Level Design Structure and Methodology

4

A level's structure and methodology are important early choices for a level designer, and they can have a large impact on the actual creation of the levels. We will look consecutively at structure and methodology, because the choices of the former impact the likely choices for the latter.

Structure

Once we have decided on the type of content a level requires, and where it exists within the hierarchy of the game, we need to determine which *structure* to apply to the level. In most cases this will be determined as part of the game design, since that determines structure as a whole across the game. Nonetheless, it is often the case that at least some choices are left to the level designer, at least on a smaller scale—for example, within the levels themselves. This kind of structure is one of gameplay flow, which is very much in the hands of the level designer.

Typically there are three main approaches to choose from in most game types: *linear*, *semi-linear* and *non-linear*. Sometimes the distinction is not clear-cut, and hybrids may occur. For example, a level may be divided between content that is 60% linear and 40% non-linear. Ultimately one of the most important determining factors of structure is a game's genre. A classic shoot-em-up is much more likely to follow a linear structure than a freeform RPG. But even within these genres, there is scope for differing progression models.

Let's look at the distinction between linear, semi-linear and non-linear in more detail.

Linear Levels

Linear level design, as the name implies, is level design where the gameplay events follow a strict *line* laid out for the player to follow.[1] Sometimes this is referred to by saying that the gameplay is *on rails.* Events unfold across this line in a strict order that the player cannot deviate from. Progression through the level is only possible if the player goes through the gameplay events in the order predetermined by the designer. To look at a diagram of such a level structure, we can envision something like the structure in Figure 4.1.

Although pure linearity is becoming less prevalent than it used to be, it still has a place in video games. There are a number of advantages to the technique. If they are appropriate to the gameplay needed for the game design, linear levels can work well.

Figure 4.1. Linear level structure.

Advantages

The most important advantage to linear level design is the amount of control it gives a level designer over the play experience. It is much easier to carefully design the experience of playing the level if the designer can determine the order in which gameplay events occur. Since this type of overall control allows the level designer to determine matters of pacing, consistency, story development, learning curve, and many others, it is as close as a level designer can get to *directorial* control. And in keeping with the analogy between directing a film, it can be compared to being allowed *final cut* on a movie.

Disadvantages

The main disadvantage of linear level structure lies in the danger that it can make players feel constrained in their gameplay freedom, possibly *arbitrarily* so. The designer needs to make sure that the roller coaster ride is an enjoyable one, as the player is not allowed to get off and find his or her own fun. If this is not done correctly, there is a possibility that the player will start to *resent the game,* which is of course to be avoided.

[1] Generally this is done in advance by the level designer.

Furthermore, if things start happening *regardless of the player's own actions*, a perception of futility may be created. And since the linear structure of a level is not the most flexible one, it is fairly common for certain gameplay events to be forced onto the player. This can be a dangerous approach. If the player's actions don't matter, than why act at all? If this is allowed to get out of hand, the experience of the entire game will be tarnished or even ruined.

Implementation Strategies

In linear level design it is vital that the designer has a good understanding of *pacing* and *play psychology*.[2] Most players will accept this kind of directed experience as long as they are directed with a sure and steady hand and the gameplay is always rewarding or interesting. If the player does not have time or inclination to *question the direction*, the level designer is successful. Unfair gameplay challenges, or dreary boring stretches, must be avoided if possible, as the player does not have a choice in engaging with these gameplay sections. Since these events are forced upon the player, they have to be worthwhile for them not to grate.

Some Typical Examples

There is no hard rule on when to choose linearity in level design, although there are times when it best supports the game's structures or general internal goals. For example, some puzzle games work best when the next puzzle will only be presented if the previous one has been solved. *Tetris* would not be very compelling if it were a free-roaming affair where the player would not feel the pressure that its linear approach enables. There are also story-driven games that rely on the progression of the story in a strictly linear fashion. These kinds of games may be becoming less fashionable, but if expected to create levels for them, we have to be prepared to rise to the challenge.

Semi-Linear Levels

As a compromise between linear and non-linear gameplay the level designer can opt for a hybrid form. Semi-linear gameplay allows players to direct their own experience in some instances, but it requires that players follow a script in other instances. This can be done by a system where players can perform a certain number of gameplay tasks or follow a number of paths of their own choosing, but eventually are led to a bottleneck. This bottleneck can be a *physical* one such

[2] All covered in great detail throughout the book.

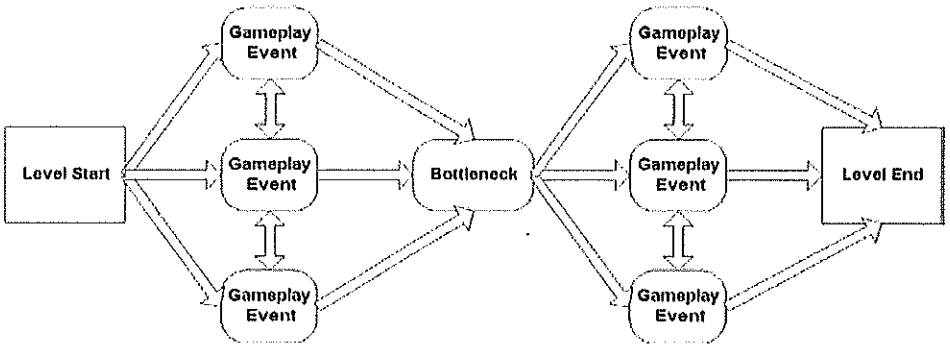

Figure 4.2. Linear level structure.

as progress to a next area via a single door, or it can be a *conditional event.* The event can be anything from reaching a certain number of experience points to hitting a time limit or having collected a certain amount of items. There are plenty of possibilities available within most game genres. (See Figure 4.2.)

Semi-linear gameplay progression is widespread in games, as it fits many game types and allows sufficient control over the experience.

Advantages

Semi-linearity, if used correctly, can offer the best of both worlds. It gives the level designer a reasonable amount of directorial control over the events the player will experience, but leaves enough freedom in the hands of players for them to feel they are authoring their own experience.

Furthermore, adroit level designers will do their best to create the *illusion* of full player freedom, to maximize players' involvement in the experience and deepen their immersion. This is an important aspect of level design and one that will be further developed in several chapters later in the book.

Disadvantages

On the other hand, semi-linearity can represent the worst of both worlds. If applied in a lazy manner, it can arbitrarily lead players by the nose when uninspired, while leaving them to their own devices at inappropriate times. When a level designer starts to give the player *some freedom,* more freedom will be expected throughout the level. If they are allowed to perform certain gameplay actions at one point in the game, it is only natural that the players expect this to be possible at other points. If the level designer does not carefully think this

through, a logically inconsistent world can be the painful end result. This can completely undermine the expectations the player has of a level and can destroy trust in the fairness of a game. At least with strict linearity, players know what they can expect.

Implementation Strategies

It is important in this kind of level design to observe consistency. Players will be acutely aware of when they are treated unfairly, and because of this, the level designer should make sure that restrictions or player direction does not feel too arbitrary.

Where possible, techniques should be used that hide as much of these restrictions as possible, and that successfully create an illusion of player freedom and choice. If players have limited progression choices, but are made to believe they have many, the best of both worlds has been achieved. The level designer has in that case invisibly guided the player's experience.

Application

The semi-linear approach is probably the most popular one in level design circles. It avoids the limiting constrictions of pure linearity and steers clear of the logistical problems of freeform gameplay. Multiple play styles can be incorporated so the levels cater to a wider variety of game players, contingencies can be included for when players find it hard to progress, and the world can be made to feel more responsive to players' choices, even if in fact the choices are limited.

These are serious advantages, and they make it easy to see why so many levels follow this kind of approach.

Non-Linear Levels

In a *non-linear level,* the order of gameplay actions is mostly left to the player. This is one of the reasons why non-linearity is linked to *sandbox design* and to *emergent* gameplay.

True non-linear gameplay is very rare indeed, but some games and game types come close to it or feature moments where it does occur. Often, this type of gameplay goes hand in hand with interactivity; in a game where it is viable to create one's own gameplay, a high level of interactivity can be very helpful. A good example of this is games where a physics system allows players to manipulate their environment on their own terms, but within the restraints of the physics system, thus leading to non-linear and non-scripted gameplay events.

The clearest example of non-linear gameplay lies within the realm of some multiplayer games. Since it is the other players, and not the level designer, who provide the bulk of gameplay, multiplayer games can be highly non-linear.

Advantages

Because the players themselves exert maximum control over the gameplay experience, they are likely to take responsibility for their own failures. Since the level designer does not directly dictate the progression choices, players can experience a sense of ownership over gameplay. This is generally perceived as a positive outcome.

A game that allows players to manufacture their own gameplay experience produces an added bonus of *free* content. After all, if *automatic* generation of gameplay occurs, it does not require work from the level designer. It is gameplay content created by somebody else, which in the busy schedule of professional level design is a very good development.

Disadvantages

A non-linear gameplay world is much harder to test and is therefore harder to design in a robust way. It may be impossible to test every single permutation of player action that may occur, which is inherently dangerous. Unforeseen player actions or tactics may "break" the game in unforeseen ways, possibly by allowing some players to dominate others in an unacceptable way, or by finding loopholes in gameplay logic that allow the player too much power.

Requirements

In non-linear levels, the level designer has to make sure that players have enough tools to play with the game world. A requirement for non linear gameplay is that players must be allowed to write their own gameplay story. To do this, a certain amount of interactivity is required. There is a need for deep and engaging gameplay resulting from the player's own actions, rather than prefabricated scenarios. Where possible, the conditions for this must be provided by the level designer.[3] This can take the shape of strategic depth, for example in layout design in multiplayer games, or of purely physics-based interaction in single-player games where the player treats the world as a toy.

---

[3] Also, in this case, extra responsibility is laid on the shoulders of the *game designer* to allow for this. The level designer can only implement what he or she is given in the game design.

Methodologies

Earlier in the book I mentioned that this text is not meant to be about level *construction*; instead, it is focused on level *design*. But this does not mean that we should not look at *methodology*. When eventually faced with technical construction issues, it is good to have spent time researching a number of methodologies. This allows us to structure a level well in advance of level creation and gives us a chance to prototype on paper. Potential problems can be found early, and the level creation process can be scheduled easier.

This part of the book will not attempt to be a detailed listing of all possible level design methods, but will rather focus on a number of *general methodologies* that may provide hints on how to proceed in other unique cases. The main goal is to give examples of various approaches that a level designer can adopt, depending on the situation he or she is in. It is up to the designer to determine what to include in the level design documentation.

After having looked at structure, methodologies are a logical next area to look at. A number of useful methodology types are shown below.

Annotated Maps

Most people are familiar with the concept of *annotated maps*. This is a method widely used throughout the game industry, where the level design is described through a gameplay breakdown of all the physical spaces in a level. This is often done by creating a map on paper, providing a legend and numbers that indicate the "rooms" or other gameplay areas. This can be extremely effective, as it allows not only the level designer, but also other team members, to retrieve useful data from the level design document.

Example of Annotated Level Design: *Stolen*, Level 4

Annotated maps are commonplace in professional level design and is an approach I have had to take myself many times. To show how this may look, the following example details a section of a level heavily weighted for stealth gameplay, made for the game *Stolen*.[4] The game's player character, Anya, needs to traverse a heavily guarded area without being spotted, as shown in Figure 4.3.

[4] Developed by Blue52.

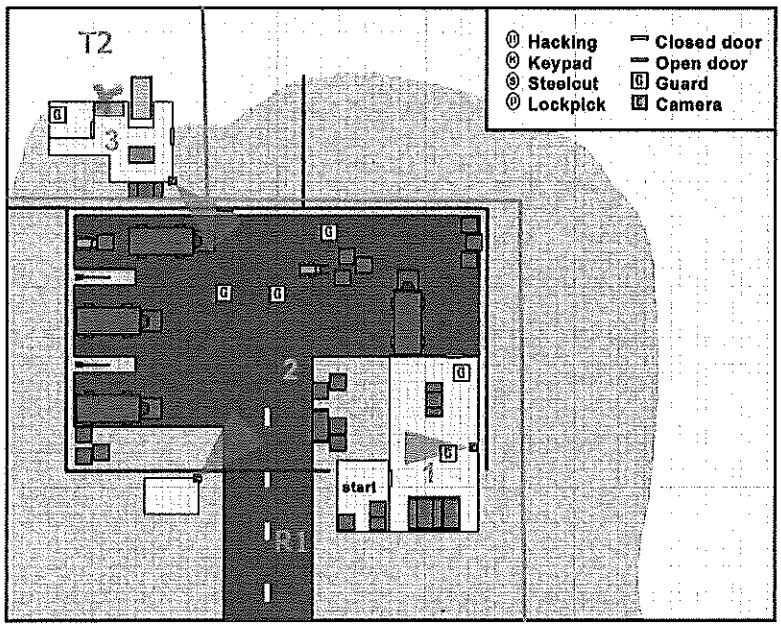

Figure 4.3. Level design detail: Satellite Array Level. (Source: *Stolen*, developed by Blue52.)

Here are some annotations for the map:

02 Lower compound (links to 3 and 1)

Anya will be in an outside area where she can clearly see the building that houses the cable car docking bay.

The player has to approach the cable car dock without being spotted, using stealth mechanics. A route is available via the roofs of several trucks and a number of poles but the final jump cannot be made.

In order to finalize the route the player needs to get near a forklift truck and raise its forks that are holding a pallet. After this the player will be able to get on the roof of the docking bay, and enter through a vent

Several guards patrol in key positions.

Anya can get to the main entrance of the docking by evading the guards and sticking to the shadows, but there will be no way in from the ground floor.

An alternative route takes her around the edge of the compound. She has to evade the guards patrolling and somehow get past a guard at the checkpoint of the area.

03 Docking bay

The player finds the cable car inside, but it is not reachable from the ground. A little tower/building gives access to it but is locked. A gantry connects with the cable car but is too high to be reached. This is the destination in this area.

A ledge route along the walls of the room leads the player to a vent system high up, and can be reached by a ledge jump. Eventually the player lands on top of gantry that triggers a real-time cutscene.

The guards holding her equipment leave the building to board the cable car. Anya quickly jumps on top of it to stay out of sight. The cable car starts to move up towards the mountain.

As you can see, the method can become quite involved. A very large amount of gameplay information is recorded in a useful manner. Other people than the level designers can also benefit. Take the following addition of information for game area 02. It is meant to provide notes on required game assets or other related information useful to other members of the development team:

Special Assets Needed/ Notes:

Code

Interactive Objects/Other:
Forklift truck needs to have operable forklift.
Two different collision states are needed.

Art

General:
Forklift model.
Special:
None

Production

Sound:
Forklift engine.
Forklift raising its forks.
Story Events/Cutscences:
Forklift raising its forks.

Concept art

Even more information can be added to the level design. For example, concept art can be very useful to provide the reader with a better visual guide of the environment and its content: see, for example, Figure 4.4.

All of this builds a picture of the final shape of the level design even before construction begins.

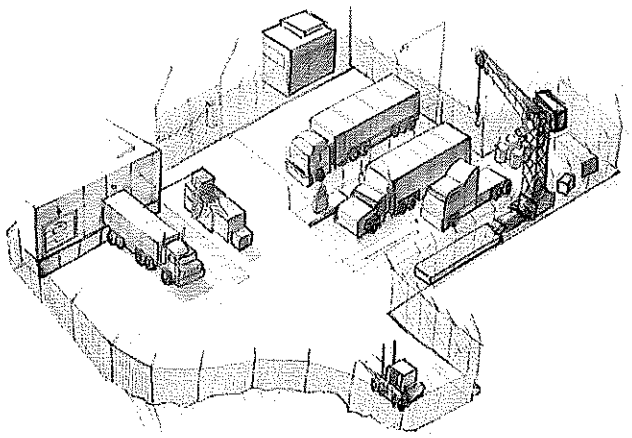

Figure 4.4. Concept art. (Source: *Stolen* Level Design, Blue52.)

Dangers

When so much information is collected into a limited amount of documents, there is a real danger that the resulting level design may become too cumbersome. A monolithic, unwieldy, and inflexible level design is problematic for all parties involved. It is hard to keep up to date and may provide too much information for some, but too little for others. If a level design document like this *is* appropriate, it still may require further explanatory documents or visual support, some of which will be shown below.

Flow Charts

To combat some of the clunkiness of annotated maps, we can express level design progression through flow charts. There are many positives attached to this method. They can be created and modified quickly, which is very important if the level design has to be flexible. They are much better able to express conditional data and, in doing so, provide a logic check for the level designers. (This can also be very important if any amount of scripting is involved.).

Here is an example of conditional level design data expressed through the flowchart in Figure 4.5:

> A player enters a room and activates a floor switch that locks the room and releases two guards. Only when the two guards have been defeated

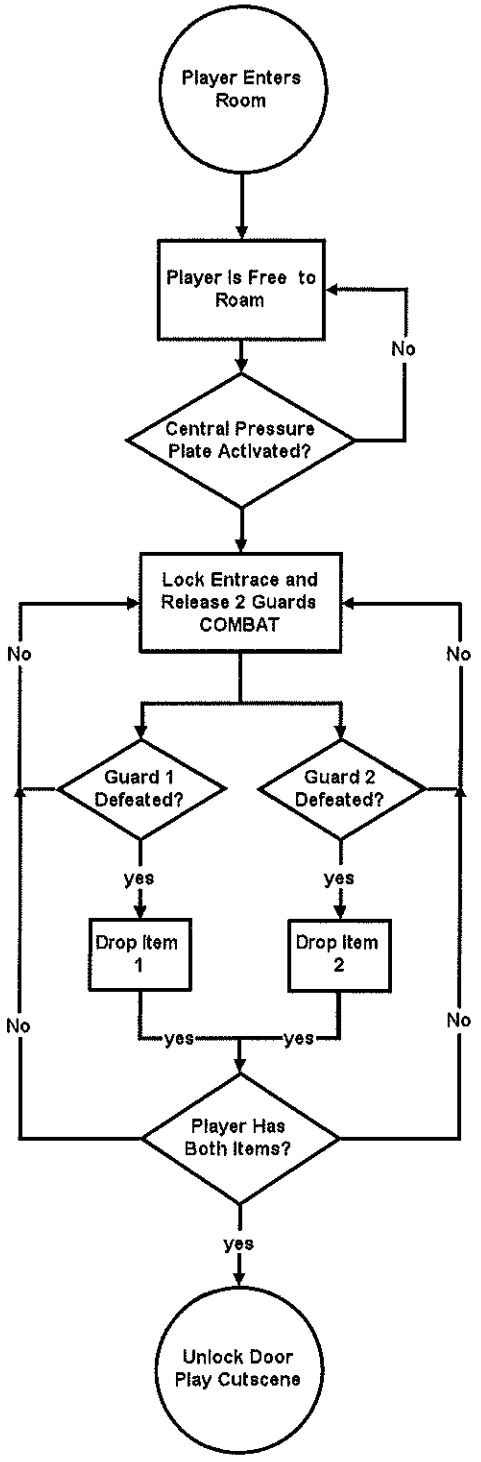

Figure 4.5. Conditional combat situation.

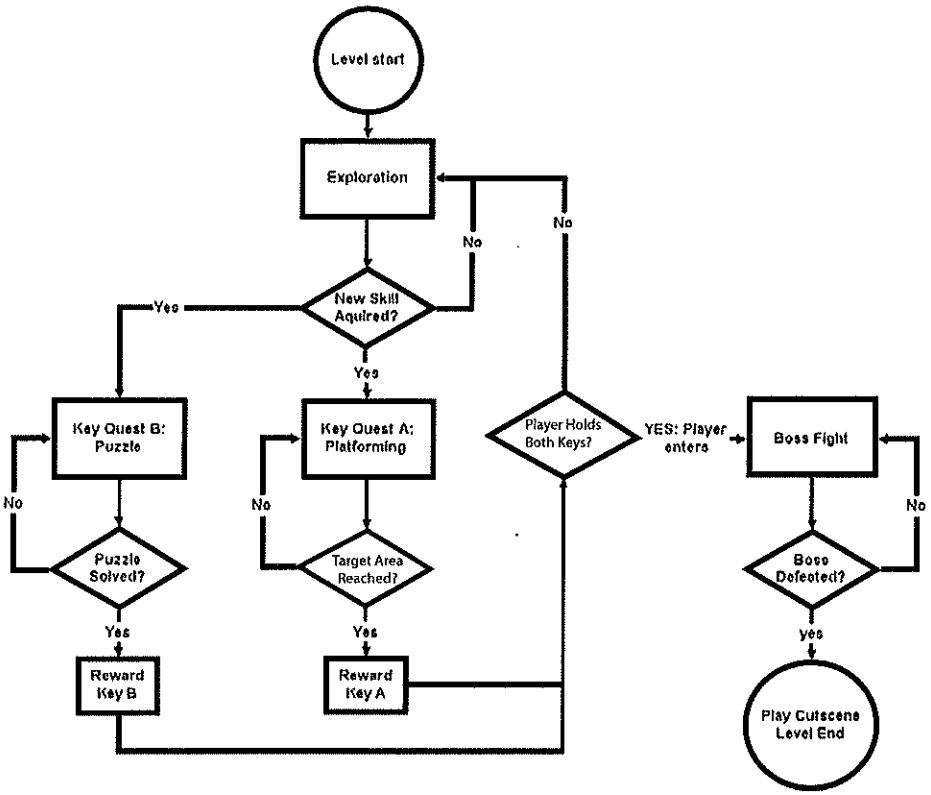

Figure 4.6. Level flowchart.

and two objects they were carrying have been collected[5] can the player leave the room. A cut scene is played to signify this.

This kind of information is hard to capture in annotated maps, no matter how complete the annotation is. Flowcharts are very good at capturing specific information and showing *how it relates to other elements*. This makes them very useful indeed to create an easy-to-read and coherent overview.

Figure 4.6 shows a flowchart applied to a whole level. Note how this kind of flowchart shows progression by providing questions and answers that signify gameplay or game states.

[5] I am sure you can figure out why this is so.

Dangers

The large degree of concision and logical overview comes at a price. There are several dangers and problems that may occur when using his method. Out of many, here are some typical examples:

- There is no information on the level's environment.
- Emotional impact is not represented.
- There is no temporal data (outside of chronology).
- There is no room for unexpected results.[6]

These and other dangers make it doubtful that a flowchart can express a level design fully. (But I have personally seen people attempting to do just that.) However, this is an extremely powerful method to be used in conjunction with other methods. The use of flowcharts, for example, is very complementary to annotated maps, described earlier.

Level Arc: Freytag's Pyramid

In 1863, Gustav Freytag published an important book on dramatic structure called *Die Technik des Dramas*. The book aimed to provide an overview of dramatic components as seen in classic plays, namely Greek tragedies and Shakespearian drama. Freytag argued that all these components follow a specific order, one component leading to the next one. The well-known visualization of this underlying structure is called *Freytag's Pyramid*. (See Figure 4.7.)

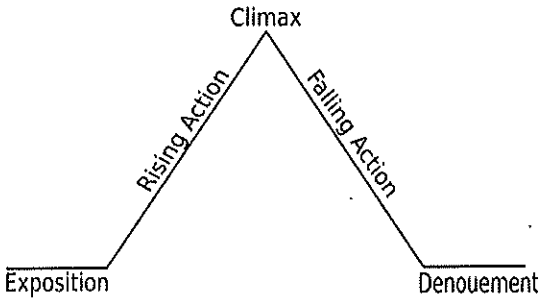

Figure 4.7. Freytag's pyramid.[7]

[6] This can also be an advantage.

[7] "Freytag's Pyramid," *Wikipedia*, http://en.wikipedia.org/wiki/Dramatic_structure, 2009.

Freytag's Five-Act Progression Breakdown

As Figure 4.7 shows, Freytag divides drama into a number of consecutive steps: *exposition, rising action, climax, falling action,* and *dénouement* or *catastrophe,* each represented as an act within a play.

Act 1: Exposition

In Act 1, the basic story elements are introduced for the first time. We learn enough about the setting and about some of the characters to form a framework in which the drama takes place. This exposition ends with an *inciting moment.* The inciting moment creates the catalyst necessary to allow the unfolding of an interesting drama. Without this, there would be no reason to tell the story to begin with. It often takes the form of a conflict of some kind.

Act 2: Rising action

After the story has been kick-started by a central inciting moment, further depth and drama is added during Act 2, *rising action.* This is done by introducing further layers of drama (possibly including new conflicts, hindrances, and obstacles to the protagonist's goals) and by introducing new characters and other complications.

Act 3: Climax (turning point)

The rising action leads naturally to a climactic moment. This is a turning point to the story: a key moment where a change occurs that is fundamental to the story. This change can be for better or for worse, depending on the type of play. Typically, in a comedy the change is for the better, while in a tragedy the opposite occurs.

Act 4: Falling action

The conflict between the protagonist and antagonist reaches a critical stage during the *falling action* part of the play. The drama will progress towards a resolution that serves either protagonist or antagonist. The final direction of this resolution can be kept ambiguous for a certain amount of time to create *suspense.*

Act 5: Dénouement or catastrophe

The final resolution of the play comes in the form of a positive conclusion (*dénouement*) in case of a comedy, or in the form of a negative ending in the shape of a *catastrophe.* Both conclusions show a progression in the experience of the protagonist. In a dénouement, the protagonist is better off than at the beginning

of the play, while a catastrophe leads to the opposite conclusion. Either way, the drama gets resolved.

Five-Act Progressions in Level Design

Nobody suggests that the above description of a five-act classical play or drama is representative of all plays. It is merely a description or analysis of a particular kind of drama that has often been used in the past. Judging by how many plays have followed this progression, it must have been very popular with a great many writers. Perhaps this is because it creates a recognizable format in which to tell the story: a frame of reference that an audience can relate to. In truth, the five-acter is still very much alive and is used throughout the arts, not in the least in Hollywood cinema.

If we extrapolate from this, it may be possible to identify similar principles within level design. Let's take Bungie's *Halo* as a test case:

Halo as a five-act level design progression.

Halo is a good subject for this experiment, because despite a certain amount of freedom in the way the game can be played, the story progression through the game is fairly linear. In other words, many of the methods are up to the player, but the gameplay goals are predetermined.

Act 1: Exposition

In Act 1, we are introduced to the protagonist, Master Chief. Through exposition, we learn much about the background to the story and the story's environment. We learn that the protagonist is a soldier and that the current action takes place on board a space ship. The inciting moment that functions as a catalyst that escalates the story takes the form of an alien attack on the space ship.

Act 2: Rising action

Act 2 serves as a vehicle for introducing elements to the game that deepen the experience. The player has landed on alien structure *Halo* and has to learn how to combat the resident hostile alien forces. In the meantime, we learn more about the background story of the conflict.

Act 3: Climax (turning point)

In the climax of *Halo,* the player learns that a far greater danger than the alien protagonist exists. It is a viral life form known as the *Flood.* It infects and takes

over other life forms in such a way that the Flood ends up corrupting and controlling its victims. The Flood have started to infect both the aliens and the humans, and their threat lies in infecting entire planets.

Act 4: Falling action

Much of Act 4 is defined by the player's epic struggle to contain the threat of the Flood.

Act 5: Dénouement and catastrophe

In the final act, we already know that Master Chief has been successful in his main goal, stopping the Flood from spreading by destroying *Halo*. The final outcome of the story centers on the personal survival of the hero.

Problems Associated with Freytag's Pyramid

Although there are many uses to the pyramid, and there are many examples of dramatic creations that fit its definition, there are a number of criticisms that can be aimed at Freytag's pyramid that are worth investigating.

Oversimplification

It can be argued that to create such a neat overview, the subject must be oversimplified beyond the point of being useful. One could argue in the case of *Halo* that the story features several turning points. Should we ignore some of them in order to point at one specific climax? If we don't, does it mean that Act2 sometimes appears after Act 3?

Restriction

Restriction is the enemy of creativity. At least, arbitrary restriction is. What if we want to start with a catastrophe and a climax, and through exposition and falling action slowly tell a compelling story? We may want to write a comedy with a fatal ending or a tragedy with a happy ending.

All of these criticisms are valid to a point. As a device for forming an interesting dramatic construction, the pyramid *is* hopelessly restrictive and oversimplified. But the pyramid's value is not one of dramatic creation; rather, it is a tool for *dramatic visualization and analysis*. After all, Freytag intended to analyze and show the dramatic structure of classical plays. Considering the ubiquity of his work to this day, he was very successful in achieving this goal. There is no need to slavishly follow this particular

method, but it can inspire us to devise our own methodology to suit our own needs.

Applications for Level Design

If we extrapolate from this, we should be able to devise similar techniques that successfully aid us in level design analysis and visualization. This is something that in the hectic and often stressful reality of professional level design can be of immense benefit.[8] There are several techniques and approaches available to level designers, but the ones most commonly used are based on *event diagrams*.

Event Diagrams

An *event diagram*[9] is a graphical representation of the content of a level, potentially including representations ranging from emotional impact to duration, or combinations of several factors. There are many elements that can be included in these diagrams, but typically they will show one of the following things:

- type of event,
- duration of event,
- chronology (if applicable),
- impact (color-coded).

Take the example in Figure 4.8.

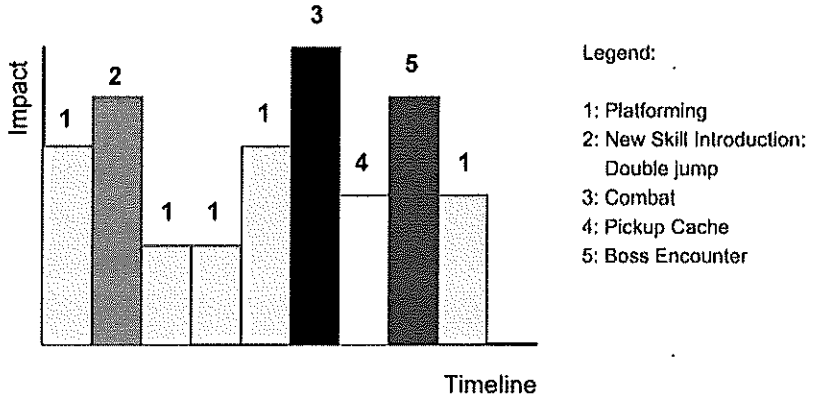

Figure 4.8. Event diagram.

[8] In fact, it is beneficial in all circumstances.

[9] Admittedly, an entirely made-up term.

As we can see in this example, the following elements contribute to a clear picture of the level:

- type of event (legend),
- event Duration (column width),
- chronology (column order),
- impact Value (column height).

These elements can each make a valuable contribution to the level design. Using this method, it is easy to identify and read them.

Type of event

The *type of event* can consist of any gameplay event intended by the level designer to occur at a specific moment. This can range from the very important unique moments (Set Piece X) to more generic or modular occurrences (Scripted AI Encounter Y). In the above example, other gameplay events include: *new skill introduction, boss encounter, pickup cache,* and *master key found.* Alternatively, or as a complement, less specific event descriptions can be used, like *combat, platforming* or *puzzle,* which still provide a good overview of a level's content.

Event duration

It is unlikely that this data can ever be exact, but it is still possible to convey helpful information. For example, it can be useful to show duration in relation to other events, or one can show patterns in gameplay that may be problematic, like a sequence of long, but low impact, gameplay. This may indicate a section of gameplay that the player may find boring. If precise information is known, this can be included alongside less specific descriptions, such as *short, medium* and *long.*

Chronology

As noted earlier in this chapter, chronology is partly dependent on linearity. Nonetheless, even in very open freeform games with a non-linear structure, things still often happen within a certain sequence. Skill trees may have to be developed, some areas may only unlock if a certain item has been found, defeating a specific creature may unlock a new skill, or the collection of a specific amount of items may allow the player to buy new gear.

In the above example, the *boss encounter* does not occur until after the *master key* is found. This kind of information makes it easier to maintain continuity and avoid many bugs.

Impact value

Emotional impact is a very vague term. How do we measure it? There is no satisfying answer to that, especially in instances where one player's subjective experience may be different from the next player's experience. But this does not mean that we should not try at all, especially in the type of levels where we are able to author the general experience of all players. Let's say, as in the above example, that we have included a set piece event in the level. Naturally, we assign a high value to it, most likely higher than for any other events in the level. If later playtests show that its impact on the player lies well below other events in the level, we know that we need to do something to raise the impact value of the event.

Instead of a general value like emotional impact, it can often be a better idea to represent a more specific element. In a survival horror game, this can be a *scare factor*, or in an action adventure it can be an *action quota*. Use whatever is right for the game you are working on.

How this is represented in the diagram is also open to many choices. Typical choices are *color coding* and column height, the latter being the one shown in our example. A line diagram can be used, or even a pie chart; it is up to the level designer to determine what is appropriate.

Event Diagrams Summary

Event maps provide an often-useful visual overview of important level design data. This allows the designer to quickly assess the level's overall *tone* and makes it easy to give others on the team a quick summary. However, in most cases it should not be seen as an alternative to a detailed and thorough level design document; instead, it is more effective in conjunction with one. This is especially true when it comes to highlighting specific elements within a level, as shown in previous examples.

It is of course entirely up to the level designer to decide which data to include and what type of visual representation to use.

The methodology is easiest to apply to linear level designs. Nonetheless, the methodology has applications even for completely open level designs.

Note that his methodology is not to be confused with *beat maps*. These provide a graphical overview of the player's "heart rate"[10] at specific moments in the level, but this is almost never a sufficiently useful technique.

[10] Typically by displaying a number that signifies heartbeats per minute.

Object-Oriented Level Design (OOLD)

As already touched upon in Chapter 3, "Level Design Goals and Hierarchies," it is helpful to think of level design components as modular objects. Another way of describing this methodology is as *object-oriented level design.* This term is based on an idea that originated in the world of computer programming, known as *object-oriented programming. Object-oriented programming* is:

> a modular approach to computer program (software) design. Each module, or object, combines data and procedures (sequences of instructions) that act on the data; in traditional, or procedural, programming the data are separated from the instructions. A group of objects that have properties, operations, and behaviors in common is called a class. *By reusing classes developed for previous applications, new applications can be developed faster with improved reliability and consistency of design.*[11] *(Emphasis mine.)*

One does not have to be a programmer to realize that from a design perspective, *reuse, improved reliability,* and *consistency of design* are major advantages. These are things that as level designers we always should try to achieve. If we look at *object-oriented programming (OOP)* in greater detail, we find further useful principles. There are a number of fundamental principles associated with OOP. These include these are: *encapsulation, inheritance,* and *polymorphism.* All of these should be of interest to a level designer.

These terms have been defined as follows:

> *Encapsulation* refers to the creation of *self-contained modules* that bind processing functions to the data. These user-defined data types are called "classes," and one instance of a class is an "object." For example, in a payroll system, a class could be Manager, and Pat and Jan could be two instances (two objects) of the Manager class. *Encapsulation ensures good code modularity...* [I]nheritance allows the structure and methods in one class to be passed down the hierarchy. That means less programming is required when adding functions to complex systems .... The ability to reuse existing objects is considered a major advantage of object technology....
>
> ... *Polymorphism* in the object-oriented approach refers to the ability of a programmer to treat many different types of objects in a uniform manner by invoking the same operation on each object. Because the

[11] "Object Oriented Programming," *The Columbia Electronic Encyclopedia,* Sixth Edition, http://www.bartleby.com/65/ob/objecto.html, 2001–2007.

> objects are instances of abstract data types, they may implement the operation differently as long as they fulfill the agreement in their common contract.[12] (*Emphasis mine.*)

Practical Examples of OOLD

In Chapter 3 we discussed modular level design. OOLD is a method that takes the principle of modular level design and applies it in practice, ensuring modularity and other similar concepts. The previously described elements of *encapsulation, inheritance,* and *polymorphism* can be easily incorporated into level design. Let's go through them one by one.

Encapsulation

In ID Software's game *Quake,* the player can often traverse a level with the aid of teleporters. The player just walks into one and is teleported to a different location in the level. This can be considered a *class* in level design terms. Once the basic elements are in place (such as trigger areas, destination areas, and three-dimensional models), we can have a class called a *teleporter.* Individual teleporters, as objects, can be one-way teleporters, or might feature different 3D models and have unique destinations and starting points. But they will always be teleporters and belong to the *teleporter* class.

Let's take a less literal example next: encapsulation applied to gameplay scenarios. Let's say that a level designer has spent a significant amount of time setting up a combat situation that makes use of specific aspects of the environment. (For example, cover, line of sight, and height difference.) If this particular situation works well in the game, the level designer may decide to replicate it by keeping the essential elements in place, but then providing different-looking environmental details, such as different AI opponents. The encounter's integrity stays in place, but the original benefit of the gameplay is encapsulated in each instance. (Or the environmental detail may stay the same and the combat can change.)

Inheritance

Early in the level design process for a game, we may want to create a generic object called a *treasure chest,* scripted in such a way that when the player opens it, a random item of limited wealth is generated (once). This may require a certain amount of scripting from the level designer, but once it has been set up, this treasure chest should be useful throughout the game. Say that play-testing now

[12] "Object-Oriented Programming," *The Free Dictionary by Farlex,* http://encyclopedia2.thefreedictionary.com/Object-oriented+computer+programming, 2009.

shows that players highly enjoy these moments of excitement where a random award is received, and the game team decides that this type of occurrence should be more prevalent in the levels. To avoid littering the level with treasure chests, the level designer decides to create a number of new objects, such as cupboards, filing cabinets, desks, or any furniture or similar object that can be opened and closed and used as a container. Players can now find treasure all through the level by opening up these new objects, without the objects feeling out of place.

Although these physical objects are new classes (in programming terms), when programmed as the same type of *object*, they can inherit much of their function from the behavior of the original *treasure chest* object.

Polymorphism

When a level designer can incorporate polymorphism into his or her work, a huge amount of flexibility is gained. Let's go back to our earlier example of treasure chests and the creation of other classes with overlapping functionality. Say that the game is nearly finished, the designer has placed hundreds of these chests and cupboard and drawers, vases, you name it, throughout the level. The publisher however now demands that a cash register sound should play whenever the player receives treasure. If polymorphism has been factored in the designer should be able to associate this sound to any object that makes use of this behavior simultaneously. This means that *making the change* once, *to the behavior* applied to all of these objects, not to all the objects individually.

Dangers

If we go back to our earlier example of *Halo*, we can find a good example of a persistent danger of this technique: if not careful, the level designer risks creating highly repetitive gameplay sections. In *Halo* this took the form of the now infamous "library levels." The library levels were created in such a way that they reused environments and gameplay in a very thorough manner. This was done in such a manner that the player is confronted with arbitrary repetition. Even if assets and gameplay sections are re-used, they still have to feel individual and worth playing. In the library levels, the player literally has to play the same thing over and over again, which is more akin to dreary work than to rewarding gameplay.

OOLD Summary

Applications of this kind of thinking, of borrowing these programming concepts and applying them to level design processes, are widespread and far reaching.

They range from simple puzzles to extended gameplay sections and even whole areas of level geometry. The examples themselves are not important; it is more about locating moments when this kind of thinking can be applied to anywhere that benefits from it.

Dense Level Design

An example of a level design methodology that combines elements of previously discussed methods can be found in *dense level design*. We speak of dense level design when a larger-than-normal amount of gameplay is incorporated in the physical environment of the level. This type of density can be achieved in many ways but generally occurs through reuse of the environment in time. New gameplay elements can be introduced to the level when a player revisits a section.

Another aspect of dense level design is density of gameplay *space*. All aspects of the gameplay space are exploited if possible. This can mean that a level will make much deeper use of vertically layered gameplay, incorporating all architectural features. For example, many doors can be opened and rooms can be entered, even if they serve no direct gameplay purpose.

Advantages

There are many clear advantages to this approach. So much so that in most cases, I advocate the use of this technique as one of the first things to look at in many level design tasks. Let's look at some of the advantages.

Reuse of assets

In most cases, as level designers we are expected to make the best use of the assets we have. If interesting gameplay can be replicated over several iterations without changing the assets needed, the resulting gameplay comes at a lower cost to the game's development.

Reuse of gameplay

This reuse of gameplay has many advantages, but an extremely important one is that it allows the level designer to teach players how to enjoy the game more easily. This is the case because the designer can take initially simple gameplay sections and slowly upgrade their difficulty and skill level in a controlled manner, without alienating the players. In this case, repetition and familiarity are your friends.

Immersive effect

If an environment is used to its fullest, if all of its features, be they interactive, artistic, or physical, are incorporated consistently in the level's gameplay, it will create a much more immersive experience. It gives the environment a sense of reality. Even if set in a non-realistic framework like fantasy or sci-fi, the environments still have to conform to the logic of that framework.

Dangers

There are several dangers associated with this technique, and some of them are hard to avoid.

Spatial confusion

Game levels are experienced through a two-dimensional interface. In most cases, this is a computer screen or television monitor and some kind of game controller. While this method is fine in principle, it does require certain concessions that in normal life do not occur. A good example of this is the way gameplay spaces in levels are often simplified to facilitate orientation.

Lack of gameplay readability

Most level designers are weary of reusing the same environment too many times without serious modification, as it is much easier to give a level area a specific purpose that is understandable to the player. If the player revisits an area and the gameplay has changed, it may need some re-evaluation on the player's side. This is not necessarily a problem, but there needs to be a good reason for this, or the player will feel ambushed. Something can only be done so many times before it becomes annoying.

An example: MINERVA—Metastasis 2

MINERVA is a modification for *Half Life 2*, consisting of a number of episodic levels designed by Adam Foster. The *MINERVA* mod actively reinterprets the gameplay of *Half life 2* by introducing a much stronger emphasis on density of play.

> Instead of relying on horizontally-sprawling, immense maps that stress the engine's area-capabilities to its max, *MINERVA* maps are *incredibly* small. This is because of Foster's ground-breaking idea to utilize every possible area to its maximum potential, and instead of expanding horizontally, he expands vertically. Rather than leave large areas wasted with inaccessible buildings, "fake" corridors and rooms to give the impres-

> sion of an immersive, realistic environment, Foster makes *every* area accessible. This doesn't mean that one can simply travel all over the maps in any manner one chooses—but instead through the use of very creatively-placed barriers Foster is able to funnel the player around the maps in a spiraled fashion.[13]

All throughout this level, it is notable that the gameplay is extremely layered. The player almost constantly revisits areas that have been traversed before, but which are updated with new gameplay features. This is especially true for combat with the game's enemy soldiers (Combine soldiers), presumably because they can use the same paths and doors as the player. So it is not illogical for them to make full use of the environment as well. The end result is that the action comes and goes, but the environment stays consistent. In most cases this is good enough, as the available tactical variation is strong enough to support this kind of repetition. Furthermore, it also contributes to the environment feeling very *real*, as if architecturally properly designed.

Where the level fails, though, partly because of the use of dense level design, is in readability. Quite often, players find themselves lost or unclear where to go now, a danger that is especially great if routes are backtracked several times. A player who has already been down a known route several time now has to wonder at every corner: "I've been here already, but has something else opened up?" Nonetheless, through the consistent use of dense level design, the final experience is very powerful. The player develops a real bond with the environment, which is very important in level design.

Further Examples

There are, of course, many other methodologies applicable to level design, some more obscure than others. Some level designers like to start their design by first working on key elements of the levels, to make sure that all the impressive features are worked out in time. Others work in a completely opposite manner, sketching out all the lighter detail first to make sure that a functioning outline of the level exists early on.

---

[13] "MINERVA: Metastasis 2," *MINERVA: Metastasis 2—Planet Half-Life*, http://planethalflife.gamespy.com/View.php?view=HLMotw.Detail&id=7, 2009.