with big teeth and green eyes

Ideally this would link back into the library, so that a player making use of the library would be able to find the clues necessary to identify exactly which painting needs to be moved to reveal the panel.

Our formal steps then:

1. Randomly generate a set of paintings.
2. Allow the player to research the secret panel in the library to find clues about the paintings.
3. When the player moves aside the right painting, they find a panel.
4. Entering the secret code into the panel opens the secret door to the hidden staircase.

And our quest story:

Art Investigator, in which you discovered what was beneath the surface.

Look, stop staring at me. I'm doing the best I can.

8.8. Conclusion

After three chapters, we now have an overview of what our quests are going to involve. You may be thinking "I don't know how I'd even begin coding any of this," but that's okay. You will when we get to actually starting to write up our new and improved village.

We've spent a lot of time talking about quests specifically because they're one of the easiest things to get wrong, and it can be hard to relate the often vague guidance you are given on how to code a quest to our own specific requirements.

†† The zebra problem is a type of logic puzzle where a set of constraints is stated and the solver has to work out the answer to one or more questions within that set of constraints. For example: The paintings are red, yellow and blue and depict a witch, an assassin and a thief. The red painting is on the left. The yellow painting does not show a witch. The painting of an assassin is in the centre. The blue painting is next to the painting of the witch. What colour is the painting of the thief?

Sadly, most of what quest design involves is just sitting and thinking, and there's no shortcut for that. However, there are plenty of places to look for inspiration — your fellow creators, the playtesting team, the books, and the Rainy Day File where we store good idea reports.

9. The Ten Commandments of Room Descriptions

9.1. Introduction

We shouldn't get bogged down in the complexities of planning features — it's part of being a creator, but so is the work of actually providing interesting, clear and entertaining descriptions for everything that we write. Indeed, certainly in our first few developments this is usually what we spend the majority of our time doing. It may seem like something that is self-evident with regards to how it should be done, but over the years we have built up numerous conventions as to how descriptions should be written.

In this chapter we're going to talk about what needs to go into a good description. It should, however, be noted that everyone has their own views on what Good Writing is supposed to look like. If you are developing for a specific domain, then the leader of the domain will be able to give you guidance on what is considered acceptable within that domain. On top of this, though, there are some guidelines that are Universal. These are the ones we're going to discuss.

9.2. The Ten Commandments

The Ten Commandments of Room Descriptions was a simple set of guidelines I wrote for the things you should and should not do when writing descriptions. They are incorporated here from the original article.

9.2.1. Commandment I: Thou Shalt Not Use "You" In Room Descriptions

All direct references to the player should be removed from the description. This is especially true of descriptions that begin with "You are standing in..." or similar. Room descriptions should make sense from all perspectives and all situations... what if I'm not standing there? What if I'm sitting, lying down, or hopping on one leg? What if I'm looking into the room from an adjacent room, scrying into the room from miles away, or sitting comfortably in an armchair at home looking at an iconograph of the room that a friend just gave me? In all these cases, the room description will no longer be accurate:

```
> look down
You are falling at great speed towards jagged rocks below.

> scream in fear
You scream in fear.

> shout Help! Help! I'm falling to my death and need someone to tell my wife I
love her!
```

```
<several uneventful seconds pass>

> think
You think carefully.

> look
You are standing at the top of a cliff. It's perfectly safe, provided you don't
take a wrong step.

> say Oh yeah!
You exclaim: Oh yeah!
```

It is jarring when this happens, and you should consider this an Inviolable Rule as to your own descriptions. Never make any assumptions on how the player is viewing the description — keep it general.

9.2.2. Commandment II: Thou Shalt Not Assume A Course Of Action From The Viewer

A room description's purpose in life is to describe — hence the word, "description". What a room description is not there for is to dictate the action of the viewer, even for the holy purposes of narrative causality.

- "You decide to try the door handle, but the door is locked."
- "You decide to follow the path to the north."

Let the player choose what they want to do, based on your descriptions... you're not here to play the game for them. Likewise, don't tell people how they think, or how they feel — they know better than you do. This is something that you see particularly in games where the courses of action through the game are more restrictive than ours — the MUD is not a Choose Your Own Adventure book.

```
This is a dark, scary trail through the forest.  To the north is a horrible
looking path, and you decide to follow it to...

> don't follow path
What?

... to follow it into the depths of the forest where the horrible spiders roam
with...

> stop following path
What?

...roam with their sharp, pointy teeth and horrible eyes glaring...

> say I don't want to follow the path, I'm scared of spiders!
You exclaim: I don't want to follow the trail, I'm scared of spiders!

...horrible eyes glaring, and yet you feel unfraid, as if the spiders hold no
fear for you.

> whimper in fear
You whimper in fear.
```

Having the game assume a course of action from you is tremendously annoying if you are personally invested in your character. It can seriously impact a player's sense of immersion.

9.2.3. Commandment III: Thou Shalt Not Write Static Descriptions Of Dynamic Objects

By dynamic objects, I mean those objects in a description that are likely to change their state or appearance during the course of the game.

"The door to the north is closed", for example, when the door may be opened by a player or NPC.

Or...

"The chairs are empty", when someone could sit in them.

```
This is a lovely tavern, with an empty stool at the bar.

Drakkos Wyrmstalker arrives from the south.
Drakkos Wyrmstalker sits on the stool.

> look This is a lovely tavern, with an empty stool at the bar.
Drakkos Wyrmstalker is sitting on the stool.

Drakkos Wyrmstalker says: Hey sailor. Buy me a drink?
```

Static descriptions should refer to static objects. If you must make a reference to an item that is likely to be dynamic (like a crowd of people), then make it vague rather than specific:

```
> look crowd
The crowd mills and jostles around you, caressing your body with the Brownian
motion of its constituent members.
```

Likewise, when describing an object that is mobile, it is better to do this as a room chat. "A bird is flying in the sky" in a long description makes the whole room look static, like an oil painting. If the bird flies across the sky occasionally in a room chat, this is less true.

9.2.4. Commandment IV: Thou Shalt Not Use Relative Directions

Unless you're going to get very clever with code, you cannot assume a direction of entry from a player. When a player enters a room with two entrances, in general you don't know which one they entered by. Likewise, in a room with only one entrance, you don't know if they arrived via that entrance, or if they portalled in, logged in there, or were dropped off there by mutant bats. For this reason, you shouldn't assume that a relative direction is going to be appropriate.

"The forest stretches ahead of you to the north."

What if I just arrived from the north? Wouldn't it be stretching behind me? Likewise with "left" and "right"... these will change depending on what direction I've just arrived from.

```
This is the kitchen of a pretty house.  The larder is to your left.  The sitting
room is to your right, the pantry to the south and the hallway is behind you to
the north.

> south
It's the pantry.

> north
This is the kitchen of a pretty house.  The larder is to your left.  The sitting
room is to your right, the pantry to the south and the hallway is behind you to
the north.

> ponder
You ponder.

> north
It's the hallway.

> south
This is the kitchen of a pretty house.  The larder is to your left.  The sitting
room is to your right, the pantry to the south and the hallway is behind you to
the north.

> shout Help! Help! I'm trapped in the house that Escher built!
```

Avoid words that assume you arrived from a particular location. It won't bother all players, but the ones who are bothered will be *very* bothered.

9.2.5. Commandment V: Thou Shalt Write (Mostly) Proper English

Although writing a description is not the same as writing a thesis, there are certain stylistic elements of formal writing that you should keep in mind when writing. One of these is that you should not write numbers as Arabic digits... instead, write them out fully. "There are 2 large stone blocks here" should instead be "There are two large stone blocks here".

This rule extends at least up to ten, and usually up to twenty. Beyond that, using the digits is acceptable, although using a general plural such as "lots" or "many" is perhaps a better approach. After all, who is going to count the exact number upon a casual glance at something?

Also with regars to writing properly: proper sentences have verbs, and so should all the sentences in your description. "A large clearing." is not a sentence. "This is a large clearing." is a sentence.

However, don't worry too much about being too formal... in many cases, writing completely formal text will detract from a quality description. Use formal writing where appropriate, and whatever sounds good for the rest.

Finally, the standard of the MUD is for the British spelling of words. Terry Pratchett was an English author, and since the MUD is based on his works, we use his spelling conventions. So "color", "center", and "emphasize" are all wrong — "colour", "centre" and "emphasise" are correct. All your descriptions should conform to this standard. If in doubt, grab hold of a British English spellchecker and run it over your descriptions.

9.2.6. Commandment VI: Thou Shalt Make Thy Text Easy On The Eye

A MUD such as Discworld is, I'm sure you've noticed, a text-based medium. As such, the presentation of the text in a clean and attractive manner is of paramount importance. Although the practice is falling out of vogue as variable width computer fonts become the norm, it is one of the standards of the MUD to double-space between sentences. The practice of double-spacing stems from the days of typewriters when each letter took up the same amount of space on a page. Many MUD and telnet clients still use fixed-width fonts to present MUD output, and using only a single space between sentences makes the whole description cramped and difficult to read..

Related to this, you should avoid the use of `colour` in descriptions and shorts. Colour is a powerful method for emphasising particular words or sentences, but using it carelessly makes everything garish and detracts from the rest of the text. Additionally, it detracts from the consistency of the MUD in general. Why does your "red sweater" have a coloured short description, when someone else's "green sweater" doesn't?

Remember too that a number of users cannot see colours on their clients, and even those that can may find your choice of colour clashes with their client's background or their own defined colour schemes.

Finally, using colour also means people have to worry about stripping colour codes from your short descriptions when they reference them in code... an additional CPU and design overhead for something that is usually undesirable in the first place.

9.2.7. Commandment VII: Thou Shalt Describe Every Noun

If you mention a noun in your room, you should also make sure you have an `add_item` for it. This goes equally for nouns within the descriptions of `add_item`s. There are few things more instantly indicative of a lack of attention in a MUD than descriptions that mention lots of things in the long description, but never provide items you can look at for them:

```
This is a nice stairwell.  There are some lovely stairs leading upwards.  There
are frogs on the stairs.

> look stairs
You do not think that the stairs is here.

> look frogs
You do not think that the frogs is here.

> sigh
You sigh.
```

Of course, you don't have to be too fanatic about this... describing the dew drop on the leaf of the stalk of the plant in the garden is unnecessary unless you really feel you want to. Most people will never look beyond a certain depth of `add_item`, but it's nice to reward those that make the effort by ensuring they have something to see if they're following your descriptions. Having a little joke at the end of a long chain of nouns is a nice touch too... it's the attention to detail that really makes a MUD special.

9.2.8. Commandment VIII: Thou Shalt Not Describe NPCs In The Description

If you have an NPC in your room for whatever reason, it should be a separate object and not a part of the room description. So rather than writing the wiry old shopkeeper with an `add_item`, he should be a separately coded NPC. If you don't do this, you get all sorts of inconsistencies such as failing to find a match when a player tries to kill them, or being told they are not a living target when spells are cast on them. It's also inconsistent with how the rest of the MUD deals with NPCs... where there is a living creature, you should be able to interact with it in the same way as with any other living creature.

```
This is a shop. There is a shopkeeper standing here.

> look shopkeeper.
What a clean old man!

> leer shopkeeper
You leer shopkeeper.

> ponder
You ponder.

> kill shopkeeper.
Cannot find "shopkeeper", no match.

> boggle
You boggle at the concept.
```

You may not want it to be possible for players to kill your precious NPCs, but that's irrelevant — there are other ways to do it than to make the NPC part of the description. It is inconsistent for there to be a mention of an NPC and no way to interact with it.

9.2.9. Commandment IX: Thou Shalt Not Put Command Hints In The Description

If you're describing a room with special commands within, it's very tempting to hint at these commands in the room description. This is Very Ugly, however, and should be avoided:

"This is a nice shop. You can 'buy' and 'sell' stuff here."

It's much, much better to put these command hints in a help file. Most standard shops will already have a help file, so you won't even need to do that. However, if your room or shop does something unusual, it's always better to give the syntax in a help file rather than in the long description. Partially this is to make the help system a homogenous, consistent thing. It's also to remove Out Of Character game information from the MUD.

Signs are also a valid way of providing information relating to how a particular room or shop works... but signs should always be in theme, prompting the player in how to use the room, but never directly quoting the syntax. Signs are most effectively used as a means of conveying game information relevant to a particular room, such as exchange rates, cost for services, and so on:

Bad:

```
                          Welcome To Bing's Bank!
                     We Give You More Bang For Your Buck!

                     You can 'apply' for an account here.
                        You can 'deposit' money here.

                          The transaction fee is 10%.

  Shop around! We guarantee you won't find a better rate anywhere else in this
                                      bank.
```

Better:

```
                          Welcome To Bing's Bank!
                     We Give You More Bang For Your Buck!

                          Apply For An Account Today!

                    We charge a low, low 10% on all deposits.

  Shop around! We guarantee you won't find a better rate anywhere else in this
                                      bank.
```

Just think how the signs in your local store, bank or supermarket look. That's the way all signs should look on Discworld. Leave the actual commands and instructions for the help file.

9.2.10. Commandment X: Thou Shalt Have Fun With Your Descriptions

This isn't just for your benefit. It's far, far more interesting and enjoyable to read descriptions written by someone who was enjoying the process of writing, rather than descriptions that have been mechanically churned out according to some rote formula. However, don't go too far with this. It's all too easy to just give in and be silly or surreal in the hopes it will make your descriptions funny and enjoyable. What it mostly does, however, is make you look like a loon. In all cases, try and keep your descriptions in theme, but fun!

```
This is a large marketplace.  Giraffes are bouncing on the stalls and pink
elephants in tutus are waltzing gently around the villagers as they shop.

> look giraffe
There isn't a giraffe here... what are you talking about?
```

```
> look elephant
Why would there be an elephant here? It's a marketplace.

> roll eyes
You roll your eyes.
```

One of the features of Terry Pratchett's work in particular is the flights of fancy he often goes into — there's nothing to stop you doing that in your own descriptions, but try to keep it thematically correct.

9.3. Conclusion

So, that's the commandments for writing room descriptions. In the next chapter we'll look at it more in terms of how to put descriptions together. Pay attention to these commandments and you'll avoid making most of the mistakes that mark out a new creator from a more experienced creator, and that's good — part of the reason you're reading these documents is to learn from everyone else's mistakes!

10. The Art Of Writing Room Descriptions

10.1. Introduction

Room descriptions are not the simplest things to write from a cold start, and are often a stumbling block for new creators. It takes a certain amount of mental exercise until you are able to write hundreds of descriptions without feeling overwhelmed at the task. Having a semi-formal process for writing is the way to gain sufficient confidence to be ready to describe anything you need, and in the quantities you require.

In the last chapter we discussed the commandments, but in this chapter we're going to take a more positivist approach to writing descriptions — how to get started, and how to be sure that they're good enough.

10.2. Your Mental Picture

The first step in writing descriptions is to picture what you're writing about. If you're describing a room, imagine what it looks like — get a feel in your dusty mind for the most distinctive features. A good room description is around 50 to 80 words in length. That may sound like a lot, but it's not really — in fact, once you get a taste for writing them you'll find it's quite restrictive.

The paragraph above is 72 words in length, so you can see that you'll soon use up the word allowance before you've really described everything you want to. What's useful then is a structured way to write the description so that you can channel your descriptive juices more effectively.

To begin with, consider what you can see in the far distance — this helps set the constraints of your "observable view". When out in the Wide Open, you'll have a far greater vista than you would in a cramped city, and you should reflect that in your description. Setting the context is a good way to begin a description — where you have great visibility, your description should begin by explaining that. Likewise, when within tight conditions, the description should reflect that too.

```
The trees here blot out the sky with their densely packed branches.  All around,
the forest is dark and forbidding... little light penetrates the shadows within.
```

Next you can focus on the main features of your immediate location — what is it that distinguishes this room particularly from any other room in the MUD? Close your eyes and imagine standing in the room — what instantly strikes you that you should mention? You don't have to include absolutely everything in the long description, just those elements that are immediately noticeable. Are the walls a particular colour? What about the floor? Are there windows? You should frame such details in such a way as to describe the constraints of the room — the bits that relate to where this room begins and ends.

```
The ground is strewn with organic matter shed from the trees above, forming a
carpet of leaves and wet, moldy bark.
```

Next, make mention of any items of particular note in the room. Does it have bookcases? Does it have a fountain? Is the ground strewn with rocks? Imagine yourself as a burglar casing the joint — which would be the things most likely to draw your eye in terms of the contents of the location?

```
Large rocks have been set in a circle on the ground, several vaguely humanoid-
looking wooden puppets hang from the nearby branches.
```

Finally, relate the room to the rest of the area. The exits line of a description gives you the available exits, but you can expand on these to provide actual, immersive detail. Don't write "the exits are north and south", because that duplicates already available information. Instead, you could do something like the following:

```
The road north leads into a clearing, while the path south plunges into the
heart of the forest.
```

Bringing these together gives you the full room description:

```
The trees here blot out the sky with their densely packed branches.  All around,
the forest is dark and forbidding... little light penetrates the shadows within.
The ground is strewn with organic matter shed from the trees above, forming a
carpet of leaves and wet, moldy bark. Large rocks have been set in a circle on
the ground, and several vaguely humanoid-looking wooden puppets hang from the
nearby branches.  The road north leads into a clearing, while the path south
plunges into the heart of the forest.
```

While the structure is important, what is also important is your choice of words. You should pick words that are evocative of the impression you are trying to convey. Be careful though not to exhaust your vocabulary! The word choice in our example description here reflects that this is a scary forest, and not a cheerful Disneyesque one. Mention is made of the darkness, and the forbidding atmosphere. The trees are portrayed as being active in blotting out the sky, rather than "The sky can't be seen because of the branches". The path into the heart of the forest doesn't lead into it, it plunges like a dagger.

You have to be careful with this kind of thing, because if you do it too much your description just sounds ludicrously over the top. Evocative phrases are to be peppered through your description like a delicious, but spicy, herb. Consider what happens when we turn the evocation up to eleven:

```
The trees loom all around, their vicious branches ripping at the sky like wooden
talons.  The forest all around emits an aura of death and despair, promising
only horror for those who dare explore its interior.  The ground is a wet
blanket of rotting leaves and decaying bark.  Jagged rocks have been set in an
ominous circle on the ground, and wooden puppets have been strung from the
branches like grotesque parodies of humanity.  The road leads north into a
clearing, with the path south leads into the darkened horrors of the interior.
```

Individually there's nothing especially wrong with any of the sentences in this description, but when they're all put together they scream "I'm trying too hard to scare you!", and the entire effect is lost. This leads onto a not insubstantial secondary problem... when it comes to writing the next description, how are you going to give the same impression without repeating yourself?

A common mistake that new creators make is to go overboard with adjectives. Not everything you see is interesting enough to justify an adjective:

```
The beautiful street is sparklingly clean.  Busy people go by carrying
interesting-looking bags of shopping.  Cheerful dogs bark with happy woofing
sounds.
```

It might look okay, but as with our "over the top" description it just doesn't ring true — it's somewhat artificial in its insistence that everything in the room is worth actually describing. You can't make banal things interesting, so don't try — don't force adjectives just because your high-school English teacher told you that they were important. Keep them in reserve for when they're actually warranted.

Finally, if you have a strong mental image you'll be able to appreciate that while you know what things are, people coming to your description without that context will be looking at a scene they can't necessarily parse. "People wander into the nearby bank" is probably fair enough, but how do the players know it's a bank? "People wander into the ornate building" is a little bit better, with the clue that the building is a bank reserved for its add item.

You can get additional flavour in a description by making use of more evocative verbs and pairing them with adverbs. "The shop dummy has a hat on its head" may be true, but is boring. "The shop dummy has a hat perched on its head" is better, but you can continue to refine this until it says something genuinely interesting, such as "The shop dummy has a hat perched jauntily on its head".

Active verbs can also be better than passive ones. "A sign is hanging on the wall" is okay, but how about "A sign hangs on the wall"? This is a more compact way of expressing things, which then gives you room to expand it again with more character: "A sign hangs crookedly on the wall".

It's hard to keep this up, description after description, item after item, but you soon get into the swing of it. The key is to have somewhere interesting to describe. Anystreet in Anytown isn't going to be easy to describe because it's not exciting. Luckily the fact we are in an amazingly detailed fantasy world makes our job that much easier. If you find it hard to write your descriptions, spend a bit of time thinking about whether you can put some kind of feature throughout your area. Maybe it has a river flowing through it, or it is nestled into a fertile valley. Having some kind of geographical feature to focus on can make everything flow a little easier.

10.3. Hand-Crafting

Every single room description should be hand-crafted. Copy and paste is the province of those who don't actually care about how their rooms look, and that's not something that's true of any Discworld creator. You also need to be able to provide several variations on the theme of your development, and that becomes exponentially harder to do if you put all your ideas into writing the first one.

It's fine to include variations on a theme in multiple room descriptions, but shy away from direct repetition. This is especially true too of "special words" in a single room description — while there are always words you'll need to repeat, the defining words in your description should be used once only. Otherwise, it jars people out of the narrative because it brings back memories of the first use of the word.

Sometimes, though, you're going to get stuck and be unable to provide a new way of referencing something in an interesting way. In such cases, you can consider going off on a tangent by not describing the thing directly but instead approaching it from a different angle.

```
Imagine trees of such beauty and splendour that simply being in their presence
is enough to make someone fall to their knees and praise the Gods that they were
alive to experience such wonder.  Imagine those trees, because they're nowhere
to be found in this forest.  These trees are just like all the other sad
examples you've seen so far in this desolate place.
```

Don't overdo it. This sort of thing is a nice surprise every now and again, but soon becomes tiresome as a narrative device. Flippant or jokey descriptions might work well the first time they are encountered, but they become progressively more irritating as familiarity grows. You don't want to be the creator who writes descriptions that all the players roll their eyes at.

The rule for uniqueness doesn't extend to add_items, although having a number of different add_items for things that appear in multiple rooms is always preferable. The more variety the better, but you get much less out of completely unique add_items than you do out of completely unique long descriptions. For one thing, the shape and length of a room description is a navigation aid for those who wander around with verbose descriptions on. Even if it's not being used for navigation, more people are going to see your long descriptions on a regular basis than will see the items in the room.

10.4. Room Chats

Room chats can be one of the most challenging things to do properly because they repeat so often in a room. A bad room chat will greatly reduce the overall "polish" of your room, and having a small set of highly repetitive chats can be worse than having no chats at all, especially in rooms where people may spend a lot of idling time.

Long descriptions and add_items shouldn't include activity; by their very nature, they are frozen in time, and so any activity you include in them is likewise frozen in an entirely artificial way. Any motion you want to describe should be migrated to the room chats. Imagine the following in an add_item:

```
Every now and again there is a soft thump as an acorn falls from a nearby tree.
```

That looks horrible, and not just because of the writing. It just doesn't feel active — it feels like you're looking at a painting of a tree dropping an acorn, rather than experiencing something in a living, breathing game world. On the other hand, if you occasionally see as a room chat something like:

```
There is a soft thump as an acorn falls from a nearby tree.
```

Suddenly that frozen activity becomes something active that adds a level of richness to your location. The exception to this, as a rule, is when your dynamic activity is an ongoing affair. The hustle and bustle of a crowd can be conveyed in the long description, and in an add_item, and also in room chats. The static descriptions should describe the activity in the abstract, and the room chat should describe time-sensitive specifics. You could have, for example, the following description:

```
The street is filled with a great hustle and bustle of people going about their
business.
```

Then in your room chats, you can include specific examples of that hustle and bustle:

- An old lady bumps into you, apologises, and then wanders off.
- A well dressed man stands on your toes, but walks off without noticing.
- A small child wanders past and laughs at you before wandering away.

The more room chats you have, the better. Like add_items, you can profitably share these between the relevant rooms. Having nine room chats shared between three rooms is better than each room having three unique room chats. Too few room chats tends to give a "groundhog day" feel to your area.

A good way to make for really dynamic areas is to have chats that are, at least in part, randomly generated. We'll look at how to do that in *LPC For Dummies 2* (although you might already be able to work it out from the material we covered in *LPC for Dummies 1*).

10.5. References

You shouldn't rely solely on your own withered cortex for writing — make use of the tools you have available. Buy yourself a good dictionary and a good thesaurus, or make use of the many free online versions. When you're struggling for a synonym, turn to your thesaurus. When you want to make sure that you've used a word exactly right, then check your dictionary. Don't leave it unresolved, and don't guess.

Be careful though with synonyms derived directly from a thesaurus — they're fine to prompt you with words you already knew but didn't immediately remember, but synonyms are often loose associations at best. Don't turn over your own internal editing process to an automated solution.

10.6. Quality Assurance

So, having written a description, how do you know if it's okay? The easiest way to do this, and I am not making this up, is to read it out loud. Out loud, mind — not just read it over.

When you read a description out loud, you instantly pick up on clumsy phrasing, repetition, and scansion problems. A good description flows, rather than stutters. To show this in action, read the following description out loud:

```
There are trees all around.  The trees are brown and look like trees normally
look.  Each of the trees have leaves.  The leaves are green and brown and found
around.  All of the trees is unhealthy and looks as if they are almost dead.
```

This is an exaggeratedly bad description for dramatic effect, but you should have instantly picked up on some of its flaws:

- Inconsistent scansion.
- Repetition of words
- Incorrect form of verbs

Scansion is perhaps the most difficult of these concepts to define for those unfamiliar with the idea, but it’s to do with the natural rhythm that accompanies verbal communication. It’s a term most often applied to poetry, but applies equally well to narrative flow. Syllables in English are either long or short, and the combination of syllables is what defines the “metre” of a written piece of work. We all instinctively have a greater or lesser feel for scansion, but when something sounds “clumsy” it’s usually because the rhythm doesn’t fit. That’s something that just doesn’t come across when you read something quietly to yourself — it only comes out when you speak out loud.

A formal discussion of scansion is way, way outside the scope of this material — it’s just something that you should bear in mind when putting together your descriptions.

Once you’ve written the description to your liking, the next step is to put it aside for a few days and come back to it. Read it aloud once more, and then change it so that any new issues you’ve identified are resolved. Sometimes you get too close to something you've just written and you need a little perspective — that only comes with time. If you can, get someone you know and trust to give you their honest feedback on what you’ve written — an outside perspective is even better than your own time-delayed thoughts.

10.7. Conclusion

Writing descriptions is a skill that comes with time and practice, but there are certain things you can do and think about to get yourself into the habit. Not every description is going to be a winner, and even descriptions that you like aren't going to appeal to everyone. That’s fine, though — the world would be very dull if we were all the same. All that matters is that you do the best you can to produce your highest quality of writing. It may seem like an insurmountable task to describe an entire village down to every item in every room, but it soon becomes a reflex. It’s also thoroughly enjoyable, when you get into the groove.

11. Non Player Characters

11.1. Introduction

Writing descriptions for NPCs introduces a new set of difficulties — people really aren't all that different, when it comes down to it. While the scenic panorama visible from one mountain top will differ dramatically from the panorama available at another, it's usual for one person to look much the same as another, within some fairly well defined parameters. How then do you write descriptions for NPCs to ensure that they look unique and interesting?

Additionally, NPCs are supposed to be responsive — they're supposed to say things and react to things. That's an additional difficulty in setting them up well — we have to write dialogue. If you've never done anything like that before, it's a brand new challenge to meet.

Finally, sometimes we don't want our NPCs to be killed because they are a part of in-game systems. We need to consider what kind of protections we can put in place for such NPCs, and why we might want to ensure they are protected in the first place.

11.2. NPC Classifications

As an informal system, we tend to categorise NPCs into one of several categories:

- Cannon fodder
- Service
- Quest
- Flavour
- Boss

Let's go over each of these categories in turn, and discuss what kind of creatures belong to each.

11.2.1. Cannon Fodder NPCs

Cannon fodder NPCs are those that you see wandering the streets. You can think of them as walking bags of tasty XP and loot. We tend not to spend a lot of time worrying about making them fully interactive, because that's not the role they fulfill in our game. It's nice if they respond to nods and waves and such, but it's only a small subset of players who will even consider interacting with cannon fodder, and an even smaller subset who will care that you made the effort.

As long as your cannon fodder NPCs are described with a long description, kitted out with suitable equipment, and contain some load_chats, you can move on to the next thing you have to develop. Remember, your time as a developer is limited, and you should focus your efforts where they are most needed.

11.2.2. Service NPCs

Service NPCs are those that exist to facilitate in-game transactions. They could be guild NPCs, shopkeepers, or city guards. A player killing these NPCs will impact on the provision of that service. One solution to this is to simply not have NPCs as part of the player transaction — for example, we can create shops without shopkeepers with no trouble. That's cheating, though — we want our game to be richer, not more artificial. What we can do is provide disincentives for players who attempt to kill them. That is the basic motivation behind practically every crime system we have in the game — to make it less appealing for people to murder NPCs that other people may need to make use of.

In terms of writing service NPCs, descriptions and chats are important, but even more important is the system you put in place to discourage their murder. There are ways in which you can simply make them impossible to kill, but that's unrealistic and lazy. We want the game to allow people to be bastards if they want to be, but we put in place punishments for that behaviour so that there is a way of balancing player freedom of choice with protection for game systems.

We'll talk about mechanisms to discourage gittishness later.

11.2.3. Quest NPCs

Quest NPCs are similar to service NPCs in that they are part of an in-game transaction, but their presence isn't continually required. Often they disappear when a quest has been completed, or are generated only when certain conditions are met. As such, we can't ensure their presence at all times, or even really necessarily tie them into a crime system. If a quest NPC is an animal, or a criminal, or in some way beneath the notice of the Law, it is not realistic for the legal system to extend protection to them. Where possible, the systems that protect service NPCs can be extended to quest NPCs, but not always.

Our solution to this in the past has been to make quest NPCs worth None XP, but while that discourages indifferent slaughter it doesn't do anything to punish someone who gets their kicks out of inconveniencing other people. Alas, we can't have everything.

For Quest NPCs, interaction is king — they should be responsive to questions and souls, as well as more chatty than other types of creature. Ideally they will mention things that are obviously related to their quest. NPCs hinting at the quests in which they are involved provide a useful mechanism for hinting that a player should spend more time conversing with this NPC than they would with another.

11.2.4. Flavour NPCs

Flavour NPCs have no formal standing in the game, but are unique and interesting regardless. Most of the Dopples in AM fall into this category — only a few of them are involved in the Doppleganger quest, most of them are just flavour NPCs. Sokkard is one example of a flavour NPC — he doesn't do anything much except spam the talker, and no one is inconvenienced if he is killed and thus unable to chat (indeed, some would say that the general convenience of the playerbase is increased while he is dead).

If NPCs exist to provide flavour, they should be fully described and interactive to ensure that flavour is... uh... delicious. The danger is that people will assume they are part of a quest, so unless you are especially vindictive don't make them say anything that could lead players to that conclusion. "Oh, where is my ring?" is an example of a chat that implies a quest is present, and it will send players off on a fruitless endeavour to solve it. The only result of this is that the players become frustrated and disillusioned.

11.2.5. Boss NPCs

Boss NPCs fill the role of "exciting combat opportunities", and are to be found usually as unique NPCs in the depths of an area. Their appeal comes from the fact they are a chance for high level players, or groups of players, to test their skills against a dangerous, unique foe.

More than anything else, Boss NPCs should offer interesting combat experiences — they should make full use of an appropriate range of skills and commands, and have a range of fun and hopefully humorous attack chats. They should kick, bite, disarm, crush, stab, cast spells, run away — the whole range of activities needed to allow them to put up a decent fight.

Their descriptions are not quite so important because, presumably, people won't get a lot of chance to read them. That's not to say you should "phone it in", but most of your effort should be spent on making the combat interesting rather than weaving clever functionality into the description and chats.

11.3. NPC Descriptions

First of all, we need a clear picture of this NPC in our mind. While we don't usually provide detailed histories of our non-player characters, it's worth spending a bit of time thinking about what kind of life the NPC is likely to have had. If they're a military type, what are the likely results of their past campaigns to have been? If they have a specific occupation, what are the physical traits that are likely to be associated with that occupation? People generally grow into the form they need for the life they have, so let that guide your mental image.

Additionally, think about what kind of upbringing are they likely to have had. Aristocrats will bring with them a haughty bearing and a particular way of behaving to people. NPCs from Forn Lands will have characterising features that can be brought out in the description. There are all sorts of connections you can make between the NPC's background and its observable behaviour.

However, identifying characteristics are usually highly individual, and not generic. Scars and broken bones add flavour to a description, but if used too freely they seem awkward and contrived. Remember that a player is often going to see many dozens of your more generic NPCs wandering around, and it'll look weird if every one of them has a scar above his or her eye, or if every one has the same broken bone that has been "clumsily reset".

One good way to add some variety is to use the basic structure as a template, and have the exact details generated randomly. For example, your NPC could get a random adjective such as "strong", "thoughtful" or "healthy", and their long description could change subtly depending on which of these adjectives were chosen. Additionally, you can slightly modify skills and statistics based on the adjective, so that characters with the adjective "strong" have a couple of extra points of strength, whereas one that is "healthy" has an extra couple of points of constitution. It doesn't make a huge difference when fighting them, but it adds a little spice. From the same basic codebase you could generate the following descriptions:

```
The thoughtful soldier is a tall, thin man with a scar across his left eye.
```

versus:

```
The strong soldier is a short, heavyset woman with a nose that has obviously
been broken and reset many times.
```

We'll talk about how to actually achieve this goal in *LPC for Dummies 2* (but as with having randomised room chats, it's something you might already be able to work out how to do from *LPC for Dummies 1*). You'll find many examples of this kind of thing in the NPCs that are already in the game, if you want to read a little bit of example code.

Your unique NPCs have the advantage that they can be fully described without having to resort to generality. You can make specific references to exact features, size, shape — whatever you like. Unique NPCs are usually named, and you can be a lot more specific about what their history is and how that has affected their appearance.

11.4. Equipment

The clothing that an NPC is wearing shouldn't be mentioned in the long description. If an NPC is supposed to be wearing something, you should of course give them that item to wear, but don't mention it otherwise. There are three reasons for this:

- The inventory system of the MUD is standard, and people expect to see items displayed in a particular way on an NPC.
- If clothing is mentioned, players should be able to look at the clothing. We have no equivalent of add_item for NPCs.
- You can't guarantee the clothes will be there — they may break, or be stolen.

The choice of clothing you give your NPC is important — as with long descriptions, it can either be something that adds to your NPC or detracts from it. If every instance of a generic NPC has exactly the same items, it's boring and jarring to your suspension of disbelief. Ideally your NPC will have at least a somewhat randomised set of equipment.

Genua, for example, has a system for randomly dressing NPCs according to their affluence, so that not only is every NPC wearing a fairly unique ensemble, it's an ensemble that they could realistically afford to wear. Poor citizens wander around in rags, and aristocrats wander around in the most expensive and stylish of clothes. Other domains may or may not have similar systems. If you're developing for a domain that doesn't have one, you could consider writing one yourself!

For unique NPCs, equipment should be fit for purpose. Make sure the Three Key Zones are covered — legs, chest and feet. Anything else you provide will enhance the NPC, but those three should always be provided. If an NPC has no shoes, it's barefoot. If it has nothing on its legs, it's naked from the waist down, and if it has nothing on its chest, it's naked from the waist up. Those kind of things get noticed...

In terms of weapons, bear in mind the environment in which the NPC will be functioning. Most people don't wander around with their weapons in hand, even in Ankh-Morpork. Indeed, most people won't even have weapons with them. The ones that do will likely have them sheathed until needed. It looks great if an NPC has a sheathed weapon in a scabbard that gets drawn when combat begins. Remember too that unless you give the NPC some skill with the specific weapon, it's likely to be more of a menace to itself than to others... if an NPC is unlikely to have had opportunities to develop meaningful proficiency then it probably shouldn't have the weapon and should rely on its fists alone.

One neat touch that adds a great deal of immersion to the game is if an NPC has random, useless items that hint that the NPC you just murdered has a spouse and children somewhere. You could very occasionally have an NPC carrying a small present containing a wooden duck, with the label "Happy Fifth Birthday, Sue!" It may not even be noticed by most players, but it'll be a nice and unexpected laugh for the ones paying attention.

11.5. Dialogue

For NPCs where responsiveness is important, it comes down to dialogue. Dialogue is really quite difficult to do properly, because at its heart it is a tension between the convenience of gameplay and the believability of the revelation. It's tremendously common in games for NPCs to blurt out every intimate detail of their life to the most casual of inquiries, but that doesn't make for a rich dialogue, nor for a rich experience.

With dialogue, we are looking to build a genuine opportunity for players to interact with our game world. If our characters are to be believable, they need motivations. They need back story, and they need reasons to talk to the player.

This is mostly an issue with quest and flavour NPCs rather than the others, but a believable character is one that requires us to tease out the information we need rather than to have it dumped entirely on us as a lengthy monologue.

Dialogue in a MUD is closer to that of a film than that of a book — dialogue in films is snappy, direct and delivered as exchanges. It's hardly ever paragraphs of unsolicited revelation.

So, our first aim in writing dialogue is to build up a tree of how revelations will unfold. Our character may be looking around for something, and that hints to the player that a quest may be somewhere in the offing. How our NPC responds to enquiries is how we build the dialogue. This is dull:

```
> You ask: Do you need any help?

Captain Beefy says: Yes!  I've lost my marbles!  I need help finding them!  I
lost them down by the Old Creek yesterday.  Could you try to find them for me?
I will give you ONE MILLION DOLLARS!
```

Where's the interaction here? Where's the drama? You're just typing a thing and getting candy. People don't talk like that. It's much more interesting, from a game design perspective, if you make the revelation a part of the interaction:

```
> You ask: Do you need any help?
Captain Beefy says: Sometimes I feel I'm beyond help.  I just have this sense of
loss.
Captain Beefy looks around helplessly.

> You ask: Have you lost something?
Captain Beefy says: Hrm?  Oh, yes.  My marbles.  Careless, really.

> You ask: Where did you lose your marbles?
Captain Beefy says: Oh... I suspect it was down by the Old Creek yesterday.

> You say: I will find them for you.
Captain Beefy says: You... you will?  My word, that's generous of you!

> You ask: Is there a reward for their return?
Captain Beefy says: Oh, yes... I suppose so.  How does ONE MILLION DOLLARS
sound?
```

If you want to build a convincing discussion, you can even build in suspicion — you can make it so that you need to talk the NPC into giving up the goods.

When writing these dialogues, bear in mind your mental picture of the NPC again. How are they likely to speak? It's very easy to lapse into stereotypes here, so try not to. Are they direct and to the point? Flowery and poetic? Do they speak clearly? Mumble? Lose their train of thought? Are they articulate? Do they communicate mainly in grunts? All of these things help you frame the dialogue appropriately.

11.6. Conclusion

This chapter hasn't really been about the writing that goes into NPCs — it's more about the thinking that lets you write them. Having a firm idea of why your NPCs exist, what they are for, and how they are supposed to interact is a great way of making sure that you're emphasising the right things in the right areas.

In the next chapter we'll talk specifically about Betterville and look at the NPCs we're going to make available, what they are for, and what dialogue opportunities they are going to present.

12. Beasts Of Betterville

12.1. Introduction

We now return to our regularly scheduled programming — first we talk about a Thing and then we relate that discussion to our design of Betterville. In this chapter we'll look at the NPCs we're going to make available in our development, including what they're for, what we need to provide for them, and how our players will be able to interact with them.

We have a fairly small development, so we can't go overboard with this — we may have cause to add more if we need to increase the feature density of the village, but at the moment we're really looking at three types of NPC — a shopkeeper, the Beast, and wandering young women who came to the village for Social Advancement.

12.2. Our NPC Manifesto

We haven't yet spoken about what to do with the Beast. My suggestion for now is that we simply make him a flavour NPC rather than a Boss NPC. We won't be speaking much about combat in *LPC For Dummies 2*, so we'll concentrate on making him interesting from a narrative perspective rather than from a combat perspective,. This also ties into our second NPC.

Our shopkeeper is a Service NPC — she keeps the shop, and makes the service of that shop available. We're going to protect her from Disrespectful Players by adding a disincentive for killing her. Namely, the Beast is going to tear the player a New One!

Finally, we have our cannon fodder NPCs — the Young Romantics. These will not offer much in the way of a challenge since it doesn't make sense that they would be skilled fighters, but they'll be generic NPCs that wander around the village ready for slaughter. We can have maybe three or four of these wandering around at any time.

Together, this creates the NPC complement for our village — it's not much, but then it's a small village and isn't likely to attract much in the way of activity.

Our task for each of these is to spec out the details — although we can write the descriptions in advance of actually writing the code, we're not going to do that. Instead we're going to develop the framework into which each is to fit.

12.3. The Young Romantics

Let's start off with the simplest NPCs — our cannon fodder. We're going to have three or four of these wandering around a time, and so our discussion about randomising some of the setup becomes relevant. Logically, not every one of these young women will be the same — they'll come from different places, have different looks, and be wearing different clothes. Ideally they'll even have different short descriptions to add a bit of interest. In Genua, as an example, you'll find cadgers, mendicants and beggars — they're all the same NPC, just with different shorts. We can make all of that Come True with only a tiny little bit of Code-Fu.

To begin with, we'll provide a list of all the possible short descriptions:

```
({ "gold-digger", "wannabe princess", "upwardly mobile socialite" })
```

Next, let's work up a list of all the places they may come from:

```
({ "Lancre Town", "Ankh-Morpork", "Genua", "Djelibeybi", "Sto Lat" })
```

The nationality will be used for us to set up a language and an accent, and maybe even the kind of money the NPC has on her. We can also use it as the basis of her description — simple to do by just passing the nationality into a switch statement and having that deal with the long.

```
This is a young woman from Lancre Town.  Lancre has few opportunities for social
mobility ever since Verence married Magrat, and so she has come here hoping to
improve her lot in life.
```

This is just the first part of her description, but already we have a mechanism for distinguishing one NPC from another:

```
This is a young woman from Genua.  Genua City is full of opportunities for
anyone looking to marry into wealth, but this is done in a cut-throat
environment of vicious competition.  She has come to Betterville in the hope of
finding an easier time of it in the Provinces.
```

We can further improve our descriptions by adding randomisation to out NPCs' physical features. So, for hair we could have length:

```
({ "long", "short" })
```

And colour:

```
({ "blonde", "brunette", "red", "black" })
```

We could also provide a body size, shape, and height:

```
({ "fat", "thin", "wiry", "plump", "well-built" })

({"tall", "short"})
```

And then we combine them together into a standard framework, such as :

```
She is $height$ and $body_type$, with $length$ $colour$ hair.
```

By slotting random elements of our lists into this, we get things like:

- She is tall and fat, with long red hair.
- She is short and wiry, with short blonde hair.

The more entries we add to the list, the more variation there can be. It's a great way to get extra Value from very little expended effort.

In terms of their outfits, it makes sense considering our previous discussions if they are wearing the outlandishly exotic princess outfits from the local shop. We'll thus make their clothing based on whatever we add to that shop, again using some kind of random setup mechanism.

Their load_chats are going to explain why they are there — otherwise there is no logical connection as to why all these Wannabe Princesses are wandering around a rural village square. We need things like:

- Have you seen the Beast? I'm going to be his one true love.
- They say a kiss will turn him back into a prince.
- We're going to get married in the spring. You know, when I find him.
- I can't believe those other fools think they might be his One.

Perhaps something on the general theme of vanity:

- The wannabe princess checks her hair in a hand mirror.
- The wannabe princess brushes a speck of dust from her dress.

When referencing items in the NPCs' inventories, we need to be especially careful to make sure that they actually have them. We talked a little bit about how to have chats that reference items in an NPC's inventory in *LPC For Dummies 1*.

The load_a_chats will similarly reference the lack of joined-up-thinking that brought them to the village:

- The Beast won't be happy you're killing his one true love!
- This isn't part of the fairy tale!
- Stop, this is part of a different story!
- No, you'll tear my dress!
- I think we all know who the real beast here is!

In terms of their weapons, they won't have any. Fantasy princesses never did — it was their role in life to be the cause of weapons in others.

Interaction possibilities revolve around their reason for being here — perhaps they could respond to kisses and cuddles with protestations that they are saving themselves for their new sweetheart and such. Maybe they could have nasty things to say about where they came from:

```
>say Are you from Lancre?
The wannabe princess says: Oh yes, but I left there. Too many gold-diggers.
```

Just because they are generic doesn't mean they can't be interesting, after all!

12.4. The Shopkeeper

Our shopkeeper already has a distinct personality we can infer from the circumstances. She is opportunistic — she has built her business around the folly of young women, and it takes a certain kind of moral flexibility to do that.

She should probably also look the part — it would be difficult to convince people they were part of a fairy tale if you just wore overalls... as such we'll make her look a little bit like the stereotypical "evil witch", with a figure-hugging black dress and long, talon like nails.

We can describe her without resort to randomisation because she is unique. And because she's unique, we should also give her a name, and it should be something grand to befit the setting — "Lady Tempesre Stormshadow". Of course, this isn't her real name. It's her professional name. We could provide hints to that elsewhere if we liked — in the library, perhaps.

Her interaction responses should centre around her shop, her clients, and also her past. We could allow things like:

```
> say Lady Tempesre Stormshadow?
Lady Tempesre Stormshadow says: Yes, of the Genuan Stormshadows of course. We
are an illustrious, old money family.

> say You come from Genua?

Lady Tempesre Stormshadow says: Yes! From Genua, which is... uh... far away from
here!

> say Tell me about the city of Genua.
Lady Tempesre Stormshadow shuffles her feet.
Lady Tempesre Stormshadow says: Would you like to buy a dress?
```

Try to keep your response in theme with how someone would actually present themselves to the outside world. These are unlikely load_chats:

- I'm conning all these romantic fools out of money!
- I can't believe they think this tat is actually going to help.

Regardless of whether she is likely to see you directly as a customer, she is unlikely to reveal her Game Plan is such an obvious way. It would be more like:

- Come buy one of these fine dresses, and show the Beast you care!
- The Beast wouldn't give his heart to a poorly-dressed bumpkin!

More important than the chats, though, is the disincentive we're going to put in place for killing this particular NPC. I've already said what that's going to be — the Beast will attempt to rip the player apart. We need to set up a link explaining why that should be so. Perhaps one of the reasons why our Beast is in a rural village rather than the Unseen University is because he fell in love... this particular lady could be his former paramour. That would set up the necessary connection, and justify why he would be displeased at the love of his life being mistreated.

In order for that to ring true, we need to signpost it somewhere. Perhaps in an add_respond_to_with:

```
> say Do you know the beast?
Lady Tempesre Stormshadow sighs. Lady Tempesre Stormshadow says: I did. Once
upon a time...
```

This is also perhaps something that could be researched in the library. It doesn't really matter how we set up the link, it just has to be there.

How the Beast will enact his punishment is up to us. Perhaps upon attacking the shopkeeper there is a tremendous roar in the distance, and the Beast runs from his tower to save her. That adds tension to the combat — it's suddenly against a time limit. However, it won't at all discourage people who can kill the shopkeeper in the time limit, or who have no fear of the Beast.

Perhaps he appears instantly, so the combat becomes two against one. That might be more effective, but is far less interesting from a gameplay perspective.

Perhaps he doesn't come to her aid at all, and simply attempts to disembowel any murderers who encounter him. It's entirely up to us, and it might be worth experimenting for a bit to get a feel for which of these you like. For the purposes of our coding to follow, we're going to go with the third option, because it leads to a discussion about some interesting coding consequences.

Our discussion of Lady Tempesre leads us naturally onto the Villain Of The Piece.

12.5. The Beast

The Beast is our Star Attraction, and the reason why all of the activity of the village is ongoing. As such, we need to consider carefully what we actually want from him. The obvious choice is for him to be a Boss NPC — a challenging, deadly fight with appropriate reward. That" not the direction we're going to go, for all the reasons we've previously discussed. Instead, the Beast will be a mostly gentle, rather sad creature. Partially this is because — well, he's a Beast. And partially it's because he has been spurned by Lady Tempesre Stormshadow since his unfortunate shape change. The only time this will change is when he is roused into action by attacks on his lost love.

So the Beast is therefore a flavour character who gets roused into action at irregular intervals. The most important thing for us then is his interactivity, but we certainly shouldn't neglect how he looks.

The Beast is an example of an NPC that doesn't necessarily need to be clothed, although there is also some humour value if he is. He also doesn't necessarily need to be described to the last detail — certainly the imagination of the average player is more than up to the task of filling in the blanks if we provide them with an intentionally fuzzy overview. Much as how the best horror movies and books leave the details to the imagination, there is a lot to be said about maintaining the mystery for our beast. It's perfectly okay to "talk around" a description without giving away the farm. Consider this:

```
The beast is seven feet tall, with brown matted fur and horrible yellow eyes.
His hands are little more than paws, each tipped off with razor sharp talons.
```

As opposed to:

```
That the Beast was once man would never be apparent from his appearance.  The
magic that worked through him has shredded away every last trace of humanity,
leaving only a core of pure animal rage.
```

In the former, we describe the appearance — that's fine if you feel you can do a good job with describing something that exists entirely in your imagination. In the second, we describe the essence, not the specifics — we let the player decide exactly how the Beast appears in their own mind. I'm not saying this is the best way to do it, just that it's an option when dealing with things that are entirely alien to our life experiences.

In terms of his interactivity, we need to make him responsive without misleading people into thinking he grants a quest. That's quite hard to do considering the nature of the character himself... we can easily mislead people with chats like this:

- The Beast says: I wish I could find a cure to this horrible curse!
- The Beast says: If only I could receive True Love's Kiss!

Of course, the adventurous response would be to say "Well, let's make that a quest!" I like your enthusiasm, but we'll talk about why you should be wary of such thinking later.

Instead, we want to make the Beast appear like he has accepted his fate, but isn't happy about it:

- The Beast sits up on his haunches and mournfully grooms himself.
- The Beast sighs sadly.
- The Beast gazes contemplatively out of the window.

All of these indicate passive acceptance of his curse, as well as hint a bit at the personality of the NPC (he's a bit of a self-pitying drama queen). There's nothing to say we couldn't upgrade him to Quest Status later, of course, but that's not what we're aiming for at the moment.

In terms of interaction, we can build the Beast into a fount of knowledge about the history of Betterville, about Genua in general, or about witch-induced curses. This rewards those who explore with a narrative payoff, and also integrates our development into a wider context. All of these are Positive Outcomes.

Remember too that our Beast used to be a wizard, so he's likely to speak in a way that is uncharacteristic for his appearance. However, as with most things in Discworld, there is complexity in that — it is Canonically Established that occupying a body of a certain type causes the mind to reform into the context in which it finds itself. Our Beast may be eloquent, with occasional lapses into animalism. That could make him an interesting, albeit frustrating, conversational partner.

At one extreme we have complete eloquence:

```
> You ask: Why are you a beast?
> The Beast says: Oh, fairytales are in the lifeblood of Genua... our land is
forever changed by the power of raw and untamed narrative.  I am one of the
victims of that legacy.
```

But we could use that same answer and mangle it a bit to make him both unable to control his lapses, and yet entirely random as to where those lapses occur. Questioning the beast could require many attempts to get him to answer the same question:

```
> You ask: Why are you a beast?
The Beast says: Oh, fairytales are... gnh...
The Beast snarls and grunts for a bit.
The Beast says: I am one of the victims...
The Beast leans back and howls.
```

The next time you ask he may be eloquent at different parts of the story. This conveys the impression of someone thoroughly damaged by his exposure to the fairy tale (which is consistent with the books and also adds a touch of pathos to what could otherwise be a simple one-dimensional parody), and also perhaps add a frisson of danger — who knows when he is going to become sufficiently animalistic to attack?

We could even build that into the NPC — every conversation with him has an expiration date at which point he forgets himself and lunges to the offensive. But, like an animal, it's nothing personal and you'll be able to converse with him again when he gains control of himself once more.

12.6. Conclusion

Our Cannon Fodder NPCs are not interesting, although they can lead into some fairly self-contained jokes about self-obsessed social mobility. Our other two NPCs are a little more complex, and only by conversing and exploring will the player be able to find the deeper backstory that links everything together. Many players won't care for the drama you are weaving, but those who do will appreciate a cohesive narrative that explains why everyone does the things that they do.

Everything in your NPCs should ring true — they shouldn't divulge things they wouldn't divulge to strangers, they shouldn't dump their entire life story as an answer to a single question, and they should have realistic motivations and desires. Taken together, the whole can be much more than the sum of its parts.

13. Feature Development

13.1. Introduction

We've now spoken about our quests, our rooms, and our NPCs. The next step is to look at the two features that mark out our development — the library and the shop. The library in particular is a unique feature that is not supported in any of the MUD base inheritables, and so we're going to have to think very carefully about how it is to be implemented.

It's also a good idea to think about the shop before we put it in place, to make sure we can actually have it produce a range of interesting and unique merchandise. If we can't do that, we can rethink our plan as necessary. Plans are malleable after all, and we put all this thinking in before we start coding so that we can completely change what we're aiming for with no cost.

13.2. A Local Shop For Local People

Let's start with the easy one first — the shop. Producing items may not be the most inspiring of tasks (although they add a huge amount of richness to the game), but if we are adding a new shop it's a cheat if we don't make its stock at least partly original. Luckily, Roundworld is the best inspiration of all for items, because whatever we may be developing, we will find examples on The Interwebs. Researching for inspiration is a big part of putting together a unique development — you don't need to copy things exactly, but they can give you the starting point you need, and still ensure that your items are grounded in reality.

For a shop to be worth its title as a "feature", it should have a reasonable amount of unique stock — the more the better, although if there is too much it may be sensible to spread it over multiple stores as to more accurately reflect the feature density of your area. It's better for two stores to have a reasonable amount of stock than for one shop to be oversupplied and another to be barren. It's also better to have two reasonably stocked shops than just one of them with a huge stock list that cannot be easily navigated.

I won't give specific URLs for inspiration here, I'll just recommend you fire up your favourite search engine and look around for a bit. Looking for fancy dress or "fairy tale" outfits will fill your browser with possibilities, and some are even safe for viewing at work. Others aren't, and if you find any of these you should send them to me so that I can... uh... properly protect you from them. Yes.

Your first task is to decide on exactly what the remit of the shop is — is it purely for gowns? Gowns and shoes? Gowns and accessories? Jewellery? You need an answer to all of these to ensure you cover all the bases — the answer itself doesn't matter as much as simply having an answer.

Once you've decided on what is to be stocked, think about what is going to be available. You should aim for a stock list of roughly eight to ten items, which doesn't give you an awful lot to work with if your remit is too large. Four categories of items will give you around two of each — that doesn't permit a wide range of possibilities, but it would be possible to buy an "outfit" rather than having to mix and match.

Let's make it easy for ourselves and have our shop selling two kinds of items — gowns and shoes. That means we need to provide four or five examples of each — that's a sufficient range to provide interest, and yet not so ambitious that we have little hope of hitting the target. It's a nice balance between the two.

Next, we need to think about the infrastructure for the shop — how we are going to handle theft and such. It doesn't make sense for there to be a rural crime system, and while the Beast may object to people killing the shopkeeper, he probably isn't going to get all that wound up about petty larceny.

We don't necessarily need to have anything in place for it — that's perfectly fine, just as long as it's a direct decision and not an oversight. Just because we're planning things out doesn't mean we need to handle absolutely everything a player might possibly do. We'll simply give the shopkeeper a healthy amount of perception and let it all sort itself out.

13.3. The Library

Now we come to the Meat and Taters of the development — our library. While we already know this is a questing hub, we also need to make it interesting on its own behalf, since that was its original intention. It doesn't need to be bizarrely elaborate, it just needs to add to the richness of the game.

We already know of the "sorting the library" quest, but once it has been sorted it would be nice if people could make use of it to find out information — this information could be about Betterville, about the Beast, about Genua, or... even better... about other players. However, we don't want to have to write hundreds of books, even fake books of a paragraph each. It's Much Too Much to do anything like that.

What would be best is if we had some kind of index card system that lists a range of topics, and allowed the player to simply "research" the topics on that list. Doing so would give a random piece of information about the topic at hand. For example, we could have a topic about the Beast, and provide a random snippet to the player who chooses to research that topic:

- You research the Beast in the library, and find a ripped entry from his journal. Apparently he used to be a wizard, but he upset some powerful witch and she cursed him into the form of a hideous, animalistic creature.
- You research the Beast in the library, and find a Hogswatch card used as a bookmark. Inside it says: "From your one true love, Hilda Higbottom."
- You research Hilda Higbottom in the library, and find a letter to the Beast in which she expresses her desire to change her name to something more romantic.

Each of the snippets can lead on to other snippets, making the library a little puzzle of its own. Indeed, it could even be a quest to piece together some great mystery — but making such a quest dynamic, randomly generated and yet still possible to solve is a tremendous challenge, and one that we will not set for ourselves for the time being.

The real meat of such a feature would be a way to research players, because that adds a measure of interest to socialisers as well as explorers. Obviously we don't want the information given to be overly powerful (no revealing of skills, or stats, or level), or freely available elsewhere (projects, plans, refers, etc). We're not limited to fact, though; there's nothing to stop us making up slanderous lies about players provided we do it in a humorous enough way.

First though, we don't want every player to appear in the library — having your name mentioned should be a mark of honour, not just a random occurrence. It's unlikely that a brand new player with no renown would appear in anyone's library. Ideally it would be possible for someone to be entered into the library as a reward for some other achievement. Perhaps those who managed to ascend to see the Beast get their name recorded in the library for others to research. In that way, it becomes another way in which a player can make an impact on the game world.

If we do it that way, we shouldn't have their information be entirely random, because it would be possible for someone to simply research the same person over and over again to find out the full range of possibilities. Instead, their information should be generated randomly when the necessary steps have been achieved, and that same information returned each time that person is researched.

The information should ideally be Funny, because that is an incentive in itself for people to research other players. Perhaps the Beast could make a note of each person he encounters in his journal:

```
   You research Drakkos in the library, and find a crudely handwritten report in
   one of the Beast's journals: "I met Drakkos today. He's an odd little man,
   and smelled vaguely of donkeys.  He grunted something unintelligible, turned
   around, and promptly fell down the stairs. I hope he comes back some day."
```

As with the randomly generated long descriptions of our Young Romantics, we can adopt a similar system for the library. First we add some descriptions:

```
({ "odd little", "small-minded", "confused", "smelly" });
```

We can add the gender to that when the information is entered to make it correct. We then add a little insult:

```
({ "smelled vaguely of donkeys", "walked with a stutter",
   "talked with a squint" })
```

Then, in a similar vein, we set up the action the player took, and the result. We then end with a little afterthought.

We could even have several of these templates into which we slot the information — after a short time a single template becomes so familiar you can see the “Oz behind the curtain”, and by mixing it up a bit you can postpone that moment so that it all appears Magical for longer.

We should also consider our library as an opportunity for some useful taskmaster increases — any feature like this is going to involve some measure of skill in teasing out useful answers. It doesn't have to be anything especially useful — `people.culture.whatever` is a good possibility, as are the various languages. It’s unlikely taskmaster chances in these skills will draw in the Punters, but it's a nice bonus for those who want to explore the Comedy we have in store for them.

To further tie the library into our plans, we could liberally sprinkle a few quest hints here and there. They don’t even have to relate to this village — but if you’re going to hint at quests elsewhere make sure you clear it with the appropriate authorities. In-game quest hints are great because they are entirely in theme in a way that the web interface can never hope to be.

13.4. Feature Creep

Now we add a word of warning to our discussion of features — the importance of restraint. There is a reason why we put a plan in place before we start coding — it’s to set the constraints of our project at something reasonable and achievable within a specific time frame. What tends to happen as you develop is that you think to yourself: “Oh, and I will do this! And this! And wouldn't it be cool if I did that too?”

This is common, and entirely understandable. The problem is this — you will always have more ideas than you will have time, and if you continually get distracted by new ideas, you will never get around to finishing your project. There's a phrase for this — “feature creep”. As a development progresses, more features start making their way into your todo list.

It’s incredibly important to stick to a solid plan. I don't mean you have to be constrained by it — you can change things around as and when you need to. However, you shouldn’t add things to it on a whim. Every addition should be on the basis of you fixing some kind of issue that was encountered during the development. There will be plenty of time when the project is completed for you to add new features.

There are dozens of projects on Discworld that were delayed by years because of feature creep. The original Lancre Castle was started way back in 1997, and entered playtesting in 2008. During that time it expanded from a fairly reasonably achievable project to a vast, lumbering monolith — all due to feature creep. As new creators came on board the project, they brought their own ideas and put forward their own plans for how the castle should be. I'm not saying that these ideas were bad, but they were expanding a development far, far beyond its original scope. Eleven years is an awfully long time for an area to be under development, and indeed Lancre Castle has still not made it into game even now. The Milords and Miladies Wedding Pavilion was started in 1998, and took until 2004 before it went into the game.

Long, prolonged developments are a death march for a domain — they are indicative of a thing called the Concorde Effect, or the Sunk Cost Effect, whereby the amount of effort invested in the past is used as a justification for future effort despite the fact that is an unsound judgement. They also exemplify Brook's Law — adding manpower to a late software project makes it later.

Throwing creators at a death-march project exacerbates our burnout problem. It's difficult to remain motivated when you're working on a development that has been ongoing for years and years, with no sign that you're going to see any of your code in the game any time soon. Best avoid this problem entirely by setting yourself realistic and achievable goals.

New features can be added into your development, but leave them until the end. Develop according to your plan, and only once that plan has been achieved should you consider what else needs to be incorporated to make your development as good as it can be. There are a few examples of this in our plan for Betterville — the village is a little sparse, and could benefit from having another feature. There is scope for another quest or two in terms of the backstory we've set up. The problem is: if we plan all that to begin with, we'll never get finished.

As a new creator, you're not going to know what your limits are. Unless you have a lot of experience with coding, you just aren't going to be able to estimate how long things are going to take you to write. Once you've added something to your plan, you are committing yourself to writing it. Many creators find it difficult to say "Actually, no — I'm not going to have that feature after all." The trend is almost universally for features to be added, not subtracted.

This brings us to the second important point — you should be prepared to make cuts in your plan if you find you were originally over-ambitious. This is something you should discuss with your supervisor, but we prefer modest but completed projects over ambitious but never-ending developments. Features can always be added later.

Exactly how long a development will take will vary from creator to creator — I have no way of explaining how you will estimate it other than "it comes with experience". Ideally a project will be relatively short, giving you a reasonable turnaround from project to project. If there is a particular feature that is going to cause problems with your estimation, then consider recasting the feature. You can scale it back, or remove it entirely and replace it with something more achievable. This is not an admission of failure, but instead a mature and responsible attitude to meeting your creator obligations.

The ideal approach, then, is to be conservative in your plans, finish what you say you will develop (and no more), and then later direct new features to plug gaps you have identified. This is the Path To Success, Enlightenment, and Love.

13.5. Conclusion

A good area has unique features to tempt players to explore. Our development will have two — a clothing store full of unique and interesting fairy tale garments, and a library chock full of jokes, narrative, and humorous backstory. It may be tempting to look at these two features and think “No, it’s not enough”, but that kind of thinking leads to the danger of feature creep, and we should avoid that when possible.

Think of this as a Law of Nature — plan the plan, but keep it reasonable. Develop that plan as-is, making adjustments only on the basis of what you can realistically achieve. Once your development plan has been completed, and the project has been coded, do a feature density analysis and decide what additional features are needed to make your project fit for the game. Only by being strict with this can you avoid turning your project into yet another Nightmare March.

14. Finishing Up

14.1. Introduction

We've pretty much planned the hell out of our little village, and we're coming to an end of this set of material. The last thing we need to talk about in terms of our specific development is its integration into the wider context of Discworld. This may be easy, or confusingly difficult, and that will depend on how well-defined your development's domain is in terms of geography and infrastructure. Adding a street to Ankh-Morpork is easier than adding a village to Genua, for an example of what I mean by this.

This is largely something that will be a problem for your domain leader rather than a problem for you specifically, but we'll talk about the issues that are likely to be encountered so that you are prepared for the discussions to follow.

14.2. Integration Mechanisms

When Genua was first released, it was very hard to get to. To begin with, it was only accessible as a stop on the T-shop route. As more and more players made their way there, portals began to become commonplace. Now, Genua is tied in to both a carriage system and a ferry system that makes it accessible to all.

It was not our specific intention to make Genua so difficult to reach — it was a consequence of geographical remoteness. The terrains in between Genua and the nearest other domain development simply did not exist, and so it couldn't be slotted into a pre-existing environment. We had to come up with alternatives.

While it's always possible to have artificial means of access to an area (as an example of this, in 1998 you got to KLK by talking to A Man in Sator Square, and he teleported you there in exchange for Some Money), these are artificial and unsatisfying for the majority of locations. The Underworld is perhaps an acceptable exception, since it technically doesn't exist in the same world as the rest of the Disc. Anything that has geographical context should be tied into the world using a less obviously fake method.

The ideal situation is that there is a placeholder available for your development. In the terrains you can wander to where Zelah is supposed to be, but cannot enter it. If your project was Zelah, all you would do is make that location lead into your development and your integration is complete. Likewise if your project is a subarea in a larger area (a street in a city, for example), you just add an exit at the appropriate location, and you're now connected to the Real Game.

For areas that are not covered by terrains, this integration has to be done in a different way. Adding a location to the T-shop works, but is unreliable and frustrating — finding the shop is a difficult enough task, but if you also then force players to wait until it stops at a random stop out of dozens — well — satisfaction is far from guaranteed. It works great as a complement to other methods, but not so much as the *only* method.

Carriages and ferries are extremely useful because they are semi-reliable — you know exactly where the carriage is going to go and where you need to catch it. For those players without access to easy transportation, though, they are an exercise in frustration, turning active play into a Waiting Game — you wait for the carriage (blink and you'll miss it), then you wait for the carriage to arrive at the location in which you are interested. Along the way, you kill time.

Stitching your development into a larger context is the best case, since it gives players an alternative that isn't simply waiting. It is quicker for an impatient player to walk to Djelibeybi from AM than it is to wait for the carriage and then wait for it to arrive in DJB. Not only is it quicker, it's active — you're actually playing the game rather than waiting until you are allowed to play it again.

Blending these approaches is always going to give the highest amount of satisfaction — the more accessible your area, the more likely people are to want to visit it, and that's in your best interest as a creator.

14.3. Advertisement

Having completed a development, it's only natural that you should want to tell people to go visit it. There are several ways to do this — the simplest way is to do it Out Of Theme, such as posting on the boards, making a note on the Recent Developments blog or, if the area is large enough, making a news announcement. There is nothing wrong with any of these approaches, but also consider making the news available in the game.

Creators can, for example, place public notices in various cities — you could buy a message at the Town Criers in AM, or have a public notice displayed in Genua. A big city being tied into the game is likely to have consequences elsewhere too — Genua had an embassy in AM which was a source of news about the progress of the domain. A new village in the Ramtops might create gossip in nearby taverns. You could add a bit of gossip to the gossip handler, making the information available throughout the Disc. Other domains have their own mechanisms for promoting internal developments, such as the Rumour Mill in Forn. You should check around to see what's available, and make use of them all.

What you're looking to do is create a buzz, not necessarily signpost everything — you're not a tour guide. Your advertisements also shouldn't undermine the thrill of exploration that people get out of finding something new. This is an example of a bad advertisement:

```
Come visit the new area in Learning!  It's called Betterville, and you'll find
it by going east, north, east, west, south, and west from the village of
Learnville!  It's a parody of Beauty and the Beast.  It has a library and some
quests, and a shop with some things!  You should check out the library
especially, because it's full of Hot Funny!
```

The problem with this is that you've removed all of the mystery — those who don't find your advertisement convincing just won't visit, and those who like to be surprised will find nothing of interest left to explore. What's much better is if you hint at the great things to come for those who wish to spend a little time investigating:

```
Word has come out of the mountains of a newly discovered village, situated in
the shade of a tall oak tree. Rumours persist about a dark secret at the heart
of this secluded area, and caution is advised...
```

People are going to assume that there are new quests when an area is introduced — those who don't assume will soon be made aware by the increase in available quest points. Saying it's a village is going to also provide some expectations — you don't have to remove the anticipation by telling people everything they should look out for. We also mentioned there's a dark secret without telling people what the secret is — they will know to pay close attention and explore.

We are criticised occasionally by players (and other creators) for our often cryptic news posts, but I personally favour them as being much more in theme with our development sensibilities. While I don't endorse cryptic posts relating to gameplay issues (such as rebalancing of weapons, skills, removal of spells or such), I do feel that they enhance the experience for in-game developments. However, this does not extend to the Recent Developments blog — posts there should be clear and unambiguous, otherwise it's not fulfilling its designed role.

As with anything though, it has to be done properly. A news post saying "There's a new thing" doesn't do anyone any favours. First, you need to give a fair idea of what the new thing is. You also need to give a hint of some kind as to where to find it — that can be built into the style of the writing. Imagine if you were advertising an area full of trolls:

```
Dere's a new place.  It has uh... many fings to look at.  There is one... two...
uh... many fings to hit.
```

The first hint is in the style of writing — "Hrm, trolls". That leads implicitly to a clue for where the development is situated — it's situated somewhere where there are a lot of trolls.

You'd be forgiven though for not making use of in-theme postings, since the most vocal minority of our players will complain bitterly. Still, we're allowed to have fun as well!

14.4. Bookkeeping

A new area always causes problems, no matter how long it is in development and how thoroughly it is tested. The more people exploring an area, the more people there are to do things that you wouldn't have dreamed of in a million years. One of the most obscure features in Genua is that you can eat the hair off of various aristocrats, because that was something Taffyd wanted to do for reasons that escape me now. It struck me as funny, so it was allowed — but imagine that same somewhat bizarre approach spread over hundreds of players. Everyone will have different issues:

```
"I tried to chew the delicious chairs here, and I got a runtime."
```

Some of the problems will be technical in nature, but a lot will simply be because of the novelty of the area. As such, upon completing any development you should do what you can to minimise the burden on the liaisons who will be in the front-line fielding questions.

The first step in this is to introduce your development with the Five Dubyas — this is a post on the Liaison board covering five pieces of information:

Dubya	Information to give
What	What is the new thing that has been added, or what is the old thing that has been changed?
Who	Who was responsible for the development? (This is so liaisons know who to contact for resolution of queries, or to whom they should direct the undoubtedly lavish praise.)
When	When is the change scheduled to go live?
Why	Why was the new thing introduced, or why was the old thing changed?
Where	How do people get to it, if it's a new area?

This gives a quick overview of some of the most important questions for a liaison to be able to answer, but by itself it's not enough detail. Sadly, we are all better at doing this in theory than we are in practice, but you should also thoroughly document each part of the system that is likely to cause questions. For example, people aren't going to ask much about the shop we've provided because shops are a common feature across the Disc. They will, however, have questions about our library, and the quests. Those questions will be directed, in the main, to liaisons.

The information you write for this is Creator Only, so you can feel free to include useful functions, exact solutions, necessary bonus levels — all of that information is useful for a liaison trying to track down player problems. Ideally you should be doing this as you go along — if you are keeping to your plan, you'll already have a fair amount written up to draw from.

Where you choose to make the information available is up to you, but the most appropriate and easily-accessible place is most likely to be the creator wiki. Make a page for your project, and link to it from the relevant domain page.

As I say, we are all better at talking about this than doing it, but please make an effort to provide this information. It will make life so much easier for everyone — you won't have to keep answering the same questions for different liaisons, liaisons won't need to keep hassling you for information, and anyone caught in the crossfire (such as a lord without the good sense to be invisible) will be able to look up the answer to any queries without needing to really pay attention. Everyone wins!

In terms of the information you provide for players, you should also consider what hints you are going to make available through the quest hint system. This is something you should decide upon before you start coding, because it will inform your development if you know what information players are likely to possess.

Every room that has unique syntax should also come with a helpfile — this doesn't cover syntax for quests, but it does cover syntax for general functionality. The helpfile for our library, for example, should explain how the referencing system works and what the commands are to use it. The MUD's help system is quite powerful, and one of the most useful frameworks we have for reducing player frustration.

14.5. Lessons Learned

Without reflection, we are little more than apes. Part of your "wind up" process should be to compile, either mentally or more formally, a list of the lessons you have learned when putting together your development. These should be corrective rather than congratulatory, although if you feel you've done a good job you should indeed take pride in that. More important than your success, though, is where you feel you either failed to meet your requirements or struggled with a piece of code or concept.

These notes should be for your own personal use, but they can be invaluable in working out how best to support yourself in future. Make a note of all the things you found difficult, and all the things you got working but only after huge trial and error. Especially take note of things that work, but you're not sure you did the right way. All of these are tremendously valuable things to reflect on and ask people about — it's how you make the big leaps in your learning.

A lessons learned list might look something like this:

- Don't copy and paste add_items, use an inherit.
- Get the skeleton of the area working after you've written a placeholder inherit.

While these are for your own personal use, they can start to expand into a general philosophy for how you should go about developing projects. They're your Documented War Stories. Make a note of them, add the result of your reflection, and use that as the basis to make your next project even better and run even smoother!

As time goes by, you'll find that other people can start to learn from the lessons you have learned — it can serve as a useful resource for anyone starting out, and with a bit of editing you may even want to make the results of your reflection available to other creators. That's how you progress from being a Frightened and Young New Creator to being an old, grizzled and practically senile oldbie! For context, pretty much everything I have spoken about in this and related texts is the result of my own lessons learned.

14.6. Conclusion

There are certain things you need to do to “wrap up” your development and put it into the game as a finished product. You need to consider how your development is to be made accessible, and you need to make the necessary information available to liaisons. You should really be planning this as you plan the details of the area to ensure you don’t have a Horrible Writing Task left to do when you should be celebrating your achievement.

Additionally, you should spend some time reflecting on what went wrong, what went right, and more importantly why. This will serve as the basis for ensuring that the next project you develop is easier to get a handle on, and as time goes by you’ll find yourself internalising these lessons in such a way as to make project development as smooth and painless as possible.

15. Lessons Learned

15.1. Introduction

Now that we have extensively planned out our development, let's reflect on what we have learned! Sadly, we can't really do that because we haven't yet started to develop our area. Developing is when when we'll see where our plans will need to be trimmed, expanded and reined in. Instead, let's reflect on the lessons learned of other people who have been through this process before, and what they've taken away from the experience.

What you take away from these lessons is up to you — I would hope at least they'd make you think before taking certain decisions with your code.

15.2. Death March Projects

Death March projects are those that have killed creators — every domain has at least a few of these, some more than others. Either their scope is so ridiculously large that there's no realistic chance of it being completed in any reasonable space of time, or their feature-set is so complex that it is outwith the grasp of most creators. Death March Projects often overlap with feature creep projects because of the sheer throughput of creators, each adding to the plan and then leaving. The net result of adding a creator is that the end point of the development gets further away.

Hopefully your project won't be a death march project; but if it is, don't give up hope. Instead, learn from the lessons of those who have gone before you. Don't add to the project — instead, recast it entirely. Look through what code has been completed, and figure out the easiest way to take it from development to a finished project. This may mean closing off substantial chunks of it so it can be developed and released incrementally. Breaking a project off into three or four subprojects that can be developed independently and then released is the easiest way to turn a Death March Project into a project for which you are the celebrated hero who conquered it.

The problem with such projects is that the plans are always very cool. You read whatever design documents there were, maybe view the board archives, and read the wiki. "Wow!" you think, "This will be awesome!" But it won't be, because you'll be crushed like everyone else who tried to get a handle on it. Make a realistic assessment of how long it'll take to bring the project into playtesting, and then ask yourself "Do I want to be spending that much time before I see any code in the game?" If the answer is no, then be aggressive — cut everything that is peripheral to the main theme of the development. Check with your domain administration while doing this — they may have opinions on which features are entirely expendable.

Sometimes you'll find that the code for your Death March Project is actually impossible to salvage — you're left with a horribly ambitious plan and no code! In such cases, with the blessing of your supervisor and the relevant domain leader, you should simply start over. Don't feel as if you are bound to the plans of creators who have gone before — they had their chance.

A Death March Project is a challenge, but one that everyone is capable of meeting if they tackle it appropriately. Usually the projects are defining features for a domain — something so integral in the books that domain leader after domain leader has felt compelled to continue development. As such, you're usually getting a pretty sweet brief for your project — it's just the cruft you are inheriting that is the problem. Just remember that a development plan is not a binding contract: you are allowed to deviate from it, and scrap it entirely if it's unworkable.

15.3. Realism Over Fun

Ho boy, are we all guilty of this one. At one point or another almost everyone has said something like "I'm going to do this because it's more realistic than this other way of doing it." This is a warning sign — realism should never be the reason why you implement or change something. We're game developers, and Discworld is not a life simulator.

If you want to make a change, make it on the basis of gameplay, or balance considerations. If your sole answer to being asked "Why do it that way?" is a variation on the theme of realism, step back, take a deep breath, and don't do it. Realism is largely a code-word for "make things fiddly", and we gain no extra game from that.

There are many things that are realistic, but in place for a different reason. Making it difficult to drink potions in combat was a change made to counterbalance the overpowered impact of healing tea — the fact it was a more realistic change was simply an added benefit.

Consider the streamlined mechanics of something like Warcraft with regards to, say, swimming, in comparison to those in Discworld. In Warcraft, you don't drown and lose all your stuff. In Discworld, that's entirely possible. One is the emphasis of playability over realism, the other is the reverse.[‡‡] It has sometimes been argued that the terrains are our biggest example of realism over playability — they make the MUD much, much larger, but conversely they can also make it feel much, much emptier.

Realism should always come as an added benefit for a change to gameplay, not as a reason in and of itself. Luckily so far we have managed to escape the appeal of compulsory eating and drinking, resting, and bathroom breaks! Every so often though, someone brings them up...

Remember what alchemical process we are here to facilitate — time goes in, fun comes out.

‡‡ Now, it may be argued that the risk of drowning can be useful for making it harder to access certain areas that shouldn't be too easy to get to, as well as in adding risk for players who enjoy that sort of thing. Those are good reasons in themselves — much better than "realism".

15.4. Randomness Is Not Challenging

Every now and again someone develops something where survival or success is based on the roll of a dice. You either get lucky or you don't... that's just the way of it. While there are a few self-contained games in which chance is the sole driver for winning and losing, they are very much the minority. Randomness does not offer the potential for interesting game decisions.

One example of randomness in the game is our T-Shop. There is no way to find the T-Shop, no way to systematically hunt it out or draw it to you. It randomly appears in random locations — if you want it, you have to wait at one of these locations and hope you get lucky. It might never come to you — you may spend an entire session waiting for the damn thing and never have it appear. Some feel that this makes finding the T-Shop a "challenge", but this is not an interesting sort of challenge — it's just a dice roll.

A *real* challenge is something against which you can pit your own skills and abilities — while randomness is often a factor in this, it should never be the sole determiner. Careful consideration and planning should allow you to change the odds in your favour. To make something like the T-Shop challenging, it needs to be responsive to strategising. For example, if it arrived at the room with the least number of people waiting, then you would have a chance to influence the system — it would be tremendously hard, but it would indeed be a challenge.

15.5. Complexity is not King

The principle of "Keep it Simple, Stupid" is substantially true for Discworld. Many people confuse complexity with quality, and think that providing complicated, intricate systems is the best way to implement new game features. It's possible to have depth of expression even with simple systems — indeed, a good design goal is to make systems as simple as possible, but inter-related. That leads to emergent game behaviour which is always more interesting than simple complexity.

Take the crime system in Genua as an example — it's based on tremendously simple rules, but the impact of those rules means that it actually creates gameplay options rather than restricts them. Emergent gameplay decisions mean that it becomes strategically important as to what witnesses are around, what time of day it is, and how busy a street is. The result of this is that if you want to kill someone safely, you drag them down an alleyway to do it. Not everyone appreciates this as a gameplay system, of course, but I have always found it compelling when I play in Genua.

When adding a substantial new feature, you're not looking to implement it realistically. Instead, you want the *essence* of the activity — enough that you can genuinely feel part of the fun, but not so much that you are overwhelmed with tedious micromanagement.

Look at how we are implementing our library — a single research command that abstracts the task. We could just as easily have the player check each book in turn, saying "You don't find anything about tedium in that book" until they find something interesting, but that's not fun — it's micromanagement.

If you're adding systems into the game, consider how they could be streamlined — lose the jagged edges, and keep the core. You can always add complexity later, but it's not always so easy to remove it.

15.6. Complaints Are Not Representative

If you handed everyone in the world a bag of money, you'd get people complaining they had nowhere to put it. It's important to realise you are not going to please everyone, and despite the effort you have put into your development, people are going to be unkind about it. There's not much I can say about that, you just have to take it in stride.

The danger comes in when you take the complaints at face value. Complaints are not generated according to a vote amongst all players, and they certainly aren't quorate. There is a (probably apocryphal) story about President Harry Truman, who, growing frustrated by his growing pile of hate-mail, turned to his wife and said "Why is it only sons of bitches know how to lick a stamp?"

It's important not to think that complaints are representative — usually they are not, they are usually from the vocal minority rather than the silent majority. That doesn't mean you should ignore them, but you certainly shouldn't ascribe any great truth value to them unless they are widely corroborated. Not being able to contextualise complaints is likely to grind you down more quickly than anything else. You will find, despite your being a volunteer developer on a completely free gaming environment, that some people will still not appreciate the effort you have put into the game. These are often the professional malcontents who derive most of their joy in the game from putting down others. In cases where you feel that you are being unfairly criticised, you can always remind people of our guarantee — if not completely satisfied, we will refund all of the money they paid us.

15.7. Know When To Step Back

On encountering an intractable problem, there are two courses of action:

- Bang your head against it until you solve it.
- Go away and come back.

You should know that the latter of these is not the "cheating option". Sometimes it takes a while for your subconscious mind to tick over and provide you with the solution. I can't count the number of times I've had a flash of inspiration in the shower, while driving, or just sitting and reading something else. Your mind is always working away, and if you temporarily stop focusing on a problem, you can view it with a fresh eye when you come back to it.

Don't feel that you have to keep struggling until you find the answer. Know when to take a step back and let the answer come to you naturally. You'll be happier, you'll be less stressed, and you'll enjoy yourself more. Which brings us to the most important lesson that everyone should learn...

15.8. For God’s Sake, Have Fun

Don’t take it all too seriously — you didn’t sign up as a creator to not have fun, you signed up to experience fun from a different perspective. Sure, there are lots of rules and guidelines and huge amounts of stuff to learn, but don’t lose perspective — it's all about having fun. There’s a lot of fun to be had from bringing something new into the game and having people enjoy it, and we sometimes lose sight of that when getting frustrated with development setbacks. It’s all part of the learning experience.

In software engineering, there is a stereotype of the “guy in the room” — tremendously hard working and dedicated, but not actually part of the team. Don't be that guy (or gal). We do suggest quite strongly that you keep your head down for a while until you get a feel for the social environment, but once you feel you have a handle on how creator conversations actually work, then join in! The social context in Discworld is one of the most rewarding parts of being a creator — there are people on Discworld who have become very real and genuine friends of mine, and one of the things that keeps most of us here are the people we know. You’re part of that community, and much of your support and enjoyment will come from participation.

Don't forget to have fun — if you’re feeling unproductive, play for a bit — you don’t want to lose touch with the game. If one part of your development isn't going the way you want it to, change tack for a while to something more fun.

If you remain stressed, unproductive, or just generally fatigued, then take that for the warning sign it is. It may be a hint that you need to step back from the MUD for a while — talk to your supervisor if you think this is the case. Every one of us knows that Real Life Comes First, and we’d all much rather someone took a break than burnt out. If you’re just not having fun here, then something is wrong and it needs to be addressed.

The reason we are all here is to have fun. That’s the most important lesson you can take away with you from this chapter.

15.9. Conclusion

We’ve reached the end of our journey, save for some concluding remarks in the next chapter. I hope you found it as enjoyable to read as I did to write. Ideally you found it more enjoyable!

Life is about learning — everything you do provides you with a lesson, and coding on Discworld is no different. You should be mindful of the lessons you can learn from your development, because they're all part of what we refer to as “experience”. That’s all experience is, after all — it’s what you learned from the mistakes you made.

16. Final Thoughts

16.1. Introduction

Cor, do you remember when we were wide-eyed and innocent, knowing only that we were going to develop something but with no firm plans as to what? Now look at us — we have a firm handle on exactly what shape our development is going to take, and all that's left for us to do is actually code it. Still, as a wise man once said “Programming is the last mile of game development”. While a lot of people plan as they code, there is much to be said for simply sitting down and thinking through what it is you’re going to do.

In this chapter we’re going to talk about where you go next with your training.

16.2. What Happens Now?

Certainly your next step is easy — you progress on to *LPC For Dummies 2*, in which we’ll actually start coding all of the stuff we’ve been planning. It’s quite a complex development involving handlers, inherits and quests, and so by the end of that particular set of material you’ll find yourself more than capable of handling most of the tasks that life as a creator may throw at you.

You’re probably going to have a project of your own you will be working on, and so you’ll be able to apply all the stuff we’ve talked about to designing out your own development. This will teach you even more — it’s one thing to follow through some worked examples when someone else is doing the thinking, but it’s another thing entirely to put your own mind to such a plan.

We’ve talked a lot about game design in terms of how it specifically relates to the MUD, because as an environment it throws up a lot of interesting challenges that aren’t present in other such games. All of us as creators have been learning it all as we go along, and other people may very well hold different views on these topics than I do. You should talk to other people about the things in this material to get a wider perspective than just mine.

16.3. Keeping Our Players Happy

Game development is a process of judgments. Not everyone is going to agree with your choices, and not every player is going to find what you offer compelling. That’s okay — as long as we provide something for everyone as we go along, we don’t have to make everyone happy within a single development. If we did, our task would rapidly become impossible.

At the core, though, what we're looking to do is improve the efficiency of a simple process. Our players invest time, we invest time, and what comes out at the end is fun for both. Sometimes we need to reduce the fun of one group of players to enhance the fun of a larger group, or to ensure that fun is had for a longer period of time. All of the most unpopular "balancing" decisions that have been taken over the years have been in service of this overall goal. While their effectiveness can be argued, what we should remember at all times is that each and every one of us is committed to making the game better. We may disagree on how that should be done, but we can all agree that everyone has the best intentions.

One of the things that being a creator does open your eyes to is how hard it actually is to do the kind of stuff that is so cavalierly thrown around on the boards. It gives you a sense of perspective that a player simply cannot have, because they are not part of the designer equation in our Grand Formula. Balance decisions are tremendously intricate, and they reverberate far beyond the direct change itself.

While it is unlikely that your first few developments will have Mud Changing Aftershocks, it is better that you start thinking in those terms sooner rather than later. You should never lose touch with the game itself, but being a creator means being willing to make decisions that will directly disadvantage your player character, in the hope that the overall satisfaction of the playerbase rises.

16.4. Further Reading

Game design is a complex subject, and one in which the medium is sufficiently new that no formalised dogma exists. Instead, we have the views and opinions of many people to draw upon in our quest to design a better game. I would recommend the following to anyone interested in expanding their understanding of the topic:[§§]

Title	Author
A Theory of Fun for Game Design	Raph Koster
Swords and Circuitry	Neal and Jana Hallford
Rules of Play: Game Design Fundamentals	Katie Salen
The Video Game Theory Reader	Mark Wolf and Bernard Perron (Eds.)
Character Development and Storytelling for Games	Lee Sheldon
Designing Virtual Worlds	Richard Bartle
My Tiny Life	Julian Dibbel

Some of these are quite difficult to buy, so you may have to hunt around for them at libraries and other sources.

§§ The lists of books and articles here were compiled pre-2010 and hence do not contain any more recent material. If you have suggestions for additions, please speak to the leader or deputy of the Developers domain.

There are also a few articles that are worth checking out:

- Zork: A Computerized Fantasy Simulation Game
- Hearts, Clubs, Diamonds, Spades: Players Who Suit MUDs
- A Rape In Cyberspace — This was the start of what would become the book *My Tiny Life*
- Quests in Context: A Comparative Analysis of Discworld and World of Warcraft — This one is poorly researched (and we're in a position to know since a lot of it is about us), but still worth reading purely because, well, it's about us
- Figuring the Riddles of Adventure Games
- Various Articles by Discworld Creators
- Imaginary Realities Archive

16.5. Conclusion

That's it! Our business is concluded for now!

Hopefully you will have found this at least a little bit helpful in managing the complexity that comes along with developing an area in our MUD. You have many constraints to work within — your own confidence with code, the thematic considerations, and the general principles of game design. You're going to make mistakes as you go along, because everyone does — the important thing is that you learn from them. Have fun!