Game Design Principles: A formal analysis
GAME DESIGN PRINCIPLES: A FORMAL ANALYSIS

BY

SALVATORE D’AMORE, B.Eng.

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE
AND THE SCHOOL OF GRADUATE STUDIES
OF MCMASTER UNIVERSITY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF APPLIED SCIENCE

© Copyright by Salvatore D’Amore, July, 2014
All Rights Reserved
TITLE: Game Design Principles: A formal analysis

AUTHOR: Salvatore D'Amore
B.Eng., (Software Engineering)
McMaster University, Hamilton, Ontario, Canada

SUPERVISOR: Dr. Jacques Carette

NUMBER OF PAGES: xvii [137]
In loving memory of Chris Thorne.
Abstract

Game design principles have been discussed and developed since the late 1990’s and are still created and discussed today. There have been many efforts to collect, identify, and create these principles, yet no effort has been made to test their implementation.

This thesis presents an approach to formally analyzing games with game design principles. We gathered 50 game design principles from textbooks, Internet journals, collection efforts, and academic papers. Informal analysis of all 50 were performed, specifically looking at their ability to apply directly to games itself, as well as how much knowledge of things external were required. Of these 50, 5 were selected and a more in depth analysis was done. For each of these 5 principles we present a specification one would use to model their game, and verify that the game design principle at hand applies.
Acknowledgements

First and foremost I would like to thank my supervisor Dr. Jacques Carette, without his guidance this thesis would never have come to fruition. Of the many struggles through which this thesis was born, getting excellent feedback was never one of them. I cannot thank him enough for that.

Secondly my family, whose support has been unending and seemingly without bounds. Without them in my corner, none of this would have been possible.

Thirdly, to my partner in life Cassie. Never warmer, never better, never happier, were the days before you in my life.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td>Contents</td>
<td>xi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xii</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xii</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Thesis organization</td>
<td>2</td>
</tr>
<tr>
<td>2 Prerequisite Knowledge</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Coverage</td>
<td>3</td>
</tr>
<tr>
<td>2.2 Avatar</td>
<td>4</td>
</tr>
<tr>
<td>2.3 Game World</td>
<td>6</td>
</tr>
<tr>
<td>2.4 Mechanics</td>
<td>9</td>
</tr>
<tr>
<td>2.5 Inventory</td>
<td>13</td>
</tr>
<tr>
<td>2.6 Attributes</td>
<td>14</td>
</tr>
</tbody>
</table>
3 Previous Work

3.1 Game Design Principles ........................................ 23
3.2 Tools for formal video game modeling .......................... 24
   3.2.1 MDA: A Formal Approach to Game Design and Game Research 24
   3.2.2 Formal Models and Game Design ............................. 25
   3.2.3 Explorations in player motivations: Game Mechanics ....... 26
   3.2.4 A Requirements Analysis for Videogame Design Support Tools 26
   3.2.5 LUDOCORE: A Logical Game Engine for Modeling Videogames 27
3.3 Tools for formal video game analysis .......................... 27

4 AsmL

4.1 The language of AsmL ........................................... 30
   4.1.1 Variables ..................................................... 30
   4.1.2 Initializing and Updating variables .......................... 30
   4.1.3 step ......................................................... 31
   4.1.4 Sets ......................................................... 31
   4.1.5 Class ......................................................... 32
   4.1.6 Searching sets .............................................. 33
   4.1.7 Maps ......................................................... 33
   4.1.8 ensure ....................................................... 35
13 Conclusion

13.1 Contributions .................................................. 127

13.2 Difficulties with Principles ................................. 128

13.2.1 Good level design constantly teaches the player something new 128

13.2.2 When a player overcomes a challenge that requires a resource, provide more 129

13.2.3 Clearly inform the player of short term goals ............. 129

13.2.4 Early levels are tutorials (Isolation Principle) ........... 129

13.2.5 Do not teach with death ........................................ 130

13.3 Future Work ...................................................... 131

Bibliography .......................................................... 131
# List of Figures

2.1 Link, found in A Link to the Past ........................................ 4  
2.2 Link, found in Ocarina of Time ......................................... 4  
2.3 Master Chief, found in the Halo series ................................. 5  
2.4 Dante, found in Devil May Cry ........................................... 5  
2.5 Main character found in the Pokemon FireRed .......................... 5  
2.6 Lara Croft, found in Tomb Raider ....................................... 5  
2.7 The Shovel Knight, found in Shovel Knight .............................. 5  
2.8 A Ship, found in Galaga .................................................... 5  
2.9 A randomly generated world in Dwarf Fortress .......................... 6  
2.10 Pokemon Island, in the game Pokemon Snap ............................. 7  
2.11 Celestic Town, in the game Pokemon Diamond Version ............... 7  
2.12 The overworld in Super Mario World ................................... 7  
2.13 A level in Super Mario World ............................................. 7  
2.14 Front facing angle of Midgar, found in Final Fantasy VII ............. 8  
2.15 Overhead angle of Midgar, found in Final Fantasy VII ................. 8  
2.16 The city of Los Santos in Grand Theft Auto: V ......................... 8  
2.17 The world of Chrono Trigger ............................................. 8  
2.18 Koopas and skeleton Koopas, enemies found in Mario’s Game World 10
2.19 Holes in the floor that will kill Mario

2.20 Megam Man Battling a series of enemies

2.21 Spikes in the floor that deal damage to Mega Man

2.22 An enemy who would attack should Lara fail the jump down

2.23 A Trap which kills Lara should she miss the ropes

2.24 Enemies attacking Master Chief from below

2.25 A button pad Master Chief uses, opens a door beside it

2.26 Link taking on two enemies

2.27 Falling into the lava will immediately kill Link

2.28 Frisk fighting the enemy known as Undyne

2.29 These NPCs will convey valuable story information to the avatar

2.30 Samus using the power beam mechanic in Metroid Prime

2.31 Samus using the phazon beam mechanic in Metroid Prime

2.32 The red potions are health potions, which are consumed by the healing mechanic of the avatar

2.33 The health bar, which is replenished by the healing mechanic

2.34 Mario wearing his fire suit, Mario Bros. series

2.35 Mario wearing the penguin suit in Super Mario Bros. Wii

2.36 The prince performs a wall climb in, prince of Persia: The Two Thrones

2.37 Faith going for a wall climb in Mirror’s Edge Catalyst

2.38 The inventory as seen in: The Legend of Zelda Ocarina of Time

2.39 The inventory as seen in: Diablo III

2.40 The inventory as seen in: Skyrim

2.41 Attributes of the player in the menu screen
2.42 Magic and health statistics in the top left ............................................ 16
2.43 Statistics in the menu page, experience, health, and mana on the main overlay ................................................................. 17
2.44 Kirby collecting beads in Kirby’s Epic Yarn ....................................... 18
2.45 Avatar standing over a felled enemy where they can collect a blood echoes ................................................................. 18
2.46 Corvo finds weapon ammunition in Dishonored ............................ 19
2.47 Goron link rolling through grass, collecting rupees ......................... 19
2.48 Mario in sight of yellow and red coins, in Super Mario 64 ............... 19
2.49 The avatar collecting Zinc, in No Man’s Sky .................................. 19
7.1 Acquisition Pattern in A Link to the Past ........................................... 46
7.2 The Bow as found in the Legend of Zelda: A Link to the Past .......... 47
7.3 Path leading to the final challenge ................................................... 48
7.4 Good level design constantly teaches the player something new, repre-sented as informal steps .................................................. 48
7.5 This room is impassable without use of the bow ............................ 50
7.6 The first Goomba blocks Mario’s way .............................................. 50
8.1 The avatar fights enemies in this challenge room ......................... 61
8.2 The player is rewarded a room over with health (Blue Bottle) and armor (Helmet) ......................................................... 61
8.3 Here we have a room that has many resources for the player .......... 62
8.4 An optional area (Bottom Half), with lots of resources for the player 63
8.5 The avatar fights this enemy with a pistol ...................................... 64
8.6 The player is not short on ammo, so is not offered any ................. 64
8.7 Master Chief battles with many enemies ........................................ 65
8.8 Avatar is rewarded with plenty of ammo for their next challenges .... 65
8.9 Avatar encounters Boom and Bewm ......................................... 65
8.10 Avatar is given a reward for defeating them ............................... 65
8.11 Pots that link can optionally pick up ........................................ 66
8.12 A town portal the player spawned to come replenish ................. 69
8.13 Where the player respawns when their trip to town is complete, exactly where they left .............................................................. 69
9.1 Player getting the goal from a partner NPC in The Last of Us [42] .... 79
9.2 Ezio getting directions from a friend in Assassin’s creed II [65] ..... 79
9.3 Firewatch giving the next goal directly to the player .................... 80
9.4 Dragon Age Inquisition using direct information .......................... 80
9.5 Quest given to the player .......................................................... 81
10.1 Fez: released April 13 2012 ..................................................... 90
10.2 Pacman: released May 22 1980 ................................................ 90
10.3 No Man’s Sky: released August 9 2016 .................................... 90
10.4 Tetris: June 6, 1984 .............................................................. 90
10.5 Minecraft: released October 7, 2011 ...................................... 91
10.6 Flappy Bird: released May 24 2013 ......................................... 91
10.7 Civilization 5: September 21, 2010 .......................................... 91
10.8 New Super Mario Bros Wii: November 11, 2009 ..................... 91
10.9 Space Invaders: released 1978 ............................................... 91
10.10 Shovel Knight: released June 26, 2014 ................................... 91
10.11 Mega Man: released December 17, 1987 ............................... 92
10.12 Devil May Cry: released 15 January 2013
10.13 Risk of Rain: released
10.14 Just Cause 2: Released March 23, 2010
10.15 Street Fighter V: February 16, 2016
10.16 (Portal) This room shows the player how to use the basic controls, only allowing the player to progress once they have experimented
10.17 (Bionic Commando) This room teaches the player all the uses for their mechanical arm, even ones they won’t actually get to use until later in the game
10.18 (Wind Waker) Here the player learns all of the possible sword attacks that are available throughout the game
10.19 (Oddworld: Stranger’s wrath) is approaching a contextual lesson, but this entire level just showcases mechanic after mechanic
10.20 (Halo 4) The tutorial rooms in halo have undergone quite the evolution, the halo 4 one being just a simple tutorial on using the action key
10.21 Assasin’s Creed II: Released November 17, 2009
10.22 Uncharted 3: Released November 1, 2011
10.23 Batman: Arkham Asylum: Released August 25, 2009
10.24 Shovel Knight: Released June 26, 2014. Player encounters rocks for the first time.
10.25 Mega man: Released December 17, 1987. Player encounters a new enemy
10.26 Hyper Light Drifter: Released March 31, 2016. Player encounters impassable area without the use of dash
Chapter 1

Introduction

Work on transforming game design principles into formal tools of analysis began with a desire to make small great levels, large great levels. I wanted to transform older brilliant levels that might have been held back because of technology, into larger levels that held the same brilliance. While beginning to explore this idea, we discovered that there was no respected way to judge the original level and subsequently no way to scale that level to a larger level with any kind of consistency or quality.

This naturally lead us to ask the question of what a "good" level is. To our knowledge there exists no known, purely objective way of judging a level. We turned to known game design principles, which have been documented as good metrics for designing games. Textbooks, bloggers, game designers, journals, have tried their hands many times at game design principles. While many principles are specific to certain scenarios and others that are probably completely wrong, it is irrelevant to this body of work. This work exists both as the first truly formal analysis of games via game design principles, and as a method for measuring if game design principles have any bearing on games themselves.
With an accurate model, we can determine if the principle itself is consistently found amongst highly rated games and inconsistently found in poorly rated games. If this is the case, then designers can model their games to determine if their game is in fact "good" in regards to that principle. While the scope of this thesis ends with analysis of game design principles and giving a model specification for each, the opportunities for use of our tools are vast.

### 1.1 Thesis organization

The primary goal of this thesis is to demonstrate a model specification that would allow a game designer to analyze their game in accordance to a game design principle. To bring the reader to the point where they can fully understand the model with which you would recreate your game world, we organize the thesis as follows.

First we present prerequisite knowledge, where we cover important elements of game design: concepts such as the avatar, game mechanics, inventories, and attributes. We follow with chapters on previous work in the space of formal video game analysis, and on the language of AsmL (Abstract State Machine Language). From here we have 5 chapters which go into great depth on the 5 game design principles we chose to analyze. We conclude with a chapter on some of the principles which were also considered for analysis, as well as future work.
Chapter 2

Prerequisite Knowledge

This chapter covers the fundamental knowledge that is required by all of the game design principles.

2.1 Coverage

One thing you will notice in this thesis is that multiplayer games are not covered. We focus specifically on single player experiences, which slightly limited our choice in game design principles. More importantly multiplayer games often heavily rely on skill, which is a metric that we did not cover in this thesis.

Open world experiences for the most part are also not included in this thesis. These experiences are generally open ended, with the play being self motivated. Most of our principles require the player to be moving forward, heading toward the game’s end state. Since these games have no end state, they do not fit the requirements for analysis.
2.2 Avatar

The avatar is an abstract representation of whatever entity the player controls, or “the player’s avatar is whatever represents him in the game world” [52]. While an avatar can take many visual forms in different games, a completely abstract model would consist of the following sets: Mechanics, Inventory, and Attributes. Each of these sets are covered in more detail later in this chapter, but briefly; Mechanics are the avatar’s and game’s functions for interacting with one another. Inventory is a set of items that have some quantity that the avatar uses to interact with the game world. Attributes are numerical properties of the avatar which are representative of what the avatar is. The following figures 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8 are all examples of avatars as found in all kinds of genres and settings.

A player controls the avatar using some physical input, and (with the exception of open ended games) tries to progress towards the game’s end state. For this thesis we do not cover the driving forces behind the direction of the avatar’s movement throughout the game world. Our goal is to provide specification for the game world
Figure 2.3: Master Chief, found in the Halo series

Figure 2.4: Dante, found in Devil May Cry

Figure 2.5: Main character found in the Pokemon FireRed

Figure 2.6: Lara Croft, found in Tomb Raider

Figure 2.7: The Shovel Knight, found in Shovel Knight

Figure 2.8: A Ship, found in Galaga
and tools for analyzing these specifications, it would be up to the system itself to
decide things such as which path an avatar is most likely to take. If you want a more
in depth analysis on different design patterns for driving player movement David
Milam and Magy Seif El Nasr looked into just that, their work examined the length
of time spent operating under the motivations of: Path movement, Pursue AI, Path
Target, Collection, and Player is Vulnerable \[39, 40\].

2.3 Game World

The game world is the environment that the player interacts with through control of
the avatar. The game world contains everything that interacts with the avatar, that
the player does not physically control. In some games a player might take over a part
of the game world for brief or extended periods of time, in these instances whatever
the player controls would become the avatar.

We present several game world examples, spanning many decades as well as genres
in the following figures: 2.9, 2.10, 2.11, 2.12, 2.13, 2.14, 2.15, 2.16, 2.17.

Figure 2.9: A randomly generated
world in Dwarf Fortress
Figure 2.10: Pokemon Island, in the game Pokemon Snap

Figure 2.11: Celestic Town, in the game Pokemon Diamond Version

Figure 2.12: The overworld in Super Mario World

Figure 2.13: A level in Super Mario World
Figure 2.14: Front facing angle of Midgar, found in Final Fantasy VII

Figure 2.15: Overhead angle of Midgar, found in Final Fantasy VII

Figure 2.16: The city of Los Santos in Grand Theft Auto: V

Figure 2.17: The world of Chrono Trigger
2.4 Mechanics

“A game mechanic is simply any part of the rule system of a game that covers one, and only one, possible kind of interaction that takes place during the game, be it general or specific” [37]. “These game mechanics are methods invoked by agents (the player), designed for interaction with the game state” [56].

Game mechanics manifest differently between games, but are found in all of them. As stated by Lundgren [37], game mechanics can encapsulate part of the game’s rule system. Physics, actions, hitboxes, and other technical details, can always be sorted into one of two categories: Mechanics of the avatar, and mechanics of the game world. Mechanics of the game world are any rules or systems where the world affects the state of the avatar or itself. Common examples of mechanics of the game world are enemies and hazards found in the environment. These examples both directly affect the avatar, but elements like buttons, doors, and NPCs, might all directly have an effect on the game world as well.

The following examples are mechanics of the game world that either affect the avatar or itself. In New Super Mario Bros. Wii [47] we can see enemies Koopa and Skeleton Koopa in Figure 2.19 and hazards taking the forms of pits where Mario will die if he falls in Figure 2.18. Mega Man X [14] sees Mega man battling foes in Figure 2.20 and taking damage from spikes in the floor 2.21. Lara croft sneaks up on an enemy within Figure 2.22 in Tomb Raider [18], after dodging death by cutting the ropes off this trap in Figure 2.23.
Figure 2.18: Koopas and skeleton Koopas, enemies found in Mario’s Game World

Figure 2.19: Holes in the floor that will kill Mario

Figure 2.20: Megam Man Battling a series of enemies

Figure 2.21: Spikes in the floor that deal damage to Mega Man

Figure 2.22: An enemy who would attack should Lara fail the jump down

Figure 2.23: A Trap which kills Lara should she miss the ropes
Mechanics of the avatar are any game mechanics where the avatar is changing the
state of the game world, or itself. Mechanics where the player inflicts damage, heals itself, moves an object, moves themselves, opens a door, etc, count as mechanics of the avatar. In Metroid Prime \[50\] one of the primary methods of interaction with the game world is through the use of the beam mechanic. This mechanic dispatches enemies, destroys debris, and open doors. Various beam mechanics have different effects throughout the world, two such beams can be found in figures 2.30 and 2.31. The healing mechanic in Diablo II consumes red potions from the inventory in Figure 2.32 replenishing the health meter in Figure 2.33. The player would command the avatar to use the healing mechanic when the health meter is low. In the Super Mario series the avatar changes its interaction with the game world through the use of power ups, that change Mario’s appearance and abilities as is seen in Figures 2.34 and 2.35. Parkour has been used as a game mechanic for avatar traversal, as can be seen in Prince of Persia 2.36 and Mirror’s Edge Catalyst 2.37.

Figure 2.30: Samus using the power beam mechanic in Metroid Prime

Figure 2.31: Samus using the phazon beam mechanic in Metroid Prime
2.5 Inventory

The inventory is a list of items, consumables, or abilities that the avatar draws from or replenishes. The inventory never acts, it is only acted upon.

Items in the inventory conceptually are hard to draw a fine line around, since items seem to have overlap with game mechanics. Naturally this makes sense since
the mechanics of the avatar and game world will be drawing from the inventory. Mechanics are hard to separate because they might require the use of inventory to operate; a gun requires ammo, a bow requires arrows, healing requires a potion etc. Notice that all of the mechanics require the inventory, but the inventory never does the consuming.

You can see in the Figures 2.38,2.40, 2.39 that inventories have many similarities, even across the huge generational gap between Ocarina of Time and the other two titles. Notice that in Figure 2.38 many of the items in the inventory have values beside them. These are positive integers that indicate uses or quantities of the items in inventory.

### 2.6 Attributes

An attribute is a numerical value that represents some facet of the avatar, and are usually never used to interact with the game world alone. Commonly used attributes might be: health, magic, experience points, dexterity, accuracy, speed, etc. What you
will notice is that the game’s systems adjust these values, but they do not directly influence anything else.

Figure 2.41 shows the attributes present in Borderlands. In this game attributes
are directly affected by what gun the player currently has equipped. Having been a
speed runner of Legend of Zelda games myself, I know that there are some attributes
that the player never knows the exact value of, even though they exist. You can see
in Figure 2.42 that the magic and health attributes are readily available to the player
for viewing, yet there is no way to know the exact damage output of your weapon.
In Figure 2.43 you can see many similarly named attributes as found in the former
eamples, but you can also see experience as the orange bar on the main overlay. The
majority of avatars have very unique appearances and abilities. Attributes are part
of what the avatar is rather than what it has.

Figure 2.41: Attributes of the player in the menu screen

Figure 2.42: Magic and health statistics in the top left
2.7 Resources

Resources are specifically the “ammunition” or “supplement” to the inventory, or attributes. A resource generally replenishes an inventory item through the process of some collection mechanic. Resources may also directly affect an attribute.

Certain avatar attributes like health, mana, and ammunition might not regenerate over time, rather only replenish through the collection of resources from the game world. Likewise items in the inventory might also be replenished through the game world, by collection of resources. A resource is specifically gained through the game world, as the result of completing some challenge.

Resources are incredibly common throughout gaming, both as incentive to explore, and necessity to continue. We present several games where the avatar collects, or can collect resources, demonstrating their appearances. In Kirby’s Epic Yarn the player collects beads, these contribute to the player’s score at the end of the level. These beads can be seen in Figure 2.44, notably the avatar is incapable of actually dying in this game, each time the player takes damage they lose some beads. “Blood echoes” in Bloodborne double as both the game’s currency and its experience points. The player collects blood echoes by felling enemies, as seen in 2.45.
uses more traditional resources to replenish the avatar’s inventory and attributes. In Figure 2.46 Corvo finds a bin with ammunition for his pistol, a common occurrence in Dishonored. The Legend of Zelda series has very consistent methods of collecting resources, in Majora’s Mask grass is a common location for resources that the avatar requires. You can see Link collecting rupees from this patch of grass in Figure 2.47. In Super Mario 64 coins are the most abundant resource, some levels require 100 yellow coins to get a star (of which you need 120 for the complete ending), some require 8 red ones for a star as well, not to mention coins restore the health attribute and with 100 of them, the lives attribute. In Figure 2.48 Mario heads towards a group of yellow and red coins, which will aid him in his completion of the game. In the game No Man’s Sky, the player collects resources to upgrade their ship and weapons. In Figure 2.49 the player collects the resource Zinc.

Figure 2.44: Kirby collecting beads in Kirby’s Epic Yarn
Figure 2.45: Avatar standing over a felled enemy where they can collect a blood echoes
Later in chapter 8 we discuss the player’s intended path in relation to resource replenishment. The intended path is the ordered set of challenges the game designer expects the player to take.

Back tracking can be defined as the player moving back through their intended path, for the sake of progressing forward. Some back tracking is generally fine, as long as new content is given or experienced. The player should not be back tracking on the intended path to collect resources because none were provided. This forces the
player to experience the same gameplay again, for the sake of restocking their supplies. Should the designer allow the player to progress without the required resources, we could end up in a state of soft deadlock.

Soft deadlock is a position where the player cannot leave the challenge they are attempting, but a reset of the game / loading of a previous save can reset the position of the avatar. In most instances, even if the player lacks the "recommended" amount of resources, with efficient play the player can probably overcome the challenge. Soft deadlock occurs in the circumstance where there are not enough resources to win, even with the best possible play. It usually occurs when a challenge requires a single resource to progress, since many challenges have generic completion requirements.

These concepts are important because simply modeling the game via its challenges reduces the risk of either of them occurring. In many cases, should the analysis of a games model fail, it is because the player will be unable to proceed due to some lack of in game content, or knowledge.

2.9 The Player

The player is the physical human being that controls the avatar within the game world. Our aim is to objectively analyze a game without modeling the player, thus we abstract them out of the picture. While there may be some studies showing that enjoyment is measurable, it greatly simplifies the analysis if we do not involve the player at all. We are really seeking an analysis of the game itself, which is independent of the player, focusing on game design rather on game play.

For any principle that involves the player somehow, we have to make assumptions about the game, or find an in game proxy that represents the player's involvement.
2.10 Challenge

Challenges are one of the most important abstract concepts in our analysis. How a game can be divided into a set of challenges will vary greatly depending on the game. The concept of a challenge is meant to encapsulate all the actions the player makes to complete some in-game objective. Every challenge is some set of tasks that the player must complete. This can be anything from simply getting to a certain location, to defeating all the enemies in some room. Perhaps a room has many different challenges, or none at all. What is actually abstracted will depend on the modeler: as long their abstraction is consistent a game can be broken down into a set of challenges.

Generally a challenge has a set of pre-conditions, a set of tasks, and set of post-conditions. A set of pre-conditions is required to ensure that a player has completed any requisite challenges as well as having all items, mechanics or attributes the challenge needs. A Task-list is what the avatar must accomplish while in the challenge, which stands as a place holder, and currently does not alter the game’s state at all. The post-condition reflects the changes in the game’s state once the player has completed all the challenge tasks; post-conditions are capable of modifying connections between challenges, replenishing the inventory with resources, and placing the avatar in a different challenge. This representation is extremely general, and each principle may not require all of these to properly model the game space. For that matter, a game may require only one of pre-requisites, task-list, and post-conditions.

If the modeler decides that a room within their game is a challenge, then they should be consistent with that thought. If the modeler decides that the largest number of enemies in a challenge is 5, than the model should consistently reflect this. A challenge might map directly to the game world, but it is possible that a section of
the game world is broken into many challenges. Challenges are often referred to in literature as objectives, tasks, or goals.
Chapter 3

Previous Work

Game studies have grown rapidly in the past two decades, as academics try to understand what makes video games so appealing. As it pertains to a game’s design there have been many approaches, among them is flow theory (Chen [15] and Csikszentmihalyi [19]). This approach has been used by at least Chen and Sweetser [62] to layout guidelines for designing game systems. Although we were unable to find any previous work attempting to use game design principles to formally analyze video games.

3.1 Game Design Principles

Game design principles are widely discussed, but very little study has been done on those that exist besides trying to procure as many as possible [20]. This is a fact that interests us since there are numerous references to them throughout textbooks and popular culture, yet apparently no formal study on the effectiveness of the principle themselves. The closest we managed to find to this work are Holopainen and Björk [28] who were trying to systematically identify patterns, and provide some themselves.
Even this effort did not provide any rationale behind the patterns, aside from basic observation.

This thesis breaks down selected game design principles offering an in depth analysis, as well as offering a mechanism for analyzing a game to prove the principle was used. This opens the door for determining if there is any correlation between a games success, and whether or not these game design principles were applied.

3.2 Tools for formal video game modeling

There are many attempts to formally model games either for academic or industry purposes. Here we document some of these, and their contribution to modeling.

3.2.1 MDA: A Formal Approach to Game Design and Game Research

More a tool for formally understanding games Hunicke et al break down game design into three phases Mechanics, Dynamics, and Aesthetics. The player is meant to consume the game as rules, system, and fun, which correspond respectively to Mechanics, Dynamics, and Aesthetics. In this framework mechanics describe particular components of the game, at the level of data representation and algorithms. Dynamics describes the run-time behavior of the mechanics acting on player inputs and each others outputs over time. Aesthetics describes the desirable emotional responses evoked in the player, when she interacts with the game system.

This framework breaks down aesthetics into sub categories, and intends for the designer to do many passes iterating until the game is fully modeled. This is a good
strategy for designing games, but does not aid in analyzing existing games. This work was intended to bridge the gap between game design and game development, allowing for one games framework to be adapted into different games simply by adjusting the aesthetic and thematic elements. This work presents an interesting approach for relating player interests with the design process, but does not provide real modeling tactics for the game itself.

3.2.2 Formal Models and Game Design

Grunvogel [26] provides a mathematical approach to formally model games. This work represents games as a triple (S,M,F), where S is a set, M a monoid, and F is an action of M onto S. F is the game’s rule system, S is the state of the game, and M represents the input of the system. This is a perfectly valid way to model games, and could possibly have been an alternative to the modeling method that we selected. The two problems brought up in this work is the complexity, and the unreadability of a model done in this style. In addition to these issues, this work is for modeling alone, and although one can simulate a game’s outcome there is no given metric for analyzing the modeled game. In fact the author states that “We will not go into methodologies of game design e.g. the top-down approach which is reflected in different kinds of design documents [Laramee, 2002], or iterative approaches, or some other methods and approaches (like the The 400 design rules project [Falstein, 2002] , formal abstract design tools [Curch, 1999], game patterns [Bjork et al., 2003] ) which are actually used to create a good game with great balanced gameplay and forth. In fact, the above theoretical formalism inhabits no model for fun and does not prevent anybody from describing very bad games”.

25
Even the simplest of examples in this paper seemed cumbersome, and modeled too many of the lower level functions to ever be able to do analysis of a modern game as a whole. That stated, a model of the same form but more abstract specification may have worked. We found using AsmL to create our own models achieved the same goal, and were far more readable. AsmL also had the given advantage of including ensures clauses, which allows the analysis to terminate if at any time some criteria is broken. The model provided in this paper did not have the same level of functionality.

3.2.3 Explorations in player motivations: Game Mechanics

Barbaros Bostan explores the motivations of the player in [10] linking game mechanics, with Murrays hierarchy of needs [41]. This is an interesting paper which breaks down the game mechanics found in Fallout 3, associating them to psychological needs: Materialistic, Power, Affiliation, Achievement, Information, and Sensual. While this paper does analyze player motivation, and provide a metric for which to judge mechanics, it does not aid in the modeling of the game space nor with an analysis of the game itself.

3.2.4 A Requirements Analysis for Videogame Design Support Tools

In their 2009 paper Nelson and Mateas [43] worked to discover what designers wanted from a design tool. They discovered that there was a split between the queried designers wanting, front end / back end design tools as well as a vocabulary for higher level design. This work researched requirements for some sort of design tool, but did not develop any method for modeling.
3.2.5 LUDOCORE: A Logical Game Engine for Modeling Videogames

LUDOCORE is a logical game engine that links game rules to formal logic used by automated reasoning tools in AI. The paper that covers this game engine actually states that “iterative-design motivation also contrasts with the goals of formal verification. While verification attempts to prove that a set of desired properties hold, it is not clear during a games prototyping phase which properties are even desirable” [57]. While there is truth to this, our motivations are to apply verification to a fully modeled game.

3.3 Tools for formal video game analysis

The majority of efforts for formal study have been from a psychology or sociological perspective [38]. There are very few efforts that look to analyze games for their in-game content, this is briefly discussed by Consalvo [16]. Of the hundreds of papers, blogs, texts, and presentations examined during the writings of this thesis, very few discussed design patterns (principles), let alone offer any formality for analysis.

The only two papers we could procure that analyzed game worlds, specifically looking at analyzing level design and the patterns within them were [40] and [39]. Both of these papers were written by Milam and Seif El Nasr, and identified 5 patterns which would guide the player through a level. The first paper defined each pattern, the second analyzed 20 games for their applications of these patterns. They then compared the scores of each game in accordance to how they measured up by their patterns. To accomplish this, each game’s first twenty minutes were transcribed to
match the amount of time the player spent adhering to each pattern. This does not create a model which can be modified, or analyzed by any other patterns, should they exist.
Chapter 4

AsmL

This thesis heavily relies on models to abstract away from details that are irrelevant for analysis. We abstract games into a series of challenges, with the avatar transitioning from challenge to challenge. We then create a state machine from this model, and have selected the language AsmL (Abstract State Machine Language) to represent these models.

We selected abstract state machines because in each principle it is important that the state of the avatar and game world over time are accurate. The avatar operates as a state machine, whose state can be changed by consequences in the game world. The game world operates as a state machine which is effected by the avatar’s actions. Our selection of AsmL stems from its readability and its executable nature. AsmL allows us to create data structures which represent the avatar and game world, and moving between states depending on a game’s logic AsmL fits our criteria well.

AsmL was created by Gurevich [27], who also formalized the notion of an abstract state machine. The specifications in this thesis were based on the AsmL tutorial found at [25]
4.1 The language of AsmL

This section contains information from the following tutorial [25] which was used for writing of AsmL in this thesis. Any notation that is required to read the specifications in this thesis are provided here.

4.1.1 Variables

Variables are defined in the following manner:

Listing 4.1: Defining a Number, and a String

```plaintext
var foo as Number
var bar as String
```

As seen in this thesis:

Listing 4.2: Defining a set of used mechanics

```plaintext
var usedMech as Set of Mechanic
```

4.1.2 Initializing and Updating variables

To initialize a variable we use:

Listing 4.3: Assigning an number to foo

```plaintext
var foo as Number
var foo = 6
```

To update a variable we use:

Listing 4.4: Updating bar with the value of foo

```plaintext
var bar := foo
```
4.1.3 step

To understand a step, the tutorial gives the following example. “The behavior of a computer chip is another example of states and state transitions. This behavior can be modeled as steps of an abstract state machine where each step corresponds to the execution of one hardware instruction. A single hardware instruction may result in many changes to the chip’s hardware registers”.

Hello, world

Each step is an execution:

**Listing 4.5: Hello World in AsmL**

```plaintext
step WriteLine("Hello, world!")
```

Fixpoint

To emulate a game which is meant to terminate once the avatar reaches its final challenge, we step until fixpoint. Fixpoint is reached once the previous step yielded no changes in the model’s state.

As seen in this thesis:

**Listing 4.6: Steping until there is no longer a change in state**

```plaintext
step until fixpoint
  do something here
```

4.1.4 Sets

Sets are represented in the following way:
Listing 4.7: Instantiating a set of strings

```plaintext
var A as set of String
var A = {"one", "two", "three"}
```

To specify the eventual instantiation of a set:

Listing 4.8: Defining a set

```plaintext
var a as set of b
```

We can add and remove elements with:

Listing 4.9: Adding and removing elements from sets

```plaintext
add a to b
remove b from a
```

### 4.1.5 Class

Classes in AsmL do not contain any member variables or functions. Classes are just abstract elements or object-identities.

Listing 4.10: Defining a Number, and a String

```plaintext
Class Person
    var a as Number
    var foo as set of bar
```

**Updating attributes**

Here is an example of instantiating and updating a person class
Listing 4.11: Instantiating a person, setting their age to ten

```javascript
Class Person
    var age as Number

p as Person
p.age = 10
```

4.1.6 Searching sets

Within this thesis, there are many times we look to iterate through sets, for this we use forall.

Listing 4.12: Defining a Number, and a String

```javascript
forall a in b
    foo := b
```

To select particular elements in a set, which meet some criteria, we can use:

Listing 4.13: Format for selecting an element from a set

```javascript
var a = {b | b (criteria)}
```

Here is an example of selection in AsmL:

Listing 4.14: Selects all elements in b that are less than 3

```javascript
var setOfLessThanThree = {a | a in b where a < 3}
```

4.1.7 Maps

A map is a set of relationships between keys and values.

Listing 4.15: Formatting for defining a map

```javascript
var foo as Map of Number to String
```

In this thesis maps are used to store the changes in the game world that may occur from completing a challenge, mapping a start challenge to its new adjacent challenge.
It is also used to store the relationship between an item and a reward of resources to that item. This is demonstrated below:

Listing 4.16: Example for using maps

```java
class Resource
    var name as String
    var max as Integer
    var count as Integer

class Challenge
    var name as String
    var adj as Set of Challenge

    var consumes as Map of Resource to Integer
    var rewards as Map of Resource to Integer

    var currentNode as Challenge

Main()
    step until fixpoint
    foreach resource in currentNode.consumes
        Writeline(Indices(resource))
        Writeline(Values(resource))
```

This example is based off of the principle "when a player overcome a challenge that requires resources, provide more". Here we have a challenge, which has a two maps, consumes and rewards. "Consumes" maps a resource to the value of what it takes from the player, "rewards" maps to the value the player receives. The key can be referenced by using the Indices command, the values by the Values command.
4.1.8 ensure

Ensure clauses document constraints on an AsmL model. At runtime, a violation of an ensures clause will throw an assertion failure. They are written in the form ensures \textit{statement}, where \textit{statement} is some statement that results in a boolean.

\textbf{Listing 4.17: An example of an ensures clause}

\begin{verbatim}
var a as Integer
a = 4
ensure (a < 3)
\end{verbatim}

The above code snippet will throw an assertion error since \(a\) is meant to be less than 3, and is assigned the number 4.

\textbf{Listing 4.18: A practical example of ensure}

\begin{verbatim}
AllTickets = {1..1024}

IsTicketID (i as Integer) as Boolean
require i > 0
return (i in AllTickets)

ChooseAnyTicket () as Integer
ensure IsTicketID(result)
return (any t | t in AllTickets)
\end{verbatim}

The above example is taken directly from [25], and is an example of a ticket system. The ensures clause here documents that a ticket must both be within 1 and 1024. The program then selects a ticket as random from AllTickets.
Chapter 5

50 Game Design Principles

Originally we collected 50 game design principles, but a few of them were extraordinarily similar to one another thus they were merged into the more easily understandable of the two. It is important to make this distinction, because all of the principles that were duplicates came from different sources. This leaves us with a total of 42 game design principles that might be looked at for analysis.

I specifically selected game design principles that pertained to the game world itself, and avoided ones that talked about the process of building the game or the design process. Our goal is to determine if a game is ”good”, more specifically that it adheres to the game design principles that have been determined to benefit games.
5.1 Levels of Analysis

5.1.1 Tags

With a collection of 42 game design principles there is not enough room in this thesis to provide a detailed explanation for each principle as to why they were not included. Several principles had fundamental problems with them and were not even considered. The provided tags indicate not only the issues with the principles, but what it would take to complete an analysis of them.

Completed in Thesis (CT)

If the principle has this tag, then it is analyzed in depth in this thesis.

Detailed in Thesis (DIT)

If the principle has this tag, it is covered in some depth in the thesis. Principles with only this tag are completable given some amount of time.

Requires Distillation (RD)

A principle with this tag either covers too great a scope to be useful, or cannot be understood in its current state. A good example here is section 12.2 in chapter 12 reward the player, where the principle simply fails to be specific as to the kind of reward, or even what it wants to say about rewards. Within this principle and its description there indications that there are other principles that will in fact be useful.
**High Complexity (HC)**

Principles with this tag require a large number of elements from the game world to be present in the model.

**Vague (V)**

If this tag is given to a principle, then the principle is too vague in its wording, and the author’s description does not help at all. No amount of research or distillation can identify what the author meant when they wrote the principle.

**Subject Undefined (SU)**

Undefined subjects are concepts that cannot be measured because they are not well understood by anyone, example; fun.

**Out of Scope (OS)**

This tag is reserved mainly for the concepts of skill and difficulty. Currently these concepts are out of scope of this work, because we are omitting the physical player’s inputs. Difficulty is directly tied to player skill, so it is also out of scope. Principles with this tag could become analyzable if we formalized a way to bring in skill and difficulty.

**Not Applicable x**

If a principle has this tag in one of it’s columns, then it is not associated to this time frame.
5.1.2 Immediate Analyzability (IA)

The tags in this category represent the status of the principle in terms of it’s analyzability within a short time. If there is an N/A here, then it is not completable in the short term.

5.1.3 Future Analyzability (FA)

If there are tags under this header, then the principle is not completable within a short time frame. This usually stems from there being a fundamental issue with the principle itself, such as an undefined subject.

5.1.4 The Principles

<table>
<thead>
<tr>
<th>Principle Name</th>
<th>IA</th>
<th>FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good Level Design constantly teaches the player something new [63]</td>
<td>CT</td>
<td>x</td>
</tr>
<tr>
<td>When a player overcomes a challenge that requires more resources, provide more [4]</td>
<td>CT</td>
<td>x</td>
</tr>
<tr>
<td>Allow the player to determine the consequences of actions [34, 28]</td>
<td>DIT</td>
<td>x</td>
</tr>
<tr>
<td>Reward the player [4, 33]</td>
<td>DIT, V, RD</td>
<td>x</td>
</tr>
<tr>
<td>Do not teach with death [36]</td>
<td>x</td>
<td>V</td>
</tr>
<tr>
<td>Early levels are tutorials [4, 36]</td>
<td>DIT</td>
<td>x</td>
</tr>
<tr>
<td>Large rewards, small punishments [36]</td>
<td>DIT</td>
<td>x</td>
</tr>
<tr>
<td>No positive feedback loops [20]</td>
<td>DIT</td>
<td>x</td>
</tr>
<tr>
<td>Provide Parallel Challenges [20]</td>
<td>DIT</td>
<td>x</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----</td>
<td>---</td>
</tr>
<tr>
<td>Provide an enticing long term goal [20]</td>
<td>DIT</td>
<td>x</td>
</tr>
<tr>
<td>Avoid Dominant strategies [20]</td>
<td>x</td>
<td>HC, RD</td>
</tr>
<tr>
<td>Good level Design empowers the player [63]</td>
<td>x</td>
<td>DIT, V, RD</td>
</tr>
<tr>
<td>Good Level design tells the player what to do, but not how to do it [63]</td>
<td>x</td>
<td>V</td>
</tr>
<tr>
<td>Clearly inform the player of short term goals [36]</td>
<td>CT</td>
<td>x</td>
</tr>
<tr>
<td>Good Level Design allows the player to control difficulty [63]</td>
<td>x</td>
<td>OS</td>
</tr>
<tr>
<td>Good level Design is driven by your game’s mechanics [63]</td>
<td>x</td>
<td>V</td>
</tr>
<tr>
<td>No words to tell the story [63]</td>
<td>x</td>
<td>V</td>
</tr>
<tr>
<td>Thesis, Antithesis, Synthesis [53]</td>
<td>x</td>
<td>V</td>
</tr>
<tr>
<td>Uniqueness [53]</td>
<td>x</td>
<td>RD, V, HC</td>
</tr>
<tr>
<td>Use negative feedback to balance difficulty and player skill [20]</td>
<td>x</td>
<td>OS</td>
</tr>
<tr>
<td>Levels should get harder [20]</td>
<td>x</td>
<td>OS</td>
</tr>
<tr>
<td>Emphasize acquisition, cater to greed [20]</td>
<td>x</td>
<td>HC</td>
</tr>
<tr>
<td>Rewards should be proportionate to the challenge [20]</td>
<td>x</td>
<td>OS</td>
</tr>
<tr>
<td>Challenges should vary in more than degree [20]</td>
<td>x</td>
<td>OS</td>
</tr>
<tr>
<td>Task</td>
<td>Cardinality</td>
<td>Group</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Vary the level’s pacing</td>
<td>36</td>
<td>HC</td>
</tr>
<tr>
<td>Vary the rate of difficulty increase</td>
<td>20</td>
<td>OS</td>
</tr>
<tr>
<td>Implement multiple difficulty settings</td>
<td>36</td>
<td>OS</td>
</tr>
<tr>
<td>Anticipation</td>
<td>5</td>
<td>V</td>
</tr>
<tr>
<td>Fight Player Fatigue</td>
<td>x</td>
<td>HC, RD</td>
</tr>
<tr>
<td>Challenges should require skill</td>
<td>x</td>
<td>OS</td>
</tr>
<tr>
<td>Good level design is fun to navigate</td>
<td>63</td>
<td>SU</td>
</tr>
<tr>
<td>Good level design is surprising</td>
<td>63</td>
<td>SU, OS</td>
</tr>
<tr>
<td>Good level design creates emotion</td>
<td>63</td>
<td>SU, OS</td>
</tr>
<tr>
<td>Make your game familiar, yet different</td>
<td>x</td>
<td>V</td>
</tr>
<tr>
<td>Challenge</td>
<td>53</td>
<td>V</td>
</tr>
<tr>
<td>Entertainment</td>
<td>53</td>
<td>V, OS</td>
</tr>
<tr>
<td>Escapism/ Immersion</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Reward the player for skill, imagination, intelligence, and dedication</td>
<td>36</td>
<td>V, OS</td>
</tr>
<tr>
<td>Foreground &gt; Background</td>
<td>20</td>
<td>OS</td>
</tr>
<tr>
<td>Make the level make sense</td>
<td>20</td>
<td>OS, V</td>
</tr>
<tr>
<td>Simple as possible</td>
<td>20</td>
<td>V</td>
</tr>
<tr>
<td>Maximize potential expressiveness</td>
<td>20</td>
<td>OS</td>
</tr>
<tr>
<td>Consistent interface</td>
<td>20</td>
<td>OS</td>
</tr>
</tbody>
</table>
Chapter 6

Analysis Methodology

The following steps lead us from a game design principle, to a fully specified model with which we can use our principle to analyze a video game.

1. Informal analysis of the principle

   This step is to inform the reader of all facets of games that should be known to get a good grasp on what the principle at hand applies to. It is important that we define, source, and explain any new elements of games previously not discussed in the prerequisite knowledge section. Any assumptions that must be made about our game will be addressed here.

2. Principle Clarifications

   Some principles have incomplete information, that is addressed in comments made within their presentation of the principle. It is difficult to capture everything the principle addresses in one simple sentence, so we make clarifications here.
3. **Requirements and Representation for a model of the Game World**

   To analyze our game, we look to specify a model of less complexity than the game itself. To do so we extract from the game a list of requirements, that will allow us to analyze our principle. Once this is done, we provide a specification for the structure of the model.

4. **Mapping of principle requirements to challenge**

   Equipped with a model specification from the last step, we specify the steps that map the model directly to the principle.
Chapter 7

Good level design constantly teaches the player something new

This principle is written by game developer Taylor in the Gamasutra article: Ten Principles of Good Level Design [63]. While there are likely many influences that lead to this principle’s development, these are not the focus of this chapter. Our aim is to break down this principle, determine what parts of a game it affects, and provide an algorithm which determines if said game does constantly teaches something new.

7.1 Informal Analysis

In this section we take an informal look at our principle, elaborating on some of its nuances and give examples of its intended applications.
7.1.1 Principle Analysis

According to Taylor [63]: “A good level should either introduce a new game mechanic, or put a spin on an old one to make the player re-evaluate his or her established paradigm”. Introducing game mechanics at regular intervals has been around since the first console generation, where game mechanics themselves became understood as a primary method for keeping the player interested. By introducing new mechanics, the player must learn new skills, keeping gameplay from becoming repetitive. While Taylor states that a level can re-evaluate an old mechanic [63], it is not clear how unique the extended mechanic has to be for no skill to transfer. It is assumed that mechanics have disjoint enough skills to have to be relearned, otherwise there would be little value in introducing a new mechanic.

Examining The Legend of Zelda: A Link to the Past [45], one can find that each dungeon embodies the topic principle. In the first dungeon of the game, the avatar acquires the bow and arrow as seen in Figure 7.2. Following this acquisition, the dungeon presents the player with many challenges that can be completed by use of the bow. In all but one of these rooms, use of the bow is optional. The developer likely thought the player would use it because of its novelty, but to ensure that the player does use the bow at least once, the room before the final boss requires the bow to defeat a Red Eyegore as shown in Figure 7.5. If the player cannot kill the Red Eyegore that is in that room, they cannot proceed, as no other mechanic can defeat it. The level’s culmination is a final boss, which is far easier if the bow is used (Boss is top left in Figure 7.3).

This first dungeon exhibits a pattern that can be found in all the dungeons in The Legend of Zelda: A Link to the Past [45].
1. Early areas in the level require only old mechanics
2. Some way through a level the avatar acquires a new mechanic
3. The avatar is presented with challenges where this new mechanic can be used
4. The player then faces a challenge where they must use the mechanic to advance any further

Figure 7.1: Acquisition Pattern in A Link to the Past

The Zelda franchise is not the only one to feature this pattern [7.1], it can also be found in games such as: Mega Man, Mario Bros, Halo, Borderlands, and Diablo III [13 14 2 11 9]. Games that uphold this principle offer many compelling and completely disjoint mechanics or consistently iterate over the same game mechanics in clever ways. This is in the hopes that diversity of mechanics will keep a player’s interest. Once a player has mastered a game mechanic, in so far as they cannot gain any more proficiency, keeping gameplay from getting repetitive will become harder. This is why this pattern exists [7.1] if mechanics require separate skills, the player will once again have to walk the road to mastery. If there are many mechanics that the player has acquired and mastered, then the level can create variety by having challenges that require more complex sets of mechanics.

While not the primary basis of this principle, correlation can be drawn with the principle of flow [15]. With flow, the player becomes bored if they have too much skill, and the game is not challenging enough. They become frustrated if the opposite is true. When the player learns a new mechanic, they will have to once again become proficient. This allows the level designer to adjust the difficulty, rather than creating purely "end-game" content. It is imperative that if a designer is adhering
to this principle, that their mechanics require different skill sets. Once the player is introduced to a new mechanic, their proficiency at dealing with challenges will drop. If you maintain the same level of difficulty from earlier challenges, players are likely to become frustrated.

Ninety Nine Nights \cite{49} is a game where the avatar clears hordes of enemies with powerful attack combinations. The player is rewarded new mechanics based on performance and receive new mechanics at a near-constant rate. Ninety Nine Nights executes the design of it’s mechanics poorly, and “although the characters have their own unique move sets and animations, the same lack of strategy applies to all of them” \cite{12}.

Games like Ninety-Nine Nights intend for the player to receive new mechanics and master them throughout the course of a level. While mechanics will always have some overlapping skills, it is expected that the player’s proficiency will be lower with new mechanics as long as there is some new skill or strategy. Ninety-Nine Nights’ game
1. The avatar acquires a new mechanic
2. The avatar demonstrates understanding of mechanic’s required skill or strategy
3. Repeat this pattern at a constant rate throughout the game

Figure 7.4: Good level design constantly teaches the player something new, represented as informal steps

world is designed for this to be true, but the mechanics are so similar that mastery is completed early on, and the player never learns anything new. The consequences here are two-fold; the gameplay becomes repetitive because the designers intended you to approach challenges differently as mechanics are supposed to be different, and the difficulty was never raised because the player was supposed to be learning once again. Following the theory of flow [15] if the game’s difficulty is constant, and the player becomes too skilled, they will eventually get bored. This game provides anecdotal evidence of never teaching something new, where the primary focus of the player is to gain new mechanics and utilize them.
As shown earlier, an implementation of this principle can be found in a Link to the Past [15]. The Legend of Zelda series is very consistent in its delivery, and it uses the same imagining of mechanics. Other games might not use items as their main delivery for mechanics, or have such clear "levels". The dungeon format in the Legend of Zelda has a clear beginning and a clear end, with clear moments for mastery and verification of a learned mechanic. Other games such as Diablo II [8] give the player far more choice, and have a more loose definition of what levels are. Diablo II also has more options for which of its mechanics can be carried through the levels, but the player is still granted mechanics at a constant rate as they progress through the game, their system for unlocking is simply different. Diablo II is also an interesting case because it can be played by multiple different avatars, which all have distinctly separate sets of mechanics. This kind of game might actually benefit from the player being forced to use each different ability a few times, rather than having complete freedom. This is a topic for later discussion.

Our goal is to formally verify the topic principle, which is represented informally in figure 7.4. We look to verify that the acquisition and verification of new mechanics happens constantly throughout the game’s levels.

### 7.1.2 Learning

Our principle seeks to ensure that the player constantly learns something new, and “on a larger scale, this constant learning should be measured out across the entire game, to make sure that each level delivers fresh gameplay” [63]. To determine if the player has in fact learned a mechanic, we measure proxies within the game world. We can assume that the player learned the functionality of a mechanic, if they successfully
used it to complete a challenge at least once. This decreases complexity of the model, and allows a designer to evaluate their game without an individual having to play it.

You might expect that simply acquiring a mechanic is enough to know that the player has learned, but there are many games where a mechanic is given to the player and never used. An excellent example of this can be found in the classic RPG Grandia [22], where the player obtains many spells and abilities, but is able to complete any conflict with abilities of their choosing. The player could use only one type of ability the whole game, and thus acquire a mechanic, yet never really learn.

![Figure 7.5: This room is impassable without use of the bow](image1)

![Figure 7.6: The first Goomba blocks Mario’s way](image2)

While it is perfectly acceptable to have optional mechanics, core mechanics such as the bow in The Legend of Zelda, always have areas that ensure you learned to use them. In figure 7.5 the avatar is stuck in a room with a red enemy and a skeleton enemy. It is impossible to by-pass this room without use of the bow. Figure 7.6 shows the very first obstacle in [46] Mario Bros., of which you already have the mechanic to complete (the jump mechanic), but have not yet used it. This technique has been discussed by Miyamoto in an interview [11] where he said they used it as a guarantee
that the player had learned how to jump. Using the same strategy, we can guarantee that the player has actually learned any mechanic.

7.1.3 Limitations

This principle focuses on the rate at which the player learns mechanics. Games where there is no general direction but the player’s own, are not within our scope. We limit our scope of coverage on this principle to progression based games, where the player moves through the intended challenges, completing them and advancing the game’s state. This eventually culminates with an end state, where the player can no longer continue.

7.2 Principle Clarifications

The principle’s text does not immediately make it clear which types of games it does not apply to. We can deduce what a game must have to be in accordance with this principle, simply by looking at its language. Since it is a principle of level design, we know that some representation of a game’s levels are required. We need a way to measure the rate at which a player learns something new, which we covered earlier in the chapter. To measure learning we will need to represent the "something new", which we established is the avatar’s mechanics. Our model has to preserve the order of the game’s challenges, so there is an accurate number of challenges between when the player learns each mechanic.

The principle is general in its coverage of what a mechanic does functionally (Chapter 2 section 2.4), but clearly focuses on mechanics that the avatar uses, rather
those of the game world. This is implied since the player is meant to be the one learning something new, but it would not be difficult to also include mechanics of the game world as well. Mechanic examples include, the bow [45], beam types [50], and new guns [51].

This principle uses the word constantly, which is vague in that we remain unaware if the game is to be teaching something new all the time, once every few challenges, etc. Running our analysis on a group of games organized by sales or rating, we may be able to discover some “good” rate of learning.

7.3 Requirements and Representation for a model of the Game World

This section builds a relationship between our model and a video game’s world. We keep in mind that we only require aspects of the game that pertain to the player learning to use their mechanics, any other part of the game does not need to be modeled at this time.
### 7.3.1 Game World Requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>The order of challenges are preserved</td>
<td>Required for ordered reception of mechanics</td>
</tr>
<tr>
<td>Avatar can be given mechanics</td>
<td>Required for measuring learning of mechanics</td>
</tr>
<tr>
<td>Challenges can require certain mechanics be used</td>
<td>Required to ensure the player has learned a mechanic</td>
</tr>
</tbody>
</table>

### 7.3.2 Game World Representation

**Model**

To model the state of each mechanic, we keep a set of those the avatar could acquire, has acquired, and has used. Since we do not know the most optimal "constant" rate for learning mechanics (which may vary between games) we need this rate as input, as well as a maximum number of challenges by which the rate can vary.
Listing 7.1: AsmL specification of the game world's structure

```java
class Mechanics
    var Name as String

class Challenge
    var requires as Set of Mechanics
    var rewards as Set of Mechanics
    var adj as set of Challenge

    var availableSet as Set of Mechanics
    var acquiredSet as Set of Mechanics
    var verifiedSet as Set of Mechanics
    var constantLearningRate as Number
    var variance as Number
```

**Variables and Definitions**

Above we provided the structure for the model, what follows are the definitions of the variables which were used.
### Variable

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>availableSet</td>
<td>Game’s mechanics that have not been acquired or used</td>
</tr>
<tr>
<td>acquiredSet</td>
<td>Game’s mechanics that have been acquired</td>
</tr>
<tr>
<td>verifiedSet</td>
<td>Game’s mechanics that have been acquired and used</td>
</tr>
</tbody>
</table>

**Mechanic**

Finite set of abstract entities which represent the interaction between the avatar and the game world.

**Challenge**

An abstract set of challenges with sets of: required mechanics, rewarded mechanics, and adjacent challenges.

**constantRateOfLearning**

This variable represents the value for the rate which the player is to learn.

### 7.4 Mapping of Principle requirements to our Model

Good level design constantly teaches the player something new. Leading up to this formal analysis, which is an algorithm that analyzes an AsmL model, we have given explanations and definition to all of the necessary variables and concepts. Below we provide the formal specification of the principle at hand, which extends the model of the game world and ensures that the player constantly learns something new.
Our AsmL model’s second part specifies the algorithm designed to implement “constantly teach the player something new”, it is broken into four steps:

1. Our first step is to check whether there are any mechanics that are required by this challenge, that the avatar has learned. Our approach is to see if all of the required mechanics for the challenge are currently not possessed by the avatar in any capacity. If all of the mechanics required here are in the set of available to acquire mechanics, than we terminate execution as we have reached a state
2. If the challenge passes the first step, then check to see if any mechanics are used in this challenge that are currently acquired by the avatar. If there are, move them to verified.

3. If the challenge passes the first step, then check to see if any mechanics rewarded by this challenge are in available. If they are, move them to the set of acquired mechanics.

4. Now that we have modeled the behavior of the system, we can ensure that our topic principle is held. We are unsure what the best constant is, or what kind of variance is allowable. We ensure that for each of the recorded learning times, they are within the constant rate of learning plus or minus the allowable variance.

Steps 1, 2, and 3 model the game world’s behavior, where step 4 captures the requirements of the principle itself. Without modeling the game’s behavior it would be impossible to determine whether or not the player learns at a constant rate. Notably avatar attributes and inventory are missing from this analysis. This is because they are irrelevant when verifying whether or not the player has learned a mechanic’s function, and would only burden the model with more information.

While we have successfully captured the essence of this principle, we do not have any values for optimal rates of learning. This will have to be determined in future work, where games of different ratings and sales are modeled to determine what a good rate of learning might be. We are unsure whether there is some universal rate of learning, or if the rates are correlate between games that are similar in style or
genre.
Chapter 8

When a player overcomes a challenge that requires a resource, provide more

Simply stated, this principle ensures that a player never undertakes a challenge that requires resources, without enough resources to complete the challenge.

8.1 Informal Analysis

We begin our analysis with examples of how resources, and resource replenishment may take their shape in games. Once an understanding of how resources are generally found in a game environment is formed, we talk about what the principle accomplishes.
8.1.1 Principle Analysis

While the contents of each challenge and the function of each resource may differ wildly from game to game, genre to genre, in the context of this principle they can be abstracted to the same notion. Resources and challenges share the following relationship: challenges are a set of tasks that may require the use of resources, and may provide resources as a reward.

To illustrate this principle, we look to first person shooter games (FPS) to provide a few examples. According to Game Design: Fundamentals \cite{4} “You should also make fresh supplies available to him immediately after he surmounts a challenge that costs him a lot of resources ... these traditionally take the form of boxes of ammunition and medical kits”. This can be seen throughout games like Doom \cite{31}, Halo \cite{3}, and Bioshock \cite{23}, where the player faces off with hoards of enemies and immediately finds themselves collecting ammunition afterwards to replenish their stores. In the most extreme cases, not replenishing the resources of the player could lead to a deadlock. In particular, a state where the player has the mechanics that they require, but not the resources needed to execute said mechanic. This can lead to player frustration, and the requirement of constant back tracking in order to replenish resources for upcoming challenges.

The frequency of replenishing resources can vary greatly between different game genres. With the first person shooter genre, a player might tackle challenges using the same weapons each time, running out of ammunition more frequently. In genres where the player uses resources less frequently, or where the player can complete some challenges without the use of resources, replenishment is usually less frequent.
8.1.2 Applied in Games

Here we present a myriad of different games, spanning many years and genres. This is to provide anecdotal evidence that challenges and resources have remained relatively similar, and can be measured by this principle in the same way. It is important to notice that these games can have very different appearances, and all come from separate publishers.

Doom

The classic game Doom was one of the first FPS to ever be released. The game helped pioneer a genre that has become a mainstay in gaming. Doom would be one of the games that inspired the principle we are specifying now. In order to have the player capable of tackling later tasks doom provided armor, health and ammunition. As seen in figures 8.1 and 8.2 the player encounters several enemies, and once defeating them is replenished with health and armor.

Figure 8.1: The avatar fights enemies in this challenge room

Figure 8.2: The player is rewarded a room over with health (Blue Bottle) and armor (Helmet)
A Link to the Past

This principle is generally used to specify the layout of First Person Shooter maps, which are heavily dependent on the resources the player has, but the Legend of Zelda franchise makes use of this as well. A Link to the Past frequently includes pots, chests, and often just resources themselves for the player to consume. The game also has resources drop from enemies as well, often times the challenge that consumes a resource, replenishes it.

Figure 8.3: Here we have a room that has many resources for the player

Figure 8.3 shows a room where there are plenty of resources, but the surrounding rooms do not necessarily require any of these. This room is prior to difficult challenges that could require any (but not all) of the resources the player has available to them. This tactic is employed frequently within the Zelda series, where the player has many different ways of completing the same challenges.

Figure 8.4 shows an optional area, which the player does not have to unlock, but
Figure 8.4: An optional area (Bottom Half), with lots of resources for the player doing so rewards them with plenty of extra resources.
BioShock

In our BioShock example figure 8.6 shows another instance of the player being rewarded within the same challenge that consumed their resources. Notice that the player will not get more ammunition, as they are not low on ammunition. It is common for the level designers to let the player get lower on resources so the game feels more real, and more challenging.

Figure 8.5: The avatar fights this enemy with a pistol

Figure 8.6: The player is not short on ammo, so is not offered any

Halo 4

Halo 4 provides an example where the player clears a challenge and is immediately rewarded with resources that they utilized. The Master Chief (Avatar) clears the horde of enemies, then collects the new guns that are waiting for them within the same challenge.

Borderlands 2

Borderlands combined the RPG (Role playing game) genre with the FPS genre to form an entertaining and compelling hybrid. This challenge is a boss battle which
begins with an interesting cut scene, and the reward for victory is a modification on an existing weapon and extra resources to replenish what was lost.

8.1.3 Intended Path

The intended path conceptualizes the order of challenges you expect the player to take, but it may not be the only possible path. If branching paths are available for the player to take, it can be difficult to determine which path a real player would take without play testing. Game designers like to put rewards for the player on branching paths, to encourage exploration and replay value. Amongst the many rewards that a player can gather, are resources.

While including extra resources on branching paths is not a negative thing, it is
imperative you place resources on the intended path. Failure to do so, could result in back tracking or deadlock 2.8. These concepts are worrisome for the intended path, because the player is meant to proceed here. This is not some awkward random situation the designer could not have anticipated, this is play as intended. Especially on this path, the player should always have access to the resources to progress. Resource management is important in games, but unless it is thematically prevalent (Horror and survival games) this pattern should occur within challenges. A player playing a game for the first time should not have to anticipate a future challenge, which could be anything, and even require new mechanics they did not yet have.

Without play testing it would be impossible to determine if a player will actually pick up the resources on the intended path. If we place them on the path in an obvious enough manner, it should be enough. Notice that in the Legend of Zelda: Twilight Princess, there are pots strewn about that the player can break. There is no guarantee that they break them, but should the player seek resources, they are there to be collected.

Figure 8.11: Pots that link can optionally pick up
8.1.4 Concerns

Ernest Adams elaborates on this principle with the following statement: "You should also make fresh supplies available to him immediately after he surmounts a challenge that costs him a lot of resources" [4]. By stating that the player should be replenished immediately, the principle constrains replenishment to the challenge that consumed resources, or perhaps the challenge directly after. Immediate resupply does accomplish the goal of providing the player with more resources. In fact it over constrains the game designer, where simply ensuring that they have enough for the upcoming challenge would be enough.

To elaborate on why this is an over constraint, here is a thought experiment. The player has a mechanic called bow, which requires arrows and has a capacity of 60. They go through a challenge that requires 10 arrows. If no challenge were to require any more arrows, there would be no reason to refill. This is unrealistic, surely, but we can extend this logic to there being only one more challenge that requires arrows. Is there any reason that the player needs more arrows right after the first challenge, if the next arrow consuming challenge is 20 challenges away. It should be adequate to ensure arrow numbers are replenished any time prior to the next usage.

We can extend our bow example further to demonstrate the lack of a need to replenish before absolutely necessary. Previously we did not specify a cost to our second challenge. Say it requires 20 resources, had we replenished we would have 60 arrows. Had we not replenished, we would have 50, still enough for the challenge. Clearly forcing the designer to replenish right away is not required. It is possible the player plays inefficiently, or uses the bow in other challenges where it was usable, but not required. Logically the designer cannot account wastefulness, because the player
could be just as wasteful even if you had replenished immediately.

Should the player be on the intended path, having them enter a challenge with too little of a required resource is poor game design. Some challenge along the way should have given them the opportunity to collect enough resources to complete this challenge. We take note that the player does not need more than what is required to complete the challenge, which is less than the principle asks. For this reason, we offer an alternative ensures clause that accomplishes the goal of not letting the player run out of resources, yet is less restrictive than having the consumed resource be immediately refreshed. Our second ensure clause checks that the player has enough resources to complete the upcoming challenge.

8.2 Principle Clarifications

When examining this principle’s application, we keep in mind its goals and its text. The principle states that when the player faces a challenge that requires a resource, provide more. All mechanics that require a finite number of resources are meant to be bound by this principle. Simply put, neither this principle nor our modification of it are not adhered to, the player will be forced to backtrack or could face deadlock.

Not all games will have a single path that is intended. Some have randomly generated levels whose shapes change as you progress, many possible paths to reach the goal, or a truly open world. Most open world experiences offer choices in which tasks to accomplish next, but once the player chooses a goal the path becomes more strict until it is done. Those that are truly open world might require a more predictable way to refresh resources. Some games such as Diablo III \[9\] have mechanics to take the player quickly back to a previous location and replenish their inventory, able to
come back to exactly the same spot they warped from. Truly open world games are not bound by this principle however, as there is no intended way to play, you cannot ensure that the player encounters resources before undertaking a challenge.

Figure 8.12: A town portal the player spawned to come replenish

Figure 8.13: Where the player respawns when their trip to town is complete, exactly where they left

8.3 Requirements and Representation for a model of the Game World

Previously we laid out the requisite background knowledge which applies to this principle, this section places requirements on our model that stem from the game world. These requirements are the bare minimum to ensure the accuracy and correctness of the principle, as we try our best to keep a minimal model.

8.3.1 Game world requirements

When ensuring whether or not the player’s resources are replenished in a timely fashion, one might expect that knowing which challenges replenish or consume a
player’s resources would be enough. While our analysis is meant to ensure that resources are replenished, there are considerations to be had of the game world itself. The game model has to connect challenges to represent an accurate ordering of the game. Without this, it would be impossible to know if a resource was replenished after a challenge, before it, or anywhere but in the consuming challenge itself.

For a player to have their resources replenished, they must also be consumed. This sets up our ordering of challenges, where resources are consumed, then replenished (either right away, or before you run out depending on analysis). Games tend to change the availability of challenges (geographically taking the form of rooms, dungeons, sections, or whole levels) as the player progresses. As we saw in figure 8.4 the player does not always acquire resources on the intended path. The room in the lower half figure 8.4 is not one that the player must visit, nor is the method to reach this area ever explained in the game. One can only reach this area by using a technique that is applied through experimentation. Often the player will find themselves stranded (locked doors, broken bridges, blocked paths), unable to visit past locations until they have completed the goal of their current level. Modification of paths is considered, but not handled by our model. Our goal is to ensure that the player has their resources replenished, by following the possible paths, the path itself is to be modified in a separate process.
8.3.2 Game World Representation

Here is our recommended model of the game world for this principle. Since we only seek to verify whether or not the player’s resources have been consumed and replenished, we present as minimal a model as possible.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources can be consumed</td>
<td>Ensures the avatar’s resource count reflects actual play</td>
</tr>
<tr>
<td>Resources can be replenished</td>
<td>Ensures we can refill the avatar’s resource count, reflecting actual play</td>
</tr>
<tr>
<td>Challenge order is preserved</td>
<td>Ensures the order of consumption and replenishment is correct</td>
</tr>
</tbody>
</table>

Model Specification

To model the results of a challenge, our model has a map of resources to how much they consume. We also have a map from resources to how much is rewarded to the player upon completion. An obvious problem comes to mind here, what if the game world gives the player resources before the challenge begins (in the pre-condition). Since this model will not require pre-conditions the modeler can make a challenge before with an empty set of tasks, which yields the resource in question as a reward. This simplifies the specification at the cost of putting a little more work on the modeler.
Listing 8.1: AsmL model of the Game World

```
class Resource
    var name as String
    var max as Integer
    var count as Integer

class Challenge
    var name as String
    var adj as Set of Challenge

    var consumes as Map of Resource to Integer
    var rewards as Map of Resource to Integer

    var avatarResources as Set of Resource
    var currentNode as Challenge
```

**Game World Variables**

These are the variables used in our specification to keep track of the game world’s state.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>avatarResources</td>
<td>A set which represents the amount of each resource that the avatar has available to them</td>
</tr>
<tr>
<td>currentNode</td>
<td>The node with which to start the graph</td>
</tr>
</tbody>
</table>
8.4 Mapping principle requirements to challenge

The code snippet below represents the analysis of our model specified in the previous section. Our model is a greatly simplified version of a game world, the requirements of that model exist to analyze a game world with this AsmL model.

Listing 8.2: AsmL model of the Game World’s behaviour and principle analysis, ensures implied by principle

```asm
Main()
    step until fixpoint
    var next = any node | node in currentNode.adj
    for each resource in avatarResources

    // Consume the resources that the challenge needs
    for each usedResource in challenge.consumes
        if (Indices(usedResource).name = resource.name) then
            resource.count := resource.count - Values(usedResource)

    // Reward the resource that the challenge needs
    for each givenResource in challenge.rewards
        if (Indices(givenResource).name = resource.name) then
            resource.count := Values(givenResource).count + resource.count
            if resource.count > resource.max then
                resource.count := resource.max

    // Implied by principle
    ensure(resource.count = resource.max)

    currentNode := next
```


Listing 8.3: AsmL model of the Game World’s behaviour and principle analysis, recommended ensure clause

```asmL
Main(
    step until fixpoint
    next := any node | node in currentNode.adj
    for each resource in avatarResources
        // Consume the resources that the challenge needs
        for each usedResource in challenge.consumes
            if (Indices(usedResource).name = resource.name) then
                resource.count := resource.count - Values(usedResource)
        // Reward the resource that the challenge needs
        for each givenResource in challenge.rewards
            if (Indices(givenResource).name = resource.name) then
                resource.count := resource.count + resource.count
                if resource.count > resource.max then
                    resource.count := resource.max
        for each nextChallengeResource in next.consumes
            if (resource.name = Indices(nextChallengeResource))
                // Better version
                ensure(resource.count - Values(nextChallengeResource) >= 0)
    currentNode := next
)
```

The first ensure clause in Listing 8.2 is implied by the principle and located at the end of the resource loop. Here we ensure that the resources of the player are back to their maximum. Placing it at the end of the loop means that the consumption and replenishment from this challenge have occurred, and the player should progress with fresh resources.

Our recommended ensure clause in Listing 8.3 comes directly after the first. Early on we capture what the next challenge in the series will be, and we use that here. We peer into the next challenge and take how much of the resource we are evaluating, is consumed. We then ensure that the avatar will have 0 or more of the resource when they move to the next challenge. As stated before, this means that the player never
runs out, but can actually have their resources depleted significantly.
Chapter 9

Clearly inform the player of short term goals

As a player looking to complete a game, it is important that there is a clear objective to progress towards. This chapter illuminates what goals in games are, as well as give examples of goal delivery systems in games. We provide an AsmL specification so that we can analyze whether or not games would be completable, without the player having to guess what their next goal is.

9.1 Informal Analysis

This section informally details the knowledge required to understand what short term goals in games are, and how games clearly inform the player of them.
9.1.1 Principle analysis

As part of their model for flow, Sweester and Federoff stated: “Games should provide players with clear goals at appropriate times” [62] [21]. This is fundamental to their GameFlow model, and supports this principle’s requirement that goals should be provided for the short term. As well as being provided in the short term, goals should also be clear [15]. In his book Sweetser [61] elaborates on intermediate goals, which are akin to short term goals in games. An intermediate goal should be clear and presented at appropriate times. While not vivid about timing, this helps us define what the designer is really looking at when it comes to short term goals. Goals are clear tasks that the player completes to progress the game towards its end goal.

9.1.2 Goals

Cowley [17] gives us a better understanding of tasks having clear goals: Survival, collection of points, gathering of objects and artifacts, solving the puzzle. A goal is a task or challenge that will eventually need to be completed, else the game will not reach its end state. It is imperative that the player be unable to continue progress without the completion of a goal.

A goal is a special kind of task, that allows the player to know that they have progressed the game forward. Games without goals do exist, and usually take the form of open world, open ended experiences. These are games like Minecraft [48], Terraria [59], and the end game of Grand Theft Auto [51]. For these kinds of games a principle such as this does not apply. As soon as you require that a task be completed to advance the player towards the game’s end state, you have provided a goal. The tasks that stand between the avatar and their known goals may also be goals. It is
up to the designer to ultimately decide how short, short-term is. For games that have many parallel challenges available, knowing the short term goal is going to be more important. If the game is linear, short term goals may just be the end of the level. One could posit that the more open world a game is, the more frequently you will have to place goals, and subsequently shorten the pace of informing.

9.1.3 A categorization of goal delivery systems

This section creates a division between the different ways that the game can inform the player of their short term goals. It should be noted that the video games often employ combinations of these delivery systems, in order to diversify how the player learns what their next goal is. This is necessary to justify the different components that the model will require in order to get a full picture of the game with respect to this principle.

Plot provisions

With games that have plot lines, it can frequently be found that the plot gives to the player their next short term goal. This mechanism is popular amongst games where the plot is the major selling point, as so much time is invested into the story and all that goes with it, designers frequently tie as much as they can to it. This can be seen in figure 9.1.3 below, in the game The Last of Us, where an NPC gives the player their next goal through the dialogue.

A second example can be found in figure 9.1.3 where the player receives their next goal from a friend.
Direct information

In Firewatch [55] short term goals are displayed overhead without disrupting any of the player’s inputs. It should be noted that the game does not tell the player exactly what to do, just gives them knowledge of what the next challenge that progresses the plot is.

While some games take Firewatch’s approach of informing the player briefly, games like Dragon Age Inquisition [7] takes the approach of persistently keeping the players goal on the screen by default.
In game content

Games might not rely on particularly obvious ways of communicating what the next goal for the player is. This includes; items, land marks, and objects. A good example here is a locked door. The player should see it and realize that their next goal is to find a key hidden throughout the level. The important part, is that it is a specific task or challenge that makes the player realize what their next goal is. It is not something that the designer introduced in conjunction with the gameplay itself.

This categorization is different from direct information, because it requires effort from the player, either in interpreting the information or finding it. Direct information is given to the player directly, and has no interaction with the avatar. In other words,
this information comes from the game world, as opposed to the game’s interface.

Below we see the player picking up a seemingly random book that informs them that the quest Forbidden Legend has begun. This method of delivery could be the least immersion breaking method around, considering that the player receives no help in locating this object. It could also be incredibly frustrating, considering that there is no distinction between this book, and the thousands of other books found throughout the game. From play experience, we noticed that optional tasks are often given this way, but game progressing goals are given out with one of the other delivery methods.

![Figure 9.5: Quest given to the player](image)

**Linear Play**

With this method the goal is always to reach the end of the level. The important distinction between this categorization and others, is that there can only be one path in linear play. If the player can optionally choose more than one route, linear play is not a valid system as it is possible that the gameplay is branching, and requires backtracking. This is a very specific and very dangerous method to use, as there is a lot of room to confuse the player as to what their goal is. Any branching can cause the player to get lost, and not know what the goal is. This method might also become
less clear to the player, if some levels are branching, and others are not.

Warnings aside, this tactic is seen almost ubiquitously throughout the platforming genre. These are games where the player moves towards a single goal, from a single starting location. Early platforming games made it impossible to backtrack, meaning that forward was the only option.

9.1.4 Assumptions

By definition non-progression based games where the player’s goals are intrinsically motivated, are ineligible for analysis. These games have no end state, and therefore no amount of information can inform them of their goals.

Of the principles that this thesis covers, this requires the largest number of assumptions. This principle requires the reader to assume that if the player accesses a challenge they actually acquire the information that is meant to be provided there. It is more than possible that the player looks away, fails to read, or fails to understand the information they were provided. We also assume that if the player learns of a short term goal, then they will not forget it. This being said, our second assumption is less of an issue in modern games. Modern games generally have an interface that keeps track of the player’s next short term goal, so it becomes almost impossible for the player not to recall what their next objective is. The third assumption is that the player cannot find their goals without being informed of them, which is of course the worst case scenario. Many games (especially older ones) do have goals, but never inform the player. These games were eventually beaten by extremely determined players, who would scour the whole world for their goals. Notably older games were smaller than new ones, it is far less likely the same process could occur in the vast
worlds built today.

These two assumptions could be researched more in the future to determine what makes a short term goal more memorable, as well as the requirements for what makes the information easier to extract from a challenge.

9.2 Principle Clarifications

A short term goal is a particular challenge with side effects that progress the game, or further the player’s position. \[17\] All of the examples presented here reinforce this suspicion. All of the given examples should have a clear goal at the end, to know that the player has indeed completed the set of challenges required. The furthest goal that should probably considered to be short, is the end of the section dedicated as ”level”, as this is the distance that would qualify as an intermediate goal.

9.3 Requirements and Representation for a model of the Game World

While the intentions of this principle are clear, the method with which we should check if it is implemented is not. It would be very easy to model the whole game, and only ensure that the player is informed about their goals, before they actually reach the goal itself. You could also ensure that the player is informed of their goals before they start on a path that pursues that goal. These approaches are perfectly valid, but we feel they do not go far enough.

The two previously mentioned methods are great, but miss out on an obvious
loophole. The first approach cannot guarantee that the player knows their goal, until literally the challenge that is the goals precursor. This is akin to giving directions from China to Italy by letting a traveller find the Italian border, and telling them Italy is just ahead as their foot is about to be on Italian land. While the directions given were correct, they might have been just a little late.

The second approach is much better. The player knows how to get where they are going, once they have reached the beginning of their journey. The flaw here is much more subtle, and requires having played some games yourself. For linear games, where the player has no choice in direction, this is actually the most minimal approach. The player has to get to the start of their quest, so you must have told them about their path before they started their journey. In more open world games, this becomes far less true. If you have choices at the beginning of the game, you may never reach the first challenge that informs you of your next goal. This will make it impossible to start your quest, and you will never be informed of your short term goal.

Our approach is more drastic, but the best way to ensure the player is informed of all their goals that lead to the games end state. First we disable all paths, so the player is cannot travel to any challenge they have not been informed of. Once a player has reached a goal, that goal is marked complete. As the player moves around the graph if they touch a node where all of that nodes required goals are complete, the player is informed of their next goals, and paths to them are enabled. This makes it "impossible" for the player not to know where they are going. This also allows the modeler to test for a more natural approach to progression, where the player discovers their goals by playing.
9.3.1 Game world requirements

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Challenge Order is preserved</td>
<td>The player should not be able to go to challenges that have not been revealed by an information node</td>
</tr>
<tr>
<td>Paths can be enabled</td>
<td>While all paths exist, the avatar will only be able to travel along paths that are enabled</td>
</tr>
<tr>
<td>Reaching certain challenges enables a path</td>
<td>This emphasizes the fact that it is the challenge that gives the player their required information</td>
</tr>
<tr>
<td>Challenges can be designated as goals</td>
<td>In our implementation, the avatar will proceed to locate the goal node, traversing edges until it has found the location that the information node designated as the goal</td>
</tr>
</tbody>
</table>

9.3.2 Game World Representation

To ensure that the player is clearly informed of their short term goals, we present a model specification and a formal algorithm to analyze that model. This model and its analysis, follow the requirements of the game world that we had set out above.
Model Specification

Listing 9.1: AsmL model of the Game World

```asm
var name as String
var adj as Set of Challenge

var isActiveGoal as Boolean
var requiredToInform as Set of Challenge
var setGoals as Set of Challenge
var informNodes as Set of (Challenge, Challenge)

var currentNode as Challenge
var completedGoals as Set of Challenge
var endNode as Challenge
```

Game World Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>currentNode</td>
<td>Since a node holds its adjacent challenges, the challenge the avatar is currently at</td>
</tr>
<tr>
<td>completedGoals</td>
<td>A list of all completed goals, used to determine if</td>
</tr>
<tr>
<td>isActiveGoal</td>
<td>Since any node could be the current goal, this variable keeps track of whether or not the node is an active goal</td>
</tr>
<tr>
<td>requiredToInform</td>
<td>The set of goals that need to be achieved to unlock the goals that this challenge has to offer</td>
</tr>
<tr>
<td>setGoals</td>
<td>The set of goals that get activated, should all prerequisite goals be met.</td>
</tr>
</tbody>
</table>
9.4 Mapping of Principle requirements to our model

Listing 9.2: AsmL model of the game world’s behaviour, and the analysis step

```asciiml
Main()
  step until fixpoint

  if (currentNode.isActiveGoal = true) then
    add currentNode to completedGoals
    currentNode.isActiveGoal := false

  // Check if all goals are completed
  if (size(currentNode.requiredToInform intersect completedGoals) =
      size(currentNode.requiredToInform)) then
    foreach edge in currentNode.informNodes
      add edge(1) to edge(0).adj
    foreach goal in currentNode.setGoals
      goal.isActiveGoal := true

  // Select the next transition
  currentNode := any node | node in currentNode.adj

  ensure (endNode = currentNode)
```

Taken to its logical conclusion, clearly informing the player of short term goals can be seen as, the player not knowing what to do without being informed. While this is not usually true, due to an often linear style employed by games, if a player is given a mostly open world situation, they could remain completely lost without some form of direction.
Chapter 10

Early levels are tutorials (Isolation Principle)

This chapter covers several kinds of tutorials that are frequently seen in video games. We take a close look at the isolation principle, breaking it down and giving a formal specification for analyzing a game with it.

10.1 Informal Analysis

Tutorial sections of a video game are designed to teach the player about the game’s various obstacles and mechanics 38. Most modern games include some form of tutorial, in order to teach the player how to interact with the avatar. We form 4 categories of tutorials, strengthening our use of these classifications in order to specify which type of tutorial we are going to be specifically examining. While we do intend to limit our scope to one kind of tutorial, it is important that we describe tutorials to give context as to our narrowing of scope to the isolation principle.
10.1.1 Tutorials in games

Three of the tutorial styles are taken from [60] while not an academic publication, the styles are consistent with most tutorials, as will be demonstrated by listing many games that follow these styles. These categories are simply fancy names for the four possible styles of tutorial: Explanation of each mechanic through reading, through forced play, through natural play, and not having a tutorial at all. There is a fifth category taken from [58] which is what we end up analyzing.

No Tutorial

No tutorial occurs when a player receives no visual, audio, or textual instruction. Instruction does not occur for mapping physical inputs to game mechanics, as well as the complexities of the mechanics themselves. Not having a tutorial can be applied to any game, in any genre.

The lack of a tutorial can be successfully utilized in simple games [6], but becomes less effective the more complex a game is. Usually games with no tutorial involve a simple set of mechanics that can be quickly understood through trial and error, with deeper game play coming later in the game’s experience. Many games also use harder variations of early challenges in the later stages, so that the player uses their mastered mechanics in roughly the same setting. Here are 11 figures which contain examples of games throughout the ages that do not have any tutorial: 10.1, 10.2, 10.9, 10.10, 10.11, 10.3, 10.4, 10.5, 10.6, 10.7, 10.8.

One might expect that a lack of instruction would lead to a situation where the player is unable to progress. Games that lack tutorials generally rely on trial and error, whereby the player is likely to discover the correct mechanic usage based on cues from
the environment, direction, or perhaps simply frustration. It should be noted that this style of tutorial is likely inadequate for mechanics that require sequences of inputs that would be almost impossible to perform randomly, something common in fighting games. Sequences of inputs are outside the scope of this work, but are worth making note of.

![Fez](image1.png) Figure 10.1: Fez: released April 13 2012

![Pacman](image2.png) Figure 10.2: Pacman: released May 22 1980

![No Man’s Sky](image3.png) Figure 10.3: No Man’s Sky: released August 9 2016

![Tetris](image4.png) Figure 10.4: Tetris: June 6, 1984

**Tutorial By Exposition**

This method of tutorial conveys player actions directly through words and images, rather than through gameplay. Instead of the player using the game mechanics, they
are tutored by the game itself. This often takes the form of text information overlaid on the game screen, not something that we can really test. This style of tutorial also includes text manuals and documentation, which are outside the game world.

Figure 10.12, 10.13, 10.15, 10.14 all show what tutorial by exposition looks like when implemented in games. There is no gameplay, just instruction on how to control the
Figure 10.11: Mega Man: released December 17, 1987

characters.

Figure 10.12: Devil May Cry: released 15 January 2013

Figure 10.13: Risk of Rain: released

Figure 10.14: Just Cause 2: Released March 23, 2010

Figure 10.15: Street Fighter V: Released February 16, 2016
The Tutorial Room

The tutorial room’s style of teaching allows a player to test out their abilities, without the threat of failure. Here the game will isolate its mechanics, but it is obvious given the context that the player is being taught to use their abilities in a safe environment, not progressing the game state forward. The tutorial room is a series of isolated tutorials which might also use text to teach the player the game’s mechanics. The tutorial room suffers from breaking the immersion of the player, taking them out of their experience, and having them learn mechanics. While the tutorial room can be effective Paul Suddaby [60] points out that with a large number of mechanics (especially those higher in complexity) it becomes unlikely that the player retains all of the learned knowledge.

The following five games all use the tutorial room to teach the player its mechanics:
- Portal: Released October 9, 2007 [10.1.1]
- Bionic Commando: May 19, 2009 [10.1.1]
- Wind Waker: December 13, 2002 [10.1.1]
- Oddworld: Stranger’s Wrath: Released January 25, 2005 [10.1.1]
- Halo 4: Released November 6, 2012 [10.1.1]

Figure 10.16: (Portal) This room shows the player how to use the basic controls, only allowing the player to progress once they have experimented.
Figure 10.17: (Bionic Commando) This room teaches the player all the uses for their mechanical arm, even ones they won’t actually get to use until later in the game.

Figure 10.18: (Wind Waker) Here the player learns all of the possible sword attacks that are available throughout the game.

Figure 10.19: (Oddworld: Stranger’s wrath) is approaching a contextual lesson, but this entire level just showcases mechanic after mechanic.

Figure 10.20: (Halo 4) The tutorial rooms in halo have undergone quite the evolution, the halo 4 one being just a simple tutorial on using the action key.
Contextual Lesson

These tutorials are found in many modern games, presented organically as part of the game’s play. The player barely notices a difference between the game’s main objectives, and the tutorials designed to teach them how to play the game. They might play as normal only for the game to slow to a crawl and display the correct button input, progressing when the player successfully follows on screen instruction. Upon the same event occurring again the game will not slow down, letting the player demonstrate their knowledge, and play the game without aid.

Many games allow the player to safely use trial and error to figure out mechanics, and give the player no external assistance. To distinguish itself from no tutorial, these sections often give little to no consequence for failure, and make it obvious that the player is required to accomplish simple tasks to progress. Things like small dialog boxes, checklists, audio queues, and the lack of risk, make it obvious that the player is being taught.

This first level in Assasin’s Creed (Figure 10.21) is entirely about bringing the player up to speed on how the avatar is controlled, while masking it as a part of the story. The player is introduced to challenges one by one, with a little help from dialogue boxes. Figures 10.22 and 10.23 show two games which follow the same pattern as Assasin’s Creed.

10.1.2 Isolation

When a challenge requires a game mechanic, we assume that if the player has not used the mechanic before, that moving past the challenge implies they must have learned the mechanic. An isolated game mechanic will not have been used by the
player before, and is also the only previously unused mechanic that the challenge requires. For the isolation principle [58] to hold in a game, each challenge must not have more than one mechanic that the player has not used previously.

An isolated mechanic’s first use might have elements of contextual lesson, or exposition, this is irrelevant as long as it is the only unused mechanic. Generally games that isolate, such as Mega Man [13] or Shovel Knight [66], rely on it and do not
include any specific kinds of tutorial. The source material lists Mega Man as a game without a tutorial, but it also utilizes the isolation principle. The isolation principle ensures that you have used the required game mechanic to proceed, but does not explicitly tell you to do so. By having only one new mechanic per challenge, the player is less likely to feel overwhelmed with the number of new things to learn. This type of tutorial never risks breaking the player’s immersion, since no text, graphics, or instructions are displayed that are not part of the game’s world.

The following games demonstrate the Isolation principle, and use it as the primary method to teach players the game’s mechanics: In figure 10.24 we see the player given a pile of rocks, no text, and no obvious tutorial. This is meant to pique the player’s curiosity, and when they tap buttons they will find that one of them strikes the rocks. This is a completely harmless environment, and the player uses the ability to strike rocks going forward, never having been formally taught. In figure 10.25 we see a game introduce a new mechanic alone, this time in the form of an enemy. Since any of the 8 levels in Mega Man are selectable in any order, the game introduces this enemy several times in a row so you get a feel for it. In figure 10.26 the player is not told how to dash, but their way is barred by an open pit. In order to cross, they must experiment until they learn how this is done. This mechanic is isolated, since it is the only task that has to be accomplished in this challenge. Figure 10.27 demonstrates another example of an enemy being introduced in isolation. In fact this enemy had been seen earlier, this variation has a ranged weapon. Due to the incredibly high difficulty of Dark Souls, they took great care that each new enemy was introduced in isolation so the player could learn its patterns.
10.1.3 Principle Analysis

In theory we could analyze tutorial by exposition ensuring that mechanics needed to progress, have been shown to the player. We could at the least check to see if these mechanics are contained in the exposition, thus knowing if the player has "learned"
the upcoming mechanics. The main reason we focus on the isolation principle, is that it can be verified by simply modeling the game’s world, in conjunction with it becoming a popular method for teaching the player. It is also possible that if a game is using a contextual lesson, tutorial by exposition, or a tutorial room that all of these might introduce mechanics in isolation. Aid outside the game’s play does not change whether or not mechanics are introduced alone.

This principle focuses on more than just the avatar’s game mechanics, it covers anything that the avatar can have some kind of interaction with. The approach that we take is to extend the coverage of game mechanics to include the mechanics that the game uses to interact with the avatar. This includes enemies, traps, bottomless pits, pushable blocks, etc. While all of these are unique in the way that they are actually implemented, they all serve the same purpose, interact with the avatar to create a challenge / task.

While our analysis so far covers the concept of tutorial, we must delve into what is meant by early. When the author wrote early they likely meant ”before crucially needed”. This is consistent with the way that most games yield game mechanics, and happens to coincide with the concept of isolating mechanics.
10.2 Principle Clarifications

The biggest clarification that must be made, is that we specifically focus on the concept of isolation as a tutorial. We will not be concerned with the player obtaining mechanics, we begin the game with the player having all game mechanics at their disposal. We are trying to determine if the player gets a suitable environment to learn about that mechanic, without the interruption of other unused mechanics.

![Figure 10.28: Shovel Knight, using the "downspike"](image)

Take for example in the game Shovel Knight [66] shown in figure 10.2 when the player must use the "downspike" ability to progress. The player must also jump to "downspike", theoretically a conflicting game mechanic? Since this is the first time that the "downspike" is required and not the first time the jump ability is required, the game mechanics do not conflict. Should this challenge also require a mechanic that has not been used previously aside from the "downspike" (introduce a new enemy, or new ability), the isolation principle would fail.
10.3 Assumptions

We make 2 assumptions for the analysis of the isolation principle. Firstly, almost every game has some set of mechanics that the player will not be taught. These mechanics are generally what the player needs to operate the avatar on its most basic level. This includes mechanics like walking, aiming, jumping, interacting with basic objects etc. This is not to say that the game cannot or will not isolate these, but there is going to be some acceptable number of mechanics that do not need to be isolated. The original Mario Bros. [44] managed to isolate the jump mechanic by placing a Goomba (enemy that simply walks left) at the right of the player’s starting position. Should the player fail to jump, they never passed by the first challenge. While some games might treat jumping as a base mechanic, it might not always be the case. Since the player starts their experience having to know how base mechanics operate, they logically can be placed in the set of previously used mechanics.

The second assumption is that once a mechanic has been isolated, if the player passes the challenge by, they learned how the mechanic operates. Practically there are some cases where this may not be enough. Games tend to let players use mechanics several times, which makes sense, because designing and implementing mechanics costs resources to the developer. If a mechanic were introduced, then not used for a long period of time, it is likely the player will forget how that mechanic operates. Essentially the mechanic would become ”unlearned”, and would once again have to be introduced in isolation. We do not take this issue into account, because the length of time between learning and forgetting is unknown and outside the scope of this thesis. It is likely that if a player should forget what mechanics did, this would come up in playtesting.
10.4 Requirements and Representation for a model of the Game World

We seek to ensure that any never before used mechanics, are used for the first time either alone or with other previously used mechanics. Here we lay out the minimum requirements from the game to accurately model and analyze this principle. Following this is our specification and a description of the variables required.

10.4.1 Game World Requirements

When formally laying out our requirements, there are a few things we try to keep in mind. One would generally expect that if a game mechanic requires another mechanic to function, that it would in fact interfere. This can be seen as a requirement of the mechanic itself, conflicting mechanics occur when two or more mechanics are needed to pass through a single challenge, or a second mechanic is actually usable. Having the option would allow the player to pass through, failing to isolate the new mechanic.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation of all the challenges in the game world</td>
<td>While inefficient, as most challenges will not have any new mechanics taught, it is imperative that any uses of a mechanic not previously used in isolation are caught.</td>
</tr>
<tr>
<td>Challenge order is preserved</td>
<td>Without preservation of challenge order knowing whether or not a new game mechanic has been isolated, will be impossible.</td>
</tr>
<tr>
<td>All Game mechanics must be represented by all the challenges that use them</td>
<td>Without representing all the game mechanics, it would be possible to miss instances where a game mechanic is not presented for the first time in isolation.</td>
</tr>
</tbody>
</table>
10.4.2 Game World Representation

Model

Listing 10.1: AsmL model of the game world

```asml
class Challenge
    var name as String
    var requiredMechs as Set of Mechanic
    var adj as Challenge

class Mechanic
    var name as String

    var unusedMech as Set of Mechanic
    var usedMech as Set of Mechanic

    var currentNode as Challenge
```

Variables and Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>unusedMech</td>
<td>The set of mechanics that the player has not yet used, all mechanics start here</td>
</tr>
<tr>
<td>usedMech</td>
<td>The set of mechanics that the player has used, and therefore learned</td>
</tr>
</tbody>
</table>

10.5 Mapping of Principle requirements to our Model

The specification for this principle is quite straightforward. We seek assurance that if a mechanic is required for a challenge and it has not been used in the past, that it
is the only mechanic being used for the first time in this challenge. This correlates to
the notion that the early use of a mechanic is meant to be in isolation.

Listing 10.2: AsmL model of the game world’s behaviour, and the analysis step

```plaintext
Main()
    step until fixpoint
    foreach mechanic in challenge.requiredMechs
        if(mechanic in unusedMech and size(unusedMech intersect
           challenge.requiredMechs) = 1) then
            add mechanic to usedMech
            remove mechanic from unusedMech
        else if(mechanic in unusedMech and size(unusedMech intersect
                   challenge.requiredMechs) != 1) then
            currentNode.adj := currentNode
            currentNode := currentNode.adj
```

Our mapping iterates through the game world’s set of challenges, until the state
cesses to change. In each challenge we ensure that we check each mechanic for whether
or not it had been previously introduced in isolation. The algorithm terminates when
there is no change in the game’s state. If it terminates at the end node, then all
mechanics were used in isolation.
Chapter 11

Do not teach with death

We begin this chapter with an analysis of setback (death), and provide a way to formally check if a game punished the player overtly whilst trying to teach them mechanics.

11.1 Informal Analysis

When a player sits down to play a video game, they find themselves in a new world with new rules. As they progress through the game, there will be new game mechanics for the player to learn about. While trial and error as a learning pattern is common for games [24], it can be incredibly frustrating to lose progress while you are learning. This principle works to that end, ensuring that the player cannot be too severely punished with death.
11.1.1 Game Mechanics

As with the isolation principle, it is important that the scope of game mechanics covers both the player’s mechanics, and the game’s mechanics. “It’s okay if the player dies because he makes a mistake, as long as he knows why, and was given fair warning or some indication that he could make a mistake” [36], illustrates that the player should be aware of the dangerous mechanic, prior to encountering it in a challenge. This of course refers to a mechanic that the game has, but because the player uses mechanics to overcome these, so they must also be included in the model.

The Dark Souls series demonstrates the need for both kinds of mechanics to be modeled. Dark Souls is a notoriously difficult series of games where the player uses a variety of different weapons to progress towards the game’s end goal. The enemies in this game have very complex patterns that can be radically different from one another, there are so many varieties that they count as their own mechanics.

You can tell that the Dark Souls publishers innately understood the spirit of this principle. New enemies are almost always introduced in isolation, and almost always near a respawn point. Dark Souls is incredibly difficult, frequently killing the player between the location of the bonfires as shown in Figure 11.1 (save points, player respawns here if they die) and the ability to recover your corpse means that when you are learning to deal with a new enemy as shown in Figure 11.2 your punishment is almost negligible.

11.1.2 Death (setback)

This section is a clarification on death’s relation to video games with respect to this principle. Death is commonly the result of failing to complete a challenge. Death is
simply a state of the avatar, the real punishment being the consequence of failure, the potential setback. So instead of looking at death, we look at ”setback” \[32\] or ”punishment”. Death is technically just a state of the avatar which, depending on the game’s rules, might not be a negative thing. Prince of Persia \[64\] simply brings the player back to life immediately, Kirby’s Epic yarn makes the player lose beads, in both situations the player may face some punishment, yet death is not involved. Games such as Dark Souls kill you frequently, but your checkpoints are generally close and you rarely lose items. Would a game like Dark souls, where death is trivialized, fail this principle?

It is important to analyze what the author meant for death to accomplish. Death is meant to be an indicator that the player failed a challenge, the author’s implication is that death is a negative state. Discussed previously was that death may not have any effect, or be negative in the slightest. In reality the player should not be taught something new, and be punished for their failure. According to Juul \[32\] there are several ways that the player can be punished, rather ironically Juul missed that they are all forms of setback. In order to be punished, something that the player has, must be taken away. Punishment comes in the form of being set back in any of the avatar’s
attributes, mechanics, inventory or position.

This principle is the first to require some kind of representation for all aspects of an avatar. Included in the model should be the avatar’s attributes, inventory, mechanics, and position within the gameworld. This will allow us to measure the setback of each, should the player fail a challenge.

**Setback in attributes**

If failing a challenge results in a decrease or loss of any attributes (Detailed in Chapter 2, section 2.6), the player is setback. Take for example experience, classically an attribute gained which boosts other attributes once a certain quantity is attained. While no longer common, some classic games would force you to lose all your experience if you died. This resulted in many frustrations since gaining experience was often required to progress the game to later states. Losing all experience would force the player to replay the same section over again. The loss of experience upon failing a challenge would be a setback to the avatar’s attributes.

**Setback in mechanics**

A mechanic is a function of either the game world, or the avatar that are used to change the state of the game world. Getting setback in mechanics is very unusual in games. This setback was very common when games heavily utilized save states, and everything was lost if the player died, and returned to the last saved state. Modern games utilize auto saves, or very frequent save states. This being said, mechanic loss still occurs, just to a less frustrating extent.

In figure 11.3 we can see that Mario is currently sporting his red and white attire.
Should he die, he loses his fire flower abilities, reverting back to mini Mario. This is the kind of mechanical setback that is most common, these mechanics are temporary as is, and the player embues themselves for a short time. In figure 11.4 the weapons that the player gets in this rogue-like, disappear each time the player dies. With rogue-likes, the game is designed to be a one-shot attempt at reaching the goal. This is why the early challenges in the game are very easy, so the player can learn anything that they have to without any large risk of loss.

![Figure 11.3: Mario sporting his fire flower ability](image)

**Figure 11.3: Mario sporting his fire flower ability**

![Figure 11.4: By the header weapons, you can see the "Dual Lasers"](image)

**Figure 11.4: By the header weapons, you can see the "Dual Lasers"**

**Setback in inventory**

Inventory (Detailed in Chapter 2 section 2.5) is meant to represent consumable values that game mechanics frequently interact with. The difference between inventory
and attributes, is that inventory items are something the avatar has, rather than something the avatar is. In reality the inventory and attributes are very similar, and the distinction between them is usually just aesthetic. Attributes are not something that the player actively interacts with, whereas inventory items are. A player might use an item that affects attributes, but they do not use the attribute itself to interact with the game world. Setback occurs in the inventory when a challenge uses some quantity, and does not replace what was consumed. Keep in mind that this is only an issue upon failure, where you can assume that the avatar’s loss of inventory will result in a search to replenish it.

**Setback in position**

Position is the representation of the avatar’s location within the game world. A setback in the position of the avatar can be difficult to quantify the less linear a game is. For this reason it is important to know where the avatar came from, to measure its setback upon failing a challenge. Position setback can be vastly different depending on the difficulty and time requirements of each challenge. It can generally be stated that the further away the avatar is placed from their point of failure, the worse it is.

Figure 11.5 shows a door that can only be opened by using a missile upgrade that was recently acquired. The developers of this game made sure that the player could not proceed without using the missiles at least once in a very safe environment. However, had the developers left enemies in this room (and the player failed the challenge), the player would essentially be forced to restart the entire game. There are no save stations prior to this challenge, which is the kind of thing this principle hopes to avoid.
In figure 11.7 we can see Mario with his brand new cape, which he would imme-
diately lose should any of these enemies hit him. Should the player die, they restart
the same course, losing all your lives takes you to the location in figure 11.8. This
particular level is early on, so dying here takes you back to only one level prior, but
this is not the case later on, for this I have a personal experience. Midway through
the game the player is finally forced to learn that there are mechanics using locks and
keys, something that is previously in the game, but not required to advance. If the
player loses all their lives here, they will find that they have to redo over four levels
to restart the level with the key requirement. A much more serious punishment for
learning a certain mechanic.

Figure 11.9 is a save state from the classic RPG Grandia. Functionally it captures
the avatar exactly as they were at that exact point, and should the avatar die, they
respawn at that point exactly as they were. Aside from the obvious flaw where the
player can repeat this segment endlessly, it also means that should the player not find
these save points, the setback for failing to complete a challenge becomes greater.
11.1.3 Learning

As with constantly teaching the player something new, we look to verify that the player has in fact learned something. We focus on the potential setback associated with learning, and are less concerned with the process of learning mechanics. With the emphasis on setback, we strictly want to identify the challenge with which the avatar must first demonstrate their understanding of a mechanic. We assume that all mechanics have been acquired, and analyze setback on the challenges meant to verify a mechanic has been learned.
11.2 Principle Clarifications

The two important clarifications for this principle were laid out above, in order to describe the necessary facets of video games. Death is meant to be "setback" or "punishment", and being taught are any of the avatar’s or game world’s mechanics.

11.3 Requirements and Representation for a model of the Game World

While the concept of avoiding any setback while imparting a game mechanic on the player is straight forward, it requires an incredibly cumbersome model to correctly analyze. To determine if a setback has occurred, we check the avatar’s position in the game world, attributes, inventory, and mechanics, upon failing a challenge.

11.3.1 Game World Requirements

These are the requirements for a specification that can accurately model the game world to be analyzed by the topic principle. Not only do we lay out the requirement, but also give its reasoning for being a part of the model in the table below.
<table>
<thead>
<tr>
<th>Requirement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order is preserved</td>
<td>Allows us to verify how much the player has been set back in position</td>
</tr>
<tr>
<td>Challenges have all game mechanics represented</td>
<td>To verify the learning of game mechanics, it follows that they should have representation</td>
</tr>
<tr>
<td>Challenge failure results in consequence</td>
<td>The failure to complete a challenge might result in the setback of the avatar in any of: Mechanics, inventory, statistics, or position</td>
</tr>
<tr>
<td>A maximum amount of item resources that can be lost</td>
<td>This ensures that failing to learn never takes away too much of the items in an inventory</td>
</tr>
<tr>
<td>A maximum number of attributes that can be removed</td>
<td>Ensures the players attributes never lose more than this value</td>
</tr>
<tr>
<td>A maximum number of mechanics that can be removed</td>
<td>A more serious punishment, removing mechanics would force the player to relearn them</td>
</tr>
<tr>
<td>A maximum distance to be set back is known</td>
<td>Also a serious punishment, since the player has to repeat everything on their way back to respawn</td>
</tr>
</tbody>
</table>

11.3.2 Game World Representation

This model in particular has the potential to be incredibly cumbersome, but with the right approach we can significantly narrow down its scope. Including everything in the model seems necessary, but our examination of this principle has revealed that we only care about failing a challenge, and the consequences that stem from that. If you
succeed at a challenge, yet it takes away all of your attributes, items, and position, it is not of consequence. Your only concern is not to teach with death, death naturally implying that you failed the challenge.

We can disregard any prerequisites that challenges have, as this analysis is searching specifically for the outcome of failure. The only two required parts of challenges are the requirements of its tasks (to determine if the player is learning), and the outcome of failure.

Model

Listing 11.1: AsmL model of the Game World’s behaviour and principle analysis
Variables and Definitions

<table>
<thead>
<tr>
<th>Variable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>acquiredMechanics</td>
<td>Keeps track of the mechanics that the player has in their possession, needed to measure the setback from a challenge</td>
</tr>
<tr>
<td>maxMechanicSetback</td>
<td>The maximum number of mechanics that the player should be able to lose from a challenge</td>
</tr>
<tr>
<td>maxPositionSetback</td>
<td>The maximum number of challenges that the player can be set back having failed to learn a game mechanic</td>
</tr>
<tr>
<td>startNode</td>
<td>Holds the node that the player started the game from</td>
</tr>
<tr>
<td>currentNode</td>
<td>Holds the position of the avatar in the game world</td>
</tr>
</tbody>
</table>

11.4 Mapping of Principle requirements to our Model

Listing 11.2: Search function used in the AsmL model

```asmL
Search(start as Challenge, returnTo as node, diedAt as Challenge)

var currentNode := start
var returnToTime as Number
var diedAtTime as Number
while currentNode != returnTo
  currentNode := currentNode . adj
  returnToTime++

currentNode := returnTo

while currentNode != diedAt
  currentNode := currentNode . adj
  diedAtTime++

return diedAtTime = returnToTime
```
Now that we have boiled our analysis down to the case that the player has failed the challenge, and that the player must be learning, we can analyze whether or not the player was taught with death. For each challenge in the game world, we first check if there are any previously unused game mechanics within this challenge. If there are, we check the loss for failing the challenge, checking to see if it goes over the maximum allotted amount of setback. If the setback is too much, terminate the program.

You should notice that unlike constantly teach the player something new, there is no verification step for each mechanic. Since the verification step is where the player learns, this is actually the only step we model. This does not mean that they unlearn them though. Each mechanic should start in both acquired and unused, and then once successfully used in a challenge, removed from unused. This allows the game to monitor the loss of mechanics from acquired, and whether or not the player has in
fact learned anything.

Our analysis progresses through the set of challenges, checking to see if we exceed the maximum setback in any of the four possible ways that a player can be setback. We check to see if they lose too many mechanics when failing to learn a new game mechanic, lose too many attributes, too many items, or have too much of their progress undone. Notice that these checks are fairly basic, and seem to require knowledge of what the maximum amount of setback is for a challenge, yet there are no values given. This is because we do not yet know what the maximum setback is, but we know that there is going to be one. The modeler might even make the allowable setback dynamic, perhaps early challenges should have less setback, later ones having more.
Chapter 12

Contenders for analysis

12.1 Good level design empowers the player

This game design principle has a few major problems when it comes to being a formal tool for analysis. While Taylor does illuminate to the reader what empowerment is: [63] “For players to experience true empowerment, their actions must have a noticeable effect on the game world”, this definition of empowerment is problematic. Given the context, it is quite apparent that Taylor meant for the physical player to notice what happens in the game world, and thus feel empowered. This is reinforced by this line: [63] “Video games are escapism... pure and simple. Why would players want to escape somewhere more mundane than their existing lives? Level Designers should never ask players to do something that they can easily do in real life your mission objectives should shun banal and repetitive chores, and always be interesting and exciting”. Where we see that the subjective qualities of a persons life should directly affect their feeling of empowerment.

Now we turn to the description of this principle for some further clarification.
There are three distinct things that Taylor wanted the reader to think of when applying this principle:

1. Games are escapism, thus there should not be menial tasks
2. Games should avoid repetitive tasks
3. The player should notice changes in the game state to feel empowered

As far as escapism is concerned, there is far too heavy a reliance on the player’s state of mind, to accurately measure the concept. There is no concrete way to determine if an act within a game is going to be considered a menial task, especially given the wide variety of players that exist. If the game is a cooking simulator, chefs may find certain tasks (even complicated ones) rather menial, since they perform the real world counter part frequently.

Avoiding repetitive tasks is actually something that we can measure for, and is worth considering as its own principle in the future. Unfortunately it is just one part of empowering the player, so it is not analyzed at this time.

The third distinction, is possible to measure as well by use of in game proxy. If we knew what kinds of changes in the game’s state affected a player’s sense of empowerment, we could analyze the amount of empowerment a player might feel. Yet the problem remains that the player still might not notice a change, or simply not care.

In the event it is determined that these three things actually are measurable, one should break this principle down, into three distinct principles. In short, too much of this principle requires knowing about the particular player, who is playing the game.
12.2 Reward the Player

This principle simply had no real explanation, and was far too general to be a true design principle. Research from other sources indicated that there might be (at least) two design principles which related to rewarding the player: classification of rewards, and reward scheduling.

Zimmerman’s rules of play \cite{54} outlines 2 distinct steps for rewarding the player. Step 1: Categorize the game’s rewards, Step 2: Measure the rate of rewards throughout the game. Our goal would be to outline methods for classification, i.e. give examples of classifications in different games, and a model for representing rewards. Then we would evaluate the rate of rewards, to see if they match any of: fixed interval, variable interval, fixed ratio, variable ratio, and escalating ratio. These schedules are known to be excellent for behavioral conditioning, and are considered to be ”good” rates of rewards.

12.3 Large Rewards, Small Punishments

Punishment was fully defined as setback according to our method of abstraction in chapter \cite{11}. This principle would require us to know the actual worth of a reward, to compare them to one another, i.e. what’s worth more: Gold, or a plot point.

It would be more than possible to determine the relationship between rewards and punishments, if each reward were isolated. If a game contained only one kind of reward for example, this principle would be quite useful. Yet the principle remains very general, there would have to be research into the area of creating an absolute quantification of the value of a type of reward, so that rewards and punishments could
be measured against one another. Without this knowledge, the intrinsic value of a reward is subjective, and two different kinds of rewards cannot be compared.

12.4 No Positive Feedback Loops

Positive feedback is defined in [4] as occurring whenever one useful achievement makes subsequent achievements easier. This is natural behavior for a video game, where early challenges make subsequent challenges easier. It is part of the prominent challenge - reward cycle.

In a feedback loop, a player remains in a fixed set of challenges, and continues to become more powerful. Feedback loops are often made possible by an oversight from the developer, who did not test thoroughly for situations where the player could abuse the game’s systems. Problematic feedback loops occur for game systems where the player power to game progression is meant to be somewhat fixed, but is circumvented early. Say that there is a room with arrows, the designer may want the player to be able to refill their arrows to capacity. This would not generally be abusable because the player has a cap to their arrow capacity, that cap prevents them from getting overtly powerful at the point of the loop. Should the player discover a loop for systems which do not have a cap, or whose cap is meant to be filled by the end of the game, we encounter a problem.

This principle was not done simply due to its difficulty. Conceptualizing how bad a feedback loop can be, and how many challenges before a loop is no longer a problem were issues that were not resolved. In the future this would be an excellent principle to model, as positive feedback loops can be incredibly problematic.
12.5 Provide Parallel Challenges with Mutual Assistance

This principle is meant to assess whether the game offers challenges along side one another, whose rewards can assist the player with their aims of completing their remaining tasks.

Direct quote from the 400 project [20]: “When presenting the player with a challenge, a monster to kill, a puzzle to solve, a city to capture provide several such challenges and set it up so accomplishing one challenge makes it a little easier to accomplish the others (that is the mutual assistance component). It is also effective to set up these parallel challenges on many levels of scale of the game, from the ultimate goal down to the small short-term steps. This eliminates bottlenecks and makes the game accessible to a wider range of players. Ideally the different challenges use different domains of player skills, e.g. strategy and action”.

It is clear that this principle is meant to guide the designer into running parallel paths, where one path’s rewards make it easier to accomplish the other paths. For games where the player’s power is specifically predicated on one attribute say, experience, this is quite easy to analyze. We start to encounter issues when a reward is given that might not affect other challenges, or even more interestingly, optionally can effect other paths. We would look to categorize rewards, so that we can clearly distinguish between rewards that effect one, more than one, or all challenges once they are given to the player. From here we can attempt to measure mutual assistance.
12.6 Provide an enticing long term goal

Even if we could just rationalize that the designer would just hope that their end
goals are enticing enough, there is the issue that this principle is too simple. It is
just a simple verification of one challenge at the end of the game, that counts as an
enticing long term goal.

We originally hoped that this principle would be more like provide clear short
term goals, in that we would have several enticing goals, but the description from the
400 project makes this pretty clear [20]: “Many (but not all) games benefit by having
an ultimate goal that is made clear to the player fairly early on. Making this goal
enticing is one way to pull the player into the game world and encourage passion”. We are to have a single goal that is made clear early on.

12.7 Predictable consequences

[28] “The player has to perceive failure as a consequence of her mistakes, not as a
random or predestined event unrelated to, and unavoidable by, any reasonable choice
of actions on her part”. The player must be able to know that the consequences of
succeeding or failing are predictable. In the Legend of Zelda franchise, there is a
game mechanic whereby the chests you usually receive treasures from, are actually
trick chests. Instead of rewarding you with rupees or hearts, they would freeze and
damage you. At first this mechanic is extremely frustrating, especially if you have no
prior experience with games in the genre, but once known, it becomes just another
thing to keep in mind. Players will ensure that there is no surrounding danger and
their health is full, removing the original frustration.
This principle was close to being included, but was not because of a lack of space and time.
Chapter 13

Conclusion

This thesis was born of the desire to capture the essence of good levels, and make them scale to any size of world. Prior to this work there was no established method of measuring the quality of a level, and capturing those qualities. Along side this, there was no way to measure if game design principles correlate to good or bad games. This thesis breaks down each game design principle to extract what would be required from a game world, to see if said principle holds.

13.1 Contributions

Prior to this thesis there is little work on the topic of game design principles, although there is quite a bit of research on appropriate ways to model games. In particular we give both the model for a game world, as well as an algorithm that represents an analysis of the game from the perspective of a game design principle. Each algorithm looks at the particular parts of a game’s world and ensures that the principle is respected.
13.2 Difficulties with Principles

In choosing to analyze games for their adherence to game design principles, it was not immediately apparent how subjective so many principles are. Even more baffling, was how many principles required information about players that would be incredibly difficult to know on design time.

For this reason we particularly selected principles which governed details of games which had very little to do with the player directly. Each principle presented certain issues which will be detailed here.

13.2.1 Good level design constantly teaches the player something new

This principle presented us with the challenge of pin pointing an in game point in time, where we could guarantee that a player had learned how a mechanic operates. As we have stated many times, there is no way to ask the player themselves, so we opted to measure a proxy within the game itself.

By searching for a challenge where the player must use a mechanic they had never used before, we determine that for them to proceed, they must have learned how the mechanic works. While this is logical, it might not be true. It might take more than one usage of a mechanic to truly internalize it, in which case we have not gone far enough. Luckily this would be easily adjustable in our model, and was dealt with by assuming the player fully learns a mechanic after one challenge.
13.2.2 When a player overcomes a challenge that requires a resource, provide more

This was arguably the most straightforward principle of all. The language was clear and required no knowledge of the player’s emotions, thoughts, or feelings.

The main challenge came with how much the principle confined the designer, essentially stating that a player should have their resources replenished immediately after using any resources. This really removes any creativity a designer might have with creating some kind of tension, whereby the player is unsure when they might get more resources. To solve this we introduced an addendum where the player must be given more resources prior to a challenge that will in fact fully deplete them.

13.2.3 Clearly inform the player of short term goals

This principle was straightforward once we researched and categorized goals, and the systems that deliver them. The issue which we made an assumption about, was whether or not the player retains the knowledge of the goal you have provided. While this is less of an issue in modern games, where the next goal is usually stored in the interface, older games and more challenging games might leave it up to the player to recall.

13.2.4 Early levels are tutorials (Isolation Principle)

This principle proved to be somewhat tricky, and was made far more useful by focusing in on one kind of tutorial. Each of the tutorial styles might have been analyzable, but were not particularly useful, the tutorial room and exposition are seeing less use
than in the past.

The isolation principle proved to be the most interesting and effective to analyze, since it applies strictly to the game, and does not require any influence from the player.

13.2.5 Do not teach with death

Quantifying death was easily one of the most time consuming and difficult tasks in this thesis. Death itself is not the real issue, easily demonstrated by death in certain games being completely negligible. In addition, it was clear from the source that the player is not meant to be punished while being taught something new, as opposed to dying. It should be noted that dying is very commonly the catalyst to punishment in games, so the principle’s language rings true frequently.

Setback is ultimately the way to punish players in games, since the player does not get physically affected, taking away what you have given them is the only way to punish their mistakes. Once setback of mechanics, inventory, attributes, and position are well defined, we look for when the player is learning and determine if they were in fact set back. Creating an algorithm that analyzes a model for setback proved to be difficult, specifically when trying to determine if the player was setback in position. It remains to be seen if the amount of allowable setback is some constant, or varies depending on the game or perhaps the person playing it.
13.3 Future Work

There is quite a bit of future work that could come from this thesis. Firstly it is quite cumbersome that each principle requires a different model for analysis. Work can be done to create a unified model that any principle could theoretically analyze. Many of the principles that were not included, discussed difficulty and skill. These two elements were avoided since they generally require some kind of measurement from the player, and difficulty can often be quite subjective. A model that included some element of absolute difficulty would open up many principles for study.

In addition, most of these principles have never been formally studied, even though widely referenced. This thesis would allow for a study of correlation to be done, to see if game design principles have bearing on the scores or sales of video games.
Bibliography


[34] Kreimeier, B. (2002). The case for game design patterns.


[60] Suddaby, P. (2012). The many ways to show the player how it’s done with in-game tutorials. unpublished.


